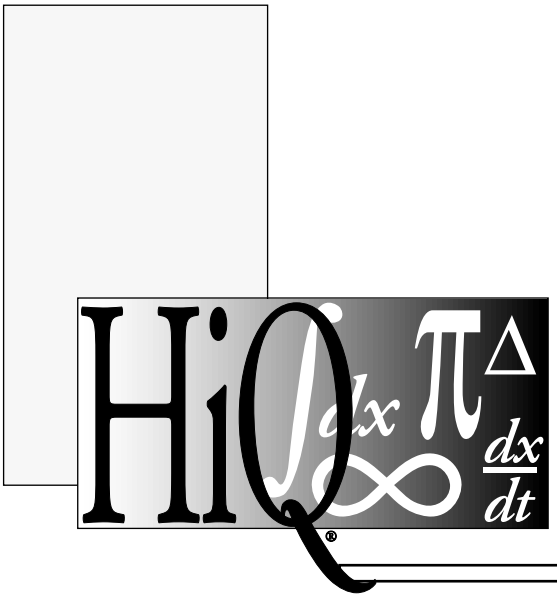


HiQ® 2.2 Release Notes

for Macintosh and Power Macintosh

Part Number 320949A-01
June 1995

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Italy 02 48301892, Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 202 2544,
Netherlands 03480 33466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085,
Sweden 08 730 49 70, Switzerland 056 20 51 51, Taiwan 02 377 1200, U.K. 01635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error-free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Warning Regarding Medical and Clinical Use of National Instruments Products

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

CONTENTS

About the HiQ 2.2 Release Notes

Manual Organization ix
Installation ix
Where To Go from Here ix
Notation Conventions ix

CHAPTER 1
Installation

System Requirements 1-1
Installation Procedure 1-1
Installing on a Power Macintosh 1-2
Installing on a 680x0 Macintosh 1-3
HiQ Files 1-4

CHAPTER 2
New Features and Improvements

New Features 2-1
HiQ for Macintoshes without an FPU 2-1
Defining Response to Mouse Actions 2-1
Controls 2-1
Locking Symbols 2-2
AppleScript and AppleEvents 2-3
Program-to-Program Communication 2-4
High-Resolution Printing 2-4
Graph Editor 2-5
Apple's Fast Math Library 2-6
Other New Features 2-7
Improvements 2-7
New Untitled Worksheet 2-7
Saving from Editors 2-8
On-Page Text Editing 2-8
Placement of Symbols 2-8
Graph Sizing 2-8
Importing and Exporting Symbols 2-8

CHAPTER 3

New Built-In Functions

Graphical Functions 3-1
change2DDataPlot 3-2
change3DDataPlot 3-3
Utility Functions 3-4
compile 3-4
getUIInfo 3-4
progress 3-5
setProgressPercent 3-7
setProgressText1 3-8
setProgressText2 3-8
setProgressTitle 3-9

CHAPTER 4

PPC Functions

PPC: An Example 4-1
PPCOpen 4-5
PPCClose 4-6
PPCRead 4-6
PPCWrite 4-7

CHAPTER 5

Import/Export Functions

Writing Your Own Import and Export Functions 5-1
importSymbol 5-4
importNumeric 5-5
exportSymbol 5-7
exportNumeric 5-8

Chapter 6

LabSuite Utilities

LabSuite: An Example 6-1
Linking HiQ and LabVIEW 6-2
File I/O 6-2
PPC 6-2
(LabVIEW) HiQ PPC Connect 6-3
(LabVIEW) HiQ PPC Disconnect 6-4
(LabVIEW) Find an Open HiQ PPC Port 6-5
(LabVIEW) HiQ PPC Read 6-5
(LabVIEW) HiQ PPC Read Real+ 6-6
(LabVIEW) HiQ PPC Write 6-7
(LabVIEW) HiQ PPC Write Integer+ 6-8
(LabVIEW) HiQ PPC Write Real+ 6-9
(LabVIEW) HiQ PPC Write Complex+ 6-10
(HiQ) PPC_HiQ_LV_ReadData 6-10
(HiQ) PPC_HiQ_LV_WriteData 6-11
AppleEvents 6-12
Open HiQ 6-12
Find an Open HiQ 6-12
Quit HiQ 6-13
Open Worksheet 6-13
Print Worksheet 6-14
Enter HiQ Script 6-14
Execute Script 6-15

Appendix

Customer Communication

Electronic Services A-1
Bulletin Board Support A-1
Internet Support A-1
FaxBack Support A-2
Fax and Telephone Support A-2

