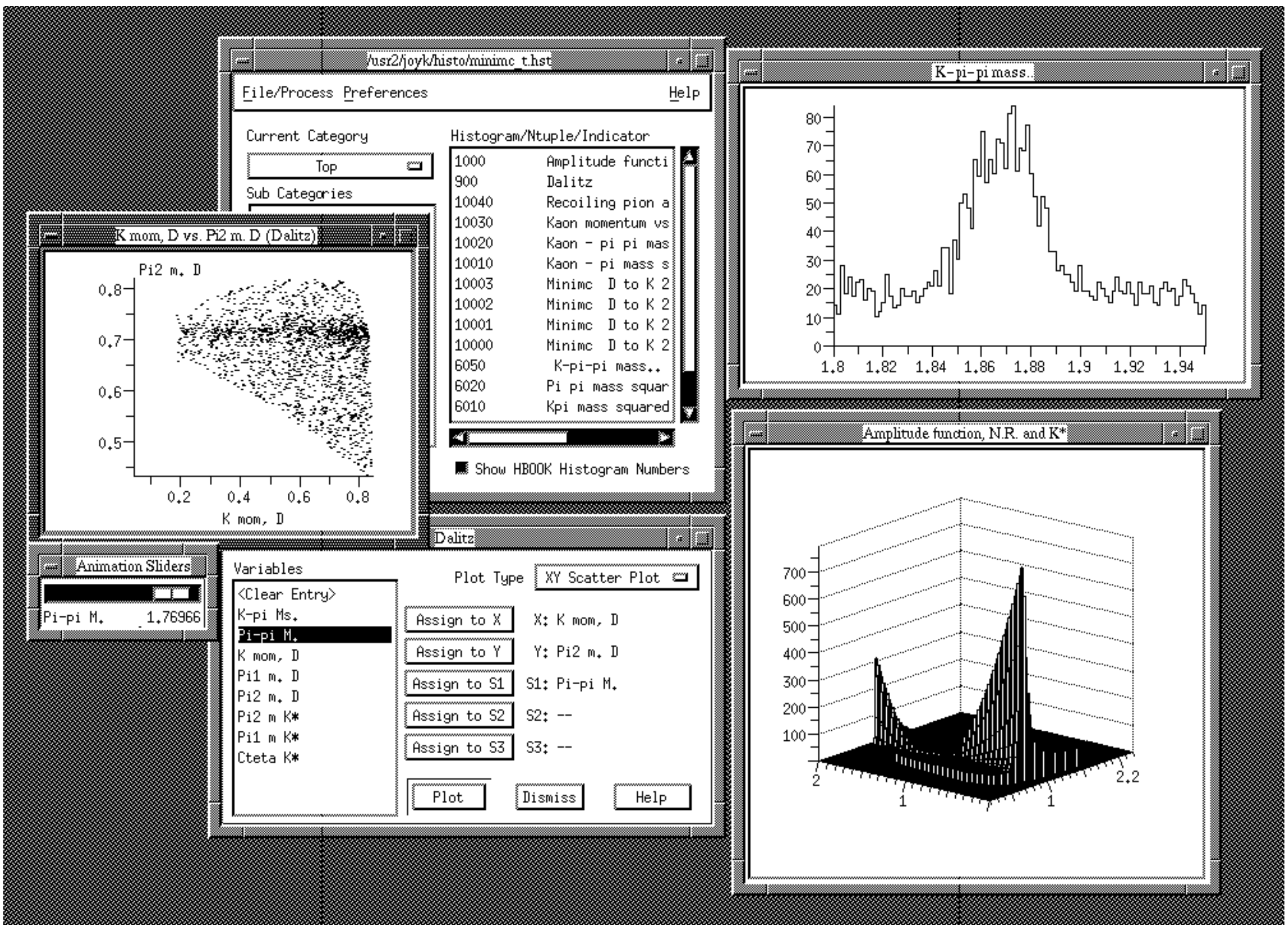
SP0034

Histo-Scope

V4.0

USER'S GUIDE



Fermilab

Physics Analysis Tools Group

Histo-Scope

Software Version:	V4.0
Date:	1/18/97
Host System Support:	DEC/ULTRIX DEC/OSF-1 SGI/IRIX Sun/Solaris IBM/AIX
Graphics Support:	X-based workstation or X-Terminal
Optional Software:	HBOOK Version 4.21 (or later)
WWW Page:	http://www-pat.fnal.gov/histo_doc/histo_ug.html

This software was produced by the Nirvana project of the Physics Analysis Tools Group. The Nirvana project goal is to provide visualization tools for Fermilab physicists that make effective use of computer workstations and Graphical User Interfaces. Histo-Scope is a pilot product for a suite of easy to use, Motif based tools for physics analysis.

Histo-Scope was written by: Mark Edel, project leader
Konstantine Iourha
Joy Kyriakopulos
Jeff Kallenbach
Paul Lebrun
Baolin Ren

User's Guide Written by Joy Kyriakopulos and Donna Reid

Title Figure: Clockwise from top right:

1D Histogram representing a Dalitz plot of $D^+ \to k^+\pi^-\pi^-$ mass
2D Histogram representing a Dalitz plot of the $D^+ \to k^-\pi^+\pi^+$ decay
Ntuple Window for Dalitz Ntuple data, used to generate scatter plot
Animation Slider window for Scatter Plot
Scatter plot representing a Dalitz plot of $\pi^-\pi^-$ invariant mass vs. kaon momentum in D rest frame.
Main Window, listing all data items in the file.

Table of Contents

Table of Contents (con't)

Table of Contents (con't)

Introduction

Histo-Scope is a tool to select and display histograms, Ntuples, and scalar values from a program as data is being created or analyzed. Using Histo-Scope, physicists can interactively "browse" through the large quantities of statistical data that their analysis and data acquisition programs gather as they run. It is intended to complement existing physics applications, and provides new interactive methods for viewing data with immediate access to data while a program is running.

Histo-Scope requires an X-based workstation (DEC/ULTRIX, DEC/OSF-1, SGI, Sun, or IBM/Risc) **or** an X Terminal or X-based workstation that can connect to one of these hosts. In addition, if HBOOK histograms or Ntuples are to be viewed, the HBOOK software is required on the host that creates the data.

Histo-Scope has two parts. One part is a library of routines which can be inserted in the physics analysis or data acquisition code without significantly changing its behavior. No restrictions are imposed upon the user analysis process except that it must periodically call an update routine. It does not need to run on a workstation, and is not linked with any graphical user interface code. These user library routines are described in Part III of this guide.

The other part of Histo-Scope is the "scope" program. Running as a separate, concurrent process (and possibly running on a different machine), the scope requests and displays data from the data creation process. The scope is easy to use. It responds to mouse and keyboard input and provides the interactive graphing and plotting that enable users to view their data quickly and effectively. Using the Histo-Scope program is described in Part I of this guide. Users familiar with graphical user interfaces will probably want to skim through this part of the manual, especially the section *User Interface Overview.*

Histo-Scope is intended to be used in an online environment as well as in offline applications. In contrast to other existing histogramming packages, Histo-Scope was designed to have somewhat limited capabilities for exhaustive analysis of histograms or Ntuples. Instead, it is a small, highly interactive tool, written exclusively for workstations or X-terminals supporting Motif and X-Windows graphics. It emphasizes graphics representation, graphics speed, and maximum exploitation of windowing and mouse interaction. For example, while Histo-Scope does not provide curve fitting, it does provide a number of animation techniques such as dynamic re-binning of Ntuple histograms and the ability to interactively rotate two dimensional histogram plots.

New Features for Histo-Scope V4.0

The most significant new features in Histo-Scope V4.0 are:

- o New "View Overlaid" button in main panel - allows you to choose plots and display them in ONE plot window, overlaid on top of each other. Some plot settings now refer only to the first or "base" plot.

- o New "View Multiple" button in main panel - allows you to choose plots to be displayed in single multi-plot window.

- o Multiple selection of plots in main panel list box. Now you can select more than one plot in the list box of the main panel. To do this, use the Ctrl and Shift keys in addition to the first mouse button. You can also hold the mouse button down and select contiguous items in the list by dragging while holding the button down.

- o You can now re-title your plot window by selecting the "Change Window Title" plot setting from the plot settings menu (the menu that pops up when you press the right mouse button when pointing to a plot).

- o You can now specify in the histo command that you want to connect to a client. Use the -clientid option followed by the string you put in the HS_INITIALIZE call, and if necessary use the -rhost and -ruser options to specify the remote host and remote username (just as you would in the Connect to Client dialog). These options can be used with the -config option which specifies the configuration file to use.

- o Drag & Drop has been extended. In V3.1.1 you used Drag & Drop to drag plots into empty multi-plot window frames. Now, in addition, you can drag (2D) plots on top of other plots to overlay them. To undo this action (if this wasn't what you wanted to do), there's a new plot setting: Remove Overlaid Plot.

- o Histograms can now have various fill patterns. To specify these, select "Set Histogram Style" from the plot settings menu.

- o Histograms can now be plotted as a marker with/without error bars. To do this, choose the "Set Histogram Style" option from the plot settings menu and click on the linestyle option menu. At the bottom of the list of linestyles you will find marker types you can select.

- o Many 2D plots now allow colors to be specified for linestyles and markers: XY Plots, Time-Series, Histograms, Adaptive Histograms, Sorted XY Plots.

- o You can now ask the Histo-Scope main panel to display the data type of the items in the list box. There is a new toggle button for this at the bottom of the panel.

- o There are new callable routines in the client library:

  hs_histo_with_config - starts histoscope, tells it to read a config file
  hs_num_connected_scopes - returns the number of connected histoscopes
  hs_load_config_string - causes all connected histoscopes to load config commands
  hs_load_config_file - causes all connected histoscopes to load a config file
  hs_create_group - create a group
  hs_delete_items - delete a list of items (for better browser performance)
  hs_kill_histoscope - allows you to kill a histoscope you started.

- o A C++ Binding for the Histo-Scope API was contributed by Roque D. Oliveira at the University of Pennsylvania. These files can be found in Histo-Scope's contrib directory ($HISTO_DIR/contrib).

This user's guide is available as a World Wide Web page at URL:

http://www-pat.fnal.gov/histo_doc/histo_ug.html

A hardcopy version of the user's guide is available from the Fermilab Computing Division Library (WH8NE). A Postscript file of the manual is available in Histo-Scope's doc directory ($HISTO_DIR/doc).

Part I - Using the Histo-Scope Program

Getting Started with Histo-Scope

The Histo-Scope program is the "scope" part of the tool. It can be run independently or invoked directly from a user's program. If you are not using the workstation console display, and have instead used telnet or rlogin to access the host system, you will need to let the operating system know which display you are using:[1]
Set the DISPLAY environment variable:

% *setenv DISPLAY devicename:0*

For example, *setenv DISPLAY fnpspc.fnal.gov:0* tells the X software on a Unix host to use your X display fnpspc.fnal.gov.

Next, set up Histo-Scope by typing:[2]

setup histo

And, if running Histo-Scope directly, type:

% *histo*

You should shortly see Histo-Scope's Main Window pop up. If you experience any problems with starting or running Histo-Scope please refer to the sections on **Release Notes** and **Setup**. If invoking Histo-Scope from your own program, please refer to **Part III - Histo-Scope Library Subroutines** in this guide.

Optionally, if you want to view a Histo-Scope data file, you can specify this file to open on the Histo-Scope command line. If you would like to specify an HBOOK file to open on the command line, precede the HBOOK filename with “-hbook ”. For example:

```
% histo histoFile.hs
% histo -hbook hbookfile.hst
```

---

[1] For details on using X display devices with a remote UNIX host, refer to TN0050, *Distributed Graphics Applications in X Protocol*, available in the Fermilab Computing Division Library.

[2] The setup facility is available on Fermilab machines with UPS/UPD (Unix) or VPP (VMS). If you are off-site and do not have access to setup, please refer to the release notes for directions on setting up Histo-Scope for your use.

When listing an hbook file on the Histo-Scope command line, you can also specify a block size to use to open the file by preceding the block size with “-blocksize ”. For example:

```
% histo –hbook hbookfile.hst –blocksize  1024
```

Another command line option is to specify a clientid causing Histo-Scope to connect directly to a client process that is currently running. The clientid is the string which is passed to hs_initialize in your client program. You can optionally specify a hostname by including the “-rhost” argument and a username by including the “-ruser” argument. These options are only required if your client is running on a remote host or under another username.

```
% histo –clientid clientname –rhost hostname –ruser username
```

You can also specify a configuration file to open automatically by using the “-config” argument. This will re-load a previously saved configuration file which contains commands for opening windows, scaling plots, adjusting plot appearances, setting slider positions, etc. For further information, see page 21.

```
% histo –config file.cfg
```

The “-clientid” and “-config” arguments can be used together. Histo-Scope would first connect to the client process and then the configuration file would be read.

User Interface Overview for Beginning Motif Users

Histo-Scope utilizes the OSF/Motif windowing environment. This section gives an overview of the graphical user interface. For additional information, refer to the OSF/Motif User's Guide. Users familiar with OSF/Motif can probably skip this section.

Whenever you move the mouse, the pointer on your screen moves correspondingly. When this guide tells you to "point to" something, it simply means move the mouse until the pointer on your screen is positioned over the "something". The mouse buttons enable you to perform the following actions:

- "Click" means press and release a button without moving the pointer.
- "Double-click" means click a button twice in rapid succession.
- "Drag" means press and hold a button while moving the pointer.

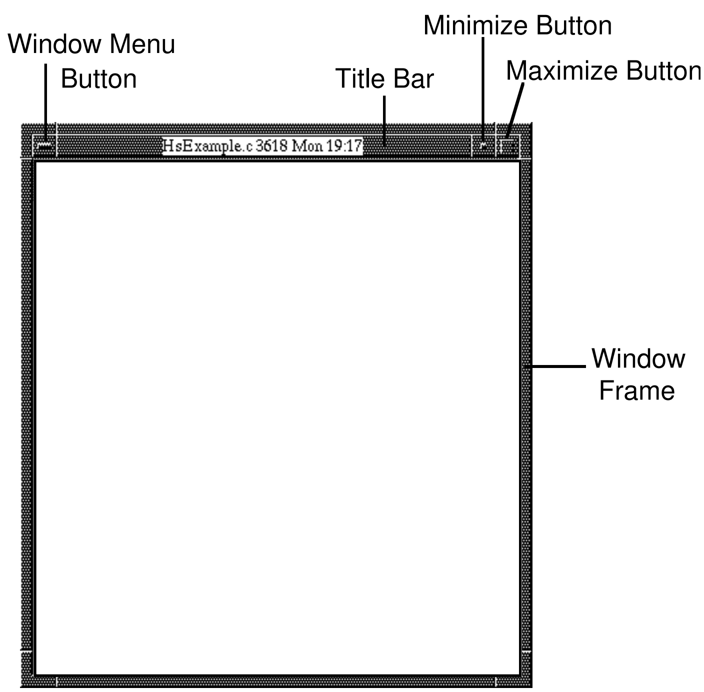


Figure 1. Motif Window Controls

Before a window can receive input from you, it must be selected as the active window. To select a window, point to any part of the window and click the left (select) button on the mouse.[1]

Dragging the **title bar** of a window will move the entire window.

Clicking the **minimize button** will change the window to an icon. (To restore the icon back into a window, point to the icon and double-click the select button.)

The **window menu button** is used to display or "pull-down" the **window menu**. It allows the following functions:

Restore	restore a window from an icon or after maximizing
Move	change the location of the window
Size	change the size of the window
Minimize	shrink the window to its icon representation
Maximize	enlarge the window to cover the entire root window
Lower	put a window visually behind other windows on the screen
Close	immediately stop the window and make it disappear

You can change the size of a window by dragging the part of the window frame that you want to alter. For instance, dragging the bottom right corner of the window frame downward and to the right will enlarge the window in both of those directions. Changing window size is useful for enlarging a histogram or plot, increasing variable assignments in the Ntuple Window, or increasing the number of visible items in the Main Window item list.

All Motif windows contain the window controls described above. Histo-Scope uses the Motif *Application Controls* described below.

---

[1]If your X display is set up with **implicit** keyboard focus policy, rather than the default **explicit** focus, you need not click the select button for the keyboard focus to change as you move the mouse. For specific information on keyboard focus policies, refer to the OSF/Motif Style Guide and the OSF/Motif Programmer's Guide.

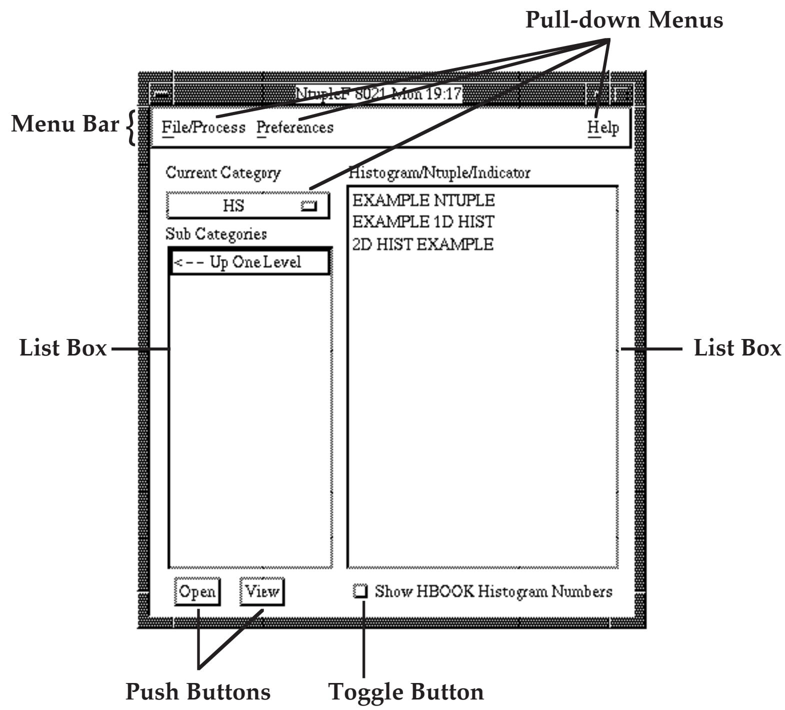


Figure 2. Motif Application Controls Used by Histo-Scope

The Histo-Scope Main window has a **menu bar** across the top of the window (just below the title bar). Each word in the menu bar represents a pull-down menu, which is displayed by pulling down on the menu bar with the mouse. To choose a function, you can use either of the following two methods:

- *Drag Method:* Point to the word for the menu you want to display, press and hold the select mouse button to display the menu, drag the pointer to highlight the desired function, and then release the button.

- *Click Method:* Point to the word for the menu you want to display, click the select button once to display the menu, point to the desired function, and then click again.

If you decide you don't want to execute a function after the menu is displayed, point away from the menu (so that no function is highlighted), and then release or click the select button.

Menus and their functions can also be accessed without using the mouse. Instead, a key combination can be typed. This is helpful when you don't want to move your hand from the keyboard to the mouse and for experienced users familiar with Histo-Scope's menu items.

- A *mnemonic* allows you to select a menu which has one of its letters underlined. To select that menu, press the *Alt** key while typing the letter that is underlined. For example to pull down the *File/Process* menu, type *Alt* F.

  After pulling down the *File/Process* menu, you can select one of the items in it by typing the letter that is underlined in that item. For example, to choose *Open Histo-Scope File*, type O.

- An *accelerator* key for most of the menu items *inside* of a pull-down menu is listed next to the item. Just type that key when the window containing that menu has keyboard focus to perform the desired function. For example, to *Open Histo-Scope File*, type *Ctrl* O (hold down the control key while you type O).

When a menu item cannot be selected, its name is greyed out (the name looks very light). Greying-out is also used for other controls (like list boxes and pull-down menus) when they cannot be selected. This is usually the case when the function that the control performs does not make sense in the current context or state of the window.

Pull-down menus outside of the menu bar work in a similar fashion. These menus typically appear as text outlined by a box containing a square menu button. When you click on the pull-down menu, the menu pops up over the box.

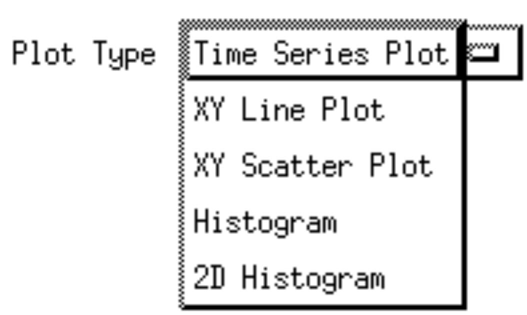


Figure 3. A Pull-down Menu

A list of items are often enclosed in a **list box**. List boxes can be used just for display purposes or can allow for selection of an item in

* on DEC keyboards without an *Alt* key use the Compose Character key.

a list. Histo-Scope lists are constrained to select only one item from a list at a time. To select an item in the list, do one of the following:

- Point to the item and click the select mouse button,
- Or, use the up- and down-arrow keys to move the highlight to the item to select

After selecting an item in a list, another control, a *push button*, is used to perform an action on the list item. Alternatively, double-clicking an item will perform the default action on that item.

A **push button** is a control that causes an immediate action when pressed. To press a push button, point to it and then click the select button. Push buttons can direct Histo-Scope to perform a desired function or pop up another window for your interaction with a related set of functions. If a push button has a dark frame outside and around it, typing <Return> is equivalent to clicking the mouse to select the push button. In addition to window-specific push buttons such as *Open* and *View*, Histo-Scope uses standard push button labels for the following common functions:

OK causes Histo-Scope to apply any changes you made in the window and dismiss it.

Apply applies any changes you made in the window without dismissing the window.

Reset cancels any changes you made to application controls in the window and resets their contents to the window's initial state (when it was popped up).

Cancel closes the window without performing any changes you made since the window popped up or the Apply button was pressed. The state of the window will be the same as if you hadn't changed anything.

Dismiss just makes the window disappear without affecting its contents or state.

Whenever a Histo-Scope list box is too small to hold all the items that need to be displayed, **scroll bars** allow you to control what portion is displayed. Scroll bars usually reside at the right edge (for vertical control) or bottom (for horizontal control) of a box. Each scroll bar has an arrow at each end, and a rectangular slider in the middle. The slider represents the current (relative) position of the displayed portion of the screen and the size of the displayed area in relation to the total area that can be displayed. When the slider is moved, that portion of the screen represented by the slider is displayed. The action of the display as it moves is called scrolling. There are three ways to scroll items and move the slider:

- *Using the arrow buttons* - point to the arrow that points toward the direction you want the slider to move; press and hold the select button until Histo-Scope displays the portion you want to see. This scrolls *line by line* (for the vertical scroll bar) or by "column" (horizontal scroll bar).

- *Using the slider* - grab the slider and move it where you want it. The portion of the screen represented by the slider is displayed.

- *Using the space between the slider and an arrow button* - click on the space between the slider and one of the arrow buttons. The text will scroll one "page" in the direction of the arrow. Pressing and holding the select button will scroll *page by page* until you release the button.

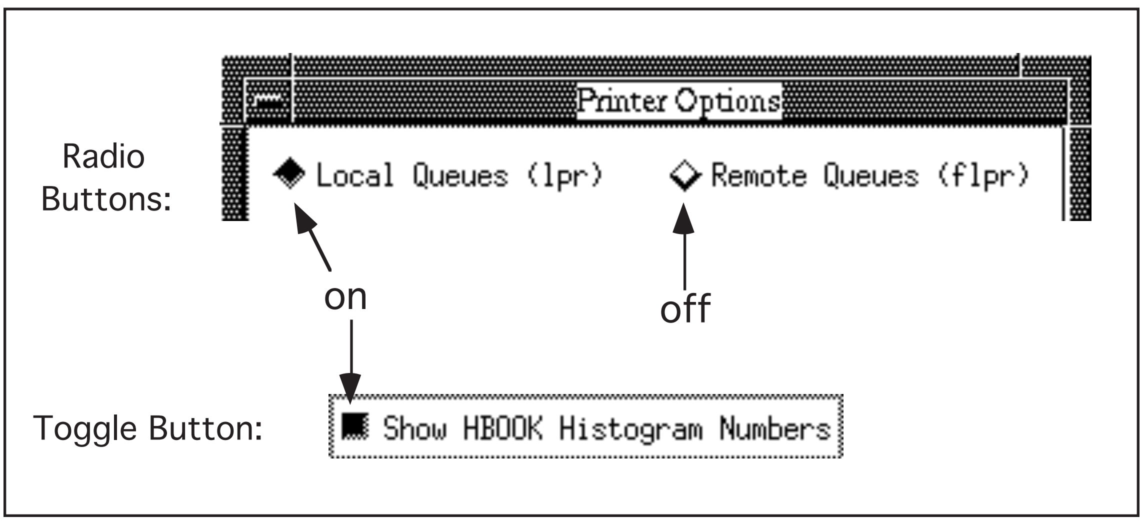


Figure 4. Toggle Buttons

A **toggle button** is used to set and indicate a condition to be on or off. When a toggle button appears "in" (or filled with shadow) the button is "on". When the toggle button appears "out" (or full of light) the button is "off". To change the state of the toggle button, just click on it with the select button of the mouse. There are two kinds of toggle buttons: A **radio button** is diamond shaped and is always part of a set. Only one of the radio buttons in the set can be on at a time. Choosing another radio button in the set clears the button that was previously on. A regular toggle button, sometimes called a check button, is square shaped and is independently on or off as set by the user.

In the *Connect To Process* and *Scaling Variables* dialog boxes, **text areas** are provided for input and/or display of text. To type into an input text area, point to it with the mouse and click* the select button before typing. The I-shaped cursor indicates the typing position. You can use the <Backspace>, <Delete>, and arrow keys inside text fields for editing your input.

Histo-Scope was designed to display messages to alert you to important conditions or errors. These messages "pop up" over the window you're currently using and typically block you from doing anything with Histo-Scope until you to acknowledge the message. To do this, just click on the appropriate push button inside the message window.

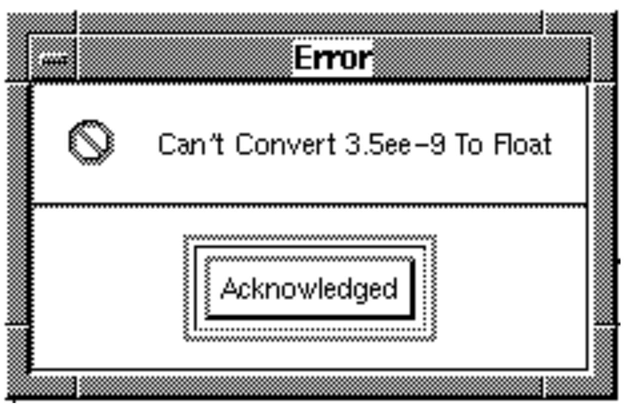


Figure 5. A Typical Error Message

---

* If your X display is set up with implicit keyboard focus policy you need not click the select button for the keyboard focus to change to the text area.

Using Histo-Scope's Main Window

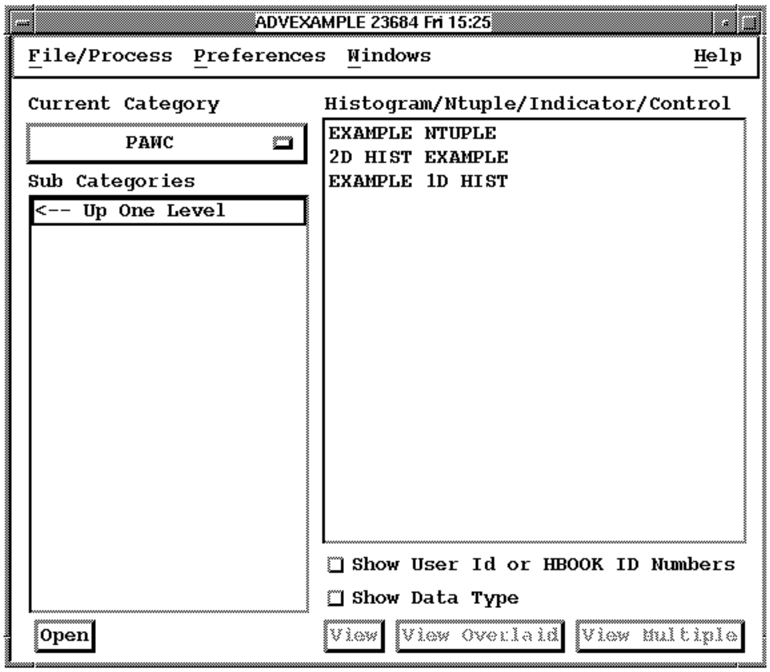


Figure 6. Histo-Scope's Main Window

When you run Histo-Scope or invoke it from a program, the main window pops up. This window allows you to specify how Histo-Scope should access your data and which data items to view. Unless Histo-Scope was invoked from a running program, the *Category* pull-down menu and the *Sub Categories* and *Histogram/Ntuple/Indicator/Control* list boxes are greyed out. This is because you need to tell Histo-Scope to open a file or connect to a running process. (If Histo-Scope was invoked from a running program, the File/Process menu is replaced by a *Window* menu, and histograms, Ntuples, and other data items in the top-level category are automatically listed.)

Histo-Scope supports the notion of categories, equivalent to HBOOK's directories, for arranging large numbers of histograms and Ntuples in a hierarchy rather than in a single long list. You can navigate through the hierarchy by clicking on items in the "Category" list on the left side of the Histo-Scope window or by using the Category pull-down menu:

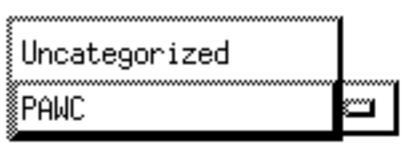


Figure 7. Category Pull-Down Menu

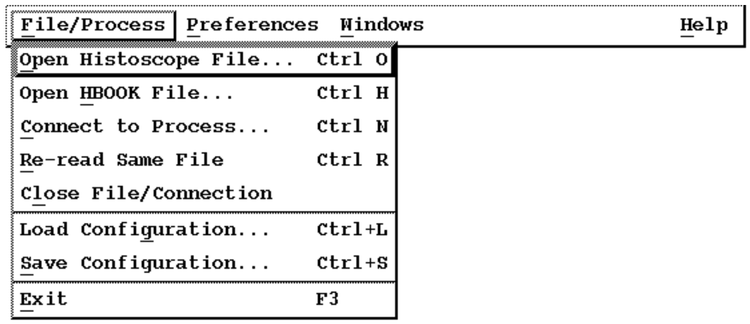


Figure 8. File/Process Pull-down Menu

You can use the *File/Process* pull-down menu in the menu bar to perform the following functions:

(Accelerator Key):

Open Histo-Scope File... *(Ctrl O)*

Histo-Scope will pop-up a file selection box through which you can specify the filename of the Histo-Scope format file. Then Histo-Scope will read the file and display the items it has read from the file. If you already have a file open or are connected to a process, Histo-Scope will first ask you if you want to close or disconnect before the file selection box appears.

Open HBOOK File... *(Ctrl H)*

Histo-Scope will pop-up a file selection box through which you can specify the filename of the HBOOK file. Then Histo-Scope will read the file and display the items it has read from the file. If you already have a file open or are connected to a process, Histo-Scope will first ask you if you want to close or disconnect before the file selection box appears.

