



# Herschel Interactive Analysis

## A Basic User's Manual

V0.4, 21 January 2005

A.P.Marston & H. Siddiqui,  
Herschel Science Center

1	The Herschel Common Science System and Interactive Analysis (IA)	7
1.1	Brief Overview.....	7
1.2	Availability of IA and Operating Systems.....	8
1.3	Related Documentation.....	8
1.4	Versioning.....	8
1.5	Previous Versions of IA User's Manual.....	8
1.6	What's New.....	9
1.7	List of Contributors:.....	10
2	HCSS Downloading and Installation.....	11
2.1	Introduction.....	11
2.2	Platform.....	12
2.3	Pre-Installation Requirements.....	12
2.4	User Installation Procedure.....	13
2.4.1	Known Installation Problems.....	14
2.5	IA Property Initialization.....	14
3	Introduction to Working in IA: Using Jconsole.....	15
3.1	Introduction.....	15
3.2	Running IA under Jconsole:.....	16
3.2.1	File Menu.....	18
3.2.2	Console Menu.....	18
3.2.3	Edit Menu.....	18
3.2.4	Run Menu.....	19
3.2.5	Help Menu.....	19
3.3	Standard Settings for Jconsole.....	22
3.4	Using <code>import.py</code> to customize Jconsole.....	23
3.5	Programming Loops in Jconsole.....	23
3.6	Multiline Statements with Jconsole.....	24
3.7	Errors and Exceptions in IA.....	25
3.7.1	Overview of the different libraries used in an IA session.....	25
3.7.2	The error back trace mechanism.....	26
3.7.2.1	The way Jython presents error messages.....	26
3.7.2.2	The way JAVA presents error messages.....	27
3.7.3	The HCSS exception and logging mechanism.....	27
3.7.3.1	Exceptions as thrown from HCSS classes.....	28
3.7.3.2	The HCSS logging mechanism.....	28
4	Some IA Basics & Beginning Jython.....	30
4.1	Basics.....	30
4.2	Lists and Dictionaries.....	31
4.2.1	Setting up and Accessing Lists.....	31
4.2.2	Slicing Lists and Arrays.....	32
4.2.3	Setting Up and Using Dictionaries.....	32
4.2.4	Nested Dictionaries.....	33
4.3	Augmenting Values and Arrays.....	34

4.4	Printing to the screen and files.....	34
4.5	Defining and Using Functions.....	35
4.6	Blocks and programming loops.....	35
4.7	Classes and Methods.....	35
4.8	Writing Scripts – Programming in IA.....	37
4.9	Some Useful Extra Items.....	37
5	Handling Arrays and Other Datasets.....	39
5.1	Introduction.....	39
5.2	Getting started.....	40
5.3	Types of Array Datasets.....	40
5.4	Creating a Simple 1D Array Dataset.....	41
5.5	Dataset attributes.....	42
5.6	Simple 1D Array Manipulation.....	42
5.6.1	1D Array Arithmetic.....	42
5.6.1.1	Addition, subtraction and concatenation:.....	42
5.6.1.2	Multiplication:.....	43
5.6.1.3	Array length:.....	43
5.6.1.4	General Application of Functions:.....	43
5.6.2	Logical Operations.....	43
5.6.3	Type Conversion.....	44
5.7	Dealing With Complex Arrays.....	44
5.8	Creating and Accessing Multi-Dimensional Array Datasets.....	45
5.9	Creating and Viewing a Table Dataset.....	46
5.10	Creating and Accessing a Composite Dataset.....	46
6	IA Numeric: Basic Functions for Herschel IA.....	48
6.1	Introduction.....	48
6.2	Getting Started.....	48
6.3	Functions and Lambda Expressions.....	49
6.4	Filtering.....	50
6.5	The 'where' and 'get' methods.....	50
6.6	Advanced Tips.....	52
6.7	Vectors and Matrices.....	53
6.8	Function Library.....	53
6.8.1	Basic Functions.....	54
6.8.2	Discrete Fourier Transform.....	54
6.8.3	Convolution.....	55
6.8.4	Boxcar and Gaussian Filters.....	56
6.8.5	Interpolation Functions.....	56
6.8.6	Basic Fitter Routines.....	58
6.9	Example Programs.....	59
7	IA Plot: Basic Plotting of Data.....	61
7.1	Introduction.....	61
7.2	What do I need to make a simple XY plot?.....	62
7.2.1	Introducing PlotXY.....	62
7.2.1.1	Using PlotXY to Plot One Numeric 1d Array Against Another.....	62
7.2.1.2	Using PlotXY to Plot Columns in a TableDataset.....	63

7.3	How to setup your PlotXY properties.....	65
7.3.1	How to modify properties .....	65
7.3.2	Plot properties .....	66
7.3.3	Layer properties .....	66
7.3.4	Axes properties. ....	68
7.3.5	How to use properties. ....	68
7.4	How to use PlotXY in IA scripts .....	69
7.4.1	What about these Layers? .....	71
7.4.2	What can I do with Axis? .....	73
7.4.2.1	Log Axes, Labels and Gridlines.....	74
7.4.2.2	Multiple Axis Labels.....	74
7.4.3	How can I annotate and decorate my plot? .....	77
7.4.4	How can I make my plots more colourful?.....	78
7.5	Creating File Output and Printing a Plot Without Displaying.....	79
7.6	Handling Units in Plots.....	79
7.7	What about a complete example? .....	81
8	Display: Handling Images with Herschel IA .....	82
8.1	Introduction.....	82
8.2	Using ImageDatasets .....	83
8.3	How can I display my image?.....	84
8.4	Display in more detail.....	86
8.5	How can I use Operations on my images?.....	87
8.5.1	Clamping (or clipping) an Image.....	88
8.5.2	Cropping an Image.....	88
8.5.3	Histogram of an Image.....	88
8.5.4	Rotating an Image.....	89
8.5.5	Scaling an Image.....	91
8.5.6	Translating an Image.....	92
8.5.7	Transposing an Image.....	92
8.6	How can I display my own numeric2d datatypes? .....	93
8.7	How to Use Different Layers.....	94
8.8	How to place annotations on the image.....	94
8.8.1	Annotations from the Command Line in your IA session .....	95
8.8.2	Annotations using the annotation toolbox .....	96
8.9	Sample JPEG Image and Extended Example Demo Script.....	98
9	Other IA Packages: What is Available? .....	99
9.1	Introduction.....	99
9.2	Overview of JavaDocs Documentation for IA Packages.....	100
9.3	Package view .....	101
9.4	Class view .....	101
9.5	Tree view .....	105
9.6	Deprecated view.....	105
9.7	Index view.....	105
9.8	IA Packages And Documentation.....	105
9.8.1	herchel.ia.dataflow .....	105
9.8.2	herchel.ia.dataset .....	106

9.8.3	herschel.ia.demo .....	106
9.8.4	herschel.ia.doc.....	106
9.8.5	herschel.ia.help .....	106
9.8.6	herschel.ia.image.....	106
9.8.7	herschel.ia.inspector.....	107
9.8.8	herschel.ia.io .....	107
9.8.9	herschel.ia.jconsole.....	108
9.8.10	herschel.ia.numeric .....	108
9.8.11	herschel.ia.plot .....	108
9.8.12	herschel.ia.task.....	108
9.8.13	herschel.ia.ui .....	108
10	Import and Export of Tabular ASCII and FITS Files .....	109
10.1	Introduction.....	109
10.2	Getting Started with ASCII Import/Export.....	109
10.3	Basic ASCII Table Import/Export Tool Usage.....	110
10.3.1	Import Parsers .....	110
10.3.2	Comma-Separated-Variable Parser.....	111
10.3.3	Fixed-Width Parser .....	111
10.3.4	Export Formatters .....	112
10.3.5	Comma-Separated-Variable Formatter.....	112
10.3.6	Fixed-Width Formatter .....	112
10.3.7	Table Template .....	113
10.4	Example of How to Import/Export ASCII Tables in IA.....	113
10.5	Overview of FITS IO .....	115
10.6	Getting Started With FITS IO.....	115
10.6.1	Basic FITS IO Tool.....	115
10.6.2	Parameter Name Conversion and FITS Header.....	116
10.6.2.1	FITS product header .....	116
10.6.3	Caveats.....	117
11	Using Time in the IA Environment.....	118
11.1	Introduction.....	118
11.2	Time Definitions .....	118
11.2.1	System time in IA ( <i>FineTime</i> ) .....	118
11.2.2	International Atomic Time (TAI) and <i>FineTime</i> .....	119
11.2.3	Coordinated Universal Time (UTC).....	119
11.2.4	DecMec Time.....	120
11.3	Time in HK Data.....	120
11.4	Time conversion.....	121
11.4.1	Time conversion in HCSS .....	121
11.4.2	CucConverter .....	121
12	Setup and Use of Databases .....	123
12.1	Introduction.....	123
12.2	Starting Up A Database: .....	124
12.2.1	Unix.....	124
12.2.2	Windows .....	124
12.3	Schema Initialization .....	125

12.4	Using an existing database and Schema Evolution.....	125
12.4.1	Initializing a schema on an old database.....	127
12.4.2	Schema Tool commands.....	127
12.5	Initializing a Database For IA Use.....	128
12.6	Quick Database Creation .....	128
12.7	Providing Database Access for an IA Session.....	128
12.7.1	Properties File Setup for Database Access .....	128
12.7.2	Using the Propgen Tool.....	129
12.8	Browsing a Database.....	130
12.9	Getting Data Frames From a Database.....	130
12.9.1	Command Line Access to Data Frames.....	131
12.9.2	From Database to ASCII File .....	132
12.9.3	Downloading Dataframes from a Database Using a GUI.....	133
12.10	Accessing Housekeeping (HK) Data .....	135
12.10.1	Accessing HK Information For a Given Obsid.....	135
12.10.2	Accessing HK Data For a Given Time Period.....	136
12.11	Removing a Database .....	139
Appendix A: Example User's Property File.....		140
Appendix B: Listing of Currently Available IA Classes.....		143



## Herschel IA Chapter 1

# 1 The Herschel Common Science System and Interactive Analysis (IA)

### Chapter 1 Contents

- 1.1 [Brief Overview](#)
  - 1.2 [Availability of IA and Operating Systems](#)
  - 1.3 [Related Documentation](#)
  - 1.4 [Versioning](#)
  - 1.5 [Previous Versions of IA User's Manual](#)
  - 1.6 [What's New](#)
- 

### **1.1 Brief Overview**

The Herschel Common Science System (HCSS) is being developed by the Herschel Science Center (HSC) and Herschel Instrument Control Centers (ICCs) to provide the complete software system for the Herschel Observatory mission. The intention is to provide a common system that is able to handle test data, observation planning, mission planning and instrument data from observations within one common development. An important element of this common development is Interactive Analysis (IA).

IA handles computed, stored or simulated data and has access to much of the software developed for other purposes within the HCSS (e.g., Quick Look Analysis, which runs on real-time data or replayed data streams from a database).

Branches of the HCSS have also been developed for handling instrument-specific tasks. So software packages for HIFI, PACS and SPIRE also reside within the HCSS framework and are available within IA.

Since the Herschel IA uses Java programming, it is very flexible and Java programs can be imported into a session. However, the basic IA system is a fully-fledged standalone

system that is being developed to specifically deal with data from the Herschel spacecraft.

## **1.2 Availability of IA and Operating Systems**

IA is available free of charge as part of the HCSS and can be downloaded for use on networked or individual desktop/laptop machines. Current operating systems supported by IA include

- Solaris 2.8+
- LINUX (Red Hat 8.0+, SuSE 9.1)
- Windows (2000, XP)

For download and installation instructions see [Chapter 2](#).

## **1.3 Related Documentation**

In earlier versions of the IA User's Manual, "HowTo" documents were available in parallel. Earlier HowTo documents for users are now incorporated into the current User's Manual. Developers of IA packages have also produced [Javadocs](#) which currently provide some basic information on some of the underlying libraries and programs that comprise the IA system.

*NOTE: Users should be aware that these are NOT fully fledged help documents and are probably most useful to system developers or advanced users only.*

Currently in development is a "User's Reference Manual" that contains a command dictionary for all available IA programs/classes.

## **1.4 Versioning**

IA is still very much a system under development and this manual will be updated with the regular user release updates of the system. The first version of this manual is associated with User Release v0.3 of the HCSS. Version numbering of the manual will be matched to that of the user release. **So the first manual is version 0.3.**

## **1.5 Previous Versions of IA User's Manual**

V0.3, 22 July 2004 (A. Marston & H. Siddiqui)

V0.3.1, 22 December 2004 (A.Marston)



## 1.6 *What's New*

### **The following new sections were added in v0.3.1**

Section 2.4.1 on updating Versant databases and schema evolution  
Section 2.6.3 on known installation problems.

Updates were included in the following places

Section 2.5.2 Windows installation instructions updated.  
Chapter 4 typo edits

### **The following was added in v0.4**

Introduction: Added full list of contributors.

Chapter 1: Changed chapter 1 to allow for description of updates. Added list of contributors.

Chapter 2: Updated installation information. Provided pointer to HCSS installation.

Chapter 3: Section 3.2.5 was added providing short descriptions on new components added to the Jconsole environment (i.e., session and dataset inspectors).

Figure 3-1 was updated to the new view of Jconsole and Figures 3-2, 3-3 and 3-4 were added in section 3.2.5.

Added section 3.7 on error and exception handling in IA.

Chapter 4: Augmented discussion on classes and methods in section 4.7.

Clarified last paragraph in section 4.9

Added section on script writing in IA (section 4.8).

Chapter 5: Changed required imports section.

Added components on complex and multi-dimensional datasets.

Basic numeric arithmetic moved into chapter 6.

Chapter 6: Changed required imports section.

Added basic numeric arithmetic from chapter 5.

Added two figures illustrating fitting capabilities.

Chapter 7: Updated introduction to reflect new setup of the HCSS.

Extended PlotXY introduction in section 7.2. First example split into two.

Added sections 7.2.1.1 and 7.2.1.2 on handling arrays and datasets in PlotXY.

Updated all examples to present system.

Added section 7.4.2.1 and 7.2.2.2 to better illustrate command axes adjustment.

Chapter 8: Updated required imports section.

Included new subsections on the use of each of the Image operations.

Updated use of numeric2d arrays.

Examples rewritten and extended to include new information on Image and Image operations.

Chapter 9: Updated import information with regard to IA startup.  
Added subsections on “inspector” and “help” packages.

Chapter 10: Updated information regarding required package imports.  
Updated introduction to highlight current FITS usage.  
Examples updated.

Chapter 11 (NEW): Chapter added on time usage within the HCSS and time conversions.  
This is based on the original user HowTo document, heavily revised.

Chapter 12 (previously chapter 11): Revised package imports needed for using databases and examples. Reworded and typo corrected sections 1 to 6.  
Significantly revised (made clearer?) sections on getting Dataframes and Housekeeping (HK) data into an IA session.

Appendix B: Updated listing of classes (including links) available in IA packages.

## **1.7 List of Contributors:**

The following people have contributed to the creation of this manual.

Jorgo Bakker  
Helen Bright  
Jon Brumfitt  
Nicola de Candussio  
Steve Guest  
Rik Huygen  
Tanya Lim  
Andrea Lorenzani  
Anthony Marston  
Wim de Meester  
Craig Porrett  
Hassan Siddiqui  
Michael Wetzstein  
Ekkehard Wieprecht  
Peer Zaal  
Rob Zondag



## Herschel IA Chapter 2

# 2 HCSS Downloading and Installation

In case of any problems during installation please email [hcss\\_help@rssd.esa.int](mailto:hcss_help@rssd.esa.int)

### Chapter 2 Contents

- 2.1 [Introduction](#)
  - 2.2 [Platform](#)
  - 2.3 [Pre-Installation Requirements](#)
  - 2.4 [User Installation Procedure](#)
    - 2.4.1 [Known Installation Problems](#)
  - 2.5 [IA Property Initialization](#)
- 

### **2.1 Introduction**

In this chapter we explain how to download and install the Herschel Common Science System (HCSS) software. For local area networks this is likely to be done by a system manager. The system can then be run by anyone on the network. However, personal versions (e.g., for laptops) can also be set up by a user.

It is hoped that an installation wizard for HCSS will be made available in the near future which will allow initial installations or upgrades. For most users, the only additional software that will need to be installed deals with the Versant databases. If you are not worried about using databases for now then the [User Installation Procedure](#) is probably all you need to follow at present.

This chapter describes how to set up a basic user (or user-as-developer) HCSS environment. A key component of the HCSS is its interaction with local and remote databases storing test data and (later) observations. Upgrading your installation to allow for database interactions is discussed in [Chapter 12](#) of this manual. Chapters 3 and 4 introduce the user to IA/Jython and do not require database interactions.

## 2.2 Platform

The reference platform used for Unit and System testing the HCSS software, prior to release, is now SuSE 9.1 (previously used RedHat 8.0) running on an Intel processor.

Note that this OS version is LSB (Linux Standard Base) v1.3 so theoretically one should be safe using another Linux distribution providing it has been certified LSB v1.3, see: [LSB certified products](#) for more information.

Current platforms also include Solaris and Windows (2000, XP Pro).

## 2.3 Pre-Installation Requirements

The following third-party software is required to be installed prior to run (or develop software for) the HCSS. This software is not included in the downloadable HCSS compressed TAR-file.

- **In many cases users will not require any additional software in order to install and run the HCSS.**
- *ALL USERS:* You will need access to Java JRE (Java Runtime Environment), which can be downloaded from: <http://java.sun.com/j2se> . A Java runtime environment is usually available as standard on most modern computer systems. Java version 1.4 or above is required for use in IA. The reference platform build is version 1.4.2\_06. You can check your version using the terminal command  
  
>> java -version
- *For database usage:* Versant Database System (see notes on Versant in the [full installation instructions](#)) will need to be installed. This will allow setting up databases and accessing databases. Not needed if you are not using HCSS databases. The setup and use of databases within IA is described in [Chapter 12](#).
- *For users of TestControl:* If you are using HCSS in a Herschel instrument testlab environment for ILT/AIV tests then you will also need TclBlend. This can be downloaded from: <http://sourceforge.net/projects/tcljava>
- For users who want to become involved in the development of HCSS, the following should be installed. [NOTE: development of IA/Jython scripts can be done with the HCSS Users software needs noted above. ]
- Java JDK (Java Development Kit), which can be downloaded from: <http://java.sun.com/j2se>
- Versant Database System (see notes on Versant below)
- JavaCC, which can be downloaded from: <https://javacc.dev.java.net/servlets/ProjectDocumentList>
- CVS (client/server version), which can be downloaded from: <http://cvs.cvshome.org/servlets/ProjectDownloadList>

- TclBlend (only needed if you are developing the TestControl package), which can be downloaded from: <http://sourceforge.net/projects/tcljava>
- Together (optional), can be downloaded from: [www.borland.com/together](http://www.borland.com/together)

**Note:** the exact version numbers of the applications listed above, can be obtained from:

<ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/refPlatformVersion>

Please note that you may/will need system administrator support and/or privileges in order to install one or more of the component(s) listed above.

All other third-party libraries required (see the [HCSS reference platform specification](#) for a complete list), can be redistributed and are included in the downloadable HCSS TAR-file, which is sufficient for a HCSS user installation.

For those who are considering HCSS development, the full third-party packages may be required (including its Javadoc, sample code, etc.). A compressed TAR-file containing these libraries (matching the latest reference platform set) can be downloaded from the HCSS ftp area: <ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/refPlatformDownloads>. Alternatively you can download all libraries from the supplier site (most of the URLs can be found in the [HCSS reference platform specification](#)).

You must now configure your environment to include the above listed packages in your `PATH` and `CLASSPATH` environment variables, following the installation instructions provided by the suppliers. In addition, developers should include the JavaCC library 'javacc.jar' in their `CLASSPATH`, because of the way that the HCSS 'jake' tool invokes JavaCC.

## **2.4 User Installation Procedure**

Installation of the HCSS/IA system is relatively straightforward and has recently been simplified for both UNIX and Windows users.

Download the pre-built HCSS release from our Herschel FTP server. This can be done using

1. [web-browser](#)
2. [ftp-client](#)

Once downloaded, some environmental properties need to be set up. Much of this is now handled automatically by running a script either in Windows or UNIX.

*Follow either of the above links for the complete (and most up to date) set of installation instructions for the downloading and installation of IA for either operating system.*

### 2.4.1 Known Installation Problems

The CLASSPATH for the HCSS system can come close to or exceed the limit available on most Windows machines which leads to some HCSS components not running properly. This and other known problems are discussed in.

<ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/Install.html#KnownInstallationProblems>

For the most recent releases of the HCSS, the grouping of external libraries into a single loadable jar file has alleviated this problem completely. Windows users should update their CLASSPATH using the batch file provided in the build available at

```
%HCSS_DIR%\config\setHcssExtLibs.bat
```

where HCSS\_DIR is the directory where the HCSS installation resides.

## 2.5 IA Property Initialization

The HCSS environment that has been set up can be configured to user specifications. This can, for example, change the database being used for interactions or change the memory allocation to `Jconsole` (the prime interface for running the HCSS and IA). For those new to the HCSS it is not necessary to adjust these properties unless database interactions are to be immediately attempted. Later, with more sophisticated interactions, users will want to make changes to their properties. Storage of user properties is in the `.hcss/myconfig` file. Changes can be made to properties while working within the HCSS – no restart is required for the updated properties to be made available. This can be useful when, for example, you are changing the database with which you wish to work.

Properties can be set in the `$HOME/.hcss/myconfig` file with the use of the HCSS tool “Property Generator”. Property setting allowing the use of databases is discussed in [Chapter 12](#) (also see [property generator user manual](#)).

After the initial download, the Property Generator tool is useful to run everytime you download and install a new build, as it will inform you of added properties that are not defined in your property files.



## Herschel IA Chapter 3

# 3 Introduction to Working in IA: Using Jconsole

### Chapter 3 Contents

- 3.1 [Introduction](#)
- 3.2 [Running IA under Jconsole:](#)
  - 3.2.1 [File Menu](#)
  - 3.2.2 [Console Menu](#)
  - 3.2.3 [Edit Menu](#)
  - 3.2.4 [Run Menu](#)
  - 3.2.5 [Help Menu](#)
- 3.3 [Standard Settings for Jconsole](#)
- 3.4 [Using import.py to customize Jconsole](#)
- 3.5 [Programming Loops in Jconsole](#)
- 3.6 [Multiline Statements with Jconsole](#)
- 3.7 [Errors and Exceptions in IA](#)
  - 3.7.1 [Overview of the different libraries used in an IA session](#)
  - 3.7.2 [The error back trace mechanism](#)
    - 3.7.2.1 [The way Jython presents error messages](#)
    - 3.7.2.2 [The way JAVA presents error messages](#)
  - 3.7.3 [The HCSS exception and logging mechanism](#)
    - 3.7.3.1 [Exceptions as thrown from HCSS classes](#)
    - 3.7.3.2 [The HCSS logging mechanism](#)

---

### **3.1 Introduction**

An IA session is typically initiated within a console window. This window will include full help and history for the session. Individual commands can be input to the console using Jython commanding, which is discussed later in this chapter. Alternately, the console allows for the construction and running of complete algorithms based on the [Jython](#) language or even sections/individual lines of algorithms. Since no separate compilation is required, individual lines or sections of algorithms can be checked for validity very quickly. IA scripts that use GUIs can also be started from within `Jconsole`.

In this chapter we discuss how the IA console is initiated, illustrate its capabilities and provide some simple Jython interactions to illustrate its use. We discuss some more detailed IA/Jython capabilities in [Chapter 4](#).

### **3.2 Running IA under Jconsole:**

The majority of IA users can expect to be working within an IA console. After [installing the HCSS](#), the user can start the console by inputting the following at any terminal prompt.

```
> jconsole
```

[For Windows users, open a command window and type in the same thing.]

Note that some feedback from the IA session is provided to the terminal window from which it was started. This includes information on the settings used on `jconsole` startup and information on database access (basically feedback on where interactions occur with systems outside the immediate IA session). The `jconsole` shell performs the following tasks:

- a. Loads a customized Jython environment (imports a set of libraries and defines a set of variables).
- b. Keeps a history of successful Jython statements.
- c. Implements a set of basic editing functions (copy and paste).

It is an extension of the standard Jython shell. Here, we provide some basic startup information

After inputting the `jconsole` command, information on preloaded elements in the IA session appear in the terminal window. Following this feedback, a separate three-paned console window should appear (see [Figure 3-1](#)).



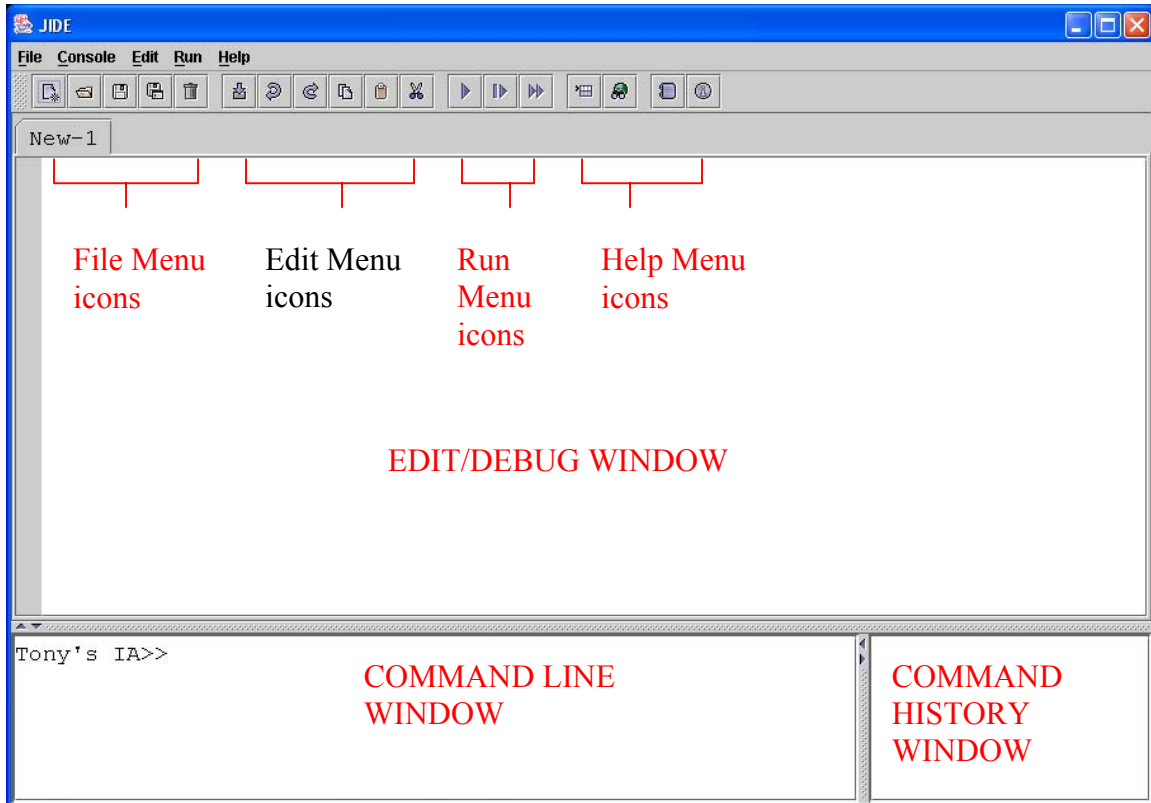


Figure 3-3-1: The Jconsole window set up.

The `jconsole` window has three components to it. An interactive **command line window** is given to bottom left of the console with a “IA>>” prompt. Individual IA/Jython commands can be run here. Click in the bottom left window with your mouse, then type in


```
IA>> print 5 + 3
```

Followed by <Enter>. The answer should be provided on the next line, prior to receiving the “IA>>” prompt back again.

The bottom right of the console contains a **command history window** that lists the commands (including those inside algorithms) used in the current session. Any command highlighted in red caused an error. Some information on the error that occurred can be obtained using the mouse to click on the command highlighted in red. A response with the error is shown in the command line window to bottom left. Try the following

```
IA>> sign 5
```


After hitting <Enter> the user will see the history window has a command highlighted in **red**. Click on this using the left button of the mouse.


The top pane of the console is available for the user to develop his/her own algorithm using the available IA/Jython commands. Click in this window, type in a similar print command to the above example. Hitting return will not run this simple script. To run the one line, click in the grey margin to the left of the line you have typed. An arrow should appear beside the line. Now go to the line of icons and click the single arrow (). This will run your one line algorithm and the result will appear in the lower left command line window (again). If you wanted to "print" a string it needs to be in quotes (e.g., `print "Hello World"`).



Now that we have a brief introduction to the three windows of `jconsole` we will consider each of the menu and icon items in turn.


### 3.2.1 File Menu

Each of the **File menu** items has an associated icon except for exit. These are the first 5 icons on the bar under the menu headings.

**New**  – creates a new window for algorithm development. New history and/or command line windows are not created.

**Open**  – allows a file to be opened in the top window (ASCII – IA/Jython files are stored in ASCII format).

**Save and Save As**  **and**  – for saving the current algorithm shown in the top window.

**Close**  – close the file in the top window pane. Only closes the window showing the current algorithm.

**Exit** – exits from the `jconsole` session.

### 3.2.2 Console Menu

**Execute Line by Line** – this requests the input of an IA script file and runs it line by line


**Execute** – this does a similar thing, except it runs the whole script in one go



**Execute in the background** – this does the same as execute, but runs the script in the background (useful for scripts that take time since the `jconsole` window becomes unavailable while executing a script remaining in the foreground).



**Save history and Save history as...** – saves a history of successful commands from this session using `jconsole`.

### 3.2.3 Edit Menu

Each of the Edit Menu functions has an associated icon at the top of the `jconsole` panel (middle section of icons).


**Import history**  – allows the import of the history of a saved `jconsole` session.


**Undo and Redo**  and  – allows edits (cut/paste or deletion from the keyboard) to be undone or redone.


**Cut and Paste**  and  – the usual cut and paste using the mouse to select and position text.

### 3.2.4 Run Menu

The next three icons at the top of the `jconsole` window relate to the **Run menu**.


**Run**  – runs a single line of script. Click mouse in grey column alongside line of an IA command in the top window that you want executed – the **Run** command from the pulldown menu (or clicking on the Run icon) will execute this line only.

**Run selection**  – select a set of commands by dragging the mouse over them. Pull down to **Run selection** (or click the icon) to run these IA commands only.

**Run all**  – using pulldown or icon, this allows all IA commands in the top pane of `jconsole` to be run in sequence.

### 3.2.5 Help Menu

The last four icons at the top of the `jconsole` window relate to various forms of help that are also available under the Help pulldown menu.

**Dataset Inspection**  – allows the user to view datasets (notably tables) currently available in the IA session in a separate dataset inspection window. See Figure 3-2.

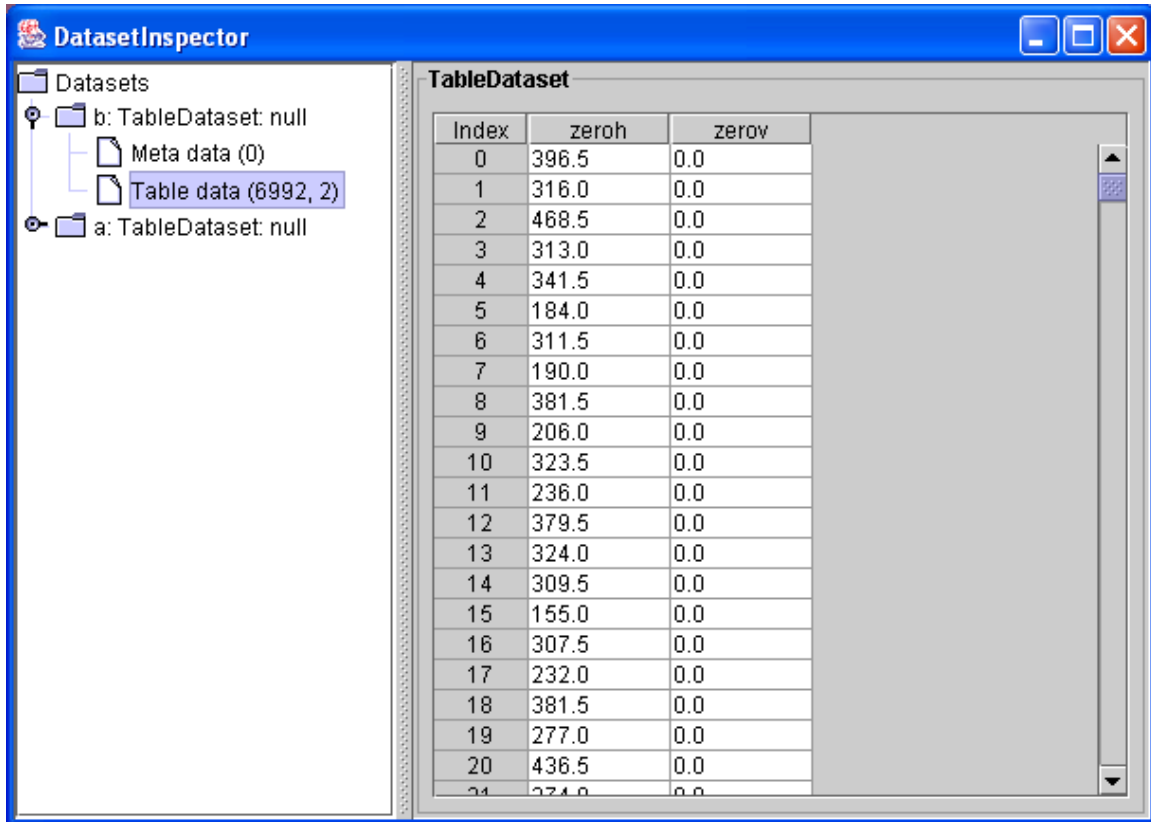



Figure 3-2: To view the contents of table or composite datasets, double click on the Datasets item that appears in the left hand column of the dataset inspection window. Clicking on the appropriate table allows its contents and associated meta data to be viewed.

**Session Inspection**  – allows the user to view the classes (programs) and functions available in the current IA session. . See Figure 3-3. Further classes and functions can be made available by importing “packages” (see Chapter \*).

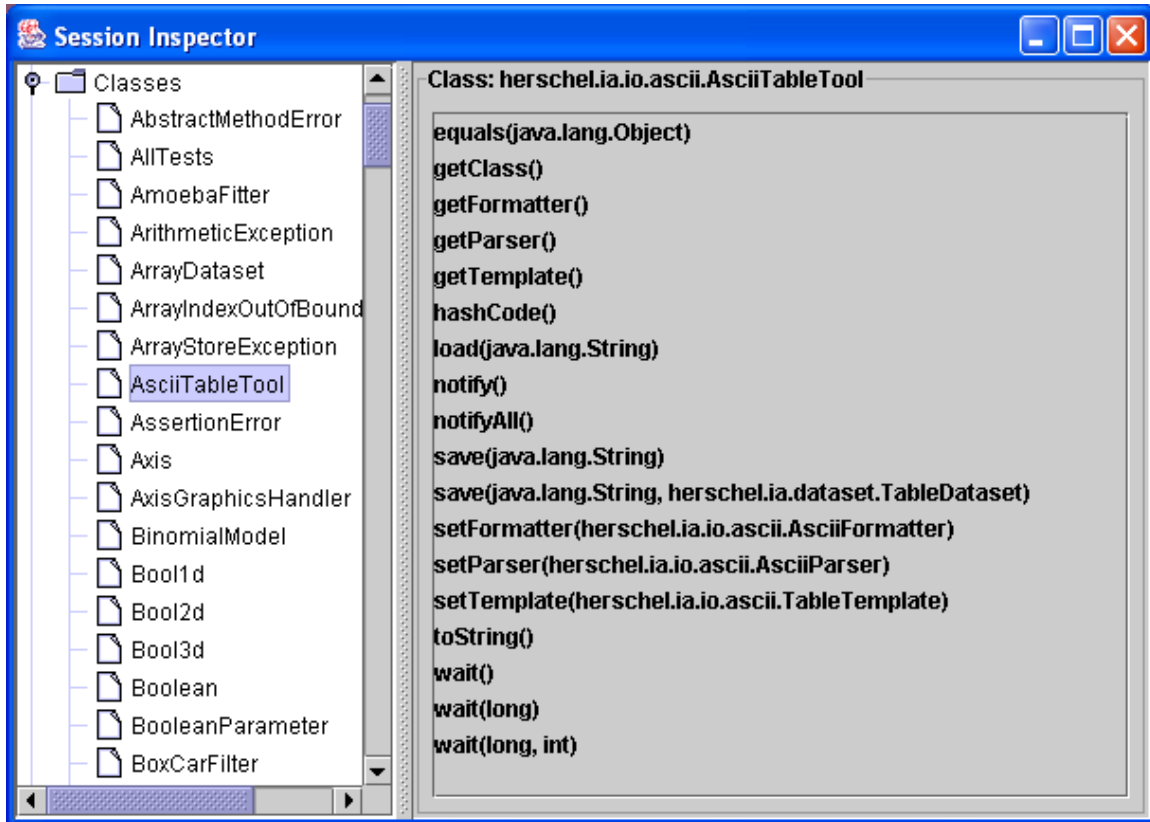




Figure 3-3 An example session inspector window. A click on the nodes shown in the far left of the first column “opens” the folder [in this case “classes”] to show commands currently available in the IA session. Clicking on one of these commands [in this case “AsciiTableTool”] shows the methods available with that class, e.g. AsciiTableTool.load(“table1.txt”) uses the load method of the program to read in an Ascii table while AsciiTableTool.save(“table2.txt”, iatable) saves a table in IA called iatable into an Ascii file called table2.txt.

