Herschel Interactive Analysis

A Basic User's Manual

V0.3, 22 July 2004

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

1 The Herschel Common Science System and Interactive Analysis (IA) 6
1.1 Brief Overview 6
1.2 Availability of IA and Operating Systems 6
1.3 Related Documentation: HowTos and Javadocs 7
1.4 Versioning 7
1.5 Previous Versions of IA Manual 7
2 HCSS Downloading and Installation 8
2.1 Introduction 8
2.2 Platform 9
2.3 Pre-Installation Requirements 9
2.4 Notes on Versant Databases 10
2.5 User Installation Procedure 11
2.5.1 UNIX 11
2.5.2 Windows 13
2.6 Developer Installation Procedure 15
2.6.1 UNIX 15
2.6.2 Windows 17
2.7 Property Initialization 19
3 Introduction to Working in IA: Using Jconsole 20
3.1 Introduction 20
3.2 Running IA under Jconsole: 20
3.2.1 File Menu 22
3.2.2 Console Menu 23
3.2.3 Edit Menu 23
3.2.4 Run Menu 23
3.3 Standard Settings for Jconsole 23
3.4 Using `import.py` to customize Jconsole 24
3.5 Programming Loops in Jconsole 24
3.6 Multiline Statements with Jconsole 26
4 Some IA Basics & Beginning Jython 27
4.1 Basics 27
4.2 Lists and Dictionaries 28
4.2.1 Slicing Lists and Arrays 28
4.2.2 Dictionaries 29
4.3 Augmenting Values and Arrays 30
4.4 Printing 30
4.5 Functions 30
4.6 Blocks 31
4.7 Classes 31
4.8 Some Useful Extra Items 33
5 Handling Arrays and Other Datasets 34
5.1 Introduction 34
5.2 Getting started 34
5.3 Creating a Simple Array Dataset 35
5.4 Simple Array Manipulation 35

5.5 Dataset attributes ... 36
5.6 Creating and Viewing a Table Dataset ... 36
5.7 Creating a Composite Dataset ... 37
6 IA Numeric: Basic Functions for Herschel IA ... 39
6.1 Introduction ... 39
6.2 Getting Started ... 39
6.3 Numeric Arrays ... 40
6.4 Arithmetic Operations ... 40
6.5 Logical Operations ... 41
6.6 Functions – Lambda Expressions ... 41
6.7 Functional Programming ... 42
6.7.1 Filtering ... 42
6.7.2 Reduction ... 43
6.8 The 'where', 'filter' and 'get' methods ... 44
6.9 Advanced Tips ... 45
6.10 Complex Arrays ... 46
6.11 Multi-Dimensional Arrays ... 46
6.12 Vectors and Matrices ... 48
6.13 Function Library ... 48
6.13.1 Basic Functions ... 49
6.13.2 Discrete Fourier Transform ... 49
6.13.3 Convolution ... 49
6.13.4 Boxcar and Gaussian Filters ... 52
6.13.5 Interpolation Functions ... 52
6.13.6 Basic Fitter Routines ... 52
6.14 Example Numeric Programs ... 55
7 IA Plot: Basic Plotting of Data ... 56
7.1 Introduction ... 56
7.2 What do I need to make a simple XY plot? ... 57
7.3 How to setup the properties ... 59
7.3.1 How to modify properties ... 60
7.3.2 Plot properties ... 61
7.3.3 Layer properties ... 61
7.3.4 Axes properties. ... 62
7.3.5 How to use properties. ... 63
7.4 How to use plot in jython scripts ... 64
7.4.1 What about these `Layers`? ... 66
7.4.2 What can I do with `Axis`? ... 68
7.4.3 How can I annotate and decorate my plot? ... 70
7.4.4 How can I make my plots more colorful? ... 71
7.5 What about a complete example? ... 71
8 Display: Handling Images with Herschel IA ... 72
8.1 Introduction ... 72
8.2 Using ImageDatasets ... 73
8.3 How can I display my image? ... 74
8.4 Display in more detail. ... 76

8.5 ... 77
8.6 How can I use Operations on my images? ... 77
8.7 How can I display my own numeric2d datatypes? ... 78
8.8 How to Use Different Layers ... 79
8.9 How to place annotations on the image ... 80
8.9.1 Annotations from the Command Line in your IA session ... 80
8.9.2 Annotations using the annotation toolbox ... 81
8.10 Download examples ... 83
9 Other IA Packages: What is Available? ... 84
9.1 Introduction ... 84
9.2 Overview of JavaDocs Documentation for IA Packages ... 85
9.3 Package view ... 86
9.4 Class view ... 86
9.5 Tree view ... 90
9.6 Deprecated view ... 90
9.7 Index view ... 90
9.8 IA Packages And Documentation ... 90
9.8.1 herschel.ia.dataflow ... 90
9.8.2 herschel.ia.dataset ... 91
9.8.3 herschel.ia.demo ... 91
9.8.4 herschel.ia.doc ... 91
9.8.5 herschel.ia.image ... 91
9.8.6 herschel.ia.io ... 92
9.8.7 herschel.ia.jconsole ... 92
9.8.8 herschel.ia.numeric ... 92
9.8.9 herschel.ia.plot ... 93
9.8.10 herschel.ia.task ... 93
9.8.11 herschel.ia.ui ... 93
10 Import and Export of Tabular ASCII and FITS Files ... 94
10.1 Introduction ... 94
10.2 Getting Started with ASCII Import/Export ... 94
10.3 Basic ASCII Import/Export Tool Usage ... 95
10.3.1 Import Parsers ... 95
10.3.2 Comma-Separated-Variable Parser ... 96
10.3.3 Fixed-Width Parser ... 96
10.3.4 Export Formatters ... 97
10.3.5 Comma-Separated-Variable Formatter ... 97
10.3.6 Fixed-Width Formatter ... 97
10.3.7 Table Template ... 98
10.4 Example of How to Import/Export ASCII Tables in IA ... 98
10.5 Overview of FITS IO ... 100
10.6 Getting Started With FITS IO ... 100
10.6.1 Basic FITS IO Tool ... 100
10.6.2 Parameter Name Conversion and FITS Header ... 101
10.6.3 Caveats ... 101
11 Setup and Use of Databases ... 103

11.1 Introduction 103
11.2 Starting Up A Database: 104
11.2.1 Unix 104
11.2.2 Windows 104
11.3 Schema Evolution 105
11.4 Using an existing database 105
11.4.1 Initializing an old database 107
11.4.2 Schema Tool commands 107
11.5 Initializing the Database: 107
11.6 Quick Database Creation 107
11.7 Providing Database Access for an IA Session 108
11.7.1 Editing Properties File 108
11.7.2 Using the `Propgen` Tool 108
11.8 Browsing a Database 109
11.9 Getting Data Frames From a Database 110
11.9.1 Command Line Access to Data Frames 110
11.9.2 From Database to ASCII File 111
11.9.3 Downloading Dataframes from a Database Using a GUI 113
11.9.4 Accessing Housekeeping Data 115
11.10 Removing a Database 125
12 Appendix A: Example User's Property File 126
13 Appendix B: Listing of Currently Available IA Classes 128

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: HowTos and Javadocs
1.4 Versioning
1.5 Previous Versions of IA Manual

1.1 Brief Overview

The Herschel Common Science System (HCSS -- ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/doc/index.html) 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.

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+)
- Windows (2000, XP)

For download and installation instructions see Chapter 2.

1.3 Related Documentation: HowTos and Javadocs

There currently exist a number of IA "HowTo" documents on the Herschel IA website. Several of these have been extensively borrowed from in the creation of parts of this manual. It is expected that as the current IA system evolves, mature "HowTo" documents will be upgraded and incorporated into the User's Manual. The user is therefore referred to the HowTo documents and Javadocs for early documentation on new capabilities of Herschel IA that may not yet be incorporated into the User's Manual.

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 Manual

None

Herschel IA Chapter 2

2 HCSS Downloading and Installation

In case of any problems during installation please email hcss_help@rssd.esa.int

Chapter 2 Contents

2.1 Introduction
2.2 Platform
2.3 Pre-Installation Requirements
2.4 Notes on Versant Databases
2.5 User Installation Procedure
  2.5.1 UNIX
  2.5.2 Windows
2.6 Developer Installation Procedure
  2.6.1 UNIX
  2.6.2 Windows
2.7 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 10 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 RedHat 8.0 running on an Intel processor.

Note that this OS version is LSB (Linux Standard Base) v1.2 so theoretically one should be safe using another Linux distribution providing it has been certified LSB v1.2, see: LSB certified products for more information.

Current platforms now 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.

- **HCSS Users :**
- **In many cases users will not require any additional software in order to install and run the HCSS.**
- *ALL USERS*: You will 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.
- *For database usage*: Versant Database System (see notes on Versant below) will need to be installed. This will allow setting up databases and accessing databases. Not needed if you are not using HCSS databases.
- *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

- **HCSS Developers :**
- 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://ccvs.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

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 Notes on Versant Databases

The HCSS allows for the setting up and interaction with both local and remote databases. Currently, Versant databases are used. For institutions using central databases (possibly on a single machine) the database software installation is the responsibility of a Herschel software administrator at the institute.

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.

Note for Versant 6.0.5 installation under UNIX: It is important that the Versant shared libraries appear in the LD_LIBRARY_PATH environment variable in the correct order: vds6051/lib should be before vds6051/lib/cxxchk

Note for Versant 6.0.5.1 installation under Linux SuSE 8.1: To install Versant on this platform you must edit the installation script `vds6051_linux.bin`. Any line containing the variable `LD_ASSUME_KERNEL` must be edited, by replacing the first character of that line with a `#`. e.g. the line:

```
export LD_ASSUME_KERNEL
```

must be changed to:

```
#xport LD_ASSUME_KERNEL
```

No other characters must be added or deleted and not tabulations should be altered or new lines added. Also you must ensure the system variable `DISPLAY` is set correctly or other errors will be reported from the Graphics Environment.

2.5 User Installation Procedure

This installation procedure is intended for all users using the pre-built version of an HCSS release.

If for any reason you require the reference platform number, please refer to the README file located in the installation directory under:
`${HCSS_DIR}/lib/ext/README_vN.N`

Note: at this stage it is assumed the "Pre-Installation Requirements" described above, have been met.

2.5.1 UNIX

Download the pre-built HCSS release from our Herschel FTP server `ftp.rssd.esa.int` (login as user anonymous) and fetch the release (hcss_nnn.tgz) using the commands:

```
ftp> cd pub/HERSCHEL/csdt/releases/builds/
ftp> bin
ftp> get hcss_nnn.tgz
```

or using a Web browser pointing to URL:
ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/builds/

Please replace 'nnn' in the filename `hcss_nnn.tgz` with the particular HCSS build you require or use `hcss.tgz` which points at the latest build.

Note: if you need to download an 'official' HCSS User Release, please follow URL:
http://www.rssd.esa.int/herschel_scripts/hscdt/ccInfo2Html.cgi?theProject=HCSS&theDisplay=RELTAG instead.

Unpack the file in a suitable directory. For example:

```
cd /local/software

gzip -dc hcss_nnn.tgz | tar xvf -  or alternatively
tar xvfz hcss_nnn.tgz
```

Note: you must use the GNU versions of zip and tar in order to be able to unpack an HCSS pre-built version.

Configure your environment by specifying the following after the section referred to in the "Pre-Installation Requirements" as follows:

set the variable ${HCSS_DIR} to point at the installation root directory:

```
setenv HCSS_DIR /local/software/hcss
```

source the configuration file that is part of the pre-built HCSS located in the directory:

```
source ${HCSS_DIR}/config/setHcssExtLibs_csh
```

or if using the bash shell

```
source ${HCSS_DIR}/config/setHcssExtLibs_bsh
```

The file setHcssExtLibs_csh can also be input line-by-line by the user, if preferred. **NOTE**: . Please make sure that the jar file `jython-2.1_hcssPatch.jar` is specified in the CLASSPATH before the file `jython.jar`

Add the HCSS directory containing all executables to your path variable:

```
set path = ($HCSS_DIR/bin $path)
```

Create variable HCSS_PROPS with value:

```
setenv HCSS_PROPS ${HCSS_DIR}/config/devel.props:${HOME}/.hcss/myconfig
```

`${HOME}/.hcss/myconfig` is the file which contains properties of your specification for an HCSS session which needs to be created by the user. It should contain the following basic components (this can be added to later when we set up HCSS properties:

```
var.database.server = @server
var.hcss.dir = /local/software/hcss
var.hcss.workdir = /tmp/hcss${user.home}
```

where "`var.database.server`" specifies the host name of the Versant Database Server (your system administrator should know your Versant Database Server see Chapter 10 regarding Databases for more information) and "`var.hcss.dir`" specifies the directory where the HCSS release is installed and "`var.hcss.workdir`" specifies a directory to which you have write access for hcss output files.

Also within this file you can contain sub system specific properties, for example:
`hcss.cus.database = cus_db${var.database.server}` #to specify which database to use for the Common Uplink System (CUS).

It is recommended, in order to make an HCSS environment available on login, that the following be placed in your `.cshrc` file.

```
setenv HCSS_DIR /local/software/hcss
source ${HCSS_DIR}/config/setHcssExtLibs_csh
set path = ($HCSS_DIR/bin $path)
setenv HCSS_PROPS ${HCSS_DIR}/config/devel.props:${HOME}/.hcss/myconfig
```

Properties that can be set in the `$HOME/.hcss/myconfig` file are discussed up Property Initialization. A list of example properties are available in Appendix A: User Properties.

Set a bookmark in your web browser to view the documentation in the installation directory:

```
file:///local/software/hcss/doc/index.html
```

For setting up of databases, see the manual chapter dealing with Databases, notably the section on Database Initialization.

2.5.2 Windows

The instructions for Windows have been tested on Windows 2000 and XP Pro, but may be slightly different for other versions of windows.

Download the pre-built HCSS release from our Herschel FTP server `ftp.rssd.esa.int` (login as user anonymous) and fetch the release (hcss_nnn.tgz) using the commands:

```
ftp> cd pub/HERSCHEL/csdt/releases/builds/
ftp> bin
ftp> get hcss_nnn.tgz
```

or using a Web browser pointing to URL:
ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/releases/builds/

Please replace 'nnn' in the filename `hcss_nnn.tgz` with the particular HCSS build you require or use `hcss.tgz` which points at the latest build.

Note: if you need to download an 'official' HCSS User Release, please follow URL: http://www.rssd.esa.int/herschel_scripts/hscdt/ccInfo2Html.cgi?theProject=HCSS&theDisplay=RELTAG instead.

Unpack the file (e.g. using WinZip) in a suitable directory, for example: `c:\herschel\`

Configure your environment with the following variables:

- Go to the Control Panel (`Start->Settings->Control Panel`) then select `System->Advanced->Environment Variables`
- Create a new variable by choosing `'New...'` and for the *Variable Name* enter `HCSS_DIR` and for the Variable Value enter the 'hcss' directory of where the HCSS software was unpacked (e.g. `c:\herschel\hcss`).
- If the variable CLASSPATH does not exist choose `'New...'` and define it, otherwise highlight the variable CLASSPATH, choose `'Edit...'`. Then add for the *Variable Value*:

```
%HCSS_DIR%\lib\hcss.jar;%HCSS_DIR%\lib\ext\junit\junit.jar;
%HCSS_DIR%\lib\ext\fits\fits.jar;%HCSS_DIR%\lib\ext\fits\im
age.jar;
%HCSS_DIR%\lib\ext\fits\util.jar;%HCSS_DIR%\lib\ext\jXDF\xd
f.jar;
%HCSS_DIR%\lib\ext\jython\jython-2.1_hcssPatch.jar;
%HCSS_DIR%\lib\ext\jython\jython.jar;
%HCSS_DIR%\lib\ext\jep\jep-
2.24.jar;%HCSS_DIR%\lib\ext\quantity\nTquantity.jar;
%HCSS_DIR%\lib\ext\jfreechart\jfreechart-0.9.18.jar;
%HCSS_DIR%\lib\ext\jfreechart\jcommon-0.9.3.jar;
%HCSS_DIR%\lib\ext\jai\jai_core.jar;%HCSS_DIR%\lib\ext\jai\
jai_codec.jar;
%HCSS_DIR%\lib\ext\jai\mlibwrapper_jai.jar;%HCSS_DIR%\lib\e
xt\jsky\jsky.jar;
%HCSS_DIR%\lib\ext\jsky\diva.jar;%HCSS_DIR%\lib\ext\jsky\do
m4j.jar;
%HCSS_DIR%\lib\ext\jsky\hcompress.jar;%HCSS_DIR%\lib\ext\js
ky\jel.jar;
%HCSS_DIR%\lib\ext\jsky\log4j.jar;
%HCSS_DIR%\lib\ext\jama\Jama-
1.0.1.jar;%HCSS_DIR%\lib\ext\jh\jh.jar
```

Note: the jar file `jython-2.1_hcssPatch.jar` must be specified in the CLASSPATH before the file `jython.jar`

- Add to your PATH `%HCSS_DIR%\bin`
- If the variable USERPROFILE does not exist choose 'New...' and add it with the value '`C:\Documents and Settings\user name`'
- Create variable (as above) HCSS_PROPS with value `%HCSS_DIR%\config\devel.props;%USERPROFILE%\hcss.props` # which is the file which contains your properties (see next bullet 4)

Create a file specified by your HCSS_PROPS variable (e.g. `%USERPROFILE%\hcss.props`) containing:

```
var.database.server = @server
var.hcss.dir = c:/herschel/hcss
var.hcss.workdir = c:/tmp
```

where "`var.database.server`" specifies the host name of the Versant Database Server (your system administrator should know your Versant Database Server)
and "`var.hcss.dir`" specifies the directory where the HCSS release is installed
and "`var.hcss.workdir`" specifies a directory to which you have write access for some hcss output files.

Note: the use of the forward slash '/' in the configuration file for Windows.

Also within this file you can contain sub system specific properties, for example:
`hcss.cus.database = cus_db${var.database.server}` #to specify which database to use for the CUS

Properties that can be set in the `$HOME/.hcss/myconfig` file are discussed in Property Initialization. A list of example properties are available in Appendix A: User Properties.

Set a bookmark in your web browser to view the documentation in the installation directory:

```
%HCSS_DIR$/doc/index.html
```

Navigate to `%HCSS_DIR$/doc/index.html`, double click (to start browser) and add to favourites/bookmarks

See chapter on Databases for setup and interaction with HCSS databases.

2.6 Developer Installation Procedure

This installation procedure applies to HCSS developers, who wish to build the software from the source code.

