ARGONNE NATIONAL LABORATORY

Intense Pulsed Neutron Source

ISAW User Manual

INTENSE PULSED NEUTRON SOURCE

ISAW User Manual

Software Developers
Dennis Mikkelson, DataSet Tools, Netcomm packages
Ruth Mikkelson, Command package NeXus data retriever
Peter Peterson, Linux install, SCD operators
Alok Chatterjee, ISAW GUI package
John Hammonds, IPNS package
Dongfeng Chen, ChopTools package
Thomas Worlton, project manager

The research is sponsored by the U.S. Department of Energy under contract no. W-31-109-ENG-38.

9700 South Cass Ave • Building 360
Phone 630-252-8755 • Fax 630-252-4163

CHAPTER 1

Introduction

Data analysis is a vital step of every neutron scattering experiment. Often this analysis depends upon the use of several different operations that span a wide range of instruments and software packages. While this diversity is valuable in modern laboratories, many users prefer consistent results with a minimal array of tools. Integrated Spectral Analysis Workbench software aims to provide a comprehensive set of tools for reading, manipulating, and visualizing neutron scattering data.

About ISAW. ISAW is cross-platform, modular

This user manual is intended to help new and veteran users better use ISAW to its fullest extent. It is simple enough for beginners, but also includes detailed information for experts who would like to modify ISAW to suit their needs.

ISAW and Java™

Why Java?

One of the major advantages of writing software in the Java language is portability. ISAW is designed so that it can be used on any computer that has the Java Runtime Environment installed on it. This facilitates the support of a variety of operating systems and computing platforms including Windows, Mac OS, and Linux.

Java Basics

Files containing Java language source code have the extension "**.java**". The compiled Java byte code files have the extension "**.class**". These class files can be created on one computing platform, such as Intel or PowerPC, and run on any other computing platform with the same version (or higher) of Java. Developing java programs and compiling them requires the Java Development Kit, which can be downloaded for free from Sun's web site: http://java.sun.com/j2se/. If you are only interested in running java programs, then you simply need the Java Runtime Environment which is also available for free at the Java web site.

Java archive files

In order to more easily distribute ISAW, class files and libraries are placed into Java archive files, or "**.jar**" files. These files are used to store the various components that constitute a program such as ISAW, and are very similar to the common "zip" file. The main difference between a jar file and a zip file is that the jar file contains a manifest that specifies how the jar will be used. For example, if one of the jar files is intended to be run as an independent program, one of their classes will be specified as the "main" class.

All of the ISAW class files are contained in **isaw.jar** and they should be extracted in order for ISAW to be fully functional; however, they are not required to use ISAW's more basic features. To extract, or "unjar," the files, the `jar` command is most commonly used, but it requires the full Java Development Kit. If the JDK is installed, you can unjar ISAW by typing the command `jar -xvf ISAW.jar` into a terminal.

Java libraries in ISAW

In Java programs, a library is a collection of subprograms that performs special tasks or services independent of the parent program. This helps contribute to the modular design of ISAW, and makes a variety of useful tools available to users. Libraries that are currently utilized by ISAW include NeXus, HDF, Scientific Graphics Toolkit, JOGL, and GSAS. Each of these libraries is available on the ISAW ftp server (ftp://zuul.pns.anl.gov/isaw). You can also find a small number of libraries in the **lib** folder in the main ISAW directory on your computer.

In order to provide support for NeXus files, it is necessary to include the HDF and NeXus libraries. The NeXus API (see http://www.neutron.anl.gov/nexus/) is layered on top of the HDF library that was developed at the National Center for Supercomputer Applications (see http://www.hdfgroup.org). These libraries are written in C or C++ and require separate versions for different computing platforms.

The Scientific Graphics Toolkit, sgt_v2, is a general purpose Java graphics package developed at the National Oceanographic and Atmospheric Administration (NOAA). It is used in the Contour View. The SGT package can be found in **gov** folder within the main ISAW directory. More information about SGT can be obtained from:

http://www.epic.noaa.gov/java/sgt/index.html.

JOGL, or Java OpenGL, is a library that allows the 3D rendering capabilities of OpenGL to be utilized in Java programs. Since JOGL is so closely tied to the OpenGL API, your machine must support OpenGL in order for JOGL to work. Currently, the only viewer in ISAW to take advantage of JOGL is the SCD Reciprocal Lattice Viewer.

The GSAS library allows ISAW to utilize a small number of GSAS and Fortran subroutines including a Gaussian peak shape function, a pseudo-Voight peak shape function, and a complementary error function. More functionality will be added to this library in the future.

Miscellaneous files included with ISAW

Other files included besides the ISAW class files and these libraries are sample scripts, sample data, Help, and Documentation. To further simplify distribution, these files and libraries are zipped or archived together. A single install jar file can be used on Windows, Linux, or UNIX systems.

Installing ISAW

Steps required for ISAW installation

The first time ISAW is installed it is necessary to ensure that the following steps are completed.

- **Install java** on the computer to be used for ISAW
- **Copy the ISAW distribution file(s)** to the local computer
- **Unzip the ISAW files** to a folder on the local disk
- **Run ISAW**

Install Java

Before ISAW can be installed, it is necessary to install the Java Runtime Environment (JRE) or the Java Distribution Kit (JDK) on your computer. If you simply want to run ISAW then you will only need the JRE; however, if you want to make changes to ISAW or develop your own tools, then you should download the JDK. ISAW currently requires Java 6. For software and instructions, see:

http://developers.sun.com/resources/downloads.html

After installation, the Java executable (java.exe) will be in the **bin** folder of the main JRE or JDK folder.

The command to start ISAW requires that either the path to this command is included on the command line or that the path environmental variable includes the Java folder. You can test whether your path environmental variable includes java.exe by opening a command pane and typing the command `java -version` into a terminal. If Java is correctly installed, this will tell you which version of Java is installed. If you get an error message, it will be necessary to either modify your path environmental variable to include the Java "bin" folder in the path or to use an explicit path when typing the `java` command.

On Windows systems, the path can be modified via Environmental Variables. To access this click **Start**, then click **Settings**, and then click **Control Panel**. Double-click the **System** icon and then select the **Advanced** tab in the window that appears. Under the **Advanced** tab, press the **Environmental Variables** button. You should now see options for **Path** and **User variables**. Add the full system path to the **bin** folder of your JRE or JDK software to the **Path** section.

Copy the ISAW distribution file

The ISAW distribution file is available through the ftp site:

ftp://zuul.pns.anl.gov/isaw

As was discussed earlier, the ISAW class files in **isaw.jar** are the same for every computing platform, but installation procedures and run procedures, as well as the non-ISAW files vary for different platforms. To simplify distribution and ease the burden on users, all of the different files have been packed inside a single install executable.

Unzipping the ISAW distribution

The ISAW distribution is a self-extracting Java archive (jar file). To extract the files and install them on your computer, double-click the downloaded file and then choose the destination directory for the files. We recommend installing them into a folder in your home directory, but you can place them anywhere.

Running ISAW

 On Windows systems, open the file called **ISAW_exec.bat** in the main ISAW directory. This file adds the folders containing the native interface libraries, HDF and NeXus libraries to the path and runs ISAW.

 On a Mac OS computer, simply open the file called **Isaw_exec.applescript** in the main ISAW directory. This will open the Apple Script Editor program and display the ISAW execution script in a new window. This script can be easily edited to perform any additional functions that you require. When you are ready to run ISAW, click the green **Run** button.

ISAW Props

Introduction

The ISAWProps.dat file is designed to store user preferences for ISAW including your default web browser, default instrument, remote data servers, default screen size, and many other options. It can usually be found in your operating system's home directory.

Editing ISAW Props

ISAW Props can be edited by selecting the **Edit Properties File** in the **Edit** menu in ISAW, or by opening IsawProps.dat with any simple text editor. To change the value for a particular field, simply type the new value in the area after the equals sign (=). There are several categories of properties that can be edited in ISAW and they will be listed below. The following documentation will list the key, or specific property, and an example value for that key.

Directory Options

These properties define the path for data, script, and/or operator locations in ISAW.

```
ISAW_HOME=C:/ISAW/
Help_Directory=C:/ISAW/IsawHelp/
Script_Path=C:/ISAW/Scripts/
Data_Directory=C:/ISAW/SampleRuns/
Instrument_Macro_Path=C:/ISAW/
User_Macro_Path=C:/ISAW/
Image_Path=C:/ISAW/images/
```

Preferred Browser

This property defines the default browser to be used by ISAW to open HTML files and external links. It should list the system path to the actual application that will be used.

```
PREFERRED_BROWSER=/Applications/Safari.app/Contents/MacOS/Safari
```

Live Data Server Options

ISAW can connect to live data servers that can provide data as it is collected. The following entries specify the locations and port numbers that are used to communicate with the server. The first line specifies the name of the instrument that you will be collecting data from, the second line specifies the server address followed by the port number. These val-

ues should be separated by a semicolon. The default port number is 6088, but this option can be changed if the server is transmitting data through a different port.

```
Inst1_Name=HRMECS
Inst1_Path=zeus.pns.anl.gov;6088

Inst2_Name=mandrake
Inst2_Path=mandrake.pns.anl.gov;6088

Inst3_Name=GPPD
Inst3_Path=gppd-pc.pns.anl.gov;6088

Inst4_Name=UW-Stout
Inst4_Path=dmikk.mscs.uwstout.edu;6088
```

Remote Data Server Options

ISAW can also connect to remote data servers to retrieve datasets. These properties are similar to those in the **Live Data Server Options**.

```
IsawFileServer1_Name=HRMECS(zeus)
IsawFileServer1_Path=zeus.pns.anl.gov;6089

IsawFileServer2_Name=Test(dmikk-Isaw)
IsawFileServer2_Path=dmikk.mscs.uwstout.edu;6089

NDSFileServer1_Name=Test(dmikk-NDS)
NDSFileServer1_Path=dmikk.mscs.uwstout.edu;6008
```

Screen Size

These options are used to change the default screen size of windows in ISAW. Values are expressed in pixels or percentage. Percentage values should be entered as values less than one; pixel values should be entered as values greater than one.

```
Isaw_Width=700
Isaw_Height=500
Tree_Width=300
Status_Height=200
```

Viewer Options

The IsawProps.dat file also contains default values for many viewer settings such as color scale, 3D view position, horizontal scrolling flag, rebinning flag, brightness, horizontal graph scale factor and others.

```
ColorScale        = Rainbow
RebinFlag         = false
HScrollFlag       = true
ViewAltitudeAngle = 5.0
ViewAzimuthAngle  = 90.0
ViewDistance      = 4.5
ViewGroups        = Medium
ViewDetectors     = SOLID
Brightness        = 40
Auto-Scale        = 0.0
```

The following are the possible values for the various viewer options.

Supported ColorScale names:

- Gray
- Negative Gray
- Green-Yellow
- Heat 1 (default)
- Heat 2
- Rainbow
- Optimal
- Multi
- Spectrum

Possible RebinFlag values:

- true   (default)
- false

Possible HScrollFlag values:

- true
- false   (default)

Valid ViewAzimuthAngle values:

- -180 to 180

Valid ViewAltitudeAngle values:

- -89 to -89

Valid ViewDistance values:

- positive float values, will be clamped to [r/2,5r]
- where r is the maximum sample to detector distance

Possible values for the ViewGroups flag

- Small
- Medium
- Large
- NOT DRAWN

Possible values for the ViewDetectors flag

- Filled
- Hollow
- Marker
- NOT DRAWN

Valid Brightness values:

- 0 to 100

Valid Auto-Scale values:

- 0 to 100

CHAPTER 2

Introduction to ISAW

The ISAW Graphical User Interface

The Graphical User Interface, or GUI, is an important part of many computer programs. ISAW uses a GUI to provide a more intuitive working environment for users. The main ISAW GUI consists of two main panels, a menu bar, and a message area. The relative sizes of the interior areas can be adjusted by dragging the gray divider bars or by clicking on one of the triangles in the bar. The entire ISAW window can be resized by dragging the corner of the window.

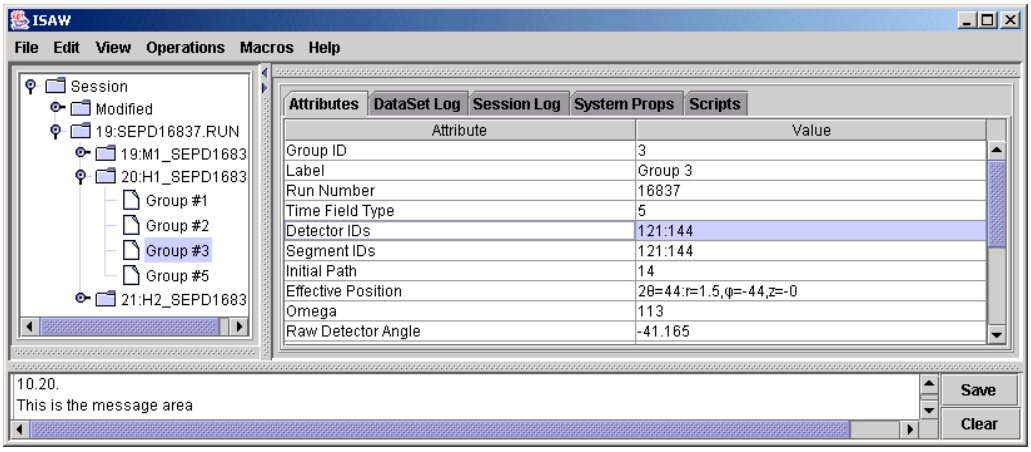

Figure 2-1, The ISAW Graphical User Interface (GUI).

The ISAW Menu Bar

The ISAW menu bar is used to load files, create views of the data, perform operations on the data, and to select options. Loading and saving of data is initiated through the **File menu**. The **Edit** menu allows you to remove nodes from the tree and/or modify DataSet attributes. The **View menu** allows you to create DataSet viewers. Each viewer also includes a menu bar that can be used to change the viewer type and the options for that view.

The **Operations menu** is context sensitive. The operations shown in the menu are the ones that are available for the object selected. If no DataSet is selected the operations menu may be empty. The **Macro menu** contains macros or scripts developed for specific instruments. The **Help menu** provides information about using ISAW.

The Tree View

The left panel of the GUI shows a tree view of loaded run files, sets of spectra (DataSet objects), and spectra (data objects). Each run file, shown as a folder icon with a file name, contains a monitor set and a histogram set, labeled with an "M" and an "H" respectively. Spectra, or group data, are shown as document icons in the tree. When you modify a DataSet, a new DataSet will be created and a new tree node will appear under the **Modified** node. DataSets and containers for DataSets are displayed as folder icons in the tree.

Tabbed Panes

In the main ISAW panel there are a series of five tabs that are used to display various types of information or interfaces. Simply click on the tab that you would like to view and a pane will open underneath it.

The Attributes Tab

The first tabbed pane shows the attributes of the selected DataSet or Data block. Attributes may be numbers or text data, and some attributes consist of more than one value. This pane is updated dynamically when different spectra or DataSets are selected.

DataSet Log Tab

When a DataSet is selected in the tree view, the DataSet Log pane shows the log of actions taken on the selected DataSet.

Session Log Tab

The Session Log tabbed pane shows a log of all of the commands issued during the current session.

System Properties Tab

The System Properties tabbed pane displays a list of information about your system while running ISAW. This information includes the ISAW build date and time, the amount of memory currently used by the Java virtual machine (JVM), the Java version, and the paths to some of the software components.

Live Data Connection Tab

When viewing live data, the Live Data Connection tab will appear to display data sets and controls available from the currently connected live data server. To terminate a live data connection, click the **Exit** button.

Scripts Tab

The Scripts tab allows users to create, modify, save, and load scripts for use in ISAW. Scripts can also be reached through the **Load Script** item on the **File** menu or through the **Macros** menu-bar item.

CHAPTER 3

ISAW Data Objects

The ISAW data model consists of two basic components: the Data block, and the DataSet.

The Data Block

The **Data block** deals with one list of values, and typically represents one spectrum. Data blocks are used to store either a histogram or a function. A **histogram** is used to record bin boundaries and total counts within a bin. A **function** is used to record samples at a specific point. Each Data block also contains errors and an extensible list of attributes associated with the spectrum. These **attributes** include metadata such as the effective detector position, detector IDs, etc, which is needed for data reduction.

The DataSet

A **DataSet** can be described as a list of Data blocks; however, it might also represent a list of all spectra from all pixels on an instrument, or a list of reduced spectra from multiple runs. Like the Data block, each DataSet contains an extensible list of attributes that apply to the entire DataSet.

DataSet Creation

Some basic operations such as addition and scalar multiplication are supported by all DataSets; however, DataSets come from several different classes of instruments and some more complex operations are not applicable to all types of instruments. Each DataSet maintains a list of relevant operators that can be applied to it. The list of appropriate operators is assigned to a DataSet when it is created by a DataSetFactory object.

DataSet Organization

An IPNS Run file generally contains data taken on a single sample at constant conditions. It is often useful to combine the data from measurements taken under different conditions. This can be done by merging DataSets from different run files. It can also be done by combining DataSets into a single experiment. Currently there are no operations that can be done on collections of DataSets, but it is useful to be able to store all the DataSets comprising an experiment in one file. To combine DataSets, select the DataSet or DataSets and right click on them. You can then elect to "send to" a common container. The "Send to" menu options are shown in the figure.