**Log Window**  – provides a listing of the feedback from running commands by the system, including error messages. These appear in a separate Log window. The log can be saved when exiting from `jconsole`.

**Access to On-line Help Documentation**  – clicking this icon allows access to full set of current on-line (website) documentation in a separate window. See Figure 3-4.

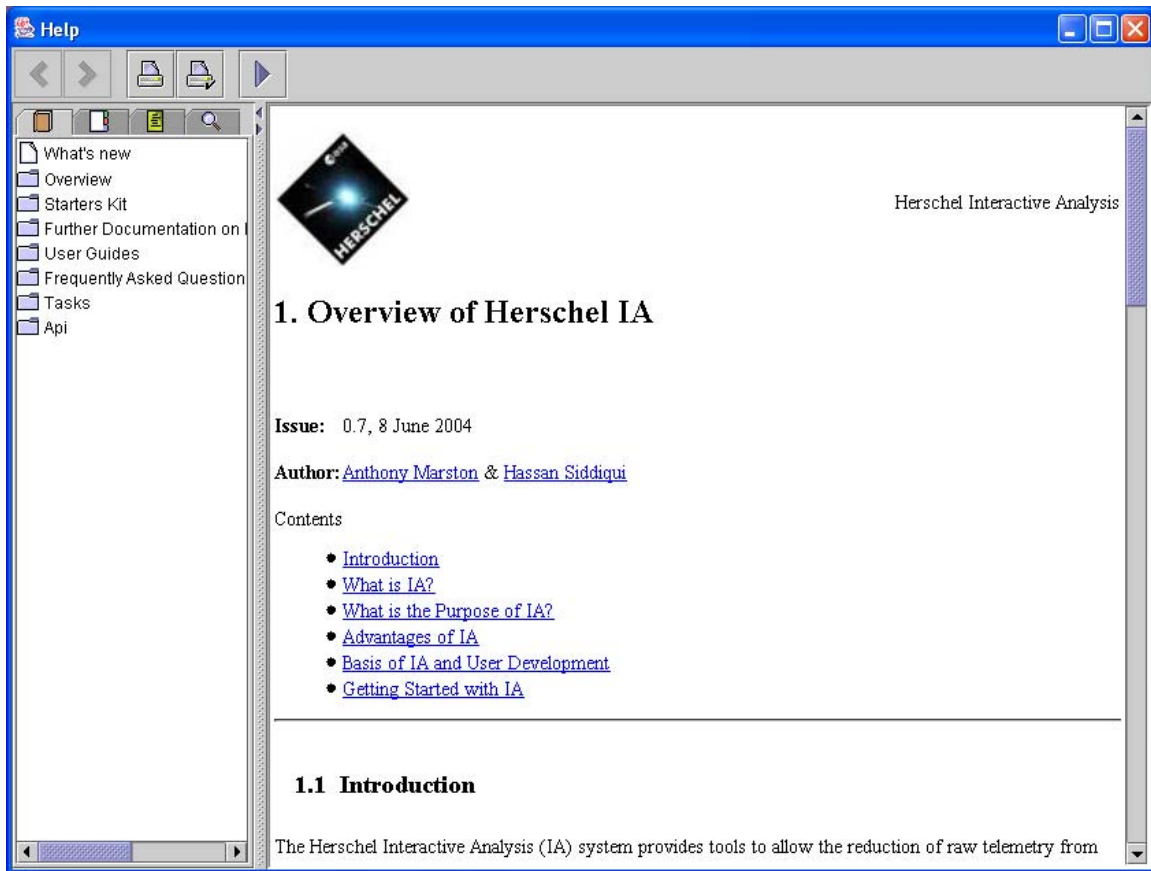


Figure 3-4: On-line documentation is available from within an IA session.

### 3.3 Standard Settings for Jconsole

Jconsole comes with a predefined memory specification of 64MB minimum size and 256MB maximum size. The settings are specified in the startup script for jconsole. This script is located in the \$HCSS\_DIR/bin directory (named jconsole or jconsole.bat for Windows). These settings can be modified by editing the jconsole script.

- a. Change `-Xms64m` to your minimum amount of memory.
- b. Change `-Xmx256m` to your maximum amount of memory.

Make sure that the environment variable `HCSS_PROPS` is properly defined ([see Chapter 2](#)).

Make sure `HCSS_PROPS` contains the specification of the standard `var.hcss.dir` property (this should be the property defined in your `$HOME/.hcss/myconfig` file IF you have set up your own environment and are not using a local network installation). And be sure that `var.hcss.dir` points to the HCSS build directory.

There are several properties for jconsole that are set up during initialization (see under [Set Up](#) in the [Jconsole HowTo](#) document). These can be used to determine such things as

window size. However, window size can be adjusted in the usual fashion by clicking and dragging corners and/or sides of the `jconsole` window.

### 3.4 *Using `import.py` to customize `Jconsole`*

One useful user controlled file for changing the settings of `jconsole` is the use of an `import.py` file. This should be placed in the user's home directory.

The `import.py` file allows a number of packages to be installed in the `jconsole` session during startup and is composed of statements such as

```
import herchel.ia.numeric
```

which imports the numeric package of the HCSS system therefore allowing the commands defined in that package to be used in your current IA session. [NOTE: this particular package is currently loaded automatically on startup of an IA session]

This can save time when running your own standard IA session. The import of HCSS software packages is noted in several places in this manual and is shown in all the sample IA scripts. However, a later chapter discusses the [IA software packages](#) currently available within the HCSS environment.

### 3.5 *Programming Loops in `Jconsole`*

Earlier in the chapter we tried some basic commands to illustrate the components of the Jython window. One particular capability of `jconsole` is allowing block support for Jython. Suppose we want to take a basic print command typed in the command line window.

```
IA>> a = 5  
IA>> print a  
5
```

Now simply input

```
IA>> for i in (1,2,3): <Enter>
```

This will return a “....” response in the command line [NOTE: the colon at the end of the line is important for starting the block]. The command is incomplete. Input a “`print i`” command. A further “....” is returned. Hit Enter once more – the command is now complete.

The whole session should look like (please note the indent prior to the print statement on line 2):

```
IA>> for i in (1,2,3) <Enter>  
..... print i <Enter>
```

```
..... <Enter>
1
2
3
IA>>
```

We could have added a number of commands to this “for” loop. The block statement continues until a blank line is produced. The history of the window is now available. The up arrow will prompt back

```
IA>> for i in (1,2,3):
      print i
```

You can edit this block statement by using the LEFT and RIGHT keys and deleting/adding characters.

Blocks within blocks (nested “for” or “if” loops) are also possible. Basic rules about the use of blocks follow Jython syntax.

1. Each statement in a block must begin in the same column;
2. Each of the Jython key statements and clauses (class, def, for/else, if/elif/else, try/except/else, try/finally and while/else) denotes the beginning of a new block;
3. A new block must be indented at least one space from the enclosing block;
4. The end of a block is marked by having the next statement after the end of the block must begin in the same column as the enclosing blocks.

For example

```
for x in (1,2,3):
    print x # outer block
    for y in (4,5,6):
        if y = 5: #inner block
            print y #inner-inner block
            print x*y #inner block
            #insert inner block statement here
        #insert outer block statement here
```

As usual, end with a blank line! Note the end of each “for” loop is determined by where the indentation ends.

### **3.6 Multiline Statements with Jconsole**

Jconsole also improves on many Jython interpreters in that it allows multiline statements.

To continue a statement onto a second or third line requires only an input of “\” at the end of a line in jconsole.



When including file statements such as

```
["a",  
"b"]
```

or

```
(1 +  
2)
```

either convert to a single line or add a “\” to the end of the first line, e.g.,

```
(1 + \  
2)
```

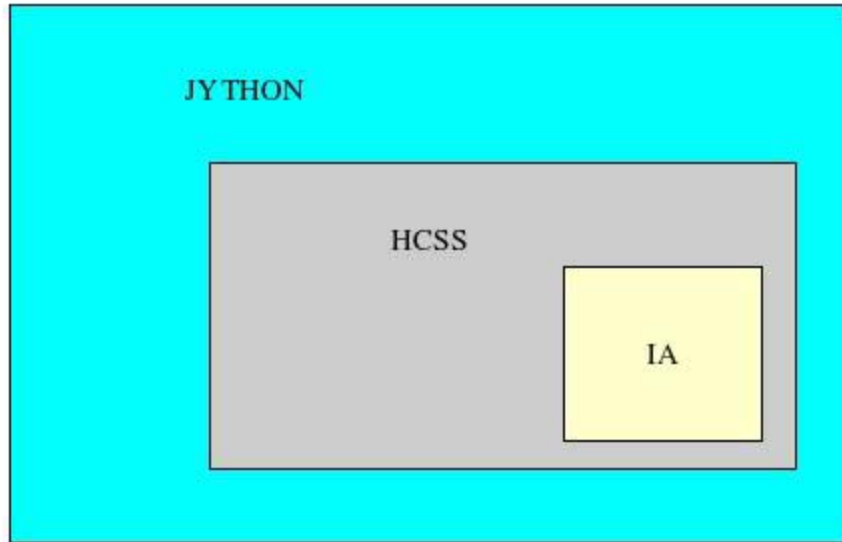
### **3.7 Errors and Exceptions in IA**

Here we explain how errors are generated within IA and how these are reported back to the user. Following from this howto the user should be able to:

- understand error messages as might show up (i) while running an application, or (ii) during an IA session.
- report the error to the custodian of a HCSS module in case an not will described exception occurred, i.e. which cannot be handled by the user

#### **3.7.1 Overview of the different libraries used in an IA session**

The base routines for IA are written in JAVA, but IA user development uses the more friendly [Jython](#). Typical user development is expected to take place in the console panel with plots and images appearing in separate windows. Within an IA session one can run commands from the [Jconsole](#) tool that enables the execution of IA/Jython commands, saves and loads scripts, and provides command history support. This tool often provides the core of a user's IA session.



**Figure 3-5: The global structure within an IA session .**

Figure 3-5 shows the overall library structure for an IA session. Errors, as thrown by Jython and or JAVA classes, have the same back trace mechanism (however, they differ in the way they present error messages to the user, as shown in the next section).

Interpretation of these error messages allows the user to the class from which the exception/error originated from.

### **3.7.2 The error back trace mechanism**

Here is described the difference in the way Jython and JAVA libraries present error messages.

#### **3.7.2.1 The way Jython presents error messages**

An example of how Jython presents error messages:

```
IA> array =[1,2,3,4,5]
IA> print array[5]
Traceback (innermost last):
  File "<string>", line 1, in ?
  IndexError: index out of range: 5
```

### 3.7.2.2 The way JAVA presents error messages

Most IA packages use JAVA classes. If JAVA classes are run within an IA session and an error occurs, an exception is thrown which is propagated upwards to the Jython level. An example:

```
IA> dbl = Double("wrong arg")
Traceback (innermost last):
  File "<string>", line 1, in ?
    java.lang.NumberFormatException: For input string: "wrong
arg"
      at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
      at
java.lang.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1207)
      at java.lang.Double.valueOf(Double.java:202)
      at java.lang.Double.<init>(Double.java:277)
      at
sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
      at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)
      at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:27)
      at
java.lang.reflect.Constructor.newInstance(Constructor.java:274)
      at
org.python.core.PyReflectedConstructor.__call__(PyReflectedConstructor.java)
      at
org.python.core.PyJavaInstance.__init__(PyJavaInstance.java)
      at org.python.core.PyJavaClass.__call__(PyJavaClass.java)
      at org.python.core.PyObject.__call__(PyObject.java)
      at org.python.pycode._pyx6.f$0(<string>:1)
      at org.python.pycode._pyx6.call_function(<string>)
      at org.python.core.PyTableCode.call(PyTableCode.java)
      at org.python.core.PyCode.call(PyCode.java)
      at org.python.core.Py.runCode(Py.java:1136)
      at org.python.core.Py.exec(Py.java:1158)
      at
org.python.util.PythonInterpreter.exec(PythonInterpreter.java)
      at
herschel.ia.jconsole.jython.Interpreter.exec(Interpreter.java:261)
      at
herschel.ia.jconsole.jython.Interpreter.exec(Interpreter.java:244)
```

### 3.7.3 The HCSS exception and logging mechanism

Next to the standard JAVA exception handling mechanism the HCSS is using, it also has a logging mechanism which forwards info, error and warning messages to the user.

### 3.7.3.1 Exceptions as thrown from HCSS classes

In case an error occurs inside the HCSS, for example due to a missing or wrong defined configuration variable, the information as part of the exception thrown should explain the user what did cause this exception. In this way the user should be capable to adjust its input arguments and/or property settings. For example:

assume the user has set the configuration variable:

```
var.database.devel = "idonotexist@iccdb.sron.rug.nl"
```

when trying to access this database in a IA-session by:

```
>> IA: from herschel.hifi.generic.task import *  
>> IA: tm = AccessPacketTask()(obsid=8, apid=1026)
```

Here, a query is done on the database as set by the above property and the exception as thrown reads:


```
Traceback (innermost last):  
  File "<string>", line 1, in ?  
    java.lang.RuntimeException: java.lang.ClassNotFoundException:  
    Exception in constructor of  
herschel.access.db.LocalConnection: herschel.access.LocationException:  
    Failed to get store herschel.store.api.StoreException:  
    Failed to create store for idonotexists@iccdb.sron.rug.nl:  
herschel.store.api.StoreException: {  
    VException(7001:UT_DB_NOT_FOUND: DB directory not found) }  
    at herschel.hifi.generic.task.AccessPacketTask.execute  
(AccessPacketTask.java:151)  
    {full trace back list}
```

In cases where the information as passed by the Exception thrown is not sufficient (for example a `NullPointerException` without any textual explanation), then there is a problem with the current system and the user is encouraged to provide feedback to the HSC regarding the lack of exception handling information (currently, this is best achieved through the SPR/SCR system).

In the above example the "access" package might improve its exception notification by adding information to the `LocationException`, including a hint for the user that the database is not existing and that the user should check whether `"var.database.devel"` is properly defined.

### 3.7.3.2 The HCSS logging mechanism

The logging mechanism allows (HCSS) classes to pass errors, warnings and/or info to the end-user. To enable the error logging mechanism, go to the Help menu or click on the

 icon (see also 3.2.5).

For the HCSS end-user this mechanism will be used more often , especially when HCSS software is fully matured. The difference between the two is that exception handling is more often used by the developer for debugging purposes, whereas the logging mechanism is intended to be used by the end-user to get insight in the behaviour of an (HCSS) application or class. The logging mechanism enables the developer to includes messages when an exception is thrown and on how the class internally handles possibly thrown exceptions.

To give an example why, next to the exception mechanism, the logging mechanism was introduced: suppose we have a layered HCSS component (i.e. within an instance of a class there are calls to instances of other classes and these will call others on their turn), deep within this component an exception occurs and at a higher level this exception is caught again. In such a scenario the end-user of the component will not be aware of the fact that this exception occurred. However, by use of the logging mechanism the developer of the component can pass a message (an error, warning or info ; depending on how severe this exception was) next to the exception thrown, as well the developer is able to pass relevant information to the user when the exception is caught.

More detailed information on the logging mechanism: [herschel.share.log.api.Log](#) (link to HCSS javadoc)



## Herschel IA Chapter 4

# 4 Some IA Basics & Beginning Jython

### Chapter 4 Contents

- 4.1 [Basics](#)
- 4.2 [Lists and Dictionaries](#)
  - 4.2.1 [Setting up and Accessing Lists](#)
  - 4.2.2 [Slicing Lists and Arrays](#)
  - 4.2.3 [Setting Up and Using Dictionaries](#)
  - 4.2.4 [Nested Dictionaries](#)
- 4.3 [Augmenting Values and Arrays](#)
- 4.4 [Printing to the screen and files](#)
- 4.5 [Defining and Using Functions](#)
- 4.6 [Blocks and programming loops](#)
- 4.7 [Classes and Methods](#)
- 4.8 [Writing Scripts – Programming in IA](#)
- 4.9 [Some Useful Extra Items](#)
  - 4.9.1 [Handling Arrays and Other Datasets](#)

---

### 4.1 *Basics*

The Herschel IA is a development that is based on programs written in Java or Jython. Jython is a Java implementation of the [Python](#) language. The syntax is therefore well defined and there is plenty of documentation freely available.

Variables don't have to be declared (integer `x`, `xmax` etc. is not required). They appear when you assign to them and disappear when you don't use them anymore. Assignment is done by the `=` operator and equality testing is via the `==` operator. You can also assign several variables at once.

```
IA>> x, y, z = 1, 2, 3
IA>> a = b = 123
```

Comments on a line can be added after a hash (`#`) mark.

Blocks are indicated by indentations and only through indentations (and can be handled within `jconsole` – [see Chapter 3](#)). No begin/end braces are required.

## #Examples

```
IA>> if x < 5 or (x > 10 and x < 20):
IA>> ... print "The value is OK"

IA>> if x < 5 or 10<x<20:
IA>> ... print "This value is OK"

IA>> for i in [1,2,3,4,5]:
IA>> ... print "This is iteration number", I
```

The first two examples are identical.

The above “for” loop goes through values in a list indicated in the square brackets. A simpler way – particularly for large numbers of iterations – is to use the inbuilt range function.

#The following prints from 0 to 99

```
IA>> for value in range(100):
IA>> ... print value
```

Note how values start from 0 and end one below the value assigned to the range function. Currently, the print output is going to the command line window of Jconsole.

## 4.2 Lists and Dictionaries

Lists and dictionaries are important data structures available in Jython.

**Lists** are simple arrays written in a specific order.

**Dictionaries** are like lists that can be accessed via a key (or label). To access an element you use a key or “name”. This is what is used to look up the value of an element.

### 4.2.1 Setting up and Accessing Lists

Lists are formulated within square brackets, which can be nested. E.g.,

```
IA>> name = ["Rolf", "Harris"]
```

(note – strings of characters need to be placed inside quotation marks)

```
IA>> x = [[1,2,3],[y,z],[[[]]]]
```

You can access lists by individual names or groups

```
IA>> print name[0], name[1] # prints "Rolf Harris"
```

```
IA>> print name[0:2] # gives list in brackets ['Rolf', 'Harris']
```

```
IA>> print name[:2] # ditto
```

In the first instance the parts of the “name” list are picked up individually, in the second part a range of list components is picked out (0 to 2) and in the last case all components up to `name[2]` are picked out. Notice how in the last two cases the command is interpreted as going up to but not including the number range being given. We can try the same with the list “x”.

```
IA>> print x[0] # gives the first element in the list "[1,2,3]"
```

Try printing the other elements of the list (`x[1]` and `x[2]`) to see if you get what you expect!

### 4.2.2 Slicing Lists and Arrays

The last two examples using the list “name” (above) are also examples of slicing. Slicing of this type can also be performed with numerical and string arrays. For instance,

```
IA>> y = ["The", "quick", "brown", "fox", "jumped", "over", \
"the", "lazy", "dog"]
IA>> print y[1:4] # prints the list ['quick', 'brown', 'fox']
```

Again – the end integer value given for the slice is not included, so the above example only gives the values for `y[1]`, `y[2]` and `y[3]`.

- Choosing `y[:4]` means “take every element from the beginning of the list up to element 4, *not including element 4.*”
- We can also to have `y[4:]` which means “take every element from number 4 up to the end” – note that this will *include* element number 4.
- Lastly, negative numbers mean count from the end of the list – `y[-3]` means take the third element from the end of the list.

### 4.2.3 Setting Up and Using Dictionaries

A dictionary has a set of {key: value} pairs. E.g.,

```
IA>> person = {"Alice" : 111, "Boris": 112, "Clare": 113, \
"Doris": 114}
IA>> print person.get("Alice")
111
```

...so we “get” the associated value within the dictionary “person”. To see all the “keys” use



```
IA>> print person.keys()
```

and to get all the values

```
IA>> print person.values()
```

The use of the empty brackets at the end indicate that we are not passing a parameter on to “keys” or “values” in order to get a printout of their current settings. In fact, no parameters are allowed for these commands, but we still need the brackets.

Also note how the commands “keys()” and “values()” are applied/work on the dictionary “person”. We will see this frequently when running IA code in the future.

If we want to change the dictionary then we need to write something like

```
IA>> person['Alice'] = 222
```

Here, the value associated with Alice in the dictionary called person has been changed to the number 222.

#### 4.2.4 Nested Dictionaries

Dictionaries can hold other dictionaries too. So advanced data structures can be made.

Let's set up a dictionary called `abc`

```
IA>> abc = {"John": 12345, "Jerry" : 23456, "Joe" : 34567}
```

We will now put this inside another dictionary called `dict`

```
IA>> dict = {"Alice" : 111, "Boris" : abc, "Charlie" : "angel"}
```

Note here that we have NOT got inverted commas around the value `abc` since we want it to point to our dictionary `abc` and not be a string.

So now we can look at the value of “Boris”

```
IA>> print dict.get("Boris")
```

Which should simply give us the dictionary `abc` printed on our screen. Whereas,

```
IA>> print dict.get("Charlie")
```

Simply prints the string we gave as the value (we know it's a string since it has inverted commas around it).

If we now want to get the value of “John” we would need to do

```
IA>> print dict.get("Boris").get("John")
```

First we get the dictionary `abc` which is pointed to by the key "Boris", then we look for the key "John" inside. This returns the value 12345.

### 4.3 *Augmenting Values and Arrays*

Jython allows a full range of augmentation assignment operators (including `+=`, `-=`, `*=`, and `/=`). These all behave in a similar fashion.

```
IA>> a = 5
IA>> a += 2      # adds 2 to the value of a
IA>> a *= 3      # multiplies a by 3
```

We can add to arrays too.

```
IA>> b = [1]
IA>> b += [2]    # now b = [1, 2]
```

Note that here we have appended an element to the end of the list.

### 4.4 *Printing to the screen and files*

We have already seen how a print command can produce a result

```
IA>> print 1, 2, 1+2
1 2 3
IA>> print a
[1, 2]
```

(... following on from the above augmentation example).

We can also print to a file.

```
IA>> file = open("output.txt", 'w') # 'w' allows write access
IA>> print >> file, 2      # puts the number 2 into output.txt
```

Or

```
IA>> print >> file, a      # puts the array "a" into output.txt
```

For printing an array/list to a file.

Note that it is not necessary to close access to a file within your IA session. If you want to overwrite the original text file then reopening the file will remove the contents.

## 4.5 *Defining and Using Functions*

Here we name a piece of code, call it with some parameters and have it return a result. Functions are set up with the keyword `def`. e.g.,

```
IA>> def square (x):  
IA>> ...return x*x  
  
IA>> print square(2) # prints the result of 4
```

In actual fact, IA has a sophisticated numeric functions package that can allow squaring of values and arrays of various types (double, integer etc.). Numeric functions available in IA are discussed in [Chapter 6](#).

If you want to call a function without arguments then the `()` brackets are required.

A useful thing to know is that functions are values in Jython. So taking an example from the previous section

```
IA>> print person.values()
```

Could be changed to

```
IA>> pvalue = person.values  
IA>> print pvalue()
```

## 4.6 *Blocks and programming loops*

Programming loops can be done in Jython with the use of blocks. These were discussed in Chapter 3.5, where their use within the `jconsole` environment was illustrated. Blocks are used with “*for*” loops, *while/else* loops and conditional (*if/elif/else*) statements.

## 4.7 *Classes and Methods*

In this section we introduce some jargon used in Jython programming and IA, which are object-oriented languages.

For non-object oriented programming thinkers, **classes** are like programs that contain callable subroutine components (referred to as **methods**) that will be applicable to an object (typically an array). This is probably best illustrated with an example. The following is an example that can be placed in the top pane of `jconsole`. Remember to keep proper/accurate indentation.

```
class Basket:
    # always remember the self argument
    def __init__(self, contents=None):
        self.contents = contents or []
        #this bit does a logical or - if a parameter is passed to it,
        #it becomes the contents, otherwise
        #we get an empty basket!
    def add(self, element):
        self.contents.append(element)
        #this adds the "element" to the contents (self.contents)
    def print_me(self):
        result = ""
        for element in self.contents:
            result = result + " " + `element`      #NOTE use upper left
                                                    #keyboard single inverted
                                                    #commas around element.
        print "Basket contains: "+result
```

We have created a class called "Basket" and it has two associated methods "add" and "print\_me" (following "def" in the above example).

Try placing the above within the top pane of `jconsole`. Here we create an object to work on, called "self" – which is customary. This is initiated by the `def __init__` command (by the way, that's two underscores on either side of `init`).

Leave a blank line at the end of the script when placing it within the edit pane of `jconsole`. Now hit the double arrow icon to load this into your IA session.

Once created, we can run the class by typing `Basket()` in `jconsole` via the command window (bottom left).

Now try the following in the command line window.

```
IA>> a = Basket()
#this line sets up an empty basket which we have called "a"
IA>> a.add("saw")
#this line adds the item "saw" to the basket. It runs the "add"
#method on the object "a".
IA>> a.add("hammer")
#...and now we have added "hammer"...
IA>> a.print_me()
# prints the contents of the basket we called "a", which
# should be 'saw' and 'hammer'. This runs the "print_me" method
#on the object "a"
```

We could equally have started our basket with one item

```
IA>> a = Basket("saw")
```

Basically we have `object.method(arg1, arg2)`

In the above case “a” is the object and we have the methods “add” and “print\_me”.

`__init__` is a special method that is said to be a constructor setting things up in the first place. The **constructor** (initial call to the routine) creates an **instance** of the object (in the above case it creates a basket we can put things in).

## 4.8 Writing Scripts – Programming in IA

Scripts take individual IA statements and combine them to make more complex routines. The user can edit a script directly in the edit/debug window of `Jconsole`. A series of IA commands/instructions can then be input and then run in the IA environment.

Following on from our Basket example. If the class Basket has already been created we can create a script that uses it. For example, we can place the following in our `Jconsole` edit pane.

```
a = Basket()
a.add("saw")
a.add("hammer")
a.add("chisel")
b = Basket()
b.add("bread")
b.add("cheese")
b.add("milk")
a.print_me()
b.print_me()
```

Now if we hit the “Run all” button then we create two baskets the contents of which will be printed to the command window (bottom left).

This script can be saved using the “File” pulldown menu or save icon (default is “.py” extension).

## 4.9 Some Useful Extra Items

- Some arguments can be optional and can be given a default value. E.g.,

```
IA>> def spam(age=32)
```

Here, `spam` can be called with zero or one parameters. If zero parameters then it will be called with the default parameter of `age=32`. If a parameter is given with the call then that will be assigned to “age” instead.

- Backquotes (top left of keyboard) convert an object to its string representation (so the number 1 can be converted to string "1").
- The + sign can be used to append string lists.
- One change to make printing easier. We can change to the special method `__str__` so that our last function starts with the line

```
def __str__(self):
```

Instead of

```
def print_me(self):
```

Now we can use

```
IA>> print a
```

to show our basket contents rather than

```
IA>> a.print_me()
```

Most useful classes and functions are put into modules or packages. These are then imported into a given environment or program with the line(s), e.g.,

```
import math  
from math import sin, cos
```

This is the means by which classes and functions are brought into the IA environment from within already existing packages. The above two lines show how to import the `math` package and how to just import `sin` and `cos` classes from the `math` package into your current IA session.

A basic set of packages most relevant to users is loaded when an IA session is started. Other packages can simply be imported into a user's session (or included in an `import.py` file that automatically imports packages when IA is started – see [Chapter 2](#)).



# Herschel IA Chapter 5

## 5 Handling Arrays and Other Datasets

### Chapter 5 Contents

- 5.1 [Introduction](#)
- 5.2 [Getting started](#)
- 5.3 [Types of Array Datasets](#)
- 5.4 [Creating a Simple 1D Array Dataset](#)
- 5.5 [Dataset attributes](#)
- 5.6 [Simple 1D Array Manipulation](#)
  - 5.6.1 [1D Array Arithmetic](#)
  - 5.6.2 [Logical Operations](#)
  - 5.6.3 [Type Conversion](#)
- 5.7 [Dealing With Complex Arrays](#)
- 5.8 [Creating and Accessing Multi-Dimensional Array Datasets](#)
- 5.9 [Creating and Viewing a Table Dataset](#)
- 5.10 [Creating and Accessing a Composite Dataset](#)
  - 5.10.1 [IA Numeric: Basic Functions for Herschel IA](#)

---

### 5.1 Introduction

This chapter aims to familiarize the user with the IA Datasets and Algorithms concepts. This is not an exhaustive reference to all the functionality provided, the full set of available dataset capabilities are discussed in the *herchel.ia.dataset* package.

There are three types of basic datasets:

- array datasets (arrays of numbers, strings, etc. in 1D, 2D, 3D, 4D or 5D)
- table datasets (x rows by y columns of numbers, strings etc.)
- composite datasets (combines multiple connected arrays/tables in a single dataset).

In this chapter, we discuss how to formulate and use each dataset type.

## 5.2 Getting started

All classes and methods associated with handling datasets and numeric functions are automatically loaded when the IA session is started in this manner.

The IA *numeric* package currently contains many functions and is discussed in more detail in [Chapter 6](#). Here we include the use of portions of it to help illustrate how datasets may be handled.

## 5.3 Types of Array Datasets

Numeric array datasets can have up to 5 dimensions and have the types noted in Table 5-1.

**Table 5-1: Dataset types available in IA**

Name	type	Dimensions		
		1	2	3+
BoolNd	boolean	yes	yes	yes
ByteNd	byte	yes	yes	yes
ShortNd	short	yes	yes	yes
IntNd	int	yes	yes	yes
LongNd	long	yes	yes	yes
FloatNd	float	yes	yes	yes
DoubleNd	double	yes	yes	yes
ComplexNd	Complex	yes	yes	yes
StringNd <sup>1)</sup>	String	yes	NO	NO
5.10.1.1	not strictly numeric			

In order to create an array dataset we only need to do something like the following.

```
IA>> a = Int1d()
```

This provides us with an empty integer array. We can now add elements to this by

```
IA>> a.append(2)
```

Or

```
IA>> a.append(Int1d([1, 2, 4, 5, 6]))
```



To append a whole 1D integer array.

Alternately, we could have created the array in one go....

```
IA>> a = Int1d([1,2,4,5,6])
```

The following show various ways in which numeric 1D arrays can be created in the IA environment.

```
y = Double1d([1.0,2.0,3.0,4.0])      # Create from a Jython array
y = Double1d(4)                      # [0.0,0.0,0.0,0.0]
y = Double1d(4, 42.0)                # [42.0,42.0,42.0,42.0]
y = Double1d.range(4)                # [0.0,1.0,2.0,3.0]
```

## 5.4 Creating a Simple 1D Array Dataset

Let's start by creating a simple dataset. Let's assume that we want to create a dataset containing one component: a 1D array of real numbers (which are held as doubles in an array we will call 'x').

Type in the following steps (without the comments preceded by '#'):

```
IA>> x=Double1d.range(10)  #'range' creates a 1D array of integers
                           #with the values 0, 1, 2...9  Putting
                           #Double1d in the front converts the array
                           #values to doubles.
IA>> s=ArrayDataset(data=x,description="range of real values")
#this actually creates the array dataset with data being the
#array x of values 0.0, 1.0, 2.0...9.0 and some associated
#information, a description.
```

This creates an object *x*, corresponding to a 1D array of 10 real numbers from 0...9, and writes that to a dataset object, *s*, which also contains a description of the dataset. In this example, a simple description is also written to the dataset. The range command produces ten integer numbers from 0 to 9. This is placed in a 1D array of real numbers (doubles) by the first line.

Now let's look at the contents of the dataset *s*:

```
IA>> print s
```

If you want to be specific and print individual components of the dataset, you may do so using the special `description` and `data` attributes:

```
IA>> print s.description      #just print the description that
                              #you attached to the array
IA>> print s.data             # print only the data contained in the array
```

And even individual elements of the data component:

```
IA>> print s.data[2]           #view the data value  
                                #of the third element in the array
```

## 5.5 Dataset attributes

In the previous section, we have seen that `ArrayDataset`, `s`, possesses at least 2 attributes: `description` and `data`. They have in addition a third attribute not so far illustrated, `meta`. The `description` and `meta` attributes are common across all dataset types.

**The `description` attribute** is used to store a human-readable text that helps the user to understand the role of the dataset.

**The `meta` attribute** stores a map of keyword-value pairs of data that can be used to identify that data in database (for example) - the so-called *meta-data*. Examples of metadata include the observation *date of the current observation*; *the name of the source*; *the coordinates of the source*, etc. **These are basically the IA equivalent of FITS keywords**. The allowed data types for meta-data elements are String, Double Boolean, Long, and Date (e.g., **StringParameter**, **DoubleParameter** etc.). See the JavaDoc on the class [MetaData](#) for more information on the allowed types.

The following code snippet shows how to add parameter information (in the form of strings or doubles) to the `meta` attribute:

```
IA>> s.meta["observation"]=StringParameter("NGC 4151")  
IA>> s.meta["principal investigator"]=StringParameter\  
    ("Anthony Marston")  
IA>> s.meta["ra"]=DoubleParameter(182.836)  
IA>> s.meta["dec"]=DoubleParameter(39.405)
```

## 5.6 Simple 1D Array Manipulation

Datasets can be manipulated using basic arithmetic and numeric functions, e.g., addition, subtraction etc. We can also do explicit type conversion with relative ease.

### 5.6.1 1D Array Arithmetic

All arrays currently support the following arithmetic operators:  
`+`, `-`, `*`, `/`, `%` (modulo), `**`

#### 5.6.1.1 Addition, subtraction and concatenation:

Arrays and constant values can be added and subtracted in a similar way, i.e.,

```
IA>> a = s + b
IA>> a = s - b
# if b is another numeric 1D array of the same dimension as s then a is just an array of
#the differences between each element of the two arrays.
#if b is a constant then that value would be subtracted from each array element of s.
```

Note that `s` and `b` need to be numeric arrays. In Jython, the above leads to concatenation.

For example:

# Adding Jython arrays

```
[0,1,2,3] + [4,5,6,7] # [0, 1, 2, 3, 4, 5, 6, 7]
```

# Adding numeric arrays

```
Double1d([0,1,2,3]) + Double1d([4,5,6,7]) # [4.0,6.0,8.0,10.0]
```

# Concatenate two numeric arrays

```
Double1d([0,1,2,3]).append(Double1d([4,5,6,7]))
# [0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0]
```

# Adding Jython arrays to numeric arrays

```
[0,1,2,3] + Double1d([4,5,6,7]) # [4.0,6.0,8.0,10.0]
Double1d([0,1,2,3]) + [4,5,6,7] # [4.0,6.0,8.0,10.0]
```

### 5.6.1.2 Multiplication:

```
IA>> m = 100 * s
#simply multiplies the size of each element of the array s by 100.
```

### 5.6.1.3 Array length:

```
IA>> print len(a)
#prints the length of array 'a'.
```

### 5.6.1.4 General Application of Functions:

We can apply more sophisticated functions such as MIN (get minimum of an array) or SIN (to get an array which has the sine of each array element value as an output), e.g.,

```
IA>> smin = MIN(s)
IA>> print smin
```

The available numeric functions and their use are discussed in [Chapter 6](#).

## 5.6.2 Logical Operations

The following relational operators are also provided, which return a `Boo1ld` array:

<, >, <=, >=, ==, !=

For example:

```
y = Double1d([0,1,2,3,4])
print y > 2                # [false,false,false,true,true]
```

### 5.6.3 Type Conversion

We can change between the various 1D array types (integer, long, double etc.) in a straightforward manner. As is illustrated by the following

```
IA>> i = Int1d([1,2,3])    # [1,2,3]
IA>> r = Double1d(i)       # [1.0,2.0,3.0]
IA>> c = Complex1d(r)      # [(1.0,0.0j),(2.0,0.0j),(3.0,0.0j)]
IA>> b = Byte1d(r)         # [1,2,3]
```

## 5.7 Dealing With Complex Arrays

The numeric library has a `Complex` class and a `ComplexNd` class for N-dimensional arrays of complex numbers.

```
z = Complex1d([1,2,3,4],[4,3,2,1]) # set up complex array
print z                            # [1.0+4.0j,2.0+3.0j,3.0+2.0j,4.0+1.0j]
print z.real()                     # print real part
print z.imag()                     # print imaginary part
print z.conjugate()                # [1.0+-4.0j,2.0+-3.0j,3.0+-2.0j,4.0+-1.0j]
print z * z                        # [-15.0+8.0j,-5.0+12.0j,5.0+12.0j,15.0+8.0j]
```

Complex numbers in the numeric package are constructed using the `Complex` constructor (with an upper-case 'c'):

```
z1 = 2 + 3j                        # Jython complex (2+3j)
z2 = Complex(2,3)                  # Numeric Complex (2.0+3.0j)
```

The following example illustrates that these may be mixed in expressions:

```
z = Complex1d([1,2,3,4],[4,3,2,1])
x = 3 + 4j                          # Jython complex
print z + x                          # Add x to each element
```

In other respects, `Complex1d` arrays are used in much the same way as `Double1d` arrays. Their main use at present is for discrete Fourier transforms.