2.6.1 UNIX

The environment variable `HCSS_DIR` is used to specify the location of generated files from the HCSS system. In this example `HCSS_DIR` is set to `$HOME/out/hcss`

Check out the HCSS from CVS (if you do not have a CVS account, contact hcss_help@rssd.esa.int):

```
setenv CVSROOT
:pserver:${USER}@astro.esa.int:/local/repositories/HERSCHEL_CVS
cvs checkout develop/main
```

Configure your environment for the HCSS as follows:

```
setenv HCSS_DIR $HOME/out/hcss
setenv CLASSPATH ${HCSS_DIR}/lib:${CLASSPATH}
setenv HCSS_PROPS
${HCSS_DIR}/config/devel.props:${HOME}/.hcss/myconfig
set path = ($HCSS_DIR/bin $path)
```

N.B. where CLASSPATH contains all the necessary libraries stated in the Pre-Installation Requirements

The HCSS is built using the 'Jake' tool, that is included with the HCSS. The default output directory is `$HOME/out`, to specify a different output directory you must change the property `hcss.jake.outdir` in the `jake.root` file and system variable in `makejake`. You should initially perform a full build, as follows:

```
cd develop/main
makejake
jake -build # Only do this the first time, on the whole tree
sh ~/out/hcss/bin/fixhcssperm # Set execute permissions on scripts
```

Jake's "-build" option performs a complete build, which includes compiling the code, generating JavaDoc and copying various files to the output directory.

The build should result in the following structure in $HCSS_DIR:

```
$HCSS_DIR/hcss/bin # Executable scripts to run HCSS applications
$HCSS_DIR/hcss/config # Global configuration files
$HCSS_DIR/hcss/doc # Documentation for the HCSS
$HCSS_DIR/hcss/doc/api # JavaDoc
$HCSS_DIR/hcss/data # Test data used for HCSS
```

Create a file `$HOME/.hcss/myconfig` containing:

```
var.database.server = @server
var.hcss.dir = ${user.home}/out/hcss
var.hcss.workdir = /tmp/hcss${user.home}
```

where "`var.database.server`" specifies the host name of the Versant database server (your system administrator should know your versant server)
and "`${user.home}/out/hcss`" corresponds to the environment variable HCSS_DIR
and "`/tmp/hcss${user.home}`" is the path to scratch area for the HCSS. A directory where you have write access as some test harnesses and applications create or output files.

You can add other entries to "`myconfig`" to override the default properties which are used to configure applications.

Set a bookmark in your web browser to view the documentation:

```
$HCSS_DIR/doc/index.html
```

See manual chapter on Databases for setup and use of databases within HCSS.

To perform a *recompilation* of a subsystem, simply run 'jake' in the appropriate sub directory, as before (See "jake -help" for further details of the new options that are available).

Note that there is a package "herschel.release". This contains files that previously resided in the top-level 'main' directory. A build will copy these into the top-level output directory.

2.6.2 Windows

The environment variable `HCSS_DIR` is used to specify the location of generated files from the HCSS system. In this example `HCSS_DIR` is set to `c:\out\hcss`

Check out the HCSS from CVS (if you do not have a CVS account contact hcss_help@rssd.esa.int):

Define an environment variable called CVSROOT

```
setenv CVSROOT
pserver:${USER}@astro.esa.int:/local/repositories/HERSCHEL_CVS
```

where user is your CVS account name

Then use a CVS gui application (e.g. WinCVS) to checkout "develop/main", or the cvs command line and issue `cvs checkout develop/main`

Configure your environment as follows:

Create variables:

```
HCSS_DIR c:\out\hcss
HCSS_PROPS %HCSS_DIR%\config\devel.props;%USERPROFILE%\hcss.props
```

Add to your CLASSPATH (N.B. where CLASSPATH contains all the necessary libraries stated in the Pre-Installation Requirements).

```
%HCSS_DIR%\lib
```

Add to your PATH

```
%HCSS_DIR%\bin
```

The HCSS is built using the 'Jake' tool, that is included with the HCSS. You should initially perform a full build, as follows:

Go to the directory where you checked out the source and then to `develop\main`.

Create a file `makejake.bat`, and insert the following lines:

```
javac -d %HCSS_DIR%\lib herschel\devel\jake\*.java
copy herschel\devel\jake\jake.defaults
     %HCSS_DIR%\lib\herschel\devel\jake
```

Run the `makejake` batch file (e.g. double click on it)

Create a file `jake.root`, and insert the line:
`hcss.jake.outdir = c:/out/hcss`
Note the use of the forward slash '/'.

Then run the command:

```
jake -build   # Only do this the first time, on the whole tree
```

Jake's "-build" option performs a complete build, which includes compiling the code, generating JavaDoc and copying various files to the output directory.

The build should result in the following structure:

`%HCSS_DIR%/hcss/bin` # Executable scripts to run HCSS applications
`%HCSS_DIR%/hcss/config` # Global configuration files
`%HCSS_DIR%/hcss/doci` # Documentation for the HCSS
`%HCSS_DIR%/hcss/doc/api` # JavaDoc
`%HCSS_DIR%/hcss/data` # Test data used for HCSS

Create a file specified by your HCSS_PROPS variable (e.g. %USERPROFILE%`\hcss.props`) containing:

```
var.database.server = @server
var.hcss.dir = c:/out/hcss
var.hcss.workdir = c:/tmp
```

where "`var.database.server`" specifies the host name of the Versant database server (your system administrator should know your versant server)
and "`var.hcss.dir`" corresponds to the environment variable HCSS_DIR.
and "`var.hcss.workdir`" is the path to scratch area for the HCSS. A directory to which you have write access for some hcss output files.

You can add other entries to "`hcss.props`" or `${HOME}/.hcss/myconfig` to override the default properties which are used to configure applications.

Note: the use of the forward slash '/' in properties for the HCSS with Windows.

Set a bookmark in your web browser to view the documentation in the installation directory:

Navigate to `%HCSS_DIR%/doc/index.html` file, double click (to start browser) and add to favourites/bookmarks

Consult the manual chapter on Databases for setup and use of databases within HCSS before attempting to interact with databases on the HCSS.

To perform a *recompilation of a subsystem*, simply run 'jake' in the appropriate sub directory, as before (See "jake -help" for further details of the new options that are available).

Note that there is a package "herschel.release". This contains files that previously resided in the top-level 'main' directory. The build copies these into the top-level output directory.

2.7 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 10 (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

3.3 Standard settings for Jconsole
3.4 Using import.py to customize Jconsole
3.5 Programming Loops with Jconsole
3.6 Multiline Statements in Jconsole

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.

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 session from which it was started. This includes information on the settings used on `jconsole` startup and information on database access (basically 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. In the User Manual we provide some basic startup information. Further information about the `jconsole` environment is available in the Jconsole HowTo document.

After inputting the `jconsole` command, the feedback noted above should appear. Following the feedback, a separate three-paned console window should appear (see Figure 3-1).

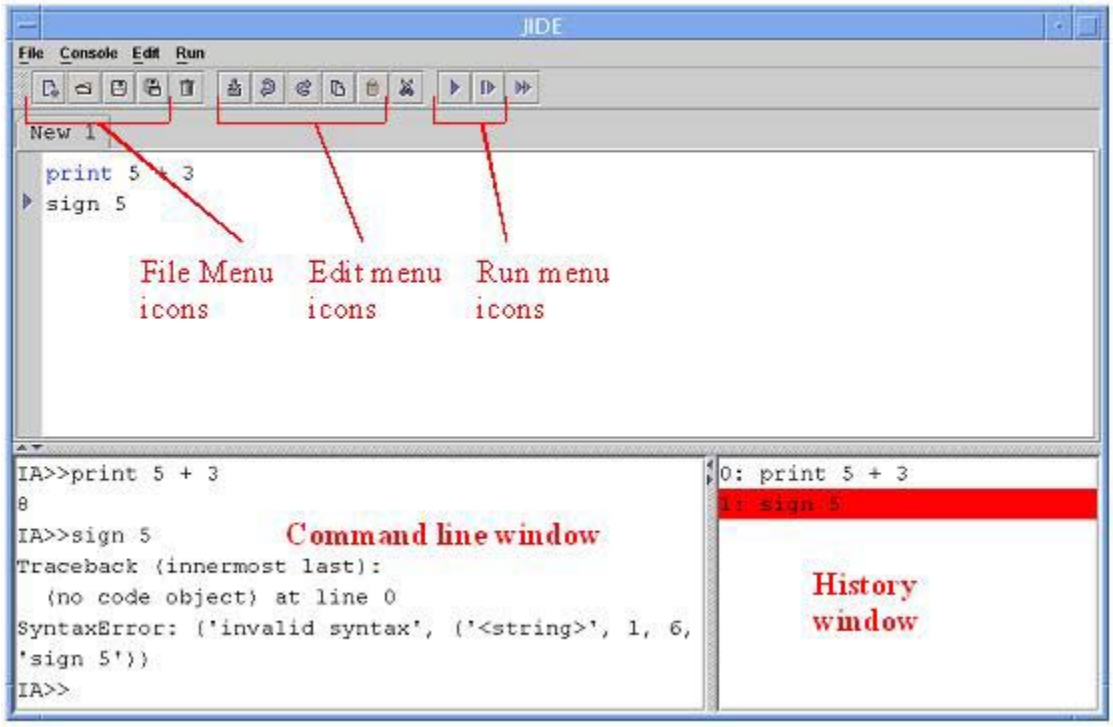

Figure 3-1: The Jconsole window set up.

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.

To the bottom right of the console contains a **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 last 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.

3.3 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  hcss.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.

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 chapter of User's Manual 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

```
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/elsif/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)
```

Herschel IA Chapter 4

4 Some IA Basics & Beginning Jython

Chapter 4 Contents

4.1 Basics
4.2 Lists and Dictionaries
4.3 Augmenting Values and Arrays
4.4 Printing
4.5 Functions
4.6 Blocks
4.7 Classes
4.8 Some Useful Extra Items

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 types so you don't have to declare them (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 identations 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 written within 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!

4.2.1 Slicing Lists and Arrays

The last two examples with name (above) are also examples of slicing. For instance,

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

Again – the end is non-inclusive.

- Choosing `y[:4]` means "take every element from the beginning of the list up to element 4, *non-inclusive*."
- 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.

4.2.2 Dictionaries

Dictionaries are like lists that are unordered. To access an element you use a key or "name". This is what is used to look up the value of an element. A dictionary therefore 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
```

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

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 += 2      # adds 2 to the value of a

IA>> b *= 3      # multiplies b by 3
```

We can add to arrays too.

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

4.4 Printing

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')
IA>> print >> file, 2      # puts the number 2 into output.txt
```

4.5 Functions

Here we name a piece of code and call it with some parameters an d 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.value
IA>> print pvalue()
```

4.6 Blocks

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

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. 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
```

After putting this in `jconsole` use the mouse to select just the above section of code and hit the run section () icon button.

The program is `Basket`, and once created we can run it using `Basket() in jconsole`. 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 after the above and then input to `jconsole`

```
IA>> a = Basket()
#this line sets up an empty basket
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 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. The **constructor** creates and **instance** of the object (in the above case it creates a basket we can put things in).

4.8 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. Packages have been developed in Herschel IA that 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 Creating a Simple Array Dataset
5.4 Simple Array Manipulation
5.5 Dataset Attributes
5.6 Creating a table dataset
5.7 Creating a composite dataset
5.8 Miscellaneous examples of using datasets

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 *herschel.ia.dataset* package.

There are three types of basic datasets:

- array datasets (arrays of numbers, strings, etc. in 1D, 2D or 3D)
- 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

Start up the jconsole application. Assuming the HCSS is installed, the user should simply type:

```
jconsole
```

Type in the following import statements:

```
IA>> from herschel.ia.dataset import *
IA>> from herschel.ia.numeric import *
IA>> from herschel.ia.numeric.function import *
```

These commands allow us to import the IA dataset package, the top level of the IA *numeric* package (currently containing many functions – discussed in more detail in Chapter 6, but used here to help illustrate the use of datasets). We also import numeric functions.

5.3 Creating a Simple 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).

Type in the following steps:

```
IA>> x=Double1d.range(10)  #This 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`. In this example, a simple description is also written to the dataset. The range command produces integer numbers from 0 to 9. This is changed to a set of real numbers (doubles).

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             #just print the data held 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
```

5.4 Simple Array Manipulation

Datasets can be manipulated using basic arithmetic and numeric functions, e.g., addition, subtraction etc.

```
IA>> a = s - b
```

if `b` is another 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`.

```
IA>> m = 100 * s
```

#simply multiplies the size of each element of the array `s` by 100.

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.

5.5 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 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 or local-store equivalent searches - 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. The allowed data types for meta-data elements are String, Boolean, Long, Double and Date. See the JavaDoc on the class MetaData for more information of the allowed types.

The following code snippet shows how to add 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 Creating and Viewing a Table Dataset

What is often required is to store data in a tabular format of M rows by N columns. The `TableDataset` provides such a means. In the following example, a `TableDataset` with 3 columns, one containing a sequence of numbers from 1 to 100, the second containing 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, is created. The column names are `x`, `sin` and y respectively. Note that `TableDataset` type **requires** all column lengths to be the same.

```
IA>> x=Double1d.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]
```

5.7 Creating 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 of data within data, to be constructed. Examples of complex datasets are grouped observation (making a map for instance). 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`.

```
#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 in 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)
```

Herschel IA Chapter 6

6 IA Numeric: Basic Functions for Herschel IA

Chapter 6 Contents

6.1 Introduction
6.2 Getting Started
6.3 Numeric Arrays
6.4 Arithmetic Operations
6.5 Logical Operations
6.6 Functions – Lambda Expressions
6.7 Functional Programming
6.7.1 Filtering
6.7.2 Reduction
6.8 The 'where', 'filter' and 'select' methods
6.9 Advanced Tips
6.10 Complex Arrays
6.11 Multi-Dimensional Arrays
6.12 Vectors and Matrices
6.13 Function Library
6.13.1 Basic Functions
6.13.2 Discrete Fourier Transform
6.13.3 Convolution
6.13.4 Boxcar and Gaussian Filters
6.13.5 Interpolation Functions
6.13.6 Basic Fitter Routines
6.14 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

Following the installation of the HCSS we start with the `jconsole` application. This is started by typing the following within any terminal window:

```
Jconsole
```

In order to use the numeric functions described in this chapter, the user should start by importing several parts of the numeric package.

Type in the following import statements at the `jconsole` prompt:

```
from herschel.ia.numeric import *
from herschel.ia.numeric.function.Double1dFunctions import *
from herschel.ia.numeric.function.DoubleFunctions import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
from herschel.ia.numeric.function.MatrixFunctions import *
```

6.3 Numeric Arrays

The library provides numerical array classes for 1, 2 and 3 dimensional arrays of numbers, including `double`, `float`, `long`, `int` and `short` values. For example, `Int1d` is a one-dimensional array of int values, whilst `Double2d` is a two-dimensional array of double values. The functions provided by the library are primarily for numerical calculations on double arrays. A `Double1d` array may be created in a variety of ways:

```
y = Double1d([1,2,3,4])
# Create from a Jython array
y = Double1d(4)
# Create an array with 4 elements [0.0,0.0,0.0,0.0]
y = Double1d.range(4)
# Create an array with values up to 4 -- [0.0,1.0,2.0,3.0]
#Arrays can be indexed to access or modify individual elements or
#slices:
y = Double1d.range(5) - 2
# Ditto the above but subtracting 2 [-2.0,-1.0,0.0,1.0,2.0]
x = y[2]
# x is the third value in the array - counting starts from 0.
# x == 0.0
y[1] = 6
# changes second value in the array to 6.0
# i.e., y == [-2.0,6.0,0.0,1.0,2.0]
print len(y)
# Returns the length of the array - 5.
x = y[1:4]
# Takes elements from 1 to 4 (non-inclusive of upper value)
# x == [6.0,0.0,1.0]
y[1:4] = [-4.0, 0.0, 4.0]
# Replaces the three elements in y with the given values
# y == [-2.0,-4.0,0.0,4.0,2.0]
```

6.4 Arithmetic Operations

Numeric arrays support arithmetic operations that are applied element-by-element. For example:

```
y = Double1d.range(5)              #y == [0.0,1.0,2.0,3.0,4.0]
print y * y * 2 + 1                #will print out the following -
                                    [1.0,3.0,9.0,19.0,33.0]
```

This is much simpler (and runs much faster) than writing an explicit loop in Jython. It is important to appreciate that the '+' operator does not concatenate arrays, as it does with Jython arrays.
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]
```

All arrays currently support the following arithmetic operators:
+, -, *, /, %, **

6.5 Logical Operations

Note that the 'modulo' operator '`%`' provides the normal Jython semantics for this operation, which is not the same as that of the Java '`%`' operator.
The following relational operators are also provided, which return a `Bool1d` array:

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

For example:

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

6.6 Functions – Lambda Expressions

Functions are implemented in the underlying Java numeric library as *function objects*. In Jython, these 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 function objects that could be applied to 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
print f(y)                                 # [2.0,5.0,10.0,17.0]
```

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 numeric library).

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

6.7 Functional Programming

6.7.1 Filtering

The numeric library provides counterparts of the Jython functional programming operations, such as '`filter`' which allows the selection of array elements based on a given criterion (e.g., elements 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
```

6.7.2 Reduction

The Jython '`reduce`' operation can be applied to a `Double1d` as follows:

```
v = Double1d([1,2,3,4,5])
print reduce(lambda x,y:x*y, v, 1)                # 120.0
```

Warning: If a lambda expression is applied to an array, remember that it is applied to the entire array and **not** mapped over the elements. This can lead to unexpected behaviour as in the following example:

```
u = Double1d.range(10)
print (lambda x: x>2 and x<4)(u)             #
[true,true,true,true,false,false,false,false,false,false]
```

This is equivalent to the following:
u > 2 and u < 4

The expression '`u>2`' results in a `Bool1d` array. The Jython '`and`' treats this as '`true`', as it is a non-empty list, and returns the result of the second expression '`u<4`', which is not the intended result.

One way of overcoming this problem is to *use the '`&`' operator instead of '`and`' to* give the intended result:

```
print (lambda x: (x>2) & (x<4))(u)      #
[false,false,false,true,false,false,false,false,false,false]
```

Another solution is to use Jython's '`map`' function and convert the result to a `Bool1d`:

```
print Bool1d(map(lambda x: x>2 and x<4,u))
```

6.8 The 'where', 'filter' and 'get' methods

The '`where`' method returns a `Selection` that contains the indices of elements that satisfy a predicate, whereas '`filter`' returns the actual elements.

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

return the actual elements

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

Note that `x[x.where(p)]` is equivalent to `x.filter(p)`.
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. Next to 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])
```

6.9 Advanced Tips

The underling 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 evalulate `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
```

6.10 Complex Arrays

The numeric library has a `Complex` class and a `Complex<N>d` class for <N>-dimensioanl arrays of complex numbers (<N>=1, 2 or 3).

```
z = Complex1d([1,2,3,4],[4,3,2,1])
print z                 # [1.0+4.0j,2.0+3.0j,3.0+2.0j,4.0+1.0j]
print z.real()          # Real part
print z.imag()          # 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.*

6.11 Multi-Dimensional Arrays

The numeric library supports 1, 2 and 3 dimensional arrays. 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. 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
```

Note: indexing multi-dimensional arrays is done differently in the numeric package and Jython. The following code examples show the syntax for Jython and for the numerics package. The reason for this is to allow slicing on multi-dimensional arrays which is technically not possible using the Jython syntax on numeric arrays.