Connect to Process... *(Ctrl N)*

Histo-Scope will pop-up a dialog box that lists all processes on the machine Histo-Scope was run from that it can connect to. If desired, a remote node name can be entered and the processes Histo-Scope can connect to on that machine will be listed. Pressing the Connect button or double-clicking on a process requests Histo-Scope to connect to the highlighted or selected process. Once the connection is established, Histo-Scope will display the top-level items and sub-categories of data items that Histo-Scope can access from that process. When selecting *Connect to Process*, if you already have a file open or are connected to another process, Histo-Scope will first ask you if you want to close or disconnect before it pops up the Connect to Process dialog box.

Re-Read Same File *(Ctrl R)*

If you currently have a file open, you can select *Re-Read Same File* to update Histo-Scope with new data that may have been written to the file since Histo-Scope opened it. To ensure that all your plot windows reflect new data, Histo-Scope will close all plot windows (after checking that it's OK to do this) before re-reading the file.

Close File/Connection

Selecting *Close File/Connection* allows you to close a file or connection, but leaves all displayed plots available for viewing. If you were connected to a process, no more updating will occur.

Load Configuration *(Ctrl L)*

Configuration files are files of Histo-Scope commands for opening windows, scaling plots, adjusting plot appearance, setting slider positions, etc. *Save Configuration...* saves the current appearance of your Histo-Scope session (excluding opening the file or making the connection). *Load Configuration* re-loads these configuration files. Use *Load Configuration* after you have opened a data file or connected to a process. More than one configuration file can be loaded in a single session. Loading additional files simply displays the additional plots specified in the new files. Note that configuration files do not specify the file or connection to be opened, nor do they specify Preferences settings. Configuration files are an editable but undocumented ASCII text file format.

Save Configuration *(Ctrl S)*

Save Configuration saves the current appearance of your Histo-Scope session (excluding opening the file or making the connection) to a file which can later be loaded with *Load Configuration*. Histo-Scope will pop up a file selection box through which you can specify the file name of the configuration file. Note that configuration files do not specify the file or connection to be opened, nor do they specify Preferences settings. Configuration files are an editable but undocumented ASCII text file format.

Exit *(F3)*

Histo-Scope will exit.

After opening a file or connecting to a process, Histo-Scope will change the title in the title bar from "Histo-Scope" to the filename or process you specified. Also, the histograms, Ntuples, and other data items belonging to the top-level category will be listed. The *Category* pull-down menu and *Sub Categories* list box can be used to select sub-categories and navigate through the category (or HBOOK directory) structure for your data. To select a sub-category, click on it and press the *Open* button, or just double-click on the sub-category.

To view a histogram, Ntuple or other data item, first, select the one or more items that you are interested in. If you have one item to select, simply point to it with the mouse and either click on it once and then press the *View* button, or just double-click on the item in the list box. If you have more than one item to view, there are a few different ways to select them. If they are contiguous items in the list, you can press the left mouse button and drag the mouse over the items you wish to select, then release the button. A second way to do this same thing would be to point and click on the first item, then move the mouse to the last item you wish to select and point and click on that item while holding down the shift key. Then you can press the *View* button. If you want to select items that are not contiguous in the list, select the first item (or group of items) as described above, then move the mouse to the next item and select it by pointing and clicking on that item while holding down the ctrl key. Then you can select the *View* button to display the plots.

If the items selected are all 1D Histograms or groups, you should notice that two additional *View* buttons will become sensitive to mouse-clicks, the *View Overlaid* button and the *View Multiple* button. The *View Overlaid* button will cause the plots to be displayed overlaid on each other and the *View Multiple* button will cause the plots to be displayed in a multiple plot window. If one or more items selected is a 2D Histogram, you can view it in a multiple plot window, but it cannot be overlaid. Therefore, only the *View* and *View Multiple* buttons will be sensitive. When viewing a histogram, indicator, control, or trigger a plot window will pop up displaying the item using the default style for that type of data. When requesting an Ntuple, the Ntuple Window will first appear to allow you to specify various kinds of plots and variable assignments.

When the item selected is a group, the *View* button will display the items in the group according to the group type specified by the user program that created the group (see HS_CREATE_GROUP). Thus plots will be displayed individually, in a multiple-plot window, or in an overlaid plot window, according to this default group type. The *View Multiple* and *View Overlaid* buttons override the default behavior.

Once an item is displayed, you can change the presentation style via pop-up menus from the plot or indicator window using the third (right) mouse button (see section **Adjusting Plot Settings**). The plots can also be rescaled by changing the window size, dragging on the axis tick marks, or typing new endpoint values in the Axis Settings dialog. Two-dimensional histogram and 3D scatter plots also allow rotation of the plot in space. Any number of histograms, Ntuples, plots of Ntuples, or alternate views of histograms and Ntuples can be displayed simultaneously in separate windows, and all plots can be printed. Further details on the capabilities and functions of each data/plot type can be found in subsequent sections of this guide.

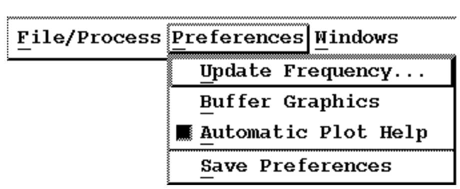


Figure 9. Preferences Pull-down Menu

The *Preferences* pull-down menu in the menu bar allows you to specify:

- how often Histo-Scope should request new data from a connected process,
- to set/unset buffer graphics mode,
- to set/unset whether plot help windows pop up the first time you view that kind of plot, and
- to save your preferences of the above.

Update Frequency...

Update frequency is used only when connected to a running process. The default value is set to three seconds, meaning that every three seconds Histo-Scope will request new data from the connected process. To change the update frequency, use the mouse to drag the slider bar to the value you desire. Increasing this value will save CPU time at the cost of a longer interval between plot updates.

Buffer Graphics

Initially Buffer Graphics mode is off (indicated by the absence of the square toggle button to the left of the words *Buffer Graphics*. Graphics buffering changes the way Histo-Scope draws moving graphics in X-Windows. Depending on your X-Windows server or X-Terminal, turning on buffered graphics may either improve or degrade animation quality. Whether your hardware will run well with buffer graphics turned on depends on a number of relatively complicated factors. The best way to find out is just to try it. Before you do, however, note that some X-Terminals and PC based X servers may not have enough memory to run properly in buffered graphics mode. When an X server runs out of memory, it will kill the process that made the last memory request, or in some cases, crash itself. Most X-Terminals and PC X servers with limited memory capacity have some kind of memory display that you can invoke to monitor their memory usage. It is advisable to monitor your X server's memory usage the first time you use buffered graphics. The amount of memory that Histo-Scope needs for buffered graphics is dependent on the number and size of plot windows that you have open.

Automatic Plot Help

For new users, Histo-Scope will by default pop up a window that explains how to interact with the type of plot just displayed the first time that kind of plot is viewed. Experienced Histo-Scope users will probably want to unset this option and save their preferences.

Save Preferences

Any of the above preferences can be set and then saved to a file which is automatically re-loaded every time HistoScope is started. Selecting *Save Preferences* generates this file in your home directory with the name .histo .

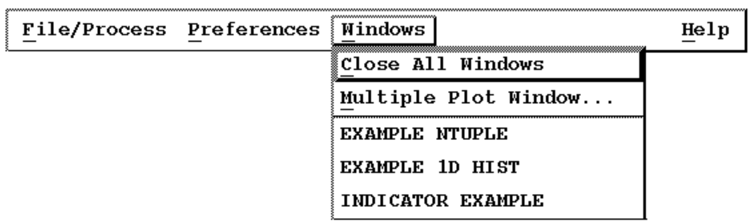


Figure 10. Windows Pull-down Menu

The *Windows* pull-down menu in the menu bar lists all of the currently displayed plot windows. Selecting one will bring that plot to the front of all other windows. In addition to this list of plot windows, the *Windows* menu also allows the user to close all plot windows and create a window containing multiple plots:

Close All Windows

Selecting *Close All Windows* closes all of the open plot windows without closing the currently open file or connection to an attached process.

Multiple Plot Window...

Selecting *Multiple Plot Window...* will pop up a window which allows the user to select plots to appear together in one larger window. This is helpful for grouping plots together that are related in some way and provides a mechanism that makes all the selected mini-plots visible at the same time.

Histo-Scope also provides a *Help* menu which gives online help to the user.

Create Multiple Plot Window

To create a plot window containing more than one plot, select *Multiple Plot Windows...* from the Windows menu of the Histo-Scope Main Window. The Create Multiple Plot Window allows the user to select plots to appear together in one larger window. This is helpful for grouping plots together that are related in some way and provides a mechanism that makes all the selected mini-plots visible at the same time.

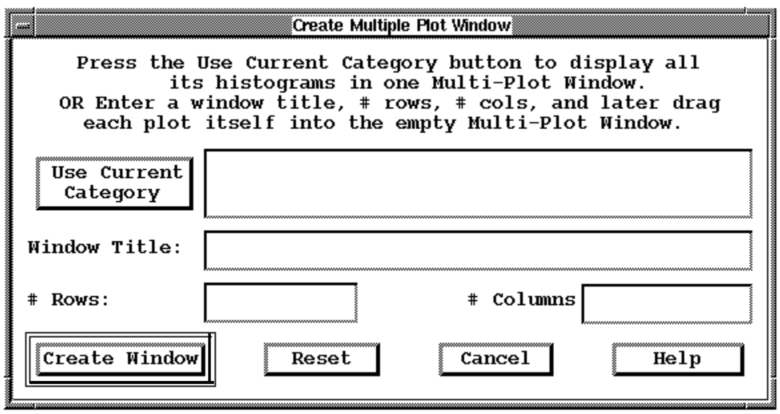


Figure 11. Create Multiple Plot Window

There are two ways these multiple-plot windows can be populated with plots:

1) All the 1-D and 2-D Histograms in the current category (as listed in the Histo-Scope Main Window) can be displayed in a single multi-plot window. To do this, click on the *Use Current Category* push button. Histo-Scope will automatically fill in the multi-plot window title and a default number of rows and columns that will accommodate the number of histograms in the current category. The window title, # Rows, and # Columns can be modified by the user as desired.

2) For more flexibility, <u>don't</u> press the *Use Current Category* button. Instead just type the window title and number of rows and columns for the Multiple Plot window, and after pressing "Create Window" use the mouse to drag the plots one-by-one into the initially empty multi-plot window. This method gives you the most flexibility for choosing and placing plots for display.

Click on the *Create Window* button to create the Multiple Plot window. To insert plots of Ntuple variables into a multiple plot window, leave one or more mini-plot "cells" empty and use the middle mouse button to drag a plot onto the empty cell of the mini-plot. (To drag the plot, press the middle mouse button while the pointer is over the plot area of the desired Ntuple plot, and, while holding the mouse button down, move the mouse pointer over to the empty mini-plot area, and then release the middle mouse button. When you press the middle mouse button, the pointer changes to indicate that your are initiating a drag.)

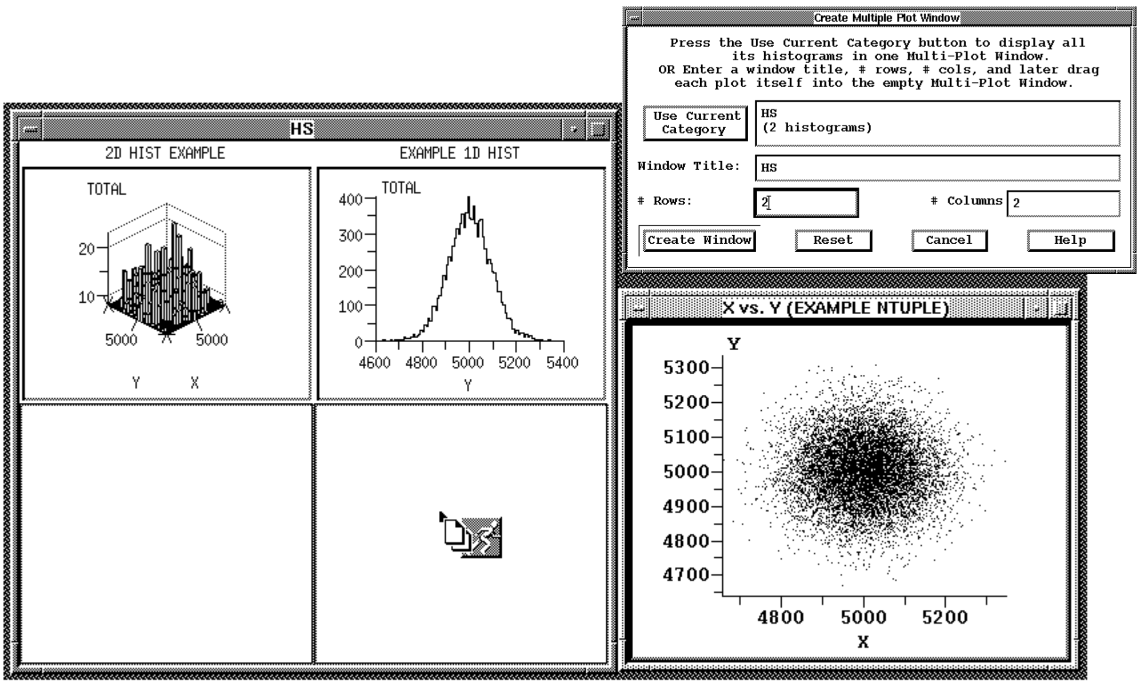


Figure 12. Dragging an Ntuple Plot into a "Mini" Plot

To change the size of the individual plots in Multiple Plot windows, use Motif window controls such as the window sides or corners to make the entire window larger or smaller.

Each mini-plot in the Multiple Plot window allows you to adjust its plot settings in a manner similar to the individual plot windows. To bring up a Plot Settings menu, point to the plot with the mouse and press the right mouse button.

Connect to Process Window

When you select *Connect to Process* from the Main Window's *File/Process* pull-down menu, Histo-Scope will pop-up a dialog box that lists all processes on the machine Histo-Scope was run from that it can connect to. That is, each running process is listed that has called the HS_INITIALIZE routine thereby making itself known to Histo-Scope.

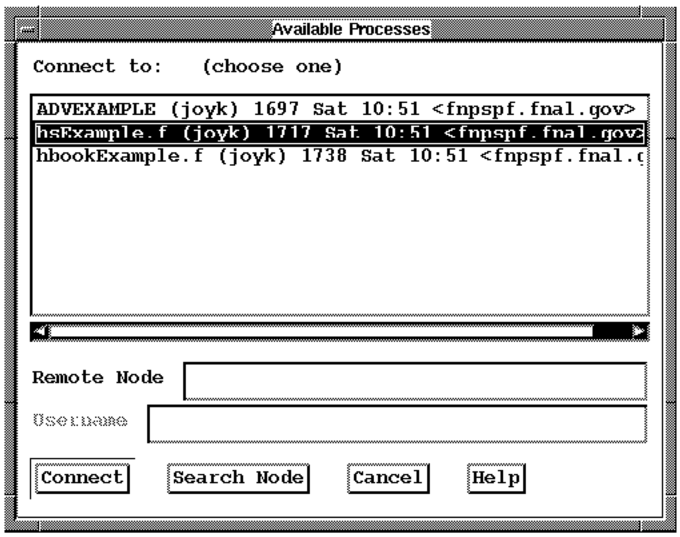


Figure 13. Available Processes Window

Histo-Scope allows heterogeneous connections. For example, you can run Histo-Scope on an SGI Unix system and connect to a process running on a DEC OSF-1 system.

If you want to connect to a process on another machine, enter a remote node name and press the *Search Node* button. The processes Histo-Scope can connect to on that machine will then be listed after a pause for searching that remote machine. The .rhosts file on the remote node must allow access from your username on the current node. If your username is different on the remote node than it is on your current system, you can specify a remote username, and that user's .rhosts file will be checked for access from your current username and system.

If more than one process is listed, look for the identity string supplied in the HS_INITIALIZE call of your program, the user-name used to start the process, and the time the process executed the HS_INITIALIZE call (or was started).

Pressing the *Connect* button or double-clicking on a process requests Histo-Scope to connect to the highlighted or selected process. Once the connection is established, Histo-Scope will display on the Main Window the top-level items and sub-categories of data items that Histo-Scope can access from the process.

When selecting *Connect to Process* from the Main Window, if you already have a file open or are connected to another process, Histo-Scope will first ask you if you want to close or disconnect before it pops up the Connect to Process dialog box.

If you are running Histo-Scope on a Unix system, do not put the Histo-Scope process in the background if you plan to connect to a process. This is because the system may put the Histo-Scope process into a wait state that it will not get out of until the process is put into the foreground. After the connection is successful, the Histo-Scope process may be put into the background, if necessary, until you need to connect to another process.

Verifying Proper Setup of Permissions

Processes whose data can be viewed using Histo-Scope use the computer's file system to communicate their existence to the Histo-Scope program. The HS_INITIALIZE call creates a small file which can be read by a Histo-Scope program that is trying to make a connection. On Unix, Histo-Scope will create and look for running process identifications in the directory pointed to by the TMPDIR environment variable, if defined, or /tmp if TMPDIR is not defined. However, for remote connections, Histo-Scope looks only in the /tmp directory on the remote system.

For remote access to a running data-generating process you can verify whether you have correctly set up your .rhosts file on that system:

Type to the shell on the system you will run Histo-Scope:

% rsh *nodename* ls /tmp

where *nodename* is the remote host name you will be connecting to (i.e. where your data-generating process will be running).

The following is an example .rhosts file:

```
fnalv.fnal.gov  smith
fnalo.fnal.gov  smith
fnala.fnal.gov  smith
fnpspf.fnal.gov  jonsmith
```

This .rhosts file should be created on the host where your data-generating process will run and specifies the access to this account **from** each node/username listed above (i.e. where you expect to run Histo-Scope). On Unix, .rhosts resides in your home directory.

After typing the above rsh command, you should see a directory of files (or a message saying no files were found). If an error message is printed, use this to help correct the situation. For instance, if the message printed is Permission Denied, check your .rhosts file to ensure it allows access to your username on the current host system.

One-Dimensional Histograms

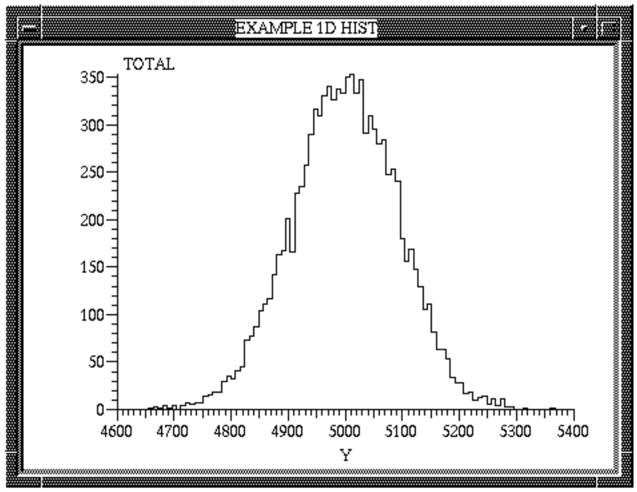


Figure 14. One-Dimensional Histogram

When a one-dimensional histogram is selected from the Main Window or from the Ntuple Window, Histo-Scope will display a one-dimensional histogram plot. A one dimensional histogram is a plot of one variable over a pre-specified range which has been divided into a number of bins.

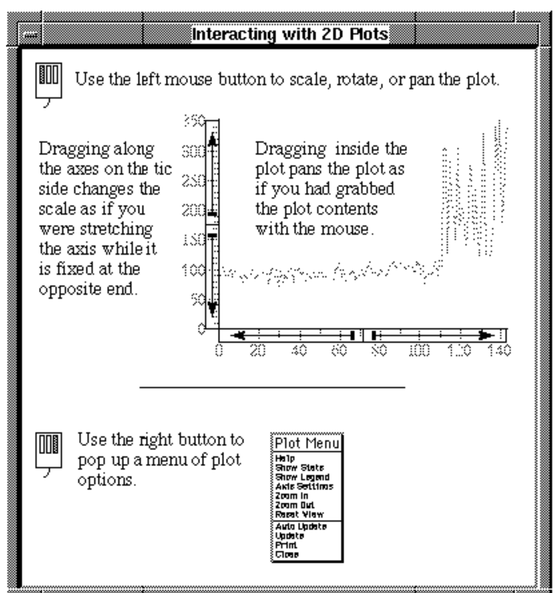


Figure 15. Interacting with 1-D Histograms

Initially, the entire range (excluding overflows) of the histogram is displayed. You can change the size of the plot by changing the size of its window (by dragging on the window frame). You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use an axis point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

Once you have scaled a plot, it is possible to grab a point inside the plot and pan the plot, moving it around inside the axes.

Histo-Scope allows one-dimensional histogram plots to be overlaid by other one-dimensional plots such as 1-D histograms, XY Plots, and Time Series Plots. Up to ten plots can be overlaid in one window. To insert a plot on top of a one-dimensional plot, use the middle mouse button to drag a plot onto the one-dimensional histogram window (To drag the plot, press the middle mouse button while the pointer is over the plot area of the desired plot, and, while holding the mouse button down, move the mouse pointer over to the receiving histogram plot, and then release the middle mouse button. When you press the middle mouse button, the pointer changes to indicate that your are initiating a drag.) If the drag succeeds, you will see both plots overlaid in the receiving window (the source plot remains unchanged). If the drag does not succeed, you will see the drag icon move back to the source plot when you release the middle mouse button

You can also use the right mouse button to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders[1]	*(Ctrl L)*
Show Rebin Slider[1]	*(Ctrl B)*
Set Histogram Style...	*(Ctrl M)*
Show Legend	*(Ctrl G)*
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Error Data[2]	

---

[1] Show Sliders and Show Rebin Slider plot settings are available only for histograms displayed from an n-tuple.

[2] Error Data plot setting is **not** available for histograms displayed from an n-tuple.

Gaussian Errors
Label at Bin Edges
Automatic Update
Update *(Ctrl U)*
Grow Only *(Ctrl O)*
Print... *(Ctrl P)*
Generate PostScript... *(Ctrl F)*
Change Window Title *(Ctrl E)*
Close *(Ctrl W)*

Error data stored for the histogram can be displayed as error bars by selecting *Error Data* from the plot settings menu. If selected from the Ntuple Window, up to three slider variables can be selected for adjusting the plot with animation sliders. See the section **Adjusting Plot Settings** in this guide for detailed explanations of plot settings.

Two-Dimensional Histograms

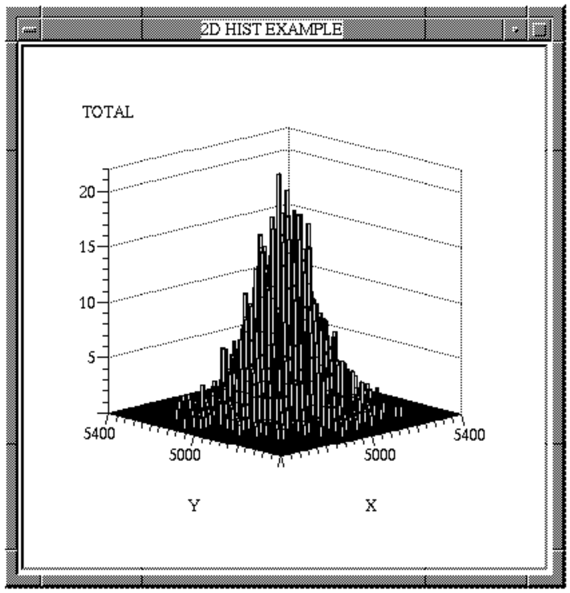


Figure 16. Two-Dimensional Histogram

When selected from the Main Window or the Ntuple Window, Histo-Scope will display a two-dimensional histogram plot. A two dimensional histogram is a plot of two variables, where each axis is divided into a number of bins and has its own range and bin size.

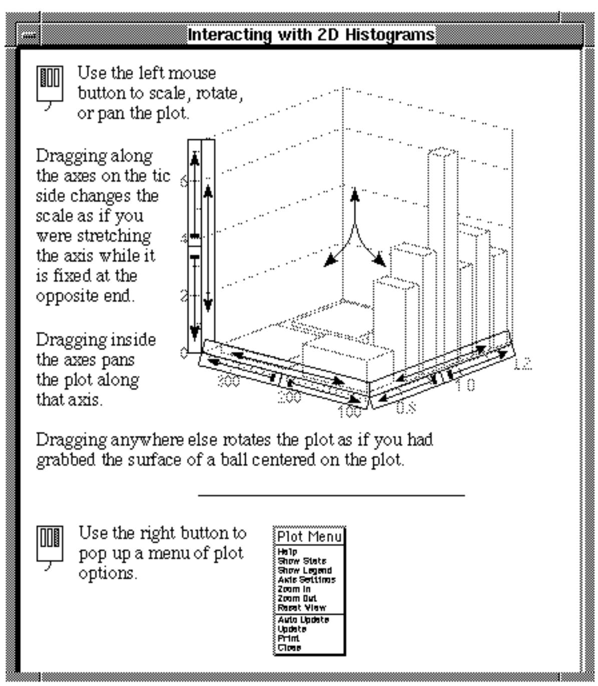


Figure 17. Interacting with 2D Histograms

Initially, the entire range (excluding overflows) of the histogram is displayed. You can change the size of the plot by changing the size of its window (by dragging on the window frame). You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use an axis point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

Once you have scaled a plot, it is possible to grab a point inside the plot (but still near the axis) and pan the plot, moving it around inside the axes.

The orientation of the histogram can be changed by grabbing a part of the display (away from an axis) with the select (left) mouse button and moving the display in space to a different orientation.

Using the Ctrl key along with the mouse button for the above scaling, panning, and rotation functions will provide a continuous display of the 2d histogram. Without using the Ctrl key, only the axes and labels are displayed while you are adjusting the plot. (This is the default mode to accommodate slow displays).

You can also use the right mouse button to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders[1]	*(Ctrl L)*
Show Rebin Slider[1]	*(Ctrl B)*
Other Views	
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Error Data[2]	
Gaussian Errors	
Show Backplanes	
Label at Bin Edges	

---

[1] Show Sliders and Show Rebin Slider plot settings are avilable only for histograms displayed from an n-tuple.

[2] Error Data plot setting is **not** available for histograms displayed from an n-tuple.

Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
Close	*(Ctrl W)*

Error data stored for the histogram can be displayed as error bars by selecting Error Data from the plot settings menu. If selected from the Ntuple Window, slider variables can be selected for adjusting the plot with animation sliders. See the section **Adjusting Plot Settings** in this guide for detailed explanations of plot settings.

Another view of the two-dimensional histogram as a two-dimensional cell plot can be requested by selecting *Other Views* from the Plot Settings menu. Please see page 45 for further information on the cell plot.

Indicators, Controls, and Triggers

The combination of indicators, controls and triggers allows the user to control the execution of his data acquisition or analysis program from Histo-Scope. For instance, should peculiar results come out of diagnostic histograms, certain code paths can be internally enabled or disabled, or the job gracefully terminated.

Indicator Window

An indicator is a scalar value set by a process to indicate the value of a variable or condition. When selected for viewing from the Main Window, an Indicator Window pops up:

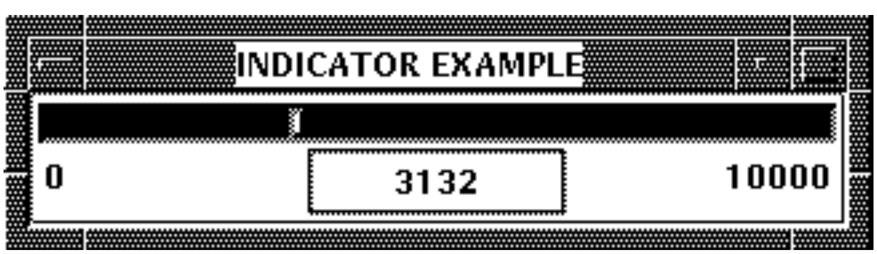


Figure 18. Indicator Window

The Indicator window by default shows the minimum and maximum values of the indicator (at the far left and right), and the value of the indicator is displayed in the middle of the window in a rectangular box. If the indicator has not yet been given a value, or has been reset, the words * not set * will be displayed in the box.

Use the right mouse button to pop up the plot settings for indicators: *Help, Show Range, Automatic Update, Update,* and *Close.* See the section **Adjusting Plot Settings** in this guide for detailed explanations of plot settings.

Control Window

Controls allow a variable in a process to be set interactively using Histo-Scope. When selected for viewing from the Main Window, a Control Window pops up:

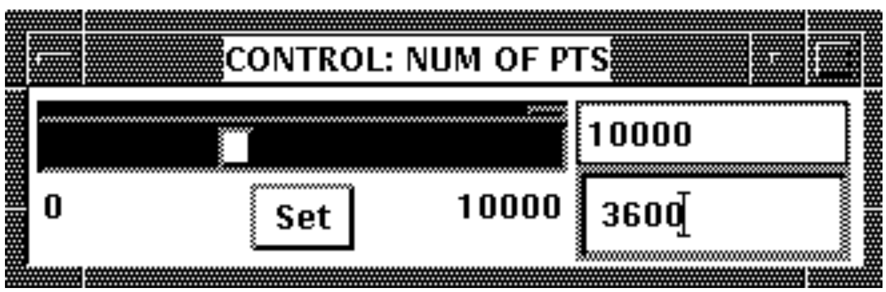


Figure 19. Control Window

To set the value of the variable, first either type a value in the text field in the lower right corner of the control window, or use the slider to enter the value. Then, press the *Set* button. When the value is read and accepted by the connected process, the current value (in the top right corner) and the indicator above the slider will change to reflect the requested value. In the connected process, controls can be read by calling hs_read_control.

Controls displayed from files or a disconnected process cannot be set.

Use the right mouse button to pop up the plot settings for controls: *Help, Show Range,* and *Close.* See the section **Adjusting Plot Settings** in this guide for detailed explanations of plot settings.

Trigger Window

Triggers are buttons which can cause an action in a connected process. When selected for viewing from the Main Window, a Trigger Window pops up:

Figure 20. Trigger Window

Each time the button in the trigger window is pressed, a message is sent to the process and, if the code in the process properly checks the trigger, an associated routine or code fragment will be executed. Until the trigger is read by the connected process, the button will read "Pending." If the button is pressed several times before it is read, it will show a count of unprocessed presses. In the connected process, triggers can be detected by calling hs_check_trigger.

Ntuple Window

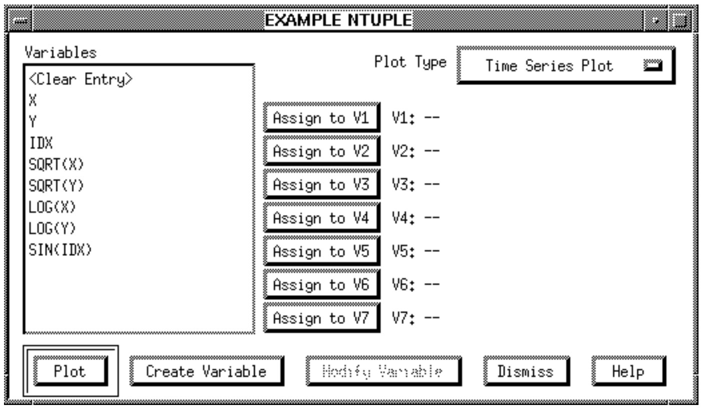


Figure 21. Ntuple Window

Whenever an Ntuple is selected from the Main Window, an Ntuple window pops up that allows you to specify the variables to plot and the type of plot desired. The name of the Ntuple appears as the window title so that you can readily identify the Ntuple Window of one data item from another.