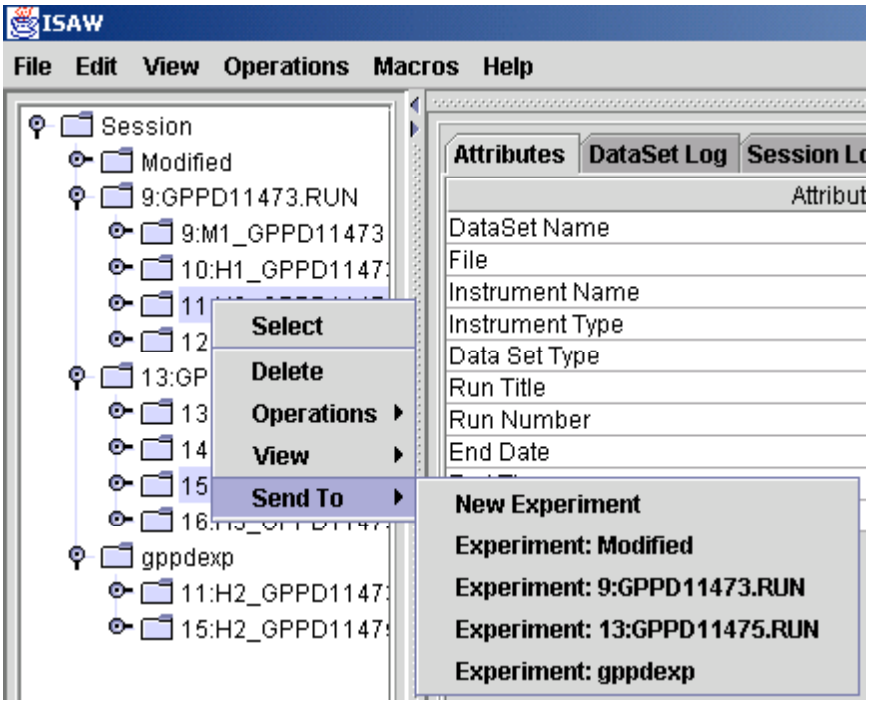

Figure 3-1, The pop-up menu that is produced by right-clicking on selected DataSets.

DataSet Operators

Each operator is an object that includes a list of parameters that the user can specify through a generic dialog box. When the user pulls down the **Operations menu** from the menu bar the menu is automatically populated with the names of the operator objects included in the current DataSet. When the user selects a particular operator, a generic dialog box pops up that allows the user to specify the parameters needed by the operation. The dialog box obtains the list of required parameters from the operator object and displays appropriate GUI components based on the types of parameters that are to be specified. This generalization of operators makes it very simple to create new operations and no changes to the user interface are needed when a programmer adds a new operation. The name of the new operation appears automatically on the operations menu and an appropriate parameter input dialog box is generated automatically.

CHAPTER 4

Reading and Saving Data

ISAW is designed in a modular fashion and its data input functions are performed by data retrievers. The retriever that is used for a data file is completely determined by the extension of the file. For example, files with the extension .run would be implemented by the IPNS runfile retriever. To see a list of the file extensions supported by ISAW, click on the File menu in the ISAW GUI, and then select Open. In the dialog box that appears click the drop-down menu that says "Files of type" as seen in figure 4-1.

The IPNS Runfile Retriever

The IPNS run file retriever is used to read IPNS binary data files. The Java code for this retriever is included in a separate package called **IPNS**. To load an IPNS runfile, open the **File** menu and click **Load Data file(s)**. This will display a file chooser box in the default directory that you specified in ISAWProps.dat. When an IPNS runfile is selected, ISAW will recognize the file and load the IPNS run file retriever. Loaded files will then be displayed as nodes in the ISAW tree view.

The NeXus File Retriever

To load a NeXus file, open the **File** menu and select **Load Data file(s)**. The same file chooser window will appear; however, when you click on a NeXus file, ISAW will recognize it and load the NeXus file retriever. If the NeXus file contains detector position information that is compatible with ISAW then you will be able to perform operations in addition to visualizing the data. It should be noted that the IPNS run file retriever is not required for reading NeXus files.

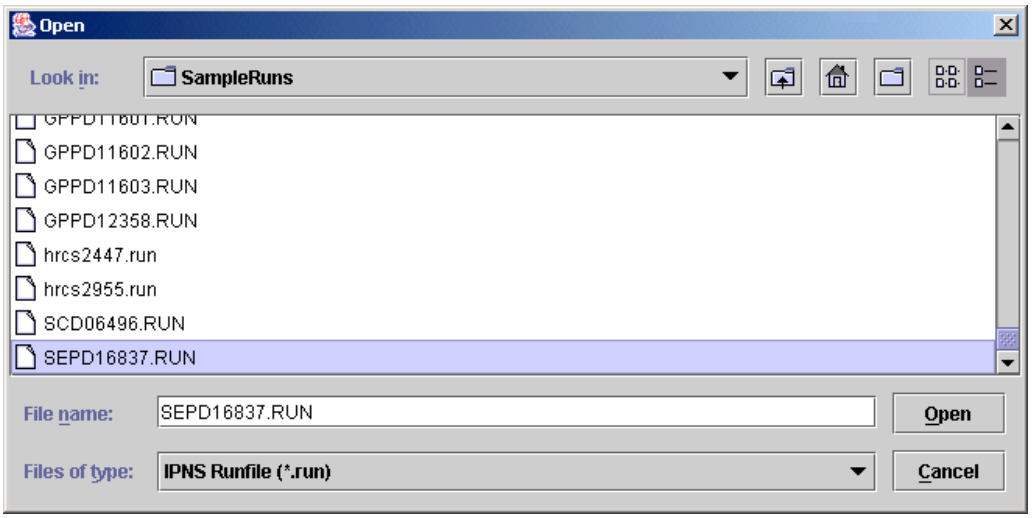

Figure 4-1, A file chooser box that appears when selecting "Load Runfile(s)" from the File menu in ISAW.

Log File Retriever

Log files are often recorded in conjunction with data collection. At IPNS, the APS Self Describing Data Set (SDDS) log files are used to record temperature, pressure, etc. (see http://www.aps.anl.gov/asd/oag/manuals/SDDStoolkit/SDDStoolkit.html). To open an SDDS log file, open the **File** menu and select **Load Data file(s)**. In the file chooser box that appears (see figure 4-1), change the **Files of type** drop down box to **SDDS files**. The

log file retriever will split up the different log datasets read according to the units of the variable saved.

Live Data Retriever

One of the primary reasons for using Java for ISAW was to allow easy integration with the Internet. Java includes built-in tools to allow interaction with network objects on different Internet nodes. Live data viewing in ISAW uses a data sender and a data server. A **data sender** runs on the data acquisition system and a **data server** runs on the control computer used to set up and control data collection for that instrument.

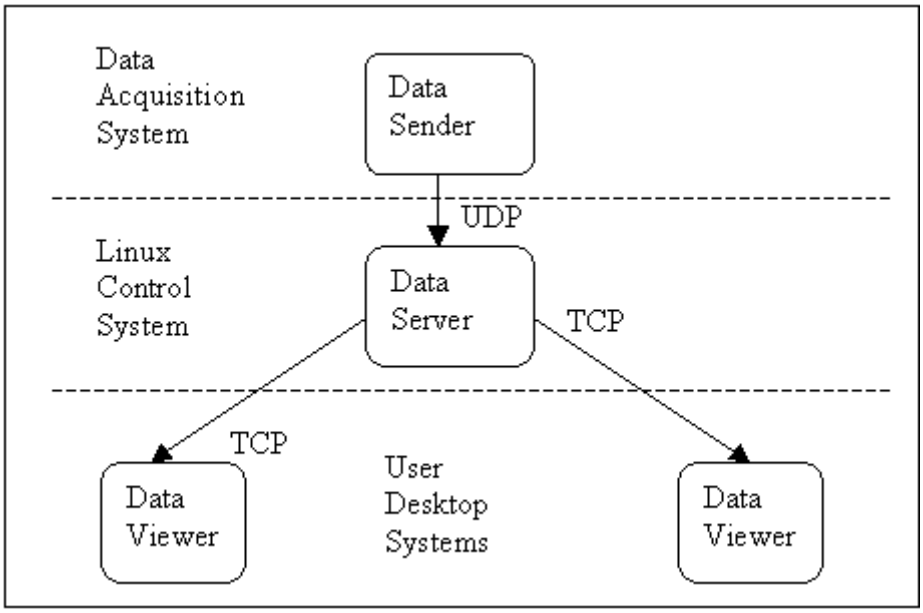

Figure 4-2, The communication paths and protocols for live data access.

The data sender uses the UDP protocol to send spectra to the data server. The data server accepts packets from one or more data senders and sends spectra to ISAW clients. The data server talks to data senders using UDP protocol and talks to ISAW clients using TCP protocol.

To connect to a live data server, open the **File** menu, select **Load Data**, and then select the **Live** submenu. This menu displays a list of the live data server names that were specified in IsawProps.dat. To query a server, simply click on it in the **Live** submenu.

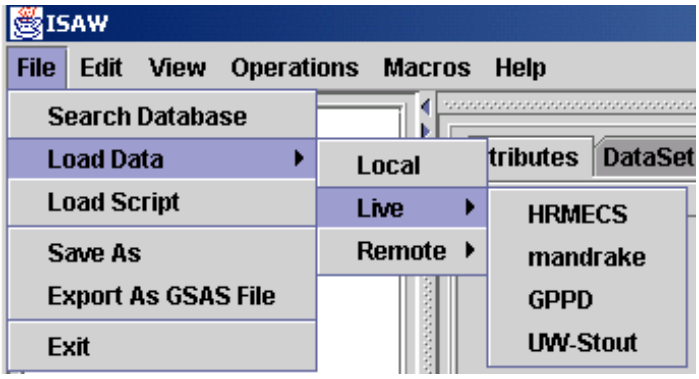

Figure 4-3, The menu paths and associated display for a live data server connection.

The only information that the **Live** submenu provides to the live data code is the node name. Selecting an Internet node name in the **Live** submenu will inquire what DataSets are available on that node. If a connection is successful, a tabbed pane listing the data source node name and the live DataSets will appear in the main ISAW panel. This window displays a number of options including **Show**, **Update**, **Auto**, and **Record**. Checking the **Show** checkbox performs one-time copy and display of the corresponding DataSet. The **Update** button will refresh the current view of the data, and checking the **Auto** checkbox will refresh the data at a preset interval as determined by the slider at the bottom of the live data pane. The **Record** button will send a copy of the current live data DataSet to the ISAW tree for further processing.

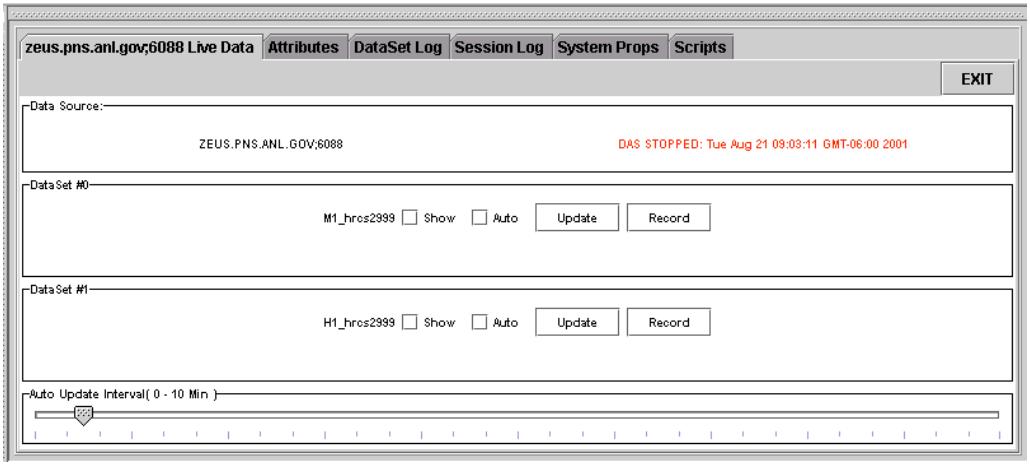

Figure 4-4, The live data display dialog box.

In figure 4-4, DataSet 0 is a monitor DataSet. Checking the **Show** checkbox on the second DataSet causes ISAW to read the remote data and pop up a view of the DataSet. Checking the "Auto" box would cause ISAW to read the remote data at the interval selected by the slider at the bottom.

Remote file servers

ISAW can retrieve datasets from remote file servers. The names, addresses, and port numbers of available remote file servers are specified in IsawProps.dat. When you attempt to connect to one of the remote file servers on the menu, a dialog box will appear asking for all the information needed to make the connection, including username and password.

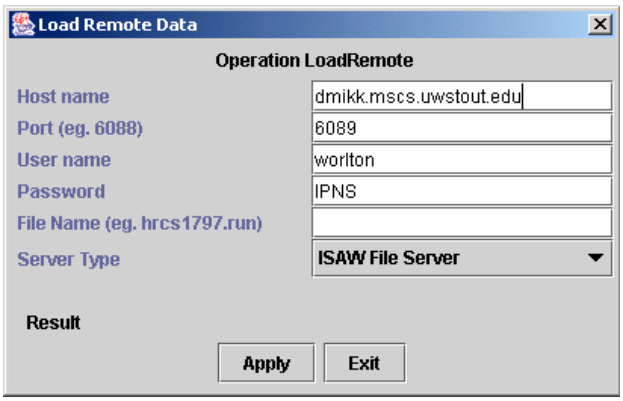

Figure 4-5, The Load Remote Data dialog box.

Saving Data

There are two ways of saving ISAW data. The first is by saving a DataSet object. This is possible because all the objects in ISAW are Java serializable objects. The serialized format is not guaranteed to remain the same in future versions of Java, so this method is not recommended for long-term storage of ISAW data. Currently, the only other way of saving data from ISAW is to export it as a GSAS file. The **GSAS** data format consists of a title card for the file plus a title card and data for each detector bank. The data is written ten integers per row of data. The bank number is used to obtain information from another file that contains calibration information for that bank. To ensure that users know which detector group a bank number refers to, comment cards are usually included at the beginning of the file. It is also possible to save ISAW DataSets as NeXus files.

To save ISAW DataSets as NeXus files, choose the desired DataSet and select "Save As" from the File menu. In the "Save As" dialog box, select "NeXus File" for "Files of type". Then choose a location and a file name and click on the Save button.

Data Viewers

ISAW provides several different methods for viewing data. In the ISAW data model, the data and views of the data are separate. This means that changing the data viewer will not change the data, but changing the data will change the data viewer. Viewers can be thought of as observers of the DataSet; whenever a DataSet changes, it notifies all observers of that DataSet so they can adjust their view. This means that changes to the data will be updated in all of the viewers for that data as they are made.

Viewer Menus

Each viewer contains a menu bar that is used for executing common functions as well as viewer-specific functions. The **File menu** is usually used to save/load a DataSet, print the viewer window, and close the viewer. Depending on the viewer, there may be different options available in this menu. The **Edit menu** is usually used to manipulate selections that have been made in the viewer. This includes summing the counts of selected/unselected spectra, deleting the counts of selected/unselected spectra, clearing select flags, and sorting data. These options may also change depending on the viewer. The **View menu** is constant in all viewers. This menu is used to either create a duplicate viewer window, or change the current view to any of the supported viewer types. The **Options menu** is the most varied menu between viewers. It allows many of the viewer-specific parameters to be manipulated. For some of the viewers, the default value of these options can be adjusted in ISAWprops.dat.

Image Viewer

The Image View

The Image Viewer works by drawing a line or block of lines for each detector group. These lines are then rasterized and combine to form an image. The colors of each line correspond to the intensity or number of counts at that time-of-flight (or other independent variable). The data is normally rebinned horizontally to match the number of pixels available. It is also possible to select horizontal scrolling so that each pixel corresponds to one channel or x value.

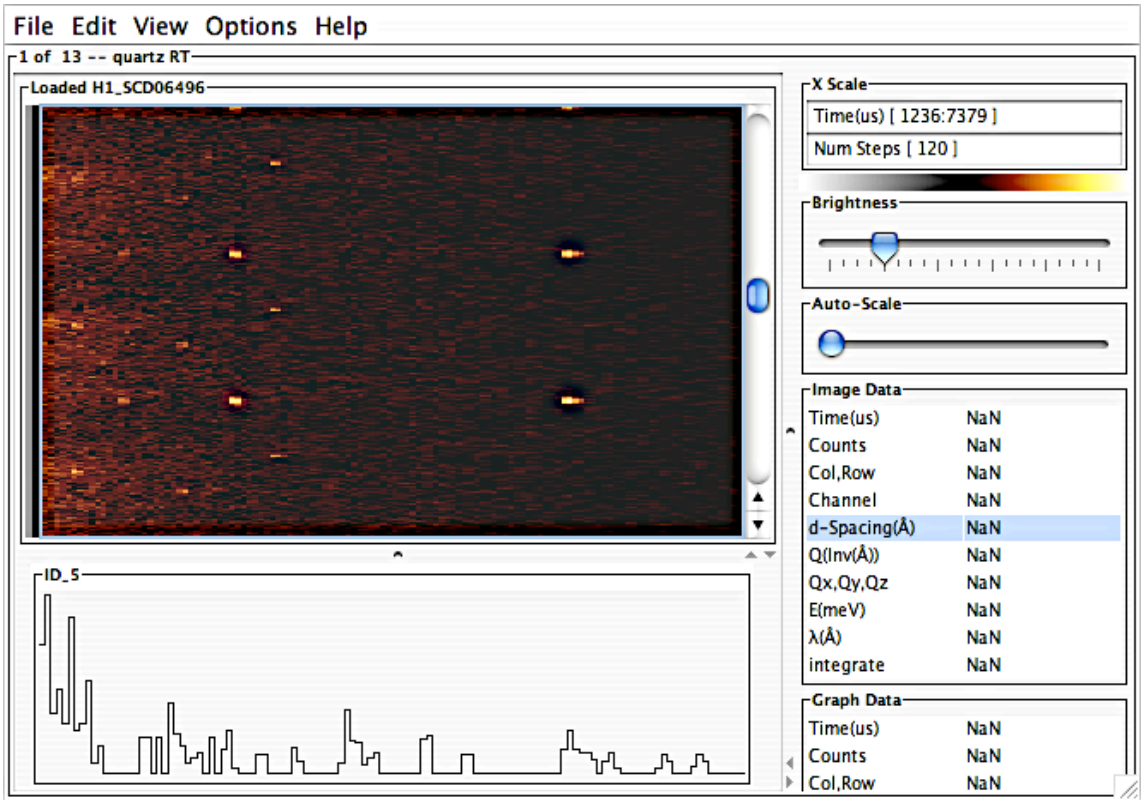

Figure 5-1, An image view of time-of-flight neutron scattering data.