```
x = [[1,2,3,4],[5,6,7,8]]
print x[1][2]                           # 7
print x[1][1:3]                         # [6, 7]

y = Int2d([[1,2,3,4],[5,6,7,8]])
print y[1,2]                            # 7
print y[1,1:3]                          # [6,7]
```

Individual elements or slices can be set as follows:

```
y[1,2] = 22                         # Set an element in place
y[0,1:3] = 42
print y                             # [
                                    # [1,42,42,4],
                                    # [5,6,22,8]
                                    # ]
```

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. The following example squares all the elements of row number 1:

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

6.12 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]
                         # ]
```

6.13 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 documention** and **demo programs**.

6.13.1 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 `Double1d`, `Double2d`, `Double3d` 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
```

6.13.2 Discrete Fourier Transform

A Disrete 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:

6.13.3 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 6.1: FFT of a modulated signal , with and without HAMMING smoothing

```
#standard import set for the chapter
from herschel.ia.numeric import *
from herschel.ia.numeric.function.Double1dFunctions import *
from herschel.ia.numeric.function.DoubleFunctions import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
from herschel.ia.numeric.function.MatrixFunctions import *

#additional imports
import java   #allows definition of pi below.
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))))
```

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

```
#standard import set for the chapter
from herschel.ia.numeric import *
from herschel.ia.numeric.function.Double1dFunctions import *
from herschel.ia.numeric.function.DoubleFunctions import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
from herschel.ia.numeric.function.MatrixFunctions import *

#import functions and convolution
from herschel.ia.numeric.function import Convolution
from herschel.ia.numeric.function.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:

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)
```

Design note: Convolution and associated filter functions are currently the only ones to supports keyword values, as an experimental feature. Other parameterized functions use a Java static method to construct a function object. Providing keyword support for such methods (rather than constructors) would require wrapping each Java function with a Jython function whereas, at present, no wrappers are required.

6.13.4 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.function import GaussianFilter
from herschel.ia.numeric.function import BoxCarFilter
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.

6.13.5 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.

A polynomial fitter (see section 6.13.6) can also be used to fit a set of data points and then interpolate between them.

6.13.6 Basic Fitter Routines

A more complete package of advanced data-fitting routines is discussed in Chapter 11. 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. The plotting package available for displaying the fit (PlotXY) is discussed more fully in Chapter 7.

#Example 6.3: Interpolation functions in IA

```
#standard import set for the chapter
from herschel.ia.numeric import *
from herschel.ia.numeric.function.Double1dFunctions import *
from herschel.ia.numeric.function.DoubleFunctions import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
from herschel.ia.numeric.function.MatrixFunctions import *

#import numeric functions
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)
# 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 = CubicSplineInterpolator(x,y)

# Nearest-neighbour interpolation
interp = NearestNeighborInterpolator(x,y)
```

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

#standard import set for the chapter
from herschel.ia.numeric import *
from herschel.ia.numeric.function.Double1dFunctions import *
from herschel.ia.numeric.function.DoubleFunctions import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
from herschel.ia.numeric.function.MatrixFunctions import *
#import the fitter functions
from herschel.ia.numeric.function import *
from herschel.ia.numeric.function.fit import *
#import plotting and colors
from herschel.ia.plot import *
import java.awt.Color

# 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) + Double1d.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.
```

6.14 Example Numeric 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?
7.3 How to setup the properties?
  7.3.1 How to modify properties?
  7.3.2 Plot properties
  7.3.3 Layer properties
  7.3.4 Axis properties
  7.3.5 How to use properties?
7.4 How to use plot in jython 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 colorful?
7.5 What about a complete example?

7.1 Introduction

This document 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 plot package. Moreover the basic concept is presented in order to support your first steps for simple visualization of two-dimensional data.

We will mainly talk about three classes: 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. The architecture of the `herschel.ia.plot` package is described in the package documentation.

Depending on how you work with plot, either writing scripts or designing your plot interactively, we recommend different approaches. For writing scripts you have to use the command line interface. If you design your plots interactively it will be easier to use the graphical interface to manipulate plot properties.

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

statements run individually in the command line window of `jconsole` or in the top scripting pane of `jconsole` (use the run arrow – see the chapter on jconsole for more details on using `jconsole`):

```
from herschel.ia.dataset import TableDataset
from herschel.ia.dataset import Column
from herschel.ia.numeric import Double1d
from herschel.ia.numeric.function.DoubleFunctions import EXP
from herschel.ia.plot import PlotXY
from java.awt import Color
```

The first import command allows us to use `TableDatasets` (see Chapter 5). The second allows the use of Column. We shall be using Doubles for data types in our examples and we will be concentrating on the program `PlotXY`. Lastly we allow various colors to be made available for our plots. **THESE IMPORT COMMANDS MUST BE INPUT BEFORE RUNNING THE EXAMPLE SCRIPTS IN THIS CHAPTER**.

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.

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

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

Double1d Data which is a one-dimensional array of doubles. You can construct a plot with only one Double1d that will be taken as y-data. The x data is automatically generated as a sequence of integers starting with 0. Alternatively, you may use two Double1ds. One as x-data and the other as y-data.

TableDatasets contain data represented 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 plotted in an overplot mode on different layers.

Example 7.1 shows how to plot a simple one-dimensional array and a `TableDataset` that contains two columns "X" and "Y".(NOTE: all examples are hyperlinked to files which can be placed in the upper `jconsole` window – run line-by-line to see what happens). DON'T FORGET THE IMPORT OF PACKAGES (see introduction).

Figure 7-1 shows the two plots that were generated by the above example where you can clearly see that the layer names and the values on the abscissa differ.
within `jconsole`

#Example 7.1 for PlotXY.

```
x = Double1d.range(20) / 10

#x is set up to be an array with the range of numbers = 0..19 divided by 10
#Placing the Double1d 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(x)

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

p = PlotXY(y = e, layername = "Layer1")

#this plots the exponent and gives a name to the layet it is plotted in. We can
refer
#to this layer at later times if we, e.g., want to change plot colors. In this
case, the x axis
#numbers are simply the array index of the number in e.

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

t = TableDataset()

#create a table - empty to start

t["X"] = Column(x)

#now we 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 "x" - see first line.

t["Y"] = Column(e)

#similar for the next column

p = PlotXY(dataset = t, lineColor = Color.blue, \
linetype = PlotXYCompositeRenderer.RECTANGLE)

#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 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.
```

A special way of constructing a plot is possible. It uses the notation *variablename* = *variablevalue* for any subset of possible variables available to PlotXY.

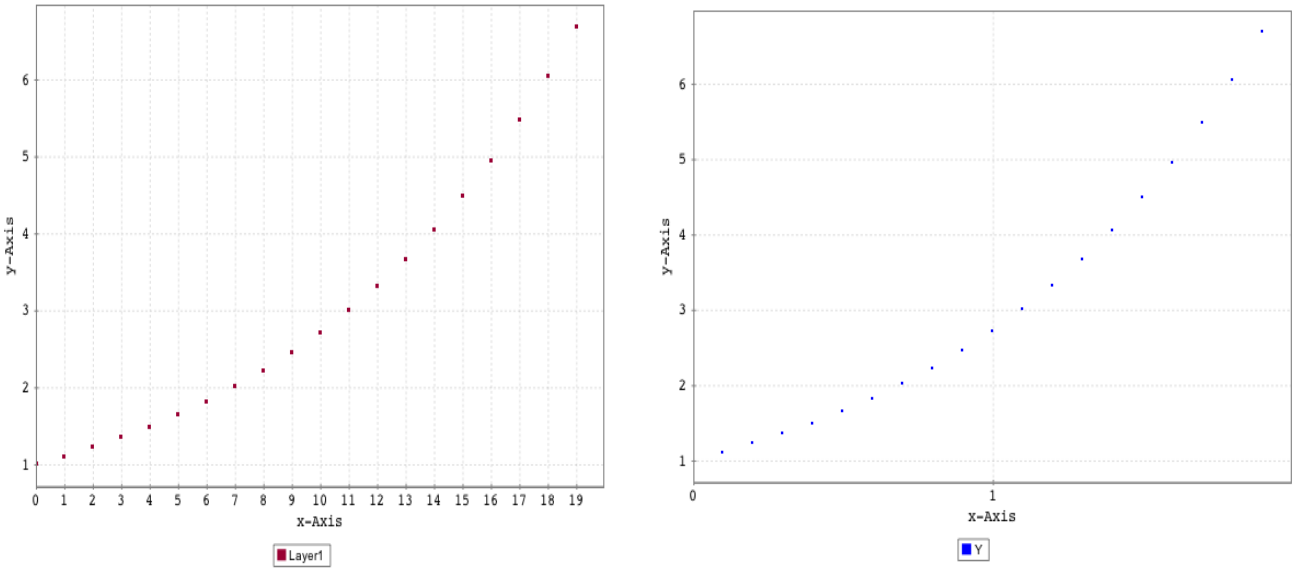

Figure 7-1 The two plots produced from example 1. Note that the first graph has an x-axis with only the index plotted.

Constructor for 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:

```
PlotXY(dataset = ds, plotIsVisible= false)
```

or

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

In the latter case we just get a window in which a plot can 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 the properties

Plot properties allow the definition of some basic properties like *colors*, *linetypes* etc. as your personal preferences. To setup do:

- 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 uses for all layers and axis and not only for the one you have chosen in the moment of pressing the button.

apply:
applys the properties of the visible register card to the plot object.

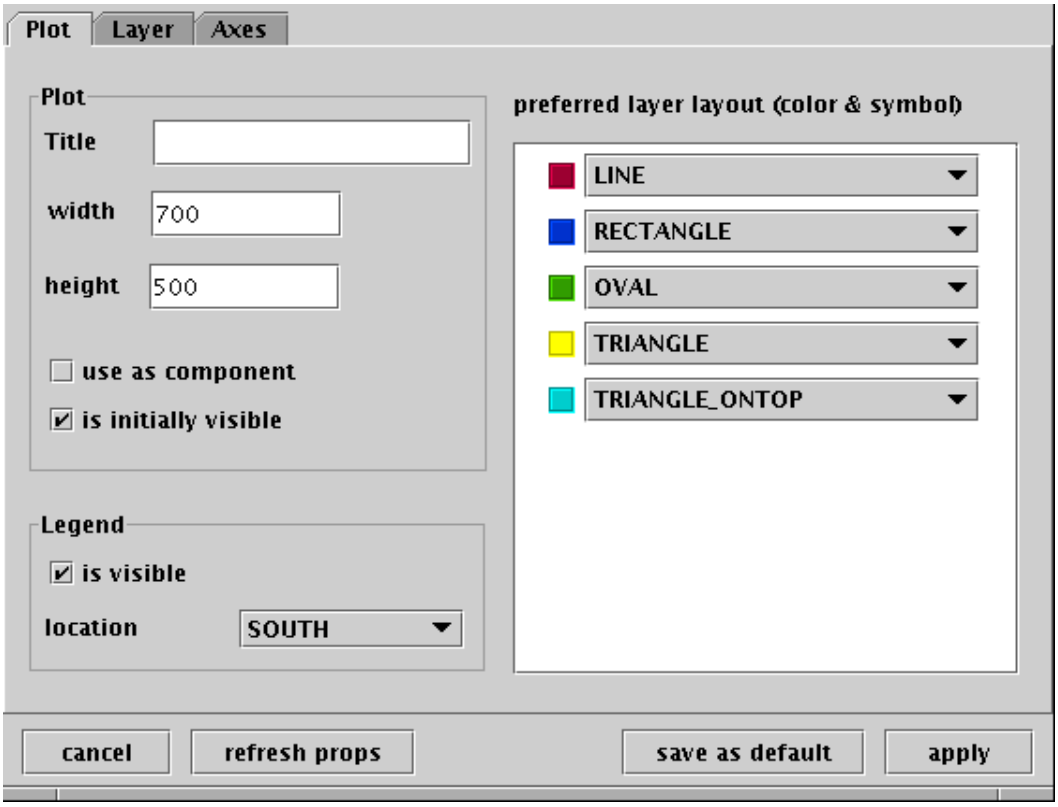

Figure 7-2: The PlotXY properties graphical interface.

7.3.2 Plot properties

Most of the properties are self explaining. Simply change them, hit apply and see the result.
The plot properties offer a section called "preferred layer layout (color & 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 color of an existing layout click into the colored square.
to change the `linetype` choose it from the 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).

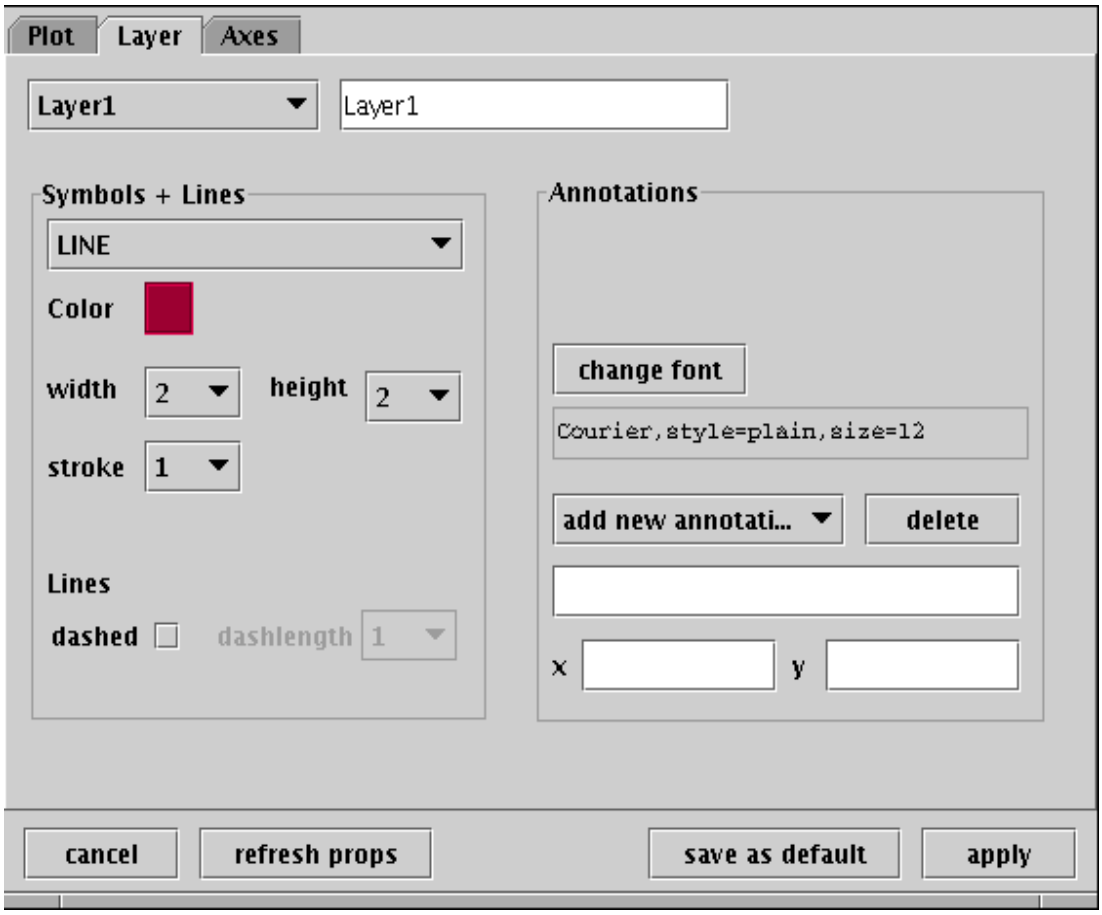

Figure 7-3: Graphical interface for PlotXY layers

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.

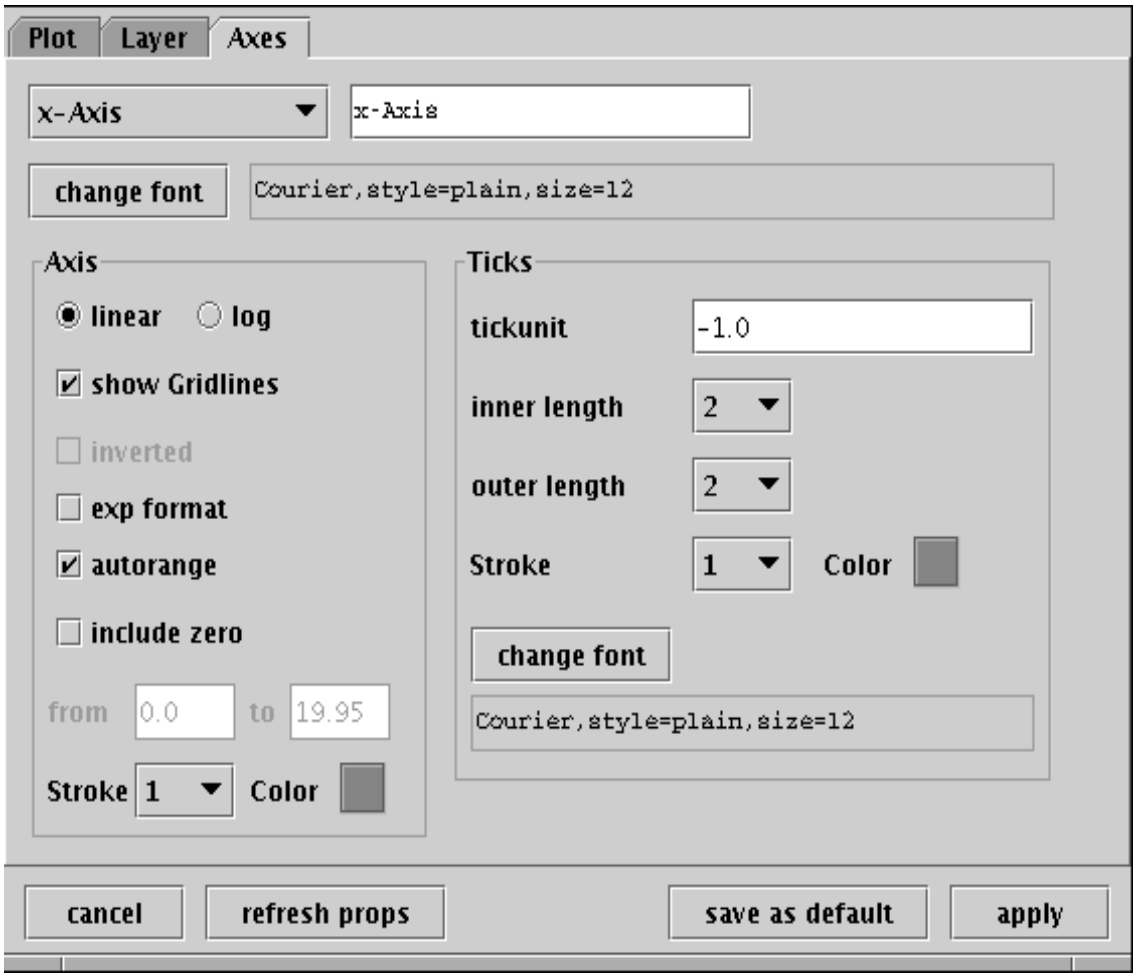

Figure 7-4: The Axes properties pane for PlotXY properties.