## 5.8 *Creating and Accessing Multi-Dimensional Array Datasets*

Creating and manipulating multi-dimensional arrays occurs in a similar way to the 1D case. The IA numeric library supports arrays of up to 5 dimensions. For example, to create a `Double2d` array:

```
x = Double2d([[2, 4, 6], [1, 3, 5]])
```

Multi-dimensional arrays are conceptually arrays of lower-dimensional arrays. For a two-dimensional array, the first subscript selects a row and the second subscript selects an element within that row (the column). Note that this is the opposite order to some other computer languages, but it is the same behaviour as in the Java programming language.

For example:

```
print x[1,:]          # Get row 1 i.e. [1.0,3.0,5.0]
print x[1,2]         # 5.0, the element in row 1, column 2
```

Individual elements or slices can be set as follows:

```
x[1,2] = 22          # Set an element in place
x[0,1:3] = 42
print x              # [[1,42,42], [5,6,22]]
```

Array-style operations can be applied to multi-dimensional arrays, in the same way as for one-dimensional arrays:

```
x = Double2d([[2, 4, 6], [1, 3, 5]])
print x + 1          # Add a scalar to each element
print 1 + x
print x + x          # Add two arrays element-by-element
print SQRT(x)        # Map a function over the elements
print MEAN(x)        # A function returning a scalar
print SIN(x[1,:]) * 2 + 1 # An expression
```

It is possible to set a row to a copy of a 1d array of the same length:

```
x[0,:] = [5,6,7]     # Set a row to (a copy of) a Jython array
x[1,:] = Double1d([9,7,6]) # Set a row to a Double1d array
```

**Advanced tip:** It is possible to use the Java API to modify a row in-place, without copying the array:

```
x[1,:].mApply(SQUARE)
```

## 5.9 *Creating and Viewing a Table Dataset*

What is often required is to store data in a tabular format with N columns. The `TableDataset` provides such a means. A `TableDataset` is made up of a number of columns. Each column contains an `ArrayDataset`. Each `ArrayDataset` can have up to 5 dimensions. In the following example, a `TableDataset` is created 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. Note that `TableDataset` type **requires** all column lengths to be the same.

```
IA>> x=DoubleIcd.range(100)
IA>> t=TableDataset(description="This is a table")
#this sets up the table
IA>> t["x"]=Column(x)
#this creates our first column and just has the data, x
IA>> t["sin"]=Column(data=SIN(x),description="sin(x)")
#we have applied the SIN function from the numeric package here
#we have also added a description for the second column
IA>> t["y"]=Column(data=x*100,description="x*100")
#Ditto the third and final column
```

The following steps show how the data can be viewed (plotting the data graphically is discussed in [Chapter 7](#)):

```
IA>> print t
IA>> print t.meta
IA>> print t["y"] # print a column by name
IA>> print t[2] # print a column by index
IA>> print t[2].data
IA>> print t[2].data[4]
# print element with index=4 in the last(!) column
IA>> print t[2].description
```

And modified:

```
IA>> print t["y"].data[0]
IA>> t["y"].data[0]=999.
IA>> print t["y"].data[0]
```

## 5.10 *Creating and Accessing a Composite Dataset*

The `ArrayDataset` and `TableDataset` types enable the user to create arrays and tables of primitive data types easily. However, they do not allow arbitrary structures of data, or data within data, to be constructed. Examples of complex datasets are grouped observations (making a map with an offset reference position, for instance), which could have 1D and 2D array data together with a table which might contain (for example) calibration data. Such complex structures can be built using the `CompositeDataset`.

[Example 5.1](#), below, creates a CompositeDataset containing in turn an ArrayDataset, a TableDataset, a few StringParameters, and another nested CompositeDataset. It also illustrates how we can access the components of the composite dataset.

```
#Example 5.1 - Example of how to create a composite data set

x=Double1d.range(10)
#x is an array of doubles that is one dimensional (0.0, 1.0...9.0)
s=ArrayDataset(data=x,description="range of real values")
#s is an array dataset which has added description
t=TableDataset(description="This is a table")
#This sets up an empty table with a description
t["x"]=Column(x)
#the array 'x' is added to the table and given a
#column heading "x"
t["y"]=Column(data=x*4,description="x*4")
#each of the array elements of 'x' is multiplied by 4
#and becomes the data in the table column labeled "y".
#The table column also has a description added to it.
c=CompositeDataset()
#c is an empty composite dataset.
c.description="This is a composite dataset. It contains\
  three datasets!"
#we add a description to c
c.meta["author"]=StringParameter("Jorgo Bakker")
#we add the author's name as a string parameter
c.meta["version"]=StringParameter("2.0")
#we input a version number as a string parameter
c["mySimple"]=s
#we put the arraydataset s into the composite dataset c
#and give it a name mysimple so that we can refer to it
c["myTable"]=t
#we do the same for the table
c["myNest"]=CompositeDataset("Empty nested composite\
  dataset")
#this just shows you can add a composite dataset into another
#composite dataset (nesting)

print c      # view contents of the complex dataset.
tab = c["myTable"]      # gets our TableDataset back. Now called "tab".
print tab      # we see that it has two columns called "x" and "y"
print tab["x"]      # prints out what is in the "x" column.
print tab["x"].data      # to just print out the data values.
```



## Herschel IA Chapter 6

# 6 IA Numeric: Basic Functions for Herschel IA

### Chapter 6 Contents

- 6.1 [Introduction](#)
- 6.2 [Getting Started](#)
- 6.3 [Functions and Lambda Expressions](#)
- 6.4 [Filtering](#)
- 6.5 [The 'where' and 'get' methods](#)
- 6.6 [Advanced Tips](#)
- 6.7 [Vectors and Matrices](#)
- 6.8 [Function Library](#)
  - 6.8.1 [Basic Functions](#)
  - 6.8.2 [Discrete Fourier Transform](#)
  - 6.8.3 [Convolution](#)
  - 6.8.4 [Boxcar and Gaussian Filters](#)
  - 6.8.5 [Interpolation Functions](#)
  - 6.8.6 [Basic Fitter Routines](#)
- 6.9 [Example Programs](#)

---

### 6.1 Introduction

This document describes how to use the IA *numeric* library from the interactive Jython environment (`jconsole`). For further details of the functions provided, or use of the library from Java programs, please see the API documentation for [herschel.ia.numeric](#).

The purpose of the numeric library is to provide an easy-to-use set of numerical array classes (programs) and common numerical functions. The library also supports arrays of booleans and strings.

### 6.2 Getting Started

The IA numeric packages are loaded and available to the user on starting an IA/`jconsole` session. Basic setup and arithmetic manipulation of array datasets of various types are discussed in [Chapter 5](#).



### 6.3 Functions and Lambda Expressions

In IA, functions can be applied very simply as follows:

```
print Sqrt(16)                    # 4.0 (applied to a scalar)
y = Double1d([1,4,9,16])
print Sqrt(y)                     # [1.0,2.0,3.0,4.0]
```

As shown by this example, functions on scalars (such as Sqrt) are implicitly mapped over each element of an array. Functions may be combined with arithmetic operators to perform complex operations on each element of an array:

```
t = Double1d([1,2,3,4])
print SIN(1000 * t * (1 + .0003 * COS(3 * t)))
# [0.6260976237441638,0.5797470124743422,0.8629107307631398,
#-0.9811675382238753]
```

The **names of functions in the numeric library have ALL LETTERS** capitalized. This is to avoid ambiguity as Jython already defines certain functions, such as 'abs', which are not applicable to our numeric arrays.

There are various types of functions in the numeric library:

```
y = Double1d([1,2,3,4])

Sqrt(4)                # double->double
Sqrt(y)                # double->double (mapped)
REVERSE(y)            # Double1d->Double1d
MEAN(y)                # Double1d->double
```

It is possible to define **new functions** as *lambda expressions* in Jython and apply them to numeric arrays. For example:

```
y = Double1d([1,2,3,4])

f = lambda x: x*x + 1 #take the given array, call it 'x' and
#return the value x^2 +1 to an array called f.
print f(y)           #[2.0,5.0,10.0,17.0]. Each element of y was
#taken -> x then each element was squared
#plus 1 added.
```

However, in this case, it much easier and faster to do this with array operations.

```
print y * y + 1
```

Lambda expressions are not as fast as the standard Java functions provided by the numeric library, but this is often not a problem. Where performance is an issue, new functions can be defined in Java (see the [JavaDoc](#) of the `herschel.ia.numeric` library).

More complex functions (equivalent to subroutines) can be created using the **def** command, which is discussed in [Chapter 4.5](#).

## 6.4 Filtering

The numeric library provides operations, such as `'filter'`, which allows the selection of array elements based on a given criterion (e.g., element with values between 3 and 6). There is no `'map'` operation because mapping is implicit with the array style of processing.

# Using the filter method (returns a Double1d)

```
u = Double1d.range(10)
print u.filter(lambda x: x>3 and x<6)
```

Note: The Jython `filter` operation can be used but returns a Jython array:

```
print filter(lambda x: x>3 and x<6, u)
# __class__ returns org.python.core.PyList
print filter(lambda x: x%2==1, u)
```

Jython *list comprehensions* can be used but also return Jython arrays:

```
print [x for x in u if x>3]
print [x*x for x in u if x>3 and x<6]
print Double1d([x*x for x in u if x>3 and x<6])
#this last now provides us with a numerical array
```

## 6.5 The 'where' and 'get' methods

If you wish to select elements of an array based on a given criterion then this section indicates how you can filter data, find out where in a sequence data of a certain type resides (e.g., where the maximum value of an array occurs) and how to get the data that fits your selection.

For example, the `'where'` method returns the array indices of elements that satisfy a predicate often given as a lambda function, whereas `'filter'` returns the actual elements themselves. Using the modulo function (%) we can find where within an array odd values occur.

```
y = Double1d([2, 6, 3, 8, 1, 9])
print y.where(lambda x: x%2==1) # [2,4,5] indices of odd elements
```

Now return the actual elements, which can be done in three ways

```
print y[y.where(y%2==1)]           # [3.0,1.0,9.0]
print y.filter(lambda y: y%2==1)   # [3.0, 1.0, 9.0]
print y.get(y%2==1)                # [3.0,1.0,9.0]
```

Predicates support standard jython operators such as `not`, `and` and `or`:

```
y = Double1d([1,2,3,4])
print y.where(lambda x: x<3 and x>1)      # [1]
```

**Java/C-style logical operators '!', '&&', and '||' are not allowed.**

It can be useful to have the indices, rather than the values, when there are two or more arrays with a predicate applied to one of them. For example:

```
x = Double1d([5,6,7,8])
s = y.where(lambda i: i%2==1)
print x[s] + y[s]                          # [6.0,10.0]
```

The **'where' function** can also be used to set values:

```
s = y.where(lambda x: x%2==1)
y[s] = 0                                     # Set all matching elements to 0
print y                                     # [0.0,2.0,0.0,4.0]
y[s] = [9,8,7]                             # Set matching elements using an array of values
print y                                     # [9.0,2.0,8.0,4.0]
```

The **'get' method** enables you to grab individual elements or a subset of element values from an array. Along with getting individual elements there are three other forms. One enables you to select element values based on a `Bool1d` mask:

```
y = Double1d([5,7,8,9])
mask = Bool1d([0,0,1,0])
x = y.get(mask)                             # x == [8.0]
```

The second form enables you to select on a set of indices, contained in a `Selection` object:

```
indices = Selection(Int1d([2,3]))
x = y.get(indices)                          # x == [8.0,9.0]
```

The third form enables you to select elements from a range, specified by a `Range` object:

```
range = Range(2,4)
x = y.get(range)                            # x == [8.0,9.0]
```

It is possible to combine 'get' calls to perform the same operation as a compound IDL `WHERE` execution. Let's set up a few arrays first:

```
a = Double1d([1, 2, 3, 4, 5, 6])
b = Double1d([2, 3, 4, 5, 6, 7])
c = Double1d([3, 4, 5, 6, 7, 8])
```

The following operations on the three arrays are the equivalent of the IDL `WHERE` statement `'where(a ge 2 and b lt 6 and c gt 5)'`:

```
q = (a >= 2) & (b < 6) & (c > 5)
x = a.get(q), b.get(q), c.get(q)           # x == ([4.0], [5.0], [6.0])
```

## 6.6 *Advanced Tips*

The underlying array operations and functions are very fast, as they are implemented in Java. The overhead of invoking them from Jython is relatively small for large arrays. However, the advanced user may find the following tips useful to improve performance in cases where it becomes a problem.

The arithmetic operations, such as '+', have versions that allow in-place modification of an array without copying. For example:

```
y = Double1d.range(10000)
y = y + 1           # The array is copied
y += 1            # The array is modified in place
```

Copying an array is slow as it involves allocating memory (and subsequently garbage collecting it). For simple operations, such as addition, the copying can take longer than the actual addition.

Function application also involves copying the array. This can be avoided by using the Java API instead of the simple prefix function notation. For example:

```
x = Double1d.range(10000)
x = SIN(x) * COS(x)      # this operation involves three copies
x = x.apply(SIN).multiply(x.mApply(COS)) # only one copy
```

Note that the mutating `mApply` is not used for the application of `SIN`, otherwise `x` would be modified, with the result that `COS(x)` would actually evaluate `COS(SIN(x))`.

When writing array expressions, it is better to group scalar operations together to avoid unnecessary array operations. For example:

```
y = Double1d([1,2,3,4])
print y * 2 * 3           # 2 array multiplications
print y * (2 * 3)        # 1 array multiplication
print 2 * 3 * y           # 1 array multiplication
```

**It is better to avoid explicit loops in Jython** over the elements of an array. It is often possible to achieve the same effect using existing array operations and functions. For example:

```
sum = 0.0
for i in y:
    sum = sum + i * i      # Explicit iteration

sum = SUM(y * y)         # Array operations
```

## 6.7 *Vectors and Matrices*

The `Double1d` class provides a `dotProduct` method for scalar multiplication of vectors:

```
x = Double1d([1,2,3,4])
y = Double1d([1,3,5,7])
print x.dotProduct(y)           # 50.0
```

Similarly, the `Double2d` class provides special methods for matrix multiplication and transposition:

```
x = Double2d([[2,4,6],[1,3,5]])
y = x.transpose()
z = x.matrixMultiply(y)
```

Hence, it is important **not** to use the Jython '\*' operator for matrix multiplication. However, the '+' operator performs element-wise addition as required.

It is also possible to multiply a matrix by a vector as follows:

```
a = Double2d([[1,2,3],[7,5,4],[7,4,9]])
x = Double1d([4,1,7])
print a.matrixMultiply(x)       # [27.0,61.0,95.0]
```

Other matrix functions are provided by the class `MatrixFunctions`. At present, the only operation provided is 'solve' for the solution of matrix equations. For example:

*# Solve the matrix equation:  $A.X = Y$*

```
x1 = matrixSolve(y) (a)
print x1              # [
                      # [0.12903225806451615,0.38709677419354843],
                      # [0.2580645161290323,-0.22580645161290328],
                      # [0.4516129032258064,0.3548387096774193]
                      # ]
```

## 6.8 *Function Library*

The numeric package includes a library of basic numeric processing functions, which will continue to grow as development of the library progresses.

The functions that are currently available are outlined below. For further details, reference should be made to the **JavaDoc documentation** and **demo programs**.

## 6.8.1 Basic Functions

Basic double->double functions applicable to double, Double1d, Double2d and Double3d arrays:

```
ABS, ACOS, ASIN, ATAN, CEIL, COS, EXP, FLOOR, LOG, LOG10, ROUND,  
SIN, SQRT, SQUARE, TAN
```

These are applied in the form

```
IA>> b = SIN(a)
```

b will be an array of the same dimension as a or a single value if a is single valued.

Array functions on Double<n>d returning a double:

```
MIN, MAX, MEAN, MEDIAN, RMS, SIGMA, SUM, INDEX
```

```
IA>> b = MIN(a) # 'b' has the minimum value of the array 'a'.
```

Double1d->Double1d functions:

```
REVERSE
```

## 6.8.2 Discrete Fourier Transform

A Discrete Fourier Transform is provided for Complex1d arrays. This uses a radix-2 FFT algorithm for array lengths that are powers of 2 and a Chirp-Z transform for other lengths. Future releases might support multi-dimensional arrays, if required, and optimized transforms of real data.

Window functions are provided for reducing 'leakage' effects using **the Hamming or Hanning** window.

[Example 6.1](#) shows the generation of a frequency modulated signal, followed by a FFT both with and without windowing:

```
#Example 6.1: FFT of a modulated signal , with and without HAMMING  
#smoothing  
#First add in some extra packages.  
from herschel.ia.numeric.function.Complex1dFunctions import *  
from herschel.ia.numeric.function.FFT import *  
from herschel.ia.numeric.function.WindowFunctions import *  
  
ts = 1E-6           # Sampling period (sec)  
fc = 200000        # Carrier frequency (Hz)  
fm = 2000          # Modulation frequency (Hz)
```

```
beta = 0.0003      # Modulation index (Hz)
n = 5000          # Number of samples

pi = java.lang.Math.PI    # define pi

t = Double1d(range(n)) * ts
# [0.0E-6, 1.0E-6, 2.0E-6...] 5000 element array holding time values

signal = SIN(2 * pi * fc * t * (1 + beta * COS(2 * pi * fm * t)))
#create the modulated signal with modulation frequency fm and carrier
frequency fc.
#t is the array we created above for the time elements.

spectrum = CABS(FFT(Complex1d(signal)))
#spectrum holds the complex absolute value (CABS) of the FFT of the
signal.
#We need to handle these arrays as Complex1d rather than Double1d.

freq = Double1d(range(n)) / (n * ts)
#The frequency values for the spectrum.

# Repeat with apodizing
spectrum2 = CABS(FFT(Complex1d(HAMMING(signal))))
```

### 6.8.3 Convolution

Convolution is currently supported for `Double1d` arrays. A direct convolution algorithm is used, although a future release might implement Fourier convolution to improve the speed for large arrays and large kernels. An example of its use is given in [Example 6.2](#).

```
#Example 6.2: Example of the use of the convolution algorithm

from herschel.ia.numeric.function.Complex1dFunctions import *
from herschel.ia.numeric.function.FFT import *
from herschel.ia.numeric.function.WindowFunctions import *
from herschel.ia.numeric.toolbox.filter.Convolution import *
x = Double1d.range(100)
# Create array [0.0, 1.0, 2.0...99.0]
kernel = Double1d([1,1,1])
#provide kernel for the convolution
f = Convolution(kernel)
#create the convolution
y = f(x)
#apply it to the array x - result is in array y
```

*This illustrates a general approach with the numeric library* i.e. general function objects may be instantiated using parameters to create a customized function which can then be applied to one or more sets of data.

The constructor of the `Convolution` class allows optional *keyword* arguments to be specified, to further customize the function:

- The 'center' parameter allow selection of a causal asymmetric filter for time domain filtering or a symmetric filter for spatial domain filtering.
- The 'edge' parameter controls the handling of edge effects, as well as allowing a choice between periodic (circular) and aperiodic convolution.

The following examples show construction of filters using these options:  
(NOTE: make sure you have input the first 4 lines of example 6.2 before trying these out)

```
# Use zeroes for data beyond edges, causal
f = Convolution(kernel, center=0, edge=ZEROES)

# Circular convolution, causal
f = Convolution(kernel, center=0, edge=CIRCULAR)

# Repeat edge values, causal
f = Convolution(kernel, center=0, edge=REPEAT)

# Use zeroes for data beyond edges with centred kernel
f = Convolution(kernel, center=1, edge=ZEROES)

# Circular convolution with centred kernel
f = Convolution(kernel, center=1, edge=CIRCULAR)

# Repeat edge values with centred kernel
f = Convolution(kernel, center=1, edge=REPEAT)
```

#### 6.8.4 Boxcar and Gaussian Filters

Finite Impulse Response (FIR) filters and symmetric spatial domain filters can be defined by instantiating the `Convolution` class with appropriate parameters. *In addition, special filter functions are provided for Gaussian filters and box-car filters:*

```
from herschel.ia.numeric.toolbox.filter.Convolution import *
f = GaussianFilter(5, center=1, edge=ZEROES)
f = BoxCarFilter(5, center=0, edge=ZEROES)
```

These filters are subclasses of `Convolution` and hence inherit the use of similar keyword arguments.

#### 6.8.5 Interpolation Functions

Interpolation functions are provided for a variety of common interpolation algorithms. [Example 6.3](#) illustrates the use of the currently available interpolation functions. The plotting package available for displaying the different interpolation forms (PlotXY) is discussed more fully in [Chapter 7](#).



```
#Example 6.3: Interpolation functions in IA

from herschel.ia.numeric.function import *
# create the array x [0.0, 1.0, 2.0, ...,9.0]
x = Double1d.range(10)
# create array y which contains the sine of each element in x
y = SIN(x)
# u contains values at which to interpolate
u = Double1d.range(80) / 10 + 1

# Linear interpolation
interp = LinearInterpolator(x,y)
# this sets up the interpolation, linear x-y fit
# Interpolate at specified values
print interp(u)
a = interp(u)
# prints out the values interpolated at each position noted in array u
# NearestNeighbour and CubicSpline interpolation may be performed
# in the same way:

# Cubic-spline interpolation
interp_c = CubicSplineInterpolator(x,y)
b = interp_c(u)
# Nearest-neighbour interpolation
interp_n = NearestNeighborInterpolator(x,y)
c = interp_n(u)

p = PlotXY(x,y, "original values")
p.addLayer(u,a, "linear", 8, java.awt.Color.red)
p.addLayer(u,b, "cubic spline", 8, java.awt.Color.blue)
p.addLayer(u,c, "nearest neighbour", 8, java.awt.Color.green)
```

The result of example 6.3 are illustrated in Figure 6-1.

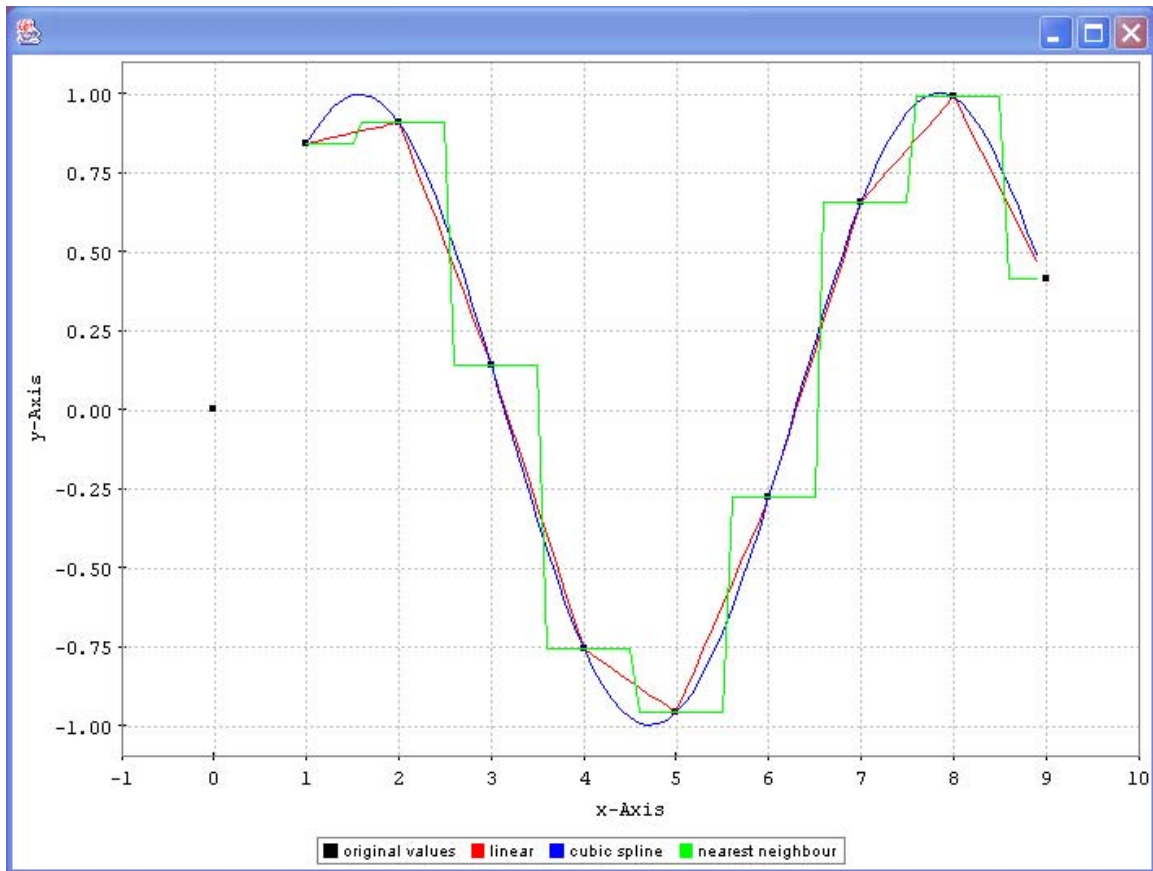


Figure 6-1: Illustration of various forms of interpolation.

## 6.8.6 Basic Fitter Routines

A more complete package of advanced data-fitting routines is available and will be more fully discussed in the next version of the User Manual. Here, we discuss some basic fitting routines available within IA.

A **least squares polynomial fitter** is provided by the numeric library. [Example 6.4](#) shows how a polynomial can be fitted to data.

#Example 6.4: Use of fitter routines in Herschel IA.

```
from herschel.ia.numeric.function import *
# Create some data
x = Double1d([3,4,6,7,8,10,11,13])
y = Double1d([2,4,5,6,5,6,7,9])

# Create a polynomial fitter of degree 3
fitter = PolynomialFitter(3)

# Fit the data
poly = fitter.fit(x,y)
```

```
#and print the fit results
print poly
#..and also get the Chi-squared. The fitter has already been applied
#and we can use the getChiSquared() method to determine the fit
print "Chi-Squared = ", fitter.getChiSquared()
#The fitted polynomial can then be applied as a function to interpolate
#between the fitted points:
# Interpolate at 'n' uniformly spaced x values
n = 100
u = MIN(x) + DoubleIeld.range(n + 1) * ((MAX(x) - MIN(x)) / n)
polyu = poly(u)

# Now we can plot the data (x vs y) and the polynomial fit (u vx polyu)
plot = PlotXY()
#set up the plot space
plot.addLayer( x, y, "data", 8, java.awt.Color.blue)
#plot x against y in blue. Linetype = 8 --> a line.
plot.addLayer( u, polyu, "fit", 8, java.awt.Color.green)
#overlay a second plot showing the polynomial fit in green.
```

The final plotted display should look like Figure 6-2

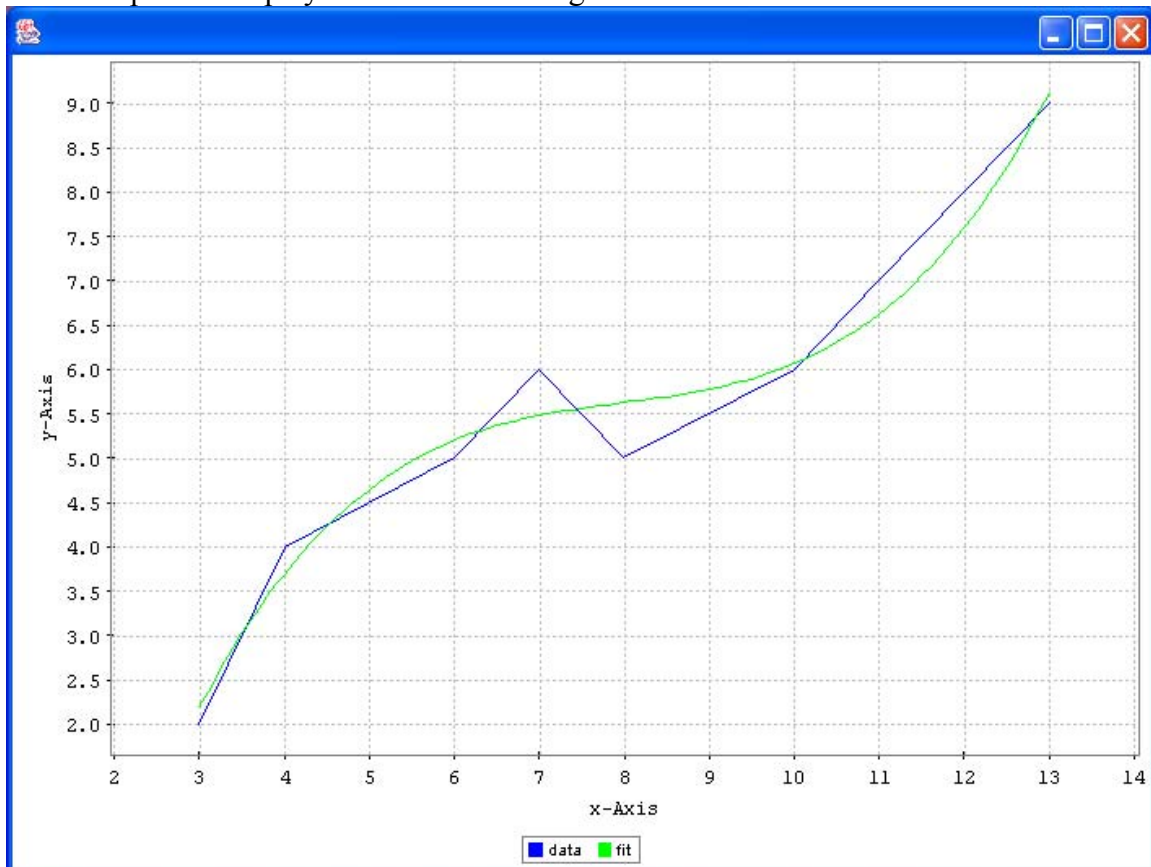


Figure 6-2: Example polynomial fit.

## 6.9 Example Programs

The HCSS distribution includes a number of Jython example programs that demonstrate not only basic arrays functions but also use of filters, fitters, Fourier transforms, etc. These are:

[numeric\\_whatisNew.py](#) Example of the newest components of the numeric package.

[numeric\\_demo.py](#) Example of how to use the 1D functionality.

[numeric\\_2D\\_demo.py](#) Example of how to use the 2D functionality

[convolution\\_demo.py](#) Example of how to use the convolution functionality

[polyfitter\\_demo.py](#) Example of how to perform polynomial fitting



## Herschel IA Chapter 7

# 7 IA Plot: Basic Plotting of Data

### Chapter 7 Contents

- 7.1 [Introduction](#)
- 7.2 [What do I need to make a simple XY plot?](#)
  - 7.2.1 [Introducing PlotXY](#)
    - 7.2.1.1 [Using PlotXY to Plot One Numeric1d Array Against Another](#)
    - 7.2.1.2 [Using PlotXY to Plot Columns in a TableDataset](#)
- 7.3 [How to setup your PlotXY properties](#)
  - 7.3.1 [How to modify properties](#)
  - 7.3.2 [Plot properties](#)
  - 7.3.3 [Layer properties](#)
  - 7.3.4 [Axes properties.](#)
  - 7.3.5 [How to use properties.](#)
- 7.4 [How to use PlotXY in IA scripts](#)
  - 7.4.1 [What about these Layers?](#)
  - 7.4.2 [What can I do with Axis?](#)
  - 7.4.3 [How can I annotate and decorate my plot?](#)
  - 7.4.4 [How can I make my plots more colourful?](#)
- 7.5 [Creating File Output and Printing a Plot Without Displaying](#)
- 7.6 [Handling Units in Plots](#)
- 7.7 [What about a complete example?](#)

---

## 7.1 Introduction

This chapter describes how to do basic 2D plots in IA. It is not intended to elaborate on the complete set of functionalities, for this please refer to the related [API documentation for the `herchel.ia.plot` package](#). The basic concept is presented in order to support your first steps for simple visualization of two-dimensional data.

Three classes are described in this chapter: the `PlotXY` class, which is the representation of a two-dimensional plot, and its related classes `Axis` and `Layer` which represent the different building blocks from which the plot is constructed.

Depending on how you work with plots, either writing scripts or designing your plots interactively, we recommend different approaches. For writing scripts you have to use the

command line interface. This way the plot is completely defined by written commands. If you design your plots interactively it will be easier to use the graphical interface to manipulate plot properties which allows for button and pulldown menu selection of plot properties such as fonts, labels, line types and colours.

The plot package (`herschel.ia.plot`) is automatically loaded on starting an IA session.

Lastly, PlotXY requires a property file `PlotXY.props` be placed in your `.hcss` directory. At a system prompt in (only) the first session that you start with PlotXY in IA you should first input

```
touch ${HOME}/.hcss/PlotXY.props
```

before starting your plotting, or simply add a dummy text file using your favourite editor.

## 7.2 *What do I need to make a simple XY plot?*

The 2D plotting package currently works on two types of data:

[Numeric1d data](#) which is a one-dimensional array of numbers of any type. Two numeric arrays are input, one as x-data and the other as y-data.

[TableDatasets](#) which contain `Double1d` data placed in columns. The default behaviour for plotting is that the first column is taken as the x-axis and the following columns as data that is to be overlaid on the same plot.

### 7.2.1 Introducing PlotXY

The class used for 2D plotting is called `PlotXY`. This produces a plot whose properties can be changed via command line input or through a properties GUI. Multiple plots can be added in “layers” to an initial base plot and the default scales for a given plot will automatically adjust to allow all points in all layers of a plot to be visible, although the x and y ranges for a plot can also be set by the user.

#### 7.2.1.1 Using PlotXY to Plot One Numeric1d Array Against Another

[Example 7.1](#) illustrates how to plot one-dimensional arrays as well as a `TableDataset` that contains two columns “X” and “Y”.

#Example 7.1 for PlotXY.

```
n = Double1d.range(20)/10
#n is set up to be an array with the range of numbers = 0..19 divided
by 10
```

```
#Placing the Doubleld element in front turns the integers created by
the
#range command into doubles. So we should have an array of 20 numbers
#going from 0 to 1.9

e = EXP(n)

#e is an array which contains the exponent of all the x array
components

p = PlotXY(x = n, y = e, title="plot example", layername = "Layer1")

#this plots the exponent and gives a title to the plot. It
#identifies the layer ("Layer1") and "p" identifies the plot window.
#We can refer to the layer at later times if we, e.g.,
#want to change plot colors.

layer1 = p.getLayer("Layer1")

#here we get the layer ("Layer1") inside the plot (called "p")

layer1.setSymbol()

#here we set the default symbol for the points on the plot layer

layer1.setSymbolSize(3)

#...and change their size.

p.close()

# This removes the plot window.
```

The result of running this example is shown on the left side of Figure 7-1.

### 7.2.1.2 Using PlotXY to Plot Columns in a TableDataset

Following on from [Example 7.1](#), we can see how we might do a similar plot using columns of data from a table (in TableDataset format). [Example 7.2](#) shows how to construct a TableDataset and then plot values from in using PlotXY.

#Example 7.2 for PlotXY.

```
#Example of plotting columns from a TableDataset.
#Now we attempt to do a similar plot to that produced in example 7.1
#but using a Tabledataset.

t = TableDataset()

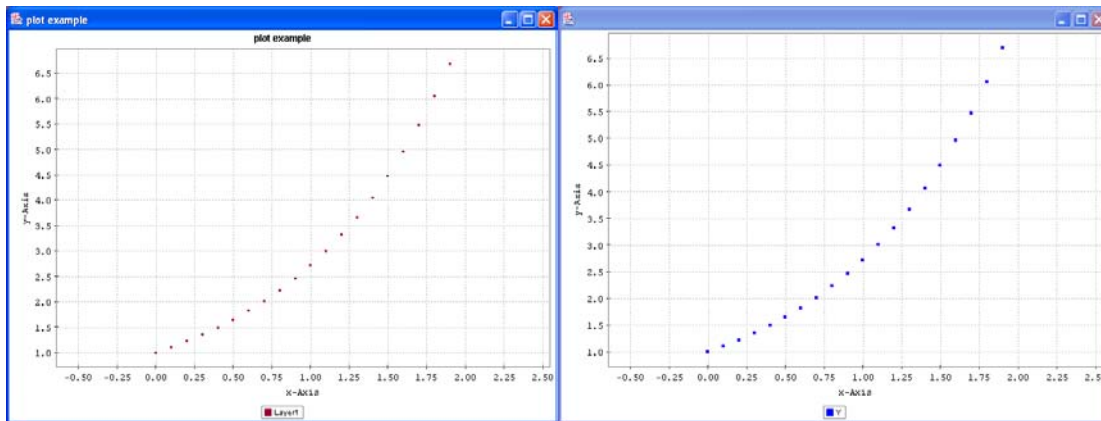
#create a table - empty to start

t["X"] = Column(n)

#now we have put a column in the table "t" and give the
#column a name "X"
```

```
#The values placed in this first column are in the array "n" -  
#see first line.  
  
t["Y"] = Column(e)  
  
#similarly for the next column  
  
pp = PlotXY(dataset = t, linetype = PlotXYCompositeRenderer.RECTANGLE)  
pp.setSymbolColor(Color.blue)  
  
#now we plot the dataset, and tell it to use blue for the  
#lines created. In this case the first column of the table is  
#plotted on the x-axis by default and  
#the second column is plotted along the y-axis.  
#NOTE: the layer name is not needed. PlotXY picks up the column  
#name for reference instead.
```

The output from this example is shown next to that of example 7.1 in Figure 7-1.