Line Graph Display

The graph area below the image is used to display the line graph for the currently pointed-at row in the image view. It is possible to select additional spectra to view in the line graph area by pressing the **S** key on the keyboard while a row is pointed at. To select additional spectra beyond the second, press **Control-S** while a row is pointed at. To select a series of contiguous spectra, first select the initial spectrum using **S** or **Control-S**, and then press **Shift-S** while the final row is pointed at. You can also use the tree view to select spectra by using the **Set Flag** option in the right-click menu to select the highlighted spectra.

View Controls

On the right side of the Image Viewer window there is an area with controls including X-Scale, Brightness, and Auto-Scale. In the **X-Scale** box there are two text fields: Time(μs), and Num Steps. The **Time(μs)** text field is used to display and adjust the time range, in microseconds, used for the image view. To adjust this range, double-click on a number in the readout and enter a new value. The **Num Steps** text field is used to display and adjust the number of equally spaced bins for the selected time range. To adjust this range, enter a value in between the brackets.

The **Brightness** slider is used to control the brightness of the image in the image display. By moving the slider left or right, you can adjust the intensity of the pixels in the image view to a suitable level. The **Auto-Scale** slider is used to control the scale of peaks seen in the line graph display. If the slider is set to its far left setting, then an auto-scale feature will be used which automatically fits the highest peak into the line graph display.

Cursor Readout Areas

Underneath the view controls there is an area containing several cursor readouts. These readouts include time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate. The first cursor readout area is called **Image Data**. The Image Data cursor readouts provide information about the currently selected point in the image display. Below the Image Data area is an area called **Graph Data**. The Graph Data cursor readouts provide information about the currently selected point in the Line Graph Display. These readouts only respond when the cursor is in the line graph display or the image display. If **Graph Cursor Tracks Image Cursor** is unselected under the **Options** menu then these readouts will only respond when the cursor is in the line graph display.

Image View Edit Menu

The **Edit** menu for the Image View contains a few common ways to manipulate the selected spectra. The **Sum** option is used to sum all counts of either the selected or unselected spectra. The **Delete** option is used to delete all counts of either the selected or unselected spectra. The **Clear** option is used to clear all select flags that have been placed. The **Sort By** option is used to sort the line graph data. The vertical bar at the left of the image illustrates the sort order of the spectra. Spectra are initially sorted according to their group ID, but they may be sorted by any attribute. For more complex sorts involving up to three attributes, open the **Operations** menu, then open the **Edit List** submenu, and finally choose **Sort on group attributes**.

Image View Options Menu

The image view contains several options to change the appearance of the image in the Image View window. The **Color Scale** option is used to select a color scale to use when ISAW draws the image. You can view the current color scale in the area just beneath the X-Scale area. The **Graph Selected** option can be used to graph the selected line graphs. Up to 16 selected line graphs can be graphed in this way. The **Graph Cursor Tracks Image Cursor** is used to allow graph data readouts to respond when the cursor is in the image display. The **Horizontal Scroll** should be selected if you want every vertical line on the graphs to represent the same time bin. This allows for viewing a long range of time values accurately. If it is left unselected, bins will be compressed to fit inside the view area. The **Graph Rebinned Data** option is used to rebin data automatically so that each vertical line represents the same time. The **Link Views** option is used to link the current view with any additional views of the same DataSet that also have this option selected. This is on by default so that pointing the cursor in one view will affect all views of the same data.

Zoom Control

Spectra can be extremely detailed and contain thousands of values. To more closely examine the data, it is possible to zoom in to a particular region. The zoom region is specified by holding down the middle mouse button and dragging a box around the desired area. For computers with a single-button mouse, simply hold the shift key while dragging a box with the mouse button. To zoom out completely, double-click in the image area. This zoom feature can also be used with the Line Graph Display.

3D Viewer

The 3D View

The 3D view is designed to view detector positions and 2D slices of time-of-flight data in three-dimensional space. Each detector is represented by a different square (2D slice) and is placed on a circular path within the three-dimensional workspace. At the middle of the circular path is the origin with xyz axes. The neutron beam is represented by a red line striking the origin.

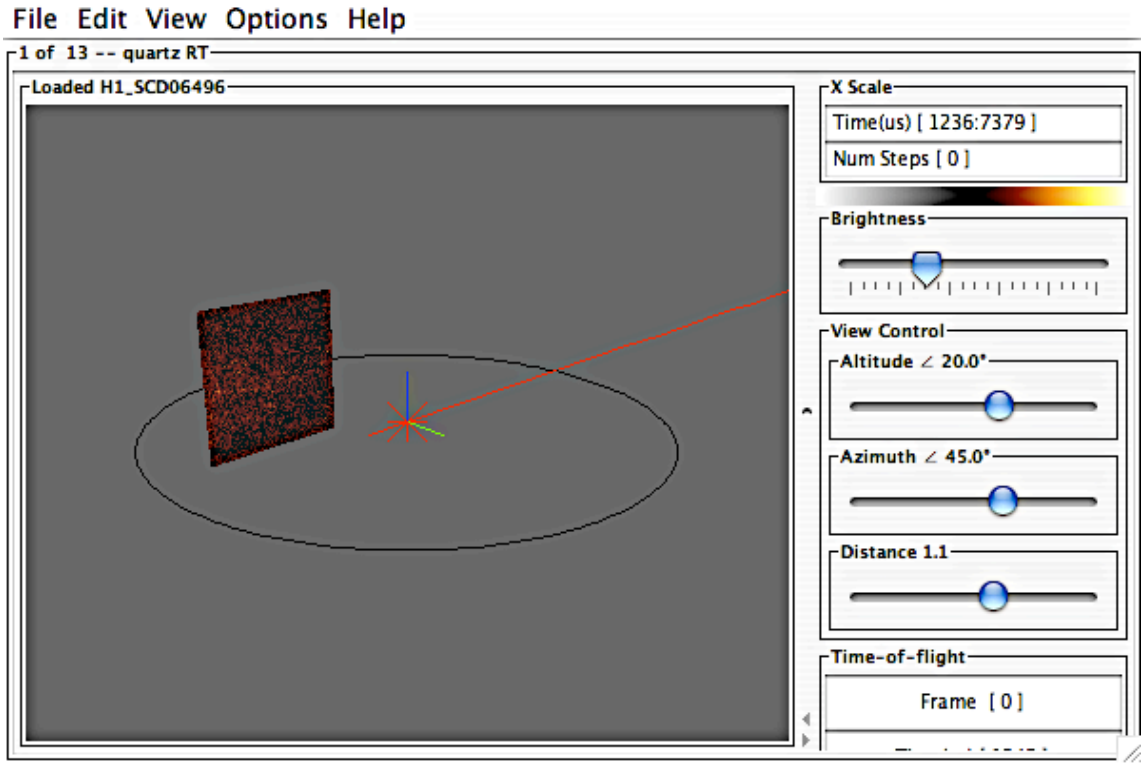

Figure 5-2, the 3D View of time-of-flight data in ISAW.

View Controls

On the right side of the 3D Viewer window there is an area with controls including X-Scale, Brightness, and View Control. In the **X-Scale** box there are two text fields: Time (μs), and Num Steps. The **Time(μs)** text field is used to display and adjust the time range, in microseconds, used for the 3D view. To adjust this range, double-click on a number in the readout and enter a new value. The **Num Steps** text field is used to display and adjust the number of equally spaced bins for the selected time range. To adjust this range, enter a value in between the brackets. The **Brightness** slider is used to control the brightness of the 2D slices in the 3D view. By moving the slider left or right, you can adjust the intensity of the pixels in the 2D slices.

In the area marked **View Control** there are three sliders: Altitude, Azimuth, and Distance. The **Altitude** slider is used to control the angle above or below the horizontal plane in the 3D view. The **Azimuth** slider is used to control the rotation around the vertical axis in the

3D view. The **Distance** slider is used to control the view distance from the origin in the 3D view.

Time-of-Flight Controls

The time-of-flight controls are used to adjust the time-of-flight data that is displayed in each slice in the 3D view. These controls are similar to those of any common video player. The **<<** and **>>** buttons are used to auto-step through time in the direction of the arrows. This auto-step will occur until you press the pause button. The **Pause** button is represented by the symbol **||** and is only used to pause the auto-step feature. The **<** and **>** buttons are used to step through the time-of-flight data one frame at a time in the direction of the arrow.

Pixel Data Readouts

The pixel data readouts include time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate. These readouts provide information about the currently selected pixel in a 2D slice within the 3D view.

3D View Edit Menu

See the Image View Edit Menu section...

3D View Options Menu

The **Options** menu provides a few options for adjusting the appearance of the 3D view. The **Color Scale** option is used to select a color scale for the 2D slices that appear within the 3D view. The current color scale can be viewed under the **X-Scale** area. The **Draw Groups** option is used to draw an asterisk at the center of the pixel element. The **Draw Detector Segments** submenu is used to select how detector segments are drawn in the 3D view. The **Link Views** option is used to link the current view with any additional views of the same DataSet that also have this option selected.

HKL Slice Viewer

The HKL Slice Viewer

The HKL Slice Viewer is designed to view two-dimensional slices of DataSets in Q_{xyz} space. This is most commonly used with single crystal diffraction data and requires an orientation matrix to be loaded. In the HKL slice view there are two axes, x and y, which both measure distance within HKL planes in units of inverse Angstroms.

Loading an Orientation Matrix

Opening the HKL Slice Viewer requires that you load an orientation matrix into the ISAW tree. An orientation matrix, or .mat file, is used by ISAW to describe the sample with respect to the diffractometer angles. These files can be written either by the user or automatically by software such as the Reciprocal Lattice Viewer. The following is a sample orientation matrix as outputted by the Reciprocal Lattice Viewer.

0.070018 0.021537 0.131717

0.021889 0.076477 -0.071833

-0.050633 0.051088 0.031275

7.324 10.217 12.994 97.610 100.479 66.227 873.089

0.000 0.000 0.000 0.000 0.000 0.000 0.000

When you have an orientation matrix, you can load it into the ISAW tree by clicking on the **Operation** menu, then the **Attributes** submenu, and finally choose the **Load Orientation Matrix** option.

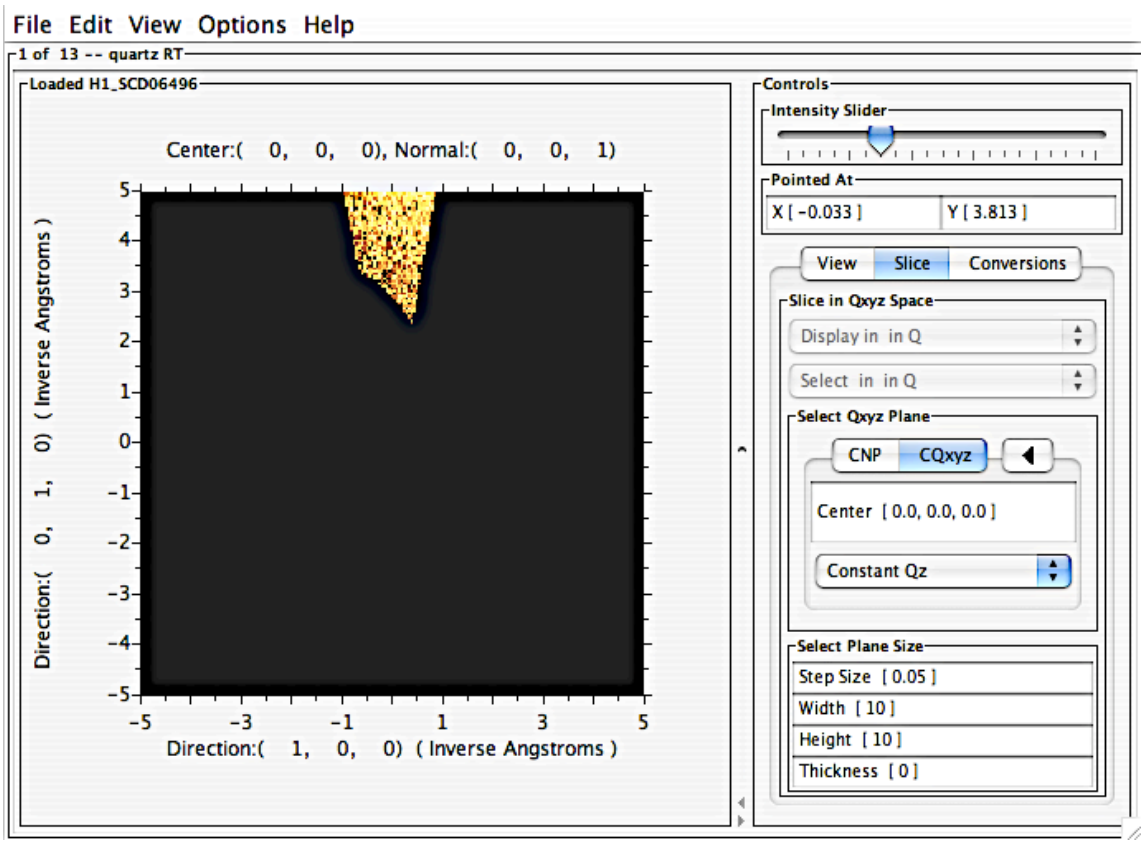

Figure 5-3, the HKL Slice View of time-of-flight data in ISAW.

View Controls

The view controls are always visible regardless of which tab is open. The **Intensity Slider** is used to adjust the intensity of the two-dimensional slice in the HKL slice view. The **Pointed At** cursor readout area will display the X and Y coordinates of the currently pointed at point within the HKL slice view.

View Tab

The HKL Slice Viewer's view tab is used to change the appearance of the HKL slice view and to add overlays to the view for visual reference. The **Color Scale** option is used to select a color scale for the two-dimensional slice that appears within the HKL slice view. The **Marker Overlay** option is used to display markers at points with integer HKL values. Selecting this option requires an orientation matrix. The **Axis Overlay** option is used to display a grid axis overlay in the HKL slice view. To modify the grid's appearance, click the **Edit** button. The **Selection Overlay** is used to display a selection overlay in the HKL slice view. To add

the overlay, simply click and drag a box around the desired area within the HKL slice view while the **Edit** window is open. To modify the overlay's appearance, click the **Edit** button. The **Annotation Overlay** option is used to display an annotation overlay in the HKL slice view. To change the annotation's appearance, click the **Edit** button.

Slice Tab

Slice in Qxyz space info...

The **Slice Tab** is used to manipulate the slice and HKL planes. The **Select Qxyz Plane** area is used to adjust the size and position of the Q_{xyz} plane. The **CN Tab** is used to select the center and the normal in either Q_{xyz} or HKL coordinates. The **CPP Tab** is used to select the the center, a vector to represent one unit in the x-direction, and a vector to represent one unit in the y-direction. The **CNP Tab** is used to select the center, normal, and one point on the plane. The **CQxyz Tab** is used to select the center and choose to step along planes of constant H,K, or L or constant Qx, Qy, or Qz.

The **Select Plane Size** area is used to specify a plane size for the HKL slices. The **Step Size** option is used to specify the step size for a series of slices in the HKL view. The **Width** and **Height** fields are used to specify the width and height of the slice shown in the HKL slice display. The **Thickness** field is used to specify a thickness for each HKL slice.

The **Step In/Out** area is used to control the step setting for the HKL viewer. The **Step Depth** input is used to specify how far the viewer should step in or out of the slice. The controls consist of **<** and **>** buttons. Pressing the **<** button will cause the viewer to step out once according to the Step Depth setting, and pressing the **>** button will cause the viewer to step in once according to the Step Depth setting.

Conversions Tab

The Conversions Tab contains a number of cursor readouts that provide information about the currently selected point in the HKL slice display including time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate.

Saving DataSets with the HKL Slice Viewer

The HKL Viewer **File** menu provides a save function that allows you to save the current DataSet as a new file. Supported formats include NeXus, GSAS, ISAW XML, ZIP, and ISD.

HKL Slice Viewer Edit Menu

See the Image View Edit Menu section...

HKL Slice Viewer Options Menu

The **Link Views** option is used to link the current view with any additional views of the same DataSet that also have this option selected.