7.3.5 How to use properties.

The result of property setup procedure (with a defined set of properties) is the following (Example 7.2):

#Example 7.2: command line control of properties

```
#This is continuation from the previous example.
#A plot of the type  p = PlotXY(x, y,"layer1")

p.props()
#allows graphical interface property setup
p.getLayer("Y").remove()
#removes layer containing plot of "Y"
l2 = p.addLayer(x, x*x, "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 = p.addLayer(x, 2*x*x, "layer3")
#adds another layer to the plot "p"
l3.setSymbolSize(7)
#…and changes symbol size on this plot too!
```

If a constructor for `addLayer` is used that defines neither color nor linetype, the predefined properties are used for subsequent layers. This is shown in line 3 and 6. The result is shown in Figure 7-5.

If you have a look at the predefined layer colors and linetypes you see that the first definition is dark red color and lines, the second blue color and rectangles.
If color and linetype are specified in the constructor, they are used as specified.

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

The result of this line is shown below in Figure 7-6.

According to the predefined layer layout the symbols should have been green ovals. But layer4 shows an orange line like it has been specified in the addLayer method.

7.4 How to use plot in jython 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)`)

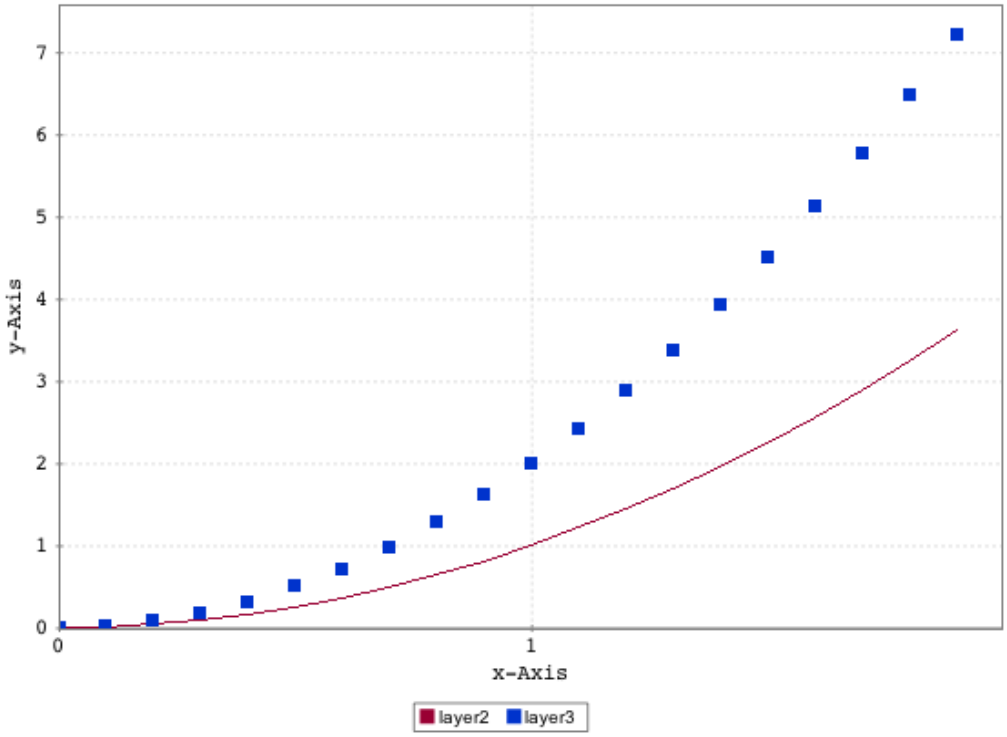

Figure 7-5: Plot obtained using example 7.2

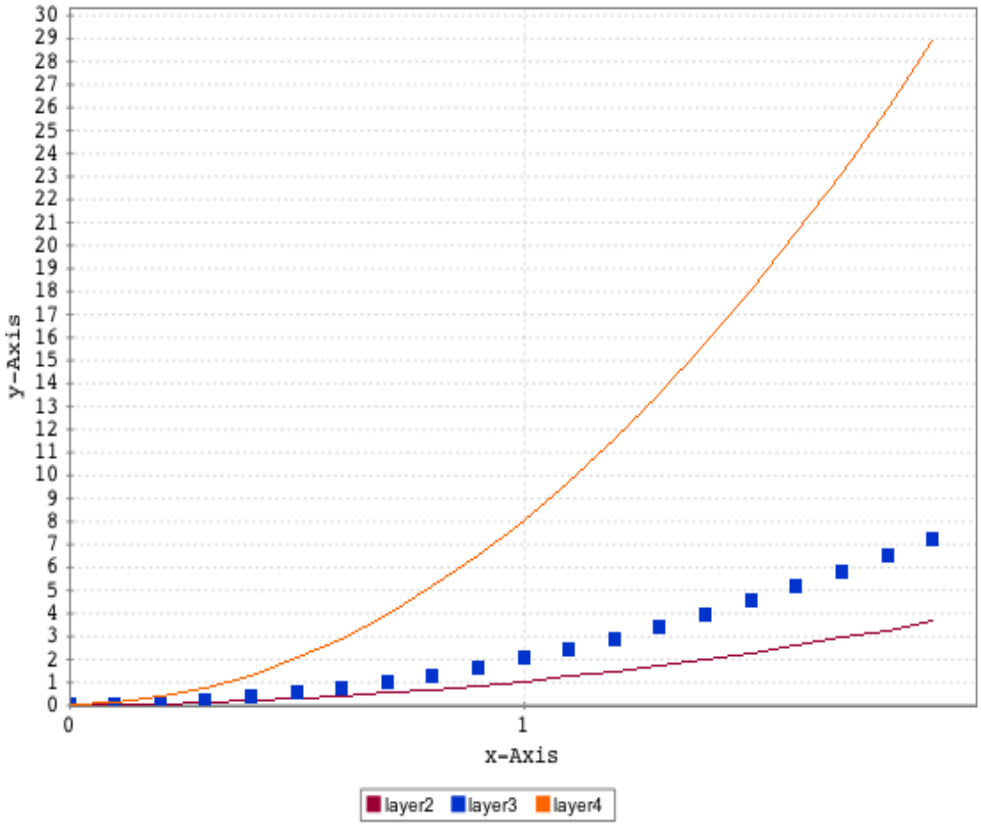

Figure 7-6: Adding in 4th layer gives the orange curve (see text).

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, 7.3, 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 how 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.

Please note that we used the `None` Object as `x`. This results in automatic generation of x values. 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 Example 7.3.

Some of the more useful methods that work on layers are listed in Table 7-1: Listing of methods of layer manipulation.. For a complete reference of the methods that can be used to manipulate layers please consult the API documentation of `Layer`.

chap7_example3.txt

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

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

Methods to change the appearance of the datapoints:

l.setLine()	change the plot to a line plot for the specified layer
l.setSymbol()	change the plot to a scatter plot for the specified layer
l.setSymbolColor(*color*)	set the color of the symbols and lines for the specified layer
l.setSymbolSize(*size*)	set the size of the symbols for the specified layer (only for scatter plots)

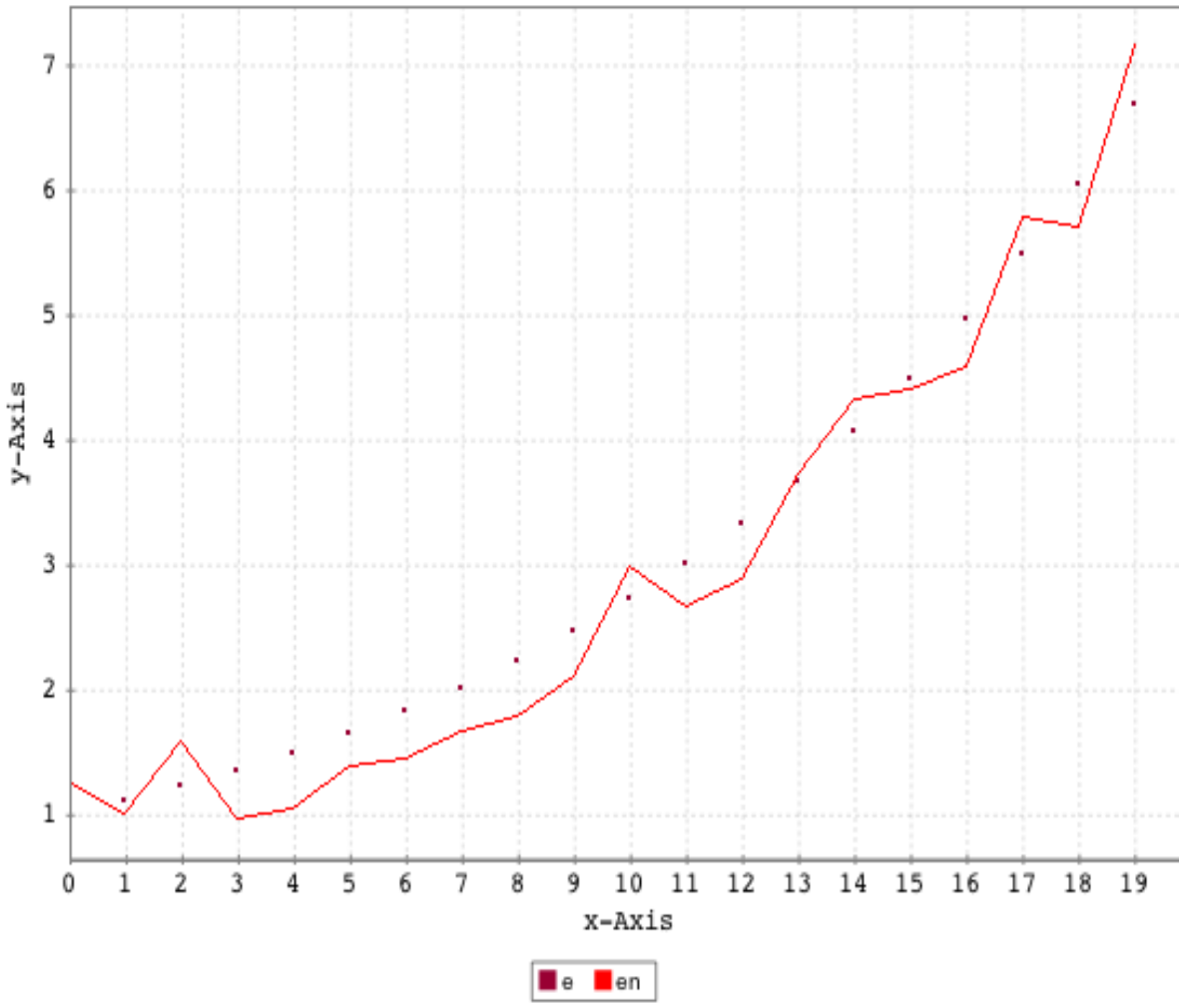

Figure 7-7: Plot showing the manipulation of layers from the command line. The above plot comes from the script given in example 7.3.

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 color. Lines can be solid or dashed with their own color. 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. Lets continue with `example 7.3` and make some changes to the axes.
Example 7.4

chap7-example4.txt

Figure 7-8 shows the results from example 7.4.

Some of the more useful methods that work on axes are listed below. 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`.

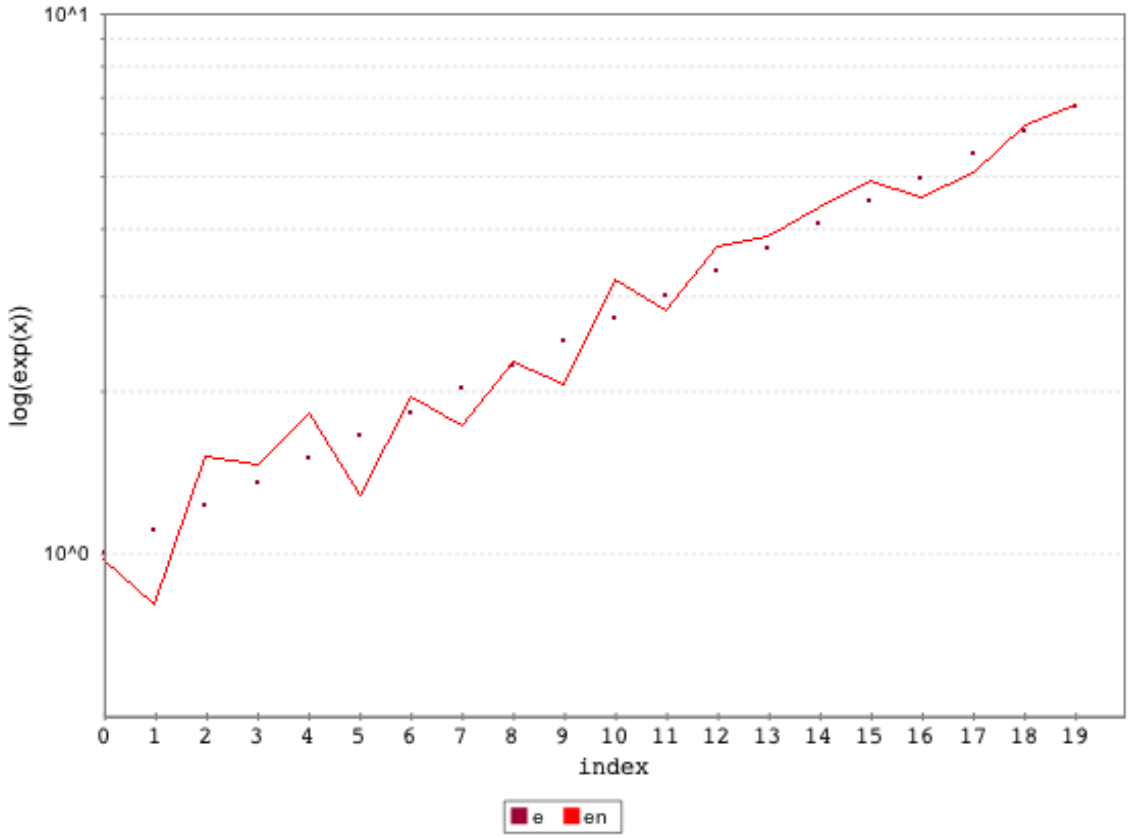

Figure 7-8 The results from example 7.4.

a = p.getAxis(*axis*)	get the `Axis` object to do direct manipulations on the specified axis
a.setLinear()	re-establishe a linear scale for the specified axis
a.setAutoRange()	adjusts the range of the specified axis so that all datapoints will be shown
a.setRange(*lower*, *upper*)	set the range of the specified axis to values between lower and upper
a.showGridLines(*flag*)	show grid lines for the specified axis if `flag` is `true`, hide the grid lines if `flag` is `false`
a.setLabelFont(*font*|*size*)	set or resize the font used for labelling the specified axis
a.setTickLabelFont(*axis*, *font*|*size*)	set or resize the font used for labelling the tikmarks 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 7.4 and add proper names for layers, annotate some datapoints and put a title on top of the figure (see Example 7.5). The example below also shows how to extract the `Layer` objects from the plot in order to manipulate them directly.

chap7-example5.txt

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 7-9.

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 where to find these methods.

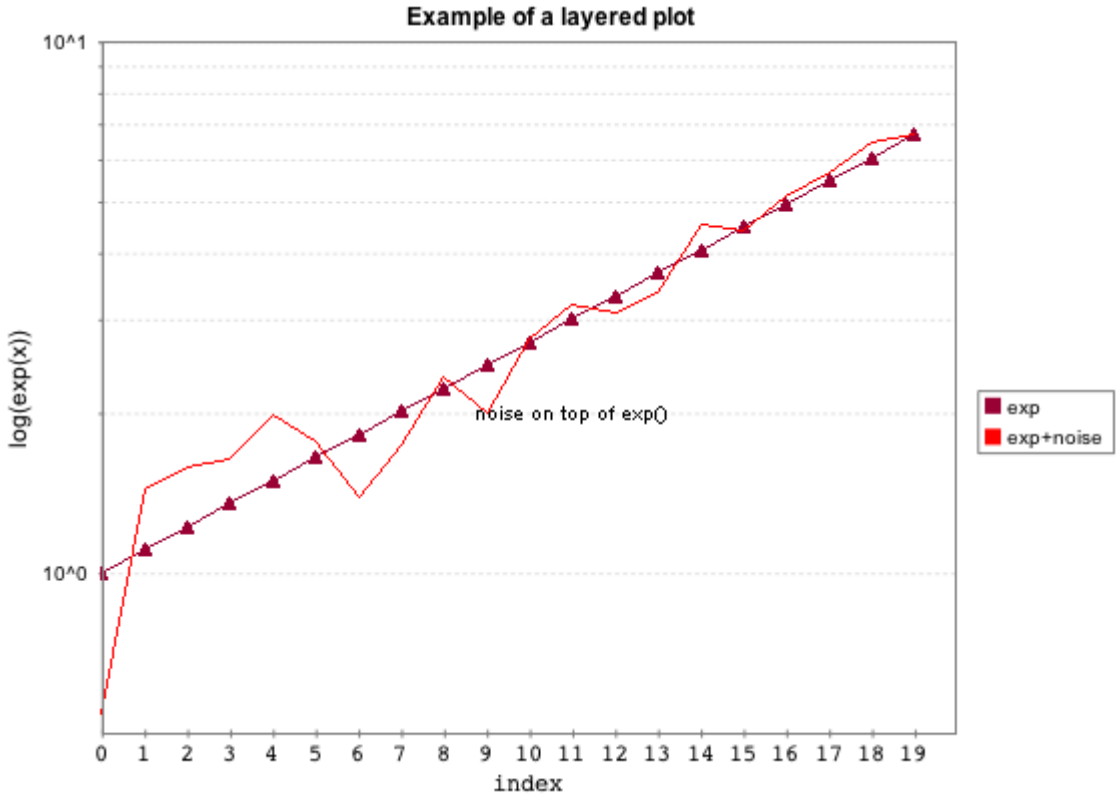

Figure 7-9: The results of Example 7.5 in the text.

7.4.4 How can I make my plots more colorful?

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

To specify a color 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.

The default names for colors are: black, blue, cyan, darkGray, gray, lightGray, green, magenta, orange, pink, red, white and yellow (all preceeded by `Color.`). Another easy way to use a custom color is to specify the red, green, blue value in ranges from 0-255. Color(red, green, blue).

7.5 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? (histogram, rotate, scale, translate)
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
8.8.3 Download examples

8.1 Introduction

This document 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 please the reader is referred to the API documentation for the image package.