**Figure 7-1: The two plots produced from examples 7.1 and 7.2. Note that the first graph has an x-axis with only the index plotted.**

A special way of constructing a plot is possible. It uses the notation *variablename = variablevalue* for any subset of possible variables when starting PlotXY: The following elements can be defined when first constructing a plot.

```
PlotXY(TableDataset dataset, String title, String layername, int width,  
int height, Color lineColor, int linetype, int plotting_depth, boolean  
useAsComponent, boolean plotIsVisible)
```

So we can construct a plot with:

```
p=PlotXY(dataset = t, plotIsVisible= 0)
```

In this situation, although the plot (labeled p) is produced it is not made visible. This could be useful if you wish to add a number of later layers/components to the plot before making it visible.

Or



```
p=PlotXY(title = "I am empty and small", width = 100, height = 50)
```

which produces a small window area of 100 by 50 pixels which contains a title and in which a plot can later be placed. All the variables that are not specified in this second startup method can be specified by *plot properties*. These are discussed in the next section.

### 7.3 How to setup your PlotXY properties

Plot properties allow the definition of items such as *colours*, *linetypes* etc. with your personal preferences. To setup your personal properties try the following:

- Construct a plot object *p* in *jconsole*,  
e.g., `p = PlotXY(title= "Empty plot", width=100, height=50)`
- Type the command "`p.props()`"
- Define your properties in the window that comes up and save them as default. The description can be found in the next section.

or

- Find the property defaults file `PlotXY.defaults` in the directory `var.hcss.dir/lib/herschel/ia/plot.`
- Rename this file to `PlotXY.props`
- Copy `PlotXY.props` into the folder `.hcss` in your home directory (this is also where your configuration file for HCSS resides – `myconfig`).

Then

Add the path `${HOME}/.hcss/PlotXY.props` to your `HCSS_PROPS` system variable source your login file

If you have done this PlotXY uses properties that can be modified according to your needs.

#### 7.3.1 How to modify properties

Properties can be manipulated with a graphical interface.

Do the following:

construct a plot object with any constructor, for example

```
p=PlotXY(dataset = ds)
```

type the command "`p.props()`"

Now the graphical interface for manipulation of the plot properties appears (see Figure 7-2). It consists of the register cards, *Plot properties*, *Layer* and *Axes* properties.

The buttons at the bottom have the following functions:

**cancel:**

makes the property frame invisible. The same happens if you use the window close button in the right corner of the frame.

**refresh props:**

reads in the properties of the visible register card (plot, layer or axis). This button is useful if you have the plot property gui visible and change properties from the commandline. Refresh updates the gui afterwards.

**save as default:**

saves the properties of the visible register card as default and thus updates the PlotXY.props file in the ~/.hcss directory. Note that if you set a property for a layer or an axis as default, the property set will be used for all layers and axis and not only for the one you have chosen in the moment of pressing the button.

**apply:**

applies the properties of the visible register card to the plot object.

### 7.3.2 Plot properties

Most of the properties are self explanatory. Simply change them, hit apply and see the result. Clicking on the plot properties tab offers a section called "preferred layer layout (colour & symbol)". There you can specify the layout of any number of layers. This setup is used if you call an `addLayer` method that does not specify `color` and `linetype`. This functionality allows you to construct plots with multiple layers with predefined layout.

*Use it as follows:*

to add a new layer layout, right click in the white space below the defined layouts.  
to remove a definition, right click into the white space at the right or left of an existing definition.

- to change the colour of an existing layout click into the coloured square.
- to change the `linetype` choose it from the pulldown list

### 7.3.3 Layer properties

The layer properties pane is used to define default layer properties or to manipulate the properties of already constructed layers (see Figure 7-3).

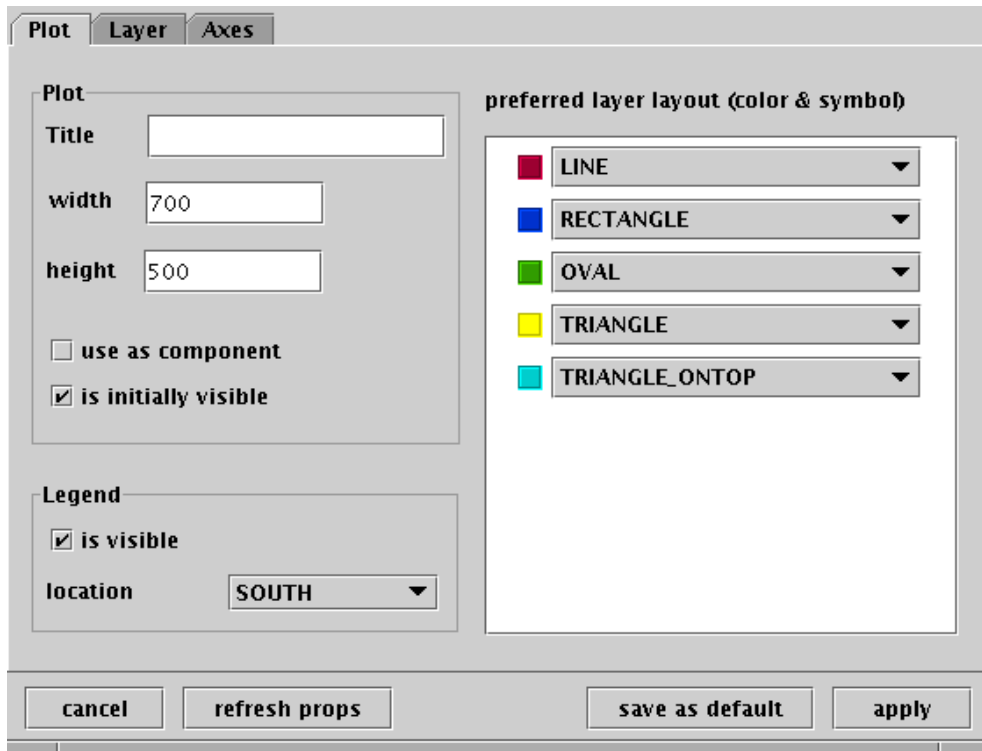


Figure 7-2: The PlotXY properties graphical interface.

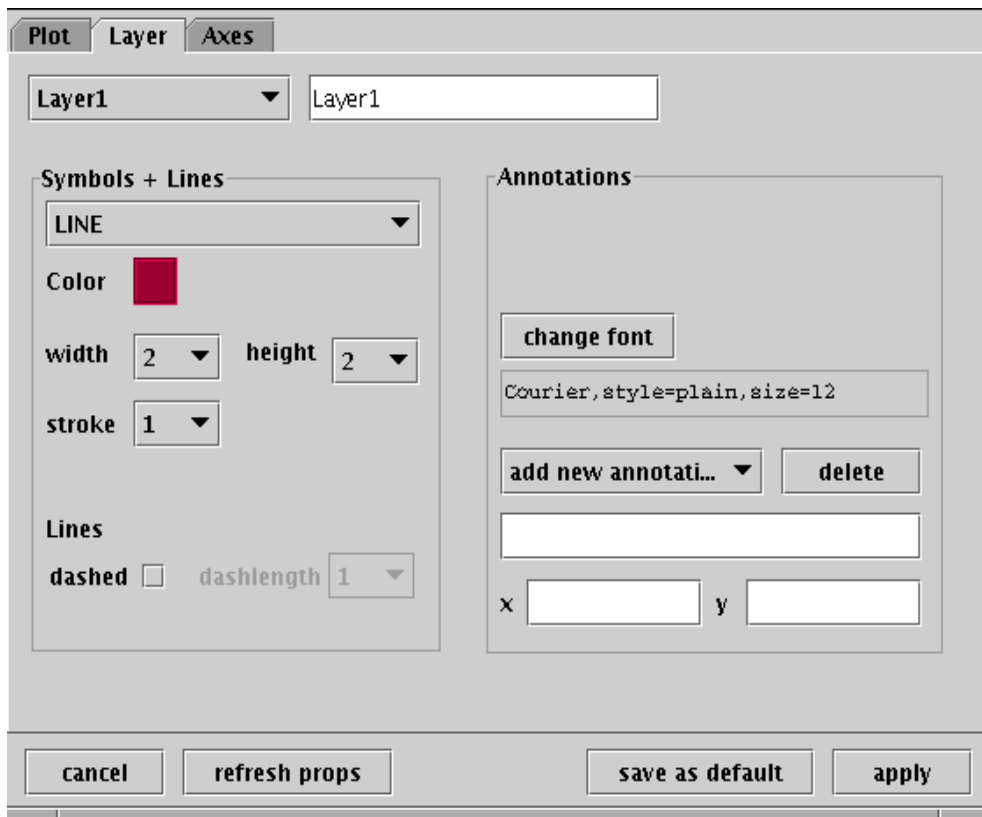


Figure 7-3: Graphical interface for PlotXY layers

### 7.3.4 Axes properties.

The axes properties pane (see Figure 7-4) is used in the same way as the layer properties. Axis properties are also straight forward with one exception and that is `tickunits`:

- Any number > 0 sets the tickunits to this number
- 0 removes tickunits from the axis
- an empty field or any number < 0 resets tickunits to autoselection.

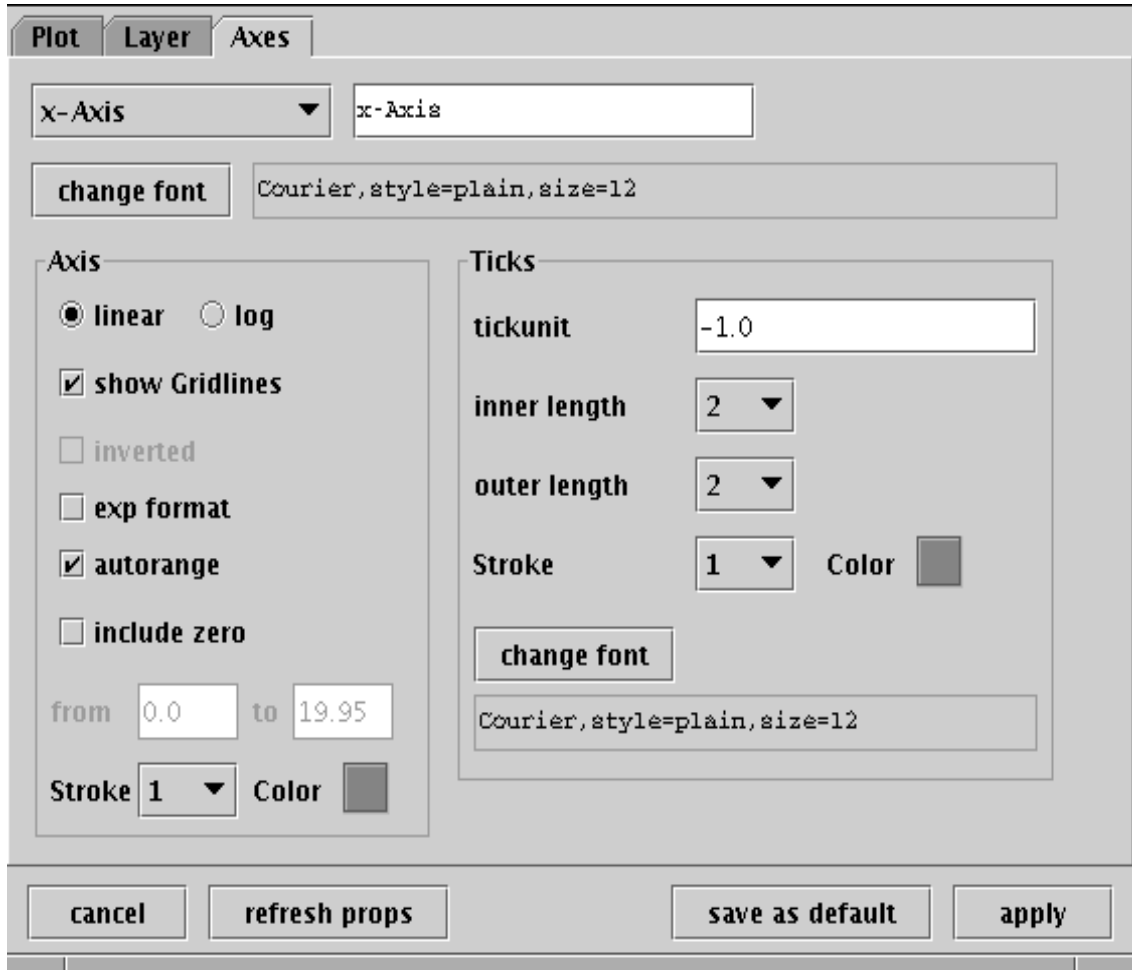


Figure 7-4: The Axes properties pane for PlotXY properties.

### 7.3.5 How to use properties.

The result of property setup procedure (with a defined set of properties) is the following [Example 7.3](#):

### #Example 7.3: command line control of properties

```
#This is a continuation from Examples 7.1 and 7.2.

pp.props()
#allows graphical interface property setup
pp.getLayer("Y").remove()
#removes layer containing plot of "Y" from example 7.2
l2 = pp.addLayer(n, n*n, "layer2")
#overlays on the graph a plot of x versus x-squared and calls it
#"layer2". l2 is the name of this overlay plot
l2.setSymbolSize(5)
#sets the symbol size for overlay plot l2
l3 = pp.addLayer(n, 2*n*n, "layer3")
#adds another layer to the plot "p"
l3.setSymbolSize(7)
#...and changes symbol size on this plot too!
```

The result of running [Example 7.3](#) is shown in Figure 7-5.

Note that if `addLayer` is used that defines neither colour nor linetype, the current set of default properties are used for subsequent layers. If you have a look at the predefined layer colours and linetypes you see that come as default in IA, the first definition is dark red colour and lines, the second layer is defined to have blue colour and rectangles.

If colour and linetype are specified in the constructor, they are used as specified.

```
pp.addLayer(n, 8*n*n, "layer4", PlotXYCompositeRenderer.LINE, \
Color(250,100,0))
```

NOTE: the “\” symbol provides continuation of the command onto the second line. The result of the above command line is shown in Figure 7-6. In this case we have also illustrated how you can **create your own colour through a mixture of red, green and blue** hues (values up to 256). In this case, the result is an orange colour for our third plot layer. According to the default layer layout the symbols would otherwise have been green ovals.

## 7.4 How to use PlotXY in IA scripts

In jython scripts you have to access all the properties from the commandline (either bottom left of `jconsole` for interactive work or in the upper pane of `jconsole` when doing script development).

There is one general rule to do so.

- a. get the object (`layer = p.getLayer(layername)` OR `axis = p.getAxis("axisname")`)
- b. use the methods provided by the object (`layer.setSymbolColor(color)`)

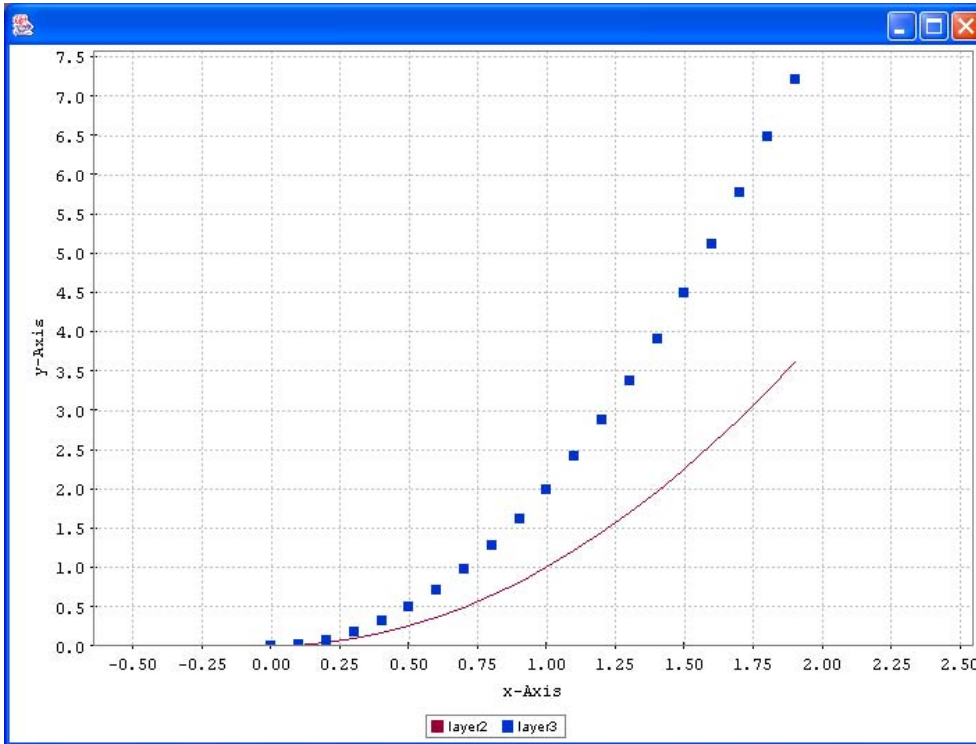


Figure 7-5: Plot obtained using example 7.3

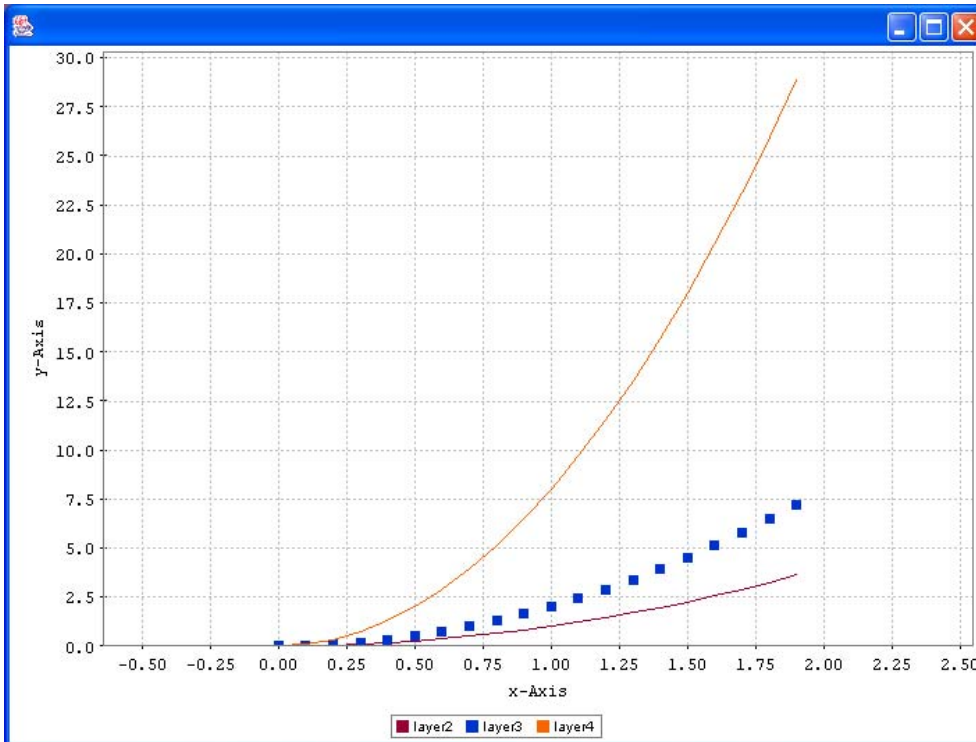


Figure 7-6: Adding in 4<sup>th</sup> layer gives the orange curve (see text).

### 7.4.1 What about these Layers?

Any plot is built up from layers. Even a simple 2D plot as we've created above has one layer that contains the data from the one-dimensional array or the second column from the TableDataset. If you need to plot multiple sets of data you add one layer for each additional set of data. This can be done step by step on the command line or automatically by the `PlotXY` command that creates a new layer for each column in a TableDataset.

As stated before the manipulation that you need to do on layers should be done through the layer object. One such command is the `setSymbolSize()` that we have used above.

Let's create a simple plot again with two layers and do some basic manipulations on the individual layers. [Example 7.4](#), plots two curves, one is the analytical function `exp` the other curve has added noise.

In the first three lines we generate some noise on top of the exponential function. [NOTE: Please do not take the above as an example of the proper way to add noise to a function, the 'noise' here is just to illustrate the layer concept.] The layer is added to the plot with the `addLayer()` method as done on line 7.

Line 6, 8 and 9 just illustrate how to set a plot to a scattered plot. Line 11 switches back to a line plot.

Figure 7-7 shows the results obtained from running [Example 7.4](#).

```
# Example 7.4: Working with layers from the command line.

r=java.util.Random()
#bring in a Java utility to produce random numbers between 0 and 1.
rn=[r.nextDouble() - 0.5 for i in range(20)]
#gives a set of 20 random double (real) numbers between -0.5 and 0.5
#the array e was defined in a previous example, but lets recreate it...
n =DoubleItd. range(20)/10
e = EXP(n)
#"e" is an array of 20 numbers which are e^0.5, e^1.0, e^1.5 etc.
en=e+rn
#adds the random numbers to the array "e" -> noise on the data
p=PlotXY(x = n, y = e, layername = "e")
#plots array "e" and gives the layer a name.
layer_e = p.getLayer("e")
#gets the layer so we can do something with it
layer_e.setSymbol()
#provides points with the default symbol.
p.addLayer(n, en, "en")
#adds a layer called "en" to the plot
layer_en = p.getLayer("en")
#now get the layer
layer_en.setSymbol()
#get default symbols for the points plotted again
```

```
layer_en.setSymbolColor(java.awt.Color.red)
#now sets the symbol's colour to red
layer_en.setSymbolSize(3)
#change the symbol size for "en"
layer_en.setLine()
#...and now draws a line between the values and changes
#it to a line plot
```

Some of the more useful methods that work on layers are listed in Table 7-1: Listing of methods of layer manipulation..

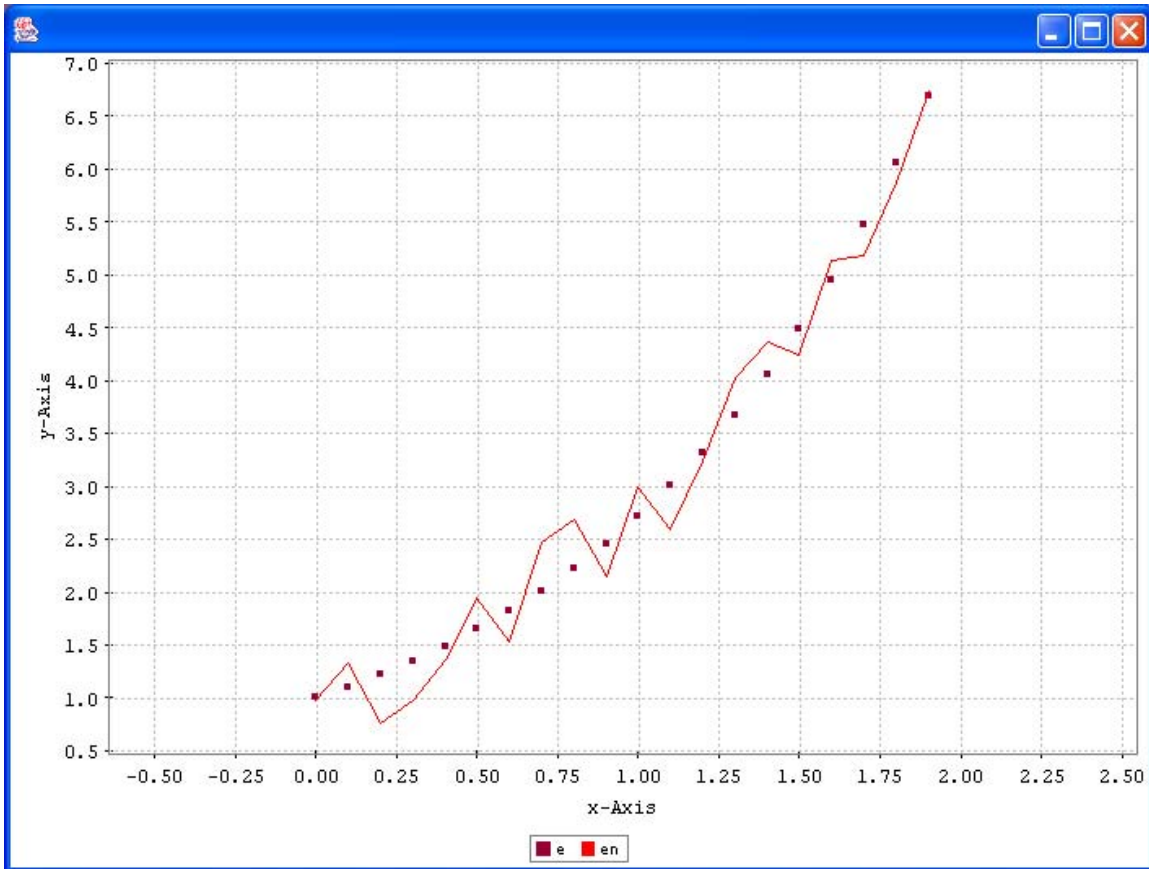
**Table 7-1: Listing of methods of layer manipulation.**

<code>l = p.getLayer(<i>layer</i>)</code>	get the Layer object to do direct manipulations on the specified layer
<code>l.update([<i>x</i>,<i>y</i>])</code>	update the specified layer with the new datapoints
<code>l.setLegend(<i>text</i>)</code>	changes the legend (and thus the layername) of the layer
<code>l.addAnnotation(<i>text</i>)</code>	write a text comment into the layer at the point chosen by the left mouse click
<code>l.addAnnotation(<i>text</i>, <i>x</i>, <i>y</i>)</code>	write a text comment into the layer at the point defined by x and y

**Methods to change the appearance of the datapoints:**

<code>l.setLine()</code>	change the plot to a line plot for the specified layer
<code>l.setSymbol()</code>	change the plot to a scatter plot for the specified layer
<code>l.setSymbolColor(<i>color</i>)</code>	set the colour of the symbols and lines for the specified layer
<code>l.setSymbolSize(<i>size</i>)</code>	set the size of the symbols for the specified layer (only for scatter plots)





**Figure 7-7:** Plot showing the manipulation of layers from the command line. The above plot comes from the script given in example 7.4.

The `Layer` provides a much larger number of methods to specify the appearance of datapoints in layers. Next to simple line and scatter plots, lines and symbols can be combined and symbols can be circles, rectangles, triangles, squares etc. which can be filled or not with a specified colour. Lines can be solid or dashed with their own colour. Find the possible predefined symbols in `PlotXYCompositeRenderer` and access them for example by `linetype = PlotXYCompositeRenderer.RECTANGLE`.

We are not going into detail for all these methods but you should try them out with the [API documentation for `Layer`](#) lying next to you.

#### 7.4.2 What can I do with `Axis`?

As with `Layers` most manipulations of both `x` and `y` axes can be done through the `Axis` object.

### 7.4.2.1 Log Axes, Labels and Gridlines

Lets continue with [example 7.4](#) and make some changes to the axes illustrating how we can adjust labels, gridlines and change axes to a logarithmic scale.

#### [Example 7.5](#)

#Example 7.5. This is an extension to the script in [example 7.4](#)

```
axis_y = p.getAxis("y-Axis")
#this grabs the y-axis. NOTE: y-Axis is the default label
#but this should be whatever label has been applied
#(e.g. using "setLabel")
axis_y.setExp(1)
#this sets the axis NOTATION as exponential, TRUE = 1
axis_y.setLog()
#this sets the axis with a log scaling
axis_y.setLabel("log(exp(x/10))")
#now we change the axis label
axis_x = p.getAxis("x-Axis")
#get the x axis
axis_x.setLabel("index")
#..and change its label
axis_x.showGridLines(0)
#this removes x axis gridlines, FALSE = 0
axis_y.setRange(0.5, 10)
#...and finally we adjust the range of y values that we
#want the plot to have.
```

Figure 7-8 shows the results from running [example 7.5](#).

### 7.4.2.2 Multiple Axis Labels

More than one label is possible per axis. A second (or subsequent) axis label for x or y is placed on the opposite side to the first axis label. This is illustrated in [example 7.6](#) which continues on from [example 7.5](#).

```
#Example 7.6. Extending example 7.5 to show how multiple axes may be
placed on the same plot.
layer=p.getLayer("en")
#get the layer we want to change
layer.addyAxis()
#add a new y axis
layer.setxRange(-0.5,2.5)
#restrict the range of the plot to x values between -0.5 and 2.5
layer.setyLog()
#make the new axis a log plot
layer.setyLabel("new Y axis")
#add a label to this new axis
layer.setyRange(0,100)
#restrict the range of y in the plot
layer.update(en/2)
```

```
#update the en layer so that it is half the value it was  
#before and replot  
layer.setyAutoRange()  
#now put the plot in a situation where the new y axis value range  
#is automatically calculated.  
p.setyRange(0,10)  
#set the y range for the main plot to be 0 to 10.
```

The result of running this example is shown in Figure 7-9.

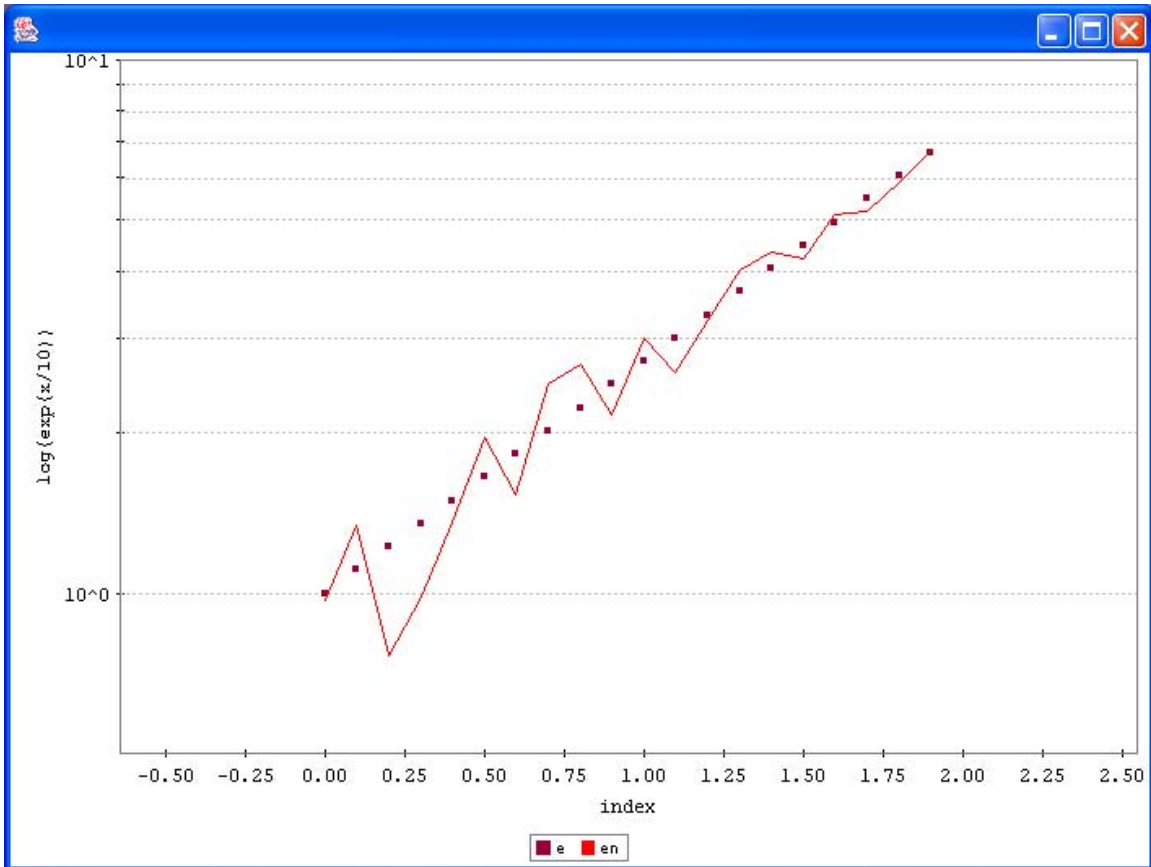


Figure 7-8 The results from running example 7.5.

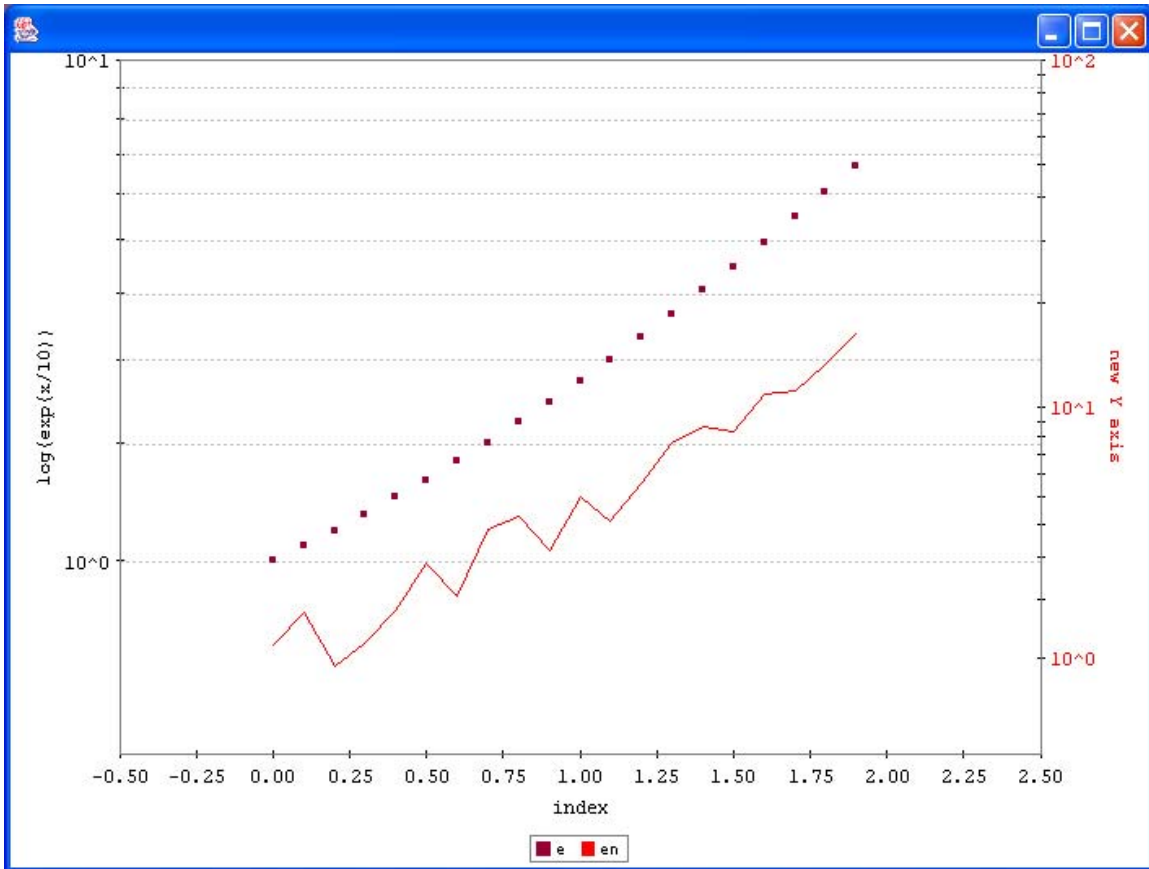


Figure 7-9: Example of a second Y axis label relevant to the red line plot.

Some of the more useful methods that work on axes are listed in **Table 7-2**. For a complete reference of the methods that can be used to manipulate and tune the appearance of the axes please consult the [API documentation of Axis](#).