First, select the type of plot from the *Plot Type* pull-down menu:

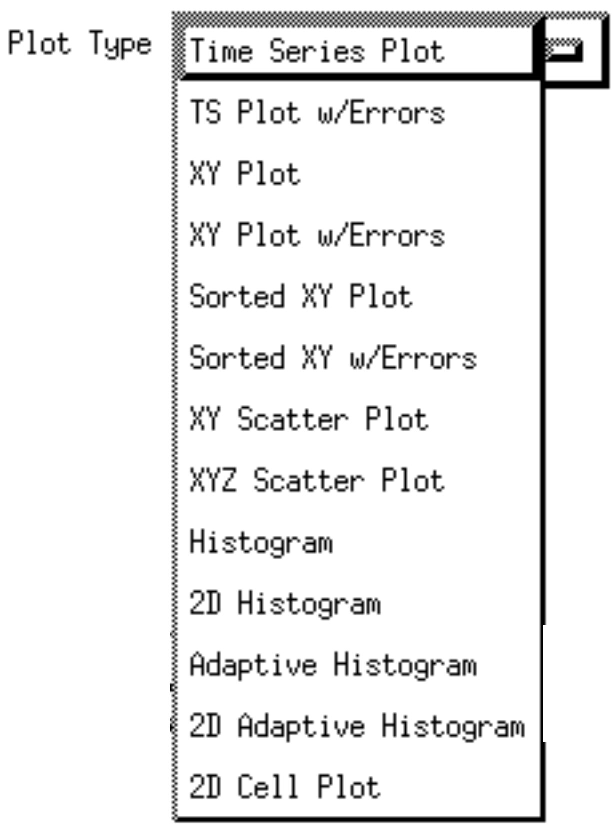


Figure 22. Plot Type Pull-Down Menu

Then, select variables from the *Variables* list box at left and assign the selected variable to the desired plot position by pressing the corresponding *Assign* push-button. Double clicking on a variable name will assign the variable to the next vacant assignment position. To de-assign a variable, click on "<Clear Entry>" and then click on the assignment position you'd like to clear. How each variable is used in the plot depends on the plot type and the assignment position:

Prefix	Use
V	Variable
X	Variable plotted against X axis
Y	Variable plotted against Y axis
Z	Variable plotted against Z axis
S	Variable not plotted, but used as a slider variable to see "cuts" of data
E+	Variable used as error values (+) for preceding variable
E-	Variable used as error values (-) for preceding variable
V+	Variable used as vertical error values for the preceding variable (XY Plot only)
V-	Variable used as vertical error values for preceding variable (XY Plot only)
H+	Variable used as horizontal error values for preceding variable (XY Plot only)
H-	Variable used as horizontal error values for preceding variable (XY Plot only)

Up to seventeen variables can be assigned for certain plot types. To expose additional plot assignments, drag the bottom frame of the Ntuple window downward with the mouse. The window will grow downward as you pull.

If only positive (+) errors are specified for a variable, Histo-Scope will display those error value as symmetric (+ and -) errors.

In addition to the variables that are part of the Ntuple, you can use the *Create Variable* button to derive additional variables and constants for plotting. This will pop up a window that allows you to enter the information for the created variable (see below). Once you have created a variable, selecting it in the Variables list and pressing *Modify Variable* will pop up a window for making changes.

When all the settings in the window are to your liking, press the *Plot* button for the chosen plot or histogram to be displayed.

Many plots can be requested from the same Ntuple Window. To request another plot, select the Ntuple window and repeat the above procedure.

The *Dismiss* button will dismiss only the Ntuple window. Plots and histograms currently displayed will remain open until explicitly closed or until Histo-Scope exits.

Use the *Create Variable* button to derive additional variables and constants for plotting. This will pop up a window that allows you to enter the name and expression for the created variable.

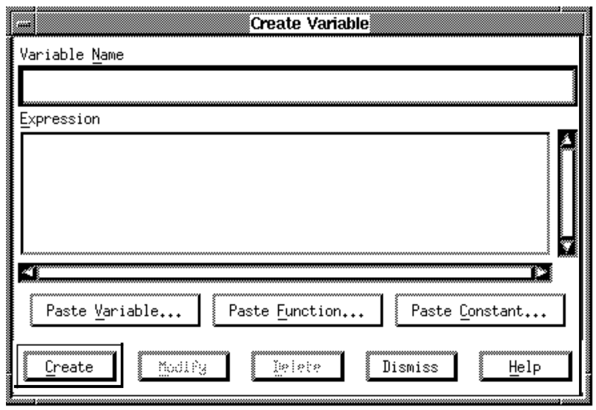


Figure 23. Create Variable Window

To create a derived variable, fill in the Variable Name and Expression fields in this dialog and press *Create*.

The variable name can be any arbitrary character string, although strings without spaces or punctuation are easier to use in expressions.

The expression should be a FORTRAN or C style arithmetic expression. Valid operators are: +, -, /, *, ^ (or **). There are also a number of built-in single-argument functions and constants. These can be pasted into the expression at the current text cursor position by clicking on the *Paste Function* or *Paste Constant* menus and selecting the expression or constant to insert. Variable names can similarly be pasted into the expression area by clicking on the *Paste Variable* menu. When typing variables that contain spaces or punctuation, or variables that start with a number, the variable name should be surrounded in double quotes ("). Multi-line expressions with temporary variable assignments are also allowed, for example:

a = sqrt(X) + 2*Y
b = Z**2 / (4*X)
a**2 + b*a**3

The value of the last line of the expression is taken as the value of the created variable.
Once the variable is created, it can be modified or deleted using the same window. If you dismiss the window or lose it under other windows on your screen, you can pop it up again by selecting the variable in the Ntuple Window and pressing *Modify Variable*.

Time Series Plots

A Time Series plot will plot the value of a variable from your Ntuple in the sequence in which that variable was collected. Thus, the x-axis values start at zero and increase to the number of occurrences in the Ntuple. The value of the Ntuple variable is plotted along the y-axis of the plot.
Up to eighteen variables can be plotted in one window. To reveal additional assignment buttons, use the mouse to grab the bottom frame of the Ntuple Window and pull downward. The window will grow as you move the mouse pointer downward.

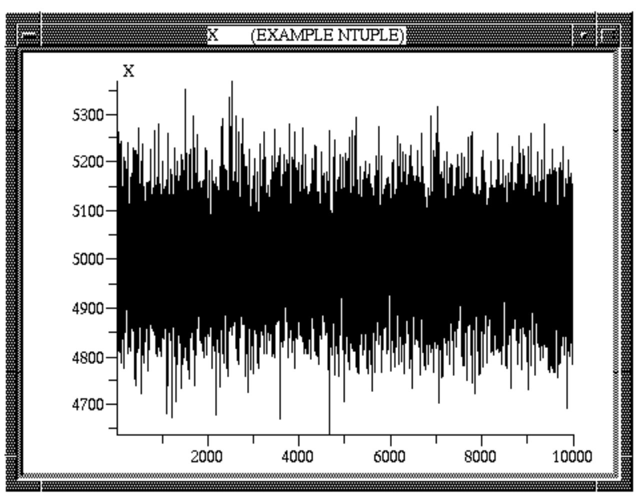


Figure 24. Time Series Plot

If only one variable is plotted, its name appears at the top of the y-axis. If more than one variable is plotted, a legend of variable names appears at the bottom of the plot.

Initially, the entire range of the plot is displayed. You can change the size of the plot by changing the size of its window (by dragging on the window frame). You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use an axis point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

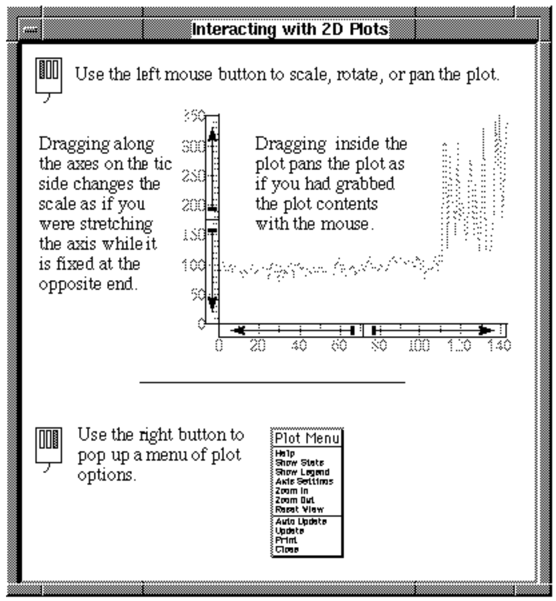


Figure 25. Interacting with Time Series Plots

Once you have scaled a plot, it is possible to grab a point inside the plot and pan the plot, moving it around inside the axes.

Histo-Scope allows Time Series plots to be overlaid by other one-dimensional plots such as 1-D histograms, XY Plots, and other Time Series Plots. Up to ten plots can be overlaid in one window. To insert a plot on top of a Time Series plot, use the middle mouse button to drag a plot onto the Time Series plot window (To drag the plot, press the middle mouse button while the pointer is over the plot area of the desired plot, and, while holding the mouse button down, move the mouse pointer over to the receiving time series plot, and then release the middle mouse button. When you press the middle mouse button, the pointer changes to indicate that your are initiating a drag.) If the drag succeeds, you will see both plots overlaid in the receiving window (the source plot remains unchanged). If the drag does not succeed, you will see the drag icon move back to the source plot when you release the middle mouse button.

You can use the right mouse button on a selected plot window to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Set Mark & Line Style...	*(Ctrl M)*
Show Legend	*(Ctrl G)*
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
Close	*(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

XY Plot

An XY Plot plots pairs of points in two dimensions. The points are taken in sequence from the Ntuple and are initially drawn with lines connecting the points and no markers. The XY Plot can also be used as an alternative to the Scatter Plot. In this case, use the Set Mark and Line Style plot adjustment to turn on markers and turn off connecting lines. (When the number of points is small, the marks drawn by the XY Plot are much more visible than the pixel-sized dots drawn by the Scatter Plot.) The XY Plot can also plot multiple sets of variable X-Y pairs from the Ntuple, distinguish them with different marker styles, and draw both horizontal and vertical error bars. Up to nine X-Y pairs can be plotted in one window. To reveal additional assignment buttons, use the mouse to grab the bottom frame of the Ntuple Window and pull downward. The window will grow as you move the mouse pointer downward.

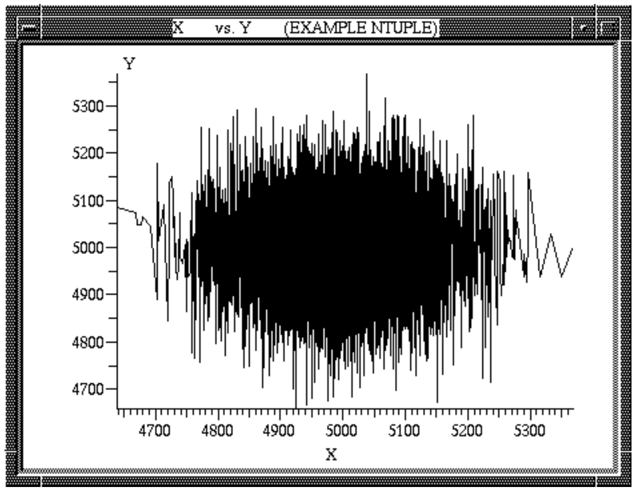


Figure 26. Sorted XY Plot

Since the points in an XY Plot are not sorted the initial appearance of the plot may include lines which loop back on themselves. Another type of XY Plot is the **Sorted XY Plot**. In a Sorted XY Plot, the x-axis variable is sorted in order of increasing value and each succeeding variable is plotted along the y-axis against the first (sorted) variable. This can often be a better way to observe how y varies with x. A Sorted XY Plot can display up to seventeen Y-variables in one window. If only one Y-variable is plotted, its name appears at the top of the y-axis. If more than one is plotted, a legend of variable names appears at the bottom of the plot.

Both types of XY Plots allow you to specify variables as error values. An XY Plot w/Errors can plot up to three X-Y pairs with their corresponding vertical and horizontal errors. A Sorted XY Plot w/Errors can plot up to five Y-variables and their corresponding (vertical) errors. (To expose hidden plot variable assignments, drag the bottom frame of the Ntuple window downward with the mouse. The window will grow downward as you pull.)

Initially, the entire range of the data in an XY Plot is displayed. You can change the size of the plot by changing the size of its window (by dragging on the window frame). You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use an axis point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

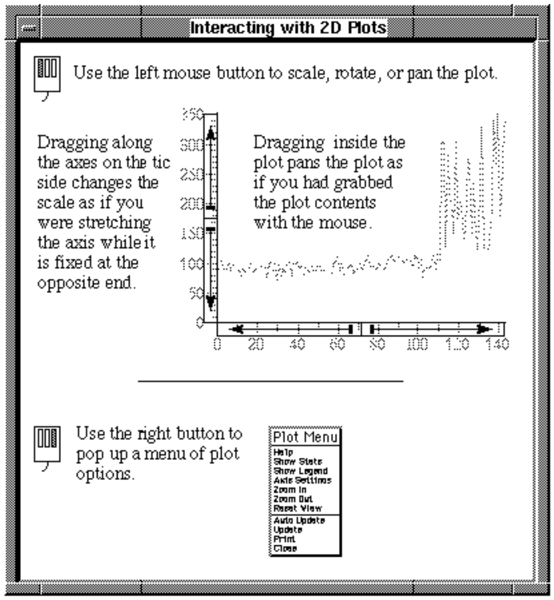


Figure 27. Interacting with XY Plots

Once you have scaled a plot, it is possible to grab a point inside the plot and pan the plot, moving it around inside the axes.

Histo-Scope allows XY plots to be overlaid by other one-dimensional plots such as 1-D histograms, Time Series Plots, and other XY plots. Up to ten plots can be overlaid in one window. To insert a plot on top of an XY plot, use the middle mouse button to drag another plot onto the XY plot window (To drag the plot, press the middle mouse button while the pointer is over the plot area of the desired plot, and, while holding the mouse button down, move the mouse pointer over to the receiving XY plot, and then release the middle mouse button. When you press the middle mouse button, the pointer changes to indicate that your are initiating a drag.) If the drag succeeds, you will see both plots overlaid in the receiving window (the source plot remains unchanged). If the drag does not succeed, you will see the drag icon move back to the source plot when you release the middle mouse button.

You can use the right mouse button on a selected plot window to pop-up a menu which allows the following functions:

	(Accelerator Key):
<u>H</u>elp	
Show S<u>t</u>atistics	*(Ctrl T)*
Set <u>M</u>ark & Line Style...	*(Ctrl M)*
Show Legend	*(Ctrl G)*

Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
Close	*(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

XY Scatter Plot

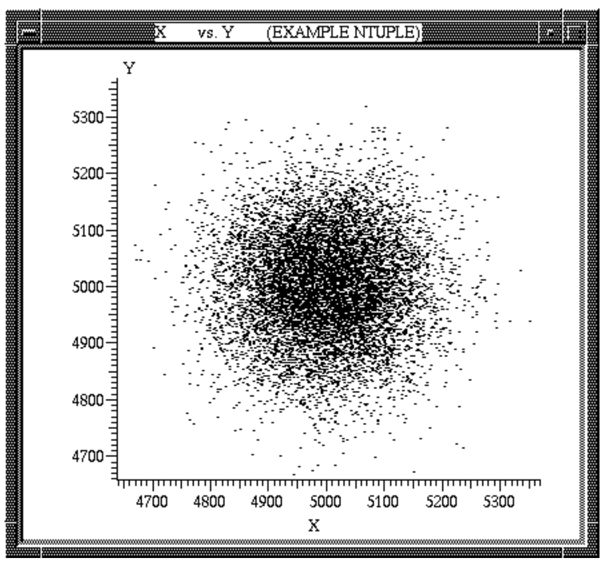


Figure 28. XY Scatter Plot

An XY Scatter Plot will plot two variables against each other using a dot for each plotted point. This kind of plot is useful when there are enough data points to be seen and their values form some visual pattern. A plot adjustment for the Scatter Plot can make the pixel-sized dots “darker” or “lighter” to optimize the data display. (For sparse data that may not show well as a Scatter Plot, try using an XY Plot with markers and turn off the connecting lines.)

In addition to the variables plotted along the x- and y-axis, up to three additional variables can be selected to adjust the plot as animation slider variables.

Initially, the entire range (excluding overflows) of the plot is displayed. You can change the size of the plot by changing the size of its window (by dragging on the window frame). You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use an axis point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

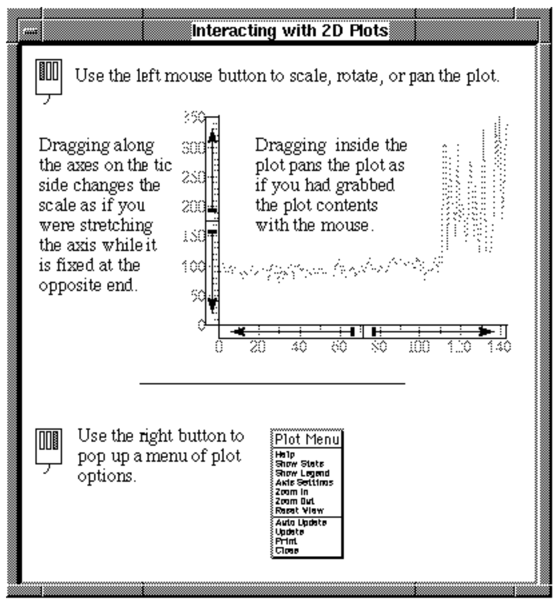


Figure 29. Interacting with 2D Scatter Plots

Once you have scaled a plot, it is possible to grab a point inside the plot and pan the plot, moving it around inside the axes.

You can use the right mouse button on a selected plot window to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders	*(Ctrl L)*
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*

Thicken Points
Automatic Update
Update *(Ctrl U)*
Grow Only *(Ctrl O)*
Print... *(Ctrl P)*
Generate PostScript... *(Ctrl F)*
Change Window Title *(Ctrl E)*
Close *(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

XYZ Scatter Plot

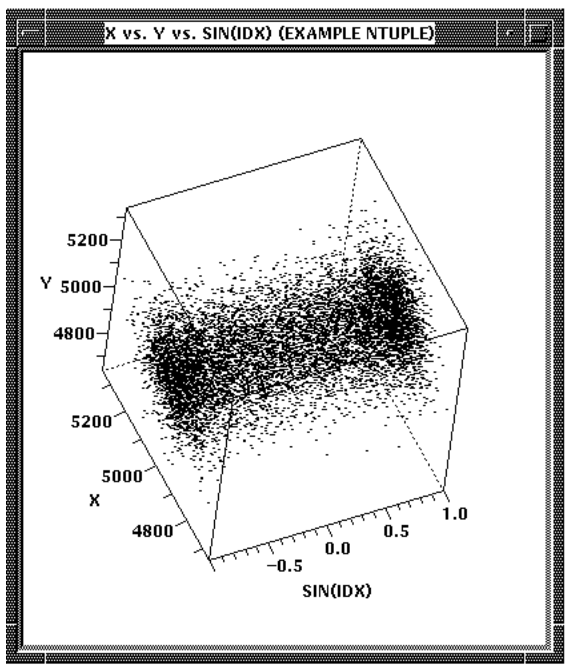


Figure 30. XYZ Scatter Plot

An XYZ Scatter Plot plots three variables against each other using a dot for each plotted point. This kind of plot is useful when there are enough data points to be seen and their values form some visual pattern. Plotted dots can be thickened or lightened by selecting *Thicken Points* from the plot settings menu. In addition to the variables plotted along the x- and y-axis, additional variables can be selected for animation sliders.

Initially, the entire range of the Ntuple data is displayed. You can change the size of the plot by changing the size of its window. Just drag on the window frame to make the window the size you desire.

To rotate the plot, use the mouse to drag inside of the boundaries of the cube frame. Dragging outside of the cube frame rotates the plot about an axis perpendicular to the plane of the window.

You can change the scale of the plot by dragging on the axis tics (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use a point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

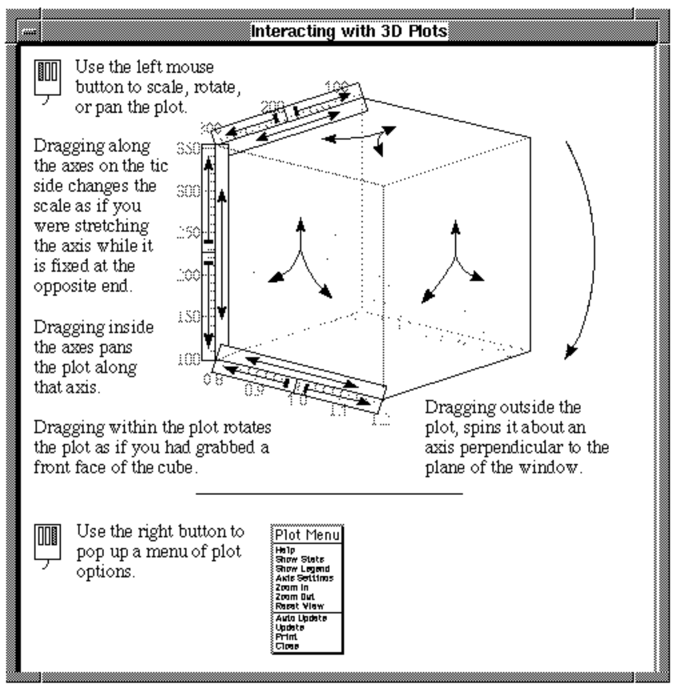


Figure 31. Interacting with 3D Scatter Plots

Once you have scaled the plot, it is possible to grab points inside the axis lines and pan the plot, moving it around inside the axes.

Using the Ctrl key along with the mouse button for scaling, panning, and rotation functions will provide a continuous display of the XYZ Scatter plot. Without using the Ctrl key, only the axes and labels are displayed while you are adjusting the plot. (This is the default mode to accomodate slow displays).

You can use the right mouse button on a selected plot window to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders	*(Ctrl L)*
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Thicken Points	
Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
Close	*(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

Adaptive Histogram

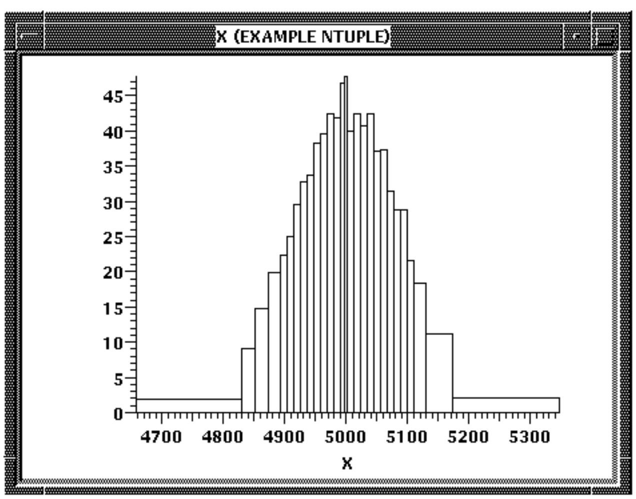


Figure 32. Adaptive Histogram

A 1-dimensional histogram is a plot of one variable over a pre-specified range divided into a number of bins. An adaptive histogram is a histogram whose bins, rather than being spaced evenly, are partitioned according to the density of the data. A threshold value determines the number of fills at which a single bin is divided into additional bins. Thus, the resolution of an adaptive histogram is highest where the data is densest, and lowest where the data is sparse.

Histo-Scope provides two adaptive histogram binning strategies: *Split in Half*, and *Center of Gravity*. The split in half strategy simply divides the bins into two equal parts when the threshold number of fills is reached. The center of gravity strategy starts with the whole data set and begins splitting the bins at their median value until the bins all contain less than the threshold number of fills. (Because there is also a limit on the number of splits (20), bins can actually have more than the threshold number of fills, but these bins will already be imperceptibly narrow except at very high magnifications.)

Initially, the entire range (excluding overflows) of a histogram is displayed. You can change the size of the plot by changing the size of its window. Just drag on the window frame to make the window the size you desire.

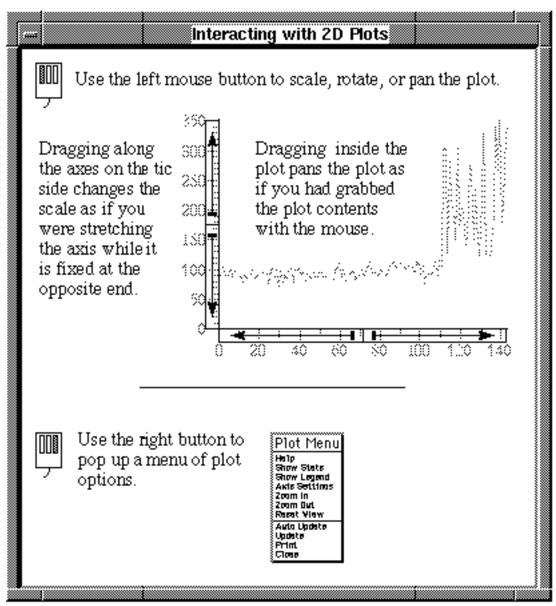


Figure 33. Interacting with Adaptive Histogram Plots

You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use a point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

Once you have scaled the plot, it is possible to grab a point inside the plot and pan the plot, moving it around inside the axes.

You can also use the right mouse button to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders	*(Ctrl L)*
Show Bin Limit Sliders	*(Ctrl R)*
Show Rebin Slider[1]	*(Ctrl B)*
Set Histogram Style...	*(Ctrl M)*
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Split in Half Strategy	
Center of Grav. Strategy	
Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
Close	*(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

Adaptive Two-Dimensional Histogram

A 2-dimensional histogram is a plot of two variables, where each axis is divided into a number of bins. An adaptive histogram is a histogram whose bins, rather than being spaced evenly, are partitioned according to the density of the data. A threshold value determines the number of fills at which a single bin is divided into additional bins. Thus, the resolution of an adaptive histogram is highest where the data is densest, and lowest where the data is sparse.

---

[1] Show Sliders and Show Rebin Slider plot settings are available only for histograms displayed from an n-tuple.

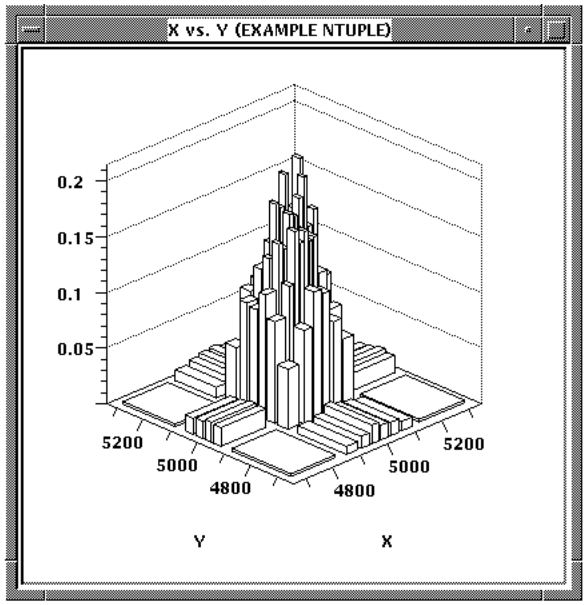


Figure 34. Adaptive 2-D Histogram

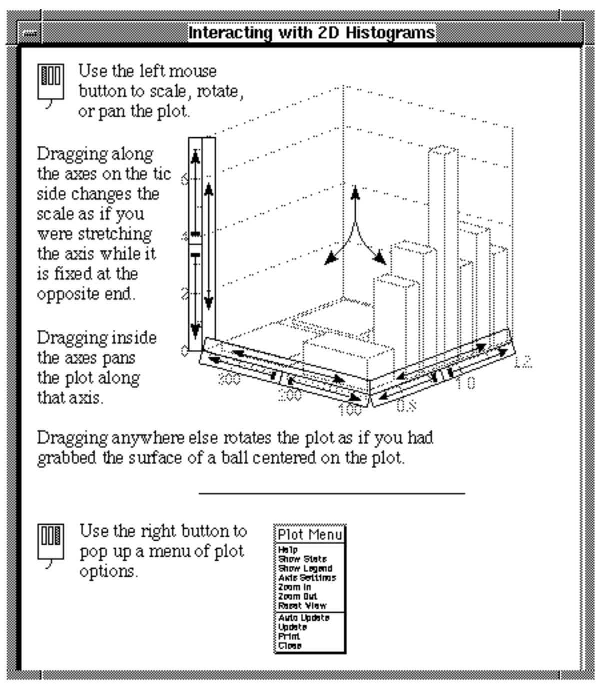


Figure 35. Interacting with 2D Histograms

Histo-Scope provides two adaptive histogram binning strategies: *Split in Half*, and *Center of Gravity*. The split in half strategy simply divides the bins into two equal parts when the threshold number of fills is reached. The center of gravity strategy starts with the whole data set and begins splitting the bins at their median value until the bins all contain less than the threshold number of fills. (Because there is also a limit on the number of splits (20), bins can actually have more than the threshold number of fills, but these bins will already be imperceptibly narrow except at very high magnifications)

Initially, the entire range of the data is displayed. You can change the size of the plot by changing the size of its window. Just drag on the window frame to make the window the size you desire.

You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use a point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

Once you have scaled the plot on an axis, it is possible to grab a point just inside of the axis and pan the plot so that it moves along the axis.

The orientation of the histogram can be changed by grabbing a part of the display (away from an axis) with the mouse and moving the display in space to a different orientation.

You can also use the <u>right</u> mouse button to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show S<u>t</u>atistics	*(Ctrl T)*
Show S<u>l</u>iders	*(Ctrl L)*
Show Bin Limit Sliders	*(Ctrl R)*
<u>A</u>xis Settings...	*(Ctrl A)*
Zoom <u>I</u>n	*(Ctrl Z)*
Zoom <u>O</u>ut	*(Shift Ctrl Z)*
<u>R</u>eset View	*(Ctrl R)*
S<u>p</u>lit in Half Strategy	
Center of <u>G</u>rav. Strategy	
Show <u>B</u>ackplanes	
Automatic Up<u>d</u>ate	
<u>U</u>pdate	*(Ctrl U)*
Gro<u>w</u> Only	*(Ctrl O)*
<u>P</u>rint...	*(Ctrl P)*
<u>G</u>enerate PostScript...	*(Ctrl F)*
Change Window Title	*(Ctrl E)*
<u>C</u>lose	*(Ctrl W)*

See the section **Adjusting Plot Settings** in this guide for a detailed explanation of each of these settings.

Two-Dimensional Cell Plot

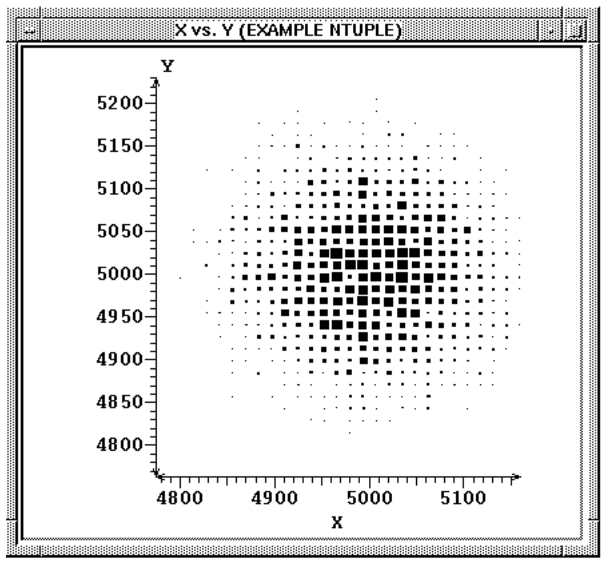


Figure 36. 2-D Cell Plot

A two-dimensional cell plot is a flat representation of a two-dimensional histogram, with bin contents represented by the size of a shaded rectangle drawn on the center of the bin. If the source of the data is an n-tuple, additional variables can be selected for animation sliders.