Please note that the display functionalities of the HCSS are still **under development**. This means that the interface can still change.

> **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 herschel.ia.image import ImageDataset
from herschel.ia.image import Display
from herschel.ia.image import Wcs
from herschel.ia.image import Rotate
from herschel.ia.numeric import Double2d
from herschel.ia.numeric import Bool2d
from nT.quantity import FluxDensity
from nT.quantity.constant import ASTRONOMICAL
```

```
from java.awt import Font
```

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

8.2 Using ImageDatasets

An ImageDataset is a special type of dataset composed of :

- the image : described as a `Double2d`
- the errors of the image : described as a `Double2d`
- a mask : described as a `Bool2d`

The ImageDataset also holds information to do coordinate conversions and information of the wavelength at which the image was taken.

When constructing an `ImageDataset`, you should usually first construct a WCS object and the `Double2d`'s and `Bool2d` needed for the image.

Example 8.1 shows how to construct an `ImageDataset` with an image of 60x40 pixels, with 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 and Dec = -22d30m
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
myMask.set(55, 35, False)
#…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*. For more possibilities on `ImageDataset`, see the `ImageDataset` javadoc.

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.

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.

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).

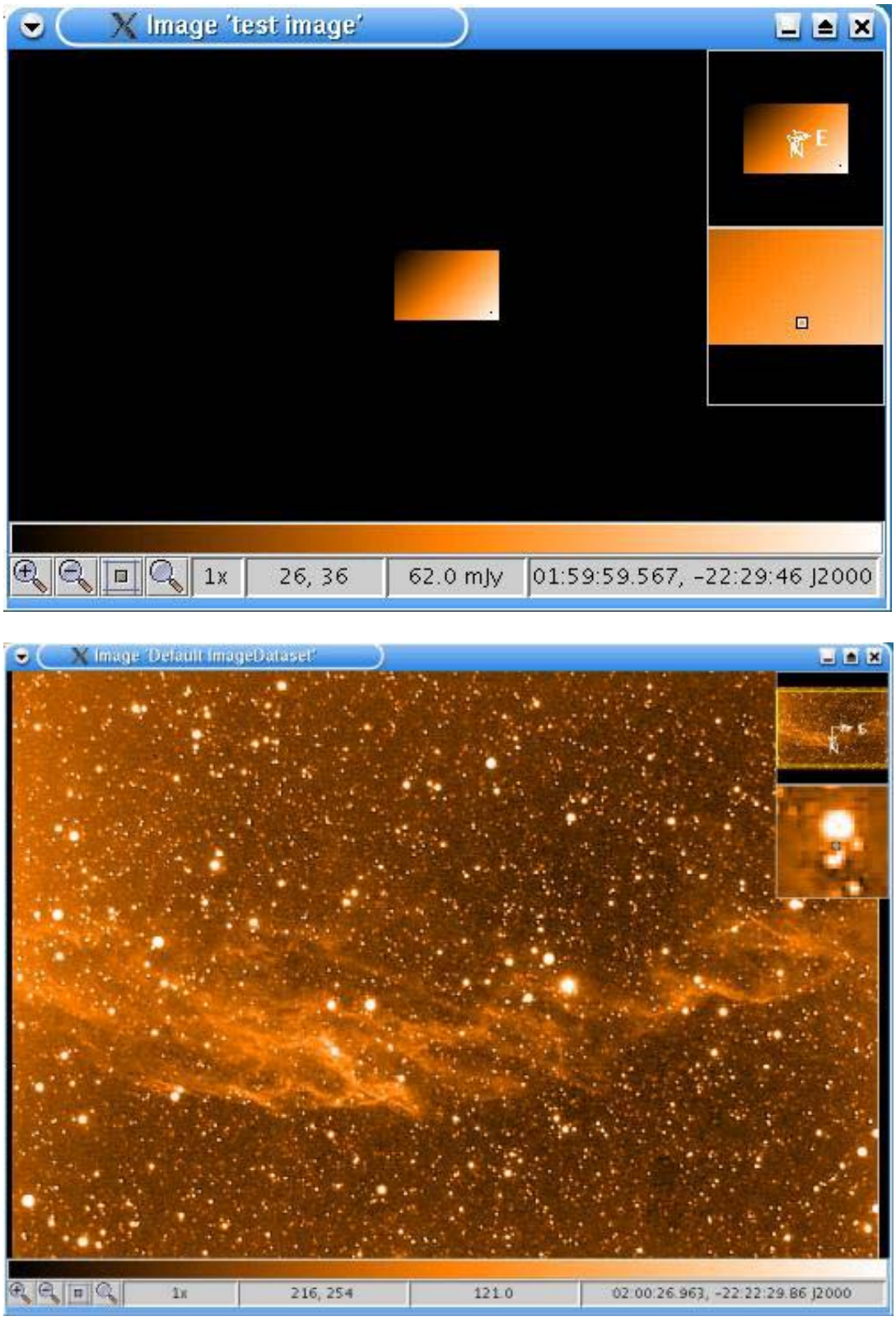

Figure 8-1: Display of image created in IA (top) and of imported JPEG file (bottom) from Example 8.1.

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 :

getImageCoordinates(*ra, decl*)	Returns the image coordinates corresponding with the given sky coordinates
getSkyCoordinates(*x, y*)	Returns the sky coordinates corresponding with the given pixel coordinates
getIntensity(*x*|*ra*, *y*|*dec*)	Returns the intensity of the pixel at the given (Sky or Image) coordinates
setColorTable(*colortableName, intensityName, scaleName)*	Set the *colortable* of the image
setCutLevels(*min, max*)	Set the cut levels between *min* and *max*
setZoomFactor(*zoomFactor*)	Zoom by the given *zoomfactor*
addAnnotation(*annotation*, *x*|*ra*, *y*|*dec*)	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("Test annotation", 506, 300)
#provides an annotation at coordinate (506,300)
```

The result can be seen in Figure 8-2.

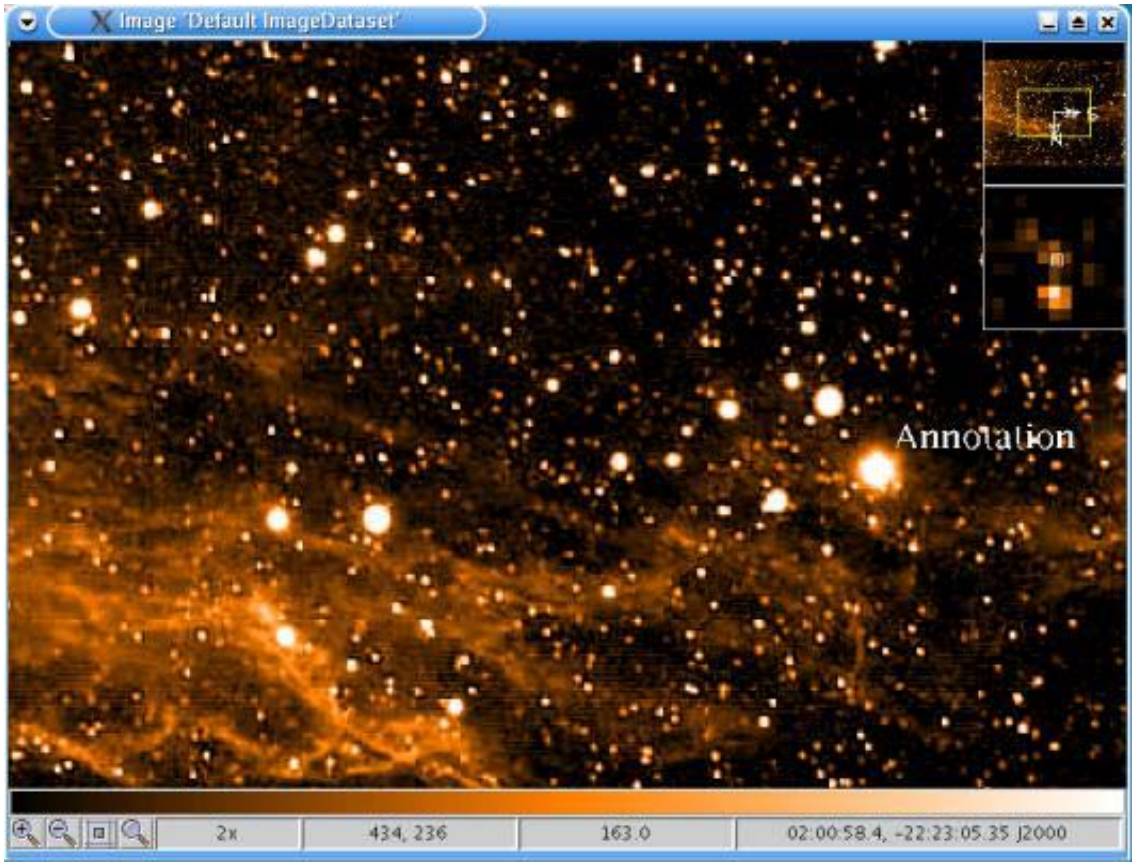

Figure 8-2: Results from Example 8.2 showing the get sky coordinates, zoom and annotation features of Display.

8.5
8.6 How can I use Operations on my images?

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

- **Histogram** : returns a tableDataset with the histogram of the image.
- **Rotate** : Rotates the image.
- **Scale** : Scales the image.
- **Translate** : Translates the image.

More information can be found in the javadoc of the operations.

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

```
Display(myImage2.apply(Rotate(angle=30.0)))
```

This example rotates the image over 30.0 degrees (clockwise). The result is shown in Figure 8-3.

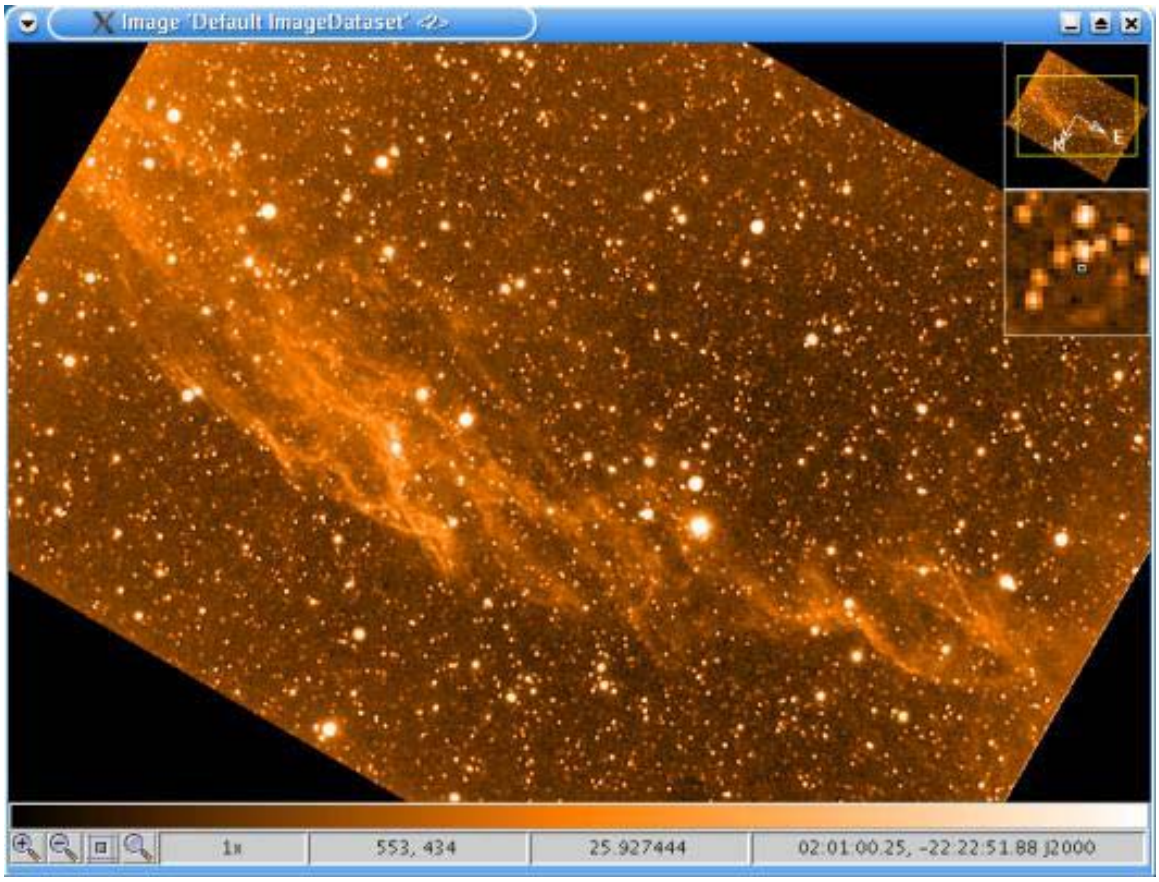

Figure 8-3: Illustration of how an image is rotated by +30 degrees (clockwise) using the IA display package.

8.7 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
from herschel.ia.image import Display
from herschel.ia.numeric import Int2d
#import the necessary packages

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 8-4 shows the results from running Example 8.3.

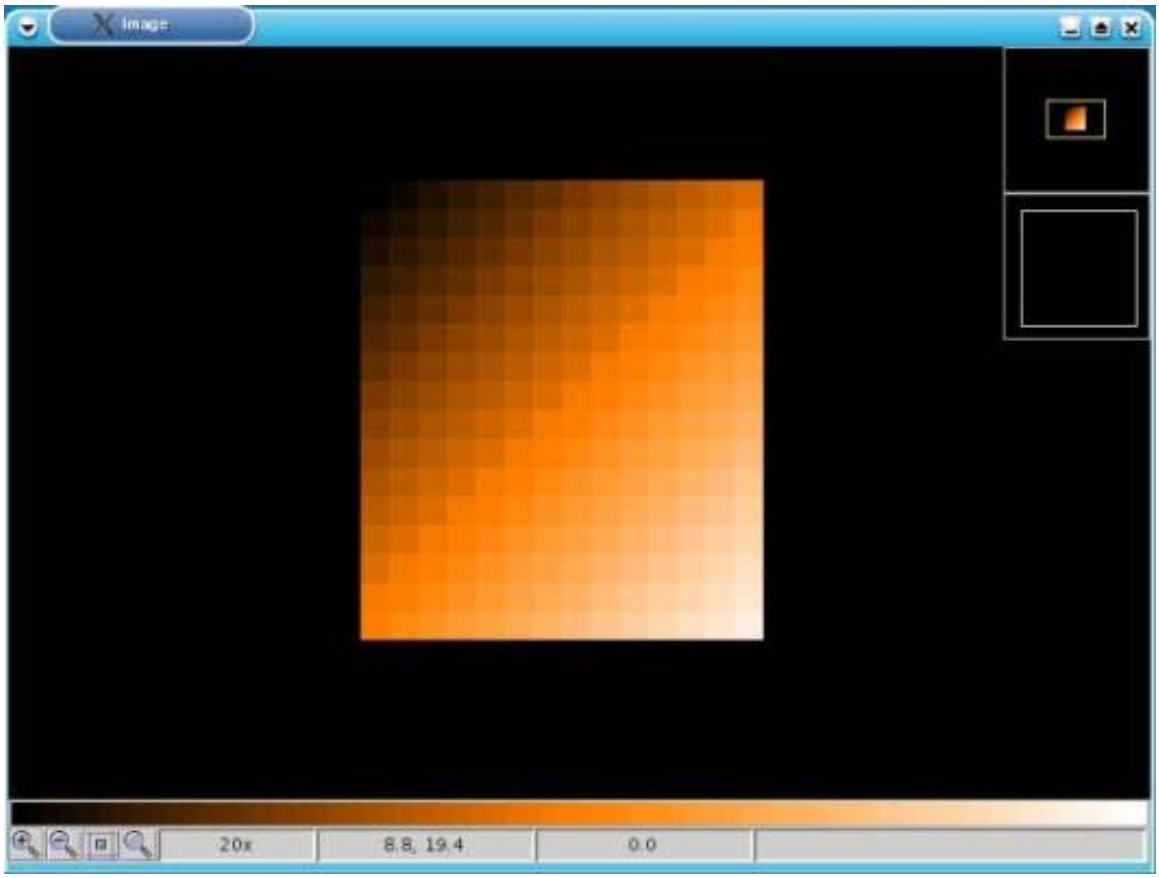

Figure 8-4: Example 8.3 – display of image created from user-supplied numeric 2D array.

8.8 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 8-5 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, ...).

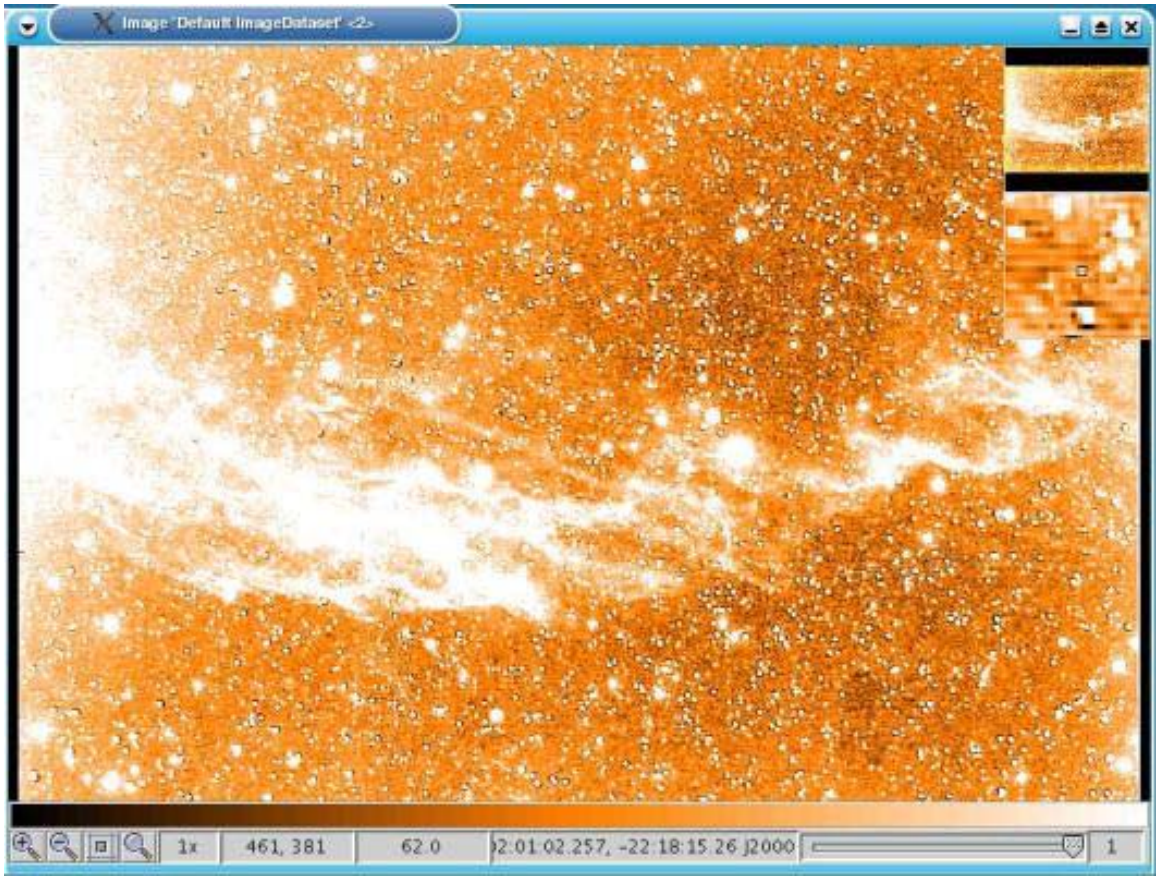

Figure 8-5: The use of layers in Display. Swapping between layers is performed using the slider at bottom right.