**Table 7-2: Useful ways of manipulating axes from the command line**

<code>a = p.GetAxis(<i>axis</i>)</code>	get the <code>Axis</code> object to do direct manipulations on the specified axis
<code>a.setLinear()</code>	re-establish a linear scale for the specified axis
<code>a.setAutoRange()</code>	adjusts the range of the specified axis so that all datapoints will be shown
<code>a.setRange(<i>lower</i>, <i>upper</i>)</code>	set the range of the specified axis to values between <i>lower</i> and <i>upper</i>
<code>a.showGridLines(<i>flag</i>)</code>	show grid lines for the specified axis if <i>flag</i> is <code>true</code> , hide the grid lines if <i>flag</i> is <code>false</code>
<code>a.setLabelFont(<i>font</i> <i>size</i>)</code>	set or resize the font used for labeling the specified axis
<code>a.setTickLabelFont(<i>axis</i>, <i>font</i> <i>size</i>)</code>	set or resize the font used for labeling the tickmarks of the specified axis

### 7.4.3 How can I annotate and decorate my plot?

There are quite a number of methods that we can use to make our plot more appealing and informative. A number of these methods were already mentioned in the sections on layers and axes, but we are going to put them into practice here. We continue with [example 7.5](#) and add proper names for layers, annotate some datapoints and put a title on top of the figure (see [Example 7.6](#)). The example below, [example 7.7](#), also shows how to extract the `Layer` objects from the plot in order to manipulate them directly.

```
#Example 7.7...again following on from examples 7.5 and 7.6

layer=p.getLayer("en")
#get the layer we want to change
layer.setLegend("exp+noise")
#change the legend for this layer to say what we want
layer.addAnnotation("noise on top of exp()", 1, 2)
#place some annotation at position 1, 2
layer=p.getLayer("e")
#get a layer with another label "e"
layer.setLegend("exp")
#...and change its legend
layer.setSymbol(PlotXYCompositeRenderer.LINE_TRIANGLE, 5, 5, 0)
#change its symbol
p.moveLegend(PlotXY.EAST)
#move the plot legend to the right side
p.setTitle("Example of a layered plot")
#give the plot a title

p.saveAsPNG("plot-05.png")
#save it as a PNG file for importing as a picture into documents etc.
#alternatively....
p.saveAsJPG("plot-05.jpg")
#to save as a JPEG file.
```

Note that we changed the legend for both layers in line 2 and line 5. Changing the legend also changes the layer name which means that we need to use the new layer name in order to access or manipulate layers from the plot. The final plot is shown in Figure 7-10.

For the "exp+noise" layer we put an annotation at a specific point (user coordinates) in the plot. *There is an equivalent method that reads the mouse position after a left mouse click and places the text at that position.* Please check the detailed package documentation of PlotXY (see [HCSS Javadocs](#)) for methods to change the font and the size of an annotation.

For the "exp" layer we have changed the appearance of the datapoints to a line with triangles on top of it. Please refer to section [What about these Layers?](#) for information on basic manipulation methods for layers.

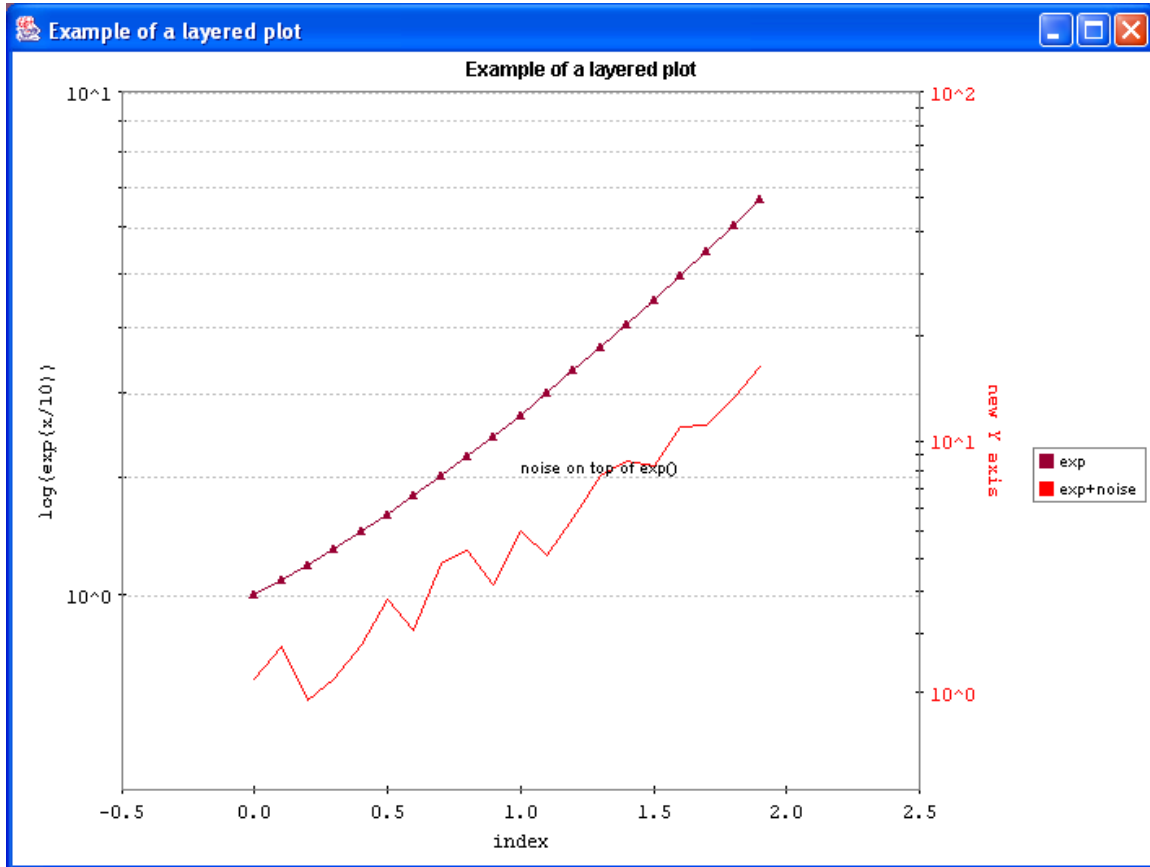


Figure 7-10: The results after running example 7.7 in the text.

#### 7.4.4 How can I make my plots more colourful?

Colours can be set for a number of parts within a plot. Methods can normally take a colour at creation time e.g. when adding a layer to the plot you can specify the colour to be used for its datapoints or for individual layers, labels etc. the colour can be specified with dedicated commands.

To specify a colour as an argument you have to pass a `Color` object. The easiest way to do this is to use their default names as e.g. `Color.blue`. Note that you first need to import `Color` as illustrated in the [introduction](#). An example for changing the symbol colour in [example 7.6](#).

```
layer.setSymbolColor(Color.green)
```

The default names for colours are: black, blue, cyan, darkGray, gray, lightGray, green, magenta, orange, pink, red, white and yellow (all preceded by `Color.`). Another easy

way to use a custom colour is to specify the red, green, blue value in ranges from 0-255. `Color(red, green, blue)`. So we could also do the following to get a similar green colour.

```
layer.setSymbolColor(Color(0,250,20))
```

## 7.5 **Creating File Output and Printing a Plot Without Displaying**

Sometimes you do not want to plot to the screen, but would rather write your plots directly to files.

```
# 1. Generate plot using the full constructor;
# the p=PlotXY(plotIsVisible=0) is equivalent to p=PlotXY(),
p.setPlotIsVisible(0)
#i.e. generate a plot and then set it to invisible. This works,
#but will cause window flashes
# Better is to completely render the plot. The last value of "0"
#indicates that the plot will not be made visible when it is created.

p=PlotXY(data,data.copy().power(2),"Title","TestLayer1",700,500,\
java.awt.Color.black,7,0,0,0)
```

To save the plot directly to file you can then use:

```
#2 Save as JPG , where filename could contain the full directory path
p.saveAsJPG("filename")
#3 or save as PNG , where filename could contain the full directory
path
p.saveAsPNG("/home/mypath/filename")

# 4. or save in the format of the default printer, with the specified
# filename (you may include the path) and page orientation
p.print("filename", "landscape")
```

## 7.6 **Handling Units in Plots**

The status is not definitive. The methods and example below gives you an idea of the current direction:

The methods for viewing and setting units on Axis objects are:

<code>a.showUnits(<i>boolean</i>)</code>	show/hide the <code>Units</code> on the specified axis
<code>a.scaleUnits(<i>Units</i>)</code>	scale specified axis with the ratio <code>oldUnits/newUnits</code>
<code>a.setUnits(<i>Units</i>)</code>	set new <code>Units</code> on the specified axis
<code>a.setUnits(<i>Units,boolean</i>)</code>	set new <code>Units</code> on the specified axis, with or without scaling if the units have the same dimensions

An example script is shown in [example 7.8](#). Step through this line by line to see what is happening. The final graphic is shown in Figure 7-11.

```
#Example 7.8 - Using units in plots.

from nT.quantity import *
#You need to import the package that knows handles units
d=DoubleIcd.range(100)
#we set up our range of numbers 0...99 in array "d"
ux=Frequency()
#we are going to have frequency on the x axis
ux.setPreferredUnits(Frequency.MEGAHERTZ)
#the frequency will be in Megahertz
uy=Length()
#we will use a length on the other axis
uy.setPreferredUnits(Length.CENTIMETERS)
#units will be centimetres
x=Column(d.copy().power(1),ux,"A frequency")
#now create a column - to be put into a table - which has
#data from our d array (to the power 1) and has frequency units
#and a description of "A frequency" associated with it
y=Column(d.copy().power(2),uy,"A length")
#now create a column - to be put into a table - which has
#data from our d array (to the power 2) and has length units
#and a description of "A length" associated with it
ds=TableDataset()
#x and y values are now placed in columns of a TableDataset
ds.addColumn(x)
ds.addColumn(y)
p=PlotXY(ds,"Sample Units Manipulation",Color.red)
#plot is made
p.getyAxis().showUnits(1)
p.getxAxis().showUnits(1)
#units for both axes are displayed

#2 scale units (and the Plot) with scaleUnits
p.getyAxis().scaleUnits(Length.METERS)

#3 set new units with setUnits
p.getyAxis().setUnits(Frequency.KILOHERTZ)

#4 Set Units work also as scaleUnits (for backward compatibility) if
the kind
# of units are the same: scale units (and Plot) with setUnits
p.getyAxis().setUnits(Frequency.HERTZ)

#5 to avoid setUnits doing a rescaling we can use the flag
allowScaling=False(or 0)
p.getyAxis().setUnits(Frequency.KILOHERTZ, 0)

#6 Hide/show the units with showUnits
p.getyAxis().showUnits(0)
p.getyAxis().showUnits(1)
```



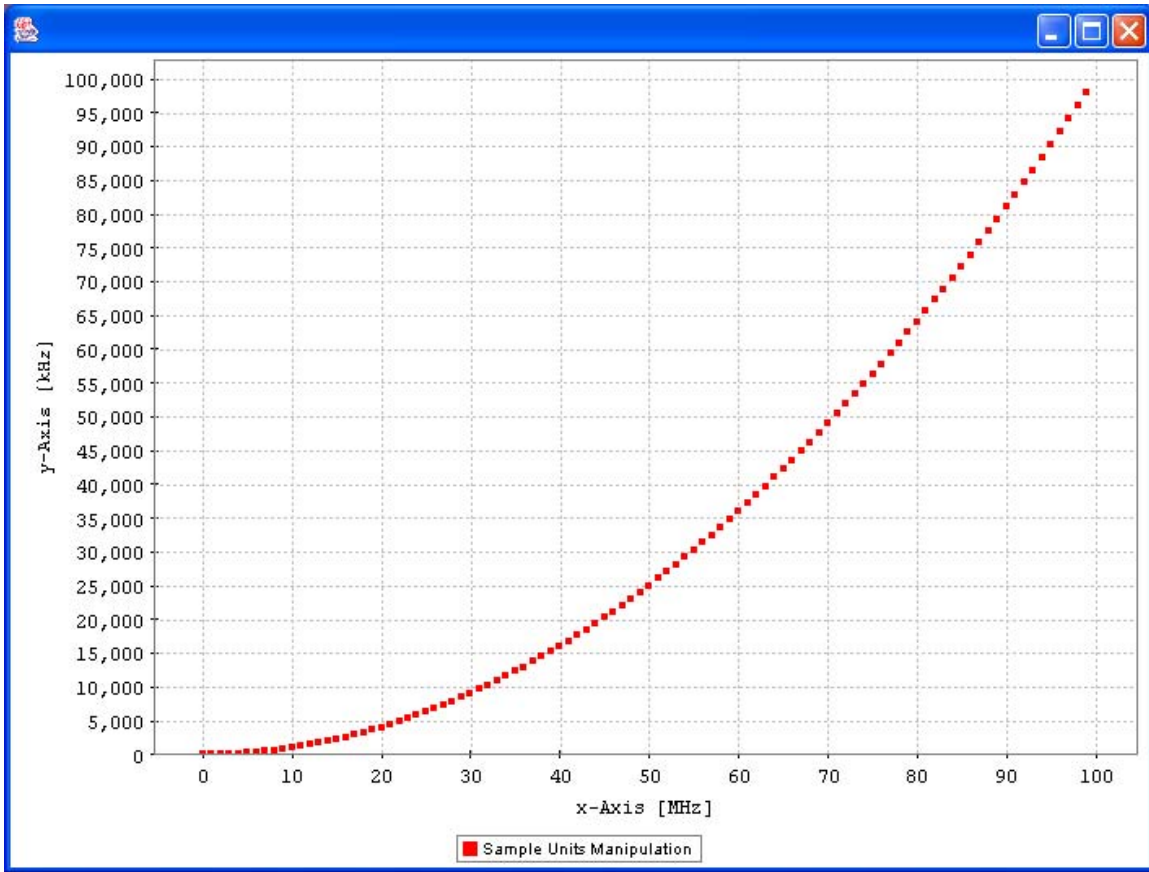


Figure 7-11: Illustration of how units can be used in PlotXY. This is the output from example 7.8.

## 7.7 What about a complete example?

You can find all examples contained in a single jython script that can be downloaded and run from within `jconsole` [here](#).



# Herschel IA Chapter 8

## 8 Display: Handling Images with Herschel IA

### Chapter 8 Contents

- 8.1 [Introduction](#)
- 8.2 [Using ImageDatasets](#)
- 8.3 [How can I display my image?](#)
- 8.4 [Display in more detail](#)
- 8.5 [How can I use Operations on my images?](#)
  - 8.5.1 [Clamping \(or clipping\) an Image](#)
  - 8.5.2 [Cropping an Image](#)
  - 8.5.3 [Histogram of an Image](#)
  - 8.5.4 [Rotating an Image](#)
  - 8.5.5 [Scaling an Image](#)
  - 8.5.6 [Translating an Image](#)
  - 8.5.7 [Transposing an Image](#)
- 8.6 [How can I display my own numeric2d datatypes?](#)
- 8.7 [How to Use Different Layers](#)
- 8.8 [How to Place Annotations on an Image](#)
  - 8.8.1 [Annotations from the Command Line in your IA session](#)
  - 8.8.2 [Annotations using the annotation toolbox](#)
- 8.9 [Sample JPEG Image and Extended Example Demo Script](#)

---

### 8.1 Introduction

This chapter describes how to use `imageDatasets` to store your image data, display your image and do some basic operations on your images. It is not intended to elaborate on the complete set of functionalities available for image manipulation. For this the reader is referred to the [API documentation for the image package](#).

Please note that the image functionalities of the HCSS are still **under development**. This means that the interface is likely to change in the future.

**Note:** A number of classes that are used with the IA image display need to be imported before use or they will produce a strange `NameError` message. You can simply do this with the following statements in `Jconsole`:

```
from nT.quantity import FluxDensity
from nT.quantity.constant import *
from nT.quantity import Angle
from java.awt import Font
```

The above imports are required for running the examples provided in this chapter.

The image package `herschel.ia.image` is automatically loaded when starting the default version of IA.

## 8.2 Using ImageDatasets

An `ImageDataset` is a special type of dataset composed of :

- the image : described as a `Double2d` (2D real number array)
- the errors of the image : described as a `Double2d` (2D real number array)
- a mask : described as a `Bool2d` (2D boolean array)

The `ImageDataset` also holds information to do coordinate conversions (using the World Coordinate System, WCS) and information of the wavelength at which the image was taken.

When constructing an `ImageDataset`, you should usually first construct a [WCS](#) object for the coordinate information, and the `Double2d`'s and `Bool2d` needed for the image.

[Example 8.1](#) shows how to construct an `ImageDataset` with WCS coordinates associated with it, an image of 60x40 pixels, all errors set to 0.0 and with one pixel masked out (the pixel 55, 35).

```
#Example 8.1.
myWcs = Wcs(crpix1 = 29, crpix2 = 29, crval1 = 30.0, crval2 = -22.5)
#The first line sets up a coordinate system for our example. Pixel
#(29,29) is indicated as having RA = 2h00m (crval1) and
#Dec = -22d30m (crval2)
myIm = Double2d(60, 40)
#Here we have created the array space for the image.
for i in range(0,60):
    for j in range(0,40):
        myIm.set(i, j, i + j)

#The above three lines fill the 2D array with values i + j
myMask = Bool2d(60, 40).not()
#Now we set up the mask image, which has "true" values for all
#positions.
myMask.set(55, 35, 0)
#...and indicate one "bad" pixel at position (55,35)
myQuant = FluxDensity(1.0, ASTRONOMICAL.JANSKYS.milli())
#The flux associated with one count in the image (equivalent to BSCALE
#in a FITS image.
```

```
myImage = ImageDataset(description="test image", image = myIm, mask =\  
myMask, quantity = myQuant, wcs = myWcs)  
  
#Creates the image Dataset itself  
myImage2 = ImageDataset(wcs = myWcs)  
#Make another image "myImage2" for displaying and apply the same WCS to  
#it  
myImage2.importFile("ngc6992.jpg")  
#Import the JPG file of NGC6992. The import will get the image and  
#apply the coordinates. There is no mask or error file.
```

In the first line, a WCS (World Coordinates Class) object is created. The center pixel is set to (29, 29) which is a projection of the sky coordinate with right ascension 2h00m00s and declination -22d30'00". For more information consult the [WCS documentation](#)

In the second to fifth line, a Double2d is constructed which describes the image and the pixel values are set to  $i + j$  (where  $i$  is the  $x$  coordinate and  $j$  is the  $y$  coordinate).

The sixth and seventh line describe the mask. The mask for all pixels is set to true in line 6 and in line 7, pixel (55, 35) is masked out.

In line 8, the quantity for the pixels is set.

Line 9 constructs the `ImageDataset`. *When the errors are not explicitly set, all errors are set to 0.0.*

Instead of constructing a new `ImageDataset`, you can also import an image (.jpg, .png, .bmp, .tiff, .xpf, .gif, or .pnm) into the `ImageDataset`. NOTE: At present, the imported image can NOT be a FITS file. The image file [ngc6992.jpg](#) is provided here. *This should be placed in the same directory that jconsole/IA is started from so that it can be most easily accessed.*

### 8.3 How can I display my image?

Let's display the images produced in [Example 8.1](#).

```
myDisplay = Display(myImage)  
myDisplay2 = Display(myImage2)
```

The labels "myDisplay" and "myDisplay2" allow us to refer to the displays and their contents separately.

The result of these two commands is shown in Figure 8-1. In each display, the image itself is shown in the window. The *masked out pixels are shown as black pixels*.

At the top right, there is a second, smaller, frame where an *overview of the image* is shown. On this overview, axes are also drawn. For this image, North is down and East is to the right.

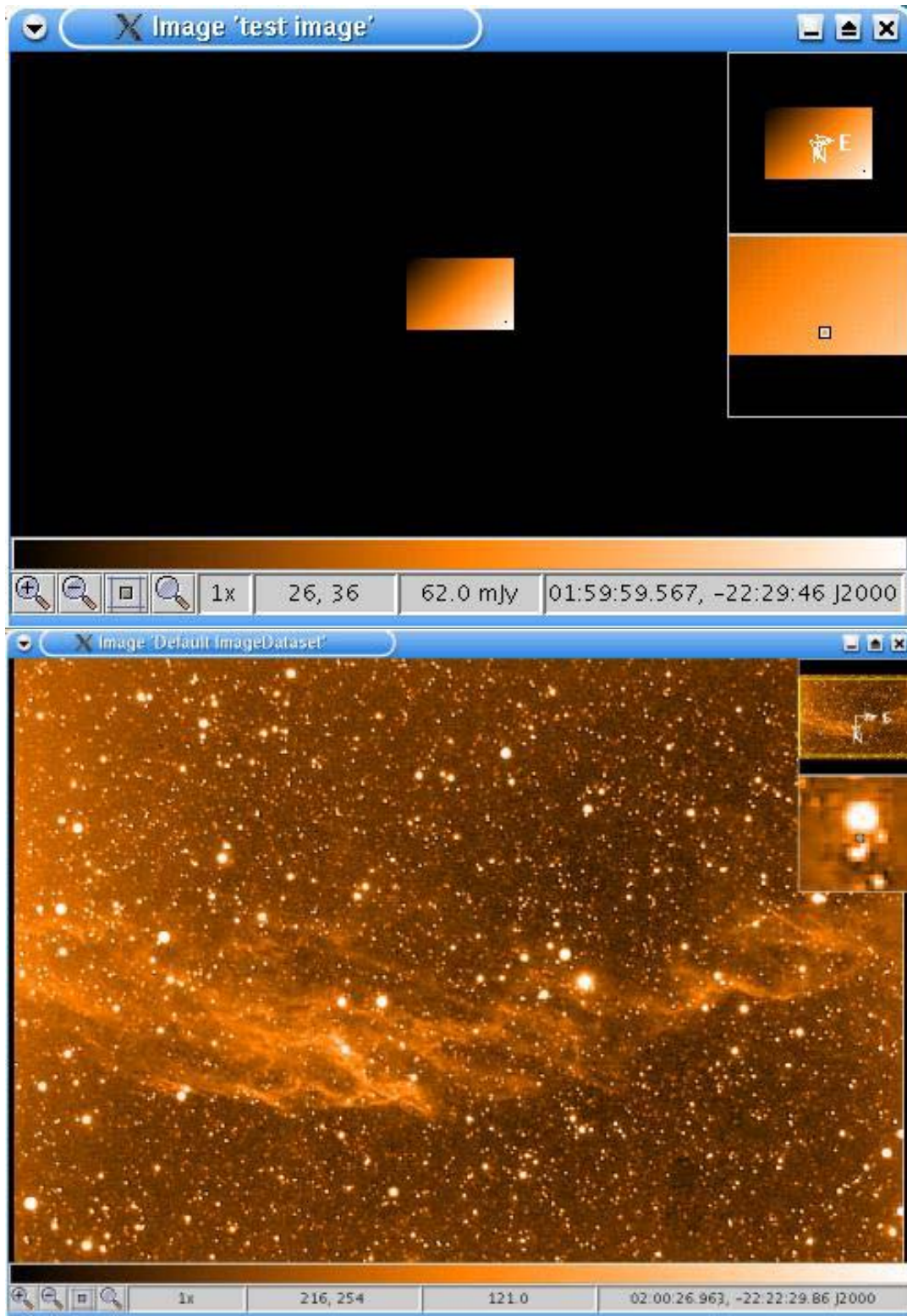


Figure 8-1: Display of image created in IA (top) and of imported JPEG file (bottom) from Example 8.1. The pixel position (x and y, the pixel value and the WCS coordinates of the mouse position are shown along the bottom of the image, together with zoom capabilities. NOTE: THE PIXEL POSITION IS CURRENTLY DISPLAYED AS Y, X.

Under the window where the overview is located, there is another frame which shows a zoomed in version of the *area located under your mouse*.

Directly under the image, you can see a color bar. It is possible to click on the *colorbar* and move your mouse to change the slope of it.

At the bottom of the window, you can see a *statusbar*. Using the four icons you can zoom in, zoom out, zoom to fit the window and return to the normal zoom (1x). Beside the icons, you can see the currently displayed magnification, the pixel coordinates at which the mouse is pointing, the pixel value and the concurrent sky coordinates (based on the WCS information provided).

When you click with the *right mouse button on the image*, you get a menu. Here, you can open a window that allows changes to the color table ('Edit colors'), open a window where you can edit the cut levels ('Edit cut levels'), zoom in on the place where the mouse is located ('Zoom in'), zoom out, create a screenshot or print the image.

## 8.4 *Display in more detail.*

From here on, we will elaborate on `myDisplay2`. On a display object, you can do a lot of things. Not all of the possibilities are described here. For an overview of all methods, have a look in the Display [javadoc](#).

Some of the more useful methods are listed below :

<code>getImageCoordinates(<i>ra</i>, <i>decl</i>)</code>	Returns the image coordinates corresponding with the given sky coordinates
<code>getSkyCoordinates(<i>x</i>, <i>y</i>)</code>	Returns the sky coordinates corresponding with the given pixel coordinates
<code>getIntensity(<i>x ra</i>, <i>y dec</i>)</code>	Returns the intensity of the pixel at the given (Sky or Image) coordinates
<code>setColorTable(<i>colortableName</i>, <i>intensityName</i>, <i>scaleName</i>)</code>	Set the <i>colortable</i> of the image
<code>setCutLevels(<i>min</i>, <i>max</i>)</code>	Set the cut levels between <i>min</i> and <i>max</i>
<code>setZoomFactor(<i>zoomFactor</i>)</code>	Zoom by the given <i>zoomfactor</i>
<code>addAnnotation(<i>annotation</i>, <i>x ra</i>, <i>y dec</i>)</code>	Adds an annotation to the image on the given coordinates

The following [Example 8.2](#) shows how some of the above can be used from the command line in `Jconsole`.

```
#Example 8.2: Illustration of Display options  
myDisplay2.setZoomFactor(2)
```

```
#sets the zoom factor on the second of our displays
print myDisplay2.getSkyCoordinates(434, 236)
#prints output to the console of the sky coordinates at pixel (434,236)
print myDisplay2.getIntensity(434, 236)
#Prints the intensity of the pixel at this position
myDisplay2.setCutLevels(50, 250)
#sets min and max intensity levels for display
myDisplay2.addAnnotation("Annotation", 506, 300)
#provides an annotation at coordinate (506,300)
```

The result can be seen in Figure 8-2.

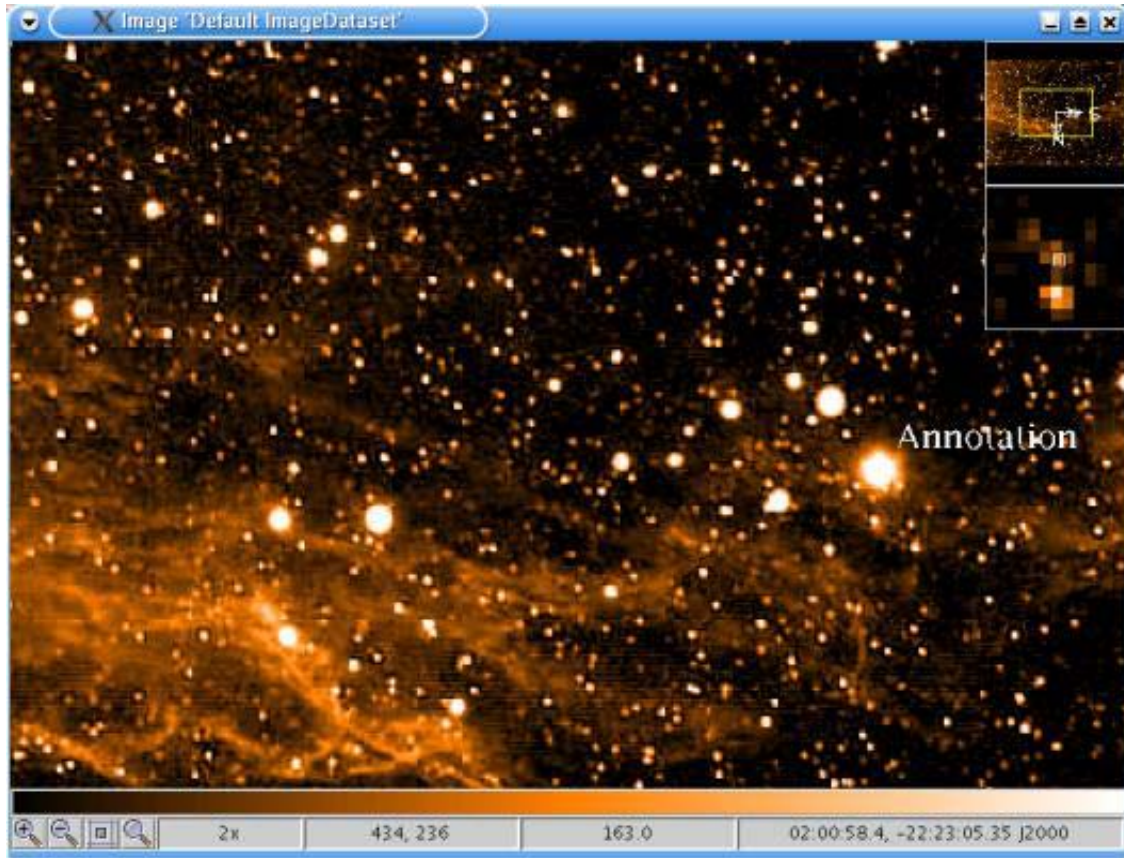


Figure 8-2: Results from Example 8.2 showing the get sky coordinates, zoom and annotation features of Display.

## 8.5 How can I use Operations on my images?

At the moment of this writing, there are only a few operations available on ImageDatasets :

At the moment of this writing, there are only a few tasks available on ImageDatasets :

- **Clamp** : Clamps the high and or low values of an image.
- **Crop** : Crops the images to a smaller image.
- **Histogram** : returns a TableDataset with the histogram of the image.

- **Rotate** : Rotates the image.
- **Scale** : Magnifies or compresses the image size.
- **Translate** : Translates the image.
- **Transpose** : Transposes the image.

### 8.5.1 Clamping (or clipping) an Image

We can set the minimum and maximum values for our display by the use of the clamping command. All pixels with values at or below the “low” parameter value are given the “low” parameter value. Similarly, all pixels with values at or above the “high” parameter value are allocated the “high” parameter value. An example of its use is given below. Here we create a clamped image in an ImageDataset called “im\_clam” and then display it

```
im_clam = Clamp() (image=myImage2, low=40.0, high=100.0)
Display(im_clam)
```

This could also have been done on one line without the necessity of creating the ImageDataset of the clamped image.

```
Display( Clamp() (image=myImage2, low=40.0, high=100.0) )
```

### 8.5.2 Cropping an Image

The size of an image can be reduced through cropping. The parameters, “x1”, “y1” define the **top left** hand corner and “x2”, “y2” define the **bottom right** hand corner of the cropped image. The following illustrates its usage. A cropped image is created, then it is displayed.

```
im_crop = Crop() (image=myImage2, x1=100, y1=50, x2=300, y2=250)
Display(im_crop)
```

### 8.5.3 Histogram of an Image

A histogram of the pixel values of an image can be obtained in a table by using the Histogram() command. In order to obtain the table for display requires the following

```
im_hist = Histogram() (image=myImage2)
```

This produces a table with pixel values in one column and frequency in the second. In order to display this we can use PlotXY (see [Chapter 7](#)).

```
vals = im_hist[0].data
freq = im_hist[1].data
p = PlotXY(x=vals, y=freq, title="Histogram example")
```



```
p.setxLabel(im_hist[0].description)  
p.setyLabel(im_hist[1].description)
```

The histogram values are placed in two arrays to be plotted against each other (vals and freq), PlotXY does the plotting and the last two commands uses the column descriptions in the table for axis labels to our plot. The final histogram is shown in Figure 8-3.

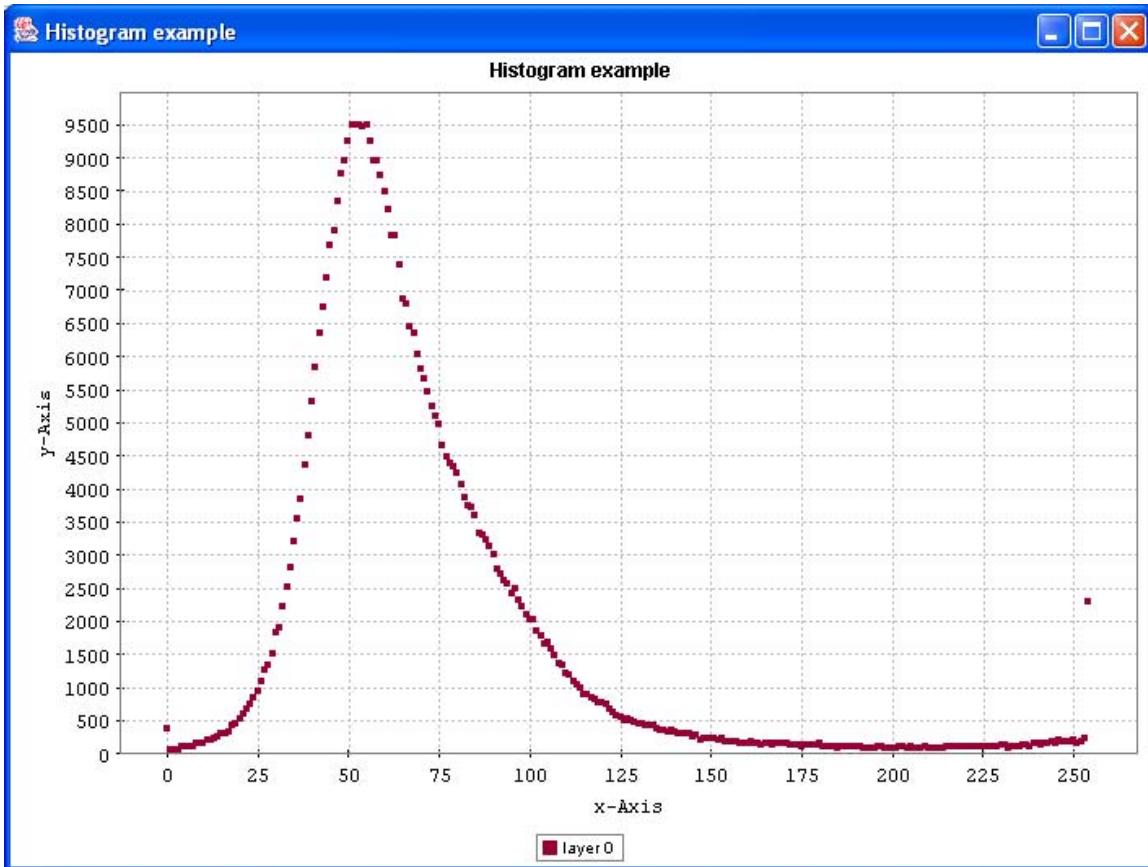


Figure 8-3: Histogram plot of the image of NGC6992 (Veil nebula) stored in the ImageDataset “myImage2”.

### 8.5.4 Rotating an Image

The following input line provides an example of how an image may be rotated.

```
rot = Rotate() (image=myImage2, angle=30.0)  
Display(rot)
```

This example rotates the image over 30.0 degrees (clockwise) then displays the rotated image. The result is shown in Figure 8-4.

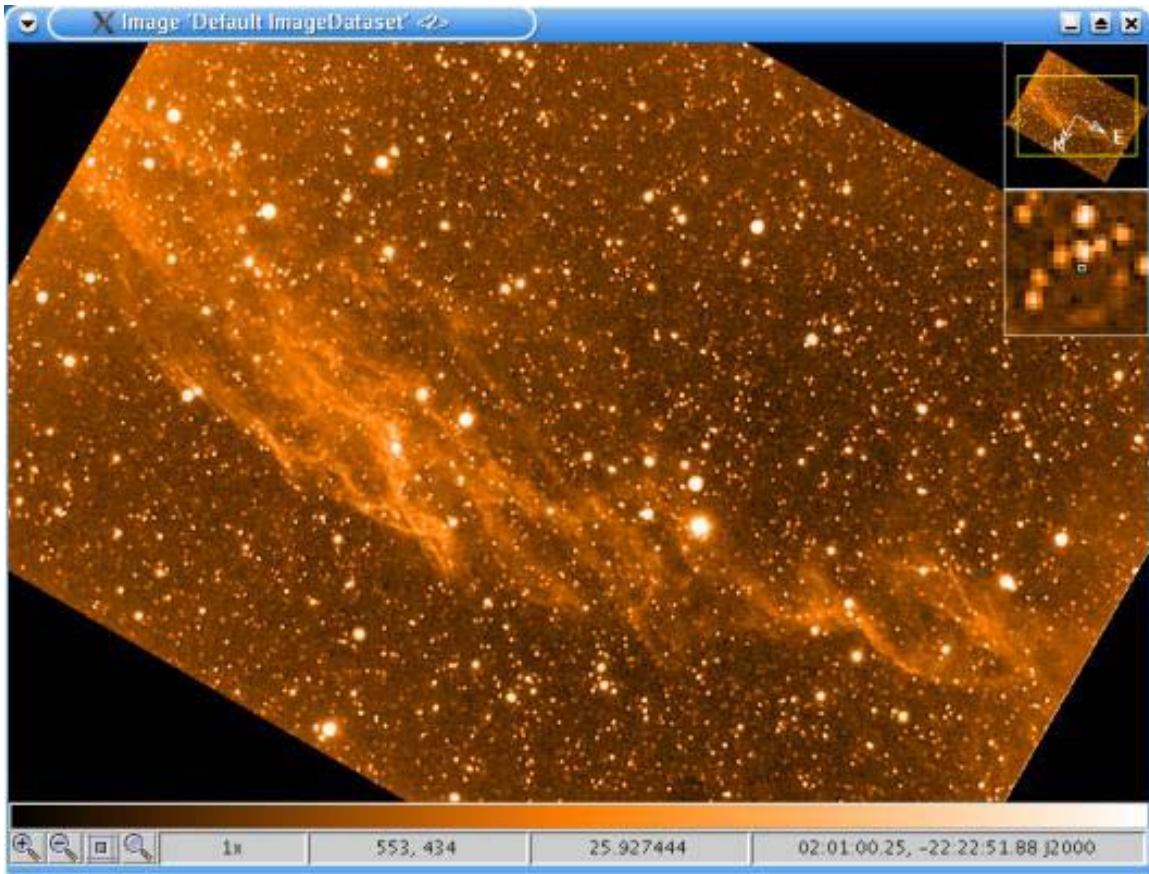


Figure 8-4: Illustration of how an image is rotated by +30 degrees (clockwise) using the IA display package.