Scrolled Graph View

The Scrolled Graph View

The Scrolled Graph view is designed to view all of the spectra for a DataSet in a separate line graph. The left side of the window contains the line graph display with a vertical scroll bar. The line graphs are arranged in ascending order. Like the line graph display in the Image Viewer, the Scrolled Graph View allows you to select spectra by setting select flags. To set a select flag, point the mouse cursor at a graph and press **s** to set a select flag. To select additional spectra beyond the first, hold **Control** while pressing **s**. To select a range of spectra, first select a starting spectra, then hold **Shift** and press **s** on the last spectra in the series that you would like to select. It is also possible to toggle a select flag on and off. To do this, press **T** while pointing at a graph. If the graph is unselected, then this will add it to the group of selections. Pressing **T** again will remove it from the group of selections.

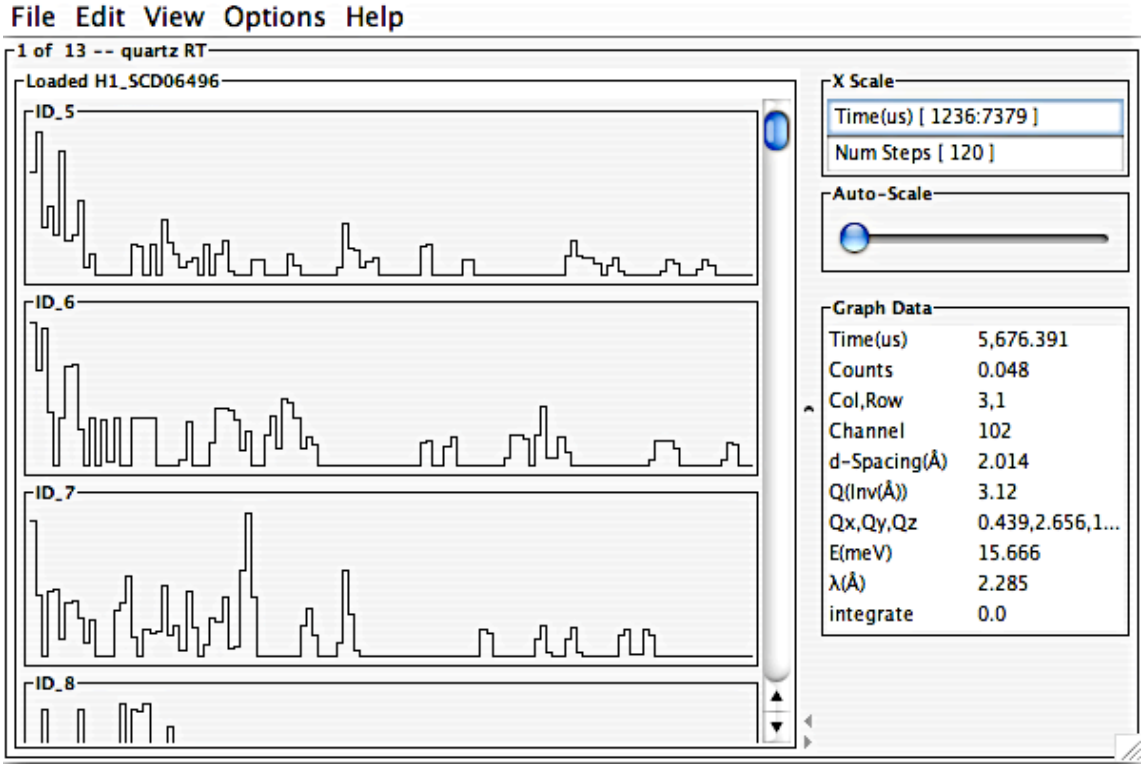

Figure 5-4, the Scrolled Graph View for time-of-flight data in ISAW.

View Controls

On the right side of the Scrolled Graph View window there is an area with controls including X-Scale and Auto-Scale. In the **X-Scale** box there are two text fields: Time(μs), and Num Steps. The **Time(μs)** text field is used to display and adjust the time range, in microseconds, used for the line graphs. To adjust this range, double-click on a number in the readout and enter a new value. The **Num Steps** text field is used to display and adjust the number of equally spaced bins for the selected time range. To adjust this range, enter a value in between the brackets.

The **Auto-Scale** slider is used to control the scale of peaks seen in the line graphs. If the slider is set to its far left setting then an auto-scale feature will be used which automatically fits the highest peak into the line graph display.

Graph Data Cursor Readouts

The Graph Data cursor readouts provide a variety of information abut the currently pointed-at point in a line graph including time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate.

Scrolled Graph View Edit Menu

See the Image View Edit Menu section...

Scrolled Graph View Options Menu

The Options menu contains several options for adjusting the appearance of line graphs in the Scrolled Graph view. The **Horizontal Scroll** option is used to make every vertical line on the graphs represent the same time bin. This generally allows for more accurate viewing of a long range of time values. If this option is left unselected, then bins will be compressed to fit inside the view area. The **Graph Rebinned Data** option is used to rebin data automatically so that each vertical line represents the same time. The **Link Views** option is used to link the current view with any additional views of the same DataSet that also have this option selected.

Selected Graph View

The Selected Graph View

The Selected Graph View is designed to allow users to view the time-of-flight line graphs and their associated counts for user-selected data blocks. The line graph display shows the selected spectra as line graphs with scattering intensities as the dependent variable and the user-defined x-value as the independent variable. The default x-value is time-of-flight in units of microseconds.

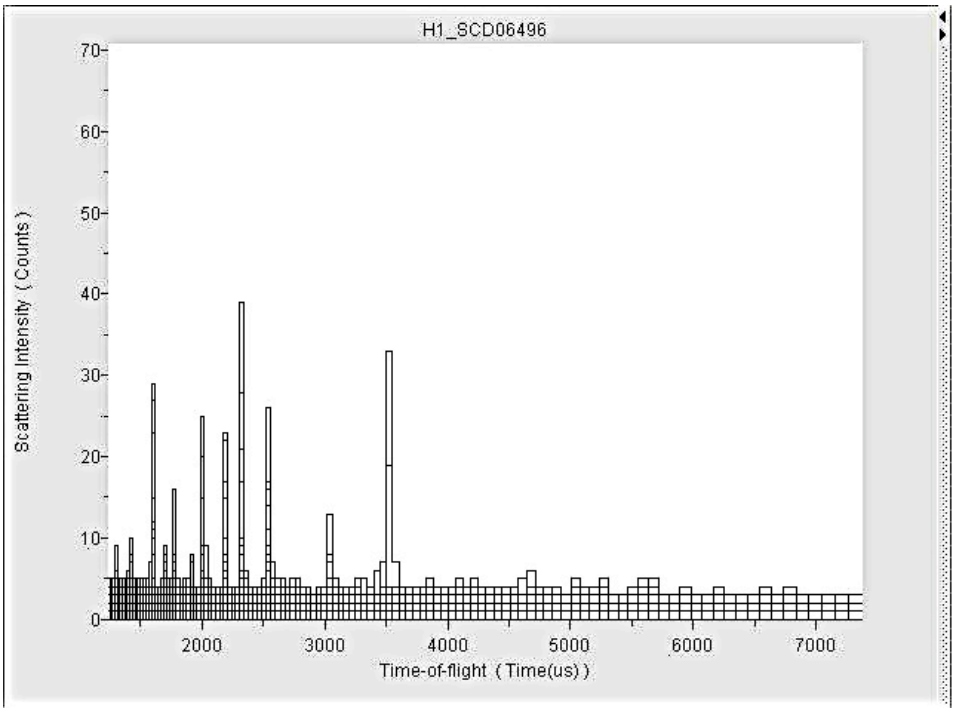

Figure 5-5, The line graph display in the Selected Graph View.

Selected Graph View Edit Menu

See the Image View Edit Menu section...

Selected Graph View Function Controls

The Selected Graph View Function Controls are used to modify the appearance of graphs in the line graph display. The function controls can be accessed by selecting the **Function Controls** option in the **Options** menu. The function controls window will then appear with several options that can be adjusted. The **Line Selected** option allows the user to select a line graph, or Group ID, to modify. The next three options – **Line Style, Line Width,** and **Line Color**, are used to modify the line style for the currently selected line graph in the Line Selected menu. For example, to change the color of a line graph, choose the Group ID of the line graph that you would like to modify from the **Line Selected** menu, and press the **Line Color** button to choose a color.

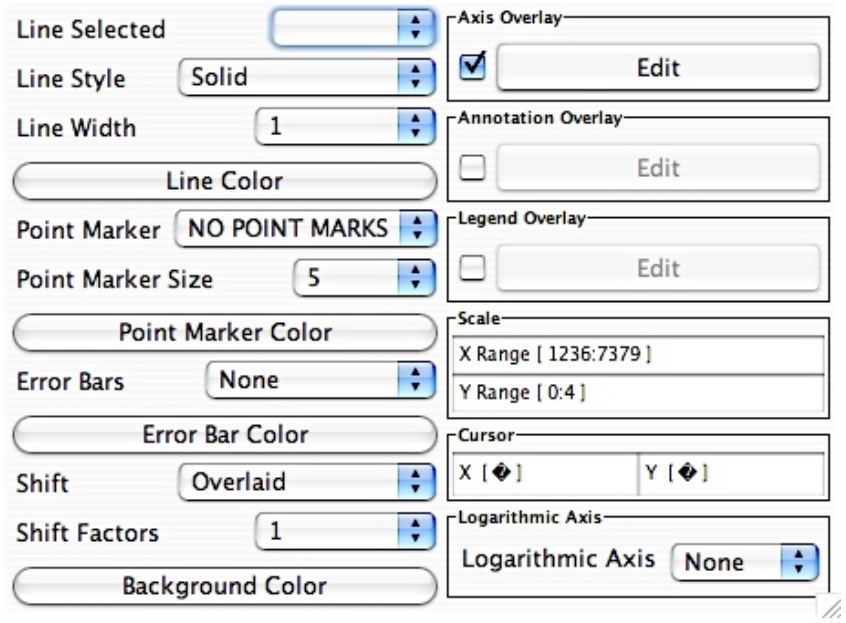

Figure 5-6, The function controls window for the Selected Graph View.

The **Point Marker, Point Marker Size,** and **Point Marker Color** are used to create a point marker in the graph display and change the marker's size and color. The **Error Bars** and **Error Bar Color** is used to display and edit the error bars in the graph display for the currently selected Group ID. **Shift** and **Shift Factors** are used to specify a shift for the graph. The **Shift** option is used to specify a type of shift for the graph, while **Shift Factors** is used to modify the shift factor that will be applied to the graph.

The next three options are used to display overlays in the graph view. The **Axis Overlay** is used to display additional axes information on the graph. The **Edit** button can be used to change its appearance. The **Annotation Overlay** option is used to add text to the graph display. This can also be edited by pressing the **Edit** button. The **Legend Overlay** option is used to add a legend to the graph display. Once again, its appearance can be modified by pressing the **Edit** button.

The next three controls are used to manipulate the axes for the graph. The **Scale** option is used to edit the X and Y range for the graph. The **Cursor** readout displays the current XY coordinates of the cursor in the graph display. The **Logarithmic Axis** option is used to specify a logarithmic scale for the X and Y axis.

Selected Graph View Options Menu

Other than the Function Controls, there are two other options in the **Options** menu. The **Show Pointed At** option is used to show the currently pointed at data block in a linked view. The **Link Views** option is used to link the current view with any additional views of the same DataSet that also have this option selected.

Difference Viewer

The Difference Viewer

The Difference Viewer is designed to allow users to view the difference between two line graphs for user-selected data blocks. Before launching the Difference Viewer, there must be at least two selected line graphs, or Group IDs. To do this, simply double click the Group IDs that you would like to select. You can also click on a Group ID, then right-click, and choose **Select Highlighted Data Blocks**. The Difference Viewer can then be opened by selecting it from the **Viewer** menu or the right-click menu. In order to use the right-click menu, first select the Group IDs, then click on the histogram icon (usually labeled H1_xxxx), then right-click and select the Difference Viewer.

The difference view shows the selected spectra and difference spectrum as line graphs with scattering intensities as the dependent variable and the user-defined x-value as the independent variable. The default x-value is the x-scale of the selected spectra. By default, the selected spectra will be shown in blue near the top of the graph along the zero line and the difference spectrum will be shown in magenta beneath the zero line. The difference spectrum will not be scaled on the x axis by default; however, this can be changed by selecting **Shift Difference** in the **Difference Options** window.

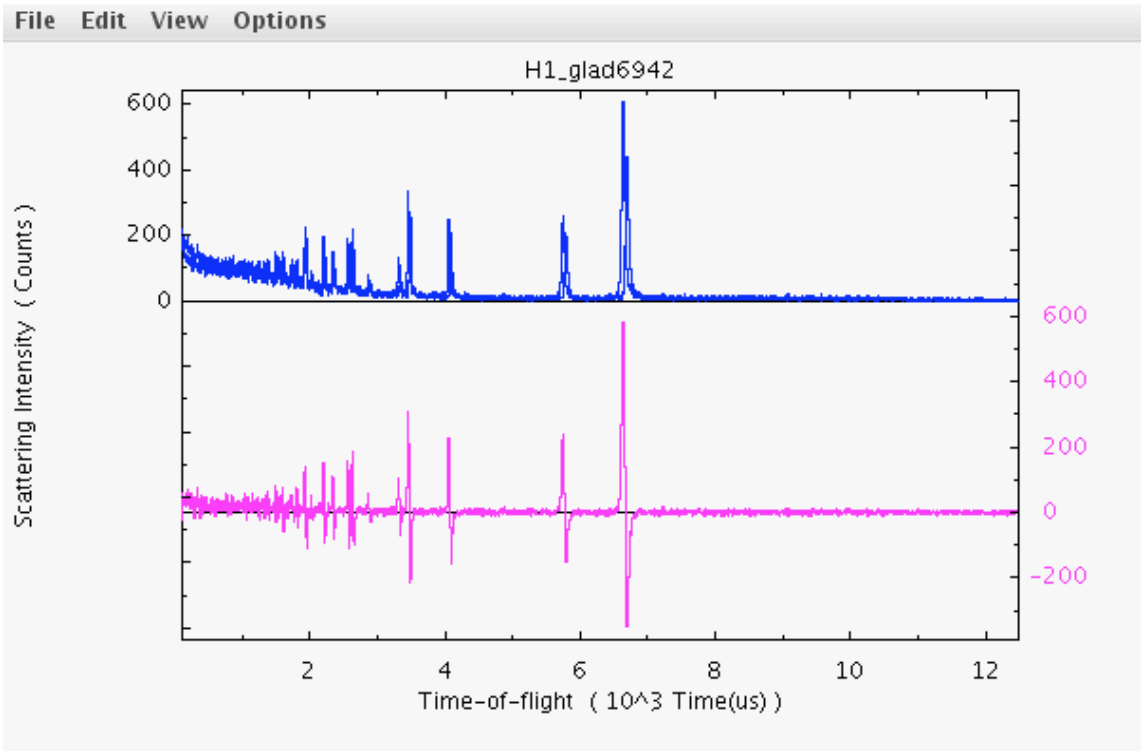

Figure 5-7, The difference view for time-of-flight data. The selected Group ID spectra are near the top of the graph in blue, and the difference spectrum is below in magenta.

Difference Viewer Edit Menu

See the Image View Edit Menu section...

Difference Viewer Function Controls

The Difference View Function Controls are used to modify the appearance of line graphs in the difference view. The function controls can be accessed by selecting the **Function Controls** option in the **Options** menu. The function controls window will then appear with several options that can be adjusted. The **Line Selected** option allows the user to select a line graph, or Group ID, to modify. The next three options – **Line Style, Line Width,** and **Line Color**, are used to modify the line style for the currently selected line graph in the Line Selected menu. For example, to change the color of a line graph, choose the Group ID of the line graph that you would like to modify from the **Line Selected** menu, and press the **Line Color** button to choose a color.

The **Point Marker, Point Marker Size,** and **Point Marker Color** are used to create a point marker in the difference view and change the marker's size and color. The **Error Bars** and **Error Bar Color** is used to display and edit the error bars in the difference view for the currently selected Group ID. **Shift** and **Shift Factors** are used to specify a shift for the graph. The **Shift** option is used to specify a a type of shift for the graph, while **Shift Factors** is used to modify the shift factor that will be applied to the graph.

The next three options are used to display overlays in the graph view. The **Axis Overlay** is used to display additional axes information on the graph. The **Edit** button can be used to change its appearance. The **Annotation Overlay** option is used to add text to the graph display. This can also be edited by pressing the **Edit** button. The **Legend Overlay** option

is used to add a legend to the graph display. Once again, its appearance can be modified by pressing the **Edit** button.

The next three controls are used to manipulate the axes for the graph. The **Scale** option is used to edit the X and Y range for the graph. The **Cursor** readout displays the current XY coordinates of the cursor in the graph display. The **Logarithmic Axis** option is used to specify a logarithmic scale for the X and Y axis.

The **Difference Options** button is used to modify the appearance of the difference graph in the difference view. Pressing the **Edit** button will open the **Difference Options** window. The first options in this window are the **Difference** menus. These drop-down menus are used to select the Group IDs of the spectra that will be used to produce the difference spectrum. The first menu is used to select the Group ID that will be operated on, and the second menu is used to select the Group ID that will be subtracted from the first Group ID. Beneath the Difference menus is a checkbox called **Shift Difference**. If this box is checked, then the difference spectrum will be placed beneath the selected Group IDs without any scale. If the box is unchecked, then the difference spectrum will be placed on the same scale and axis as the selected Group IDs.