8.9 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.

8.9.1 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

myDisplay.addAnnotation("Veil nebula", 321, 224)
#Places annotation at position (331, 224) on image 'mydisplay'
myDisplay.setAnnotationFont(321 , 224, Font("Dialog", 0, 24))
#change font type and size for the image 'mydisplay'
myDisplay.setAnnotationFontColor(321, 224, Color(0, 0, 255))
#annotation color changed
myDisplay.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.
myDisplay.addGreekAnnotation("a = 12.34, d = +30.30", 100, 500)
#adds a position label with greek letter notation at position (100,500)
myDisplay.setAnnotationFont(100, 500, Font("Dialog", 0, 18))
#change font of annotation of at (100, 500)
myDisplay.setAnnotationFontColor(100, 500, Color(0, 0, 0))
#…and change its color to black too
```

The result is shown Figure 8-6.

8.9.2 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 = myDisplay.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 8-7.

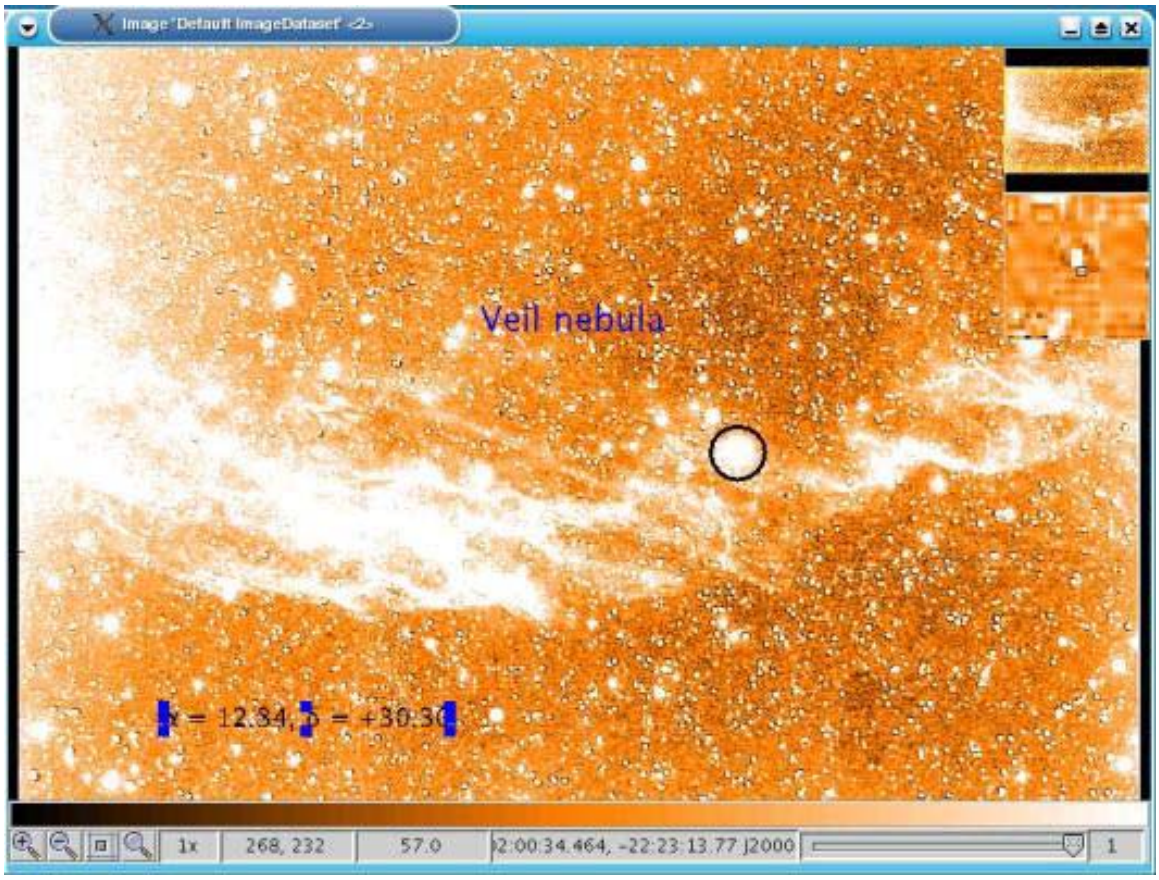

Figure 8-6: Output from the use of Example 8.4 illustrating how annotations can be added in various forms.

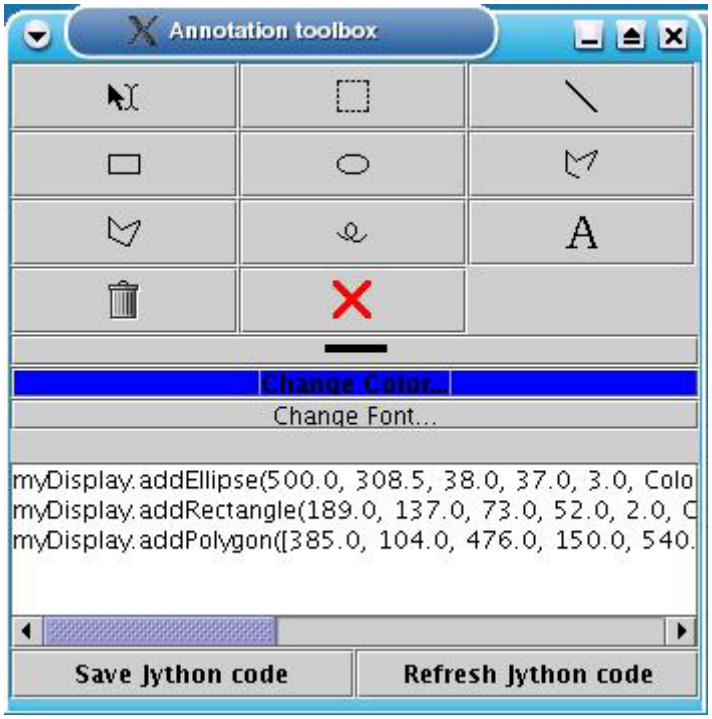

Figure 8-7: The annotation toolbox made available for "myDisplay"

The icons in the annotation toolbox appearing in Figure 8-7 have the following usage (from left to right and from top to bottom) :

- Select annotation
- Select all annotations in a region
- Draw a line
- Draw a rectangle
- Draw an ellipse
- Draw a polyline
- Draw a polygon
- Draw with the free hand on the image
- Add a text annotation
- Remove the selected annotation(s)
- Remove all annotations

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.

8.10 Download examples

You can download the example file and the two example python scripts that are described in this document.

- ngc6992.jpg : The example picture of the veil nebula.
- Example 1 : A python script with the first example.
- Example 2 : A python script with the second example, displaying an Int2d.

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
9.8.1 herschel.ia.dataflow
9.8.2 herschel.ia.dataset
9.8.3 herschel.ia.demo
9.8.4 herschel.ia.doc
9.8.5 herschel.ia.image
9.8.6 herschel.ia.io
9.8.7 herschel.ia.jconsole
9.8.8 herschel.ia.numeric
9.8.9 herschel.ia.plot
9.8.10 herschel.ia.task
9.8.11 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. 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.

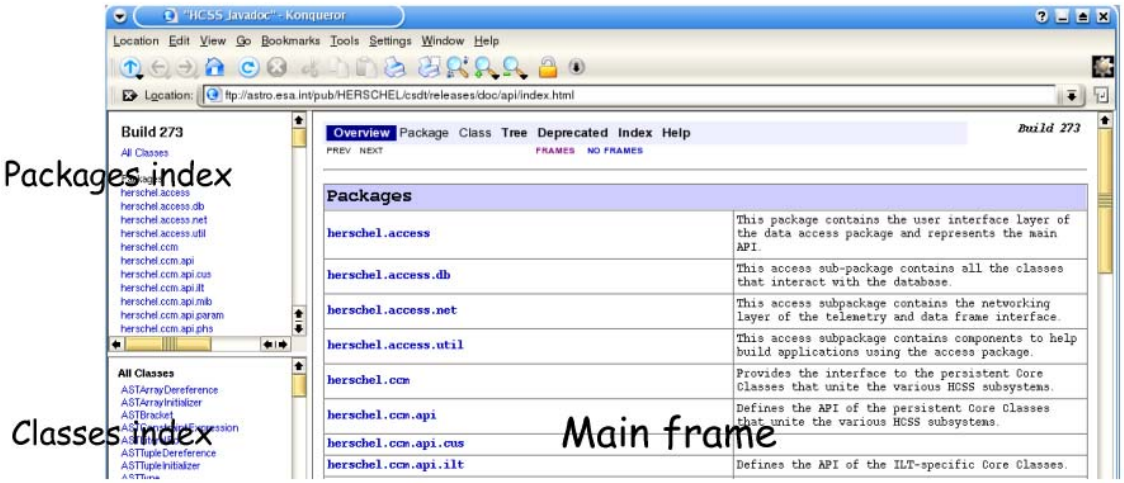

Figure 9-1: Web browser page of JavaDocs top level frame

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 9-2shows 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 shows 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)*.

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.

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

TableDataset() Constructs an empty table.
TableDataset(java.lang.String description) Constructs a TableDataset with a description.
TableDataset(TableDataset copy) Constructs a TableDataset that is a deep copy of specified argument.

Method Summary

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

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. A Dataflow HowTo document is available describing dataflow setup and use.

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.image

herschel.ia.image – package containing classes for handling images. The Display capabilities from this package were discussed in Chapter *. 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 WorldCcoordinate System to an ImageDataset

Sub-package:
herschel.ia.image.gui – classes that handle GUIs. These should ONLY be called from within the Display program.

9.8.6 herschel.ia.io

herschel.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:
herschel.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.
herschel.ia.io.ascii – Allows the input/output to and from ASCII files from within the IA environment.
herschel.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 ** for information about the setup and use of databases with IA.

9.8.7 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*).

9.8.8 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.

9.8.9 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.

9.8.10 herschel.ia.task

herschel.ia.task – This package explains how to set up a IA "task". 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. More on tasks within IA are available in the IA Task HowTo document.

9.8.11 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 Swing components within IA are discussed in Chapter 11 in association with database access.

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 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 document describes how to read and write tabular ASCII and FITS data files within IA. Illustrations are provided that can be run in the `jconsole` environment.It should be noted that not all FITS files created outside of the HCSS may be able to be imported.

10.2 Getting Started with ASCII Import/Export

Assuming you have succesfully started `jconsole`, then the following packages should be imported. These allow ASCII import/export and the use of TableDatasets within your IA environment.

```
# ascii tools
    from herschel.ia.io.ascii import *

# table dataset et.al.
    from herschel.ia.dataset import *
```

More general documentation on ASCII IO is available here.

10.3 Basic ASCII 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:

parser=yourParser	Changes the line parsing behavior at import.
formatter=yourFormatter	Changes the line formatting behavior at export
template=yourTemplate	Specifies how to interpret raw cell data.

10.3.1 Import Parsers

A parser controls how to break-up a line into table cell data. All parsers share the following attributes:

ignore=expression	Lines containing expression are ignored. By default the expression skips lines starting with a hash, possibly preceded by whitespace: `"^\\s*#"`
skip=value	First number of lines can be skipped by specifying a value>0. Default is 0.
trim=0|1	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 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:

delimiter=character	The character used to distinct cells within a line of data. Default is a comma (**,**).
quote=character	The character 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:

sizes=array	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:

commented=0|1	States whether comments will be allowed in the output or not, default=0 (false).
commentPrefix=string	Prefix used for all comments, default="# ".
header=0|1	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 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:

names=array	Specifies names that will be attached to the columns.
types=array	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".
units=array	Specifies the units of all columns. Uses SI units, and units that are accepted for use with SI.
descriptions=array	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

Example 10.1 shows how to handle ASCII tables in the IA environment. In order to run the program the user will also require an input file, which is available here.

```
# Example 10.1 - Handling ASCII tables
from herschel.ia.io.ascii import *

# --- 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 Witdh format, with header info:
ascii.formatter=FixedWidthFormatter(sizes=[8,16,8,30]);
ascii.save("table.out2",table)

# --- importing it back requires Fixed Witdh parser
ascii.parser=FixedWidthParser(sizes=[8,16,8,30])
table=ascii.load("table.out2")

# --- export using Fixed Witdh 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, you have to import the facilities needed to create products as well as to create FITS files:

```
# the FITS archive tool
   from herschel.ia.io.fits import *

# product & dataset et.al.
   from herschel.ia.dataset import *

# numeric array data classes
   from herschel.ia.numeric import *
```

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 a 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")
```

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 nested datasets, stores it into a FITS file, and retrieves it again.

10.6.3 Caveats

The current implementation can not yet read fits files that are generated by another package than package *herschel.ia.io.fits*.

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

#Example 10.2: FITS IO from within Herschel IA

```
from herschel.ia.io.fits import *
from herschel.ia.dataset import *
from herschel.ia.numeric import *
from herschel.ia.numeric.function.DoubleFunctions import *

from java.lang.Math import PI
from  nT.quantity.constant.ENERGY import ELECTRON_VOLTS
from  nT.quantity.constant.TEMPERATURE import KELVINS

# --- construction of a product. note this is only for demonstration
#     purposes. For more infomation, please see the demo's provided in:
#     *  herschel.ia.dataset
#     *  herschel.ia.numeric
points=50
x=Double1d.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)
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="As a composite, I contain three \
datasets!")
c.meta["exposureTime"]=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("This keyword you would not \
find in the 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"]
```

Herschel IA Chapter 11

11 Setup and Use of Databases

Chapter 11 Contents

11.1 Introduction
11.2 Starting Up A Database:
  11.2.1 Unix
  11.2.2 Windows
11.3 Schema Evolution
11.4 Using an existing database
  11.4.1 Initializing an old database
  11.4.2 Schema Tool commands
11.5 Initializing the Database:
11.6 Quick Database Creation
11.7 Providing Database Access for an IA Session
  11.7.1 Editing Properties File
  11.7.2 Using the Propgen Tool
11.8 Browsing a Database
11.9 Getting Data Frames From a Database
  11.9.1 Command Line Access to Data Frames
  11.9.2 From Database to ASCII File
  11.9.3 Downloading Dataframes from a Database Using a GUI
  11.9.4 Accessing Housekeeping Data
11.10 Removing a Database

11.1 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 install a database capability on your own computer/laptop. Unix and Windows versions are available.

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.

11.2 Starting Up A Database:

11.2.1 Unix

The following commands create a database within the HCSS and make it available for use.

```
>> 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.

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.

11.2.2 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 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 configuration files.

11.3 Schema Evolution

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 multiple databases, then each database must be initialized in this way.

It is essential to apply the schema to new databases by running the command '`schema_tool -i`' (see below).

11.4 Using an existing database

The Versant database stores objects defined by database *schema*. These *schema* correspond with the class definitions of the persistence-capable classes in the HCSS CCM. The persistent objects in the database are instances of these schema. When a new release contains certain kinds of changes to the persistent class definitions, it may break compatibility with existing data stored in the database. In such cases, it is necessary to perform a *schema evolution*, to convert the existing database to use the new schema and to convert all persistent instances of the schema. 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 '`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 firewall to allow 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` # Evolve the database

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.

11.4.1 Initializing 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

schema_tool -i dbname@server
```

If the HCSS installation has more than one database, each database must be initialized

11.4.2 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 which has a different schema version then 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*:

11.5 Initializing the Database:

On Unix or Windows:

```
initv <dbname>
```

11.6 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*.

11.7 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 a 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.

11.7.1 Editing Properties File

There are two ways of setting up your properties to allow access to a particular database during an IA session. First, the file *hcss.props* (on Windows) or the file *myconfig* (on Unix) can be edited.

On Windows, the *hcss.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/.hcss.

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.

11.7.2 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 (see Chapter **).
The command

```
> propgen
```

will start up the `propgen` tool (see Figure 11-1). Using the tabs at the top of the `propgen` screen, the user should click on "**Variables**".

Now edit the variables **var.database.server** (input *servername*) and **var.database.devel** (input *dbname@${var.database.server}*).

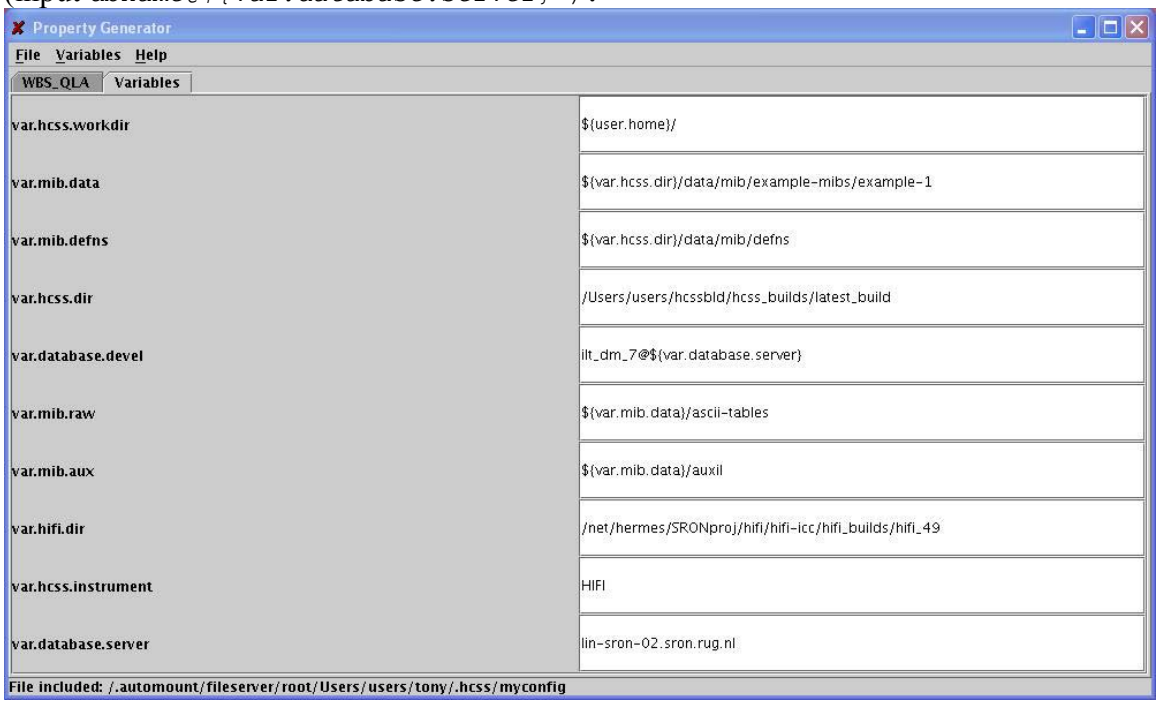

Figure 11-1: The propgen window available at the SRON-G HIFI site.

11.8 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 11.1 allows the user to view the database connected to.