When rotating the image, several types of interpolation are possible. By default linear interpolation is used. There are four types of interpolation possible.

- **Rotate.INTERP\_BILINEAR** (*the default* – interpolates one pixel to the right and one below)
- **Rotate.INTERP\_NEAREST** (direct pixel copying)
- **Rotate.INTERP\_BICUBIC** (uses interpolation via a piecewise cubic polynomial)
- **Rotate.INTERP\_BICUBIC2** (variant of bicubic interpolation that can give sharper results than bicubic)

In order to use the different interpolation schemes requires the “interpolation” parameter to be defined. The following illustrates this point.

```
b = Rotate() (image=myImage2, angle=50.0, \  
            interpolation=Rotate.INTERP_NEAREST)  
Display(b)
```

To use the bicubic interpolations also requires a declaration of the number of subsample bits to be used. More bits gives more accuracy but at extra computational costs. The

default bit subsampling is 16. An example of using bicubic interpolation with 32 bit subsampling is

```
b = Rotate() (image=myImage2, angle=50.0, \  
            interpolation=Rotate.INTERP_BICUBIC, subsampleBits=32)  
Display(b)
```

### 8.5.5 Scaling an Image

The size of an image can be magnified in the x and y directions independently using the Scale() command. It can be used in a similar way to the Rotate() command (see section 8.5.4) and has the same set of interpolations available to it. The following magnifies in the x direction by a factor of 0.5 and in the y direction by a factor of 2. This is then displayed.

```
s = Scale() (image=myImage2, x = 0.5, y = 2, \  
            interpolation=Scale.INTERP_BICUBIC, subsampleBits=32)  
Display(s)
```

This provides a very elongated image (see Figure 8-5).

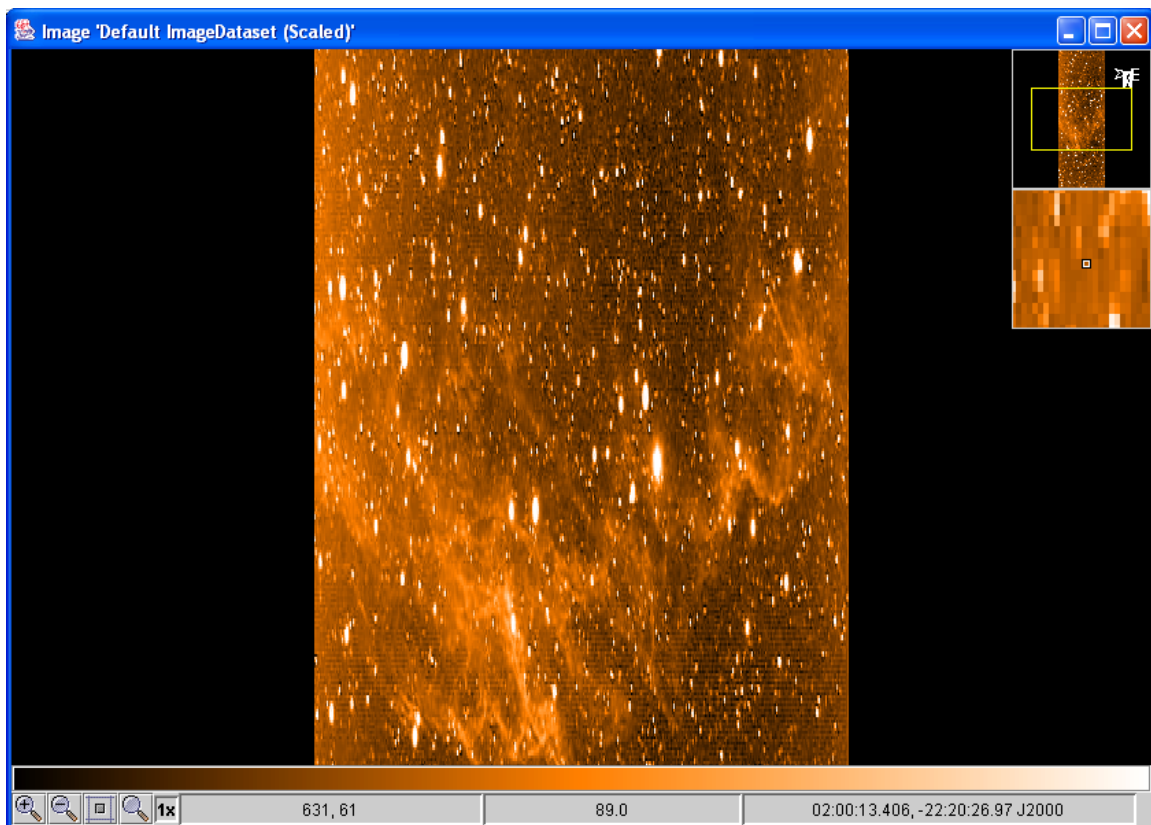


Figure 8-5: An example of independent x and y axis magnification of images using Scale().

### 8.5.6 Translating an Image

An image can be translated in x and y pixels (fractional pixels allowed) or by an angle (in degrees) in ra and dec (or sky coordinates).

**Pixel translation:**

```
im_trans_pix = Translate() (image=myImage2, x= 50.4, y = -5.3)
```

**Sky coordinate translation:**

```
im_trans_sky = Translate() (image=myImage2, ra=Angle(0.02, \
    ANGLE.DEGREES), dec=Angle(-0.05, ANGLE.DEGREES))
```

The angle can also be expressed in radians (ANGLE.RADIANS). An alternate form for the angle is degrees/minutes/seconds or hours/minutes/seconds.

**Ddmms:**

```
Angle(dd, mm, ss, 0)
```

Where dd = integer number of degrees, mm = integer number of minutes and ss=double number of seconds of arc.

**Hhmmss:**

```
Angle(hh, mm, ss, 1)
```

This is similar, but now the first number is interpreted as being hours rather than degrees.

So to translate an image 10 minutes (of time) in RA requires the following

```
im_trans_sky = Translate() (image=myImage2, ra=Angle(1, 0, 0, 1))
```

### 8.5.7 Transposing an Image

Images can be transposed in a number of ways.

- 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 (clockwise rotation)
- ROTATE\_180
- ROTATE\_270

An example of how to use this command is

```
im_pose = Transpose() (image=myImage2, type=Transpose.FLIP_DIAGONAL)
```

This command flips the image so that pixels to bottom left appear at top right a flip around the image axis going from topleft to bottom right of the image (*antidiagonal* works on the opposite diagonal).

## 8.6 *How can I display my own numeric2d datatypes?*

In many cases, you may have constructed your own 2D array, possibly with a datatype other than Double2d (e.g., Int2d) that you want to display. This can be done by simply feeding the datatype to Display. [Example 8.3](#) provides an illustration of how to input a 2D array of Int2d (of 16 by 18 pixels) and displays it. A zoom factor of 20 is also used.

```
#Example 8.3: Using Display with 2D integer arrays
```

```
image = Int2d(15, 17)
#image created of 2D integer array 15x17
for i in range(0, 15):
    for j in range(0, 17):
        image.set(i, j, i + j)
#loop around placing 'intensity' values into the array

d = Display(image, zoomFactor = 20, cutLevels = (0, 30))
#display the image with appropriate min/max levels and zooming
```

Figure 8-6 shows the results from running [Example 8.3](#).

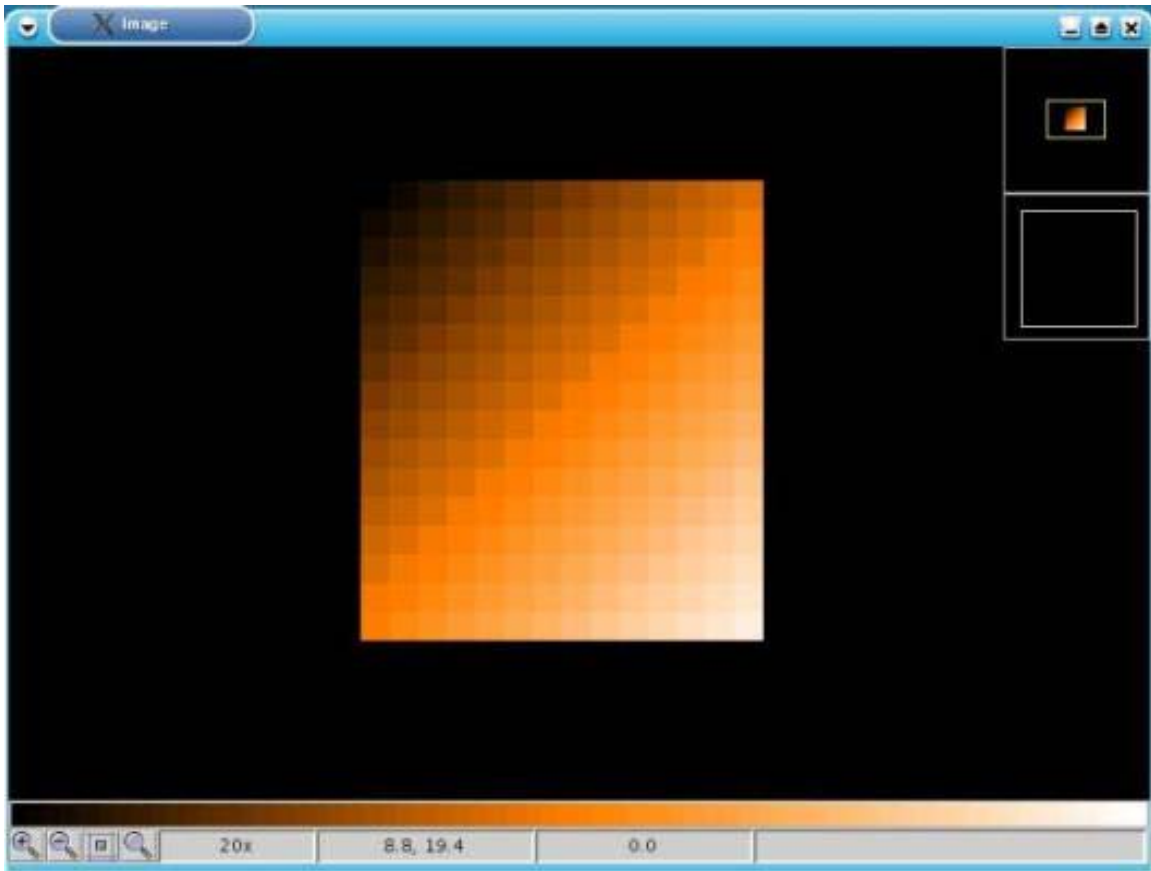


Figure 8-6: Example 8.3 – display of image created from user-supplied numeric 2D array.

## 8.7 How to Use Different Layers

It is possible to show different layers of an image. This can be done by adding a layer to the existing image, but it can also be done by displaying a cube of numeric3d datatype (like an Int3d, Double3d, ...).

The example in Figure 8-7 is an elaboration of the first example. It has been created using the command

```
myDisplay.addLayer(myImage2)
```

We add myImage2 to myDisplay. The screenshot shows that there appears a *slider in the statuspanel*, where you can switch between the different layers. The settings of the new layer will be the same as the settings of the old layers (cut levels, annotations, zoom factor, ...).

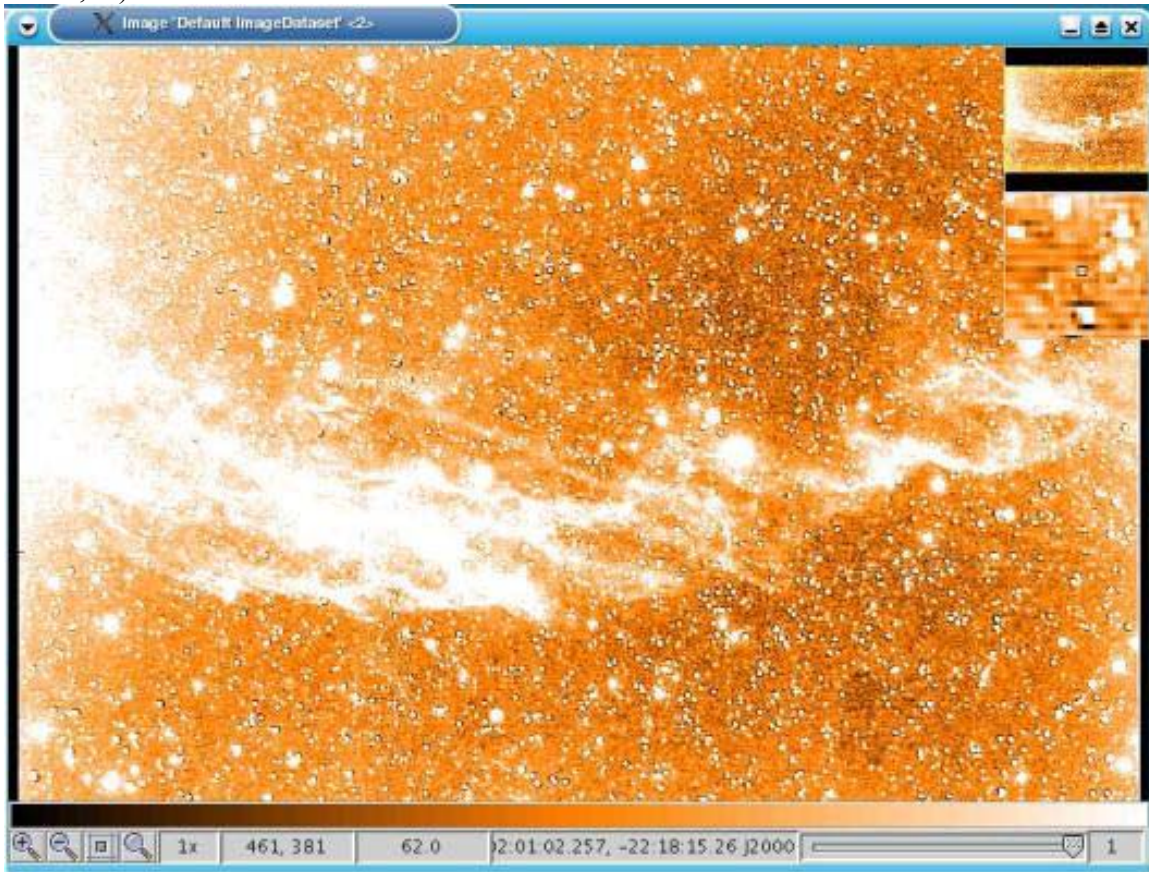


Figure 8-7: The use of layers in Display. Swapping between layers is performed using the slider at bottom right.

## 8.8 How to place annotations on the image

It is possible to add annotations to the image. This can be done in two different ways :

1. Using the command line from your IA session
2. Using the annotation toolbox

Once the annotations are on the image, it is possible to use the mouse to select an annotation, move it around and change the size of the annotation.

### 8.8.1 Annotations from the Command Line in your IA session

It is possible to add annotations to the image from the IA prompt.

The following is possible :

- *Text annotations* : Using `addAnnotation(...)`, `setAnnotationFont(...)` and `setAnnotationFontColor(...)` methods.
- *Greek text annotations* : Using `addGreekAnnotation(...)`, `setAnnotationFont(...)` and `setAnnotationFontColor(...)` methods. The `addGreekAnnotation` method translates the normal characters to greek characters (a becomes alpha, b beta, c gamma, ...). Numbers are not adapted.
- *Figures as annotations* : Using `addEllipse(...)`, `addLine(...)`, `addPolygon(...)`, `addPolyLine(...)` and `addRectangle(...)` methods. The `addPolygon` and `addPolyLine` methods need an array of doubles as parameter. In this array, the coordinates of the points should be added in this way : `polygon([x1, y1, x2, y2, ...], ...)`

[Example 8.4](#) illustrates how to place annotations onto displays.

```
#Example 8.4: command line addition of annotations to images

myDisplay2.addAnnotation("Veil nebula", 321, 224)
#Places annotation at position (321, 224) on image 'mydisplay2'
myDisplay2.setAnnotationFont(321, 224, Font("Dialog", 0, 64))
#change font type and size for the image 'mydisplay2'
myDisplay2.setAnnotationFontColor(321, 224, Color(0, 0, 255))
#annotation color changed
myDisplay2.addEllipse(500.0, 308.5, 38.0, 37.0, 3.0, Color(0, 0, 0))
#ellipse added with center at (500, 308.5) width=38, height=37
#linewidth = 3.0 and black color.
myDisplay2.addGreekAnnotation("a = 12.34, d = +30.30", 100, 500)
#adds a position label with greek letter notation at position (100,500)
myDisplay2.setAnnotationFont(100, 500, Font("Dialog", 0, 64))
#change font of annotation of at (100, 500)
myDisplay2.setAnnotationFontColor(100, 500, Color(0, 0, 0))
#...and change its color to black too
myDisplay2.setAnnotationFontColor(100, 500, Color.white)
#...but white is more visible.
```

The result is shown Figure 8-8.

## 8.8.2 Annotations using the annotation toolbox

It is much easier to add the annotations using an annotation toolbox. The annotation toolbox can be shown using :

```
a = myDisplay2.annotationToolbox(_interpreter)
```

The `_interpreter` variable is needed to give the toolbox the needed information about the variables that are defined in the `jconsole` session. Thanks to this, the toolbox will be able to generate the `jython` code that is needed to reconstruct all annotations. If the `_interpreter` variable is not given, the annotation toolbox will not give the sourcecode.

It is also possible to fire up the annotation toolbox by clicking with the right mouse button on the image. A popup will appear where you can select 'Annotation toolbox'. If you fire up the annotation toolbox using the popup menu, the `jython` code can not be generated.

The annotation toolbox is shown in Figure 8-9.

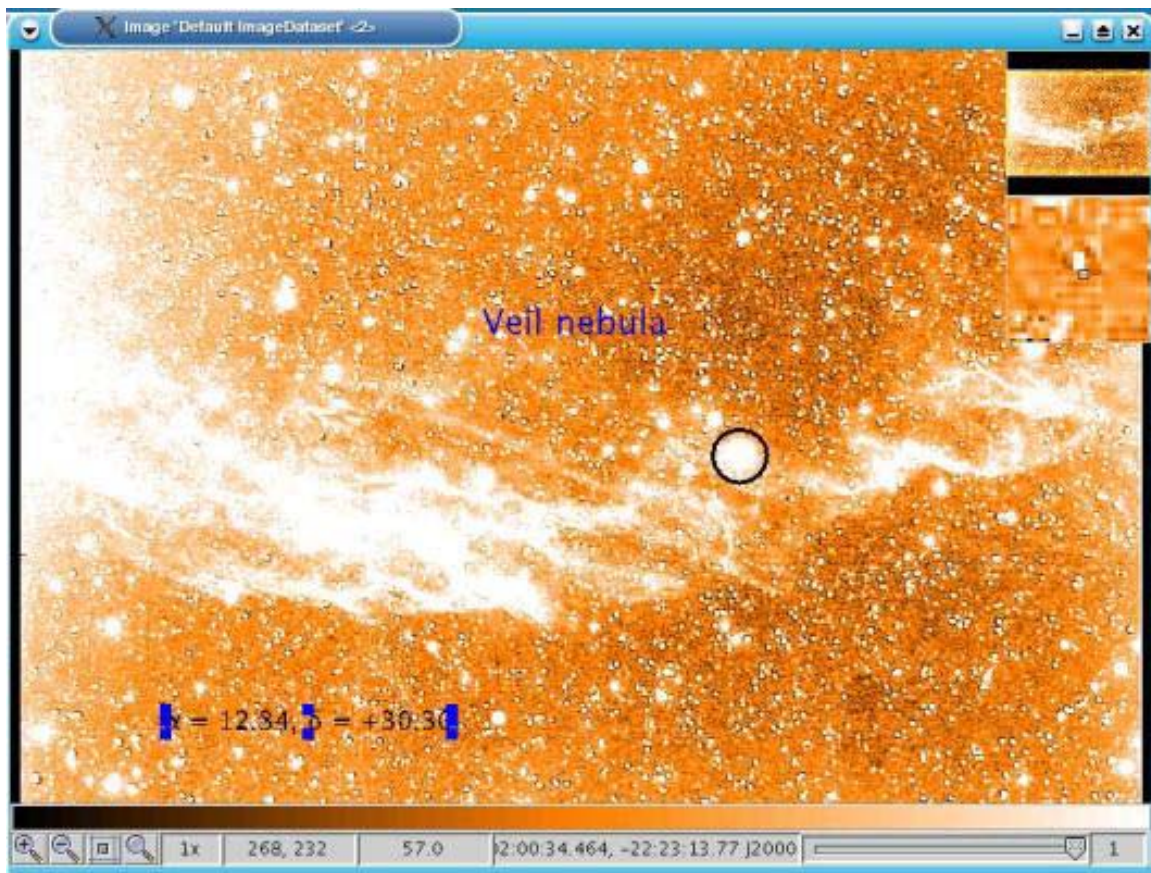


Figure 8-8: Output from the use of Example 8.4 illustrating how annotations can be added in various forms.



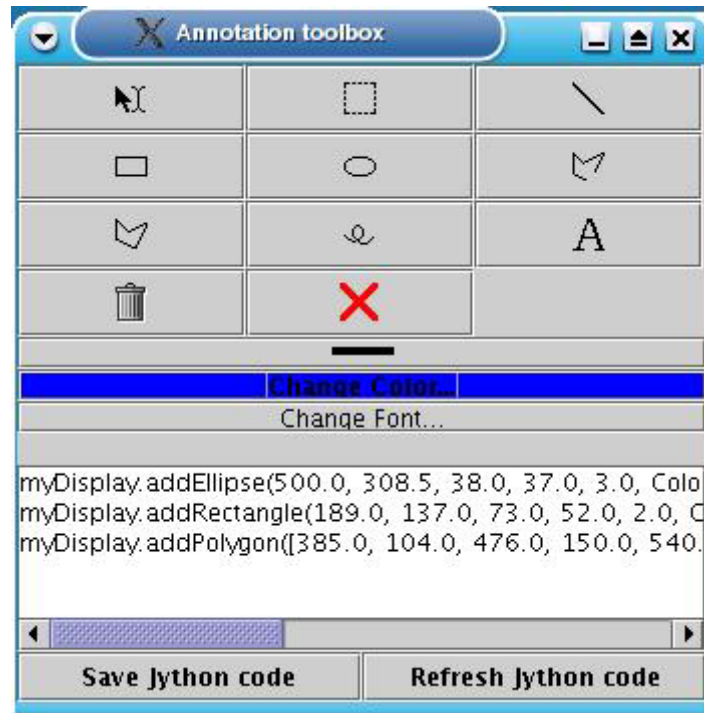


Figure 8-9: The annotation toolbox made available for “myDisplay”

The icons in the annotation toolbox appearing in Figure 8-9 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 following three buttons change the view of the annotation :

- Change the thickness of the line

- Change the color of the annotation
- Change the font of the text annotation

The lower part of the annotation toolbox is only visible if the `_interpreter` variable is given with the `annotationToolbox(...)` method. The jython code needed to regenerate all annotations is given there. If you change the size of a text annotation, this will not be reflected in the jython code.

## **8.9      *Sample JPEG Image and Extended Example Demo Script***

The JPEG file of NGC6992 used as an example throughout this chapter is available here ([ngc6992.jpg](#)).

An extended example demo script is available here ([Example1.py](#)).



## Herschel IA Chapter 9

# 9 Other IA Packages: What is Available?

### Chapter 9 Contents

- 9.1 [Introduction](#)
  - 9.2 [Overview of JavaDocs Documentation for IA Packages](#)
  - 9.3 [Package view](#)
  - 9.4 [Class view](#)
  - 9.5 [Tree view](#)
  - 9.6 [Deprecated view](#)
  - 9.7 [Index view](#)
  - 9.8 [IA Packages And Documentation](#)
    - 9.9.1 [herschel.ia.dataflow](#)
    - 9.9.2 [herschel.ia.dataset](#)
    - 9.9.3 [herschel.ia.demo](#)
    - 9.9.4 [herschel.ia.doc](#)
    - 9.9.5 [herschel.ia.image](#)
    - 9.9.6 [herschel.ia.io](#)
    - 9.9.7 [herschel.ia.jconsole](#)
    - 9.9.8 [herschel.ia.numeric](#)
    - 9.9.9 [herschel.ia.plot](#)
    - 9.9.10 [herschel.ia.task](#)
    - 9.9.11 [herschel.ia.ui](#)
- 

### 9.1 Introduction

To use the various packages within HCSS the user needs to import the packages into the HCSS session. This can be done automatically using the `import.py` file, editable by the user, for packages that are used frequently. Whether in the `import.py` or via a `jconsole` command line, all packages are imported via command lines of the type

```
from herschel.ia.numeric import *
```

There are several packages available within the HCSS. In this chapter we provide an overview of the main IA packages only. There are also a number of external library sets that are imported into IA when it is initiated (*these will be described in a later update to*

the manual). A full listing of classes (programs) available in the HCSS system is given in [Appendix B](#).

A number of IA packages have already been discussed in some detail. The *IA numeric* package was discussed in [Chapter 5](#), the *IA plot* package in [Chapter 7](#) and the *IA display* package is described in [Chapter 8](#). Illustrations of how to use parts of several other HCSS packages are also shown in earlier chapters.

Most packages also have sub-packages containing classes that presently require separate import statements, e.g.,

```
from herschel.ia.numeric.function import *  
from herschel.ia.numeric.function.fit import *
```

The contents of these sub-packages are also briefly described in this chapter.

## 9.2 Overview of JavaDocs Documentation for IA Packages

The javadoc is normally started up as three frames in a web browser as illustrated in Figure 9-1. The upper left frame contains the *packages index* which is a clickable list of all packages in the system. The title in that frame represents the HCSS build number for which this documentation is valid. The lower left frame contains the *classes index* which is a clickable list of all classes. The selection of classes shown in this frame depends on the package that was selected in the packages index frame. The *Main frame* contains overview information on the system and packages or shows the javadoc for a selected class.

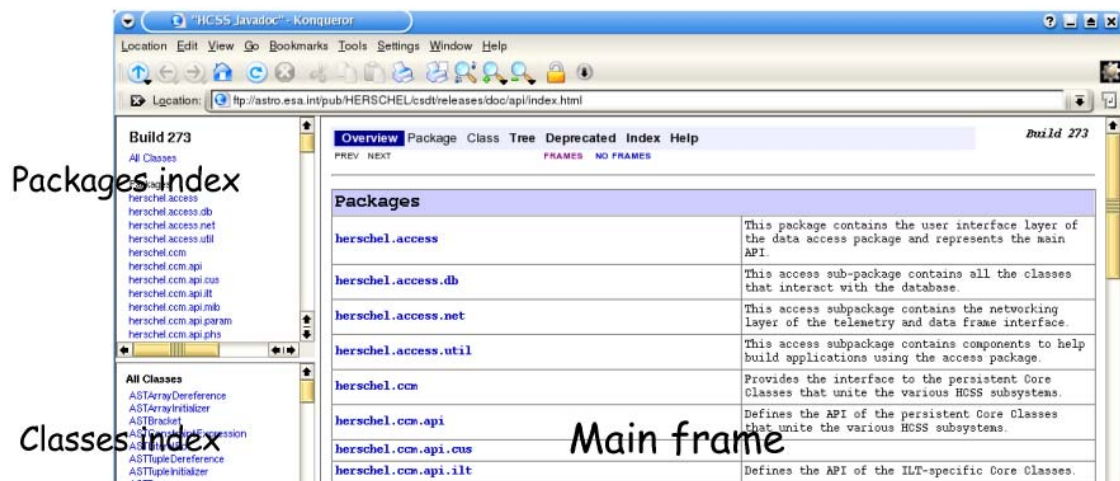


Figure 9-1: Web browser page of JavaDocs top level frame

Click in the *Packages index* frame to select a package and update the *Classes index* frame to show those classes for the selected package. Click the *Classes index* frame to show the javadoc of a particular class in the *Main frame*.

The *Main frame* contains a kind of navigation bar at the top where the view in this frame can be selected. The figure above shows the overview of all the packages. Other views are: Package, Class, Tree, Deprecated, Index, and Help. These views will be explained in more detail below. In the overview the Package and Class views are disabled, they become available when a package or class is selected. Figure 9-2 shows the slightly expanded navigation bar for the Class view.



Figure 9-2: Navigation bar on the “class view” of JavaDocs

Note that the navigation bar provides the possibility to browse through packages and classes with `NEXT` and `PREVIOUS` and provides direct access to the specific parts of the class documentation e.g. constructors (start class/program) or methods (which can be thought of as sub-routine components of programs that can be applied). It is also possible to switch between `FRAMES` and `NO FRAMES`. With `NO FRAMES` only the *Main frame* of the `javadoc` will be shown and index frames become unavailable.

### 9.3 Package view

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain four categories: *Interfaces summary*, *Classes summary*, *Exceptions and Error summary*. Not all categories are always present. At the end there is the package description and possible links to specific and/or related documentation.

Figure 9-3 shows the `herschel.ia.dataset` package which contains a number of interface and classes e.g. *Dataset* and *TableDataset*. You can see that the *Classes index* frame provides a clear separation of interfaces and classes and the *Main frame* shows the interface and class summaries and provides a brief package description with links to package specific info at the bottom (The image of the *Main frame* has been manipulated to show the categories available without too much cluttering the picture). You can navigate to the interface and class detailed documentation by clicking the names in the summary tables or in the *Classes index* frame.

### 9.4 Class view

Each class and interface has its own separate page in the *Main frame*. Each of these pages has three sections consisting of a class/interface description, summary tables for constructors and methods, and detailed descriptions of constructors, methods and attributes. The information shown in the class view is restricted to the public *API* (*Application Programming Interface*).

The screenshot shows a web browser displaying the JavaDoc for the 'herschel.ia.dataset' package. The browser's address bar shows 'http://localhost/~rik/hcss-current-docs/api/index.html'. The page has a navigation menu at the top with 'Overview', 'Package', 'Class Tree', 'Deprecated', 'Index', and 'Help'. The 'Package' tab is selected, and the page title is 'Package herchel.ia.dataset'. Below the title, there is a brief description: 'This package provides a uniform approach for holding, annotating, quantifying and attributing data as is defined in the herchel.ia.numeric package.' A 'See:' section points to 'Description'. The page is divided into two main sections: 'Interface Summary' and 'Class Summary', each containing a table of items with their descriptions. The 'Interface Summary' table lists 'Algorithm', 'Annotatable', 'Attributable', 'Composite', and 'Dataset'. The 'Class Summary' table lists 'AbstractDatasetAndDataVisitor', 'AbstractDatasetVisitor', 'ArrayDataset', 'BooleanParameter', 'Column', 'CompositeDataset', 'DatasetUtil', 'DateParameter', 'DoubleParameter', 'LongParameter', 'Metadata', and 'Product'. Below these tables is a 'Package herchel.ia.dataset Description' section, which includes an 'Introduction' and 'Basic elements' subsection. The 'Introduction' notes that the package is a library and is geared toward usage in both Jython and Java. The 'Basic elements' section states that the package can be split into 'Products' and 'Datasets & Algorithms', with a table listing 'Product' as a collection of datasets, meta-data, and history.

Figure 9-3: Package description page in JavaDocs

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Figure 9-4 is taken from the *Main frame* of the *TableDataset* class and shows the class description together with its hierarchy. You can see that the *TableDataset* implements a number of interfaces and also has one known sub-class i.e. *SpectrumDataset*. The second part of the figure shows a more detailed description of the class usage. This description is provided by the programmer in the source code.

```
herschel.ia.dataset
Class TableDataset

java.lang.Object
├─herschel.ia.dataset.AbstractAnnotatable
│   └─herschel.ia.dataset.TableDataset

All Implemented Interfaces:
  Annotatable, Attributable, Dataset
Direct Known Subclasses:
  SpectrumDataset
```

---

```
public class TableDataset
  extends herschel.ia.dataset.AbstractAnnotatable
  implements Dataset

A TableDataset is a tabular collection of Columns. It is optimized to work on array Data
as specified in the herschel.ia.numeric package.

This approach is convenient in many cases. For example, one has an event list, and each
algorithm is adding a new field to the events (i.e. a new column.

The orthogonal approach (adding rows) is therefore expensive and therefore currently no
mechanism is implemented to add rows to the table.

Jython usage:
creation:

  creation:
  $ x=TableDataset(description="This is my table")
  $ x["Time"]=Column(data=time, quantity=SECONDS)
  $ x["Energy"]=Column(data=energy, quantity=ELECTRON_VOLTS)
```

**Figure 9-4:** The class view of *TableDataset* showing a brief description and a short example of its usage.

Scrolling down in the *Main frame* brings you to the summary section which is shown in the Figure 9-5. The constructor summary shows all public constructors for this class with their specific argument list. To see detailed information on the constructor click the name of the constructor that you need. Constructors are methods that create objects of a particular type. The code example in the description section above shows you how to create a *TableDataset* on the *jython* command line.

Constructor Summary	
<code>TableDataset()</code>	Constructs an empty table.
<code>TableDataset(java.lang.String description)</code>	Constructs a TableDataset with a description.
<code>TableDataset(TableDataset copy)</code>	Constructs a TableDataset that is a deep copy of specified argument.

Method Summary	
<code>column</code>	<code>__getitem__(int index)</code> <b>Jython only(!)</b> wrapper for abbreviated access to a column by index.
<code>column</code>	<code>__getitem__(java.lang.String key)</code> <b>Jython only(!)</b> wrapper for abbreviated access to a column by name.
<code>void</code>	<code>__setitem__(int index, Column value)</code> <b>Jython only(!)</b> wrapper for abbreviated replacement of a column by index.
<code>void</code>	<code>__setitem__(java.lang.String key, Column value)</code> <b>Jython only(!)</b> wrapper for abbreviated addition/replacement of a column by name.
<code>void</code>	<code>accept(DatasetVisitor visitor)</code> Accepts a visitor of this Dataset.
<code>void</code>	<code>add(Column column)</code> <b>Deprecated. and replaced by</b> <code>addColumn(herschel.ia.dataset.Column).</code>
<code>void</code>	<code>add(java.lang.String name, Column column)</code> <b>Deprecated. and replaced by</b> <code>addColumn(herschel.ia.dataset.Column).</code>
<code>void</code>	<code>addColumn(Column column)</code> Adds the specified column to this table, and creates a dummy name for this column, such that it can be accessed by <code>get(int)</code> .
<code>void</code>	<code>addColumn(java.lang.String name, Column column)</code> Adds the specified column to this table, and attaches a name to it.
<code>void</code>	<code>addRow(java.lang.Object[] array)</code> Adds the specified array as a new row to this table.
<code>Dataset</code>	<code>apply(Algorithm algorithm)</code> Applies the specified algorithm on a dataset.
<code>protected java.lang.String</code>	<code>contentsToString()</code>

Figure 9-5: Page showing the constructor mechanism (how to create a TableDataset) and the associated set of methods (what you can do with the TableDataset you created)

The method summary shows all public methods for this class in alphabetical order. For detailed information on a specific method, click its name. In this method summary there are a number of things to note. The return values of the methods are in the left column while the method signature and a summary line is in the right column. The summary line can be preceded with a **deprecation** note. Deprecation means that this method should not



be used anymore because it is marked to be removed from future releases. The deprecation comment normally provides the alternate or new method to be used instead. An overview of all deprecated methods in the whole system is available from the navigation bar at the top of the *Main frame*.

Sometimes method names can start and end with two underscore characters like in '`__getitem__`' above. These methods are special constructs which allow you to use the specific jython syntax to access and manipulate objects from this class.

## 9.5 *Tree view*

There is a Class Hierarchy page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with *java.lang.Object*. The interfaces do not inherit from *java.lang.Object*. When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages. When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

## 9.6 *Deprecated view*

The Deprecated API page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually suggested. **Be warned that deprecated APIs may be removed in future implementations.**

## 9.7 *Index view*

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

## 9.8 *IA Packages And Documentation*

The following short paragraphs outline the packages currently available within the Herschel IA system. A full listing of the classes (programs) available in these packages is given in [Appendix B](#). To go to the on-line JavaDocs (fairly terse documentation on the packages), click on the appropriate heading.

### 9.8.1 *herschel.ia.dataflow*

*herschel.ia.dataflow* – a package for handling processing threads. Particularly useful for Quick Look Analysis (QLA) and Standard Product Generation (SPG). It can be used in interactive sessions too. Allows the user to connect scripts from process modules as is typically required for a set of data reduction steps

**Sub-packages:**

*herschel.ia.dataflow.data.process* ...classes for handling the processes used in a dataflow session.

*herschel.ia.dataflow.example.indicator\_control.monothread* ...classes used to illustrate the control of a dataflow.

*herschel.ia.dataflow.example.indicator\_control.multithread* ...ditto but for multiple threads.

*herschel.ia.dataflow.template* ...class to allow template dataflow to be created.