Initially, the entire range of the data is displayed. You can change the size of the plot by changing the size of its window. Just drag on the window frame to make the window the size you desire.

You can change the scale of the plot by using the mouse to drag a part of an axis (just outside of the plot) toward one direction or another. Use a point near the upper half of the axis to change the maximum axis value. Use a point near the lower half of the axis to change the minimum axis value. This will zoom in or out, depending on the direction you drag that axis point.

Once you have scaled the plot on an axis, it is possible to grab a point just inside of the axis and pan the plot so that it moves along the axis.

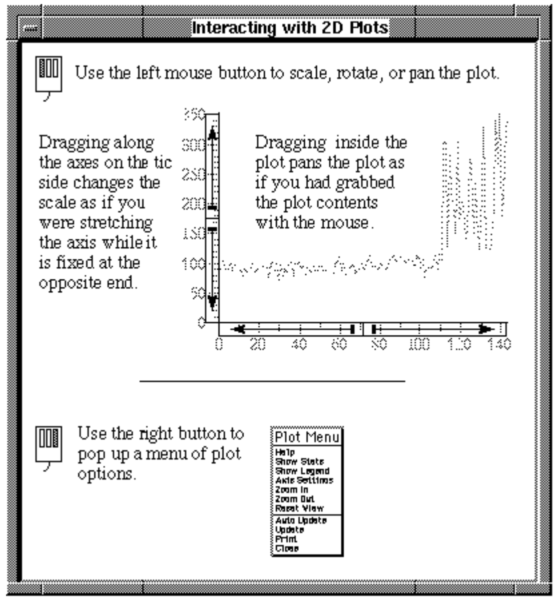


Figure 37. Interacting with 2D Cell Plots

You can also use the right mouse button to pop-up a menu which allows the following functions:

	(Accelerator Key):
Help	
Show Statistics	*(Ctrl T)*
Show Sliders	*(Ctrl L)*
Show Rebin Sliders	*(Ctrl R)*
Show Normalization Slider	*(Ctrl N)*
Other Views	
Axis Settings...	*(Ctrl A)*
Zoom In	*(Ctrl Z)*
Zoom Out	*(Shift Ctrl Z)*
Reset View	*(Ctrl R)*
Automatic Update	
Update	*(Ctrl U)*
Grow Only	*(Ctrl O)*
Print...	*(Ctrl P)*
Generate PostScript...	*(Ctrl F)*
Close	*(Ctrl W)*

A Normalization Slider can be selected from the Plot Settings menu which will adjust the cell size in relation to the bin value of the cell. See the section **Adjusting Plot Settings** in this guide for a detailed explanation of the Normalization Slider and each of the other plot settings.

Overlay Plots

An overlay plot is the result of using the "View Overlaid" button in the main panel, viewing a plot group whose group type is HS_OVERLAY, or dragging one plot over another. Some of the features and options offered for the original plots are not available in the overlayed state. To access these features, you must remove the plot from the overlay (use "Remove Overlayed Plot"), re-create it individually, and merge it back in to the overlay. Sliders, including rebin sliders, in an overlayed plot, refer to the original plot, to which others plots have been dragged.

Interactive sliders and rebinning can still be used on the original(bottom) plot in the overlay. The menu items: Show Statistics, Show Sliders, and Show Rebin Slider, all refer to this original plot.

Adjusting Plot Settings

Each plot window described in the previous sections allows you to pop up a menu to change the default settings used to display the plot. In addition, the plot settings menu allows printing, writing a PostScript file of the plot, and closing the plot. To bring up the plot settings menu, make sure the plot is selected (using the first or select mouse button) and then press the right-most mouse button. The plot settings menu is used just like the other pull-down menus throughout Histo-Scope, except that you are using the right-most button instead of the left mouse button for selecting items in the menu.

Getting Help

The plot settings menu for each plot includes on-line help that describes each plot setting.

Getting Statistics on the Data

You can get statistics on the data used in each plot type. Each statistics window includes the name of the data item, the number of fills (for histograms) or entries (for Ntuples), the Sum of all the Fills (for histograms only), the mean, standard deviation, minimum, and maximum for each variable, and overflow sums (histograms only).

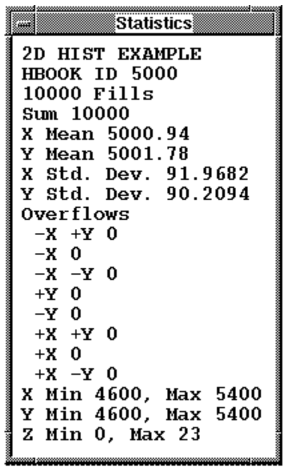


Figure 38a.
Example Statistics Window

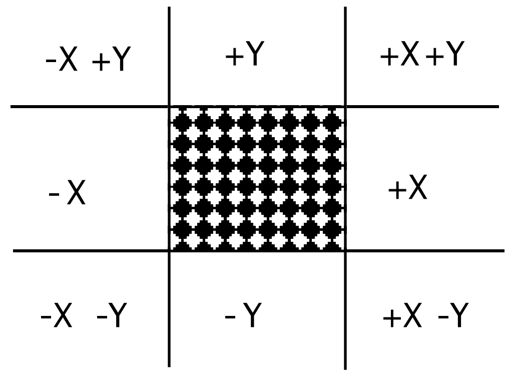


Figure 38b.
2D Histogram Overflows

One-dimensional histograms have only two overflows: -X and +X.

If the statistics window for a plot is displayed at the same time the user requests a plot to be printed, the statistics are printed to the side of the plot (except for Multiple Plot Windows).

Setting Styles and Colors for Markers, Lines, and Histogram Fill Patterns

Time Series, XY, and Histogram plots allow you to place markers on the data points of the plots. Time Series and XY plots allow you to set line styles for each variable in the plot. Histogram plots allow you to set the fill style pattern and color for each histogram.

If you are using a color display, the color of these markers, lines, or fill paterns can also be set.

When you select *Set Histogram Style...* from the plot setting menu the following window pop up:

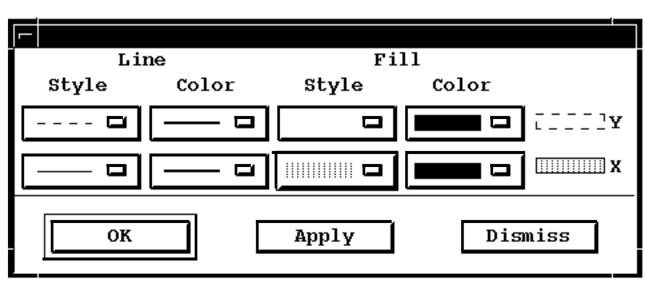


Figure 39a. Histograms Style Window

When you select *Set Mark & Line Style...* from the plot setting menu the following window pop up:

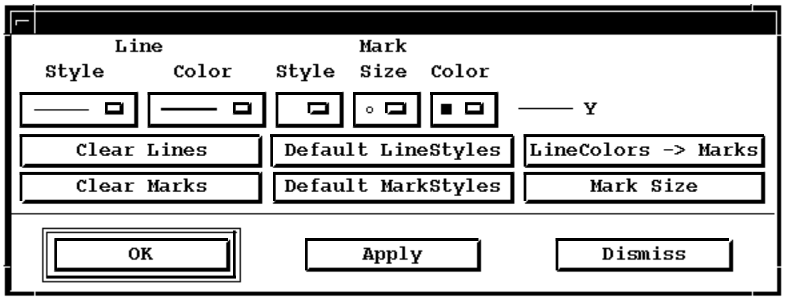


Figure 39b. Mark & Line Styles Window

The Histograms style window contains four pull-down menus; the Mark & Line Styles window contains five. The first two menu boxes are for lines, style and color. When you click on the first menu box, all possible line styles are displayed. Click and drag the mouse to select the line style you desire. As you select a new style, the line to the right of the option menu boxes changes accordingly so you can see what your line will look like. In the Histograms window, at the bottom of the lines styles options, there will be markers displayed. If selected the histogram will be plotted with markers with or without error bars depending on whether errors were previously selected. The second menu box allows you to select a color for the line. When markers are selected as the line style, markers are displayed without connecting lines or fill pattern.

In the Histograms Styles window, the next two menu boxes are for histogram fill style and color. They work the same way as the line syle and color menu boxes, Click and drag the mouse to select the histogram fill style and color you desire.

In the Mark & Line Styles Window, the second group of menu boxes allow you to select the marker characteristics: type, size and color. These menu boxes work in the usual way as well. The variable names are listed at the top right of the window in both Histograms and Mark & Line Styles Windows.

The six push buttons in the middle of the Mark & Lines Styles Window allow you to clear the marker type and line style or set the styles to a default. In order for these changes to take effect, press the Apply or OK button. The Mark Size button pops up a window which allows you to set all markers to the same size. The LineColors button pops up a window which allows you to change the marker colors.

To see how the plot will look with the assignments you've made, click on the Apply button. The OK button will make the assignments and dismiss the window. The Dismiss button will cancel your selection and dismiss the window.

Showing/Hiding the Variables Legend

By default, the variables legend in Time Series and XY plots is displayed. To make the legend disappear, select *Show Variables Legend* from the Plot Settings menu. To make it reappear, select *Show Variables Legend* once again.

Sliders: Observing the Effect of Other Ntuple Variables on the Plot

Slider variables can be defined for many kinds of plots from Ntuple data. Slider variables are used to study correlations among variables in the Ntuple. They allow the user to make simple cuts in the Ntuple sample. From the Ntuple Window, assign these variables to the positions S1, S2, S3... Then select *Show Sliders* from the Plot Settings menu. A window similar to Figure 40 will pop up.

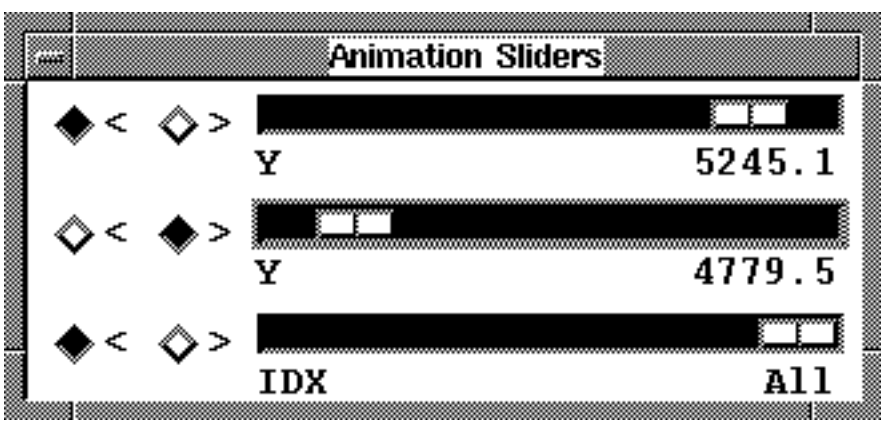


Figure 40. Animation Sliders Window
(shown larger than actual size)

By default, each slider is set up so that all points are included and the radio button shows that all values less than (<) the value displayed are included in the plot. As you move the slider to the left, entries will be eliminated that are associated in the Ntuple with

values greater than the displayed value of the slider variable. As you move the sliders, scatter plots lose/add points and histograms lose/add height.
When the radio button for greater than (>) is selected, the opposite is true: all values greater than the value displayed are included in the plot and values less than are excluded. Thus it is possible by selecting the same Ntuple variable twice to achieve a range of values for the cut. For example in Figure 40, all points are displayed where the variable Y is between 4779.5 and 5245.1.

In addition to dragging the slider with the mouse, you can click on the space to the left or right of the slider to move it in pre-defined increments. For smaller increment adjustments you can use the left and right arrow keys. Typing a tab will allow you to select which slider is affected when using the arrow keys.

Rebinning Ntuple Data Displayed as a Histogram

To re-bin histograms displayed from Ntuple data, or to find out the number of bins Histo-Scope used to display the Ntuple, select *Show Rebin Sliders* from the Plot Settings menu. A window similar to Figure 41 will pop up.

To adjust the slider to the exact number of bins you desire, use the left and right arrow keys. For two-dimensional histograms with two sliders, you may need to tab to the slider you want to change. You can also drag the slider with the mouse or click on the space to the left or right of the slider for larger adjustments.

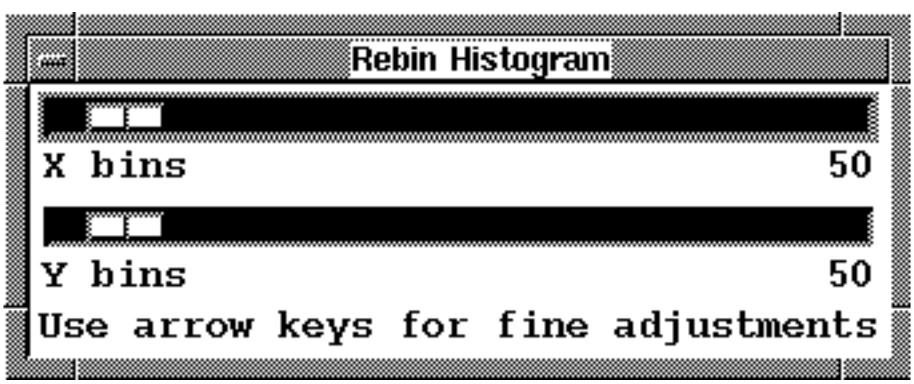


Figure 41. Rebin Histogram Window
(shown larger than actual size)

To set the bin splitting threshold for adaptive histograms displayed from Ntuple data, or to find out its current value, select *Show Bin*

Limit Sliders from the Plot Settings menu. A window similar to the following will pop up:

Figure 42. Bin Limits Window
(shown larger than actual size)

To adjust the slider to a new threshold value, use the left and right arrow keys. (Use the tab key to select the slider you want to change for two-dimensional histograms if the slider you want to change isn't selected.) You can also drag the slider with the mouse or click on the space to the left or right of the slider.

Labeling of Histogram Bin Edges

Histogram axes can optionally be numbered using the exact values of the bin edges rather than the ordinary plot scale. To change the labeling style, select *Label at Bin Edges* from the Plot Settings menu. When the bins are very narrow, the plot reverts to the ordinary axis labeling scheme regardless of the setting of this option.

Altering Cell Size for Data Displayed as a Cell Plot

To alter the cell size and density of data displayed from a Cell Plot, select *Show Normalization Slider* from the Plot Settings menu. A window similar to Figure 43 will pop up.

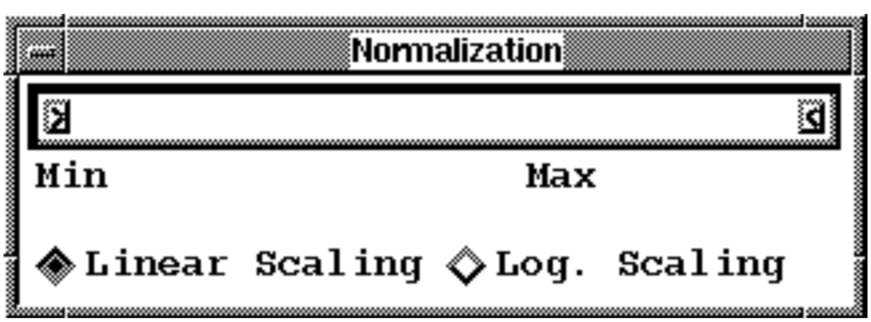


Figure 43. Normalization Slider

The size of a rectangle in a cell plot is proportional to the contents of the bin it represents. The maximum cell size (filled in black) represents a value greater than or equal to the right (maximum value on the Normalization Slider window. An empty cell represents a bin value less than or equal to the left (minimum) value on the Normalization Slider window.

Initially, the normalization slider shows the entire range of the bin values and therefore displays the entire range of the data. The number displayed in the right side of the slider instructs Histo-Scope to display everything from that value up to the maximum value to be displayed at the maximum cell size. (Similarly, the number displayed in the left side of the slider instructs Histo-Scope to eliminate displaying everything from that value to the minimum value on the plot.) The data between the two sliders is displayed with cell size proportional (linear scaling) to the bin contents within the range of the normalization minimum and maximum. If the toggle-button is set to Log. Scaling, the cell size is logarithmically proportional to the bin contents within the range.

To adjust the slider drag the arrow slide you want to change with the mouse in the direction you desire.

Other Views

The *Other Views* plot setting is available from two-dimensional histogram plots and cell plots requested as another view of two-dimensional plots. When selecting *Other Views* from a two-dimensional histogram, a 2D Cell Plot is popped up (and vice versa).

Show Titles

By default, each plot title in a Multiple Plot window is displayed. In order to use the entire area for the plots, the user can elect to make these titles disappear. To do so, select *Show Titles* from the Plot Settings menu. To make the titles reappear, select *Show Titles* again.

Scaling and Panning the Plots

There are three different ways to scale a plot after it is displayed:

- Use the mouse directly on the plot to grab the axis ticks and pull (or stretch) the axis. Point the mouse just to the outside of the axis using a point near the upper half of the axis to change the maximum axis value and a point near the lower half of the axis to change the minimum axis value.

- Use: *Zoom In* *(Ctrl Z)*
  Zoom Out *(Shift Ctrl Z)*
  Reset View *(Ctrl R)*

  from the Plot Settings menu. *Zoom In* and *Zoom Out* will immediately change the plot. *Reset View* will restore the plot to its initial orientation and scale with all its data displayed.

- Use the Axis Settings window by selecting *Axis Settings...* from the Plot Settings menu. This menu allows you to enter values for the minimum and maximum value of each axis. In addition, you can change the scaling of an axis from linear to logarithmic (and back) using the diamond-shaped radio buttons.

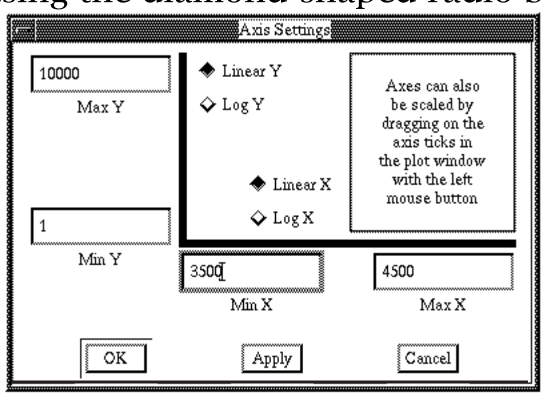


Figure 44. Axis Settings Window

All plots can be panned by grabbing a point inside the plot and dragging with the mouse. On 2D Histograms, you must grab a point near the axes and drag along the axis.

Showing/Hiding Backplanes for 2D Histograms

By default, the backplanes of 2D Histograms are displayed. To make the backplanes disappear, select *Show Backplanes* from the Plot

Settings menu. To make them reappear, select *Show Backplanes* once again.

Showing/Hiding Errors for Histograms

By default, error data in a histogram is not displayed. If error data is available for the plot, selecting *Error Data* from the Plot Settings menu will show error bars using the error data from the file or process. Selecting *Error Data* once again, (or selecting *Gaussian Errors*) from the Plot Settings menu will de-select Error Data.

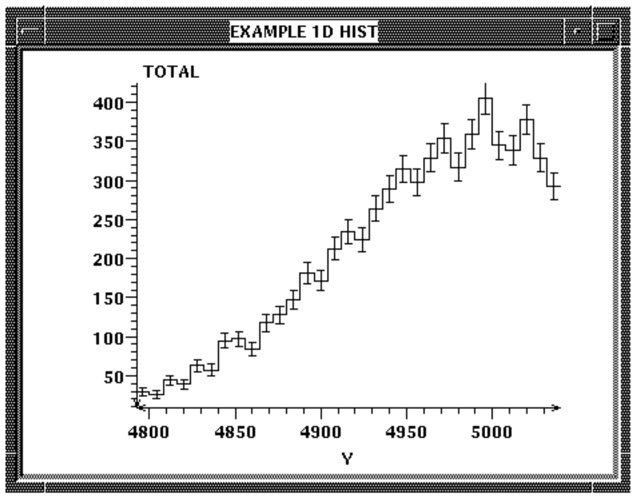


Figure 45. Histogram with Gaussian Errors

Selecting *Gaussian Errors* from the Plot Settings menu will create error bars for the histogram showing the square root of the bin contents. These errors are computed by Histo-Scope and thus do not incur communications overhead when connected to a process.

Thickening/Lightening Scatter Plot Points

By default, the points in a scatter plot are drawn as thick points. To make the plot lighter, select *Thicken Points* from the Plot Settings menu to lighten the points. To make the points thick again, select *Thicken Points* once again.

Update Options for all Plots

Automatic Update, *Update,* and *Grow Only* from the Plot Settings menu are useful only when connected to a running process. These three update settings affect how the data is re-displayed as it changes in the client program. (The Main Window's Preferences menu is used to set the *Update Frequency,* which affects how often data is requested from the running process.)

Initially *Automatic Update* mode is on. This is indicated by the presence of a square toggle button to the left of the words "Automatic Update". Selecting the *Automatic Update* item in the plot settings menu turns on or off *Automatic Update* mode.

When Histo-Scope is connected to a running process, each displayed plot with *Automatic Update* mode *on* will be updated according to the frequency set in the Preferences Menu in the Main Window (initially every 3 seconds).

With *Automatic Update* off, a plot can still be individually updated with data Histo-Scope has received from the client by selecting *Update* from the plot settings menu.

When new data arrives, the plot may be re-scaled to include the new data. Setting *Grow Only* mode will prevent Histo-Scope from reducing the scale of the plot if the new data covers less area than the old data. Grow Only mode is useful for maintaining comparable scales between several plots, particularly in conjunction with using the *Axis Settings* dialog to set the plot limits beyond the limits of the data. When using sliders, the plot is temporarily set in *Grow Only* mode while the sliders are affecting the display of the data.

Printing a Plot

Select *Print...* from the Plot Settings menu to print the plot. Histo-Scope will pop up a Printer Options window for specifying the printer queue name:

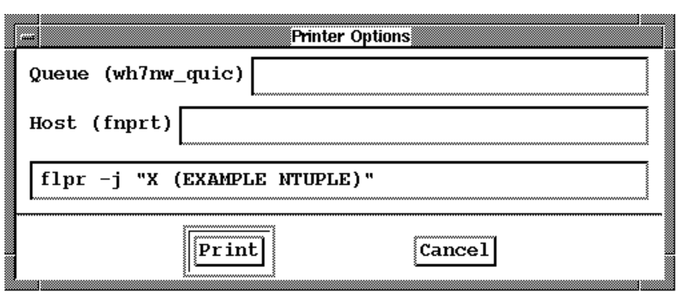


Figure 46. Printer Options Window

The default queue names initially displayed are the environment variables defined, if any, for lpr and flpr. If no such defaults are available, the menu displays the system default queue.

The last field in this dialog is the Unix command that will be used to queue the plot for printing. Since this command is generated from the other options in the dialog, there is normally no reason to change it, but you can use the field to add options and change the command in ways not otherwise supported by the print dialog.

Enter the queue name if the default queue is not the printer you would like to use. Press *Print* to print the file.

Pressing *Cancel* will make the Printer Options window disappear without printing anything.

If the statistics window for a plot is displayed while printing a plot, the statistics will also be printed to the side of the plot.

If the plot window to be printed is a Multiple Plot window, all mini-plots in the window will appear on a single page similar in appearance to how they appear on the screen. If titles are displayed in the Multi-Plot window, they will also be printed. For Multi-Plot windows, statistics are not printed. If you find that the Multiple Plot window is larger than the size paper you are using to print the window, try reducing the size of the Multiple Plot window and re-printing or use a POSTSCRIPT printer that can accommodate larger paper.

Please note that 2D Histogram plots can take some time to print because of PostScript processing time in the printer. As the number of bins goes up, print processing time increases so that histograms of 100x100 bins can take over nine minutes for the printer to process and print the file (depending on the speed of the printer).

Generating a PostScript File of a Plot

Select *Generate PostScript...* if you would like to save a PostScript representation of your plot to a file. Histo-Scope will pop-up a file selection dialog for you to enter a filename for the PostScript output. Histo-Scope produces encapsulated PostScript output so that this file can be pasted into other documents.

Showing/Hiding the Range for Indicators and Controls

By default, a slider showing the minimum and maximum values defined for an indicator or control are displayed. To omit these from the window, select *Show Range* from the plot settings menu. The filled square to the left of the *Show Range* will disappear, indicating that showing the range is disabled. To enable *Show Range*, just select it once again.

Deleting a MiniPlot

Select *Delete MiniPlot* from the Plot Settings menu to delete the mini-plot from the Multiple Plot window. The remainder of the Multi-Plot window is unaffected. After a mini-plot is deleted, it is possible to drag another plot into that empty “cell” so that it is displayed in the Multi-Plot window.

Changing the Window Title

Select *Change Window Title* from the Plot Settings menu if you would like to change the title of the plot window A new window will pop up which will allow you to enter a new window title. After you have entered a desired new title, click on the OK button. If you wish to cancel this selection, there is a Cancel button to click on.

Removing an Overlayed Plot

Select *Remove Overlayed Plot* from the Plot Settings menu to remove a plot which you have overlayed on another plot in the current plot window After you select Remove Overlayed Plot..., there will be a list of plots that are currently overlayed in that window. You may then drag the mouse to select the plot you wish to remove.

Closing a Plot

Select *Close* from the Plot Settings menu to close the plot window and make it disappear. It is a good idea to close plots that are no longer needed when connected to a process to avoid the CPU overhead of transferring and displaying data.

Part II

HSDIR Directory Tool

hsdir is a directory tool which will display a directory of the contents of a Histo-Scope formatted file, including uid, category, title, and data type. Simply type the command **hsdir** followed by a filename specification. For example:

% hsdir random.hs

Directory of Histo-Scope format file: random.hs

uid	category	title	type
---	--------	-----	----
21	HS	INDICATOR EXAMPLE	Indic
22	HS	2D HIST EXAMPLE	2dHist
23	HS	EXAMPLE 1D HIST	1dHist
24	HS	EXAMPLE NTUPLE	Ntuple

Total of 4 items listed in file

%

The filename may have a wildcard and if there is more than one file meeting the criteria of the wildcard, the directories for all these files will be listed.

tohbook Tool

tohbook is a tool which can be used to convert a Histo-Scope-format file to a HBOOK-format file. Simply type the command **tohbook** followed by a filename specification. For example:

% tohbook random.hs

In the above example, tohbook will produce a HBOOK-format file called random.hbook of all of the histograms and ntuples in random.hs.

Part III - Histo-Scope Library Subroutines

The second part of Histo-Scope is a library of routines which make up the application program interface to Histo-Scope. Calls to these routines are inserted in the physics analysis or data acquisition code so that it can be "scoped" by the Histo-Scope process. No restrictions are placed on the user analysis process except that it must call a routine to initialize the Histo-Scope application interface and then periodically call an update routine. The user application does not need to run on a workstation and is not linked with any graphical user interface code.

At the start of the analysis or data acquisition code, a call to the Histo-Scope library declares this task to be a potential customer of the scope tool, much as an engineer might provide a plug for his oscilloscope or logic analyzer on his circuitry. In the event loop, or wherever it would be useful to see changes in the data, the program calls the Histo-Scope library update routine. Only when a scope is connected and a user is watching the evolution of a particular histogram or indicator, will the update routine communicate changes in the data to the scope. Otherwise the update routine just returns. Even while connected to a Histo-Scope process, the Histo-Scope update routine also returns immediately if the update frequency period set by the Histo-Scope user has not yet elapsed (initially set at three seconds, but can be changed by the Histo-Scope user). Thus, a programmer can feel free to call the update routine as often as is convenient without concern for unnecessary impact to CPU usage. Just before exiting, the program calls a completion routine to notify anyone watching that the program is finished and optionally allows the user(s) to finish browsing data before the program exits.

Histo-Scope can display HBOOK histograms and Ntuples, and Histo-Scope's own type of histograms, Ntuples, indicators, controls, and triggers. The data-collection routines in the Histo-Scope library offer some performance improvement and are simpler than those in HBOOK. <u>Both types of data can be freely intermixed</u> in a program with no danger of interference between them.

If using HBOOK data exclusively, only four calls to Histo-Scope need to be added to your program:

HS_INITIALIZE
HS_HBOOK_SETUP
HS_UPDATE
HS_COMPLETE

Histo-Scope provides both FORTRAN and C include files that define the Histo-Scope routines for the convenience of the programmer. The FORTRAN include file, **histoscope.inc**, declares the function subprograms as externals and defines their data type. The C include file, **histoscope.h** defines the function prototypes.

Two FORTRAN libraries are supplied for linking with the Histo-Scope routines:

libFHisto	for programs that do **not** use HBOOK
libFHistoHB	for programs that **do** use HBOOK

libFHisto does not require HBOOK to be installed on the system.

Similarly, two C libraries are supplied for C language programs: libCHisto and libCHistoHB.

Histo-Scope Data Routines

The Histo-Scope libraries discussed above also include a robust set of data creation, collection, calculation, and manipulation routines that can be used in addition to or instead of HBOOK. These routines were designed to be simple to use and efficient in execution. Complete routine descriptions can be found following the Example Programs. This section presents an overview of the Histo-Scope library capabilities.

Histo-Scope's data types are:

- 1-d histograms
- 2-d histograms
- Ntuples
- indicators
- controls
- triggers

A <u>one dimensional histogram</u> stores a frequency distribution of one variable over a pre-specified range: the range is divided into a specified number of bins, and Histo-Scope keeps a weighted frequency count per bin. This data is then available for display as a histogram in Histo-Scope or saved to a file for later program access or display by Histo-Scope. The pertinent calls for one-dimensional histograms are:

hs_create_1d_hist
hs_fill_1d_hist
hs_set_1d_errors
hs_1d_hist_block_fill
hs_1d_hist_num_bins
hs_1d_hist_range
hs_1d_hist_bin_contents
hs_1d_hist_errors
hs_1d_hist_overflows
hs_1d_hist_x_value
hs_1d_hist_bin_value
hs_1d_hist_minimum
hs_1d_hist_maximum
hs_1d_hist_stats
hs_1d_hist_derivative
hs_hist_integral
hs_hist_set_gauss_errors
hs_sum_histograms
hs_multiply_histograms
hs_divide_histograms
hs_sum_category
hs_sum_file

A <u>two dimensional histogram</u> stores a weighted frequency distribution of two variables, where each variable has its own pre-specified range and number of bins. This data is then available for display as a two-dimensional histogram in Histo-Scope or saved to a file for later program access or display by Histo-Scope. The pertinent calls for two-dimensional histograms are:

hs_create_2d_hist
hs_fill_2d_hist
hs_set_2d_errors
hs_2d_hist_block_fill
hs_2d_hist_num_bins
hs_2d_hist_range
hs_2d_hist_bin_contents
hs_2d_hist_errors
hs_2d_hist_overflows
hs_2d_hist_xy_value
hs_2d_hist_bin_value
hs_2d_hist_minimum
hs_2d_hist_maximum
hs_2d_hist_stats

hs_hist_integral
hs_hist_set_gauss_errors
hs_sum_histograms
hs_multiply_histograms
hs_divide_histograms
hs_sum_category
hs_sum_file

An Ntuple is a two-dimensional array composed of a fixed number of named variables times a growing number of occurrences. The Histo-Scope tool allows the user to specify which variables in the Ntuple to plot and the type of plot to display. Histo-Scope currently supports only in-memory Ntuples of real numbers; i.e. Ntuples must fit in virtual memory in order to be stored and displayed by Histo-Scope. (Always keep in mind the performance limitations of your machine when using very large Ntuples.) Pertinent calls for Ntuples are:

hs_create_ntuple
hs_fill_ntuple
hs_num_entries
hs_num_variables
hs_variable_name
hs_variable_index
hs_ntuple_value
hs_ntuple_contents
hs_row_contents
hs_column_contents
hs_merge_entries
hs_sum_category
hs_sum_file

The combination of indicators, controls and triggers allows the user to control the execution of his data acquisition or analysis program from Histo-Scope. For instance, should peculiar results come out of diagnostic histograms, certain code paths can be internally enabled or disabled, or the job gracefully terminated.