Selected Table Views

The Selected Table Views

The selected table views are designed to view user-selected spectra, or Group IDs, as a table of values that can be viewed and sorted in a variety of different ways.

The GRX_Y Table

The GRX_Y table is used to produce a table of the selected spectra with either X vs. Y or time bin boundary vs. Y. It is usually used when all of the spectra do not have the same X values. Spectra are listed vertically in ascending order and each X-value represents the time bin for that spectrum, while each Y-value represents intensity.

Parallel y(x) Table

The Parallel y(x) Table is used to produce a table of the selected spectra with either X vs. Y or time bin boundary vs. Y. This table displays spectra horizontally in ascending order. Each X-value represents the time bin for that spectrum, while each Y-value represents intensity.

The Parallel y(x) Table also contains special controls and readouts. Checking the **Show All Groups** checkbox will display all available groups in the table view. If this option is not checked, then only user-selected groups will be displayed in the table view. The **Format** text box is used to enter Fortran formats such as F5.3, I4, E8.2, etc. The **Table Readout Area** provides information about the currently selected cell in the table view including time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate. These readouts only respond when the cursor is in the image area.

The Parallel y(x) Table also contains a special viewer menu called **Select**. The **Rectangle** option under this menu is used to mark the spectra corresponding to the selected rectangle from the table view.

Instrument Table

The Instrument Table is used to produce a table that lists attributes of all spectra. The attributes that are displayed can be specified by selecting the desired fields. The table can then be sorted, and selected spectra in a given range can be marked. The table displays selected spectra vertically in ascending order, and default fields include group ID, scattering angle, and total count.

The Instrument Table also contains several special controls and readouts. The **Add Field** option is used to add a field to the current table. The **Delete Field** option is used to delete a field from the current table. The **Sort on Field** option is used to arrange values for a specific field in ascending order. To use this option, click on a cell in the field that you would like to arrange, and click the Sort on Field button. The **Format** text box is used to enter Fortran formats such as F5.3, I4, E8.2, etc. The **Table Readout Area** provides information about the currently selected cell in the table view including time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate. The readouts only respond when the cursor is in the image area.

The Instrument Table also uses a special viewer menu called **Select**. The **Rectangle** option under this menu is used to mark the spectra corresponding to the selected rectangle from the table view. The **Clear All** option is used to clear all select flags from the table. The **New Attribute** option is used to add row, column, crate, slot, and input attributes to the table.

Common Viewer Menus

The selected table views all contain similar information in their menus. The **File** menu can be used to print the viewer window as it appears on the screen, save the current view as a new DataSet, and save the viewer window as a JPEG picture file.

The **Edit** menu contains a **Spreadsheet** option that can be used to copy the selected area of the current table for pasting into a spreadsheet program. An integrate function is also available for the Spreadsheet option.

The **Options** menu is used to add information to the current table view. The **Show Errors** option is used to display a new field within the table that displays errors. The new field name will begin with **Er:**. The **Show Indices** option is used to display a new field within the table that displays indices. The new field name will begin with **Ind:**.

Table Generator

The Table Generator

The Table Generator is used to create a custom table that is arranged by user-specified fields. The viewer contains three basic columns that are used to create a custom table. The

first column is called **Possible Fields**. It contains a list of fields available for use within a table in ISAW. To add one of the fields to a table, simply select it and press the **Add** button. This action will move the field into the **Display Fields** column. This column is used to display all of the fields that will be used in the table. Pressing the **Up** and **Down** buttons will change the order of the display fields, and pressing **Remove** while a field is selected will remove it from the Display Fields column. The last column is the **Controls** column. It is used to modify the appearance of the table, save the table, create the table, and choose data for the table. The **Selected Indices** readout displays the identifying numbers of the currently selected data blocks. The **Select Group Indices** button will open a text field that allows you to enter the Group ID of the blocks that you would like to select. Check the **Use All Groups** checkbox to add all available groups for the current DataSet to the table. The **Order** menu is used to change the appearance and layout of the table.

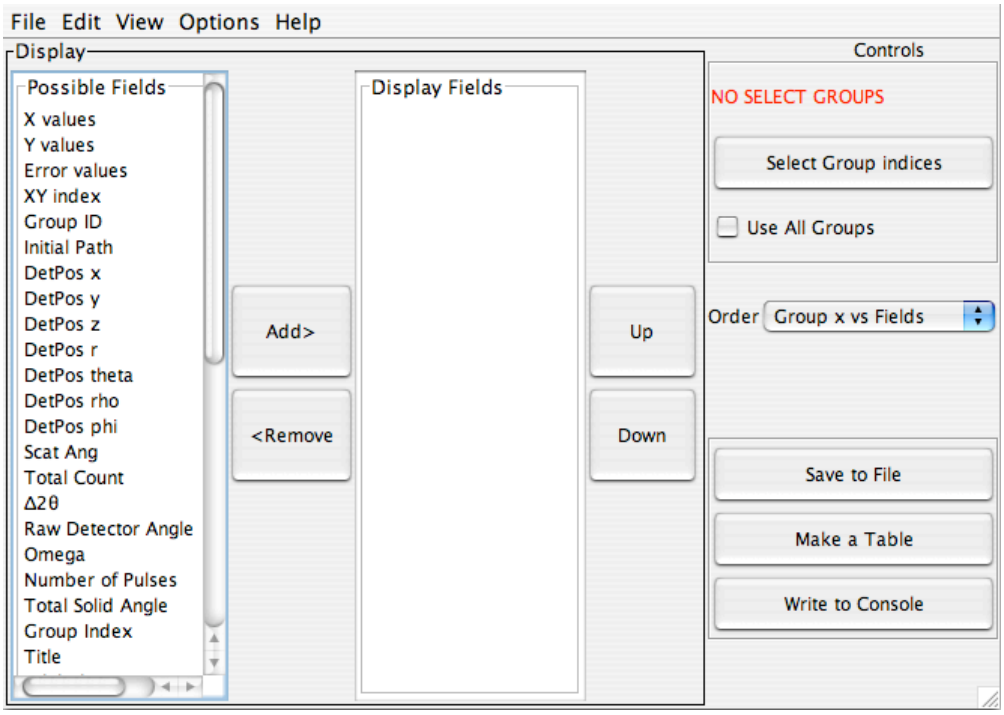

Figure 5-8, The ISAW Table Generator window.

In addition to the variety of customization options in the Table Generator, there are also several different ways to view the table data that is produced by this viewer. The **Save to File** button is used to save the current table as a separate file that can be opened by any program; however, when saving, it is necessary to assign a file extension that is supported by the program that will be used to open it. The **Make a Table** button is used to display a table with the currently selected display fields and group indices. The table will be displayed in a new window within ISAW. The **Write to Console** button is used to display table data in the console that ISAW was launched from. For example, in Mac OS X, the table data will be displayed in the system's bash terminal. In Windows XP, the table data will be displayed in the system's command prompt window.

Reciprocal Lattice Viewer

The Reciprocal Lattice Viewer

The Reciprocal Lattice Viewer is designed to allow users to work with single crystal diffraction data in a three-dimensional environment by displaying voxels in reciprocal space that correspond exactly to time-of-flight histogram bins from the area detectors. To launch the Reciprocal Lattice Viewer, click on the Macros menu, then open the Instrument Type submenu, then open the TOF_NSCD submenu, and then click on SCDReciprocalLattice. This will prompt a new window to open that contains several customization options.

The Reciprocal Lattice Viewer Launch Window

After selecting the Reciprocal Lattice Viewer from the Macros menu, a launch window will appear with several customization options. Modifying these correctly is an important step in initializing the viewer for use. The first option is **Data Directory**. Use this option to choose the directory where your run data is stored. The next option is called **Calibration File** and it is used to choose a calibration file for the run data. The **Orientation Matrix** option is used to select an orientation matrix for the run data. It is not necessary to load an orientation matrix at this point if you plan to create one using the Reciprocal Lattice Viewer.

The **Run Numbers** field is used to specify all of the run numbers that will be used by the Reciprocal Lattice Viewer, separated by commas. A series os runs can be delineated by separating values with a colon. For example, you might want to use all of the run files from 8336 to 8339, and the run file 8341. To specify these run numbers, you would type 8336:8339,8341 into the Run Numbers field.

The next three options have to do with threshold levels and contour levels. The first option is called **Threshold Levels** and is used to limit how many counts are displayed in the viewer. If the value is set too low then many low-count bins will be included which causes the system to slow down; however, in cases where the experimental data is weak, then the threshold level can be reduced to increase the number of bins that appear in the viewer. A normal value for this setting is 60.

It is also possible to view contour levels in the Reciprocal Lattice View. The **Show Contours** option is used to add a contour surface to the voxels in the Reciprocal Lattice View. It should be noted that selecting this option requires significantly more computing power. Once contours are enabled, it is possible to define how many should be included in the view. The **Contour Level** option is used to limit how many contour levels are displayed in the viewer. A high setting for this option is usually recommended for the initial run.

The next two options deal with customization of the reciprocal space graph. The first option is called **Show Regions**. Checking this checkbox will display a three-dimensional outline of the volume covered by the detector for the instrument that collected the data. The run numbers for the data will be displayed on the large end of the region outline. The sec-

ond option is called **Show HKL Markers** and is used to display marks at integer HKL points in the Reciprocal Lattice View. Selecting this option requires that an orientation matrix is loaded.

The final option is **Calculate FFTs**. Checking this checkbox will allow for initial calculation of FFTs when the data is loaded. FFTs are covered in more detail later in the **Calculate FFT's of Projections** section.

When you are finished making changes to the launch window, press **Apply** and the Reciprocal Lattice Viewer will open.

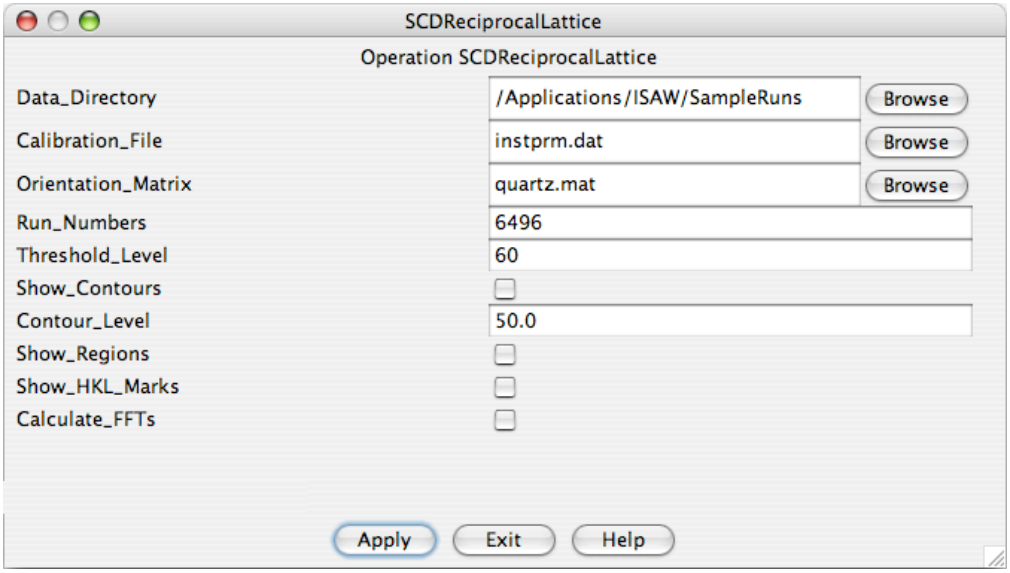

Figure 5-9, The Reciprocal Lattice Viewer launch window.

View Controls

The Reciprocal Lattice Viewer uses a three-dimensional environment to display time-of-flight data in reciprocal space. A series of controls similar to those found in the 3D Viewer are used to help you navigate this environment. The **Altitude** slider is used to control the angle above or below the horizontal plane that the Reciprocal Lattice Viewer draws. The **Azimuth** slider is used to control rotation around the vertical axis in the viewing window. The **Distance** slider is used to control the distance from the origin that is displayed in the viewing window. It is often useful to use your keyboard's arrow keys for making fine adjustments to these sliders.

The Reciprocal Lattice Viewer also uses two different types of projection for viewing data in three dimensions. The **Projection** checkbox is used to toggle between orthographic projection and perspective projection. When this box is checked, an orthographic projection will be used; otherwise a perspective projection will be used.

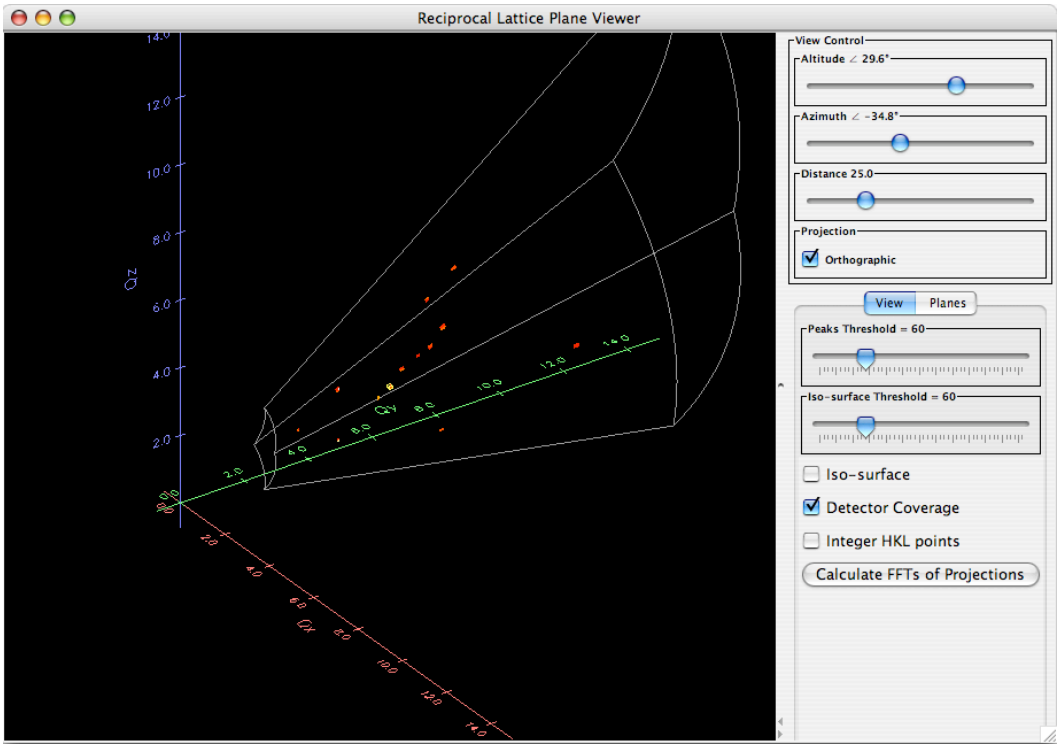

Figure 5-10, The Reciprocal Lattice Viewer showing single-crystal diffraction data in reciprocal space.

The View Tab

The View Tab is used to change how data appears in the Reciprocal Lattice Viewer. The first options are the **Range of |Q| Shown** entries. These entries are used to restrict the range of |Q| that will be displayed in the viewing window. Peaks that have |Q| outside the [Q_min, Q_max] interval will not be shown. Enter the desired interval in the form [Q_min : Q_max]. The default interval is [0 : 30].

The next items in the View Tab are the threshold sliders. The **Peaks Threshold** slider is used to control the cutoff level above which bins are displayed and used in subsequent processing. In cases where experimental data is weak, it is useful to move this slider to a lower setting so that low-count bins can be viewed.

It is possible to toggle contour surfaces on and off. If the **Iso-surface** checkbox is checked, then contours will be drawn over voxels in the Reciprocal Lattice View. The **Iso-surface Threshold** slider can then be used to control the single level at which iso-level contours are drawn. The lower the setting on this slider, the more computing power that will be required to draw the contours.

The **Detector Coverage** checkbox is used to turn on and off a three-dimensional outline of the volume covered by the detector for the instrument that collected the data. The run numbers for the data will be displayed on the large end of the region outline.

The final option is called **Show HKL Markers** and is used to display marks at integer HKL points in the Reciprocal Lattice View. Selecting this option requires that an orientation matrix is loaded.

Selected Point Data

The Reciprocal Lattice Viewer can also provide data about the selected point in the reciprocal space. These readouts will respond when a point on a voxel is selected and include time, the number of counts, the currently selected column/row, channel, d-spacing, Q value, Q space coordinates, energy, wavelength, and integrate.

Calculate FFTs of Projections

The final button in the View Tab is called **Calculate FFTs of Projections**. The Calculate FFT's of Projections operation is used to approximate the direction of normals for planes in a crystalline lattice within reciprocal space. The direction of three linear independent plane normals can then be used to generate an orientation matrix for the sample.

FFT, or Fast Fourier Transform, projects the Q values in reciprocal space into one-dimensional lines pointing in various directions. These lines are Fourier transformed to identify directions that represent normals to families of planes in reciprocal space. The Fourier transformed projected data is then displayed as rows in an image view where peaks are arranged in order of increasing d-spacing.

The plane normals are displayed in the three-dimensional view as a sequence of light blue boxes pointing in the direction of the normal. The distance between these boxes corresponds to the length of an edge of the unit cell in real space. The d-spacing can be found on the border of the graph part of the image view.

Planes Tab

The Planes Tab provides tools for specifying user-defined HKL planes in reciprocal space. This is useful if you want to create an orientation matrix based on specific planes of symmetry that you choose.