*herschel.ia.dataflow.util* ...contains a class for identifying dataflows.

### 9.8.2 **herschel.ia.dataset**

*herschel.ia.dataset* – a package for dealing with *TableDataset*, *ArrayDatasets* and *CompositeDatasets*. These datasets contain information to which an algorithm can be applied. The package contains classes that deal with the set and handling of these datasets and also the handling of products (which can contain multiple datasets). An example product may be one that contains several tables plus metadata that describes the table contents which might have similarities to FITS header information.

**Sub-packages:**

*herschel.ia.dataset.demo* – contains classes that demonstrate the use of datasets and construct a user-defined *SpectrumDataset*.

### 9.8.3 **herschel.ia.demo**

*herschel.ia.demo* – package containing classes for use in an IA demo of end-to-end processing. See sub-package *herschel.ia.demo.endtoend* and [demo script](#).

### 9.8.4 **herschel.ia.doc**

*herschel.ia.doc* – currently a place holder for a documentation package.

### 9.8.5 **herschel.ia.help**

This package contains the utilities and classes needed for providing the help facilities in an IA/`Jconsole` session. Access to the on-line help is discussed in Chapter 3 of this manual.

### 9.8.6 **herschel.ia.image**

*herchel.ia.image* – package containing classes for handling images. The Display capabilities from this package were discussed in Chapter 8. The following classes exist in the package.

- *Display* – an image display implementation based on JSKY. User gets 800x600 window for image. Can handle, 1D, 2D and 3D image representations. Allows standard display capabilities such as annotation, rescaling, coordinate display.
- *Histogram* – currently a basic histogram capability. The histogram is based on the values taken from an *imageDataset*.
- *ImageDataset* – a special form of a composite dataset that presents an image. Has layers which are image data, mask data, error data. World Coordinate System (WCS) information is held as metadata in the *ImageDataset*.
- *Layer* – constructs a layer of an *ImageDataset*.
- *Rotate* – allows rotation of an *ImageDataset*. Four different types of interpolation are possible. The WCS coordinates of the image are also rotated with the image.
- *Scale* – allows the scale of an image to be changed. Four different types of interpolation are possible.
- *Translate* – moves an *ImageDataset*. The WCS is also adapted.
- *WCS* – associates a WorldCoordinate System to an *ImageDataset*

**Sub-package:**

*herchel.ia.image.gui* – classes that handle GUIs. These should ONLY be called from within the Display program.

### 9.8.7 herchel.ia.inspector

This package contains the classes and utilities for providing the dataset and session inspectors available in `Jconsole` (see [Chapter 3](#)).

### 9.8.8 herchel.ia.io

*herchel.ia.io* – This is a package that provides a means of accessing local archives where Products can be saved or loaded from. Products are combinations of data and information and can be likened to the contents of a single FITS file.

**Sub-packages:**

*herchel.ia.io.fits* – A FITS implementation that can write Products to a FITS file and read such FITS files back into the system. Allows the production of a FITS archive.

*herchel.ia.io.ascii* – Allows the input/output to and from ASCII files from within the IA environment.

*herchel.ia.io.dbase* – Allows data/products to be put into objects that can be stored in databases (Versant databases are currently available for use with the HCSS). See [Chapter 12](#) for information about the setup and use of databases with IA.

### 9.8.9 `herschel.ia.jconsole`

*herschel.ia.jconsole* – Package containing the classes used in running `jconsole`, a GUI for running/editing of IA/Jython scripts. Allows control of the `jconsole` setup and access to classes that setup the components of the GUI interface (in *herschel.ia.jconsole.gui*).

### 9.8.10 `herschel.ia.numeric`

*herschel.ia.numeric* – This package is discussed in some detail in [Chapter 6](#).

#### **Sub-packages:**

*herschel.ia.numeric.function* – Provides the numeric classes currently available within Herschel IA. These include such functions as FFT, fitter, interpolation and matrix functions.

*herschel.ia.numeric.function.fit* – Provides the classes that allow data fitting of various types.

*herschel.ia.numeric.function.util* – Provides further mathematical functions, including hyperbolic functions and exponentiation.

### 9.8.11 `herschel.ia.plot`

*herschel.ia.plot* – This package provides access to the IA plotting utilities available with IA (callable from `jconsole`). This includes PlotXY and access to plot properties. The use of the plotting capabilities in Herschel IA is discussed in [Chapter 7](#).

### 9.8.12 `herschel.ia.task`

*herschel.ia.task* – This package provides the tools needed to create an IA “task” which a user can then incorporate when constructing his/her own IA software package. This can be used by a user to set up a script which has an associated “signature” (parameter setup). In setting up a task, parameter checks can be performed and a history of the processing can be made.

### 9.8.13 `herschel.ia.ui`

*herschel.ia.ui* – Provides the programs dealing with the GUI interfaces available within Herschel IA. The setup and use of GUIs and incorporating JAVA



## Herschel IA Chapter 10

# 10 Import and Export of Tabular ASCII and FITS Files

## Chapter 10 Contents

- 10.1 [Introduction](#)
  - 10.2 [Getting Started with ASCII Import/Export](#)
  - 10.3 [Basic ASCII Table Import/Export Tool Usage](#)
    - 10.3.1 [Import Parsers](#)
    - 10.3.2 [Comma-Separated-Variable Parser](#)
    - 10.3.3 [Fixed-Width Parser](#)
    - 10.3.4 [Export Formatters](#)
    - 10.3.5 [Comma-Separated-Variable Formatter](#)
    - 10.3.6 [Fixed-Width Formatter](#)
    - 10.3.7 [Table Template](#)
  - 10.4 [Example of How to Import/Export ASCII Tables in IA](#)
  - 10.5 [Overview of FITS IO](#)
  - 10.6 [Getting Started With FITS IO](#)
    - 10.6.1 [Basic FITS IO Tool](#)
    - 10.6.2 [Parameter Name Conversion and FITS Header](#)
    - 10.6.3 [Caveats](#)
- 

### 10.1 Introduction

This chapter describes how to read and write tabular ASCII and FITS data files within IA. Illustrations are provided that run in the `jconsole` environment. **It should be noted that FITS files created outside of the IA environment can NOT currently be imported.**

### 10.2 Getting Started with ASCII Import/Export

Assuming you have successfully started `jconsole`, then the following packages should already be available within the standard IA setup

```
# ascii tools
  herchel.ia.io.ascii
```

```
# table dataset handling
    herschel.ia.dataset
```

### 10.3 Basic ASCII Table Import/Export Tool Usage

The tool to read and write tabular ASCII files is called **AsciiTableTool**. In your `jconsole` session, you may have multiple instances of this tool -each with a different configuration to suit the format of the input/output tables being used.

*In general*

create the ascii tool with default settings  
`ascii=AsciiTableTool()`

import a table from an ascii file. Take the tool and use it to load a table labeled "table.input" in your current directory. It is called "table" within the IA environment and can be viewed with the command "print table" within IA.

```
table=ascii.load("table.input")
```

export a table to an ascii file. Take the IA table (called "table") and using the tool (ascii) apply the method (save) to save into a file called "table.output" in your current directory

```
ascii.save("table.output",table)
```

You can change the behavior of the tool to allow various formatting changes with the following attributes:

<b>parser=yourParser</b>	Changes the line parsing behavior at import.
<b>formatter=yourFormatter</b>	Changes the line formatting behavior at export
<b>template=yourTemplate</b>	Specifies how to interpret raw cell data.

e.g.

```
ascii.parser=CsvParser
```

indicates to use the `CsvParser`

```
ascii.formatter=CsvFormatter(delimiter = `&`)
```

which indicates that we want to use a non-standard delimiter (ampersand rather than a comma).

#### 10.3.1 Import Parsers

*A parser controls how to break-up a line into table cell data. All parsers share the following attributes:*

<b>ignore=expression</b>	Lines containing expression are ignored. By default the expression skips lines starting with a hash, possibly preceded by whitespace: "^\s*#"
<b>skip=value</b>	First number of lines can be skipped by specifying a value>0. Default is 0.
<b>trim=0 1</b>	Whether to strip lines from leading and trailing spaces, default is 0 (false).

*Example:*

```
#skip first 20 lines of the table – read or write.
  ascii.parser.skip=20
#indicate whether to remove leading and trailing blank spaces or not.
  ascii.parser.trim=1
```

### 10.3.2 Comma-Separated-Variable Parser

The Comma(Character)-Separated-Variable Parser named **CsvParser** breaks up a line into cells using a delimiter symbol. The delimiter character can be part of one or more cell-data itself.

In addition to the common attributes of any parser, a CsvParser gives you control over the following extra attributes:

<b>delimiter=character</b>	The character used to distinguish cells within a line of data. Default is a comma (,).
<b>quote=character</b>	The character used if cell-data contains a delimiter character. Default is a double quote (").

*Example:*

use a CSV parser overriding default settings. This example skips 2 lines and makes the delimiter symbol a semi-colon. The \* character is used to indicate cells containing the delimiter symbol.

```
ascii.parser=CsvParser(skip=2,delimiter=';',quote='*')
```

### 10.3.3 Fixed-Width Parser

The **FixedWidthParser** breaks up a line into cells by interpreting every cell to be of a fixed number of characters.

In addition to the common attributes of any parser, a FixedWidthParser gives you control over the following extra attributes:

<b>sizes=array</b>	An array $n$ elements, where $n$ is the number of columns, and each element specifies the width of that cell.
--------------------	---------------------------------------------------------------------------------------------------------------

*Example:*

```
# use a FixedWidth parser that expects 3 columns in the table with widths
#10, 20 and 10 characters respectively – and in that order.
  ascii.parser=FixedWidthParser(sizes=[10,20,10])
```

### 10.3.4 Export Formatters

A formatter controls how to format a row of cells into a line of ascii. All formatters share the following attributes:

<b>commented=0 1</b>	States whether comments will be allowed in the output or not, default=0 (false).
<b>commentPrefix=string</b>	Prefix used for all comments, default="#".
<b>header=0 1</b>	Whether to precede the actual data with header information, default is 0 (false). This header may contain name, type, units and description of each column

*Example:*

```
#First indicate that a header is to be added to the output table
  ascii.formatter.header=1
#Indicate that comments will be allowed in the output
  ascii.formatter.commented=1
#Indicate how comments are prefixed in the table
  ascii.formatter.commentPrefix="$$$ "
```

### 10.3.5 Comma-Separated-Variable Formatter

Please read its counterpart **CsvParser** (section 10.3.2) for parameters and defaults.

```
#The default comma(character) separated variable formatter has a ',' delimiter
#and a '#' quote.
  formatter=CsvFormatter()
#The delimiter and quote can be changed – the & symbol is useful for
#creating latex tables
  formatter=CsvFormatter(delimiter='&', quote='<')
```

### 10.3.6 Fixed-Width Formatter



Please read its counterpart `FixedWidthParser` for parameters and defaults.

```
#Take default width for table cells
formatter=FixedWidthFormatter()
#Set the width of 3 columns of cells to specific sizes
formatter=FixedWidthFormatter(sizes=[5,12,3])
```

### 10.3.7 Table Template

Many tabular ascii files contain only raw data. Though the human eye may interpret cell-data being a string or a rational number, the computer needs some more information. The **TableTemplate** allows you to specify such information. The only mandatory argument for a table template is the number of columns that are expected. Its optional attributes are:

<b>names=array</b>	Specifies names that will be attached to the columns.
<b>types=array</b>	Specifies the types of all columns. If not specified, the template assumes that all columns are of type String. Allowed types are: "Boolean", "Integer", "Float", "Double" and "String".
<b>units=array</b>	Specifies the units of all columns. Uses SI units, and units that are accepted for use with SI.
<b>descriptions=array</b>	Specifies comments for all columns.

*Example:*

```
#The following table template indicates a table with 4 columns with associated names
#character/number types and associated units
ascii.template=TableTemplate(4, \
    names=["Frame", "Energy", "Foo", "Bar"], \
    types=["Integer", "Double", "Double", "Double"], \
    units=["s", "eV", "N m -1", "kg L-1"])
```

## 10.4 Example of How to Import/Export ASCII Tables in IA

Section 10.3 introduced the various import and export capabilities of the **AsciiTableTool**. We can put these together to illustrate how a user can import and export Ascii tables of virtually any type. [Example 10.1](#) provides an illustration of how to handle ASCII tables in the IA environment. A number of ASCII tables are created and reimported. These can be viewed by opening them with the `Jconsole` window (or within any other text editor). In order to run the program the user will also require an input file, which is given below and can be downloaded from [here](#).

```
#file "ascii_demo_data.txt" for use in example 10.1
```

```
# sample file, using default settings of AsciiTable object
# table=AsciiTable().load("table-default.txt")
Frame,Counts,Valid,Comments
Integer,Double,Boolean,String
s,eV,,
'''
1,1.0,true,
2,5.0,true,
3,0.0,false,incomplete data
4,0.0,false,missing data
5,1.234567E-8,true,

# Example 10.1 - Handling ASCII tables

# --- import a table that complies to default settings
ascii=AsciiTableTool()
table=ascii.load("ascii_demo_data.txt")

# --- export a table using defaults settings:
ascii.save("table.out1",table)

# --- export using Fixed Width format, with header info:
ascii.formatter=FixedWidthFormatter(sizes=[8,16,8,30])
ascii.save("table.out2",table)

# --- importing it back requires Fixed Width parser
ascii.parser=FixedWidthParser(sizes=[8,16,8,30])
table=ascii.load("table.out2")

# --- export using Fixed Width format, only raw data:
ascii.formatter.header=0
ascii.save("table.out3",table)

# --- importing a raw "fixed width" table that has only data. So we
have to
#   define the template ourselves:
ascii.template=TableTemplate(4,names=["Frame","Counts","Valid",\
    "Comments"], types=["Integer","Double","Boolean","String"])

table=ascii.load("table.out3")

# --- saving current state of AsciiTableTool:
ascii.save("table.template")

# --- quick save table with default settings, equivalent to
#"table.out1":
AsciiTableTool().save("table.out4",table)

# -- reloading state:
mine=AsciiTableTool("table.template")
table=mine.load("table.out3")
mine.save("table.out5",table)

# --- saving with comments
table.description="Sample description can be found here"
mine.formatter.header=1
```

```
mine.formatter.commented=1  
mine.formatter.commentPrefix="; "  
mine.save("table.out6",table)
```

## 10.5 Overview of FITS IO

In the next few sections we describe how to write and read [Products](#) (which contains one or more datasets, a history of how it was created and meta-data describing the contents – the latter two are typical FITS header components) to and from FITS files within the IA environment.

FITS stands for Flexible Image Transport System, a format adopted by the astronomical community for data interchange and archival storage.

## 10.6 Getting Started With FITS IO

Assuming you have successfully started JIDE, the facilities needed to create products as well as to create FITS files should already be available in your session.

### 10.6.1 Basic FITS IO Tool

The tool to write and read Products to and from FITS files is **FitsArchive**. In your `jconsole` session, you may have multiple instances of this tool -each with a different configuration.

In general, we can set up a FITS file for archiving, export IA products to it and retrieve back a product from a FITS file.

*For example:*

```
# create the FITS Archive with default settings
```

```
fits=FitsArchive()
```

```
# export a product to a FITS file
```

```
fits.save("product.fits",product)
```

```
# import products from a FITS file
```

```
product=fits.load("product.fits")
```

[NOTE: At present, the program is unable to load FITS files that were not created by the Herschel IA system]

## 10.6.2 Parameter Name Conversion and FITS Header

The current implementation of the FITS archive converts long, mixed-case parameter name, defined in the meta data of your product, into a FITS compliant notation. The latter dictates that parameter names must be uppercase, with a maximum length of eight characters. Clearly, we do not want to force all our parameters to have names that fit within such a FITS specific restriction.

The FITS Archive uses lookup dictionaries that convert well known FITS parameter names into a convenient and human readable name. Currently the following dictionaries are in use:

[Common keywords](#) widely used within the astronomical community. Taken from [HEASARC](#),

[Standard](#) FITS keywords, and

[HCSS keywords](#) containing keywords that are not defined in the above dictionaries.

For example the following Meta data is transformed into a known FITS keyword:

### JCONSOLE

```
product.meta["softwareTaskName"]=StringParameter("FooBar")
```

### 10.6.2.1 FITS product header

```
HIERARCH key.PROGRAM='softwareTaskName'  
PROGRAM = 'FooBar '
```

#### *Example*

A full demonstration of FITS IO is available in [example 10.2](#). The script creates a product with several (nested) datasets, stores it into a FITS file, and then retrieves it again.

```
#Example 10.2: FITS IO from within Herschel IA  
# first we will get some unit definitions for our example  
from nT.quantity.constant.ENERGY import ELECTRON_VOLTS  
from nT.quantity.constant.TEMPERATURE import KELVINS  
from java.lang.Math import PI  
  
# --- construction of a product. note this is only for demonstration  
# purposes. For more information, please see the dataset and numeric  
# chapters of the manual (chapters 5 and 6).  
points=50  
x=DoubleId.range(points)  
x*=2*PI/points  
#Create an array dataset that will eventually be exported  
s=ArrayDataset(data=x,description="range of real\  
values",quantity=ELECTRON_VOLTS)
```

```
#provide some metadata for it (header information)
s.meta["temperature"]=LongParameter(long=293,\
description="room temperature",quantity=KELVINS)

#create a tabledataset for export
t=TableDataset(description="This is a table")
t["x"]=Column(x)
t["sin"]=Column(data=SIN(x),description="sin(x)")

c=CompositeDataset(description="Composite with three datasets!")

c.meta["exposeTime"]=DoubleParameter(double=10,description="duration")
c["childArray"]=s
c["childTable"]=t
c["childNest"]=CompositeDataset("Empty child, just to prove nesting")

p=Product(description="FITS demonstration",creator="demo.py")
p.creator="You?"
p.modelName="demonstration"
p.meta["sampleKeyword"]=\
StringParameter("Example keyword not in FITS dictionaries")

p.meta["observationInstrumentMode"]=StringParameter("UnitTest")
p["myArray"]=s
p["myTable"]=t
p["myNest"]=c

# --- demonstration of the FITS archive
fits=FitsArchive()
# save it ...
fits.save("demo.fits",p)
# ... load it back into a new variable...
n=fits.load("demo.fits")
# ... and show it!
print n
print n["myArray"]
print n["myNest"]
print n["myNest"]["childNest"]
```

### 10.6.3 Caveats

**The current implementation can not yet read fits files that are generated by any other package than *herchel.ia.io.fits* within the Herschel IA system. This is expected to change so that FITS files can be imported from and exported to other software packages.**

A FITS header card is limited to 80 characters. Within those limitations the FitsArchive will try to store the abbreviated FITS keyword, parameter value, and in the comment area optionally a quantity and description. The latter two might be truncated due to these limitations. Also a StringParameter with a long value can be truncated.

For more information see the [FITS IO](#) general documentation



# Herschel IA Chapter 11

## 11 Using Time in the IA Environment

### Chapter 11 Contents

- 11.1 [Introduction](#)
- 11.2 [Time Definitions](#)
  - 11.2.1 [System time in IA \(FineTime\)](#)
  - 11.2.2 [International Atomic Time \(TAI\) and FineTime](#)
  - 11.2.3 [Coordinated Universal Time \(UTC\)](#)
  - 11.2.4 [DecMec Time](#)
- 11.3 [Time in HK Data](#)
- 11.4 [Time conversion](#)
  - 11.4.1 [Time conversion in HCSS](#)
  - 11.4.2 [CucConverter](#)

---

### 11.1 Introduction

This note describes which and how Time is defined within HCSS and how to deal with it. Unfortunately, there are several ways in which time can be represented. The standard for the HCSS/IA is a *FineTime* – which is the number of microseconds since the beginning of 1 January 1958. This provides the kind of accuracy needed to represent time on a space mission.

However, there are several other time representations and it is often the case that conversions between times/dates is necessary. This chapter indicates how to deal with times within IA.

### 11.2 Time Definitions

#### 11.2.1 System time in IA (*FineTime*)

There are many ways to access the system time within IA. See also the description of the class `Date` for a discussion of slight discrepancies that may arise between "computer time" and coordinated universal time (UTC).

Two possibilities are:

```
# To get the current time in milliseconds :
```

```
# The difference, measured in milliseconds, between the current
```

```
# time and midnight, January 1, 1970 UTC.

print java.lang.System.currentTimeMillis()

# To get the number of milliseconds since
# January 1, 1970, 00:00:00 GMT represented by a Date object.

d = java.util.Date()
#printing this gives the current time and date at the location of the
#system on which the java is being run.
print d

#We can also get the number of milliseconds since Jan 1, 1970 using
#this Java Date
print d.getTime()
```

Note that while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger.

If we want to get the number of milliseconds since 1 January 1970 for any other date then we can use the java Date capability.

```
#Format of date is year (in units of true year - 1900), month (number
0...11), day, hour, minute, second
#So the following gives us the number of milliseconds between the
beginning of 1 January 1970
# and 3:15:00 pm on 23 October 2004.

d = java.util.Date(104, 9, 23, 15, 15, 0)
print d # should indeed show we have 3:15pm on 23 October 2004
print d.getTime() # provides the number of milliseconds between this
#date and 1 Jan 1970.
```

### 11.2.2 International Atomic Time (TAI) and *FineTime*

**TAI** is an international standard measurement of time based on the comparison of many atomic clocks. TAI is the basis for Coordinated Universal Time (UTC). *Finetime* is based on TAI as measured from 00:00:00 1 January 1958.

### 11.2.3 Coordinated Universal Time (UTC)

**UTC**, World Time, is the standard time common to every place in the world. UTC is derived from International Atomic Time (TAI) by the addition of a whole number of "leap seconds" to synchronise it with Universal Time 1 (UT1), thus allowing for the eccentricity of the Earth's orbit, the rotational axis tilt (23.5 degrees), but still showing the Earth's irregular rotation, on which UT1 is based.

#### 11.2.4 DecMec Time

The commands `DPUSelectTime` and `DPUWriteTime` are selecting and setting a start time which is written to the `TMP1` and `TMP2` fields of the Dec/Mec headers. It is possible to construct an absolute time by adding counters (CRDC) to the start time considering an offset between setting and writing the start time.

This offset is expected to be a number with an uncertainty depending on the system load. It might require a calibration file. Currently this offset is not considered.

In case the commands are not given the `TMP1` and `TMP2` fields are zero. To avoid software confusions the time will be related to a fixed date (1.Jan 1970, 0:00).

During construction of the `SpuBuffer` the time is computed from the `TMP1`, `TMP2` entries in the Dec/Mec header and the CRDC counter. This time is used during construction of the `DataFrameSequence` and the associated Tables holding the spu science data.

Between the Dec/Mec time and the packet time (see `PusTmBinStruct`) we have an offset. Therefore the association between HK and science data will be within an accuracy of 2 seconds.

### 11.3 Time in HK Data

The most convenient method of obtaining timestamped HK information is the use of the “`herschel.binstruct`” package. The use of this is illustrated in Chapter 12.10 where HK data is obtained from a database and then read/converted for use within the IA environment.

When dealing with HK time information directly, it is important to know that telemetry packets contain the time as defined within the “PUS Data Field Header”. The field represents the on-board reference time of the packet, referenced to TAI, expressed in spacecraft time units – CCSDS Unsegmented Time Code (CUC) units. CUC units are multiples of 1/65536 sec from 1 January 1958 in TAI time. CUC units cannot be expressed in whole microseconds but can be converted to the *FineTime* standard (see below).

CUC time is written for HK by the data processing unit (DPU).

Current `PusTmBinStruct` methods related to time :

**long getTime()**

Returns the packet time of the Pus telemetry packet.

**boolean isTimeSynchronized()**

Returns true if the telemetry packet is synchronized, false otherwise.



**java.util.Date getTimeAsDate()**

Returns the packet time as a Date object.

**FineTime getTimeAsFineTime()**

Returns the packet time of the Pus telemetry packet as FineTime.

## **11.4 Time conversion**

### **11.4.1 Time conversion in HCSS**

It can often be the case that we need to convert between FineTime (TAI) and Date (UTC). Coordinated Universal Time is expressed using a 24-hour clock and uses the Gregorian calendar. FineTime represents a TAI time (epoch 1958), whereas the Java Date class is used to represent UTC, by resetting the system clock whenever a leap second occurs and don't need to handle leap seconds. In order to convert between Java dates and the FineTime standard requires the use of the DateConverter() class. Long integers can also be directly converted to FineTimes and are interpreted as representing the number of microseconds since 00:00:00 1 January 1958. [Example 11.1](#) illustrates how we can create a FineTime from a long integer and convert back and forth between FineTime and Java Dates.

```
#Example 11.1 - time conversion of Date to FineTime (and back)
from herschel.ccm.util import *
from herschel.share.fltdyn.time import *

#FineTime to Date
#Enter a time in seconds (a long integer - put letter "l"
#at the end of the number)
c = FineTime(14360944497154001)#convert to a FineTime
#Prints corresponding date and time
print DateConverter.fineTimeToDate(c)

#Date to a FineTime
d = java.util.Date() #gets today's date and time
#prints corresponding FineTime
print DateConverter.dateToFineTime(d)
```

### **11.4.2 CucConverter**

Converts between Spacecraft Elapsed Time, in CCSDS Unsegmented Time Code (CUC) format and FineTime (TAI). This implementation is for the Herschel CUC format, which is corrected on-board the spacecraft to TAI (epoch 1 Jan 1958). This representation uses 32-bits for seconds and 16 bits for fractional seconds. CUC times are multiples of 1/65536 sec and cannot be expressed as an exact multiple of 1 microsecond (the resolution of FineTime). However, the following relations hold for 'coarse' and 'fine' values in the allowed range:

**long coarse(FineTime t)**

Return the number of whole seconds since the epoch 1 Jan 1958.

**long cucValue(FineTime t)**

Return the number of 1/65536 fractional seconds since the epoch 1 Jan 1958.

**int fine(FineTime t)**

Return the fractional part of the number of 1/65536 seconds since the epoch 1 Jan 1958.

**FineTime toFineTime(long cuc)**

Return a new FineTime constructed from a 48-bit CUC time.

**FineTime toFineTime(long coarse, int fine)**

Return a new FineTime constructed from CUC coarse & fine fields.

For example:

```
d=CucConverter.toFineTime(5000000000000001)
# Converts the long integer - representative of a CUC time -
#into a FineTime. The FineTime is stored in d.
e = CucConverter.coarse(d)
#provides the number of whole seconds since 1 Jan 1958
#and stores it in e.
```



## Herschel IA Chapter 12

# 12 Setup and Use of Databases

### Chapter 12 Contents

- 12.1 [Introduction](#)
  - 12.2 [Starting Up A Database:](#)
    - 12.2.1 [Unix](#)
    - 12.2.2 [Windows](#)
  - 12.3 [Schema Initialization](#)
  - 12.4 [Using an existing database and Schema Evolution](#)
    - 12.4.1 [Initializing a schema on an old database](#)
    - 12.4.2 [Schema Tool commands](#)
  - 12.5 [Initializing a Database For IA Use](#)
  - 12.6 [Quick Database Creation](#)
  - 12.7 [Providing Database Access for an IA Session](#)
    - 12.7.1 [Properties File Setup for Database Access](#)
    - 12.7.2 [Using the Propgen Tool](#)
  - 12.8 [Browsing a Database](#)
  - 12.9 [Getting Data Frames From a Database](#)
    - 12.9.1 [Command Line Access to Data Frames](#)
    - 12.9.2 [From Database to ASCII File](#)
    - 12.9.3 [Downloading Dataframes from a Database Using a GUI](#)
  - 12.10 [Accessing Housekeeping \(HK\) Data](#)
    - 12.10.1 [Accessing HK Information For a Given Obsid](#)
    - 12.10.2 [Accessing HK Data For a Given Time Period](#)
  - 12.11 [Removing a Database](#)
- 

### **12.1 Introduction**

If you want to work with databases, which is one of the main ways in which test and (later) observational data are to be stored within the HCSS, then you will need to have a Versant Database System available to you. For most large sites your system manager will have installed a Versant license which allows the setup and use of databases at your home institution. You can also install a database capability on your own computer/laptop. Unix and Windows versions are available.

Most users will not need to set up a database but rather just access for reading stored data. In this case, sections 12.2 and 12.3 may be skipped.

**Note for Versant in general:** Versant is commercial software and procurement has been done centrally for Herschel, please contact the Herschel software administrator at your institute for more details on how to proceed.

Alternatively you can contact the following people:

HIFI:

- [Albrecht de Jonge](#)
- [Peer Zaal](#)

PACS:

- [Ekkehard Wieprecht for PACS/MPE](#)
- [Wim de Meester for PACS/KUL](#)

SPIRE:

- [Steve Guest](#)

Some notes on Versant database setup are available in [Chapter 2.4](#). For further information please also consult the [Known issues with Versant Databases](#) document.

## **12.2 Starting Up A Database:**

### **12.2.1 Unix**

The following commands create a database within the HCSS and make it available for use. They should be executed at a terminal prompt.

```
>> makedb <dbname>           #initializes directory and log files  
>> createdb <dbname>         # creates db itself
```

Database names should be given in the format  
[tony\\_hcss@lin-sron-02.sron.rug.nl](mailto:tony_hcss@lin-sron-02.sron.rug.nl).

The database now being used should be in the properties file (use “propgen” to check this out – Just put “propgen” on the command line and hit the “General” tab at the top. The database currently in use is on the second line down. Change if needed).  
Now we can fill the database.

### **12.2.2 Windows**

The Unix setup will also work under windows, once the Versant database software has been installed. Alternately, a database can be created using a “wizard”.

- Go to  
"Start->Programs->Versant Develop Suite->Administration Console"  
and then click  
"File->Create Database".

In the wizard that comes up on the screen, specify *dbname* and *server* where *dbname* and *server* are the name of the database and the database server, which must match those specified in your properties if the database is to be used.

### 12.3 Schema Initialization

The database you have created will need a schema associated with it. On Unix or Windows machines the same command can be used.

- `schema_tool -ni dbname@dbserver # Dummy run without making changes (optional)`
- `schema_tool -i dbname@dbserver # Initialize schema and set schema version`

This has two effects: firstly, it creates schema definitions for all persistent classes and, secondly, it places a schema version object in the database, so that its schema version can easily be identified. This allows proper schema evolution if and when the structure of HCSS databases change in the future.

If you use multiple databases, then each database must be initialized in this way.

It is essential to apply a schema to a new database. This indicates how the component layout of a database. This is achieved simply by running the command

```
schema_tool -i (see below).
```

### 12.4 Using an existing database and Schema Evolution

On occasions, new database formats need to be created for the HCSS. In such cases, it is necessary to perform a *schema evolution* on old databases to update for use in current IA environment. For development purposes, it may of course be acceptable to simply create a new database, if there is no data to be preserved.

Schema Evolution is supported for databases created by versions of the HCSS back to HCSS-v0.1.3 (build number 168) although, in principle, it should be possible to go back to HCSS build number 162.

Schema evolution is necessary when a new version of the HCSS is installed that has a higher schema version than the database. The schema version of the CCM can be found by examining the file '%HCSS\_DIR%/doc/SCHEMA\_VERSION' in the HCSS distribution and is displayed against releases in the HCSS download web page. The schema version of the database and the currently installed CCM can be examined by using the '-v' option of the schema tool.

The procedure when installing a new HCSS release with an existing database is as follows:

Install the new HCSS release, then check whether the schema are compatible as follows:

```
>> schema_tool -v dbname # Check the schema versions
```

If the class (CCM) schema version is the same as the database schema version, no schema evolution is needed.

**a.** On a Unix system, to update the schema by any number of schema versions, evolve the database as follows:

Stop all applications accessing the database

Put the database in single user mode using 'dbinfo -1'

#### **Backup the database**

Check the firewall allows the schema evolver to open a ftp connection to <ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/schemaToolData/>.

```
>> schema_evolver dbname@server # Evolve the database
```

Put the database back in multi-user mode using 'dbinfo -m'

**b.** When using Windows, or if the class(CCM) schema version is only one greater than the database schema version, evolve the database as follows:

Stop all applications accessing the database

Put the database in single user mode using 'dbinfo -1'

#### **Backup the database**

```
>> schema_tool -en dbname@server
```

```
# Dummy run without making changes (optional)

>> schema_tool -e dbname@server

# Evolves the database
```

You can put the database back in multi-user mode using 'dbinfo -m'

If the HCSS installation has more than one database, each database must be evolved in this way to the same schema version.

#### 12.4.1 Initializing a schema on an old database

Databases that were created with old versions of the HCSS (build numbers 168 to 240) should be initialized to schema version 1 as follows:

```
Install HCSS build number 241
```

Use the terminal command:

```
>> schema_tool -i dbname@server
```

If the HCSS installation has more than one database, each database must be initialized.

Further schema evolution can then occur through installation of old HCSS builds that progressively had higher schema versions.

#### 12.4.2 Schema Tool commands

The schema tool can be invoked using one of the following commands:

```
>> schema_tool [options] database

>> schema_evolver database
```

For more information about the operation of the schema tool, see [schema tool user manual](#).

The schema tool should only be used after having downloaded and installed a new HCSS build that has a different schema version than the HCSS build used until now

**Note:** changes have been made to the HCSS installation procedure to include *schema evolution* for databases. The 'schema\_tool', allows you to evolve the schema of an existing HCSS database to the current HCSS *schema version*:

## 12.5 *Initializing a Database For IA Use*

In order for a new database to be used, the database itself needs to be initialized. This can be done with the same command in either Windows or Unix. At a terminal prompt, go to the directory where the new database is stored and then input the command

```
>> initv <dbname>
```

## 12.6 *Quick Database Creation*

Based on the chapter information above, the following four lines placed at a terminal prompt will create a new database in most circumstances.

```
>> makedb <dbname>           #initializes directory and log files
>> createdb <dbname>         # creates db itself
>> schema_tool -i <dbname>   #maps schema so that it can be adapted
                             #(if necessary) at a later date.
>> initv <dbname>           # initializes the database ready for use
```

where <dbname> has the format *name@server*.

## 12.7 *Providing Database Access for an IA Session*

Database access can be changed during an IA session without the need to exit `jconsole`. After editing properties or saving changes made using the properties tool (`propgen`), the user can use the new settings immediately within an ongoing IA session.

There are two methods for changing properties to allow database access.

### 12.7.1 *Properties File Setup for Database Access*

There are two ways of setting up your properties to allow access to a particular database during an IA session. First, the file *hcsc.props* (on Windows) or the file *myconfig* (on Unix) can be edited.

**On Windows**, the *hcsc.props* file is usually in the top directory of the user (e.g., `C:\Documents and Settings\<username>`).

**On Unix**, the *myconfig* file is in the directory `~<username>/hcsc`.

The following three lines should be placed in the file being edited if they are not already there.

```
var.database.server = servername
```



```
var.database.devel = dbname@${var.database.server}
hcss.access.database = dbname@${var.database.server}
```

where *servername* is the name of the server where the database is located (e.g. lin-sron-02.sron.rug.nl) and *dbname* is the name of the given database to be used in the IA session.

## 12.7.2 Using the Propgen Tool

Alternately, the `propgen` tool can be used to indicate the server and database to be used. The `propgen` tool can be started from a terminal prompt assuming the HCSS system has been installed and it has been setup to run on the system.

At a terminal prompt, the command

```
> propgen
```

will start up the `propgen` tool (see Figure 12-1). Using the tabs at the top of the `propgen` screen, the user should click on the tab marked “**General**”.

Now edit the variables **var.database.server** (input *servername*) and **var.database.devel** (input `dbname@${var.database.server}`) at the top of the tabbed page.

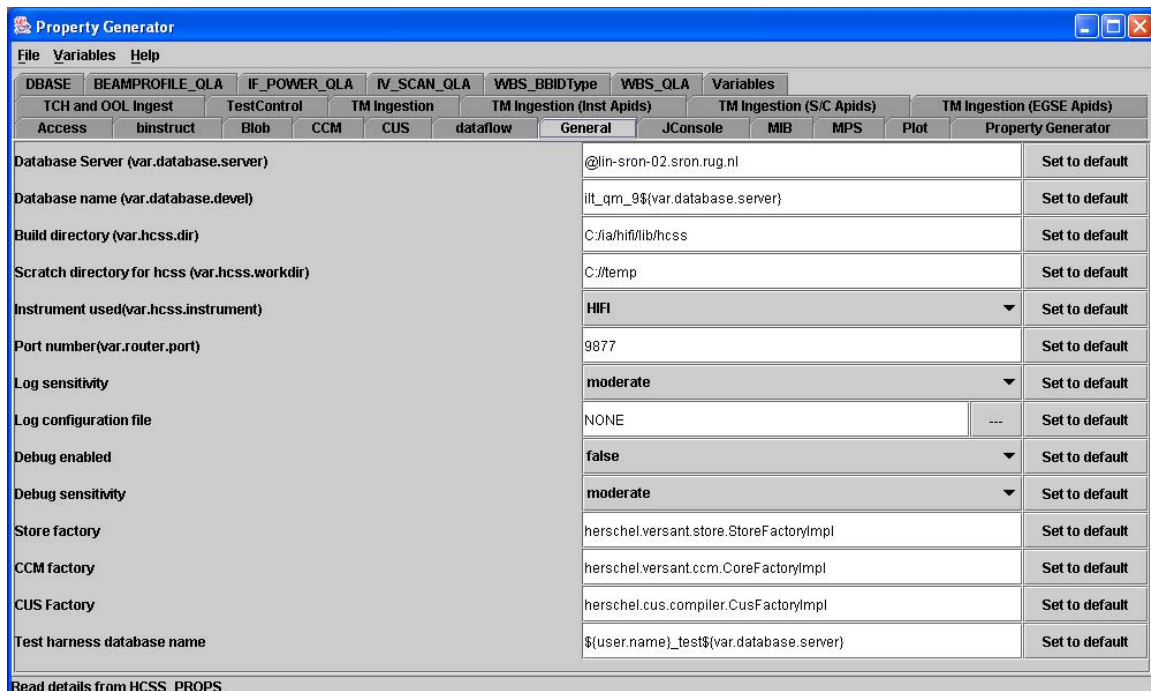


Figure 12-1: The `propgen` window. Most properties that can be changed using the property generator are created automatically in the IA environment. The first screen “**General**” allows the identification of a default database and server name to be set up.