```
#Example 11.1
from herschel.access import *
from herschel.access.util import *

# The observation chooser can not be used interactively at present.
# Just there to view contents.
print "Just a viewer, not a selector for now. To see obsid scroll \
all the way to the right"
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.

11.9 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 the 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 the DataSelector GUI

11.9.1 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 11.2 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.

```
#Example 11.2 - basic command line for getting
#data frames from a database
from herschel.ia.dataset import *
from herschel.ia.numeric import *
from herschel.ia.io.ascii import *
from herschel.ia.numeric.function import *
from herschel.access.util import *
from herschel.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 obsid = 180
dfaccess.setObsid(180)
#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 the 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
  table[""+`j`]=Column(datad)
```

This brings in the 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.

11.9.2 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 11.2.

```
#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 11.3 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 11.3 - database to ASCII tables for multiple spectra
from herschel.ia.dataset import *
from herschel.ia.numeric import *
from herschel.ia.io.ascii import *
from herschel.ia.numeric.function import *
from herschel.ia.numeric.function.DoubleArrayFunctions import *
#import Java swing for GUI components
import javax.swing as sshwing
from herschel.access.util import *
from herschel.access import *

# These are the obsid values for the set of data you want.
# The data will be placed in a comma-delimited table.
#prompt the user for first obsid
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 = Double1d(df.getFrame())
   table[""+`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_"+`i`,table)
```

11.9.3 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 the *DataSelector* tool. This is available via the *ProcessConnect* command. Example 11.4 provides a downloadable script that uses just such an interface for obtaining HIFI dataframes.

```
#Example 11.4 - GUI interface to a database
from herschel.ia.dataset import *
from herschel.ia.dataflow.IaProcessImpl import *
from herschel.ia.numeric import *
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.
#need to also get HK data out
    self.pc = ProcessConnect("pc")
    self.out = self.pc.getConnector("df-output")
    self.a = java.lang.reflect.Array.newInstance(HifiDataFrame,1000)
    self.out.pass(self.a)
    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):
    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[""+`j`] = Column(datad)
    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 11-2– 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 11-3). 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 11-2: The play tab opened for the HIFI dataselector tool

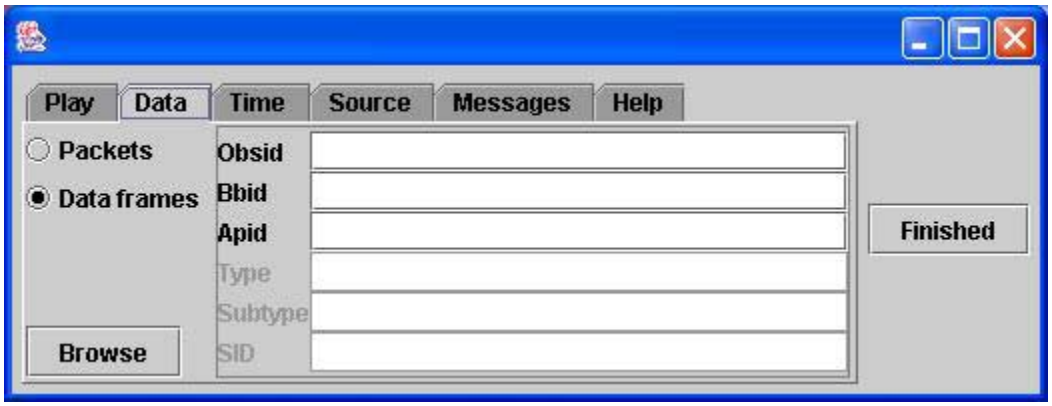

Figure 11-3: The Data tab of the HIFI dataselector tool. The example program works with Data frames only, at present.

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.

11.9.4 Accessing Housekeeping Data

Assuming you have access to a database who's 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.

Example 11.5 illustrates basic housekeeping packet access for an observation with an obsid of 180. The end product is a table with two columns, time in the first column and the housekeeping parameter value (raw) in the second column.

```
#Example 11.5 – basic HK packet access
#import packages needed
from herschel.access import *
from herschel.access.util import *
from herschel.ia.numeric import *
from herschel.binstruct import *
from herschel.pus import *

#look to access packets associated with obsid = 180
pk = PacketAccess(180)

#connect to the default database to find the packets
hk_set = HcssConnection.get(pk)

#create and 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 to 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
#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
plot_fpu_hk = seq_FPU_HK.getMeasures("FPU_b_body_top")
```

A more elaborate means of obtaining and plotting housekeeping data for a given observation is presented in Example 11.6. This uses swing components to interact with the user who can obtain plots for multiple parameters simply by selecting a housekeeping parameter to plot via the pulldown menus. (see Figure 11-4).

```
#Example 11.6 – accessing HK information for a given observation
from herschel.access import *
from herschel.access.util import *
from herschel.ia.numeric import *
from java.awt import *
from herschel.ia.plot import PlotXY
from herschel.binstruct import *
from herschel.pus import *
from herschel.share.swing import *
from java.awt import Color
import java.util
import java.awt.event
import java.net.URL
import javax.swing as swing
import java

# Which obsid packets do you want?
input_obsid = swing.JOptionPane.showInputDialog\
   ("Enter obsid with HK info: ")

getid = int(input_obsid)


pk = PacketAccess(getid)
#pk.setApid(1026) ... use this to pick out certain HK data only
#e.g. APID=1026 is HIFI HK and HIFI single FCU, 2017 = FPU HK and Temps
#2022 = WPT HK data.
hk_set = HcssConnection.get(pk)
arrList = java.util.ArrayList()
#arrList = ArrayData()
for x in range(len(hk_set)):
 arrList.add(hk_set[x])
# arrList.add(PusTmSourcePacket(hk_set[x].getContents()))

print arrList
pseq = PacketSequence(arrList)
print pseq
seq_FPU_HK = pseq.select(TypeEquals("FPU_Housekeeping_rev1"))
seq_FPU_Temp = pseq.select(TypeEquals("FPU_Temperatures"))
seq_HIFI_HK = pseq.select(TypeEquals("HIFI_HK"))
seq_WPT_Housekeeping = pseq.select(TypeEquals("WPT_Housekeeping"))
seq_HIFI_single_FCU = pseq.select(TypeEquals("HIFI_single_FCU"))
par_FPU_HK = seq_FPU_HK[0].getParametersContained()
par_FPU_Temp = seq_FPU_Temp[0].getParametersContained()
par_HIFI_HK = seq_HIFI_HK[0].getParametersContained()
par_HIFI_single_FCU = seq_HIFI_single_FCU[0].getParametersContained()
par_WPT_Housekeeping = seq_WPT_Housekeeping[0].getParametersContained()

#define what to do for plotting each of the 5 sets of HK data
def graphChoice1(event):
 fpu_temp_in = str(addPull.getSelectedValue())
 display.text = fpu_temp_in
# fpu_temp_in2 = swing.JOptionPane.showInputDialog\
   ("Enter calibration factor: ")
# fpu_temp_cal = Integer(fpu_temp_in2).intValue()
 plot_fpu_temp = seq_FPU_Temp.getMeasures([fpu_temp_in])
 a = TableDataset()
```

```
 a["Time"] = Column(plot_fpu_temp[0].data)
 a["FPU_Temp_par"] = Column(plot_fpu_temp[1].data)
 p = PlotXY(a, fpu_temp_in,java.awt.Color.red)
 p.setLine(fpu_temp_in)
 p.setLabel("x-Axis", "Time (secs)")
 p.setLabel("y-Axis", fpu_temp_in)
 p.setTitle("Plot of "+fpu_temp_in+" with time for obsid "+input_obsid)

def graphChoice2(event):
 fpu_hk_in = str(addPull2.getSelectedValue())
 display2.text = fpu_hk_in
 plot_fpu_hk = seq_FPU_HK.getMeasures([fpu_hk_in])
 p = PlotXY(plot_fpu_hk, fpu_hk_in,java.awt.Color.red)
 p.setLine(fpu_hk_in)
 p.setLabel("x-Axis", "Time (secs)")
 p.setLabel("y-Axis", fpu_hk_in)
 p.setTitle("Plot of "+fpu_hk_in+" with time for obsid "+input_obsid)

def graphChoice3(event):
 hifi_hk_in = str(addPull3.getSelectedValue())
 display3.text = hifi_hk_in
 plot_hifi_hk = seq_HIFI_HK.getMeasures([hifi_hk_in])
 p = PlotXY(plot_hifi_hk, hifi_hk_in,java.awt.Color.red)
 p.setLine(hifi_hk_in)
 p.setLabel("x-Axis", "Time (secs)")
 p.setLabel("y-Axis", hifi_hk_in)
 p.setTitle("Plot of "+hifi_hk_in+" with time for obsid "+input_obsid)

def graphChoice4(event):
 hifi_single_in = str(addPull4.getSelectedValue())
 display4.text = hifi_single_in
 plot_hifi_single = seq_HIFI_single_FCU.getMeasures([hifi_single_in])
 p = PlotXY(plot_hifi_single, hifi_single_in,java.awt.Color.red)
 p.setLine(hifi_single_in)
 p.setLabel("x-Axis", "Time (secs)")
 p.setLabel("y-Axis", hifi_single_in)
 p.setTitle("Plot of "+hifi_single_in+" with time \
   for obsid "+input_obsid)

def graphChoice5(event):
 hifi_wpt_in = str(addPull5.getSelectedValue())
 display5.text = hifi_wpt_in
 plot_wpt_hk = seq_WPT_Housekeeping.getMeasures([hifi_wpt_in])
 p = PlotXY(plot_wpt_hk, hifi_wpt_in,java.awt.Color.red)
 p.setLine(hifi_wpt_in)
 p.setLabel("x-Axis", "Time (secs)")
 p.setLabel("y-Axis", hifi_wpt_in)
 p.setTitle("Plot of "+hifi_wpt_in+" with time for obsid "+input_obsid)

#Set up the frame and the flow for the layout of win (a GUI window)
win = swing.JFrame(title="FPU Temperature data", size=(500,300))
win.contentPane.layout=java.awt.FlowLayout()
#Create the pull-down list from the array of parameters
addPull = swing.JList(par_FPU_Temp)
#add scrolling to this
scrollpane = swing.JScrollPane(addPull)
#create the panel to house the list with scrolling
```

```
pulldown = swing.JPanel(size=(100,30),border=swing.border.TitledBorder\
("FPU Temp HK Choice"),layout=java.awt.BorderLayout(0,10))
#add in the srolled list into this panel area
pulldown.add(scrollpane)
#place in the window
win.contentPane.add(pulldown, java.awt.BorderLayout.NORTH)
#add a text display next to it that will display the HK choice made
display = swing.JTextField(preferredSize=(200,30), \
   horizontalAlignment=swing.SwingConstants.CENTER)
win.contentPane.add(display)
#add a button that will allow updates
choose = swing.JButton("Update", size=(65,70), \
   actionPerformed=graphChoice1)
win.contentPane.add(choose)
#package it nicely for display
win.pack()
#display it
win.show()

#Second set of HK data -- FPU HK data
#Ditto the above procedure for showing HK data
win2 = swing.JFrame(title="FPU HK data", size=(500,300))
win2.contentPane.layout=java.awt.FlowLayout()
addPull2 = swing.JList(par_FPU_HK)
scrollpane2 = swing.JScrollPane(addPull2)
pulldown2 =swing.JPanel(size=(100,30),border=swing.border.TitledBorder\
("FPU HK Choice"),layout=java.awt.BorderLayout(0,10))
pulldown2.add(scrollpane2)
#win2.contentPane.add(buildPulldown(), java.awt.BorderLayout.NORTH)
win2.contentPane.add(pulldown2, java.awt.BorderLayout.NORTH)
display2 = swing.JTextField(preferredSize=(200,30), \
   horizontalAlignment=swing.SwingConstants.CENTER)
win2.contentPane.add(display2)
choose2 = swing.JButton("Update", size=(65,70), \
   actionPerformed=graphChoice2)
win2.contentPane.add(choose2)
win2.pack()
#Move from default location on the screen so that
#it does not fall on the other HK windows.
win2.setLocation(0,201)
win2.show()

#Third set of HK data -- HIFI HK data
win3 = swing.JFrame(title="HIFI HK data", size=(500,300))
win3.contentPane.layout=java.awt.FlowLayout()
addPull3 = swing.JList(par_HIFI_HK)
scrollpane3 = swing.JScrollPane(addPull3)
pulldown3 =swing.JPanel(size=(100,50),border=swing.border.TitledBorder\
("HIFI HK Choice"),layout=java.awt.BorderLayout(0,10))
pulldown3.add(scrollpane3)
#win2.contentPane.add(buildPulldown(), java.awt.BorderLayout.NORTH)
win3.contentPane.add(pulldown3, java.awt.BorderLayout.NORTH)
display3 = swing.JTextField(preferredSize=(200,30), \
horizontalAlignment=swing.SwingConstants.CENTER)
win3.contentPane.add(display3)
choose3 = swing.JButton("Update", size=(65,70), \
actionPerformed=graphChoice3)
```

```
win3.contentPane.add(choose3)
win3.pack()
win3.setLocation(0,402)
win3.show()

#Fourth HK data set -- HIFI single FCU HK
win4 = swing.JFrame(title="HIFI single FCU data", size=(500,300))
win4.contentPane.layout=java.awt.FlowLayout()
addPull4 = swing.JList(par_HIFI_single_FCU)
scrollpane4 = swing.JScrollPane(addPull4)
pulldown4 =swing.JPanel(size=(100,30),border=swing.border.TitledBorder\
("HIFI FCU Choice"),layout=java.awt.BorderLayout(0,10))
pulldown4.add(scrollpane4)
#pulldown4.add(addPull4, java.awt.BorderLayout.SOUTH)
#win2.contentPane.add(buildPulldown(), java.awt.BorderLayout.NORTH)
win4.contentPane.add(pulldown4, java.awt.BorderLayout.NORTH)
display4 = swing.JTextField(preferredSize=(200,30), \
horizontalAlignment=swing.SwingConstants.CENTER)
win4.contentPane.add(display4)
choose4 = swing.JButton("Update", size=(65,70), \
actionPerformed=graphChoice4)
win4.contentPane.add(choose4)
win4.pack()
win4.setLocation(501,0)
win4.show()

#Fifth set of HK data -- WBS housekeeping
win5 = swing.JFrame(title="WPT HK data", size=(500,300))
win5.contentPane.layout=java.awt.FlowLayout()
addPull5 = swing.JList(par_WPT_Housekeeping)
scrollpane5 = swing.JScrollPane(addPull5)
pulldown5 =swing.JPanel(size=(100,30),border=swing.border.TitledBorder\
("WPT HK Choice"),layout=java.awt.BorderLayout(0,10))
pulldown5.add(scrollpane5)
#win2.contentPane.add(buildPulldown(), java.awt.BorderLayout.NORTH)
win5.contentPane.add(pulldown5, java.awt.BorderLayout.NORTH)
#win5.contentPane.add(pulldown5, java.awt.BorderLayout.NORTH)
display5 = swing.JTextField(preferredSize=(200,30), \
horizontalAlignment=swing.SwingConstants.CENTER)
win5.contentPane.add(display5)
choose5 = swing.JButton("Update", size=(65,70), \
actionPerformed=graphChoice5)
win5.contentPane.add(choose5)
win5.pack()
win5.setLocation(501,201)
win5.show()
```

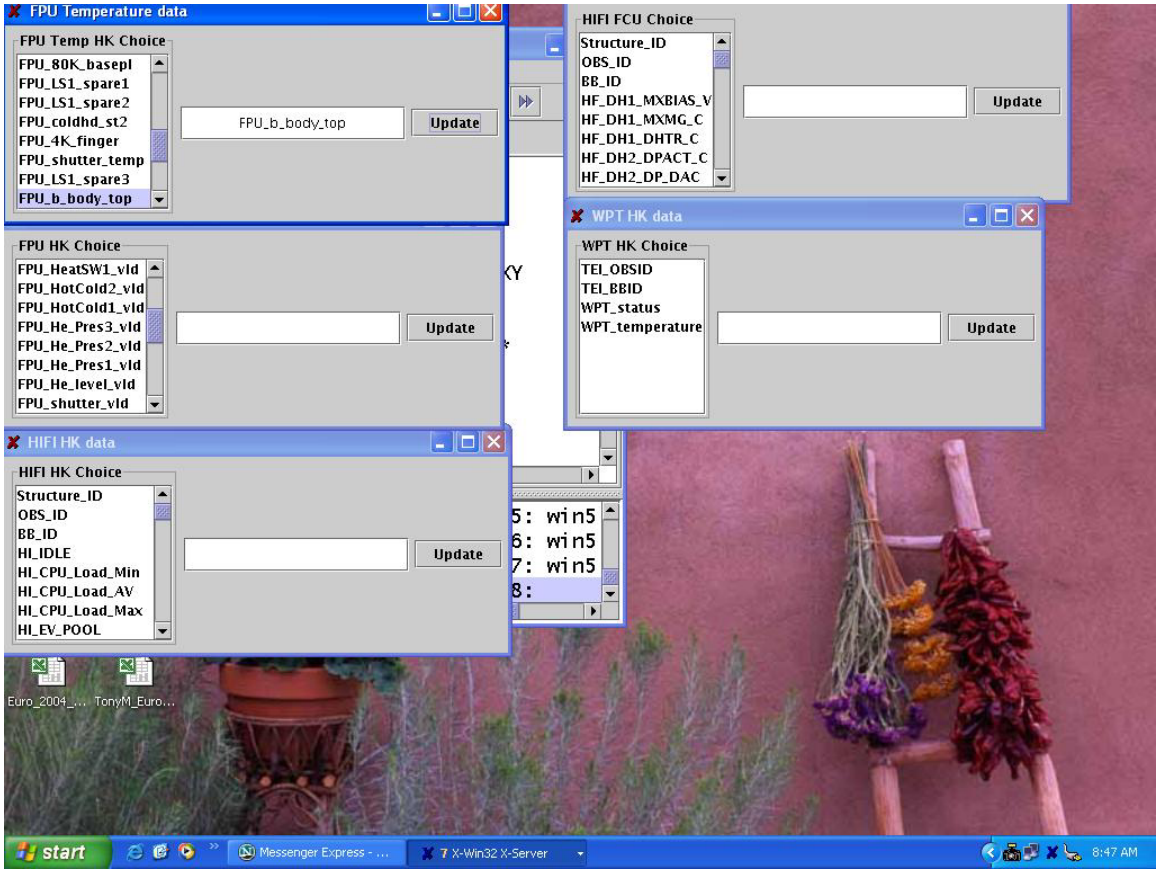

Figure 11-4: Pulldown menus available when example 11.6 is run. The FPU_b_body_top parameter has been highlighted via a mouseclick. Hitting the "Update" button will produce a timeline plot of the this housekeeping parameter value for the given observation id.

Housekeeping parameter values can also be obtained for a given time period (rather than observation id). Time is required to be put in for the start and finish time for the data of interest. This should of the format *Finetime* (see the *herschel.ccm.api* package). A script that plots housekeeping values (it has plotted values for up to 10 days worth of housekeeping packets) is provided in Example 11.7. This program only deals with packets holding the APID value of 2017, which contains HIFI housekeeping for the FPU and FPU Temps. Running this script provides an output similar to Figure 11-5.