The first section contains several general tools for use in defining planes. The first is called **Qxyz Readout**, and it displays the Qxyz coordinates of the currently selected peak. To select a peak, simply click on a point on a voxel in reciprocal space. The **Origin Control** is used to change the view center of the viewer window by clicking on a peak and then pressing the Select button. Pressing the select button will also choose the initial point for two user-specified vectors that are used to identify a family of planes. The Reset button will reset the view center to the Qxyz origin.

The next two options are used to specify vectors that help define planes of symmetry in reciprocal space. The **(+)Control** is used to change the endpoint of one user-specified vector to the last peak that you clicked on. The **(*)Control** is used to change the endpoint of a second user-specified vector to the last peak that the user clicked on. By defining vectors with these buttons, it is possible to specify a plane of symmetry using the following tools.

Constant h Planes, Constant k Planes, Constant l Planes

These areas allow you to specify a family of planes with a constant h, k, or l value in reciprocal space based on the vector that is defined with the tools under the Planes Tab. The first option is called **User**. The User button will create the constant h, k, or l plane information based on the normal to the plane determined by the two user-specified vectors + and *. The **FFT** button is used to create the constant h, k, or l plane information based on the normal to the plane determined from the last selected row in the FFT image view window. This window can be opened by choosing the **FFTs of Projections** button in the

View Tab. The **Filter On/Off** option is used to restrict visible data to those which lie within 0.1 units of an integer value.

When constant h, k, and l planes have been defines, the corresponding orientation matrix can be written to a file by clicking on **Write Orientation Matrix** and then specifying a file name. A list of the bins above the specified threshold can also be written to a file by clicking on **Write Peak Data File** and then specifying a file name.

SAND Wedge Viewer

The SAND Wedge Viewer

The SAND Wedge Viewer is an interactive analysis tool for visualizing small-angle neutron diffraction data, including the ability to make wedge, double wedge, annular, and elliptical selections. Once a selection is made on the image, the graph will display the intensity values per hit as a function of distance in Q. To run the SAND Wedge Viewer, open the **Macros** menu, then open the **Instrument Type** submenu, then open the **TOF_NSAS** submenu, and then select **Sand Wedge Viewer.** A file-chooser box will appear to select the data file that you would like to analyze. Click the **Apply** button to launch the SAND Wedge Viewer.

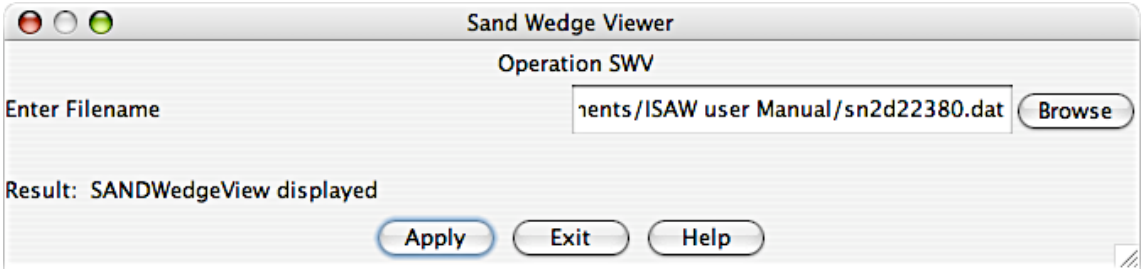

Figure 5-11, The SAND Wedge Viewer launch window.

Opening Files

In order to analyze data properly, the SAND Wedge Viewer requires that you first run reduction and integration routines on your small-angle diffraction DataSet. The result of these routines should be a **.dat** file with four columns of numeric data: x-values, y-values, intensity, and error. By default, the Sand Wedge Viewer assumes that data in the .dat file will construct a 200 row by 200 column array. If other dimensions are desired then the file should contain header information. Lines with header information should begin with a **#** symbol, followed by the attribute with a colon (**:**), and then finally the value. For example, to specify a data file having dimensions of 100 x 150, with an x-axis labeled "Angle" in units of degrees, you would enter the following into the header of the data file.

ROWS: 100
COLUMNS: 150
X Label: Angle
X Units: Degrees

The attributes that are supported by the SAND Wedge Viewer include row, column, x label, y label, z label, x units, y units, and z units. The attribute labels are not case sensitive.

The SAND Wedge View

After clicking Apply on the SAND Wedge Viewer launch window, the SAND Wedge Viewer should open and appear similar to the image in Figure 5-11. If it does not, go back to the launch window and make sure that you are loading a correctly formatted **.dat** file.

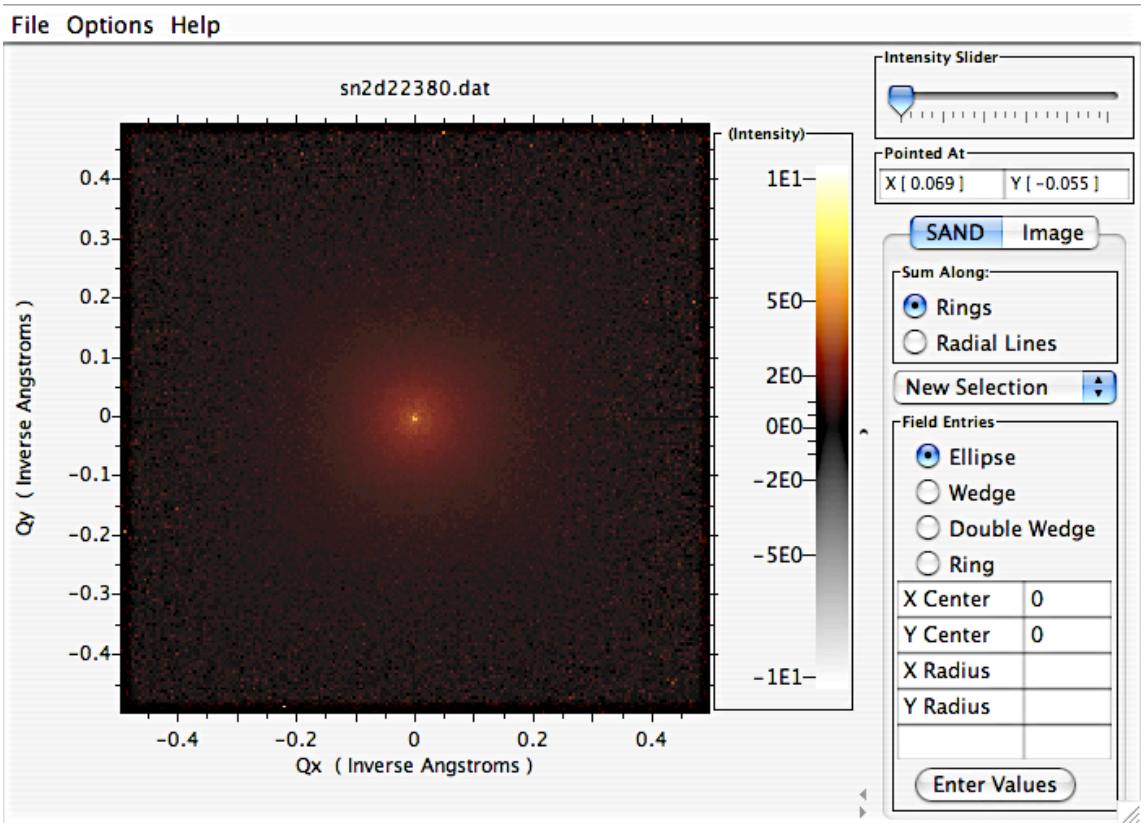

Figure 5-12, The SAND Wedge Viewer.

The SAND Wedge graph can be seen in the large left part of the SAND Wedge Viewer. It has axes of Qx and Qy, both measured in inverse Angstroms. The file name of the currently viewed data is shown at the top of the graph and the intensity scale is shown at the right of the graph. The intensity scale is linked to the image view so that when an area in the SAND Wedge graph is selected, the corresponding intensity will be indicated on the intensity scale. The intensity can be adjusted by moving the Intensity Slider left and right. Moving the intensity slider will also modify the intensity scale to reflect the appropriate intensity values for the SAND Wedge graph.

Common Readouts

The Selection Editor can be found on the right side of the SAND Wedge Viewer. The first option is the **Intensity Slider**. This slider is used to adjust the intensity of the currently displayed image in the SAND Wedge View. The **Pointed At** boxes are used to display the x,y coordinates of the currently pointed-at peak in the SAND Wedge View.

SAND Selection

The next area is displayed when the **SAND** button is selected on the right side of the SAND Wedge Viewer. This is used to manually specify and edit selections in the SAND Wedge Viewer. The first option, **Sum Along**, is used to specify whether selections will be summed along rings or radial lines. Underneath this option is the selection menu. This menu is used to choose from currently existing selections in the SAND Wedge View. The **Field Entries** area is used to specify the type of selection that you would like to make on the image – wedge, double wedge, annular, and ellipse. The fields below this, marked X Center, Y Center, X Radius, and Y Radius, are used to enter the selection parameters. Once a selection is made on the image, the graph will display the intensity values per hit as a function of distance in Q.

Image Selection

The Image Selection method for making selections and annotations can be used by clicking on the **Image** button on the right side of the SAND Wedge Viewer. These options are used to add overlays to the image, each with a different function and appearance. Clicking on the checkbox under each option will enable the corresponding overlay, and clicking the **Edit** button next to each checkbox will open a dialog box containing several options for changing the appearance of each overlay. To add an overlay to the image, simply open the **Edit** box, click on the type of marker to add to the image, and then click and hold in the image view to edit the marker's shape and location. When the **Selection Overlay** is used, a window will appear displaying the a graph with the intensity values per hit as a function of distance in Q.

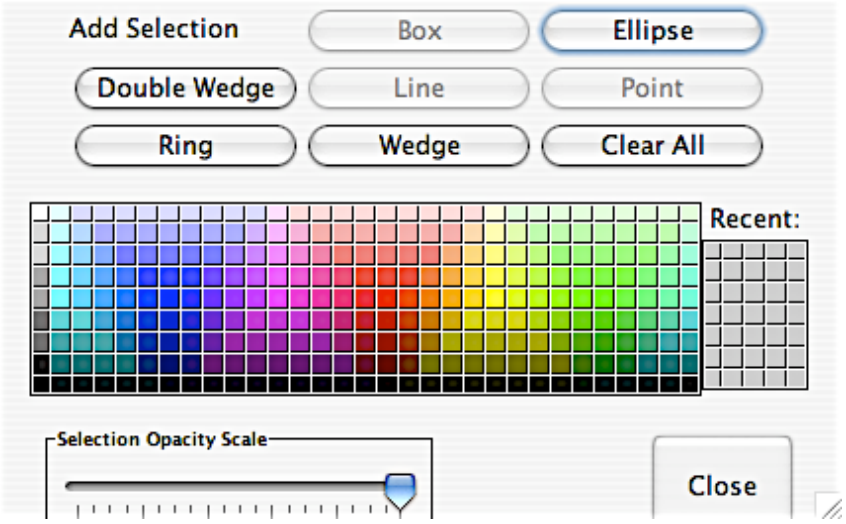

Figure 5-13, The Selection Overlay Edit window.

SAND Wedge Viewer File Menu

The SAND Wedge Viewer File Menu is used to load and save a variety of data from the SAND Wedge Viewer. The first option, **Open Project**, is used to open a previously saved session including all project-specific information. Similarly, the **Save Project** option is used to save all session-specific information, such as selections or annotations, to a user-named file with the extension .isv.

The **Print Image** option is used to print the currently displayed image in the Sand Wedge View. The **Make Image** option is used to to save the image that is currently displayed in the Sand Wedge View to a .jpeg file.

SAND Wedge Viewer Option Menu

The SAND Wedge View Option Menu allows for adjustments to the image and data display. When a selection is made in the SAND Wedge Viewer, a results window will appear. The **Hide Results** option is used to hide the results window, and conversely **Show Results** will cause the results window to reappear.

The next option, **Save Results to File**, is used to save the contents of the result window to a separate file. The new file will have three columns: Q, Intensity, and Error Bounds. Information about the region is listed at the top of the file prefixed by a # symbol. If multiple selections are made, only the last selection can be written to a file.

Note: All selections must be made before using the View Results or Save Results options.

The **Save User Settings** option is used to save your preferences automatically into the SandProps.isv files in your home directory. This option will not save project specific information such as selections or annotations. Use the File menu's **Save Project** to save project specific details.

The **Color Scale** option is similar to those found in other viewers, and allows you to choose a color scale to use in the SAND Wedge Viewer image view. The scale will be displayed on the right side of the graph with its corresponding intensities.

The **Preserve Image Aspect Ratio** option is used to ensure that the original aspect ratio of the image is preserved when making adjustments to the image or window. Deselecting this will fit the image view into the window to match its shape and size.

CHAPTER 6

Scripts

ISAW provides several ways for users to interact with their data. One of these ways is through the use of scripts. A script is a series of commands that is executed to perform a specific function. In ISAW, scripts can be used to perform virtually any task including wizard and viewer functions, and all scripts can be run in the command pane or in a wizard.

The Scripts Tab

To begin using scripts in ISAW, click the **Scripts** tab in the main ISAW pane. The Scripts tab consists of several buttons, a program editor, and an immediate editor. The **Program Editor** field is the place where most of your script commands will be entered. This field can be used to write scripts of any size, and it is also where loaded scripts will appear. Underneath the Program Editor is a pane called **Immediate**. This pane is used to write and immediately execute a script. This should generally be used only by experienced script writers. At the bottom of the window is a pane called **Status**. The Status pane is used to display error messages and outputs.

Near the top of the Program Editor pane is a series of buttons that can be used to interface with the Program Editor. The first button is called **Run Script**, and it is used to execute the script that appears in the Program Editor pane. Next to the Run Script button is the **Open Script** button. This button is used to open a file chooser window that can be used to open a previously saved script. The default directory for this file chooser window is the ISAW Scripts directory, but any directory can be specified by editing the Script Path option in ISAWprops.dat. The **Reset** button is used to clear the status line and the immediate pane. Next to the Reset button is a drop-down menu that contains two options. Select the **ISAW Script** option to write your scripts in the ISAW scripting language; otherwise choose the **Jython Script** option to write your scripts in the Jython language. The final button in the Scripts tab is the **Help** button. Press this button to display the help documentation associated with scripting in ISAW.

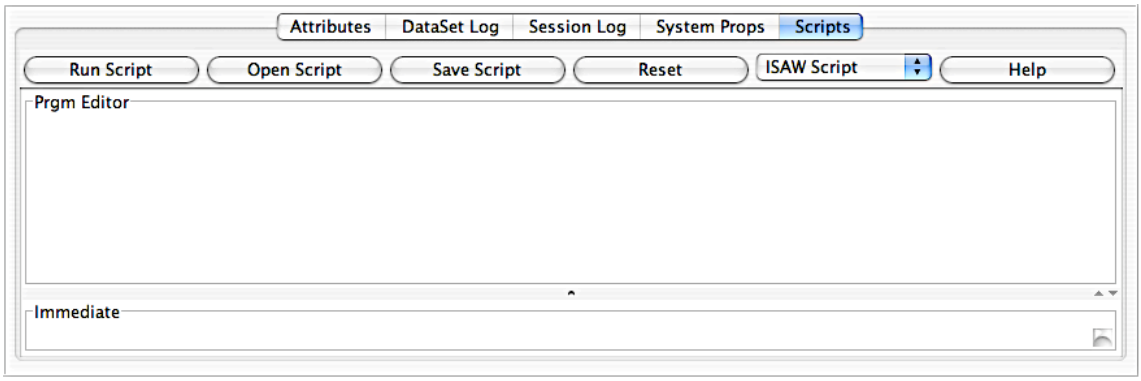

Figure 6-1, The scripts tab in the main ISAW window.

ISAW Script Data Types

ISAW uses a proprietary scripting language that is similar to Basic or Fortran. In the ISAW scripting language, variables can store one of five data types:

```
Integer
Float
```

```
String
Boolean
File
DataSet
Array
```

The first data assigned to a variable determines the data type of the variable. Integer constants have no decimal points while float constants must have a decimal point. Boolean constants are either `true` or `false`. DataSets do not have constants. Array constants are specified between square brackets ([ ]), and elements are separated by commas. Entries that specify a range of integers are separated by a colon (ie. 1:10) and they can be any Java object data type.

Other Data Types

ISAW operators can return any Object data type. These return values can be an element of an array or an argument in an argument list, but not as a variable.

The dimension of an Array can grow normally by one, but if an array has five elements and an instruction assigns a value to the seventh element, an error will result. It is also possible to concatenate two or more arrays to grow the array by using the & symbol.

Variable names are not case sensitive and they follow standard Fortran rules for variables. The leading character must be alphabetic, while the other characters are alphanumeric. The variable names cannot be reserved words like Load, Display, Save, Send, Return, for, end for, if, else, elseif, endif, on, end.

Type checking is incorporated in the ISAW scripting language so that if a variable has stored float data, that variable cannot be assigned an integer data value. Type conversion is not done on assignment statements; however, type conversions between numbers are done with the numeric, logical, and relational operations.

Numeric, Logical, and Relational Operations

There are several different numeric, logical, and relational operations available in the ISAW scripting language. The list below contains each of these operations and its corresponding symbol.