## 12.8 *Browsing a Database*

In order to know what you might want from out of a database, you need to be able to browse through the database contents. The *TestExecutionBrowser* program allows the user to do just that. Input of the following short [example 12.1](#) allows the user to view the database connected to. The first two lines import the packages developed in IA that allow database access. These need to be imported into our session before using the browser to display the contents of the database.

```
#Example 12.1
from herschel.access import *
from herschel.access.util import *

# Used in this mode, the browser is not set up to allow mouse selection
ObservationChooser.showTestBrowser (TelemetryAccess ("TmSourcePacket") )
```

Successful execution of this command will bring up a separate window displaying information on the data contained in the database. This includes information on the script used to create data, the observation ID (scroll to the far right of the window) and the time (local) for when the data was placed in the database.

At a future date, a filtered display of a database is expected to be possible (see bottom left of current *TestExecutionBrowser* window). It is also expected that selection and download following a mouse click will become available.

## 12.9 *Getting Data Frames From a Database*

Once connected to a database and knowing the date or observation id of your data (see previous section), we can retrieve both data frames and housekeeping data from the database. In this section we discuss the basic means for obtaining dataframes from a database. Here we are handling RAW dataframes for which there is no directly associated meta-data, although housekeeping data for is available from the period of time during which the data was taken (see next section).

There are two main methods for obtaining dataframes.

- Command line access
- Through a DataSelector GUI

When accessing dataframes it is particularly useful to use the navigation property available in IA. This speeds up the accessing of dataframes in a database. In order to do this, start the progen tool (see Section 12.7.2) and then go to the “Access” tab. Near the bottom of the window, change navigation value from false to true (click on the cell containing the word “false” and pulldown to “true”).

### 12.9.1 Command Line Access to Data Frames

The basic idea for command line access is to

- Create a means to access data frames
- Indicate which data you want to get (e.g., by observation identification, obsid)
- Go search for it in the database
- Actually get the frames and put them into an array (or table).

[Example 12.2](#) illustrates how the above is done within an IA environment. In this example, an observation made up of several frames is placed in a table with each column of the table being a single 1D spectrum. Something similar could also be set up for multi-dimensional data. In these cases, each “column” of a table would have an N-dimensional object.

```
#Example 12.2 - basic command line method for getting
#data frames from a database
from herchel.access.util import *
from herchel.access import *

#create a tabledataset for the data frames to go into
table=TableDataset()
#start means by which we will access the dataframes
#in the database
dfaccess=DataFrameAccess()
#provide an id for the frames we are looking for
#in this case the observation has an identification number of 1400
dfaccess.setObsid(1400)
#find the data in the database (query).
#This just provides a set of references
#to where data frames fitting the criteria reside in the database
data=HcssConnection.get(dfaccess)
#if there is something found, length of the references > 0
if len(data) > 0:
#then loop around and get all the frames associated with the obsid
    for j in range(len(data)):
        df = data[j]
#now actually get each frames and put them in a real 1D array
        datad = Double1d(df.getFrame())
#and we make each frame into a column in a table
#so that table[0] is the first column and contains
#the first 1D spectrum of the observation, value for each channel
#The column label is the string value of j, i.e., 0, 1, 2, 3...
        table[str(j)]=Column(datad)
```

This brings in a set of spectra as a table. To see what is in the table we can

```
#get general overview
print table
#see what is in the first column
print table[0]
```

```
#see just the data for the first column. No quantities, column
headings etc.
print table[0].data
```

A plot of `table[0].data` will show a channel versus value 1D spectrum.

### 12.9.2 From Database to ASCII File

Following on from the previous section. If we want to have the spectra be placed in an ASCII table output file, then we can add the following code to example 12.2.

```
#set up an output table
mine=AsciiTableTool()
#add a description to our table
table.description="Sample WBS spectra"
#make sure there is a header on the output - see AsciiTableTool help
mine.formatter.header=1
#make sure that comments are allowed
mine.formatter.commented=1
#indicate the prefix for comments in the file
mine.formatter.commentPrefix="; "
#provide a name for the ascii output file and save the data
mine.save("sample_wbs_spectra`,table)
```

Being a little more sophisticated, we can add in a prompt and also iterate around to obtain several observations from a database and place them in ASCII files. [Example 12.3](#) provides a basic Java Swing component to prompt the user for a starting and ending obsid. The data is then passed onto appropriately named ASCII table files.

```
#Example 12.3 - database to ASCII tables for multiple spectra
#import Java swing for GUI components
import javax.swing as sshwing
from herschel.access.util import *
from herschel.access import *

# The data will be placed in comma-delimited tables.
#prompt the user for first obsid using a JAVA Swing component
input_obsid = sshwing.JOptionPane.showInputDialog\
    ("Enter first obsid in list: ")
start_obsid = int(input_obsid)
#prompt again for last obsid
input_obsid = sshwing.JOptionPane.showInputDialog\
    ("Enter last obsid in list: ")
end_obsid = int(input_obsid)

for i in range(start_obsid,end_obsid+1):
    table=TableDataset()
    dfaccess=DataFrameAccess()
    dfaccess.setObsid(i)
    data=HcssConnection.get(dfaccess)
    if len(data) > 0:
        for j in range(len(data)):
```

```
df = data[j]
datad = DoubleIcd(df.getFrame())
table[str(j)] = Column(datad)
mine = AsciiTableTool()
table.description = "Sample WBS spectra"
mine.formatter.header = 1
mine.formatter.commented = 1
mine.formatter.commentPrefix = "; "
mine.save("wbs_spectra_" + str(i), table)
```

The inner loop in the example 12.3 allows us to get each frame in an observation in turn and place it into a table “column”. The outer loop takes the tables formed for each observation id and places them in an ASCII file called “wbs\_spectra\_<obsid>.txt”. These are comma-delimited ASCII tables viewable in any text editor.


### 12.9.3 Downloading Dataframes from a Database Using a GUI

A somewhat more sophisticated method of accessing a database from within an IA session involves the use of a GUI interface such as a *DataSelector* tool. This is available via the *ProcessConnect* command. [Example 12.4](#) provides a downloadable script that uses just such an interface for obtaining HIFI dataframes.

```
#Example 12.4 - GUI interface to a database
from herschel.hifi.generic import *
from herschel.ccm.api import *
import java.lang.reflect
import javax.swing as swing

#the following defines a class we can then run in an IA session
class Hifids:
    def __init__(self):
#Connect the processor so that we get data output to 'a'.
        self.pc = ProcessConnect("pc")
#Now set up place for output of dataframe
        self.out = self.pc.getConnector("df-output")
#create an array which will hold HIFI data frames - up to 1000 of them
        self.a = java.lang.reflect.Array.newInstance(HifiDataFrame,1000)
#provide passage for the dataframes into 'a'.
        self.out.pass(self.a)
#now setup a user GUI for the process connector and a completion button
        self.win = swing.JFrame()
        self.win.contentPane.layout=java.awt.FlowLayout()
        self.win.contentPane.add(self.pc.getJComponent())
        choose = swing.JButton("Finished", size=(65,70), \
            actionPerformed=self.dataChoice)
        self.win.contentPane.add(choose)
        self.win.pack()
        self.win.show()
    def dataChoice(self, event):
#This subroutine creates a table when the GUI's "Finished"
#button is clicked.
        table=TableDataset()
        table.description=("Data output")
```

```
#allow the table (output) to be seen within the session,  
#not just the class.  
    global table  
    for j in range(1000):  
        if (self.a[j] != None):  
            datad = Double1d(self.a[j].getFrame())  
            table[str(j)] = Column(datad)  
#...and gets rid of the pop-up window to finish.  
    self.win.dispose()
```

To use the program, download it into your `jconsole` session and hit the  button. Now, whenever you want to run the program during the rest of your IA session, type the following (e.g., at the `IA>>` prompt)

```
Hifids()
```

This brings up a window similar to that shown in Figure 12-2 – showing the “play” tab screen. You can browse the database with the button (bottom left), choose between dataframes or source packets (**the example script handles HIFI dataframes only for now**) under the “data” tab and get the data under the “play” tab. Dataframes associated with particular APID, building block ID or observation ID can be chosen (see Figure 12-3). A timeframe can also be indicated.

Once the dataframes have been identified, they can be obtained by hitting the play button under the “play” tab. This is the single arrowed button to the left. The buttons on under this tab have similar functions to those on a DVD player! Once play is complete, hitting the Finished button exits the GUI and places the dataframes in a table available to the IA session of the user.



Figure 12-2: The play tab opened for the `dataselector` tool

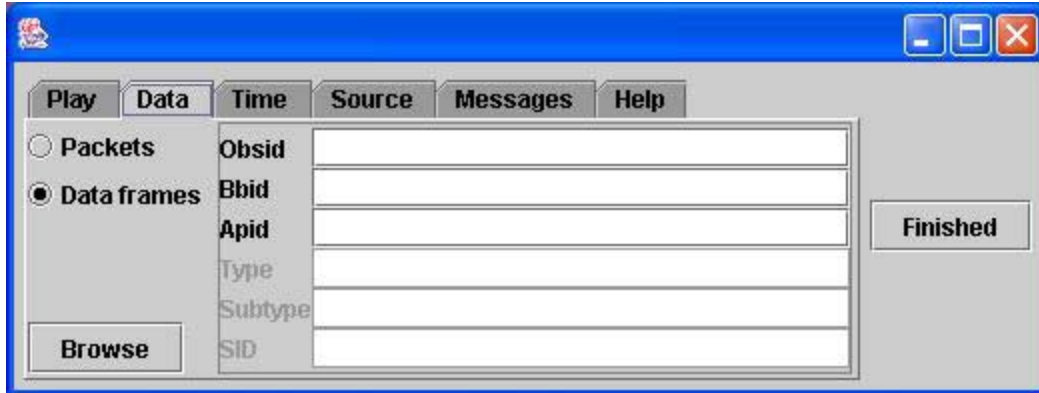


Figure 12-3: The Data tab of the dataselector tool. The example program works with Data frames only, at present.

Output for this program is placed in a TableDataset, called *table*, where one column holds a single 1D spectrum. This table is then available for use in the user's IA session.

## 12.10 Accessing Housekeeping (HK) Data

Assuming you have access to a database whose schema is compatible with the version of the software you are running (see above for information regarding schema evolution) then the HCSS package *binstruct* can be used to access housekeeping information. Housekeeping packets are dealt with in a somewhat different way to dataframes, but there are some similarities in structure.

### 12.10.1 Accessing HK Information For a Given Obsid

[Example 12.5](#) illustrates basic housekeeping packet access for an observation with an obsid of 1400. The end product is a table with two columns, time in the first column and the housekeeping parameter value (raw) in the second column. Although an example relative to HIFI is given, this can be adapted to dealing with HK data from the SPIRE and PACS too.

```
#Example 12.5 - basic HK packet access
#import packages needed
from herschel.access import *
from herschel.access.util import *
from herschel.binstruct import *
from herschel.pus import *

#look to access HK packets associated with obsid = 1400
pk = PacketAccess(1400)

#connect to the default database to find the packets
hk_set = HcssConnection.get(pk)
```

```
#create an empty Java array list - needed for the
#PacketSequence routine below.
arrList = java.util.ArrayList()

#loop around adding the housekeeping dataset into our array
for x in range(len(hk_set)):
    arrList.add(hk_set[x])

#we can look at our array
print arrList
#...but to get something sensible we need packets in a time order.
pseq = PacketSequence(arrList)

#find packets in the sequence which contain information on
#temperatures within the focal plane unit
seq_FPU_Temp = pseq.select(TypeEquals("FPU_Temperatures"))

#find out what parameters are contained in the selected packets
#by obtaining the housekeeping parameter names from the first
#selected packet in the sequence
par_FPU_Temp = seq_FPU_Temp[0].getParametersContained()

#print out to the IA session the names of all the parameters contained
print par_FPU_Temp

#choose the FPU Temperature parameter you want to get info on
#...and get a time ordered set of housekeeping data for it
#The output file plot_fpu_hk is a TableDataset with one column for time
#(a Finetime of microseconds since 1 January 1958)
#and one for the value of the parameter (RAW rather than engineering
#value). Here we choose the parameter FPU_b_body_top for the table
#output and get the converted values (in degrees K)
plot_fpu_hk = seq_FPU_Temp.getConvertedMeasures(["FPU_b_body_top"])
time = Double1d(plot_fpu_hk[0].data/1000000.0) # puts time into seconds
data = Double1d(plot_fpu_hk[1].data)

#Now we can plot the timeline of the HK data over the
#time period of the observation (obsid=1400) by plotting the table
p = PlotXY(time, data, "A title")
```

### 12.10.2 Accessing HK Data For a Given Time Period

We may be interested in looking at HK data for longer periods of time, e.g., over an extended period covering several observations within the same database. This is particularly useful when looking for trends in instrument data.

In [example 12.6](#) we show how HK data can be obtained for a set of parameters over a given time period entered as Java Dates.

**NOTE:** Care needs to be taken that time periods being sampled are not too long since the HK data is held in memory and days of HK data can lead to an “OutOfMemory” error.



```
#Example 12.6 - Getting HK data over a given time period.
#Import needed packages for handling databases and HK data
from herschel.access import *
from herschel.access.util import *
from herschel.binstruct import *
from herschel.pus import *
#And this allows us to deal with times.
from herschel.share.fltdyn.time import *

#First we enter a start and stop time for HK information.
#We enter Java Dates, given as year (-1900), Month (-1),
#day, hour, minute, second.
#Our start_time is therefore 01:10:00 on 25 October 2004
    start_time = java.util.Date(104, 9, 25, 1, 10, 0)
#stop_time is 01:15:00 on the same day
    stop_time = java.util.Date(104, 9, 25, 1, 15,0)
#Need to convert final numbers into a FineTime used in database.
    start_1 = DateConverter.dateToFineTime(start_time)
#Date/time of start for plotted data
    prod_date = DateConverter.fineTimeToDate(start_1)
# Ditto for stop time
    stop_1 = DateConverter.dateToFineTime(stop_time)
    end_date = DateConverter.fineTimeToDate(stop_1)
#initialize some parameters
    pk=0
    hk_set = 0
#get object ready for sorting packets.
    pseq = PacketSequence()
#set up the query for accessing packets of HK data
#Here we ask for packets with an APID of 1026, which carries
#HIFI HK data. The database identified by the user's
#properties is accessed for packets of this type
#between the given start and stop FineTimes.
    pk = PacketAccess(1026,start_1,stop_1)
#Now we know where to look, we can get the packets!
#First we create an array with the packets in
    hk_set = HcssConnection.get(pk)
#.....then we loop over the array to get the contents and
#put packets into our packet sequence
    for x in range(len(hk_set)):
        pseq.add(PusTmSourcePacket(hk_set[x].getContents()))

#Now we get the parameters in the packets that we can plot.
    seq_HIFI_HK = pseq.select(TypeEquals("HIFI_HK_rev_3"))
#Lets pick out some of them
    mnemonics = ["HF_AH1_MXMG_V", "HF_AV1_MXMG_V"]
#...and get their converted (physical unit) measurements.
#"plot_HIFI_HK" is a TableDataset with a first column measuring time
#and the next 2 columns holding the HK parameter values
#at those times. We can now plot any of the parameters versus
#time, or against each other, by picking out the appropriate
#column of the table.
    plot_HIFI_HK = seq_HIFI_HK.getConvertedMeasures(mnemonics)
#This is what to do to set up the plot. Since time
#is in microseconds we convert it to
```

```
#seconds first.
#Get the first column and divide by 1 million
time = plot_HIFI_HK[0].data/1000000.0
#Let's measure time on the plot from the beginning of the observation...
#We subtract the initial time value
plot_time = time - time[0]
#We will plot the two voltages contained in columns 2 and 4
h_voltage = plot_HIFI_HK[1].data
v_voltage = plot_HIFI_HK[2].data

#Now plot the data
p = PlotXY(plot_time, h_voltage, "H Mixer Plot")
#Change to a line plot
p.setLine("H Mixer Plot")
#change the labels
p.setLabel("x-Axis", "Time (hours)")
p.setLabel("y-Axis", "Mixer voltage [V]")
#and add a title
p.setTitle("Plot of Mixer Voltages. Start: "+str(prod_date)+"
End: "+str(end_date))
#Now we can also overlay the second voltage trend in blue.
p.addLayer(plot_time, v_voltage, "V Mixer Plot")
p.setLine("V Mixer Plot")
```

An example of the kind of output one can expect from this is given in Figure 12-4.

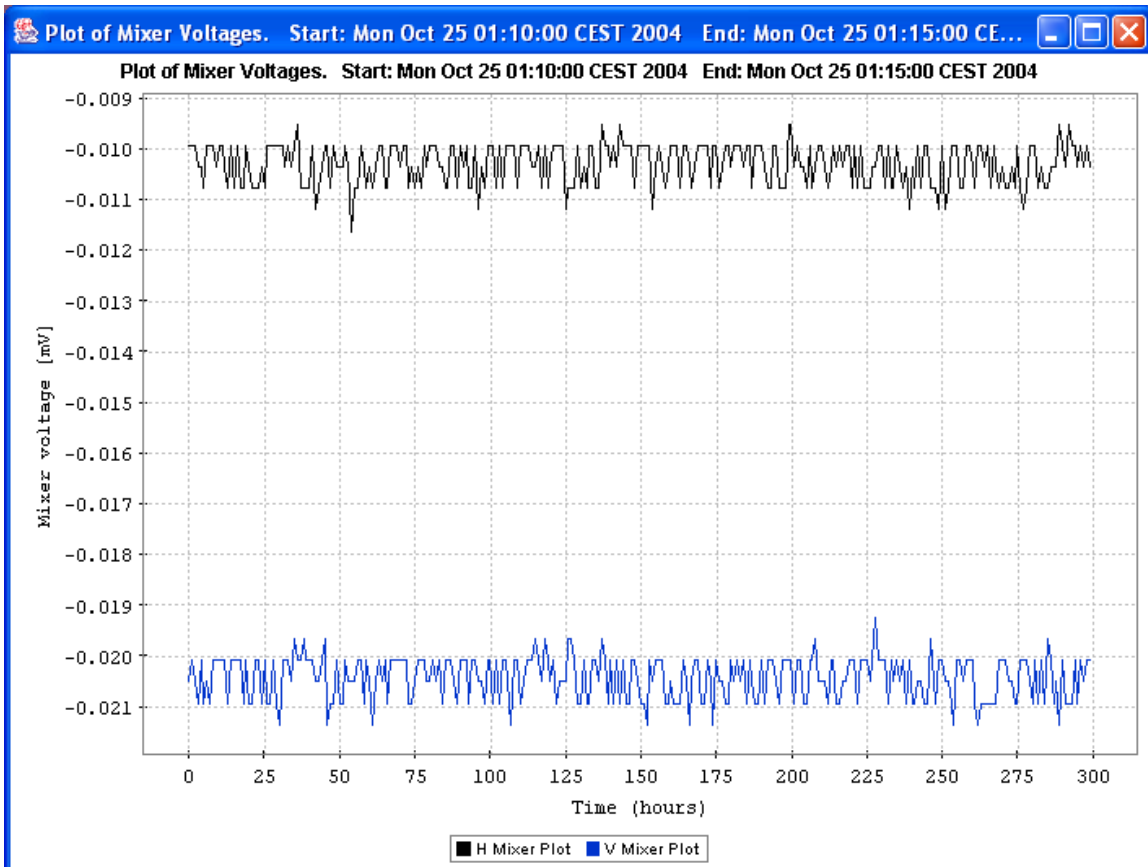


Figure 12-4: Sample timeline plot of HK data from example 12.6.

## **12.11 Removing a Database**

To remove a database that you have created can be done simply AT A TERMINAL PROMPT (not within the `jconsole` session).

```
>> removedb -rmdir mydatabase
```

## Appendix A: Example User's Property File

An example properties file to be placed in `~<user>/hcss/myconfig` file of the user (or `C:\Documents and Settings\<user>\hcss.props` for Windows users).

For most users, the first few lines are the most important ones.

```
var.hcss.workdir=C://temp
hcss.access.ccm = herschel.versant.ccm
hcss.access.query.navigate=true
hcss.access.database = ilt_qm_9${var.database.server}
dbname = ilt_qm_9
dbfactory = herschel.versant.store.StoreFactoryImpl
hcss.pg.useList = true
hcss.pg.xml.listLocation =
{${var.hcss.dir}/config/defns,${var.hcss.dir}/../../config/defns}
****add old myconfig
# HCSS Properties File - location SRON
#
# Author: Craig Porrett
#

#To show queries submitted by access
#hcss.store.verbosity = 1

# General

var.database.server = @lin-sron-02.sron.rug.nl
var.database.devel=tony_hcss${var.database.server}
dbfactory = herschel.versant.store.StoreFactoryImpl
dbname = tony_hcss

hcss.cus.database=tony_hcss@${var.database.server}
hcss.cus.instrument = HIFI
hcss.cus.tabledir = ${user.dir}/CUS/custables

var.hcss.dir=C:/ia/hifi/lib/hcss
#var.hcss.workdir = ${user.home}/
#changed by Peer on 27-09-2002
#var.hcss.dir = ${user.home}/hcss_builds/latest_build
#var.hcss.dir = /Users/users/hcssbld/hcss_builds/latest_build

# Access

hcss.access.database = ${var.database.devel}
hcss.access.test.database = ${var.database.devel}
hcss.access.connection = herschel.access.db.LocalConnection
hcss.access.network = socket
hcss.access.socket.host = localhost
```

```
hcss.access.socket.port = 8050
hcss.access.url = http://lin-sron-02.sron.rug.nl:5019/servlets/
hcss.access.packetprocessor = HIFI
hcss.access.instrumentmodel = Engineering
hcss.access.factory.query = herschel.access.db.VersantQueryFactory
hcss.access.router.host = localhost
hcss.access.router.port = 9877
hcss.access.query.allpks = select selfoid from
herschel.versant.ccm.TmSourcePacketImpl
hcss.access.query.alldfs = select selfoid from
herschel.versant.ccm.DataFrameImpl

# CCM

hcss.ccm.test.database = ${var.database.devel}

# following from Kevin's email on 29th Jan. 2004 siteid = 1 for hifi-
icc
# this following from the ICD

hcss.ccm.siteid = 1
hcss.ccm.mission.config = democonfig
hcss.ccm.mission.database = ${var.database.server}

# Formatter
# formatter package needs to be changed to use the var.hcss.dir system
hcss.formatter.directory.root = ${var.source.dir}

# MIB
var.mib.defns = ${var.hcss.dir}/data/mib/defns
var.mib.data = ${var.hcss.dir}/data/mib/example-mibs/example-1
var.mib.aux = ${var.mib.data}/auxil
var.mib.raw = ${var.mib.data}/ascii-tables
hcss.mib.database = ${var.database.devel}
#hcss.mib.database = hcssbld_hcss
hcss.mib.datadir = ${var.mib.raw}
hcss.mib.tablelist = ${var.mib.aux}/tablelist
hcss.mib.tc_command_durns = ${var.mib.aux}/tc-durns
hcss.mib.tm_param_list = ${var.mib.aux}/tmparams
hcss.mib.test_tc_command_list = ${var.mib.aux}/tcmds
hcss.mib.test_tm_param_list = ${hcss.mib.tm_param_list}
hcss.mib.tabledefs = ${var.mib.defns}/table-defns/
hcss.mib.dbroot = hcss_mib_root
hcss.mib.uplink_id = 1
hcss.mib.test_uplink_id = 1
hcss.mib.downlink_id = 1
hcss.mib.test_downlink_id = 1
hcss.mib.erroronly = false
hcss.mib.logfile = mibchecker.log
hcss.mib.readallcmds = true
hcss.mib.tc_command_list = xxx

# TM Ingest
hcss.tmingest.database = ${var.database.devel}
hcss.tmingest.port = 9877
```

```
# TM Proc

# Store
hcss.store.test.database = ${var.database.devel}

#ia dataflow
herschel.ia.dataflow.maxbuffersize = 50

# pcss needed for ia demo 28th January
hcss.mib.cus_file = gencus_scripts.out
hcss.mib.instrument = HIFI

#hcss.ccm.mission.config = democonfig
#hcss.ccm.mission.database = hcssbld_hcss@lin-sron-02.sron.rug.nl

# binstruct
hcss.binstruct.ip_filename = instr_props.ip
hcss.binstruct.tm_version_map=TmVersions.tbl
hcss.binstruct.mib=C:/ia/binstruct
hcss.binstruct.services = herschel.binstruct.mib.MibAsciiServices
hcss.binstruct.mib_source = ascii
# JConsole
hcss.jython.user.import=${user.home}/iltscripts_qm_reports.py
hcss.jconsole.buffer.size=320000
hcss.jconsole.prompt = "Tony's IA>>"
hcss.jconsole.width = 900
hcss.jconsole.height = 600
```

## Appendix B: Listing of Currently Available IA Classes

A listing of IA classes within the HCSS. Many of these are low-level tasks available to the user but not generally used in a data analysis session. Links are to on-line JavaDoc documentation.

<p><a href="#">herchel.ia.dataset</a></p> <p><a href="#">Annotatable</a>  <a href="#">Attributable</a>  <a href="#">Composite</a>  <a href="#">Dataset</a>  <a href="#">DatasetVisitor</a>  <a href="#">DataWrapper</a>  <a href="#">History</a>  <a href="#">NumericParameter</a>  <a href="#">Parameter</a>  <a href="#">ParameterVisitor</a>  <a href="#">Quantifiable</a></p> <p>Classes</p> <p><a href="#">AbstractDatasetAndDataVisitor</a>  <a href="#">AbstractDatasetVisitor</a>  <a href="#">ArrayDataset</a>  <a href="#">BooleanParameter</a>  <a href="#">Column</a>  <a href="#">CompositeDataset</a>  <a href="#">DatasetUtil</a>  <a href="#">DateParameter</a>  <a href="#">DoubleParameter</a>  <a href="#">LongParameter</a>  <a href="#">MetaData</a>  <a href="#">Product</a>  <a href="#">StringParameter</a>  <a href="#">TableDataset</a></p> <p><a href="#">herchel.ia.help.sessioninspector</a></p> <p>Classes</p> <p><a href="#">SessionInspector</a></p>	<p><a href="#">herchel.ia.dataset.demo</a></p> <p>Classes</p> <p><a href="#">Demo</a>  <a href="#">SpectrumDataset</a>  <a href="#">TableAdd</a></p> <p><a href="#">herchel.ia.demo.endtoend</a></p> <p>Classes</p> <p><a href="#">Double1dPlotter</a>  <a href="#">MakeHifiProduct</a>  <a href="#">ProcessProductMaker</a>  <a href="#">ProcessRunningAverage</a></p> <p><a href="#">herchel.ia.help.api</a></p> <p>Classes</p> <p><a href="#">Help</a>  <a href="#">HelpTask</a></p> <p><a href="#">herchel.ia.image</a></p> <p>Classes</p> <p><a href="#">Axis</a>  <a href="#">AxisGraphicsHandler</a>  <a href="#">Clamp</a>  <a href="#">Crop</a>  <a href="#">Display</a>  <a href="#">Histogram</a>  <a href="#">ImageDataset</a>  <a href="#">Layer</a>  <a href="#">Rotate</a>  <a href="#">Scale</a>  <a href="#">Translate</a>  <a href="#">Transpose</a>  <a href="#">Wcs</a></p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><a href="#">herchel.ia.image.gui</a></p> <p>Classes</p> <p><a href="#">AnnotationToolbox</a></p> <p><a href="#">ImageCutLevels</a></p> <p><a href="#">ImageDisplayStatusPanel</a></p> <p><a href="#">ImageSaveDialog</a></p> <p><a href="#">StatusPane</a></p> <p><a href="#">herchel.ia.io.ascii</a></p> <p>Interfaces</p> <p><a href="#">AsciiCsy</a></p> <p><a href="#">AsciiFixedWidth</a></p> <p><a href="#">AsciiFormatter</a></p> <p><a href="#">AsciiParser</a></p> <p>Classes</p> <p><a href="#">AbstractAsciiFormatter</a></p> <p><a href="#">AbstractAsciiParser</a></p> <p><a href="#">AsciiTableTool</a></p> <p><a href="#">CsvFormatter</a></p> <p><a href="#">CsvParser</a></p> <p><a href="#">FixedWidthFormatter</a></p> <p><a href="#">FixedWidthParser</a></p> <p><a href="#">TableTemplate</a></p>	<p><a href="#">herchel.ia.io.fits</a></p> <p>Interfaces</p> <p><a href="#">FitsDictionary</a></p> <p>Classes</p> <p><a href="#">FitsArchive</a></p> <p><a href="#">FitsRules</a></p> <p><a href="#">herchel.ia.io.fits.dictionary</a></p> <p>Classes</p> <p><a href="#">AbstractFitsDictionary</a></p> <p><a href="#">DictionaryHess</a></p> <p><a href="#">DictionaryHeasarc</a></p> <p><a href="#">DictionaryStandard</a></p> <p><a href="#">ResourceFitsDictionary</a></p> <p><a href="#">WrappedFitsDictionary</a></p> <p><a href="#">herchel.ia.jconsole.tools</a></p> <p>Classes</p> <p><a href="#">JIDE</a></p>
<p><a href="#">herchel.ia.numeric</a></p>	<p><a href="#">herchel.ia.numeric.function</a></p> <p>Classes</p>



<p><b>Interfaces</b> (note 2d to 5d versions also available) <a href="#"><u>Array1dData</u></a> <a href="#"><u>ArrayData</u></a> <a href="#"><u>ArrayDataVisitor</u></a> <a href="#"><u>ComplexData</u></a> <a href="#"><u>Logical1dData</u></a> <a href="#"><u>LogicalData</u></a> <a href="#"><u>Numeric1dData</u></a> <a href="#"><u>NumericData</u></a> <a href="#"><u>Ordered1dData</u></a> <a href="#"><u>OrderedData</u></a></p> <p><b>Classes</b> (note 2d, 3d, 4d and 5d versions also available) <a href="#"><u>AbstractArray1dData</u></a> <a href="#"><u>AbstractArrayData</u></a> <a href="#"><u>AbstractArrayDataVisitor</u></a> <a href="#"><u>AbstractComplex1dData</u></a> <a href="#"><u>AbstractLogical1dData</u></a> <a href="#"><u>AbstractNumeric1dData</u></a> <a href="#"><u>AbstractOrdered1dData</u></a> <a href="#"><u>Bool1d</u></a> <a href="#"><u>Byte1d</u></a> <a href="#"><u>Complex</u></a> <a href="#"><u>Complex1d</u></a> <a href="#"><u>Double1d</u></a> <a href="#"><u>Float1d</u></a> <a href="#"><u>Int1d</u></a> <a href="#"><u>Long1d</u></a> <a href="#"><u>Range</u></a> <a href="#"><u>Selection</u></a> <a href="#"><u>Short1d</u></a> <a href="#"><u>String1d</u></a></p>	<p><a href="#"><u>BoxCarFilter</u></a> <a href="#"><u>Complex1dFunctions</u></a> <a href="#"><u>ComplexFunctions</u></a> <a href="#"><u>Convolution</u></a> <a href="#"><u>CubicSplineInterpolator</u></a> <a href="#"><u>Double1dFunctions</u></a> <a href="#"><u>Double2dFunctions</u></a> <a href="#"><u>DoubleArrayFunctions</u></a> <a href="#"><u>DoubleFunctions</u></a> <a href="#"><u>FFT</u></a> <a href="#"><u>GaussianFilter</u></a> <a href="#"><u>Interpolator</u></a> <a href="#"><u>LinearInterpolator</u></a> <a href="#"><u>LinearLeastSquaresFitter</u></a> <a href="#"><u>MatrixFunctions</u></a> <a href="#"><u>NearestNeighborInterpolator</u></a> <a href="#"><u>Polynomial</u></a> <a href="#"><u>PolynomialFitter</u></a> <a href="#"><u>Shift</u></a> <a href="#"><u>WindowFunctions</u></a></p> <p><a href="#"><u>herchel.ia.numeric.function.util</u></a></p> <p><b>Classes</b> <a href="#"><u>MoreMath</u></a></p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><a href="#">herschel.ia.numeric.toolbox.basic</a></p> <p>Classes</p> <ul style="list-style-type: none"><li><a href="#">Abs</a></li><li><a href="#">AbstractShiftProcedure</a></li><li><a href="#">All</a></li><li><a href="#">Any</a></li><li><a href="#">ArcCos</a></li><li><a href="#">ArcSin</a></li><li><a href="#">ArcTan</a></li><li><a href="#">Basic</a></li><li><a href="#">Ceiling</a></li><li><a href="#">Concatenate</a></li><li><a href="#">Cos</a></li><li><a href="#">CosH</a></li><li><a href="#">Exp</a></li><li><a href="#">Floor</a></li><li><a href="#">IsFinite</a></li><li><a href="#">IsInfinite</a></li><li><a href="#">IsNaN</a></li><li><a href="#">Log</a></li><li><a href="#">Log10</a></li><li><a href="#">LogN</a></li><li><a href="#">Max</a></li><li><a href="#">Mean</a></li><li><a href="#">Median</a></li><li><a href="#">Min</a></li><li><a href="#">Product</a></li><li><a href="#">Reshape</a></li><li><a href="#">Reverse</a></li><li><a href="#">Rms</a></li><li><a href="#">Round</a></li><li><a href="#">Shift</a></li><li><a href="#">Sin</a></li><li><a href="#">SinH</a></li><li><a href="#">Sort</a></li><li><a href="#">Sqrt</a></li><li><a href="#">Square</a></li><li><a href="#">StdDev</a></li><li><a href="#">Sum</a></li><li><a href="#">Tan</a></li><li><a href="#">TanH</a></li><li><a href="#">Variance</a></li></ul>	<p><a href="#">herschel.ia.numeric.function.fit</a></p> <p>Classes</p> <ul style="list-style-type: none"><li><a href="#">AbstractModel</a></li><li><a href="#">AbstractModelFunction</a></li><li><a href="#">AmoebaFitter</a></li><li><a href="#">BinomialModel</a></li><li><a href="#">DataFormatter</a></li><li><a href="#">Fitter</a></li><li><a href="#">FunnyModel</a></li><li><a href="#">FunnyModelInput</a></li><li><a href="#">Gauss2DModel</a></li><li><a href="#">GaussMixModel</a></li><li><a href="#">GaussModel</a></li><li><a href="#">GaussNoPartial</a></li><li><a href="#">IndexSet</a></li><li><a href="#">IterativeFitter</a></li><li><a href="#">LevenbergMarquardtFitter</a></li><li><a href="#">LinearModel</a></li><li><a href="#">MatrixSelections</a></li><li><a href="#">ModelInput</a></li><li><a href="#">NonLinearModel</a></li><li><a href="#">NullModel</a></li><li><a href="#">PolynomialModel</a></li><li><a href="#">PowerModel</a></li><li><a href="#">SineAmpModel</a></li><li><a href="#">SineMixedModel</a></li><li><a href="#">SineModel</a></li></ul>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p><a href="#">herchel.ia.plot</a></p> <p>Interfaces</p> <p><a href="#">CoordMouseListener</a></p> <p>Classes</p> <p><a href="#">Axis</a></p> <p><a href="#">ComponentList</a></p> <p><a href="#">CompositePlot</a></p> <p><a href="#">CoordMouseEvent</a></p> <p><a href="#">FontChooser</a></p> <p><a href="#">FontDisplay</a></p> <p><a href="#">Layer</a></p> <p><a href="#">LocationDisplay</a></p> <p><a href="#">PlotProperties</a></p> <p><a href="#">PlotPropertyManager</a></p> <p><a href="#">PlotXY</a></p> <p><a href="#">PlotXYCompositeRenderer</a></p> <p><a href="#">PlotXYSeries</a></p> <p><a href="#">View</a></p> <p><a href="#">herchel.ia.task</a></p> <p>Classes</p> <p><a href="#">_Dummy</a></p> <p><a href="#">Share</a></p> <p><a href="#">Signature</a></p> <p><a href="#">Task</a></p> <p><a href="#">TaskParameter</a></p>	
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--