An indicator is a scalar value set by a process to indicate the value of a variable or condition. Indicators are set by the application program and viewed by the user of the Histo-Scope tool. The pertinent calls for indicators are:

hs_create_indicator
hs_set_indicator

Controls allow a variable in a process to be set interactively using Histo-Scope. The application creates the control and supplies a default value. A user of the Histo-Scope tool can then view the control and change its value. It is up to the application program to read the control and detect a new value. The pertinent calls for controls are:

hs_create_control
hs_read_control

A trigger is created by the application program and acts like a push-button the Histo-Scope user can set which, when detected by the application program, can cause the it to perform some action. Histo-Scope remembers whenever a trigger is set and will buffer trigger presses. Trigger presses are detected by calling hs_check_trigger. Pertinent calls for triggers are:

hs_create_trigger
hs_check_trigger

Groups are a pseudo-data type that allow an application to specify a grouping of already-created data items. Once data items are specified as a group, they can be displayed all at once from the Histo-Scope Main Panel rather than asking for each plot separately. Groups can be displayed as a Multiple Plot Window, an Overlaid Plot, or simply used as a convenience for putting plots up individually with fewer mouse clicks. One of these display types are specified as a default when creating the group:

hs_create_group

In addition to routines specific to a particular data type, the following routines apply to each data type above:

hs_reset	-	Resets the item. For histograms, all of the bins and overflows are set to 0. For Ntuples all of the data is removed. Indicators are set to *not set*. Controls are set to their default value. For triggers, any pending trigger presses are cleared.
hs_delete	-	Deletes the specified item.

hs_delete_items	-	Deletes a list of items.
hs_delete_category	-	Deletes all items in the specified category.
hs_save_file	-	Saves all current histograms, Ntuples, indicators, and controls to a Histo-Scope-format file. Triggers are not saved to files.
hs_save_file_items	-	Saves the specified items to a Histo-Scope-format file.
hs_read_file	-	Reads in the items from a Histo-Scope-format file. Allows a prefix to be added to the categories of all items read so that items from various runs can be differentiated.
hs_read_file_items	-	Selectively reads in items matching a specified category and list of uid's from a Histo-Scope-format file. Allows a prefix to be added to the categories of all items read so that items from various runs can be differentiated.
hs_change_uid	-	Changes the uid of the specified item.
hs_change_category	-	Changes the category of the specified item.
hs_change_title	-	Changes the title of the specified item.
hs_num_items	-	Returns the total number of Histo-Scope data items defined so far.
hs_id	-	Returns the histo-scope defined id from an item's uid and category pair.
hs_uid	-	Returns the user-defined id for a specified item.
hs_category	-	Returns the category string and its length for an item.

hs_title	-	Returns the title of an item.
hs_type	-	Returns the data type of an item.
hs_id_from_title	-	Returns the histo-scope defined id of an item from a specified title and category.
hs_list_items	-	Lists all of the item id's matching a specified category and title. Exact and inexact matches are allowed.

Building and Running the Example Programs on Unix

The example programs in the next section can be found on Unix systems in:

```
$HISTO_DIR/examples
```

A makefile for building the programs is also supplied. Copy the sources and the Makefile to your own directory to compile and link the programs.

To copy, compile, and run the example programs on Unix type:

```
% setup histo
% cp $HISTO_DIR/examples/* .
```

If you have the CERN software installed on your system type:

```
% setup cern
% make
```

Otherwise (if you do **not** have CERN installed):

```
% make hsExample
% make HsExample
```

If necessary, go to an X display and set the DISPLAY environment variable (refer to page 7):

```
% setenv DISPLAY devicename:0
```

Then run any of the example programs you'd like to see, for example:

% HbookExample

The example programs automatically run Histo-Scope as a sub-process. Removing the HS_HISTOSCOPE call will make the program run independently of Histo-Scope. When Histo-Scope is run as a sub-process, it is automatically connected to the running process. Refer to Part I on using the Histo-Scope tool.

Following the example programs is a description of each routine in the Histo-Scope library for FORTRAN and C.

Linking your own Histo-Scope application:

Once the setup histo command has been invoked, all user binary object libraries are accessible via the environment variable HISTOLIB_DIR. Which library to use depends on whether your program is written in FORTRAN or C, and whether or not you are using HBOOK. Use one of the following libraries:

libFHisto.a
libCHisto.a
libFHistoHB.a
libCHistoHB.a

For example, to link a FORTRAN user program that includes calls to the Histo-Scope and HBOOK libraries, use the library libFHistoHB.a.

FORTRAN EXAMPLE PROGRAMS

I. Using Histo-Scope with HBOOK:

```
      PROGRAM HBOOKEXAMPLE
C
C PROGRAM TO GENERATE RANDOM NTUPLE AND HISTOGRAM DATA
C
      IMPLICIT NONE
#include "histoscope.inc"
C
C    IF COMPILING PROGRAM ON VMS OR IBM/AIX SYSTEM, COMMENT OUT
C    ABOVE INCLUDE AND REMOVE THE C IN COLUMN 1 OF ONE OF THE FOLLOWING:
C
C    IBM/AIX:
C     INCLUDE 'histoscope.inc'
C    VMS:
C     INCLUDE 'HISTO_INC:histoscope.inc'
      INTEGER I, J, K, N_VARIABLES
      REAL VALUES(8)
      REAL RNDOM
      CHARACTER*8 TAGS(8)
      COMMON/PAWC/H
      REAL*4 H(1 000 000)

      N_VARIABLES = 8
          TAGS(1) = 'X'
          TAGS(2) = 'Y'
          TAGS(3) = 'IDX'
          TAGS(4) = 'SQRT(X)'
          TAGS(5) = 'SQRT(Y)'
          TAGS(6) = 'LOG(X)'
          TAGS(7) = 'LOG(Y)'
          TAGS(8) = 'SIN(IDX)'
      CALL HS_INITIALIZE('hbookExample.f')
      CALL HS_HISTOSCOPE(1)
      CALL HLIMIT(250 000)
           CALL HBOOKN(900, 'EXAMPLE NTUPLE', N_VARIABLES, ' ', 1000, TAGS)
           CALL HBOOK2(5000,'2D HIST EXAMPLE',100,4600.,5400., 100,4600.,
     &       5400.0, 0.)
           CALL HBOOK1(6000,'EXAMPLE 1D HIST',100,4600.,5400.,0.0)
      PRINT *, ' HLDIR CALLED: '
      CALL HLDIR('//PAWC', 'T')
      CALL HS_HBOOK_SETUP('//PAWC')
C
C          RANDOM DATA COMPUTATION - THIS COULD BE ANY CODE YOU WISH
```

```
C
      DO I = 1, 10000
          DO J = 1, 2
           VALUES(J) = 0
           DO K = 1, 1000
                VALUES(J) = VALUES(J) + RNDOM(0) * 10.
           ENDDO
          ENDDO

          VALUES(3) = I
           VALUES(4) = SQRT(VALUES(1))
           VALUES(5) = SQRT(VALUES(2))
           VALUES(6) = LOG(VALUES(1))
           VALUES(7) = LOG(VALUES(2))
           VALUES(8) = SIN(VALUES(3))

           CALL HFILL(5000, VALUES(1), VALUES(2), 1.)
           CALL HFILL(6000, VALUES(2), 0., 1.)
           CALL HFN(900, VALUES)
           CALL HS_UPDATE

          IF ( MOD(I, 200) .EQ. 0)  THEN
                   PRINT *, ' Reaching computation ...', I
             ENDIF
      END DO
      CALL HS_COMPLETE
      CALL HRPUT(0, 'hbookexample.hst', 'TN')
      STOP
      END

        REAL FUNCTION RNDOM()
        INTEGER A,M,Q,R,HI,LO,TEST
        DATA A, M, Q, R, ISEED /16807, 2147483647, 127773, 2836, 19283755/

        HI = ISEED/Q
        LO = MOD(ISEED,Q)
        TEST = A*LO - R*HI
        IF(TEST.GT.0)  THEN
          ISEED = TEST
        ELSE
          ISEED = TEST + M
        END IF
        RNDOM = FLOAT(ISEED)/M
        RETURN
        END
```

II. Using Histo-Scope Routines only (no HBOOK calls)

```
      PROGRAM HSEXAMPLE
C
C PROGRAM TO GENERATE RANDOM NTUPLE AND HISTOGRAM DATA
C
      IMPLICIT NONE
#include "histoscope.inc"
C
C    IF COMPILING PROGRAM ON VMS OR IBM/AIX SYSTEM, COMMENT OUT
C    ABOVE INCLUDE AND REMOVE THE C IN COLUMN 1 OF ONE OF THE FOLLOWING:
C
C    IBM/AIX:
C     INCLUDE 'histoscope.inc'
C    VMS:
C     INCLUDE 'HISTO_INC:histoscope.inc'
      INTEGER I, J, K, N_VARIABLES, HS1_ID, HS2_ID, NTUPLE_ID
      INTEGER IND_ID, ISTAT
      REAL VALUES(8)
      REAL RNDOM
      CHARACTER*8 TAGS(8)

      N_VARIABLES = 8
         TAGS(1) = 'X'
         TAGS(2) = 'Y'
         TAGS(3) = 'IDX'
         TAGS(4) = 'SQRT(X)'
         TAGS(5) = 'SQRT(Y)'
         TAGS(6) = 'LOG(X)'
         TAGS(7) = 'LOG(Y)'
         TAGS(8) = 'SIN(IDX)'
      CALL HS_INITIALIZE('hsExample.f')
      CALL HS_HISTOSCOPE(1)
      NTUPLE_ID = HS_CREATE_NTUPLE(101, 'EXAMPLE NTUPLE', 'HS',
     &     N_VARIABLES, TAGS)
      HS1_ID = HS_CREATE_1D_HIST(102, 'EXAMPLE 1D HIST', 'HS', 'Y',
     &     'TOTAL', 100, 4600., 5400.)
      HS2_ID = HS_CREATE_2D_HIST(103, '2D HIST EXAMPLE', 'HS','X', 'Y',
     &     'TOTAL', 100, 100, 4600., 5400., 4600., 5400.)
      IND_ID = HS_CREATE_INDICATOR(104, 'INDICATOR EXAMPLE', 'HS', 0.,
     &      10000.)
C
C        RANDOM DATA COMPUTATION - THIS COULD BE ANY CODE YOU WISH
C
```

```
DO I = 1, 10000
    DO J = 1, 2
     VALUES(J) = 0
     DO K = 1, 1000
         VALUES(J) = VALUES(J) + RNDOM() * 10.
     ENDDO
    ENDDO

    VALUES(3) = I
    VALUES(4) = SQRT(VALUES(1))
    VALUES(5) = SQRT(VALUES(2))
    VALUES(6) = LOG(VALUES(1))
    VALUES(7) = LOG(VALUES(2))
    VALUES(8) = SIN(VALUES(3))

    ISTAT = HS_FILL_NTUPLE(NTUPLE_ID, VALUES)
    CALL HS_FILL_1D_HIST(HS1_ID, VALUES(2), 1.)
    CALL HS_FILL_2D_HIST(HS2_ID, VALUES(1), VALUES(2), 1.)
    CALL HS_SET_INDICATOR(IND_ID, FLOAT(I))
    CALL HS_UPDATE

    IF ( MOD(I, 200) .EQ. 0)  THEN
            PRINT *, ' Reaching computation ...', I
     ENDIF
END DO
ISTAT = HS_SAVE_FILE('hsExample.hs')
CALL HS_COMPLETE
STOP
END

 REAL FUNCTION RNDOM()
 INTEGER A,M,Q,R,HI,LO,TEST
 DATA A, M, Q, R, ISEED /16807, 2147483647, 127773, 2836, 19283755/

 HI = ISEED/Q
 LO = MOD(ISEED,Q)
 TEST = A*LO - R*HI
 IF(TEST.GT.0)  THEN
   ISEED = TEST
 ELSE
   ISEED = TEST + M
 END IF
 RNDOM = FLOAT(ISEED)/M
 RETURN
 END
```

III. Using Histo-Scope Application Program Interface

```
      PROGRAM APIEXAMPLE
C
C PROGRAM TO GENERATE RANDOM N-TUPLE AND HISTOGRAM DATA AND DEMONSTRATE
C ADVANCED FEATURES OF THE HISTO-SCOPE APPLICATION PROGRAMMING INTERFACE
C
      IMPLICIT NONE
#include "histoscope.inc"
C
C    IF COMPILING PROGRAM ON VMS OR IBM/AIX SYSTEM, COMMENT OUT
C    ABOVE INCLUDE AND REMOVE THE C IN COLUMN 1 OF NEXT ONE
C
C     INCLUDE 'histoscope.inc'
      INTEGER I, J, K, N_VARIABLES, HS1_ID, HS2_ID, NTUPLE_ID, LEN
      INTEGER CTRL_ID, IND_ID, CTRL_NPTS_ID, CTRL_MERRS_ID, I1, I2
      INTEGER TRIGGER_ID, NUM_PTS, ISTAT, ID(20), NUM_ITEMS
      REAL VALUES(8), CVAL, FVALPTS, ERRVAL_M, R1, R2, R3
      REAL ERRS1D(100), ERRS2D(10000), ERRS1D_M(100), ERRS2D_M(10000)
      REAL RNDOM, XMIN, XMAX, OVERFLOWS(9)
      CHARACTER*8 TAGS(8)
      CHARACTER *80 NAME
C
C    NTUPLE PARAMETERS:
C
      N_VARIABLES = 8
      NUM_PTS = 10000
         TAGS(1) = 'X'
         TAGS(2) = 'Y'
         TAGS(3) = 'IDX'
         TAGS(4) = 'GAUS-1'
         TAGS(5) = 'GAUS-2'
         TAGS(6) = 'LOG(X)'
         TAGS(7) = 'LOG(Y)'
         TAGS(8) = 'SIN(IDX)'
C
C    INITIALIZE ERROR ARRAYS FOR HISTOGRAMS:
C
      DO I = 1, 10000
          ERRS2D(I) = .1
          ERRS2D_M(I) = 0.
      ENDDO
      DO I = 1, 100
          ERRS1D(I) = .1
          ERRS1D_M(I) = 0.
```

```
      ENDDO
C
C    INITIALIZE THE HISTOSCOPE API, START A PRE-CONNECTED HISTO-SCOPE
C    PROCESS, AND CREATE ITEMS TO LOOK AT:
C
      CALL HS_INITIALIZE('APIEXAMPLE')
      CALL HS_HISTOSCOPE(1)
      NTUPLE_ID = HS_CREATE_NTUPLE(101, 'EXAMPLE NTUPLE', 'HS',
     &      N_VARIABLES, TAGS)
      HS1_ID = HS_CREATE_1D_HIST(102, 'EXAMPLE 1D HIST', 'HS', 'Y',
     &      'TOTAL' ,100, 4600., 5400.)
      HS2_ID = HS_CREATE_2D_HIST(103, '2D HIST EXAMPLE', 'HS',
     &      'GAUS-1','GAUS-2', 'TOTAL', 100, 100, -3.5, 3.5, -4., 4.)
      IND_ID = HS_CREATE_INDICATOR(104, 'INDICATOR EXAMPLE', 'HS',
     &      0., 10000.)
      CTRL_ID = HS_CREATE_CONTROL(201, 'CONTROL: PAUSE IF .LT. 0',
     &      'HS', -500., +500., 0.)
      CTRL_NPTS_ID = HS_CREATE_CONTROL(202, 'CONTROL: NUM OF PTS',
     &      'HS', 0., 10000., 10000.)
      CTRL_MERRS_ID = HS_CREATE_CONTROL(203,
     &      'CONTROL: ACCUM NONSYM NEG ERRS', 'HS', 0., 1., 0.)
      TRIGGER_ID = HS_CREATE_TRIGGER(301, 'TRIGGER EXAMPLE', 'HS')
      PRINT *
      PRINT *, ' NUMBER OF ITEMS CREATED: ', HS_NUM_ITEMS()
      PRINT *
C
C     RANDOM DATA COMPUTATION - THIS COULD BE ANY CODE YOU WISH
C
      I = 1
      DO WHILE (I .LE. NUM_PTS)
          DO J = 1, 2
            VALUES(J) = 0
            DO K = 1, 1000
               VALUES(J) = VALUES(J) + RNDOM() * 10.
            ENDDO
          ENDDO

          VALUES(3) = I
          VALUES(6) = LOG(VALUES(1))
          VALUES(7) = LOG(VALUES(2))
          VALUES(8) = SIN(VALUES(3))

 50       R1 = RNDOM()
          R2 = RNDOM()
          R1 = 2.0 * R1 - 1.0
```

```
        R2 = 2.0 * R2 - 1.0
        R3 = R1 * R1 + R2 * R2
        IF (R3 .GT. 1.0 ) GOTO 50
        VALUES(4) = R1 * SQRT((-2.0*LOG(R3))/R3)
        VALUES(5) = R2 * SQRT((-2.0*LOG(R3))/R3)

        ISTAT = HS_FILL_NTUPLE(NTUPLE_ID, VALUES)
        CALL HS_FILL_1D_HIST(HS1_ID, VALUES(2), 1.)
        CALL HS_FILL_2D_HIST(HS2_ID, VALUES(4), VALUES(5), 1.)
        CALL HS_SET_INDICATOR(IND_ID, FLOAT(I))
        IF (HS_CHECK_TRIGGER(TRIGGER_ID) .NE. 0)
     &          PRINT *, ' Trigger set by HistoScope'
C
C    CALL HS_UPDATE EVERY FILL ITERATION, SO THAT THE NEW DATA CAN
C    BE SEEN BY THE HISTO-SCOPE USER
C
        CALL HS_UPDATE

        IF ( MOD(I, 200) .EQ. 0)  THEN
                PRINT *, ' Reaching computation ...', I
C
C           SET POSITIVE ERRORS FOR HISTOGRAMS; IF HISTO-SCOPE USER
C           DIRECTED, COMPUTE AND SET NON-SYMMETRIC NEGATIVE ERRORS:
C
             CALL HS_SET_1D_ERRORS(HS1_ID, ERRS1D, %VAL(0))
             CALL HS_SET_2D_ERRORS(HS2_ID, ERRS2D, %VAL(0))
             CALL HS_READ_CONTROL(CTRL_MERRS_ID, ERRVAL_M)
           DO K = 1, 10000
               ERRS2D(K) = ERRS2D(K) + .1
                  IF (ERRVAL_M .GT. 0)
      &                             ERRS2D_M(K) = ERRS2D_M(K) + ERRVAL_M
           ENDDO
           DO K = 1, 100
               ERRS1D(K) = ERRS1D(K) + .1
                  IF (ERRVAL_M .GT. 0)
      &                             ERRS1D_M(K) = ERRS1D_M(K) + ERRVAL_M
           ENDDO
              IF (ERRVAL_M .GT. 0) THEN
                  CALL HS_SET_1D_ERRORS(HS1_ID, %VAL(0), ERRS1D_M)
                  CALL HS_SET_2D_ERRORS(HS2_ID, %VAL(0), ERRS2D_M)
              ENDIF
            ENDIF
C
C    READ CONTROLS AND SET THE NUMBER OF ITERATIONS ACCORDINGLY.
C    ABSTAIN FROM DATA COLLECTION IF THE HISTO-SCOPE USER SETS
```

```
C     "PAUSE IF .LT. 0" CONTROL < 0, BUT STILL READ "NUMBER OF POINTS"
C     CONTROL AND CHECK THE TRIGGER:
C
          CALL HS_READ_CONTROL(CTRL_ID, CVAL)
          CALL HS_READ_CONTROL(CTRL_NPTS_ID, FVALPTS)
          NUM_PTS = INT(FVALPTS)
          DO WHILE (CVAL .LT. 0.)
            IF (HS_CHECK_TRIGGER(TRIGGER_ID) .NE. 0)
     &                  PRINT *, ' Trigger set by HistoScope'
            CALL HS_UPDATE
            CALL HS_READ_CONTROL(CTRL_ID, CVAL)
             ENDDO

      I = I + 1
      END DO
C
C     DATA COLLECTION IS FINISHED.  SAVE ALL DATA TO A FILE FOR LATER
C     REFERENCE AND PRINT SOME STATISTICS.  ALLOW THE HISTO-SCOPE USER
C     TO SEE ALL THE DATA HE WISHES BEFORE STOPPING THE PROGRAM.
C
      ISTAT = HS_SAVE_FILE('apiExample.hs')
      NUM_ITEMS = HS_LIST_ITEMS(' ','HS', ID, 20, 1)
      DO J = 1, NUM_ITEMS
          PRINT *
          IF (HS_TYPE(ID(J)) .EQ. HS_1D_HISTOGRAM) THEN
                CALL HS_1D_HIST_RANGE(ID(J), XMIN, XMAX)
                PRINT *, ' ITEM UID#', HS_UID(ID(J)),
     &                ' IS A 1-D HISTOGRAM.'
                PRINT *, '   ITS RANGE IS: ', XMIN, ', ', XMAX
                PRINT *, '   ITS INTEGRAL IS: ', HS_HIST_INTEGRAL(ID(J))
                CALL HS_1D_HIST_STATS(ID(J), R1, R2)
                PRINT *, '   ITS MEAN IS: ', R1, 'STD DEV:', R2
          ENDIF
          IF (HS_TYPE(ID(J)) .EQ. HS_2D_HISTOGRAM) THEN
                CALL HS_2D_HIST_OVERFLOWS(ID(J), OVERFLOWS)
                PRINT *, ' ITEM UID#', HS_UID(ID(J)),
     &                ' IS A 2-D HISTOGRAM.  '
                PRINT *, '   ITS OVERFLOWS ARE: ', OVERFLOWS
                CALL HS_2D_HIST_NUM_BINS(ID(J), I1, I2)
                PRINT *, '   NUMBER OF BINS IN X:', I1, ', IN Y: ',
     &                I2
                PRINT *, '   BIN VALUE AT (0., 0.) IS: ',
     &                HS_2D_HIST_XY_VALUE(ID(J), 0., 0.)
                CALL HS_2D_HIST_MAXIMUM(ID(J), R1, R2, I, K, R3)
                PRINT *, '   MAX VALUE IS AT (', R1, ', ', R2, '): ',
```

```
&                   R3
     ENDIF
     IF (HS_TYPE(ID(J)) .EQ. HS_NTUPLE) THEN
          PRINT *, ' ITEM UID#', HS_UID(ID(J)),
&                   ' IS AN N-TUPLE WITH ',
&                   HS_NUM_VARIABLES(NTUPLE_ID), ' VARIABLES.'
          LEN = HS_VARIABLE_NAME(ID(J), 1, NAME)
          PRINT *, '   THE FIRST VARIABLE NAME IS: ', NAME(1:LEN)
          PRINT *, '   THE VALUE OF ', NAME(1:LEN), '[250] IS: ',
&                   HS_NTUPLE_VALUE(ID(J), 250, 1)
     ENDIF
     IF (HS_TYPE(ID(J)) .EQ. HS_CONTROL) THEN
          LEN = HS_TITLE(ID(J), NAME)
          PRINT *, ' ITEM UID#', HS_UID(ID(J)),
&                   ' IS A CONTROL WITH TITLE: ', NAME(1:LEN)
          CALL HS_READ_CONTROL(ID(J), CVAL)
          PRINT *, '   ITS VALUE IS: ', CVAL
     ENDIF
     IF (HS_TYPE(ID(J)) .EQ. HS_INDICATOR) THEN
          LEN = HS_TITLE(ID(J), NAME)
          PRINT *, ' ITEM UID#', HS_UID(ID(J)),
&                   ' IS AN INDICATOR WITH TITLE:  ', NAME(1:LEN)
     ENDIF
     IF (HS_TYPE(ID(J)) .EQ. HS_TRIGGER) THEN
          LEN = HS_TITLE(ID(J), NAME)
          PRINT *, ' ITEM UID#', HS_UID(ID(J)),
&                   ' IS A TRIGGER WITH TITLE:  ', NAME(1:LEN)
     ENDIF
 END DO
 PRINT *
 PRINT *, ISTAT, ' ITEMS SAVED TO FILE:  apiExample.hs'
 CALL HS_COMPLETE_AND_WAIT
 STOP
 END


  REAL FUNCTION RNDOM()
  INTEGER A,M,Q,R,HI,LO,TEST
  DATA A, M, Q, R, ISEED /16807, 2147483647, 127773, 2836, 19283755/

  HI = ISEED/Q
  LO = MOD(ISEED,Q)
  TEST = A*LO - R*HI
  IF(TEST.GT.0)  THEN
    ISEED = TEST
```

```
ELSE
  ISEED = TEST + M
END IF
RNDOM = FLOAT(ISEED)/M
RETURN
END
```

C EXAMPLE PROGRAMS

I. Using Histo-Scope with HBOOK on Unix:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <histoscope.h>
/*
 * HbookExample.c - program to generate random Ntuple and histogram data
 *
 *    ** Note:  This program will not work on VMS, since HBOOK will    **
 *    **              expect descriptors for all string parameters.    **
 *    **         Also, on VMS, FORTRAN does not append underscores     **
 *    **             to routine names.                                 **
 */
#define TRUE 1
#define HLIM 250000
struct pawC { int h[HLIM]; } pawc_;
main()
{
    int n_variables = 8;
    int i, j, k, hlim = HLIM;
    float values[8];
    float con_1 = 1., con_0 = 0., con_3400 = 3400., con_5400 = 5400.;
    int hb_1d = 6000, hb_2d = 5000, hb_nt = 900, ic_100 = 100, ic_1000 =
                                1000;
    static char hbNames[8][8] = { "X      ", "Y      ", "idx    ", "sqrt(X)",
                                "sqrt(Y)", "log(X) ", "log(Y) ", "sin(idx)" };

    hs_initialize("HbookExample.c");
    hs_histoscope(TRUE);
    hlimit_(&hlim);
    hbookn_(&hb_nt, "example_ntuple", &n_variables, " ", &ic_1000,
       hbNames, 14, 1, 8);
    hbook2_(&hb_2d, "2d hist example", &ic_100, &con_3400, &con_5400,
       &ic_100, &con_3400, &con_5400, &con_0, 15);
    hbook1_(&hb_1d, "example 1d hist", &ic_100, &con_3400, &con_5400,
       &con_0, 15);
    printf(" HLDIR called:\n");
    hldir_("//PAWC", "T", 6, 1);
    hs_hbook_setup("//PAWC");

    for (i = 1; i <= 10000; ++i) {

       /* Random data computation - This could be any code you wish */
       for (j = 0; j < 2; ++j) {
            values[j] = 0;
            for (k = 0; k < 1000; k++)
                values[j] += fmod((float)random(), 10.);
       }
       values[2] = i;
       values[3] = sqrt(values[0]);
       values[4] = sqrt(values[1]);
       values[5] = log(values[0]);
       values[6] = log(values[1]);
       values[7] = sin(values[2]);
```

```
        hfill_(&hb_2d, &values[0], &values[1], &con_1);
        hfill_(&hb_1d, &values[1], &con_0, &con_1);
        hfn_(&hb_nt, values);
        hs_update();

        /*  Print one period per Ntuple fill, to get  */
        /*   visual feedback of program operation   */
        printf(".");
        fflush(stdout);
    }

     hs_complete();
     hrput_(&con_0, "hbookexample.hst", "TN", 16, 2);
}
```

II. Using Histo-Scope Routines only (no HBOOK calls)

```
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <histoscope.h>
/*
 *      HsExample.c - program to generate random Ntuple and histogram data
 */
#define TRUE 1
main()
{
     int n_variables = 8;
     float values[8];
     int i, j, k, ntuple_id, hs1_id, hs2_id, indic_id;
     static char *names[8] = {      "X",       "Y",   "index",     "sqrt(X)",
                               "sqrt(Y)", "log(X)", "log(Y)", "sin(index)"  };

      hs_initialize("HsExample.c");
      hs_histoscope(TRUE);
      ntuple_id  = hs_create_ntuple(101, "Example Ntuple", "HS", n_variables, names);
      hs1_id     = hs_create_1d_hist(102, "Example 1d Hist", "HS", "Y", "total", 100,
                   3400., 5400.);
      hs2_id     = hs_create_2d_hist(103, "2d Hist Example", "HS","X", "Y", "total",
                   100, 100, 3400., 5400., 3400., 5400.);
      indic_id   = hs_create_indicator(104, "Example Indicator", "HS", 0., 10000.);

     for (i = 1; i <= 10000; ++i) {

        /* Random data computation - This could be any code you wish */
        for (j = 0; j < 2; ++j) {
             values[j] = 0;
             for (k = 0; k < 1000; k++)
                 values[j] += fmod((float)random(), 10.);
        }
        values[2] = i;
        values[3] = sqrt(values[0]);
        values[4] = sqrt(values[1]);
        values[5] = log(values[0]);
        values[6] = log(values[1]);
        values[7] = sin(values[2]);

        hs_fill_ntuple(ntuple_id, values);
```

```
        hs_fill_1d_hist(hs1_id,  values[1],  1.);
        hs_fill_2d_hist(hs2_id,  values[0],  values[1],  1.);
          hs_set_indicator(indic_id, (float) i);
        hs_update();