- Assignment operator: =
- Numeric Operations: +, -, *, /, ^
- Relational Operations: <, <=, >, >=, <>, = or ==
- String Operations: &
- DataSet Operations: +, -, *, / (with floats and other DataSets)
- Boolean Operations: And, or, not
- Array Operations: +, -, *, /, &
    - +, -, *, / with a number applies that operation and number to each element of the array.
    - +, -, *, / with another array applies that operation to corresponding elements of an array.
    - & concatenates two arrays.

These operations obey the standard conventions of precedence of operations.

Script Structure

Scripts in ISAW have a definite structure that should be followed to ensure that they are executed properly. The structures that are supported by the ISAW scripting language are: for, if, if – else, if – elseif, –else, and on error.

The **for** statement is commonly used to define a variable in a script. There is only one form for this statement that is supported and it is shown in the following example.

```
For I in [0,3,5:9]
     Display I
Endfor
```

In this example there are several minor variations that should be considered. First, the [0,3,5:9] can be an array variable. The values in this array can be strings, a float, an integer, etc. The variables should all be of the same data type otherwise type mismatch errors occur. Also, in this example the variable "I" is used, but any variable can be defined instead.

The **if**, **if -else**, **if -elseif**, and **-else** statements are all very common in scripting. Only the multiline form of these structures is allowed. The **then** is not required at the end of a line starting with the reserved words **if** or **elseif**. Nesting with any of the other structures are also supported. The following example illustrates the use of these statements in a script.

```
# This is a comment
If X > 0
   Display X
Endif

If x< 0
   Y = x + 1
Else
   Y = x - 1
Endif
```

```
if X>=0 and done then
    Display X
endif

if X>=5 then
    Y = X + 1
elseif X >=3
    Y = X
else
     Y = X - 1
Endif
```

The last set of statements are the **on error** structures. There are two structures for this statement as illustrated in the following example.

```
On Error - End Error
On Error - Else Error - End Error
```

If an error occurs in the On Error block the program will not crash but will transfer control to the block after the Else Error line or if there is no such line, control will be transferred to the statement after the line with the associated End Error instruction. Nesting is supported with these structures as well.

Intrinsic Operators

The ISAW scripting language contains five intrinsic operators including Load, Display, Save, Send, and Return. These are considered to be reserved words and are not case sensitive.

The **Load** instruction has several forms which are illustrated in the following examples.

```
Load ("c:\\Data\\xxx.run")
Load ("c:\\Data\\xxx.run", "Varname")
n = Load("C:\\Data\\xxx.run")
n = Load("c:\\\Data\\xxx.run", "Varname")
```

All of these examples create DataSets from the specified file. The cases that contain a second string argument allows the user to specify a variable name for the DataSets. In the above cases, the variable names will be Varname [0], Varname [1], etc. If the second string argument is not specified, then they will be stored with default variable names. The default variable names are those that appear in ISAW 's DataSet tree when the file is loaded (ie. M1_xxx, H1_xxx, etc.). Files that are not run files can also be loaded with the Load instruction, including Java Binary files (.isd).

The **Display** command has two forms as illustrated by the following examples.

```
Display  expression
Display DataSet_Expression , "SCROLLED_GRAPH"
```

The first line in the example displays the value of the expression in the Status pane if possible. If the result is a DataSet, then a viewer will appear to display the DataSet. The second line in the example can be used to display a viewer. The second argument in this example gives the type of viewer that will appear, and can only be "IMAGE", "SCROLLED_GRAPH", "SELECTED_GRAPH", "TABLE", "THREE_D", or any view menu entry.

The **Send** command has only one form as illustrated by the following example.

```
Send DataSetExpression
```

This is used to send the DataSet to any program, such as ISAW, that is listening for the DataSet. ISAW will add the resultant DataSet to its tree.

The **Return** command is used in instances when scripts can become operators that may want to return values. The value returned is the value of the last expression that was evaluated. Most of the time this will be null. To return a non-null value without a Return instruction, smiply place the variable or expression on the last line of the script. This can be seen in the following example.

```
X = 3
Y = 2
Display X+Y
#The following will be returned
X-Y
```

Other Operators

All other commands that are translated by the Command Pane's ScriptProcessor are packaged as operators. Some operators are installed before ISAW or the Command Pane is started. Others can be installed by specifying their location via the Script_Path variable in the IsawProps.dat file. These will be added as either system loads.

Script Operators

Any script without errors can become a script operator. The script can then be run from the Command Pane by loading the instructions, and then pressing the **Run Prgm** button.

The command used to invoke this operator as an instruction in another script is its file-name without its path or extension. Extra instructions in scripts can be added to specify the parameters and prompts for these operators.

```
$ Title = This will appear in dialog boxes and menus
# varname    Data Type        Prompt
$ x               Integer               x=
$ y               String("abcd")        y=
```

The data types that can occur in the previous example are any of the supported data types that are listed in the Data Types section of this chapter. In addition to these, several other data types can be used, and the list is as follows.

```
DSFieldString
InstrumentNameString
DataDirectoryString
DSSettableFieldString
```

Parentheses are used after the data type specifier to enclose a suggested initial value for this variable when users are allowed to specify the parameters via the dialog box.

Writing Java operators

There is a slight difference between DataSetOperators and GenericOperators. In DataSet Operators the first DataSet is not a parameter, so addDataSet must be used instead of add-Parameter for that argument. This difference is transparent when writing scripts using these operators. Furthermore, these operators can implement Iobservable and/or Customizer if the required add methods would be needed. The system automatically adds any available listeners of these types to your operator.

Input-Output Considerations

Functions and Subroutines need inputs in order to produce outputs. The input values must be obtained through the use of parameters; however, getting the results back to the calling program requires some additional considerations. Output parameters can only have the data types of DataSet and Array (in Java this is called a vector). Other unsupported data types may also be used for sending results back in java-class operators if the code is written carefully.

Considerations for other cases

If an operator has several results that must be sent back to the calling program, these results could be packaged into an Array.

Interface to ISAW

The Command Pane was specifically designed to be integrated with ISAW. Most menu choices in ISAW translate to commands in the Command Pane.

ISAW to Command Pane

All DataSets in ISAW's DataSet tree can be accessed by the scripts in the Command Pane according to the following convention:

If the tree DataSet is 25:M1_hrcs2204 , then the script must refer to it as ISAWDS25.

Again, as mentioned in the Load instruction, the default name given to the variable storing a DataSet after the Load instruction would be M1_hrcs2204 because that is how the DataSet appears in ISAW's DataSet tree.

ISAW to ScriptProcessor

There is a submenu option called **Load Scripts** under the ISAW's File menu. This will execute the script selected by the file chooser without entering the script into the script editor window.

Command Pane to ISAW

The Send instruction will send a DataSet to ISAW if ISAW is running. If the Command Pane is run by itself, then the send command will not work.

Parameter GUI's

In the ISAW scripting language, it is possible to define GUI elements for your scripts. By utilizing the following Parameter GUI's, it is possible to construct a fully functional graphical user interface in your scripts.

- Array: produces a text field that allows users to enter values into an array. The result is a vector.
- Boolean: is a class that produces the checkbox GUI element for specifying true or false. The result is a boolean.
- ChoiceList: is a class that produces the drop-down menu GUI element containing a list of values to choose from. The return value can be either a string or a numerical value depending on the input list. For example, if a list of strings is input then the return value will be a string.
- DataDir: produces a directory chooser window that is used for selecting a directory. The result is a string.
- DataSet: produces a drop-down menu containing a list of DataSets that are currently in the tree. The result is a DataSet.
- FloatArrayArray: is used to create a medium-sized, two-dimensional float array. The result is a vector of vectors of float values.
- FloatArray: is used to create a medium-sized, one-dimensional float array. The result is a vector of float values.
- Float: produces a text field that is used to enter data containing a floating point value. The result is a float.
- FuncString: produces a text field that is used to enter a string that represents a mathematical function. The result is a string.
- InstName: produces a text field that is used to enter instrument names. If the initial value is left blank then the default instrument will be used. The default instrument can be specified in ISAWprops.dat under "Viewer Options" by editing the value for Default_Instrument. The result is a string.
- IntArray: produces a text field that is used to enter an increasing sequence of consecutive integers. Its value is stored as a formatted string. Individual values are separated by a comma and a set of consecutive values can be represented by inserting a colon between the start and end values. The result is a string.
- IntegerArray: produces a list box with editing buttons. It is used to create medium-sized, one-dimensional integer arrays. The result is a vector of integers.

- Integer: produces a text field that is used to enter integers. The result is an integer.
- LoadFileArray: produces file chooser windows that allow the user to choose a list of files from multiple directories. The result is a vector containing file names.
- LoadFile: produces a file chooser window that allows the user to choose a single file. The value is a string.
- Material: produces a text field that is used to enter material names. Chemical formulas must be typed in the correct format using commas to separate elements and underscores to indicate the number of atoms. The result is a string.
- MonitorDataSet: produces a drop-down menu that contains a list of available monitor data sets. The result is a data set.
- PlaceHolder: saves an opaque object value which cannot be altered by the GUI. Parameters can be passed between operators but cannot be modified by the scripting language. Since the data type is determined by the first value assigned to a variable, a variable has this opaque data type if an operator returns an object that does not match one of the specific data types.
- PrinterName: produces a drop-down menu that contains a list of printer names available to the system. The result is a string.
- PulseHeightDataSet: produces a drop-down menu that contains a list of available pulse height data sets. The result is a data set.
- Qbins1: produces a window with text fields that allow the user to enter start, end, and number Q bins for a sublist. Constant dQ or dQ/Q choices are also supported. The result is a vector of floats.
- Qbins: produces a list box with editing buttons that allows the user to enter a large list of Qbin boundaries with either constant Q widths or constant ratios. This Parameter GUI allows for concatenating of several of these lists. The result is a vector of floats.
- RadioButton: is a class that produces radio button GUI elements. The result is a string.
- RealArray: produces a text field that allows for entry of small lists. The input can be multi/uni dimensional arrays of integers, floats, doubles, or strings. The result is a multi-dimensional array that depends on the initial value.
- SampleDataSet: produces a drop-down menu that contains a list of available sample data sets.
- SaveFile: produces a file chooser window that allows the user to choose a single file in a directory. The value is a string.
- StringArray: produces a list box with editing buttons that allows users to enter one-dimensional string arrays. The result is a vector of strings.
- String: produces a text field that is used to enter a string. The result is a string.
- UniformXScale: produces a command pane that allows the user to specify a start value, an end value, and the number of X values for a uniform X scale. The result is a vector of float values.
- VariableXScale: produces a list box with editing buttons that allows the user to enter arbitrary X scales. The return value is a vector of floats.

The Parameter GUI's can be implemented in a number of different ways, but the basic format for defining a Parameter GUI can be seen by the following example.

```
$x Array(2.00,5.00,10.0) Enter Values
```

```
     Display x
```

In this very simple example, the ArrayPG is used to specify a text field that contains an array having values of 2, 5, and 10. Following the array is the area where the title for the text field should be specified; in this case it is Enter Values. The `Display x` command is used to print the array to the Status pane in the Scripts tab.

There are also Paramter GUI's that have very specific formats for their values. In general, multiple values should be separated by commas, ranges of values should be separated by colons, strings should be encapsulated by quotation marks, and vectors should be encapsulated by square brackets. Float PG's should only specify non-whole numbers, and integer PG's should only specify whole numbers.

For Parameter GUI's that specify a chemical formula, such as MaterialPG, it is necessary to arrange your formulas in a very specific way. Individual elements should be separated by commas, and element subscripts can be delineated with an underscore followed by the number of atoms. For example, trifluoromethanesulfonic acid (CHF_3O_3S) can be specified by typing the following.

```
$x Material(C,H,F_3,O_3,S) Enter Material
     Display x
```

Parameters such as "Qbins1" and "UniformXScale" also have special requirements for input of their values. In "Qbins1", the values include start, end, and number of Q bins for a sublist. This is entered using the following format.

```
([Start Value, End Value, Number of Steps, Constant dQ or dQ/Q])
```

Constant dQ or dQ/Q is a boolean value and is specified by either true or false. True indicates constant dQ, and false indicates constant dQ/Q.

The UniformXScale has a similar format for listing a start value, an end value, and the number of X values for a uniform X scale. These values are entered using the following format.

```
([Start Value, End Value, Number of X Values])
```

Building a Script

Using the information and techniques listed in this chapter, it is possible to construct a script capable of performing several complex tasks within ISAW. The following example, written by John Hammonds, illustrates how a script can be used in ISAW to create and run the Daily Peaks Wizard.

```
#Specify the command name and category list where the operator appears
$Command = CommandName
$Category = Macros, Examples, Scripts (ISAW)
$filename  LoadFileString    Enter default file

#Set up the Script directory
isawHome=getSysProp ("ISAW_HOME")
formHome=isawHome&"/Wizard/TOF_SCD/Scripts_new/"
#Assign the forms to use
form1=formHome&"find_multiple_peaks1.iss"
```

```
form2=formHome&"JIndxSave1.iss"
form3=formHome&"LSqrs.iss"
form4=formHome&"JIndxSave2.iss"
form5=formHome&"integrate_multiple_runs.iss"

#Create an instance of a Wizard object
a = createWizard("DailyPeaksWizardScriptVers",false)

#Add forms to the Wizard
addScriptForm(a, form1, "PlaceHolder", "Peaks", "", [])
addScriptForm(a, form2, "Array", "Result1", "", [0])
addScriptForm(a, form3, "Array", "Result", "", [O,1,2,4])
addScriptForm(a, form4, "PlaceHolder", "Peaks", "", [0,3,4,8])
addScriptForm(a, form5, "String", "Result", "", [0,1,2,3,4,5,8,9])
#Create links between forms
#LINK FOR PATH DATA
links[0]= [0,-1,-1,-1,0]
#Link for output path
links[1]= [1,-1,4,3,1]
#Link for Runnums
links[2] = [2,-1,2,4,2]
#Link for exp name
links[3] = [3,-1,1,8,3]
#Link for peaks
links[4] = [12,0,0,0,-1]
#Link for instName
links[5] = [10,-1,-1,-1,8]
#Link for FileExt
links[6] = [11,-1,-1,-1,9]
#Link for calibFilename
links[7] = [8,-1,-1,-1,5]
#Link for RestrRuns
links[8] = [-1,2,-1,5,-1]
#Link for Filename to save peaks to
links[9] = [-1,4,-1,-1,-1]

#Add links to this Wizard
wizardLinkParameters(a,links)

#Load the Wizard
wizardLoader(a)
```

As you can see from the above example, it is possible to create a wizard from a script instead of Java code. This was done in only five steps. First it is necessary to create a wizard object. Second, you must add forms and then put wrappers around the forms. The third step is to link the parameters. The fourth step is to make a new class containing Public Static Methods; in the above example this is WizardMethods.java. Finally it is necessary to write the wizard inside the script. These steps illustrate how implementing the concepts in this chapter will allow you to create powerful scripts, wizards, and operators.

CHAPTER 7

Operator Generator

The Operator Generator is a tool in ISAW that is used to quickly and easily create custom operators from existing static methods. The resulting operators can be invoked either through the Macros menu or a script command.

The Operator Generator

To run the Operator Generator Wizard, simply type the following into your command line.

```
java devTools.Method2OperatorWizard
```

When the Operator Generator opens you will see a window containing four tabs.

Information Tab

The Information Tab is used to enter data about the operator's creator. This data will automatically be included as part of the documentation for the operator's code. Information entered in this tab can also be stored in a separate file for use in other operators by choosing **Save Contact Info** under the **File** menu.

Method Info Tab

The Method Info tab is where the static methods that will be used in your operator are chosen and defined. The first option is called **Select Class with static Method** and is used to find the Java class that contains the methods that you would like to use. To view the Java code for the selected class in a separate window, press the **View File** button. The **Select Method** drop-down menu is used to choose a method to work with in the **Set GUI Info for Each Parameter** section. The **Arguments** box lists all of the available parameters fro the currently selected method. To edit the GUI info, select a parameter under **Arguments**, then enter appropriate information into the text fields following **Prompt**, **Param Name**, and **Init Value**. The **Param GUI** button is used to view information about the currently selected Parameter GUI. The **Param GUI Menu** lists all the available Parameter GUI's that represent the selected argument.

Operator Info Tab

The Operator Info tab is used to define a location for the new operator, and write descriptions for the new operator. The **Select Operator Output File** is used to select a file to save the completed operator to. If the file does not exist then it will be written to the specified location; however, doing this will not save the current session. The next three options are used to edit the operator's properties. Use the **Operator Title** text box to enter a title for the operator; this title will be used as the operator's name in all menus. The **Command for Scripts** text field is used to define how the operator is called from scripts. The **Category** text field is used to specify the menu location of the operator. Operators will be placed under the Macros menu. Any additional sub-menus that are specified will be created if they do not already exist.

Documentation Tab

The Documentation Tab is used to enter all of the relevant information about your operator that you would like to provide to users of your operator. Each text box provides ample room to type your comments and will appear when a user loads the operator in ISAW. The **Overview** text box should be used to enter a brief description of your operator. The **Assumption** text box should be used to address any assumptions within the operator

including special DataSet requirements, positive/negative values, and units of measurement. The **Algorithm** text box should be used to enter relevant information about the mathematical processes used in the operator. The **Return** text box is used to enter information about the operator's return values. It is important to include information about the return type and units of measurement. The **Parameter** text box should be used to describe the operator's variables and stored values. The **Error** text box should be used to describe any known errors that exist within the operator. Exceptions will be translated into error messages as specified by this field.