```
#Example 11.7 - HK data plot in IA for a long time period.
from herschel.access import *
from herschel.access.util import *
from herschel.ccm import *
from herschel.ccm.util import *
from herschel.ccm.api import *
from herschel.ia.numeric import *
from java.awt import *
from herschel.ia.plot import PlotXY
from herschel.binstruct import *
from herschel.pus import *
from herschel.share.swing import *
from java.awt import Color
```

```
import java.util
import java.awt.event
import java.net.URL
import javax.swing as swing
import java
import java.text

#Get start time. First we create an instance of the calendar date/time.
#This actually gives us the current date and time. We will then change
#each appropriate part of the calendar to fit the date we start the
#search at.
start_date = java.util.Calendar.getInstance()
start_year = swing.JOptionPane.showInputDialog\
("Enter start time (Year) : ")
#Change the input to an integer
s_year = int(start_year)
#Change date to the right year
start_date.set(java.util.Calendar.YEAR, s_year)
start_month = swing.JOptionPane.showInputDialog\
("Enter start time (Month) : ")
s_month = int(start_month)
#Now we change the month ....
start_date.set(java.util.Calendar.MONTH, s_month)
start_day = swing.JOptionPane.showInputDialog\
("Enter start time (Day) : ")
s_day = int(start_day)
start_date.set(java.util.Calendar.DAY_OF_MONTH, s_day)
start_hour = swing.JOptionPane.showInputDialog\
("Enter start time (Hour) : ")
s_hour = int(start_hour)
start_date.set(java.util.Calendar.HOUR_OF_DAY, s_hour)
start_minute = swing.JOptionPane.showInputDialog\
("Enter start time (Minute) : ")
s_minute = int(start_minute)
start_date.set(java.util.Calendar.MINUTE, s_minute)
#Just go to the nearest minute for now.
start_date.set(java.util.Calendar.SECOND, 0)
start_date.set(java.util.Calendar.MILLISECOND, 0)
#Change the time into milliseconds
start_time = ((start_date.getTimeInMillis()/1000) \
+ 376012832) * 1000000

stop_date = java.util.Calendar.getInstance()
stop_year = swing.JOptionPane.showInputDialog\
("Enter stop time (Year) : ")
so_year = int(stop_year)
stop_date.set(java.util.Calendar.YEAR, so_year)
stop_month = swing.JOptionPane.showInputDialog\
("Enter stop time (Month) : ")
so_month = int(stop_month)
stop_date.set(java.util.Calendar.MONTH, so_month)
stop_day = swing.JOptionPane.showInputDialog\
("Enter stop time (Day) : ")
so_day = int(stop_day)
stop_date.set(java.util.Calendar.DAY_OF_MONTH, so_day)
stop_hour = swing.JOptionPane.showInputDialog\
("Enter stop time (Hour) : ")
```

```
so_hour = int(stop_hour)
stop_date.set(java.util.Calendar.HOUR_OF_DAY, so_hour)
stop_minute = swing.JOptionPane.showInputDialog\
("Enter stop time (Minute) : ")
so_minute = int(stop_minute)
stop_date.set(java.util.Calendar.MINUTE, so_minute)
#Just go to the nearest minute for now.
stop_date.set(java.util.Calendar.SECOND, 0)
stop_date.set(java.util.Calendar.MILLISECOND, 0)
#Change the time into milliseconds. The conversion gets the date into
#FineTime in microseconds. Problem is that FineTime and date
#time in milliseconds are measured from a different date,
#hence the conversion given below.
stop_time = ((stop_date.getTimeInMillis()/1000) + 376012832) * 1000000

#Need to convert final numbers into a FineTime.
start_1 = FineTime(start_time)

#Date/time of start for plotted data heading info - see later
prod_date = TaiConverter.fineTimeToDate(start_1).toString()

stop_1 = FineTime(stop_time)

#now extract the packets
pk=0
hk_set = 0
pk = PacketAccess(2017,start_1,stop_1)
hk_set = HcssConnection.get(pk)
pseq = PacketSequence()
for x in range(10):
 pseq.add(PusTmSourcePacket(hk_set[x].getContents()))

#Now we append the parameters to get a complete list
#of available HK parameters to plot.
par_pseq = java.util.ArrayList()

par_FPU_HK = pseq.select(TypeEquals\
("FPU_Housekeeping_rev1"))[0].getParametersContained()
par_FPU_Temp = pseq.select(TypeEquals\
("FPU_Temperatures"))[0].getParametersContained()
list = java.util.Arrays.asList(par_FPU_HK)
list2 = java.util.Arrays.asList(par_FPU_Temp)
par_pseq.addAll(list)
par_pseq.addAll(list2)
parlist = par_pseq.toArray()

#This is what to do once a graph has been chosen
def graphChoice1(event):
 fpu_temp_in = str(addPull.getSelectedValue())
 display.text = fpu_temp_in
 tim = Double1d(range(len(hk_set))) * 0.0
 temppar = Double1d(range(len(hk_set))) * 0.0
 j = 0
 for x in range(len(hk_set)/40):
  pseq = PacketSequence()
  pseq.add(PusTmSourcePacket(hk_set[40*x].getContents()))
  plot_fpu_temp = pseq.getMeasures([display.text])
```

```
  if (plot_fpu_temp[0].data > 0):
   tim[j] = plot_fpu_temp[0].data[0]
   temppar[j] = plot_fpu_temp[1].data[0]
   j = j+1
 tim[0] = tim[1]
 gtim = tim.get(Range(0,j))
 gtemppar = temppar.get(Range(0,j))
 a = TableDataset()
 a["Time"] = Column((gtim - gtim[0])/60)
 a["FPU_Temp_par"] = Column(gtemppar)
 p = PlotXY(a, display.text,java.awt.Color.red)
 p.setLine(display.text)
 p.setLabel("x-Axis", "Time (mins)")
 p.setLabel("y-Axis", display.text)
 p.setTitle("Plot of "+display.text+" with time. \
 Start date "+prod_date)

#The following sets up the GUI interface so that the user can
#choose the HK data to plot

#Set up the frame and the flow for the layout of win
win = swing.JFrame(title="HK data", size=(500,300))
win.contentPane.layout=java.awt.FlowLayout()

#Create the pull-down list from the array of parameters
addPull = swing.JList(parlist)

#add scrolling to this
scrollpane = swing.JScrollPane(addPull)

#create the panel to house the list with scrolling
pulldown = swing.JPanel(size=(100,30),border=swing.border.TitledBorder\
("FPU Temp HK Choice"),layout=java.awt.BorderLayout(0,10))

#add in the srolled list into the panel pulldown area
pulldown.add(scrollpane)

#place in the window
win.contentPane.add(pulldown, java.awt.BorderLayout.NORTH)

#add a text display next to it that will display the HK choice made
display = swing.JTextField(preferredSize=(200,30), \
horizontalAlignment=swing.SwingConstants.CENTER)
win.contentPane.add(display)
#add a button that will allow updates
choose = swing.JButton("Update", size=(65,70),\
 actionPerformed=graphChoice1)
win.contentPane.add(choose)

#package it nicely for display
win.pack()

#place it at position 100, 540 on the screen (below graph)
win.setLocation(100,540)

#actually show it
win.show()
```

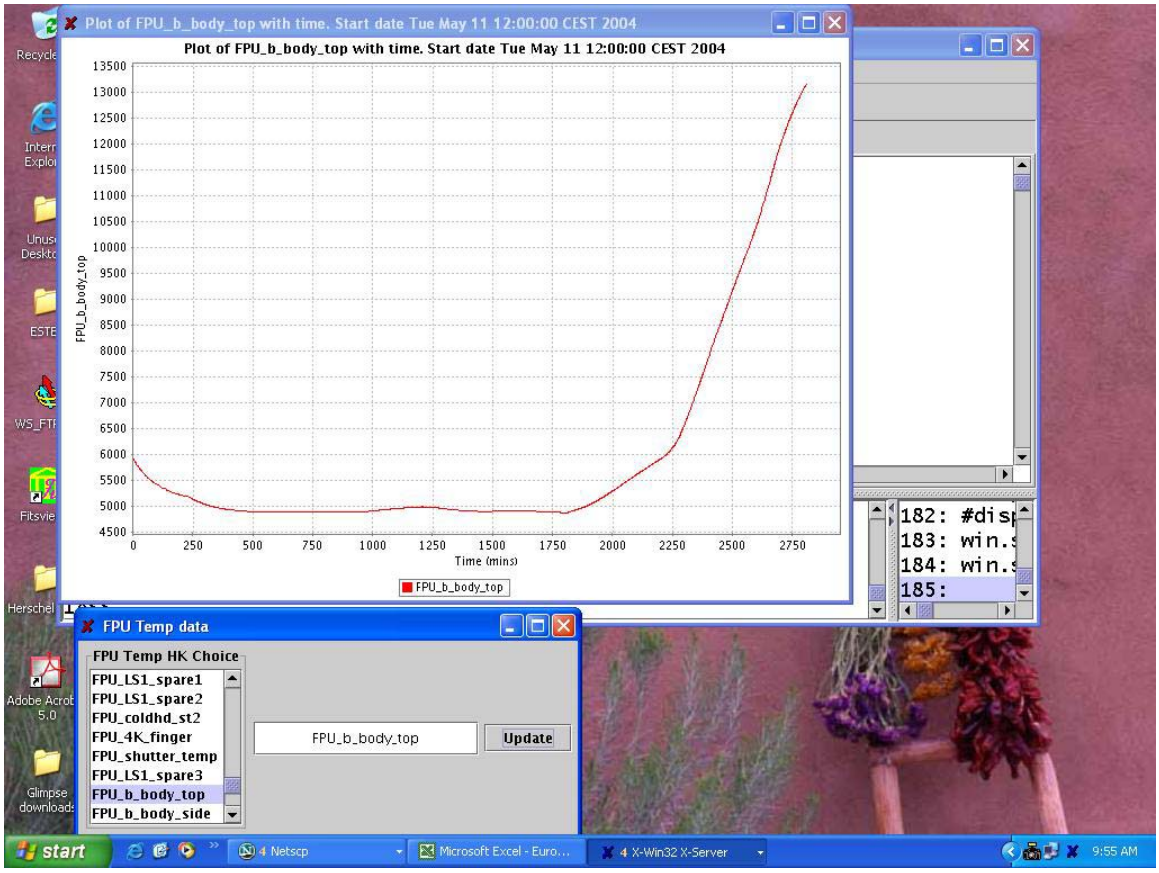

Figure 11-5: Example timeline plot obtained using example 11.7. The plot provides the parameter value change over a period of two days.

11.10 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
```

12 Appendix A: Example User's Property File

An example properties file to be placed in ~<user>/.hcss/myconfig file of the user.

```
# HCSS Properties File - location SRON
#
# Author: Anthony Marston
#
# General
#where the database is held
var.database.server = lin-sron-02.sron.rug.nl
#the database to use
var.database.devel=tony_hcss@${var.database.server}
dbfactory = herschel.versant.store.StoreFactoryImpl
dbname = tony_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.errorsonly           = 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

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
# CCM
hcss.ccm.mission.database = ${var.database.devel}
# Jconsole - increase buffer memory size
hcss.jconsole.buffer.size = 1024000
# Variables
var.hcss.instrument = HIFI
```

13 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.

herschel.ia.dataflow

Interfaces

Connectable
ControlListener
DataFlowListener
IaGuiProcess
IaProcess
InputListener
Launchable
ProcessConnector
ProcessControl
ProcessIndicator
ProcessInput
ProcessOutput

Classes

ConnectorBox
DataFlow
DataFlowDesktopManager
DataFlowEvent
DataFlowManager
IaProcessImpl
JConnectorBox
ProcessControlBaseImpl
ProcessControlEvent
ProcessDistributor
ProcessIndicatorBaseImpl
ProcessInputBaseImpl
ProcessInputEvent
ProcessOutputBaseImpl
TaskWrapper
ThreadDynamicProcessImpl

herschel.ia.dataflow.data.process

Classes

InProc
LaunchableInProc
NotListeningProc
NumberControl
OutProc
SlowListeningProc
StringViewProc
ThreadDynInProc
ThreadOutProc

herschel.ia.dataflow.example.indicator_control.monothread

Classes

ControllingGuiProcess
DataFlowApplication
IndicatingGuiProcess
SquareProcess

herschel.ia.dataflow.example.indicator_control.multithread

Classes

ControllingGuiProcess
DataFlowApplication
IndicatingGuiProcess
SquareProcess

herschel.ia.dataflow.template

Classes

ThreadDynamicProcessImplTemplate

herschel.ia.dataset

Interfaces

Algorithm
Annotatable
Attributable
Composite
Dataset
DatasetVisitor
DataWrapper
History
NumericParameter
Parameter
ParameterVisitor
Quantifiable

Classes

AbstractDatasetAndDataVisitor
AbstractDatasetVisitor
ArrayDataset
BooleanParameter
Column
CompositeDataset
DatasetUtil
DateParameter
DoubleParameter
LongParameter
MetaData
Product
StringParameter
TableDataset

herschel.ia.dataset.demo

Classes

Demo
MySpectrumFormula
SpectrumDataset
TableAdd

herschel.ia.demo.endtoend

Classes

Double1dPlotter
MakeHifiDFProduct
MakeHifiProduct
ProcessProductMaker
ProcessRunningAverage

herschel.ia.image

Classes

Display
Histogram
ImageDataset
Layer
Rotate
Scale
Translate
Wcs

herschel.ia.image.gui

Classes

AnnotationToolbox
ImageCutLevels
ImageDisplayStatusPanel
ImageSaveDialog

herschel.ia.io.ascii

Interfaces
AsciiCsv
AsciiFixedWidth
AsciiFormatter
AsciiParser

Classes
AbstractAsciiFormatter
AbstractAsciiParser
AsciiTableTool
CsvFormatter
CsvParser
FixedWidthFormatter
FixedWidthParser
TableTemplate

herschel.ia.jconsole.api

Interfaces
Console
ConsoleEditor
Editor
JConsoleFactory

Classes
JConsoleDefaults
JConsoleFactoryManager

herschel.ia.jconsole.jython

Interfaces
History
Log

Classes
BufferedLog
Command
Interpreter
JConsoleStream
JConsoleWriter
ObservableHistory
Output

herschel.ia.io.fits

Interfaces
FitsDictionary

Classes
FitsArchive
FitsRules

herschel.ia.io.fits.dictionary

Classes
AbstractFitsDictionary
DictionaryHcss
DictionaryHeasarc
DictionaryStandard
ResourceFitsDictionary
WrappedFitsDictionary

herschel.ia.jconsole.gui

Classes
DebugBar
JConsole
JConsoleAction
JConsoleEditor
JConsolePane
JIDEComponent
JIDEFrame
ScriptDebugger
ScriptPane

herschel.ia.jconsole.util

Classes
CopyStream
ExtensionFileFilter
JFilesSaveDialog
JRefreshFileChooser
JythonDocument
JythonUndoManager
TextEditor
UndoableTextEditor
XSplitPane

herschel.ia.jconsole.tools

Classes

herschel.ia.numeric.function

Classes

JIDE

herschel.ia.numeric

Interfaces
(note 2d and 3d versions also available)
Array1dData
ArrayData
ArrayDataVisitor
ComplexData
Logical1dData
LogicalData
Numeric1dData
NumericData
Ordered1dData
OrderedData

Classes
(note 2d and 3d versions also available)
AbstractArray1dData
AbstractArrayData
AbstractArrayDataVisitor
AbstractComplex1dData
AbstractLogical1dData
AbstractNumeric1dData
AbstractOrdered1dData
Bool1d
Byte1d
Complex
Complex1d
Double1d
Float1d
Int1d
Long1d
Range
Selection
Short1d
String1d
BoxCarFilter
Complex1dFunctions
ComplexFunctions
Convolution
CubicSplineInterpolator
Double1dFunctions
Double2dFunctions
DoubleArrayFunctions
DoubleFunctions
FFT
GaussianFilter
Interpolator
LinearInterpolator
LinearLeastSquaresFitter
MatrixFunctions
NearestNeighborInterpolator
Polynomial
PolynomialFitter
Shift
WindowFunctions

herschel.ia.numeric.function.util

Classes
MoreMath

herschel.ia.numeric.function.fit

Classes
AbstractModel
AmoebaFitter
BinomialModel
DataFormatter
Fitter
FunnyModel
FunnyModelInput
Gauss2DModel
GaussMixModel
GaussModel
GaussNoPartial
IndexSet
IterativeFitter
LevenbergMarquardtFitter
LinearModel
MatrixSelections
ModelInput
NonLinearModel
NullModel
PolynomialModel
PowerModel
SineAmpModel
SineMixedModel
SineModel

Exceptions
NonConvergenceException

herschel.ia.plot

Interfaces
CoordMouseListener

Classes
Axis
ComponentList
CoordMouseEvent
FontChooser
FontDisplay
Layer
LocationDisplay
PlotProperties
PlotPropertyManager
PlotXY
PlotXYCompositeRenderer
View

herschel.ia.task

Classes
_Dummy
Share
Signature
Task
TaskParameter

Exceptions
SignatureException

herschel.ia.task.api

Interfaces
ShareApi
SignatureCheck
SignatureChecker
SignatureVisitor
TaskApi
TaskHelp
TaskInfo
TaskParameterChecker
TaskParameterVisitor
TaskVisitor

Classes
SignatureEntry
TaskFactory

herschel.ia.task.impl

herschel.ia.ui

Classes SignatureVisitorCheckerImpl SignatureVisitorFormalCheckerImpl TaskVisitorHelpImpl TaskVisitorInfoImpl herschel.ia.task.util Classes JythonFacade	Interfaces *FrameMaker* *SystemPopupReturnable* *WindowActivationListener* *WindowCleaner* *WindowClosedListener* *WindowOpenedListener* Classes ComponentPrinter GuiUtils Help PrintControl ScreenshotGenerator SystemPopup SystemPopupMouseHandler TextPrinter WindowManager