        /*   Print one period per Ntuple fill, to get   */
        /*    visual feedback of program operation   */
        printf(".");
        fflush(stdout);
    }

     hs_save_file("HsExample.hs");
     hs_complete();
}
```

FORTRAN Binding Library Routines

Routines for all Users

HS_INITIALIZE(identity_string**)**

**CHARACTER ***n identity_string *(input, n ≤ 40)*

Initialize the Histo-Scope connection software and set up a potential connection to a Histo-Scope process. The identity string identifies the process in the Histo-Scope connection panel and may be up to 40 characters long. This routine should be called before any other Histo-Scope routine is called.

HS_UPDATE

This routine must be called periodically to keep the Histo-Scope display up to date and to accept connections from Histo-Scopes. You may call this as often as you like with no wasted CPU usage. It will only transmit the data for the items that are currently displayed, and only as frequently as requested by the user of the Histo-Scope process, even if you call HS_UPDATE very often.

HS_COMPLETE

Close all connections with Histo-Scope processes and notify them that the program is finished. This routine gives connected Histo-Scope processes a chance to request a last update. This may take a few seconds to accomplish.

If you are using HBOOK, this routine should be called <u>before</u> calling HREND, since HS_COMPLETE may still access the HBOOK data.

Under rare circumstances, an HS_COMPLETE call can take up to 1 minute to complete if one or more Histo-Scope(s) are viewing a large amount of data and have somehow fallen behind in receiving the data. This call allows a maximum of one minute for sending all data requested by Histo-Scopes before the HS_COMPLETE was called.

HS_COMPLETE_AND_WAIT

Notify any connected Histo-Scope processes that the program is finished and is waiting for them to exit or disconnect. While any Histo-Scope is connected, this call will not return to the calling program. Instead all Histo-Scope requests, including new connections will be serviced. After all connected Histo-Scopes have disconnected, all data memory is freed and control is returned to the calling program. As noted above, this call also should be called before calling the HBOOK routine HREND.

HS_HISTOSCOPE(return_immediately**)**

INTEGER return_immediately *(input)*

For users who want to build Histo-Scope into their program rather than using it separately as an inspection tool. HS_HISTOSCOPE invokes Histo-Scope as a sub-process. The scope process is pre-connected to display the data generated in the user process. The routine can either start the Histo-Scope and return immediately (`return_immediatly` .NE. 0), or it can start it and return after the user closes the Histo-Scope window (`return_immediately` .EQ. 0).

HS_HISTO_WITH_CONFIG(return_immediately, config_file**)**

INTEGER return_immediately *(input)*

**CHARACTER ***n config_file *(input, n ≤ system maximum for filenames)*

For users who want to build Histo-Scope into their program **and** have it automatically display plots. HS_HISTO_WITH_CONFIG invokes Histo-Scope as a sub-process and displays the plots specified in the configuration file `config_file`. The scope process is pre-connected to display the data generated in the user process. The routine can either start the Histo-Scope and return immediately (`return_immediatly` .NE. 0), or it can start it and return after the user closes the Histo-Scope window (`return_immediately` .EQ. 0).

HS_KILL_HISTOSCOPE()

This routine allows you to "kill" any Histo-Scope processes started via the HS_HISTOSCOPE or HS_HISTO_WITH_CONFIG calls. The "kill" is accomplished through a message request to the Histo-Scope process for it to exit.

num_scopes **= HS_NUM_CONNECTED_SCOPES()**

INTEGER num_scopes *(returned)*

Returns the number of connected Histo-Scopes. This routine is useful for checking that a Histo-Scope created with HS_HISTOSCOPE has successfully begun executing and is connected to its client program. Since this takes some time, it is suggested that HS_NUM_CONNECTED_SCOPES be called to check that a scope is connected before calling HS_LOAD_CONFIG_FILE or HS_LOAD_CONFIG_STRING.

HS_LOAD_CONFIG_FILE(config_file**)**

**CHARACTER ***n config_file *(input)*

Calling HS_LOAD_CONFIG_FILE causes all connected Histo-Scopes to read the configuration file specified by `config_file` and execute all of the commands to display plots in the file.

HS_LOAD_CONFIG_STRING(config_string**)**

**CHARACTER ***n config_string *(input)*

Calling HS_LOAD_CONFIG_STRING causes all connected Histo-Scopes to execute all of the commands to display plots in the string specified by `config_string`.

Routines for HBOOK users

HS_HBOOK_SETUP(top_directory**)**

**CHARACTER ***n top_directory *(input, n ≤ HBOOK's maximum)*

Call this routine after you book the histograms or Ntuples you will use in the program and after you call HS_INITIALIZE. The routine will set up all current HBOOK histograms and Ntuples in the specified top directory for use with Histo-Scope. The `top_directory` string supplied should begin with "//". (**Note:** unpredictable results may occur if this routine is called without first calling the appropriate HBOOK routines for the top directory.)

HS_RESET_HBOOK_SETUP(top_directory**)**

**CHARACTER ***n top_directory *(input, n ≤ HBOOK's maximum)*

Call this routine if you have booked new histograms or Ntuples, or deleted, renamed, rebinned, or re-set existing ones since calling HS_HBOOK_SETUP. The `top_directory` string supplied should begin with "//".

Histo-Scope data routines

The Histo-Scope data routines provide a simple interface for creating data items, performing computations on this data, and managing the characteristics of the data items. They can be used instead of HBOOK routines or in addition to them.

Whenever a Histo-Scope data item is created, it is assigned a Histo-Scope identification number, **`id`**, which is returned to the calling routine. These id's are sequentially assigned beginning at the number 1 (one) and are NOT preserved across saving the item to a file and reading it back from the file. They are used, instead, for internal identification of an item. For the user's convenience a User-specified identification number, **`uid`**, can be specified which, with its category, always identifies the item. uid's specified by the user must therefore be unique within the category specified for the item or the item will not be created. All item creation routines include a uid parameter. This uid may be specified as 0, in which case it need not be unique within the category. However specifying uid's of 0 will hamper utilization of such routines as HS_SUM_FILE, HS_SUM_CATEGORY, HS_READ_FILE_ITEMS, and HS_SAVE_FILE_ITEMS, which use uid lists to perform their function.

Item Creation Routines

```
id = HS_CREATE_1D_HIST(uid, title, category, x_label, y_label,
                            num_bins, min, max)
```

INTEGER uid *(input)*

CHARACTER ***n** title *(input, n ≤ 80)*

CHARACTER ***n** category *(input, n ≤ 255)*

CHARACTER ***n** x_label *(input, n ≤ 80)*

CHARACTER ***n** y_label *(input, n ≤ 80)*

INTEGER num_bins *(input)*

REAL min *(input)*

REAL max *(input)*

INTEGER id *(returned)*

Create (or book) a one-dimensional histogram. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the histogram window when it is displayed. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. The histogram axes are given the labels `x_label` and `y_label`. `num_bins` are the number of bins for for the histogram. Any values .LE. `min` or .GE. `max` are collected into under- or over-flow bins. The histoscope-defined `id` for the histogram is returned to the user and is used in later calls to identify the histogram just created. If `id` is .LE. 0, an error occured in creating the histogram (check the arguments supplied to the function). The value of `min` should be less than `max`, and `num_bins` should be greater than 0.

```
id = HS_CREATE_2D_HIST(uid, title, category, x_label, y_label,
                            z_label, x_bins, y_bins, x_min, x_max, y_min,
                            y_max)
```

INTEGER uid *(input)*

CHARACTER ***n** title *(input, n ≤ 80)*

CHARACTER ***n** category *(input, n ≤ 255)*

CHARACTER ***n** x_label *(input, n ≤ 80)*

CHARACTER ***n** y_label *(input, n ≤ 80)*

CHARACTER ***n** z_label *(input, n ≤ 80)*

INTEGER x_bins *(input)*

INTEGER y_bins *(input)*

REAL x_min *(input)*

REAL x_max *(input)*

REAL y_min *(input)*

REAL y_max *(input)*

INTEGER id *(returned)*

Create (or book) a two-dimensional histogram. uid defines a user-specified identification number which can later be used along with the category to specify the item created. title defines the title for the histogram window when it is displayed. category is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the histogram will appear in the top level category. The histogram axes are given the labels x_label, y_label, and z_label. x_bins and y_bins are the number of bins for each axis of the histogram. Any values .LE. min or .GE. max are collected into under- or over-flow bins. The histoscope-defined id for the histogram is returned and is used in later calls to identify the histogram just created. If id is .LE. 0, an error occured in creating the histogram (check the arguments supplied to the function). The value of x_min and y_min should be less than its corresponding maximum, and x_bins and y_bins should both be greater than 0.

id = **HS_CREATE_NTUPLE(**uid, title, category, num_variables, names**)**

INTEGER uid *(input)*

**CHARACTER ***n title *(input, n ≤ 80)*

**CHARACTER ***n category *(input, n ≤ 255)*

INTEGER num_variables *(input)*

CHARACTER ***n names(**num_variables**)** *(input, n ≤ 80)*

INTEGER id *(returned)*

Defines an Ntuple with num_variables variables and variable names taken from names**(**num_variables**)**. uid defines a user-specified identification number which can later be used along with the category to specify the item created. title defines the title for the histogram window when it is displayed. category is a string defining a hierarchical "location" for the Ntuple. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the Ntuple will appear in the top level category. Storage allocation of the Ntuple is automatic: as the Ntuple grows, more space will be allocated. **However, Ntuples must fit into virtual memory.** The Histo-Scope-defined id for the histogram is returned and is used in later calls to identify the Ntuple just created. If id is .LE.

0, an error occured in creating the Ntuple (check the arguments supplied to the function).

`id = HS_CREATE_INDICATOR(uid, title, category, min, max)`

INTEGER uid *(input)*

CHARACTER *n title *(input, n ≤ 80)*

CHARACTER *n category *(input, n ≤ 255)*

REAL min *(input)*

REAL max *(input)*

INTEGER id *(returned)*

Creates an indicator. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `category` is a string defining a hierarchical "location" for the indicator. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the indicator will appear in the top level category. The Histo-Scope-defined id for the indicator is returned and is used in later calls to identify the indicator just created. If `id` is .LE. 0, an error occured in creating the indicator (check the arguments supplied to the function).

`id = HS_CREATE_CONTROL(uid, title, category, min, max, default_value)`

INTEGER uid *(input)*

CHARACTER *n title *(input, n ≤ 80)*

CHARACTER *n category *(input, n ≤ 255)*

REAL min *(input)*

REAL max *(input)*

REAL default_value *(input)*

INTEGER id *(returned)*

Creates a control. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `category` is a string defining a hierarchical "location" for the control. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the control will appear in the top level category. The Histo-Scope-defined `id` for the control is returned and is used in later calls to identify the control just created. If `id` is .LE. 0, an error occured in creating the control (check the arguments supplied to the function).

`id = HS_CREATE_TRIGGER(uid, title, category)`

INTEGER uid *(input)*

CHARACTER *n title *(input, n ≤ 80)*

CHARACTER *n category *(input, n ≤ 255)*

Creates a trigger. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `category` is a string defining a hierarchical "location" for the trigger. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the trigger will appear in the top level category. The Histo-Scope-defined id for the trigger is returned and is used in later calls to identify the trigger just created. If `id` is .LE. 0, an error occured in creating the trigger (check the arguments supplied to the function).

`id` **`= HS_CREATE_GROUP(`**`uid, title, category, groupType, numItems, itemId, ersDisp`**`)`**

`INTEGER` `uid` *`(input)`*

`CHARACTER *``n title` *`(input, n ≤ 80)`*

`CHARACTER *``n category` *`(input, n ≤ 255)`*

`INTEGER` `groupType` *`(input)`*

`INTEGER` `numItems` *`(input)`*

`INTEGER` `itemId(numItems)` *`(input, numItems <= 81)`*

`INTEGER` `errsDisp(numItems)` *`(input, numItems <= 81)`*

`INTEGER` `id` *`(returned)`*

Creates a group , which is an item that points to other data items, thus grouping them into one addressable entity. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `category` is a string defining a hierarchical "location" for the group. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the group will appear in the top level category. The Histo-Scope-defined id for the group is returned and is used in later calls to identify the groups just created. If `id` is .LE. 0, an error occured in creating the group (check the arguments supplied to the function). `groupType` is the group's suggested window type. The number of items in this group, specified by numItems, should not be larger than 81. `itemId` is an array of Histo-Scope id numbers for the items in the group. `errsDisp` is an array of values (NO_ERROR_BARS, DATA_ERROR_BARS, GAUSSIAN_ERROR_BARS) that corresponds to the itemId array and tells Hist-Scope how to display errors for the corresponding histogram (used only when the corresponding data item is a histogram).

A group can be made out of a number of items in order to display them as a Multiple Plot Window, an Overlaid Plot, or simply as a convenience for putting plots up individually with fewer mouse clicks. Data items may be put into more than one group. Including an item in the group does not effect the display of the item individually in the Histo-Scope main panel. The item still appears in the main panel list box and can be viewed individually. The category string can be used to affect how Histo-Scope items appear in the main panel.

When a group is selected in Histo-Scope's main window and the user presses the "View" Button, the groupType field tells Histo-Scope how to display the items in the group. This can be overridden in the Histo-Scope main panel by using the "View Multiple" or "View Overlaid" buttons. If groupType is HS_MULTI_PLOT, pressing the View button displays all the 1- and 2D histograms in the group as one Multi-plot window. Similarly, if you would like to see all the 1D histograms in a group displayed over each other in one plot window, you can specify a groupType of HS_OVERLAY_PLOT. Triggers, controls, and indicators cannot appear inside a Multiple Plot Window or Overlaid Plot and so these items in a group are displayed in their own windows.

Specifying HS_INDIVIDUAL for groupType means that if a group is selected, and the "View" button is pressed on the Histo-Scope main panel, each data item in the group will be viewed individually. In the case of Ntuples, the Ntuple panel will be displayed so the the user can choose the variables to plot and the type to use.

Filling Histograms/Ntuples and Setting Errors

HS_FILL_1D_HIST(`id, x, weight`**)**

INTEGER `id` *(input)*

REAL `x` *(input)*

REAL `weight` *(input)*

Adds a value (`weight`) to the one-dimensional histogram identified by `id`.

HS_FILL_2D_HIST(`id, x, y, weight`**)**

INTEGER `id` *(input)*

REAL `x` *(input)*

REAL `y` *(input)*

REAL `weight` *(input)*

Adds a value (`weight`) to the two-dimensional histogram identified by `id`.

HS_1D_HIST_BLOCK_FILL(`id, data, pos_errors, neg_errors`**)**

INTEGER `id` (*input*)

REAL `data`**(**`num_bins`**)** (*input*)

REAL `pos_errors`**(**`num_bins`**)** (*input*)

REAL `neg_errors`**(**`num_bins`**)** (*input*)

Replaces the accumulated bin and error data supplied for the one-dimensional histogram identified by `id` and clears the overflow bins. `data` is a one-dimensional array containing the new data for the histogram and must be of the appropriate size (i.e. the correct number of bins) for the one-dimensional histogram being filled. `pos_errors` and `neg_errors` are the errors of the one-dimensional histogram. Either error argument can be specified as %VAL(0), in which case the corresponding errors are not changed. `pos_errors` and `neg_errors` should have the same number of elements as the histogram has bins. If the histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** real numbers.

HS_2D_HIST_BLOCK_FILL(`id, data, pos_errors, neg_errors`**)**

INTEGER `id` (*input*)

REAL `data`**(**`x_bins, y_bins`**)** (*input*)

REAL `pos_errors`**(**`x_bins, y_bins`**)** (*input*)

REAL `neg_errors`**(**`x_bins, y_bins`**)** (*input*)

Replaces the accumulated bin and error data supplied for the two-dimensional histogram identified by `id` and clears the overflow bins. `data` is a two-dimensional array containing the new data for the histogram and must be of the appropriate size (i.e. the correct number of bins) for the two-dimensional histogram being filled. `pos_errors` and `neg_errors` are the errors of the two-dimensional histogram. Either error argument can be specified as %VAL(0), in which case the corresponding errors are not changed. `pos_errors` and `neg_errors` should have the same number of elements as the histogram has bins. If the histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** real numbers.

HS_SET_1D_ERRORS(`id, pos_errors, neg_errors`**)**

INTEGER `id` *(input)*

REAL `pos_errors`**(**`num_bins`**)** *(input)*

REAL `neg_errors`**(**`num_bins`**)** *(input)*

Copies one or both arrays of real numbers representing the errors of a one-dimensional histogram. Either error argument can be specified as %VAL(0), in which case the corresponding errors are not set. Arrays should have the same number of elements as the histogram has bins. If a histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** real numbers.

HS_SET_2D_ERRORS(id, pos_errors, neg_errors**)**

INTEGER id *(input)*

REAL pos_errors**(**x_bins, y_bins**)** *(input)*

REAL neg_errors**(**x_bins, y_bins**)** *(input)*

Copies arrays of real numbers representing the errors of a two-dimensional histogram. Either error argument can be specified as %VAL(0), in which case the corresponding errors are not set. Arrays should have the same number of elements in each dimension as the histogram has bins. If positive errors are supplied, but negative ones aren't, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** real numbers.

istatus **= HS_FILL_NTUPLE(**id, values**)**

INTEGER id *(input)*

REAL values**(**num_variables**)** *(input)*

INTEGER istatus *(returned)*

Adds an array of real `values`, in the order they are named in HS_CREATE_NTUPLE, to an Ntuple identified by `id`. Returns `istatus`. Check this status to ensure the array was added and to be certain your Ntuple still fits within the virtual memory of your machine. `istatus` .LE. 0 indicates memory was exhausted. `istatus` .EQ. 0 indicates an incorrect id or other user error. `istatus` .GT. 0 indicates success. The Ntuple id number is returned upon success.

Setting Indicators, Reading Controls, and Checking Triggers

HS_SET_INDICATOR(id, value**)**

INTEGER id *(input)*

REAL value *(input)*

Sets the value of an indicator to `value`. The value should be within the range specified for the indicator identified by `id`. <u>However the indicator is set to the value even if the value is out of range.</u>

HS_READ_CONTROL(id, return_value**)**

INTEGER id *(input)*

REAL return_value *(returned)*

Returns the value of a control in the variable `return_value`.

`istatus = `**`HS_CHECK_TRIGGER(`**`id`**`)`**

INTEGER id *(input)*

INTEGER istatus *(returned)*

Returns 1 if the trigger has been set by Histo-Scope; otherwise 0 is returned. Please note that this routine will return 1 once for each button press by the Histo-Scope user. Thus, if the user has pressed the trigger 5 times, the program must call `HS_CHECK_TRIGGER` at least 5 times.

Functions for looking up items

`id = `**`HS_ID(`**`uid, category`**`)`**

INTEGER uid *(input)*

CHARACTER ***n** category *(input, n ≤ 255)*

INTEGER id (*returned*)

Returns the Histo-Scope defined id for a specified uid and category pair, or -1 if a match was not found. `uid` is the user-specified identification number which was specified when the item was created. `category` is a string defining a hierarchical "location" for the item and was also specified when the item was created.

`id = `**`HS_ID_FROM_TITLE(`**`title, category`**`)`**

CHARACTER ***n** title *(input, n ≤ 80)*

CHARACTER ***n** category *(input, n ≤ 255)*

INTEGER id (*returned*)

Returns the Histo-Scope id for a specified item title and category pair, or -1 if a match was not found. `title` is the title specified for the item when it was created. `category` is a string specified at item creation defining a hierarchical "location" for the item.

`num_match = `**`HS_LIST_ITEMS(`**`title, category, ids, num, match_flag`**`)`**

CHARACTER ***n** title (*input, n ≤ 80*)

CHARACTER ***n** category (*input, n ≤ 255*)

INTEGER ids**(**num**)** (*returned*)

INTEGER num (*input*)

INTEGER match_flag (*input*)

INTEGER num_match (*returned*)

Fills a list of Histo-Scope id numbers. The user must allocate at least `num` integer elements for the array `ids`. These ids are returned in the order in which the histograms were created, or read in from a file. Returns the number of matching the title/category strings, `num_match`, which may be larger than `num`, the maximum number of items that can be returned. `title` specifies the title of the data items to list. `category` is a string specified at item creation defining a hierarchical "location" for the item. Optionally, the category can include a trailing "..." which will return the id numbers for items in subcategories of the specified category. For example, "HS/..." matches "HS" and "HS/Nts" but not "HSNTuples", "HSNTuples" nor "hs". If the category parameter is specified as "..." all categories will match including uncategorized items. `match_flag` specifies exact or inexact matches for the title string specified. If match_flag .EQ. 0, then all id's returned will exactly match the title specified. If match_flag .NE. 0, then leading spaces are ignored and id's *containing* the specified title are returned. Specifying " " for title and an inexact match for match_flag will match all titles. If the category supplied is not valid, -1 will be returned in num_match.

Functions for getting properties of all items

`uid = `**`HS_UID(`**`id`**`)`**

INTEGER id (*input*)

INTEGER uid (*returned*)

Returns the user-specified identification number (`uid`) value for a given Histo-Scope `id`, or 0 if `id` was invalid or not found.

`length = `**`HS_CATEGORY(`**`id, category`**`)`**

INTEGER id (*input*)

CHARACTER *n category (*input, n≤255*)

INTEGER length (*returned*)

Returns the `category` and its `length` for a given Histo-Scope `id`, or -1 if `id` was invalid.

`length = `**`HS_TITLE(`**`id, title`**`)`**

INTEGER id (*input*)

CHARACTER *n title (*input, n ≤ 80*)

INTEGER length (*returned*)

Returns the `title` and its `length` for an item specified by `id`, or -1 if `id` was invalid.

`data_type = `**`HS_TYPE(`**`id`**`)`**

INTEGER id (*input*)

INTEGER data_type (*returned*)

Returns the type of data, `data_type`, referred to by `id`. Valid types are:

HS_1D_HISTOGRAM	HS_2D_HISTOGRAM	HS_NTUPLE
HS_INDICATOR	HS_CONTROL	HS_TRIGGER
HS_NONE		

HS_NONE refers to a non-existent item.

These values are defined in histoscope.inc.

Functions for reading/writing items from/to files

`istatus` **= HS_READ_FILE(**`filename, prefix`**)**

CHARACTER *`n filename` *(input, n ≤ system maximum for filenames)*

CHARACTER *`n prefix` *(input, n≤255)*

INTEGER `num_items` (*returned*)

Reads all of the items from the Histo-Scope formatted file specified by `filename`. All category strings read will be prefixed by a new top level category specified in `prefix`. This may be specified as %VAL(0) (or " "), meaning that no prefix should be added. Returns the number of items read, or -1 if error.

`num_items` **= HS_READ_FILE_ITEMS(**`filename, prefix, category, uids, n_uids`**)**

CHARACTER *`n filename` *(input, n ≤ system maximum for filenames)*

CHARACTER *`n prefix` *(input, n≤255)*

CHARACTER *`n category` *(input, n ≤ 255)*

INTEGER `uids`**(**`n_uids`**)** *(input)*

INTEGER `n_uids` *(input)*

INTEGER `num_items` *(returned)*

Selectively reads items from a specific category and all its subcategories that match a list of `uids` from the Histo-Scope formatted file specified by `filename`. All category strings read from the file will be prefixed by a new top level category specified in `prefix`. This may be specified as %VAL(0) (or " "), meaning that no prefix should be added. If an item with the same (prefixed) category and uid already exists, the item in the file will not be read and an error message will inform the user. If n_uids is .GT. 0, the array `uids` specifies the uid's of items to read from the file. If `uids` is %VAL(0) or n_uids .EQ. 0, all items in the specified category and its subcategories will be read. `n_uids` specifies the number of items in array uids. Returns the number of items read, or -1 if an error occurred.

For example, if "HS" is specified for `category` and "Run1" for `prefix`, items read from the file with category "HS" will become "Run1/HS" and items read from the file with category "HS/Ntuples" will become "Run1/HS/Ntuples". The new (prefixed) category must be less than 255 characters in length, or the item will not be read from the file. To read non-selectively regardless of the category, specify " " as `category`.

num_items = **HS_SAVE_FILE(**filename**)**

**CHARACTER ***n filename *(input, n ≤ system maximum for filenames)*

INTEGER num_items *(returned)*

Saves all current histograms, Ntuples, indicators, and controls created by Histo-Scope library routines or read from Histo_scope files to a Histo-Scope-format file. Triggers are not written to the file. The number of items written is returned in num_items, or -1 if an error occurred. The named file is open for writing ("w"), discarding previous contents, if any.

num_items = **HS_SAVE_FILE_ITEMS(**filename, category, uids, n_uids**)**

**CHARACTER ***n filename *(input, n ≤ system maximum for filenames)*

**CHARACTER ***n category *(input, n ≤ 255)*

INTEGER uids**(**n_uids**)** *(input)*

INTEGER n_uids *(input)*

INTEGER num_items *(returned)*

Selectively saves the histograms, Ntuples, indicators, and controls in a specified category that match a list of uids to a Histo-Scope format file filename. Items will be written only when they match the specified category *exactly*. Specify " " as the category to save only uncategorized items. uids is an array of user id values of length n_uids specifying which items to write. If n_uids is .LE. 0 or uids is %VAL(0), all items in the category specified are saved. The number of items written is returned, or -1 if an error occurs. The file named is open for writing ("w"), discarding previous contents, if any.

Functions for managing data items

HS_RESET(id**)**

INTEGER id *(input)*

Resets the data item. For histograms, all of the bins and overflows are set to 0. For Ntuples all of the data is removed. Indicators are set to *not set*. Controls are set to their default value. For triggers, any pending trigger presses are cleared.

HS_DELETE(id**)**

INTEGER id *(input)*

Deletes a histogram, Ntuple, indicator, control, or trigger identified by id.

HS_DELETE_CATEGORY(category**)**

**CHARACTER ***n category *(input, n ≤ 255)*

Removes and deallocates all items in the named `category`. All data content is lost. The category string must *exactly* match the category of the item to be deleted *unless a trailing* "..." is specified as part of the category. When the "..." is specified, data items in **all subcategories** of the category will be deleted. Specifying " " or %VAL(0) for category will delete all uncategorized items. For example, specifying a category "HS/..." will delete all items with category "HS" and "HS/Nts" but not "HSNTuples" nor "hs". **If just "..." is specified, all items are deleted.**

HS_CHANGE_UID(`id, new_uid`**)**

INTEGER `id` *(input)*

INTEGER `new_uid` *(input)*

Changes the user-defined identification number (uid) to `new_uid` for an item specified by its Histo-Scope identification number, `id`. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new uid.

HS_CHANGE_CATEGORY(`id, newcategory`**)**

INTEGER `id` *(input)*

CHARACTER *`n category` *(input, n ≤ 255)*

Changes the category of an item specified by its Histo-Scope identification number, `id`. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new category.

HS_CHANGE_TITLE(`id, newtitle`**)**

INTEGER `id` *(input)*

CHARACTER *`n newtitle` *(input)*

Changes the title of an item specified by its Histo-Scope identification number, `id`. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new title.

`num_items` **= HS_NUM_ITEMS()**

INTEGER `num_items` *(returned)*

Returns the number of Histo-Scope data items defined so far.

Functions for Accessing Histogram Data

`num_bins` **= HS_1D_HIST_NUM_BINS(**`id`**)**

INTEGER `id` *(input)*

INTEGER `num_bins` *(returned)*

Returns the number of bins for a one-dimensional histogram identified by `id`, or -1 if `id` does not refer to a one-dimensional histogram.

HS_2D_HIST_NUM_BINS(`id, num_x_bins, num_y_bins`**)**

INTEGER `id` *(input)*

INTEGER `num_x_bins` (*returned*)

INTEGER `num_y_bins` (*returned*)

Returns the number of bins in the two-dimensional histogram specified by `id`. `num_x_bins` is the number of bins in X. `num_y_bins` is the number of bins in Y. -1 is returned for num_x_bins and num_y_bins if `id` does not refer to a two-dimensional histogram.

HS_1D_HIST_RANGE(`id, min, max`**)**

INTEGER `id` *(input)*

REAL `min` (*returned*)

REAL `max` (*returned*)

Returns the minimum (low edge of first bin) value in `min` and the maximum (high edge of last bin) value in `max` representing the horizontal limits of a one-dimensional histogram specified by `id`. If `id` does not refer to a one-dimensional histogram, zero is returned for `min` and `max`.

HS_2D_HIST_RANGE`(id, x_min, x_max, y_min, y_max`**)**

INTEGER `id` *(input)*

REAL `x_min` (*returned*)

REAL `x_max` (*returned*)

REAL `y_min` (*returned*)

REAL `y_max` (*returned*)

Returns the minimum and maximum values representing the horizontal limits of a two-dimensional histogram specified by `id`. If `id` does not refer to a two-dimensional histogram, zero is returned for the minimum and maximum values.

`num_entries = HS_NUM_ENTRIES(id)`

INTEGER id *(input)*

INTEGER num_entries (returned)

Returns the number of fill operations that have been performed on the histogram referred to by `id`, or -1 if the id does not refer to a histogram or Ntuple item.

HS_1D_HIST_BIN_CONTENTS(id, data**)**

INTEGER id *(input)*

REAL data**(**num_buns**)** *(returned)*

Returns the bin data from a one-dimensional histogram. Data must be dimensioned appropriately for the number of bins in the histogram.

HS_2D_HIST_BIN_CONTENTS(id, data**)**

INTEGER id *(input)*

REAL data**(**num_buns**)** *(returned)*

Returns the bin data from a two-dimensional histogram. Data must be dimensioned appropriately for the number of bins in the histogram.

status **= HS_1D_HIST_ERRORS(**id, pos_errors, neg_errors**)**

INTEGER id *(input)*

REAL pos_errors**(**num_bins**)** *(returned)*

REAL neg_errors**(**num_bins**)** *(returned)*

Return the error bar data from a one-dimensional histogram. `pos_errors` is a one-dimensional array to receive data for upper error bars. Must be dimensioned appropriately for the number of bins in the histogram. `neg_errors` is a one-dimensional array to receive data for lower error bars. neg_errors can be %VAL(0) which means only positive errors are returned. If non %VAL(0), neg_errors must be dimensioned appropriately for the number of bins in the histogram. The status returned can be:

HS_NO_ERRORS - Item has no error data

HS_POS_ERRORS - Positive errors (only) are returned

HS_BOTH_ERRORS - Positive and negative errors are returned

HS_ITEMNOTFOUND_ERRORS - id does not refer to a one-dimensional histogram

These values are defined in histoscope.inc.

`status = HS_2D_HIST_ERRORS(id, pos_errors, neg_errors)`

INTEGER `id` *(input)*

REAL `pos_errors(x_bins, y_bins)` *(input)*

REAL `neg_errors(x_bins, y_bins)` *(input)*

Returns the error bar data from a two-dimensional histogram. `pos_errors` is a two-dimensional array to receive data for upper error bars. Must be dimensioned appropriately for the number of bins in the histogram. `neg_errors` is a two-dimensional array to receive data for lower error bars. neg_errors can be %VAL(0), which means only positive errors are returned. If not %VAL(0), neg_errors must be dimensioned appropriately for the number of bins in the histogram. The status returned can be:

HS_NO_ERRORS - Item has no error data

HS_POS_ERRORS - Positive errors (only) are returned

HS_BOTH_ERRORS - Positive and negative errors are returned

HS_ITEMNOTFOUND_ERRORS - id does not refer to a one-dimensional histogram

These values are defined in histoscope.inc.

`HS_1D_HIST_OVERFLOWS(id, underflow, overflow)`

INTEGER `id` *(input)*

REAL `underflow` *(returned)*

REAL `overflow` *(returned)*

Returns the overflow data from a one-dimensional histogram.

`HS_2D_HIST_OVERFLOWS(id, overflows)`

INTEGER `id` *(input)*

REAL `overflows(x_bins, y_bins)` *(returned)*

Returns the overflow data from a two-dimensional histogram as a 3 x 3 array:

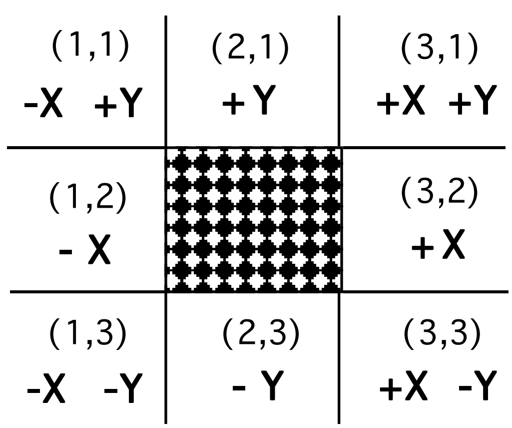


Figure 47

`value = HS_1D_HIST_X_VALUE(id, x)`

INTEGER id *(input)*

REAL x *(input)*

REAL valu *(returned)*

Returns the value stored in the histogram bin referenced by x. The result is undefined if id is not a one-dimensional histogram, or if x is not within the range of the histogram.

`value = HS_2D_HIST_XY_VALUE(id, x, y)`

INTEGER id *(input)*

REAL x *(input)*

REAL y *(input)*

REAL value *(returned)*

Returns the value stored in the histogram bin referenced by (x,y). The result is undefined if id is not a two-dimensional histogram, or if x or y are not within the range of the histogram.

`value = HS_1D_HIST_BIN_VALUE(id, bin_num)`

INTEGER id *(input)*

INTEGER bin_num *(input)*

REAL value *(returned)*

Returns the value in the histogram bin referred to by bin (or channel) number `bin_num`. The result is undefined if id is not a one-dimensional histogram or bin_num is less than 1 or greater than the number of bins.

`value = HS_2D_HIST_BIN_VALUE(id, x_bin_num, y_bin_num)`

INTEGER id *(input)*

INTEGER x_bin_num *(input)*

INTEGER y_bin_num *(input)*

REAL value *(returned)*

Returns the value in the histogram bin referred to by bin (or channel) number (`x_bin_num` , `y_bin_num`). The result is undefined if id is not a two-dimensional histogram or bin_num is less than 1 or greater than the number of bins for the axis.

Calculations on histogram data

HS_1D_HIST_MINIMUM(id, x, bin_num, value**)**

INTEGER id *(input)*

REAL x *(returned)*

INTEGER bin_num *(returned)*

REAL value *(returned)*

Returns the x-coordinate (x), the bin number (bin_num), and the bin content (value) where the histogram data reaches a minimum. The routine returns a bin_num of 0 if the histogram does not refer to a one-dimensional histogram or the histogram is empty (i.e. has not yet been filled).

HS_2D_HIST_MINIMUM(id, x, y, x_bin_num, y_bin_num, value**)**

INTEGER id *(input)*

REAL x *(returned)*

REAL y *(returned)*

INTEGER x_bin_num *(returned)*

INTEGER y_bin_num *(returned)*

REAL value *(returned)*

Returns the x- and y-coordinate (x and y) the bin numbers (x_bin_num and y_bin_num), and the bin content (value) where the two-dimensional histogram data reaches a minimum. The routine returns a bin number of 0 if the histogram does not refer to a two-dimensional histogram, or the histogram is empty (i.e. has not yet been filled).

HS_1D_HIST_MAXIMUM(id, x, bin_num, value**)**

INTEGER id *(input)*

REAL x *(returned)*

INTEGER bin_num *(returned)*

REAL value *(returned)*

Returns the x-coordinate (x), the bin number (bin_num), and the bin content (value) where the histogram data reaches a maximum. The routine returns a bin_num of 0 if the histogram does not refer to a one-dimensional histogram or the histogram is empty (i.e. has not yet been filled).

HS_2D_HIST_MAXIMUM(id, x, y, x_bin_num, y_bin_num, value**)**

INTEGER id *(input)*

REAL x *(returned)*

REAL y *(returned)*

INTEGER x_bin_num *(returned)*

INTEGER y_bin_num *(returned)*

REAL value *(returned)*

Returns the x- and y-coordinate (x and y) the bin numbers (x_bin_num and y_bin_num), and the bin content (value) where the two-dimensional histogram data reaches a maximum. The routine returns a bin number of 0 if the histogram does not refer to a two-dimensional histogram, or the histogram is empty (i.e. has not yet been filled).

HS_1D_HIST_STATS(id, mean, std_dev**)**

INTEGER id *(input)*

REAL mean *(returned)*

REAL std_dev *(returned)*

Calculates and returns the mean and standard deviation of a one-dimensional histogram.

HS_2D_HIST_STATS(id, x_mean, y_mean, x_std_dev, y_std_d**)**

INTEGER id *(input)*

REAL x_mean *(returned)*

REAL y_mean *(returned)*

REAL x_std_dev *(returned)*

REAL y_std_dev *(returned)*

Calculates and returns the mean and standard deviation in x and y of a two-dimensional histogram.

integrl **= HS_HIST_INTEGRAL(**id**)**

INTEGER id *(input)*

REAL integrl *(returned)*

Calculates the integral (the sum of the bin contents * binwidth) of the bin data in a histogram. Overflow data is not counted. The result returned is -1.0 if id does not exist or does not refer to a one-dimensional histogram.

HS_HIST_SET_GAUSS_ERRORS(id**)**

INTEGER id *(input)*

Calculate and store Gaussian errors in a histogram. If positive errors have been previously stored, they will be disregarded. The new errors will be stored in the "positive" error array. The errors are computed assuming the central limit theorem: they are equal to the square root of the bin contents. If one or more bin value is negative, no errors are created or changed, and an error message is printed. If the item referred by id is not a one-dimensional or a two-dimensional histogram, no action is taken besides printing an error message.

new_id **= HS_SUM_HISTOGRAMS(**uid, title, category, id1, id2, const1, const2**)**

INTEGER uid *(input)*

CHARACTER ***n title *(input, n ≤ 80)*

CHARACTER ***n category *(input, n ≤ 255)*

INTEGER id1, id2 *(input)*

REAL const1, const2 *(input)*

INTEGER new_id *(returned)*

Creates a new histogram (one-dimensional or two-dimensional) whose data is the sum, bin by bin, of two histograms specified by HistoScope identification numbers `id1` and `id2`. The bin content of the new histogram will be equal to `const1` * bin value of histogram id1 + `const2` * bin value of histogram id2. const1 or const2 can be 0 but not both. Also one of the id's can be 0, in which case the corresponding histogram will not be referenced and the new histogram will simply be the other histogram multiplied by its constant. Errors are propagated assuming that these errors are Gaussian ("Low" errors are ignored). `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the histogram window when it is displayed. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " ", or ' ') the histogram will appear in the top level category. The routine returns an error (an id of -1) if both histograms are nonexistent, the histogram types are different, or the number of bins, or low/upper edges are inconsistent. Returns the id of a new histogram containing the sum of the two histograms, or -1 if the operation could not be performed.

new_id **= HS_MULTIPLY_HISTOGRAMS(**uid, title, category, id1, id2, const**)**

INTEGER uid *(input)*

CHARACTER ***n title *(input, n ≤ 80)*

CHARACTER ***n category *(input, n ≤ 255)*

INTEGER id1, id2 *(input)*

REAL const *(input)*

INTEGER new_id *(returned)*

Creates a new histogram (one-dimensional or two-dimensional) whose data is the multiplication, bin by bin, of two histograms specified by `id1` and `id2`. The bin content of the new histogram will be equal to `const` * (bin value of histogram id1) * (bin value of histogram id2). If one of the id's is specified as 0, the other histogram will just be multiplied by the constant. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the new histogram. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. Returns the id of the new histogram containing the result of the operation, or -1 if the operation could not be performed. Both histograms specified should be of the same type and have the same number of bins and bin edges.

`new_id` **= HS_DIVIDE_HISTOGRAMS(**`uid, title, category, id1, id2, const`**)**

INTEGER `uid` *(input)*

CHARACTER ***n** `title` *(input, n ≤ 80)*

CHARACTER ***n** `category` *(input, n ≤ 255)*

INTEGER `id1, id2` *(input)*

REAL `const` *(input)*

INTEGER `new_id` *(returned)*

Create a new histogram (one-dimensional or two-dimensional) whose data is the division, bin by bin, of two histograms specified by id1 and id2. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the new histogram window when it is displayed. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. The bin content of the new histogram will be equal to `const` * (bin value of histogram id1) / (bin value of histogram id2). If the bin value of histogram id2 is 0, the value 1 is used in the computation. Returns `new_id` which is the id of a new histogram containing the result of the operation, or -1 if an error occurred.

`new_id` **= HS_1D_HIST_DERIVATIVE(**`uid, title, category, id`**)**

INTEGER `uid` *(input)*

CHARACTER ***n** `title` *(input, n ≤ 80)*

CHARACTER ***n** `category` *(input, n ≤ 255)*

INTEGER `id` *(input)*

INTEGER `new_id` *(returned)*

Create a new histogram (one-dimensional) whose data is the derivative of the histogram specified by `id`. This derivative is computed bin by bin:

Dh/db = (H(j) - H(j-1))/bin_size.

If the input histogram, specified by its HistoScope `id`, has error bars, the errors are computed assuming that there is no statistical correlation between bins and the error values are Gaussian. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the histogram. `category` is a string defining a hierarchical "location" for the histogram. Returns `new_id` which is the id of a new histogram containing this derivative.

Summing Groups of Histograms and Ntuples

The two routines, HS_SUM_CATEGORY and HS_SUM_FILE, are designed to allow the user to sum together the data items from two different runs of the same program. For example, if on one day, a program creates the following items:

category	uid	item type
mydata	101	1-d histogram
mydata	102	2-d histogram
mydata/ntuples	202	Ntuple

and writes these items to a file, the next day, he can use HS_SUM_FILE to sum today's one-dimensional histogram with its corresponding 1-d histogram on the file, sum today's two-dimensional histogram with its corresponding 2-d histogram on the file, and merge together the two Ntuples. The HS_SUM_FILE call for the above example would look as follows:

```
CALL HS_SUM_FILE(FILENAME, 'mydata', 'sum')
```

and would create the following new items:

category	uid	item type
sum	101	1-d histogram
sum	102	2-d histogram
sum/ntuples	202	Ntuple

These routines search the items in the category specified and match their uid's with the data items found in the file (for HS_SUM_FILE) or second category (for HS_SUM_CATEGORY) specified. An error message will be printed if the data types do not match when the data items' uid's do match. An error message is also printed if two histograms or Ntuples with matching uid's do not have the same properties, for instance a different number of bins or variables. The resulting summed data items are created in memory and are given the new category specified (result_category). Items other than histograms and n-tuples are ignored.

HS_SUM_CATEGORY(category1, category2, result_category**)**

CHARACTER ***n** category1, ***n** category2 *(input)*

CHARACTER ***n** result_category *(input)*

Create a collection of new histograms (one-dimensional or two-dimensional), and/or Ntuples, based on two existing categories of data items. All such items within these categories are considered, and it is assumed that the categories have parallel category trees below a top prefix (e.g. from a HS_READ_FILE call), the uids/subcategories and binning properties are identical. Items with uid .EQ. 0 are skipped. For instance, category1 can be specified as 'run23', category2 as 'run25', result_category as 'run23_25'. Then, if a histogram exists both in 'run23/tracking' and in 'run25/tracking' with identical uid's and static properties, a new histogram with the same uid will be created which is the sum, bin by bin, in category 'run23_25/tracking'. Since this routine is intended to accumulate statistics, no arbitrary constant is provided. category1 and Category2 are the top level categories of the categories to be summed. These cannot be NULL nor empty (Uncategorized), nor contain "...". result_category is the top category for the newly created category.

HS_SUM_FILE(filename, category, result_category**)**

CHARACTER ***n** filename *(input, n ≤ system maximum for filenames)*

CHARACTER ***n** category *(input, n ≤ 255)*

CHARACTER ***n** result_category *(returned)*

Read all of the items from filename, sum histograms and/or merge Ntuples with those existing under the top category specified by category and store all newly created items in a category specified by result_category. This can be the same as category. Thus, to sum a bunch a files, one can read the first one by calling HS_READ_FILE(FILE1, 'allsum') and loop over the others, calling repetitively HS_SUM_FILE(FILE*n*, 'allsum', 'allsum'). The items read from the file will not be kept in memory. A known deficiency of this routine is that it assumes that there is no top category already named "tmp_sum_file" or "tmp_sum_result".

Functions for accessing Ntuple data

num_var = **HS_NUM_VARIABLES(**id**)**

INTEGER id *(input)*

INTEGER num_var *(returned)*

Returns the number of variables (num_var) in the Ntuple referred to by id. Returns -1 if the id is not valid or not an Ntuple.

`num_entries = HS_NUM_ENTRIES(id)`

INTEGER id *(input)*

INTEGER num_entries (returned)

Returns the number of entries stored in the Ntuple referred to by `id`, or -1 if the id is not a valid item.

`length = HS_VARIABLE_NAME(id, column, name)`

INTEGER id *(input)*

INTEGER column *(input)*

CHARACTER *n name *(returned)*

INTEGER length *(returned)*

Returns the `name` and `length` of a given variable for the Ntuple specified by `id`. `column` is the variable's index (i.e., the column number) and `name` is the mnemonic name of this variable in the Ntuple.

`index = HS_VARIABLE_INDEX(id, name)`

INTEGER id *(input)*

CHARACTER *n name *(input)*

INTEGER index (returned)

Returns the `index` (i.e. the column number in the Ntuple) corresponding to a given variable specified by its mnemonic `name`, for a given Ntuple specified by `id`. An exact match is required between `name` and one of the variable's names defined in the Ntuple. If no match is found, returns -1. This routine does not check for multiple matches, it successfully completes at the first occurrence of name.

`value = HS_NTUPLE_VALUE(id, row, column)`

INTEGER id *(input)*

INTEGER row *(input)*

INTEGER column *(input)*

REAL value (returned)

Returns the value (`value`) in an Ntuple, at a specified `row` and `column` index. `value` is undefined (-1.0 is returned, however) if `id` does not exist or refers to an item other than an Ntuple.

`HS_NTUPLE_CONTENTS(id, data)`

INTEGER id *(input)*

REAL data(num_entries, num_variables) *(returned)*

Returns all of the data in an Ntuple, specified by `id`, in the form of a two-dimensional array named `data` which contains floating point values. The array must be dimensioned to the number of rows (entries) by the number of columns (elements or variables) in the Ntuple.

HS_ROW_CONTENTS(id, row, data**)**

INTEGER id *(input)*

INTEGER row *(input)*

REAL data**(**num_variables**)** *(returned)*

Returns the contents of a row (or entry) of Ntuple data specified by id and places it in the array data. Data must be dimensioned to the number of variables (elements) in the Ntuple. A row is the collection of values in a particular Ntuple entry, one value for each variable (A variable could also be called a column or an element).

HS_COLUMN_CONTENTS(id, column, data**)**

INTEGER id *(input)*

INTEGER column *(input)*

REAL data**(**num_entries**)** *(returned)*

Returns the contents of a column of Ntuple data specified by id and places it in the array data. A column, which can also be called an element or a variable, is the set of values taken by a single Ntuple variable for each entry in the Ntuple.

new_id **= HS_MERGE_ENTRIES(**uid, title, category, id1, id2**)**

INTEGER uid *(input)*

**CHARACTER ***n title *(input, n ≤ 80)*

**CHARACTER ***n category *(input, n ≤ 255)*

INTEGER id1, id2 *(input)*

INTEGER new_id *(returned)*

Creates a new Ntuple consisting of two existing Ntuples, specified by id1 and id2. These two Ntuples must have an identical number of variables. uid is the User Identification number for the newly created Ntuple. title defines the title for the new Ntuple. category is a string defining a hierarchical "location" for the histogram. Returns id, the Histo-Scope id of the newly created Ntuple, or -1 if the Ntuples referred by id1 or id2 don't exist, are not Ntuples or have a different number of variables. The variable names are taken from id1, and are assumed to be identical to those of id2.

C Binding Library Routines

Routines for all Users

```
hs_initialize(identity_string);
```

```
char *identity_string; (input)
```

Initialize the Histo-Scope connection software and set up a potential connection to a Histo-Scope process. The identity string identifies the process in the Histo-Scope connection panel and may be up to 40 characters long. This routine should be called before any other Histo-Scope routine.

```
hs_update();
```

This routine must be called periodically to keep the Histo-Scope display up to date and to accept connections from Histo-Scopes. You may call this as often as you like with no adverse impact on CPU usage. It will only transmit the data for the items that are currently displayed, and only as frequently as requested by the user of the Histo-Scope process, even if you call hs_update() very often.

```
hs_complete();
```

Close all connections with Histo-Scope processes and notify them that the program is finished. This routine gives connected Histo-Scope processes a chance to request a last update. This may take a few seconds to accomplish.

If you are using HBOOK, this routine should be called <u>before</u> calling hrend, since hs_complete() may still access the HBOOK data.

Under rare circumstances, an hs_complete() call can take up to 1 minute to complete if one or more Histo-Scope(s) are viewing a large amount of data and have somehow fallen behind in receiving the data. This call allows a maximum of one minute for sending all data requested by Histo-Scopes before the hs_complete() was called.

```
hs_complete_and_wait();
```

Notify any connected Histo-Scope processes that the program is finished and is waiting for them to exit or disconnect. While any Histo-Scope is connected, this call will not return to the calling program. Instead all Histo-Scope requests, including new connections will be serviced. After all connected Histo-Scopes have disconnected, all data memory is freed and control is returned to the calling program. As noted above, this call also should be called before calling the HBOOK routine hrend.

hs_histoscope(return_immediately**);**

int return_immediately; *(input)*

For users who want to build Histo-Scope into their program rather than using it separately as an inspection tool. hs_histoscope invokes Histo-Scope as a sub-process. The scope process is pre-connected to display the data generated in the user process. The routine can either start the Histo-Scope and return immediately (`return_immediately` != 0), or it can start it and return after the user closes the Histo-Scope window (`return_immediately` == 0).

hs_histo_with_config(return_immediately, config_file**)**

int return_immediately; *(input)*

**char ***config_file *(input)*

For users who want to build Histo-Scope into their program **and** have it automatically display plots. hs_histo_with_config invokes Histo-Scope as a sub-process and displays the plots specified in the configuration file `config_file`. The scope process is pre-connected to display the data generated in the user process. The routine can either start the Histo-Scope and return immediately (`return_immediately` != 0), or it can start it and return after the user closes the Histo-Scope window (`return_immediately` == 0).

hs_kill_histoscope()

This routine allows you to "kill" any Histo-Scope processes started via the hs_histoscope or hs_histo_with_config calls. The "kill" is accomplished through a message request to the Histo-Scope process for it to exit.

num_scopes **= hs_num_connected_scopes()**

int num_scopes; *(returned)*

Returns the number of connected Histo-Scopes. This routine is useful for checking that a Histo-Scope created with hs_histoscope has successfully begun executing and is connected to its client program. Since this takes some time, it is suggested that hs_num_connected_scopes be called to check that a scope is connected before calling hs_load_config_file or hs_load_config_string.

hs_load_config_file(config_file**)**

**char ***config_file *(input)*

Calling hs_load_config_file causes all connected Histo-Scopes to read the configuration file specified by `config_file` and execute all of the commands to display plots in the file.

hs_load_config_string(config_string**)**

char *config_string *(input)*

Calling hs_load_config_string causes all connected Histo-Scopes to execute all of the commands to display plots in the string specified by `config_string`.

Routines for HBOOK users

hs_hbook_setup(top_directory**);**

char *top_directory; *(input)*

Call this routine after you book the histograms or Ntuples you will use in the program and after you call hs_initialize(). The routine will set up all current HBOOK histograms and Ntuples in the specified top directory for use with Histo-Scope. The `top_directory` string supplied should begin with "//". (**Note:** unpredictable results may occur if this routine is called without first calling the appropriate HBOOK routines for the top directory.)

hs_reset_hbook_setup(top_directory**);**

char *top_directory; *(input)*

Call this routine if you have booked new histograms or Ntuples, or deleted, renamed, rebinned, or re-set existing ones since calling hs_hbook_setup(). hs_reset_hbook_setup will reset Histo-Scope's list for the specified top directory. The `top_directory` string supplied should begin with "//".

Histo-Scope data routines

The Histo-Scope data routines provide a much simpler interface than HBOOK. They can be used instead of HBOOK routines or in addition to them. Many of these routines have common parameters that have limitations on their length. `title` should be less than or equal to 80 characters, `category` should be less than or equal to 255 characters, `filename` should be less than or equal to the system maximum for filenames, and `prefix` should be less than or equal to 255 characters.

```
id = hs_create_1d_hist(uid, title, category, x_label, y_label,
                       n_bins, min, max);
```

int uid; *(input)*

char *title; *(input)*

char *category; *(input)*

char *x_label; *(input)*

char *y_label; *(input)*

int bins; *(input)*

float min; *(input)*

float max; *(input)*

int id; *(returned)*

Create (or book) a one-dimensional histogram. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the histogram window when it is displayed. `category` is an optional string (i.e. can be NULL or empty) defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the histogram will appear in the top level category as uncategorized. The histogram axes are given the labels `x_label` and `y_label`. `n_bins` is the number of bins in the histogram. Any values ≤ `min` or ≥ `max` are collected into under- and over-flow bins. The histoscope-defined `id` for the histogram is returned to the user and is used in later calls to identify the histogram just created. If `id` is ≤ 0, an error occured in creating the histogram (check the arguments supplied to the function). The value of `min` should be less than `max`, and `bins` should be greater than 0.

```
id = hs_create_2d_hist(uid, title, category, x_label, y_label,
                        z_label, x_bins, y_bins, x_min, x_max, y_min,
                        y_max);
```

int uid; *(input)*

char *title; *(input)*

char *category; *(input)*

char *x_label; *(input)*

char *y_label; *(input)*

char *z_label; *(input)*

int x_bins; *(input)*

int y_bins; *(input)*

float x_min; *(input)*

float x_max; *(input)*

float y_min; *(input)*

float y_max; *(input)*

int id; *(returned)*

Create (or book) a two-dimensional histogram. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the histogram window when it is displayed. `category` is an optional (can be NULL or empty) string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the histogram will appear in the top level category as uncategorized. The histogram axes are given the labels `x_label`, `y_label`, and `z_label`. `x_bins` and `y_bins` are the number of bins for each axis of the histogram. Any values ≤ `min` or ≥ `max` are collected into under- and over-flow bins. The Histo-Scope-defined `id` for the histogram is returned and is used in later calls to identify the histogram just created. If `id` is ≤ 0, an error occurred in creating the histogram (check the arguments supplied to the function). The value of x_min and y_min should be less than its corresponding maximum, and x_bin and y_bin should both be greater than 0.

id = **hs_create_ntuple(**uid, title, category, n_variables, names**);**

int uid; *(input)*

char ***title; *(input)*

char ***category; *(input)*

int n_variables; *(input)*

char ***names**[**n_variables**];** *(input)*

int id; *(returned)*

Defines an Ntuple with `n_variables` variables. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the Ntuple, and `category` is an optional (i.e. can be NULL or empty) string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the Ntuple will appear in the top level category as uncategorized. `names` is a pointer to an array of pointers to null-terminated strings naming each Ntuple variable. Storage allocation is automatic: as the Ntuple grows, more space will be allocated. However, **Ntuples must fit into virtual memory.** The Histo-Scope-defined `id` for the Ntuple is returned and is used in later calls to identify the Ntuple just created. If `id` is ≤ 0, an error occurred in creating the Ntuple (check the arguments supplied to the function).

id = **hs_create_indicator(**uid, title, category, min, max**);**

int uid; *(input)*

char ***title; *(input)*

char ***category; *(input)*

float min; *(input)*

float max; *(input)*

int id; *(returned)*

Creates an indicator. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the indicator, and `category` is an optional string (can be NULL or an empty string) defining a hierarchical "location" for the indicator. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the indicator will appear in the top level category. `min` and `max` is the minimum and maximum values for the indicator. The Histo-Scope-defined `id` for the indicator is returned and is used in later calls to identify the indicator just created. If `id` is <= 0, an error occurred in creating the indicator (check the arguments supplied to the function).

```
id = hs_create_control(uid, title, category, min, max, default_value);
    int uid; (input)
    char *title; (input)
    char *category; (input)
    float min; (input)
    float max; (input)
    float default_value; (input)
    int id;  (returned)
```

Creates a control. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the control, and `category` is an optional string (can be NULL or an empty string) defining a hierarchical "location" for the control. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the control will appear in the top level category. `min` and `max` is the minimum and maximum values for the control. The Histo-Scope user will not be able to set the control outside of this range. The Histo-Scope-defined `id` for the control is returned and is used in later calls to identify the control just created. If `id` is <= 0, an error occurred in creating the control (check the arguments supplied to the function).

```
id = hs_create_trigger(uid, title, category);
    int uid; (input)
    char *title; (input)
    char *category; (input)
    int id; (returned)
```

Creates a trigger. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `title` defines the title for the trigger, and `category` is an optional string (can be NULL or an empty string) defining a hierarchical "location" for the trigger. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "", or NULL) the trigger will appear in the top level category. The Histo-Scope-defined `id` for the trigger is returned and is used in later calls to identify the trigger just created. If `id` is <= 0, an error occurred in creating the trigger (check the arguments supplied to the function).

```
id = hs_create_group(uid, title, category, groupType, numItems, itemId,
   ersDisp)
```

int uid; *(input)*

char *title; *(input)*

char *category; *(input)*

int groupType; *(input)*

int numItems; *(input)*

int itemId[numItems]; *(input, numItems <= 81)*

int errsDisp[numItems]; *(input, numItems <= 81)*

int id; *(returned)*

Creates a group , which is an item that points to other data items, thus grouping them into one addressable entity. `uid` defines a user-specified identification number which can later be used along with the category to specify the item created. `category` is an optional string (can be NULL or an empty string) defining a hierarchical "location" for the group. Subcategories can be specified using the "/" character as in Unix file specifications. If a category or subcategory does not exist, it is created. If the category argument is missing (either "'', or NULL) the group will appear in the top level category. The Histo-Scope-defined id for the group is returned and is used in later calls to identify the group just created. If `id` is <= 0, an error occured in creating the group (check the arguments supplied to the function). `groupType` is the group's suggested window type. The number of items in this group, specified by `numItems`, should not be larger than 81. `itemId` is an array of Histo-Scope id numbers for the items in the group. `errsDisp` is an array of values (NO_ERROR_BARS, DATA_ERROR_BARS, GAUSSIAN_ERROR_BARS) that corresponds to the itemId array and tells Hist-Scope how to display errors for the corresponding histogram (used only when the corresponding data item is a histogram).

A group can be made out of a number of items in order to display them as a Multiple Plot Window, an Overlaid Plot, or simply as a convenience for putting plots up individually with fewer mouse clicks. Data items may be put into more than one group. Including an item in the group does not effect the display of the item individually in the Histo-Scope main panel. The item still appears in the main panel list box and can be viewed individually. The category string can be used to affect how Histo-Scope items appear in the main panel.

When a group is selected in Histo-Scope's main window and the user presses the "View" Button, the groupType field tells Histo-Scope how to display the items in the group. This can be overridden in the Histo-Scope main panel by using the “View Multiple” or “View Overlaid” buttons. If groupType is HS_MULTI_PLOT, pressing the View button displays all the 1- and 2D histograms in the group as one Multi-plot window. Similarly, if you would like to see all the 1D histograms in a group displayed over each other in one plot window, you can specify a groupType of HS_OVERLAY_PLOT. Triggers, controls, and indicators cannot appear inside a Multiple Plot Window or Overlaid Plot and so these items in a group are displayed in their own windows.

Specifying HS_INDIVIDUAL for groupType means that if a group is selected, and the “View” button is pressed on the Histo-Scope main panel, each data item in the group will be viewed individually. In the case of Ntuples, the Ntuple panel will be displayed so the the user can choose the variables to plot and the type to use.

Filling Histograms/Ntuples and Setting Errors

hs_fill_1d_hist(`id, x, weight`**);**

int `id;` *(input)*

float `x;` *(input)*

float `weight;` *(input)*

Adds a value (`weight`) to the one-dimensional histogram identified by `id`.

hs_fill_2d_hist(`id, x, y, weight`**);**

int `id;` *(input)*

float `x;` *(input)*

float `y;` *(input)*

float `weight;` *(input)*

Adds a value (`weight`) to the two-dimensional histogram identified by `id`.

hs_1d_hist_block_fill(`id, data, pos_errors, neg_errors`**);**

int `id;` (*input*)

float `*data;` (*input*)

float `*err;` (*input*)

float `*err_m;` (*input*)

Replaces the accumulated bin and error data supplied for the one-dimensional histogram identified by `id` and clears the overflow bins. `data` is a one-dimensional array containing the new data for the histogram and must be of the appropriate size (i.e. the correct number of bins) for the one-dimensional histogram being filled. `pos_errors` and `neg_errors` are the errors of the one-dimensional histogram.

Either error argument can be specified as NULL, in which case the corresponding errors are not changed. `pos_errors` and `neg_errors` should have the same number of elements as the histogram has bins. If the histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** floating numbers.

`hs_2d_hist_block_fill(``id, data, pos_errors, neg_errors`**`);`**

`int` `id;` *(input)*

`float` `*data;` *(input)*

`float` `*err;` *(input)*

`float` `*err_m;` *(input)*

Replaces the accumulated bin and error data supplied for the two-dimensional histogram identified by `id` and clears the overflow bins. `data` is a two-dimensional array containing the new data for the histogram and must be of the appropriate size (i.e. the correct number of bins) for the two-dimensional histogram being filled. `pos_errors` and `neg_errors` are the errors of the two-dimensional histogram. Either error argument can be specified as NULL, in which case the corresponding errors are not changed. `pos_errors` and `neg_errors` should have the same number of elements as the histogram has bins. If the histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both `pos_errors` and `neg_errors` should be specified as **positive** floating numbers.

`hs_set_1d_errors(``id, pos_errors, neg_errors`**`);`**

`int` `id;` *(input)*

`float` `*pos_errors;` *(input)*

`float` `*neg_errors;` *(input)*

Copies one or both arrays of real numbers representing the errors of a one-dimensional histogram. Either array pointer can be NULL, in which case the corresponding errors are not set. Arrays should have the same number of elements as the histogram has bins. If a histogram has positive errors, but not negative ones aren't, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying only positive errors when errors are symmetric. Both pos_errors and neg_errors should be specified as **positive** floating numbers.

`hs_set_2d_errors(``id, pos_errors, neg_errors`**`);`**

`int` `id;` *(input)*

`float` `*pos_errors;` *(input)*

`float` `*neg_errors;` *(input)*

Copies one or both arrays of real numbers representing the errors of a two-dimensional histogram. Either array pointer can be NULL, in which case the corresponding errors are not set. Arrays should have the same number of elements in each dimension as the histogram has bins. If a histogram has positive errors, but not negative ones, Histo-Scope assumes errors are symmetric. Thus memory can be saved by specifying an array of positive errors and NULL as the negative errors when errors are symmetric. Both pos_errors and neg_errors should be specified as **positive** floating numbers.

`status = `**`hs_fill_ntuple(`**`id, values`**`[`**`n`**`]);`**

`int` `id;` *`(input)`*

`float *``values;` *`(input)`*

`int` `status;` *`(returned)`*

Adds an array of real `values`, in the order they are named in HS_CREATE_NTUPLE, to an Ntuple identified by `id`. Returns `istatus`. Check this status to ensure the array was added and to be certain your Ntuple still fits within the virtual memory of your machine. `istatus` ≤ 0 indicates memory was exhausted. `istatus` == 0 indicates an incorrect id or other user error. `istatus` ≥ 0 indicates success. The Ntuple id number is returned upon success.

Setting Indicators, Reading Controls, and Checking Triggers

`hs_set_indicator(``id, value`**`);`**

`int` `id;` *`(input)`*

`float` `value;` *`(input)`*

Sets the value of an indicator to `value`. The value should be within the range specified for the indicator. <u>However the indicator is set even if the value is out of range.</u>

`hs_read_control(``id, ret_value`**`);`**

`int` `id;` *`(input)`*

`float *``ret_value;` *`(returned)`*

Returns the current value of a control.

`status = `**`hs_check_trigger(`**`id`**`);`**

`int` `id;` *`(input)`*

`int` `status;` *`(returned)`*

Returns 1 if the trigger identified by `id` has been set by a Histo-Scope user. Otherwise 0 is returned. Please note that this routine will return 1 once for each button press by the Histo-Scope user. Thus, if the user has pressed the trigger 5 times, the program must call `hs_check_trigger` at least 5 times.

Functions for looking up items:

```
id = hs_id(uid, category);
    int uid; (input)
    char *category; (input)
    int id; (returned)
```

Returns the Histo-Scope id for a specified uid and category pair, or -1 if a match was not found. `uid` is the user-specified identification number which was specified when the item was created. `category` is a string defining a hierarchical "location" for the indicator also specified when the item was created. The Histo-Scope defined id is returned.