Programmer Notes

Parameter GUI elements are created for each method argument that is used. Each static method must return an object value. For each static method, objects include one and two-dimensional arrays, strings and error strings. Objects include the primitive data types; however, "Integer" cannot be used instead of "int". This rule applies for other parameters as well. Error strings and exceptions can be reported by objects.

Operator Generator File Menu

The **File** menu contains several options for storing operator information, saving your Operator Generator session, and creating your custom operator. The **Load Contact Info** option is used to load your saved contact information into the **Information Tab** in the Operator Generator. The **Load Session** option is used to load your saved session information into all of the tabs in the Operator Generator. Use the **Create Operator** option to create the currently defined operator and save it to the file chosen in **Select Operator Output File** under the Info tab. Please note that using the Create Operator option will not save the current session; to save your session, select the **Save Session** option under the File menu. The **Save Contact Info** option is used to save the data contained in the Info tab as a separate file for use in other operators. The **Exit** option can be used to exit the Operator Generator.

Building DataSets

A DataSet object, as used by ISAW, is a container object that contains zero or more Data objects. Each Data object represents a tabulated function or histogram using a collection of "y" values and corresponding "x" values. Both the containing DataSet and each Data object that it contains also hold several types of auxiliary information. Some of the auxiliary information is in the form of a fixed set of data fields in the objects. Some of it is in an extensible list of "attributes" maintained by each object. Various operations can be performed on a DataSet and the DataSet includes an extensible list of operators that can operate on the DataSet. Finally, the DataSet keeps a "log" of the operations that have been applied to the DataSet.

Steps in Building a DataSet

1. Construct the empty DataSet complete with appropriate operators.
2. Add "attributes" to the DataSet.
3. Construct a Data block to add to the DataSet.
4. Add "attributes" to the Data block.
5. Add the Data block to the DataSet.

Creating a DataSet

Construct the Empty DataSet

This is very easy for a time-of-flight DataSet. There is a DataSetFactory that can be used to build the empty DataSet and add the needed operators to the DataSet. For a time-of-flight DataSet this is used as shown below. The "title" parameter that is passed to the constructor of the DataSetFactory will specify what title will be used for subsequent DataSets produced by the factory.

```
DataSetFactory ds_factory = new DataSetFactory( title );

DataSet   ds    = ds_factory.getTofDataSet( instrument_type );
```

The instrument_type is an integer code for the type of instrument. This is used by the DataSetFactory to determine which operators should be added. The values for the integer codes are defined in the file:

```
.../DataSetTools/instruments/InstrumentType.java
```

and currently include:

```
InstrumentType.TOF_DIFFRACTOMETER

   InstrumentType.TOF_SCD

   InstrumentType.TOF_SAD

   InstrumentType.TOF_DG_SPECTROMETER
```

```
InstrumentType.TOF_IDG_SPECTROMETER

InstrumentType.TOF_REFLECTROMETER
```

At this time DataSetFactory provides a larger set of operations for the types `TOF_DIFFRACTOMETER` and `TOF_DG_SPECTROMETER`. Support, by way of special operators for the other instrument types is still being developed. In all cases, very basic operations such as add, subtract, multiply and divide by DataSets and scalars are included. If you are constructing a "generic" DataSet with axis labels and units other than those for a time-of-flight instrument, you can use a different constructor for the DataSetFactory such as:

```
DataSetFactory factory = new DataSetFactory( title,
                                             "Angstroms",
                                             "d-Spacing",
                                             "Counts",
                                             "Scattering Intensity" );

DataSet ds = factory.getDataSet();
```

In this case, the factory will produce DataSets with the given title, axis units and labels. The method "getDataSet" will only add the generic operators to the DataSet, not the operators specific to time-of-flight DataSets.

Add Attributes to the DataSet

Attributes can be added to DataSets and Data blocks at any time. It's probably best to add the attributes you'll need in an organized manner at the time that you are constructing the DataSet. Attributes are name, value pairs where the value can be things like an integer, float, array of integers, character string, etc. Attributes are classes that are derived from the abstract base class defined `in .../DataSetTools/dataset/Attribute.java`. This file also contains a list of the names that we have been using for the attributes. The names are given by "constant" strings. Since each attribute is stored in it's own object, it is usually necessary to create the attribute objects as they are added to the DataSet (or Data block). If a few individual attributes are being added to the DataSet (or Data block) they can be added using the setAttribute(attribute) method. For example, assuming that the original data file name is to be stored as an attribute of the DataSet, you could write:

```
    ds.setAttribute( new StringAttribute( Attribute.FILE_NAME,
file_name ) );
```

to set an attribute for the file name in DataSet "ds". This assumes that the variable `file_name` is a string containing the file name. This will construct a new StringAttribute object with the name of the attribute given by the constant `Attribute.FILE_NAME = "File"` and the value of the attribute given by the file_name string. Other attributes are treated similarly.

```
    ds.setAttribute( new IntAttribute(Attribute.NUMBER_OF_PULSES,
num_pulses));
```

The attribute can also be constructed separately and then set in the DataSet like:

```
    int_attr = new IntAttribute( Attribute.NUMBER_OF_PULSES, num_pulses
);

    ds.setAttribute( int_attr );
```

Finally, there are also routines to get and set the entire list of attributes at once, but the routines to get and set individual attributes are actually more efficient to use in most cases.

Construct a Data Object

The three most crucial pieces of information held in each Data object are the list of y-values, an XScale object specifying the corresponding x-values and a unique integer ID. These three pieces of information are needed by the constructor for a Data object. The y-values are just an array of type float[] and the ID is just an integer value. However, the XScale is an object that either contains the x-values, or contains enough information to calculate uniformly spaced x-values. The x-values are stored in an XScale object for space efficiency. That is, in many cases the x-values associated with a Data block are evenly spaced. In that case, they can be easily calculated as needed based on the first point, the last point and the number of points. Since we may have thousands of spectra with thousands of y-values in each, it would be a serious waste of space to store corresponding evenly spaced x-values in such cases. In this case, the x-values can be stored in a UniformXScale object, derived from an XScale object. For example, a uniform XScale object with 101 points evenly spaced on the interval [0,10] can be constructed as:

```
    XScale x_scale = new UniformXScale( 0, 10, 101 );
```

If the x-values are not evenly spaced, a VariableXScale object can be used to explicitly store all of the x-values. Specifically, if an array of floats named "x-vals" contained the x-values we could create a VariableXScale object as follows:

```
   XScale x_scale = new VariableXScale( x-vals );
```

In either case, software using the x_scale can get at information such as the min, max, number of points and the actual x-values using the methods of the base class XScale. For a time-of-flight Data object, the operators assume that the times are specified in microseconds. It also should be noted, that Data objects are used to store either histogram data, or tabulated function data. These two cases are distinguished based on the relationship between the number of x-values and the number of y-values. Specifically, for a tabulated function Data object, the number of x-values will be the same as the number of y-values. In this case, the y-values give the value of a function at the corresponding x-value. On the other hand, for a histogram, the Data object records the x-values at the boundaries of the histogram bins. The y-values are considered to be the y-values at the bin centers. Thus for histogram data, the number of x-values is one more than the number of y-values. The number x-values is restricted by the Data object constructor to be either the number of y-values, or the number of y-values plus one.

An example of building a simple Data block for the function `y = (x/10)^2` on the interval `[0,49]`, with ID = 1 is given below:

```
  float  y_values[] = new float[50];
```

```
XScale x_scale    = new UniformXScale( 0, 49, 50 );

for ( int i = 0; i < 50; i++ )

  y_values[i] = (i/10.0) * (i/10.0)


Data data = Data.getInstance( x_scale, y_values, 1 );
```

Add Attributes to a Data Object

Both DataSet objects and the Data objects that they contain implement the IAttributeList interface. As a result, attributes are added to Data objects in exactly the same way as they are added to DataSets. For example, if data is a Data object, we could add an attribute specifying that the initial energy was 120.0 as follows:

```
   data.setAttribute( new FloatAttribute( Attribute.ENERGY_IN, 120.0f )
);
```

Add the Data Object to the DataSet

Once a Data object has been constructed, and its attributes set, it should be added to a DataSet. For example, to add a Data object "data" to a DataSet "ds" just do:

```
   ds.addData_entry( data );
```

Attributes Required for a DataSet and Data Object

Although the above discussion describes how to construct a Data block and DataSet, more information is needed to construct a DataSet to hold time-of-flight data in a way that will allow useful operations to be done on the Data. In particular, most of the "interesting" operators for neutron scattering rely on specific attributes of the DataSet and Data objects. The attributes that are currently used by various operators include:

```
Attribute.DETECTOR_POS         <- object

Attribute.INITIAL_PATH         <- float

Attribute.ENERGY_IN            <- float


Attribute.NUMBER_OF_PULSES     <- int

Attribute.SOLID_ANGLE          <- float


Attribute.DELTA_2THETA         <- float

Attribute.RAW_ANGLE            <- float
```

To allow for comparing and scaling DataSets, some measure of the number of neutrons that hit the sample is needed. For use in scripts, this should probably be the number of pulses, at least that is what has been used for GPPD. If the number of pulses is not directly available, it could possibly be approximated based on a start time and end time. At

any rate, it would be useful to have the number of pulses stored as a DataSet attribute for any instrument.

The attributes listed above are primarily used as attributes of each Data object. The attributes are listed in decreasing order of importance. Interpreting the time-of-flight data almost always requires the effective detector position. The convention used in the DataSetTools package is that the effective detector position gives the position of the detector relative to the center of the sample. Since there are different ways of specifying this position, a class was constructed to hold the position information and provide some extra information as needed. A "Position3D" object contains a 3D position, specified in any of the usual coordinate systems, Cartesian, cylindrical or spherical. There are methods to get and set the position in any of these coordinate systems, as well as some additional convenience routines.

The convention for the instruments at IPNS is that the coordinate system has its origin at the sample position, the x-axis points in the direction the incident beam is traveling, the y-axis is horizontal, perpendicular to the incident beam and z-axis is perpendicular to the earth's surface. This is also the convention followed by the DataSetTools package. Unfortunately, that coordinate system is somewhat inconvenient for describing the scattering angle (the angle between the positive x-axis and the vector from the sample to the detector). Since the operators frequently need to use the scattering angle a class "DetectorPosition" was derived from the Position3D class. The DetectorPosition class adds a method to get the scattering angle, and so it should be used to represent the position of the detector relative to the sample. An example of code to set a detector position attribute corresponding to a detector that is at an angle of 50 degrees, 0.1 meter above the xy plane, and above a horizontal circle of radius 4.0 meters centered at the sample is shown below:

```
DetectorPosition position = new DetectorPosition();

float angle      = 50.0f * (float)(Math.PI / 180.0);

float final_path = 4.0f;

float height     = 0.1f;

position.setCylindricalCoords( final_path, angle, height );

data.setAttribute( new DetPosAttribute( Attribute.DETECTOR_POS,

                                        position ) );
```

Lengths are assumed to be in meters, and angles are stored in radians. If the detector position is easier to specify in Cartesian coordinates, (x, y, z), the method position.setCartesianCoords(x, y, z) can be used instead.

The initial path attribute is needed for the diffractometer instruments. The initial flight path is the source to sample distance in meters. If this can be obtained, as say the float variable "length", it is easily added to the Data block as:

```
    data.setAttribute( new FloatAttribute( Attribute.INITIAL_PATH,
length ) );
```

The operators to process data from direct geometry spectrometers require the initial energy of the neutrons incident on the sample. The initial energy is assumed to be in meV. It is often necessary to calibrate this value, but at least some initial approximation will be needed by these operators. The more advanced operators for direct geometry spectrometers will require the number of pulses to be stored with each Data block, in addition to being stored with the DataSet as a whole. Finally, these operators need the solid angle subtended by the detector group, measured in steradians.

The operator to produce a display of S(Q,E) for spectrometers will need an approximate value for the interval of scattering angles covered by each detector. That is, each detector has non-zero dimensions. Consequently, even though the detector might be nominally at say 50 degrees, it actually covers some interval, say 49.95 to 50.05 degrees. Some approximation to the range of angles covered should be stored in a `DELTA_2THETA` attribute. This value is assumed to be stored in degrees.

The operator to produce a "TrueAngle" display of a DataSet requires the `DELTA_2THETA` attribute, as well as the `RAW_ANGLE`. The `RAW_ANGLE` is the actual physical scattering angle for the detector, without regard to time-focusing. (Time focusing may adjust the raw angle to a different effective angle.) A `DETECTOR_POS` attribute is assumed to hold the effective 3D position of the detector, while a `RAW_ANGLE` attribute is assumed to hold the physical, unfocused scattering angle.

Data Retriever

In order to easily work with different sources of data, such as IPNS runfiles, Nexus files, data acquisition hardware, etc. the system was designed to access data through subclasses of the abstract class `.../DataSetTools/retriever/Retriever.java`. Currently the only derived class is a RunfileRetriever that accesses IPNS runfiles. New data sources should be supported by making a new class derived from the Retriever class, since in that way, all data sources can be used in the same way. The Retriever class is quite simple:

The constructor accepts a string giving the name of the data source. For a data file, this would most likely be the file name, and the file would most likely be opened in the constructor. The Retriever class then provides three methods, a method to get the number of DataSets available from the source, a method to get the type of each available DataSet (`MONITOR_DATA_SET` or `HISTOGRAM_DATA_SET`) and a method to get a specific DataSet from the source. For the special case of the IPNS runfile retriever this gets used as simply as:

```
RunfileRetriever rr    = new RunfileRetriever( "gppd9898.run" );

DataSet A_monitor_ds   = rr.getDataSet( 0 );

DataSet A_histogram_ds = rr.getDataSet( 1 );
```

where we've used the simplifying assumption that the "zeroth" DataSet is always the monitor DataSet and the "first" always the first histogram DataSet. These simplifying assump-

tions make it unnecessary to find out the number of DataSets and find out their types before reading.

As other types of files or data sources are supported, the Retriever class may need to expand slightly. However it is best to keep this class as simple as possible, since any new functionality introduced in the Retriever will have to be supported by ALL types of retrievers.

Example

A simple program to demonstrate building a DataSet is in the file:

```
.../DataSetTools/trial/BuildDataSetDemo.java
```

in the latest version of DataSetTools. It can be compiled from within the directory containing it using:

```
javac  BuildDataSetDemo.java
```

and then can be run using

```
java BuildDataSetDemo
```

Assuming that all `PATH` and `CLASSPATH` values have been set properly. The code for the demo is listed below:

```
/*

 *  @(#)  BuildDataSetDemo.java    1.0  2000/9/19    Dennis Mikkelson

 *

 */

import DataSetTools.dataset.*;

import DataSetTools.viewer.*;

import DataSetTools.math.*;

/**

  *  This class provides a basic demo of how to construct a DataSet.

  */

public class BuildDataSetDemo

{

  /**

   *  This method builds a simple DataSet with a collection of 10 sine
waves.
```

```
   *
   *  @return  A sample DataSet with 10 sine waves.
   */
  public DataSet BuildDataSet()
  {
   //
   // 1. Use a "factory" to construct a DataSet with operators --------
   //
    DataSetFactory factory = new DataSetFactory( "Collection of Sine
Waves",
                                                 "time",
                                                 "milli-seconds",
                                                 "signal level",
                                                 "volts" );
    DataSet new_ds = factory.getDataSet();
    //
    // 2. Add attributes, as needed to the DataSet --------------------
    //
    new_ds.setAttribute( new StringAttribute( Attribute.FILE_NAME,
                                              "BuildDataSetDemo.java" )
);
    new_ds.setAttribute( new IntAttribute(Attribute.NUMBER_OF_PULSES,
10000) );
    //
    // Now, repeatedly construct and add Data blocks to the DataSet
    //
    Data          data;    // data block that will hold info on one
signal
    float[]       y_values;// array to hold the y-values for that sig-
nal
    XScale        x_scale; // "time channels" for the signal
```

```
    for ( int id = 1; id < 10; id++ )            // for each id
    {
      //
      // 3. Construct a Data object
      //
      x_scale = new UniformXScale( 1, 5, 50 );  // build list of time
channels
      y_values = new float[50];                 // build list of counts
      for ( int channel = 0; channel < 50; channel++ )
        y_values[ channel ] = 100*(float)Math.sin( id * channel / 10.0
);
      data = new Data( x_scale, y_values, id );


      //
      // 4. Add attributes as needed to the Data block
      //
                                          // "simple" energy in attribute
      data.setAttribute( new FloatAttribute( Attribute.ENERGY_IN,
120.0f ) );
                                          // more complicated, position
                                          // attribute has a position
                                          // object as it's value
      DetectorPosition position = new DetectorPosition();
      float angle      = 50.0f * (float)(Math.PI / 180.0);
      float final_path = 4.0f;
      float height     = 0.1f;
      position.setCylindricalCoords( final_path, angle, height );
      data.setAttribute( new DetPosAttribute( Attribute.DETECTOR_POS,
                                              position ) );
      //
```

```
      // 5. Add the Data object to the DataSet
      //
      new_ds.addData_entry( data );
    }
    return new_ds;
  }
  /*
-------------------------------------------------------------------
*/
  /**
   *  The main program method for this object
   */
  public static void main(String args[])
  {
    BuildDataSetDemo demo_prog = new BuildDataSetDemo();// create the
class
    DataSet test_ds = demo_prog.BuildDataSet();      // call the method
to
                                                     // construct a Da-
taSet
                                                     // create a viewer
for
                                                     // the DataSet
    ViewManager view_manager = new ViewManager( test_ds,
IViewManager.IMAGE );
  }
}
```