```
id = hs_id_from_title(title, category);
    char *title;(input)
    char *category; (input)
    int id; (returned)
```

Returns the Histo-Scope id for a specified item title and category pair, or -1 if a match was not found. `title` is the title specified for the item when it was created. `category` is the string specified at item creation defining a hierarchical "location" for the item.

```
num_match = hs_list_items(title, category, ids, num, matchFlg);
    char *title; (input)
    char *category; (input)
    int *ids; (returned)
    int num; (input)
    int matchFlg; (input)
    int num_match; (returned)
```

Fills a list of Histo-Scope id numbers. The user must allocate at least `num` integer elements for the array `ids`. These ids are returned in the order in which the histograms were created, or read in from a file. Returns the number of matching title/category strings, `num_match`, which may be larger than `num`, the maximum number of items that can be returned. `title` specifies the title of the data item(s) to list. `category` is a string defining a hierarchical "location" for the indicator. Optionally, the category can include a trailing "..." which will return the id numbers for items in subcategories of the specified category. For example, "HS/..." matches "HS" and "HS/Nts" but not "HSNTuples", "HSNTuples" nor "hs". If the category parameter is specified as "..." all categories will match including uncategorized items. `match_flag` specifies exact or inexact matches for the title string specified. If match_flag .EQ. 0, then all id's returned will exactly match the title specified. If match_flag .NE. 0, then leading spaces are ignored and id's *containing* the specified title are returned. Specifying "" for title and an inexact match for match_flag will match all titles. Specifying NULL for title and an exact match will match all uncategorized items. If the category supplied is not valid, -1 will be returned in num_match.

Functions for getting properties of all items

`id_user = `**`hs_uid(`**`id`**`);`**

`int` `id; (`*`input`*`)`

`int` `id_user; (`*`returned`*`)`

Returns the user-specified identification number (uid) value for a given Histo-Scope `id`, or 0 if id was invalid or not found.

`length = `**`hs_category(`**`id, category`**`);`**

`int` `id; (`*`input`*`)`

`char *``category; (`*`input`*`)`

`int` `length; (`*`returned`*`)`

Returns the `category` string and its `length` for a given Histo-Scope `id`, or -1 if id was invalid.

`length = `**`hs_title(`**`id, title_string`**`);`**

`int` `id; (`*`input`*`)`

`char *``title; (`*`input`*`)`

`int` `length; (`*`returned`*`)`

Returns the title and its `length` for an item specified by id, or -1 if id was invalid.

`data_type = `**`hs_type(`**`id`**`);`**

`int` `id; (`*`input`*`)`

`int` `data_type; (`*`returned`*`)`

Returns the type of data, `data_type`, referred to by `id`. Valid types are:

HS_1D_HISTOGRAM	HS_2D_HISTOGRAM	HS_NTUPLE
HS_INDICATOR	HS_CONTROL	HS_TRIGGER
HS_NONE		

HS_NONE refers to a non-existent item.

These values are defined in histoscope.inc.

Functions for reading/writing items from/to files

```
istatus = hs_read_file(filename, prefix);
    char *filename; (input)
    char *prefix; (input)
    int num_items; (returned)
```

Reads all of the items from the Histo-Scope formatted file specified by `filename`. All category strings will be prefixed by a new top level category specified in `prefix`. This may be NULL (or " " (spaces)), meaning no prefix is added. Returns the number of items read; otherwise returns -1 and prints an error message.

```
num_items = hs_read_file_items(filename, prefix, category, uids,
    n_uids);
    char *filename; (input)
    char *prefix; (input)
    char *category; (input)
    int *uids; (input)
    int n_uids; (input)
    int num_items; (returned)
```

Selectively reads items from a specific category and all its subcategories that match a list of `uids` from the Histo-Scope formatted file specified by `filename`. All category strings read from the file will be prefixed by a new top level category specified in `prefix`. This may be specified as %VAL(0) (or " "), meaning that no prefix should be added. If an item with the same (prefixed) category and uid already exists, the item in the file will not be read and an error message will inform the user. If n_uids is > 0, the array `uids` specifies the uid's of items to read from the file. If `uids` is %VAL(0) or n_uids == 0, all items in the specified category and its subcategories will be read. `n_uids` specifies the number of items in array uids. Returns the number of items read, or -1 if an error occurred.

For example, if "HS" is specified for `category` and "Run1" for `prefix`, items read from the file with category "HS" will become "Run1/HS" and items read from the file with category "HS/Ntuples" will become "Run1/HS/Ntuples". The new (prefixed) category must be less than 255 characters in length, or the item will not be read from the file. To read non-selectively regardless of the category, specify " " as `category`.

```
num_items = hs_save_file(filename);
```

char *filename; *(input, n ≤ system maximum for filenames)*

int num_items; *(returned)*

Saves all current histograms, Ntuples, indicators, and controls created by Histo-Scope library routines or read from Histo-Scope formatted files to a Histo-Scope-format file. Triggers are not written to the file. The number of items written is returned in `num_items`, or -1 if an error occurred. The named file is open for writing ("w"), discarding previous contents, if any.

```
num_items = hs_save_file_items(name, category, uids, n_uids);
```

char *filename; *(input)*

char *category; *(input)*

int *uids; *(input)*

int n_uids; *(input)*

int num_items; *(returned)*

Selectively saves the histograms, Ntuples, indicators, and controls in a specified `category` that match a list of `uids` to a Histo-Scope format file `filename`. Items will be written only when they match the specified category *exactly*. Specify “ “ or NULL as the `category` to save only uncategorized items. `uids` is an array of user id values of length `n_uids` specifying which items to write. If n_uids is ≤ 0 or uids is NULL, all items in the category specified are saved. The number of items written is returned, or -1 if an error occurs. The file named is open for writing ("w"), discarding previous contents, if any.

Functions for managing data items

```
hs_reset(id);
```

int id; *(input)*

Resets the data item. For histograms, all of the bins and overflows are set to 0. For Ntuples all of the data is removed. Indicators are set to *not set*. Controls are set to their default value. For triggers, any pending trigger presses are cleared.

```
hs_delete(id);
```

int id; *(input)*

Deletes a histogram, Ntuple, indicator, control, or trigger identified by `id`.

hs_delete_category(category**);**

char *category; *(input)*

Deletes all items in the named category. All data content is lost. The category string must *exactly* match the category of the item to be deleted *unless a trailing* "..." is specified as part of the category. When the "..." is specified, data items in the category **and all subcategories** of the category will be deleted. Specifying " " or NULL for category will delete all uncategorized items. For example, specifying a category "HS/..." will delete all items with category "HS" and "HS/Nts" but not "HSNTuples" nor "hs". **If just "..." is specified, all items are deleted.**

hs_change_uid(id, newuid**);**

int id; *(input)*

int newuid; *(input)*

Changes the user-defined identification number (uid) to new_uid for an item specified by its Histo-Scope identification number, id. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new uid.

hs_change_category(id, newcategory**);**

int id; *(input)*

char *newcategory; *(input)*

Changes the category of an item specified by its Histo-Scope identification number, id. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new category.

hs_change_title(id, newtitle**);**

int id; *(input)*

char *newtitle; *(input)*

Changes the title of an item specified by its Histo-Scope identification number, id. If a Histo-Scope is connected when this routine is called, the Histo-Scope main window will be updated with the new title.

num_items **= hs_num_items();**

int num_items; (*returned*)

Returns the number of Histo-Scope data items defined so far.

Functions for accessing histogram data

```
num_bins = hs_1d_hist_num_bins(id);
    int id; (input)
    int num_bins; (returned)
```

Returns the number of bins for a one-dimensional histogram identified by id, or -1 if the id does not refer to a one-dimensional histogram.

```
hs_2d_hist_num_bins(id, num_x_bins, num_y_bins);
    int id; (input)
    int *num_x_bins; (returned)
    int *num_y_bins; (returned)
```

Returns the number of bins in the two-dimensional histogram specified by `id`. `num_x_bins` is the number of bins in X. `num_y_bins` is the number of bins in Y. -1 is returned for num_x_bins and num_y_bins if id does not refer to a two-dimensional Histogram.

```
hs_1d_hist_range(id, min, max);
    int id; (input)
    float *min; (returned)
    float *max; (returned)
```

Returns the minimum (low edge of first bin) value in `min` and maximum (high edge og last bin) value in `max` representing the horizontal limits of a one-dimensional histogram. If id does not refer to a one-dimensional histogram, zero is returned for min and max.

```
hs_2d_hist_range(id, x_min, x_max, y_min, y_max);
    int id; (input)
    float *x_min; (returned)
    float *x_max; (returned)
    float *y_min; (returned)
    float *y_max; (returned)
```

Returns the minimum and maximum values representing the horizontal limits of a two-dimensional histogram specified by id. If id does not refer to a two-dimensional histogram, zero is returned for min and max.

`num_entries = `**`hs_num_entries(`**`id`**`);`**

`int` `id;` *`(input)`*

`int` `num_entries; (returned)`

Returns the number of fill operations that have been performed on the histogram referred to by `id`, or -1 if the id does not refer to a histogram or Ntuple item.

`hs_1d_hist_bin_contents(``id, data`**`);`**

`int` `id;` *`(input)`*

`float *``data;` *`(returned)`*

Returns the bin data from a one-dimensional histogram. The data array must be large enough and be dimensioned appropriately for the number of bins in the histogram.

`hs_2d_hist_bin_contents(``id, data`**`);`**

`int` `id;` *`(input)`*

`float *``data;` *`(returned)`*

Returns the bin data from a two-dimensional histogram. The data array must be large enough and be dimensioned appropriately for the number of bins in the histogram.

`status = `**`hs_1d_hist_errors(`**`id, pos_errors, neg_errors`**`);`**

`int` `id;` *`(input)`*

`float *``pos_errors;` *`(returned)`*

`float *``neg_errors;` *`(returned)`*

Return the error bar data from a one-dimensional histogram. `pos_errors` is a one-dimensional array to receive data for upper error bars. Must be dimensioned appropriately for the number of bins in the histogram. `neg_errors` is a one-dimensional array to receive data for lower error bars. neg_errors can be NULL which means only positive errors are returned. If non NULL, neg_errors must be dimensioned appropriately for the number of bins in the histogram. The status returned can be:

HS_NO_ERRORS - Item has no error data

HS_POS_ERRORS - Positive errors (only) are returned

HS_BOTH_ERRORS - Positive and negative errors are returned

HS_ITEMNOTFOUND_ERRORS - id does not refer to a one-dimensional histogram

These values are defined in histoscope.inc.

```
status = hs_2d_hist_errors(id, pos_errors, neg_errors);
    int id; (input)
    float *pos_errors(x_bins, y_bins); (returned)
    float *neg_errors(x_bins, y_bins); (returned)
```

Returns the error bar data from a two-dimensional histogram. `pos_errors` is a two-dimensional array to receive data for upper error bars. Must be dimensioned appropriately for the number of bins in the histogram. `neg_errors` is a two-dimensional array to receive data for lower error bars. neg_errors can be NULL, which means only positive errors are returned. If not NULL, neg_errors must be dimensioned appropriately for the number of bins in the histogram. The status returned can be:

HS_NO_ERRORS - Item has no error data

HS_POS_ERRORS - Positive errors (only) are returned

HS_BOTH_ERRORS - Positive and negative errors are returned

HS_ITEMNOTFOUND_ERRORS - id does not refer to a one-dimensional histogram

These values are defined in histoscope.inc.

```
hs_1d_hist_overflows(id, underflow, overflow);
    int id; (input)
    float *underflow; (returned)
    float *overflow; (returned)
```

Returns the overflow data from a one-dimensional histogram.

```
hs_2d_hist_overflows(id, overflows);
    int id; (input)
    float *overflows; (returned)
```

Returns the overflow data from a two-dimensional histogram. This is a 3 x 3 array:

[0][0] -X +Y	[1][0] +Y	[2][0] +X +Y
[0][1] - X		[2][1] +X
[0][2] -X -Y	[1][2] - Y	[2][2] +X -Y

Figure 48

```
value = hs_1d_hist_x_value(id, x);
   int id; (input)
   float x; (input)
   float value; (returned)
```

Returns the value stored in the histogram bin referenced by x. The result is undefined if id is not a one-dimensional histogram, or if x is not within the range of the histogram.

```
value = hs_2d_hist_xy_value(id, x, y);
   int id; (input)
   float x; (input)
   float y; (input)
   float value; (returned)
```

Returns the value stored in the histogram bin referenced by x and y. The result is undefined if id is not a two-dimensional histogram, or if x or y are not within the range of the histogram.

```
value = hs_1d_hist_bin_value(id, bin_num);
   int id; (input)
   int bin_num; (input)
   float value; (returned)
```

Returns the value in the histogram bin referred to by bin (or channel) number `bin_num`. The result is undefined if id is not a one dimensional histogram or bin_num is less than 1 or greater than the number of bins.

```
value = hs_2d_hist_bin_value(id, x_bin_num, y_bin_num);
   int id; (input)
   int x_bin_num; (input)
   int y_bin_num; (input)
   float value; (returned)
```

Returns the value in the histogram bin referred to by bin (or channel) number `x_bin_num, y_bin_num`. The result is undefined if id is not a two-dimensional histogram or if either bin number is less than 1 or greater than the number of bins for the axis.

Calculations on histogram data

```
hs_1d_hist_minimum(id, x, bin_num, value);
    int id; (input)
    float *x; (returned)
    int *bin_num; (returned)
    float *value; (returned)
```

Returns the x-coordinate (x) the bin number (`bin_num`) and the bin content (`value`) where the histogram data reaches a minimum. The routine returns a bin_num of 0 if the histogram does not refer to a one-dimensional histogram or the histogram is empty (i.e. has not yet been filled).

```
hs_2d_hist_minimum(id, x, y, x_bin_num, y_bin_num, value);
    int id (input)
    float *x; (returned)
    float *y; (returned)
    int *x_bin_num; (returned)
    int *y_bin_num; (returned)
    float *value; (returned)
```

Returns the x- and y-coordinate (`x and y`) the bin numbers (`x_bin_num and y_bin_num`), and the bin content (`value`) where the two-dimensional histogram data reaches a minimum. The routine returns a bin number of 0 if the histogram does not refer to a two-dimensional histogram, or the histogram is empty (i.e. has not yet been filled).

```
hs_1d_hist_maximum(id, x, bin_num, value);
    int id; (input)
    float *x; (returned)
    int *bin_num; (returned)
    float *value; (returned)
```

Returns the x-coordinate (`x`) the bin number (`bin_num`), and the bin content (`value`) where the histogram data reaches a maximum. The routine returns a bin_num of 0 if the histogram does not refer to a one-dimensional histogram or the histogram is empty (i.e. has not yet been filled).

```
hs_2d_hist_maximum(id, x, y, x_bin_num, y_bin_num, value);
    int id; (input)
    float *x; (returned)
    float *y; (returned)
    int *x_bin_num; (returned)
    int *y_bin_num; (returned)
    float *value; (returned)
```

Returns the x- and y-coordinate (`x` and `y`) the bin numbers (`x_bin_num` and `y_bin_num`), and the bin content (`value`) where the two-dimensional histogram data reaches a maximum. The routine returns a bin_num of 0 if the histogram does not refer to a two-dimensional histogram, or the histogram is empty (i.e. has not yet been filled).

```
hs_1d_hist_stats(id, mean, std_dev);
    int id; (input)
    float *mean; (returned)
    float *std_dev; (returned)
```

Calculates and returns the mean and standard deviation of a one-dimensional histogram.

```
hs_2d_hist_stats(id, x_mean, y_mean, x_std_dev, y_std_dev);
    int id; (input)
    float *x_mean; (returned)
    float *y_mean; (returned)
    float *x_std_dev; (returned)
    float *y_std_dev; (returned)
```

Calculates and returns the mean and standard deviation of a two-dimensional histogram.

```
integrl = hs_hist_integral(id);
    int id; (input)
    float integrl; (returned)
```

Calculates the integral (the sum of the bin contents * binwidth) of the bin data in a histogram. Overflow data is not counted. The result returned is -1.0 if id does not exist or does not refer to a one-dimensional histogram.

hs_hist_set_gauss_errors(id**);**

int id; *(input)*

Calculate and store Gaussian errors in a histogram. If positive errors have been previously stored, they will be disregarded. The new errors will be stored in the "positive" error array. The errors are computed assuming the central limit theorem: they are equal to the square root of the bin contents. If one or more bin value is negative, no errors are created or changed, and an error message is printed. If the item referred by id is not a one-dimensional or a two-dimensional histogram, no action is taken besides printing an error message.

new_id **= hs_sum_histograms(**uid, title, category, id1, id2, const1, const2**);**

int uid; *(input)*

char *title, *category; *(input)*

int id1, id2; *(input)*

float const1, const2; *(input)*

int new_id; *(returned)*

Creates a new histogram (one-dimensional or two-dimensional) whose data is the sum, bin by bin, of two histograms specified by HistoScope identification numbers `id1` and `id2`. Each bin value of the new histogram will be equal to `const1` * bin value of histogram id1 + `const2` * bin value of histogram id2. const1 or const2 can be 0., or one of the id's can also be set to 0, in which case the corresponding histogram will not be referenced and the new histogram will simply be the other histogram multiplied by its constant. Errors are propagated assuming that these errors are Gaussian (i.e. “negative” errors are ignored). `uid` is the User Identification for the newly created histogram. `title` defines the title for the new histogram. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. The routine returns an error (a new_id of -1) if both histograms do not exist, the histogram types are different, or the number of bins or low/upper edges are inconsistent. Returns the id of the new histogram containing the sum of the two histograms, or -1 if the operation could not be performed.

new_id **= hs_multiply_histograms(**uid, title, category, id1, id2, constant**);**

int uid; *(input)*

char *title, *category; *(input)*

int id1, id2; *(input)*

```
float constant; (input)
int new_id; (returned)
```

Creates a new histogram (one-dimensional or two-dimensional) whose data is the multiplication, bin by bin, of two histograms specified by `id1` and `id2`. The bin content of the new histogram will be equal to `constant` * (bin value of histogram id1) * (bin value of histogram id2). If one of the id's is specified as 0, the other histogram will just be multiplied by the constant. `uid` is the User Identification for the newly created histogram. `title` defines the title for the new histogram. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. Returns the id of the new histogram containing the result of the operation, or -1 if the operation could not be performed. Both histograms specified should be of the same type and have the same number of bins and bin edges.

```
new_id = hs_divide_histograms(uid, title, category, id1, id2,
    constant);
    int uid; (input)
    char *title, *category; (input)
    int id1, id2; (input)
    float constant; (input)
    int new_id; (returned)
```

Create a new histogram (one-dimensional or two-dimensional) whose data is the division, bin by bin, of two histograms specified by id1 and id2. `uid` is the User Identification for the newly created histogram. `title` defines the title for the new histogram window when it is displayed. `category` is a string defining a hierarchical "location" for the histogram. Subcategories can be specified using the "/" character as in UNIX file specifications. If a category or subcategory does not exist, it is created. If the category argument is blank (either " '', or ' ') the histogram will appear in the top level category. The bin content of the new histogram will be equal to `constant` * bin value of histogram id1 / bin value of histogram id2. If the bin value of histogram id2 is 0, the value 1 is used in the computation. Returns `new_id` which is the id of a new histogram containing the result of the operation, or -1 if an error occurred.

```
new_id = hs_1d_hist_derivative(uid, title, category, id);
    int uid; (input)
    char *title, *category; (input)
    int id; (input)
    int new_id; (returned)
```

Create a new histogram (one-dimensional) whose data is the derivative of the histogram specified by `id`. This derivative is computed bin by bin:

Dh/db = (H(j) - H(j-1))/bin_size.

If the input histogram, specified by `id`, has error bars, the errors are computed assuming that there is no statistical correlation between bins and the error values are gaussian. `uid` is the User Identification for the newly created histogram. `title` defines the title for the histogram. `category` is a string defining a hierarchical "location" for the histogram. Returns `new_id` which is the id of a new histogram containing this derivative.

Summing Groups of Histograms and Ntuples

The two routines, hs_sum_category and hs_sum_file, are designed to allow the user to sum together the data items from two different runs of the same program. For example, if on one day, a program creates the following items:

category	uid	item type
mydata	101	1-d histogram
mydata	102	2-d histogram
mydata/ntuples	202	Ntuple

and writes these items to a file, the next day, he can use hs_sum_file to sum today's one-dimensional histogram with its corresponding 1-d histogram on the file, sum today's two-dimensional histogram with its corresponding 2-d histogram on the file, and merge together the two Ntuples. The hs_sum_file call for the above example would look as follows:

```
hs_sum_file(filename, 'mydata', 'sum')
```

and would create the following new items:

category	uid	item type
sum	101	1-d histogram
sum	102	2-d histogram
sum/ntuples	202	Ntuple

These routines search the items in the category specified and match their uid's with the data items found in the file (for hs_sum_file) or second category (for hs_sum_category) specified. An error message will be printed if the data types do not match when the data items' uid's do match. An error message is also printed if two histograms or Ntuples with matching uid's do not have the same properties, for instance a different number of bins or variables. The resulting summed data items are created in memory and are given the new category specified (result_category). Items other than histograms and n-tuples are ignored.

hs_sum_category(`category1, category2, result_category`**);**

char *`category1`, *`category2`; *(input)*

char *`result_category`; *(input)*

Create a collection of new histograms (one-dimensional or two-dimensional), and/or Ntuples, based on two existing categories of data items. All such items within these categories are considered, and it is assumed that the categories have parallel category trees below a top prefix (e.g. from a `hs_read_file` call) and that the uids/subcategories and binning properties are identical. Items with uid == 0 are skipped. For instance, `category1` can be specified as 'run23', `category2` as 'run25', `result_category` as 'run23_25'. Then, if a histogram exists both in 'run23/tracking' and in 'run25/tracking' with identical uid's and static properties, a new histogram with the same uid will be created which is the sum, bin by bin, and with category 'run23_25/tracking'. Since this routine is intended to accumulate statistics, no arbitrary constant is provided. `category1 and category2` are the top level categories of the categories to be summed. These cannot be NULL, nor empty (Uncategorized), nor contain "...". `result_category` is the top category for the newly created category.

hs_sum_file(`file, category, result_category`**);**

char *`file`; *(input)*

char *`category`; *(input)*

char *`result_category`; *(returned)*

Read all of the items from `file`, sum histograms and/or merge Ntuples with those existing under the top category specified by `category` and store all newly created items in a category specified by `result_category`. This can be the same as `category`. Thus, to sum a bunch a files, one can read the first one by calling `hs_read_file(file1, 'allsum')` and loop over the others, calling repetitively `hs_sum_file(file`*n*`, 'allsum', 'allsum')`. The items read from the file will not be kept in memory. A known deficiency of this routine is that it assumes that there is no top category already named "tmp_sum_file" or "tmp_sum_result".

Functions for accessing Ntuple data

`num_entries = hs_num_entries(id);`

int id; *(input)*

int num_entries; (returned)

Returns the number of fill operations that have been performed on the histogram referred to by `id`, or -1 if the id does not refer to a histogram or Ntuple item.

`num_var = hs_num_variables(id);`

int id; *(input)*

int num_var; *(returned)*

Returns the number of variables (`num_var`) in the Ntuple specified by `id`. Returns -1 if the id is not valid or not an Ntuple.

`length = hs_variable_name(id, column, name);`

int id; *(input)*

int column; *(input)*

char *name; *(returned)*

int length; *(returned)*

Returns the `name` and `length` of a given variable for the Ntuple specified by `id`. `column` is the variable index (e.g., a column number) and `name` is the mnemonic name of this variable in the Ntuple.

`index = hs_variable_index(id, name);`

int id; *(input)*

char ***name; *(input)*

int index; (returned)

Returns the `index` (i.e. the column number in the Ntuple) corresponding to a given variable specified by its mnemonic `name`, for a given Ntuple specified by `id`. An exact match is required between `name` and one of the variable's names defined in the Ntuple. If no match is found, returns -1. This routine does not check for multiple matches, it successfully completes at the first occurrence of name.

```
value = hs_ntuple_value(id, row, column);
   int id; (input)
   int row; (input)
   int column; (input)
   float value; (returned)
```

Returns the value (`value`) in an Ntuple, at a specified `row` and `column` index. `value` is undefined (-1.0 is returned) if `id` does not exist or refers to an item other than an Ntuple.

```
hs_ntuple_contents(id, data);
   int id; (input)
   float *data; (returned)
```

Returns all of the data in an Ntuple, specified by `id`, in the form of a two dimensional array named `data` which contains floating point values. The array must be dimensioned to the number of rows (entries) by the number of columns (elements or variables) in the Ntuple.

```
hs_row_contents(id, row, data);
   int id; (input)
   int row; (input)
   float *data; (returned)
```

Returns the contents of a row (or entry) of Ntuple data specified by `id` and places it in the array `data`. Data must be dimensioned to the number of variables (elements) in the Ntuple. A `row` is the collection of values in a particular Ntuple entry, one value for each variable (A variable could also be called a column or an element).

```
hs_column_contents(id, column, data);
   int id; (input)
   int column; (input)
   float *data; (returned)
```

Returns the contents of a column of Ntuple data specified by `id` and places it in the array `data`. A `column`, which can also be called an element or a variable, is the set of values taken by a single Ntuple variable for each entry in the Ntuple.

```
new_id = hs_merge_entries(uid, title, category, id1, id2);
    int uid; (input)
    char *title, *category; (input)
    int id1, id2; (input)
    int new_id; (returned)
```

Creates a new Ntuple consisting of two existing Ntuples, specified by `id1` and `id2`. These two Ntuples must have an identical number of variables. `uid` is the User Identification number for the newly created Ntuple. `title` defines the title for the new Ntuple. `category` is a string defining a hierarchical "location" for the Ntuple. Returns `id`, the Histo-Scope id of the newly created Ntuple, or -1 if the Ntuples referred by `id1`, or `id2` don't exist, are not Ntuples or have a different number of variables. The variable names are taken from id1, and are assumed to be identical to those of id2.

Hints for Programmers

- If the linker cannot find the routines: hs_hbook_setup and hs_reset_hbook_setup when you are linking your program, you may be linking with the wrong library. Try linking with libFHistoHB (or libCHistoHB if you are coding in C).

- If you call HS_HISTOSCOPE and get an error that the histo executable was not found, type *setup histo* to the shell (UNIX) before executing your program.

- If running on IBM AIX systems and your FORTRAN program does not compile, make sure that the program uses the FORTRAN INCLUDE statement instead of the #include statement for including the Histo-Scope include file. Also check that you used the compiler option for specifying the path to the include file for your system. $HISTO_DIR/user_sources contains a makefile for the example programs and can be editted for your own use.

- If you have re-compiled and linked a pre-V3.0 program with a V3 Histo-Scope library and run the program without adding the **`uid`** parameter to the item creation routine calls, you may find that the program gets a segmention violation, or bus error at or near the item creation routine call. Add the **`uid`** parameter in the correct place to fix this error.

- If you have booked new HBOOK histograms or Ntuples, or deleted, renamed, rebinned, or reset existing HBOOK histograms or Ntuples, and these changes are not reflected in the Histo-Scope display, you need to call HS_RESET_HBOOK_SETUP in your program. Refer to the description of the routine.

- If your program prints an error indicating that it cannot perform a malloc or that virtual memory has been exhausted, make sure your Histo-Scope defined Ntuples don't grow too large for the virtual memory of your machine. Also check the return status of HS_FILL_NTUPLE.

- If Histo-Scope rejects connections, or updating is too infrequent, call HS_UPDATE more often. There is no cpu penalty for calling HS_UPDATE more frequently than is necessary.

- To our knowledge, HBOOK does not let a program access multiple top directories without opening/closing them sequentially. Therefore, we recommend using Histo-Scope with only one top directory at a time.

- If your program does an HRPUT, put the call to HRPUT <u>after</u> the call to HS_COMPLETE. This is necessary because under some circumstances HRPUT will make the HBOOK data inaccessible, and HS_COMPLETE may access the HBOOK data for Histo-Scope one last time before completing.

- Histo-Scope **can** be used with any process embedded in a CPS application.

Release Notes

Requirements - Histo-Scope requires an X-based workstation (DEC-ULTRIX, DEC-OSF/1, SGI UNIX, Sun (Solaris) UNIX, or IBM/Risc UNIX) **or** an X Terminal or X-based workstation that can connect to one of these hosts.

HBOOK Circular Ntuple Support - Histo-Scope supports, in a limited way, the display of data in the new HBOOK Circular Ntuple data structure. Because HBOOK does not, at the current time, provide a way for Histo-Scope to inquire the current "insertion point" of the circular buffer, and because the user can at any time change the fills from circular to linear and back again, Histo-Scope uses the following algorithm for determining that an HBOOK Ntuple is circular and entries should be re-sent from the data generating process to the scope starting from the first entry (row):

- The number of entries in the Ntuple is nonzero
- The total number of entries is identical to the number recorded at the previous call to hs_update
- Any element (column) of the first row in the currently displayed Ntuple plot is different from the corresponding element of the first row now in the HBOOK Ntuple

Note that this strategy will fail if the HBOOK Ntuple is filled circularly with data that is exactly the same in the first row as what was there before. It will also fail if the "insertion point" changed without changing the data in the first Ntuple row. In these cases, the Ntuple viewed by the Histo-Scope user will still be the old data, rather than the new.

UNIX restriction on putting Histo-Scope process in background - If you are running Histo-Scope on a UNIX system, do not put the Histo-Scope process in the background if you plan to connect to a process. This is because the system may put the Histo-Scope process into a wait state that it will not get out of until the process is put into the foreground. After the connection is successful, the Histo-Scope process may be put into the background, if necessary, until you need to connect to another process.

Some X-Servers do not destroy child dialog windows - Some, possibly older, X-Terminals and servers do not properly destroy dialogs when the parent window is destroyed. On these servers doing anything with these orphan windows can cause Histo-Scope to terminate with an error such as Segmentation Violation. To avoid this on X servers with this problem, be sure to close slider, statistics, and axis settings windows before closing their corresponding plot window.

Printer Options window delay - Occasionally while printing a plot using flpr you may experience a delay for up to a minute. This is usually because the host you have specified is wrong or the queue does not exist. After the delay, Histo-Scope will display the error that occurred**.**

Floating Point Errors - It is possible for Histo-Scope to produce floating point errors such as Floating Point Overflow when floating point values approach the limits on the machine for single-precision floating point values. Problems can occur when floating point values are in the range of 10E+38 or 10E-38**.**

Setting up Histo-Scope on UNIX without Fermilab's ups/upd - For off-site UNIX users of Histo-Scope without ups/upd, the add the following commands to your .cshrc file:

setenv HISTO_DIR *path-to-Histo-Scope*
set path = ($PATH, $HISTO_DIR/bin)

These variables must be defined before invoking Histo-Scope. They must also be added to your .cshrc file on any node you intend to call the Histo-Scope library routine hs_histoscope. If you will be building programs with Histo-Scope library routines, also define:

setenv HISTO_INC $HISTO_DIR/include

Index