About The HiQ 2.2 Release Notes

Manual Organization

This version of HiQ ships with a documentation set consisting of three manuals:

- ***HiQ 2.2 Release Notes***: shows how to install HiQ on your computer and explains new features in HiQ 2.2. Information in this document supplements and, in some cases, supersedes information in the *HiQ User Manual* or *HiQ Reference Manual.* If you purchased HiQ 2.2 as an upgrade, you will only receive the *HiQ 2.2 Release Notes* with your software. Please refer to your previous versions of the *HiQ User Manual* and *HiQ Reference Manual* for information not included in these Release Notes.
- ***HiQ User Manual***: introduces HiQ concepts.
- ***HiQ Reference Manual***: a full reference to all of HiQ's built-in functions, including a discussion of the HiQ algorithms.

Installation

The installation information in Chapter 1 of these *Release Notes* supersedes the installation information in the *HiQ User Manual*. Use only Chapter 1 of the *Release Notes* to install HiQ 2.2.

Where To Go from Here

New HiQ Users should refer to the *HiQ User Manual* after installation for an introduction to the HiQ environment. When you are comfortable with the HiQ environment, you should return to this document to learn about new features that are not included in the *HiQ User Manual.*

Experienced HiQ Users should read this document to become familiar with HiQ's new features.

Notation Conventions

Examples of HiQ-Script code look like the following excerpt.

```
function factorial(y)
   x = y;
   z = x;
   while(x > 1)
      x = x-1;
      z = z*x;
   end while;
```

```
        return z;
    end function;
```

Keywords are in bold. When HiQ-Script keywords such as *while* are referred to within text, they are set in the `Courier` typeface.

CHAPTER 1
INSTALLATION

The information in this chapter supersedes the information in the *HiQ User Manual*. Use only this chapter to install HiQ 2.2.

SYSTEM REQUIREMENTS

HiQ 2.2 runs as a native application on both Power Macintosh and 680*x*0 Macintosh computers. HiQ runs on any Power Macintosh computer. For 680*x*0 Macintosh computers, you must have a 68020 or later processor.

You also need at least 5 MB of RAM on your computer (8 MB recommended). HiQ also requires Macintosh System 7 or later. The relevant system requirements for HiQ are summarized as follows:

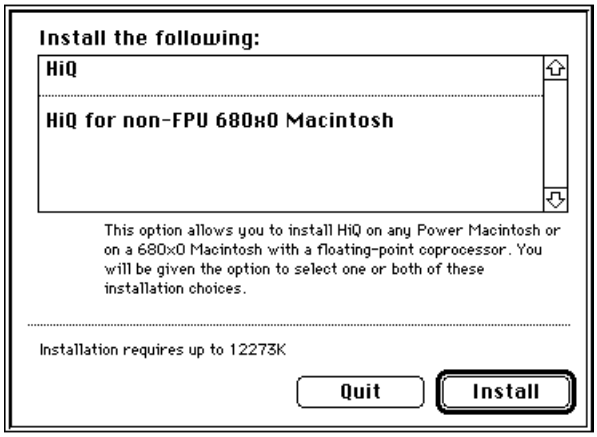
- Power Macintosh or a Macintosh with a 68020 or later processor
- At least 5 MB of RAM
- System 7 or later

INSTALLATION PROCEDURE

All HiQ files are compressed in a self-extracting archive. To install HiQ 2.2 on your hard disk, perform the following steps:

1. **Insert the HiQ 2.2 Program Disk #1 into your disk drive.**
2. **Double-click on the `HiQ 2.2 Program Installer` icon and follow the instructions on the screen.**

The following dialog box appears:



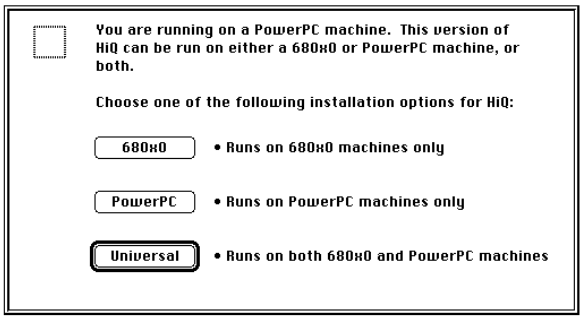
You have two installation options to choose from.

- **HiQ**—This option allows you to install HiQ on any Power Macintosh or on a 680*x*0 Macintosh with a floating-point coprocessor. You will be given the option to select one or both of these installation choices.
- **HiQ for non-FPU 680x0 Macintosh**—This option allows you to install HiQ on a 680*x*0 Macintosh that doesn't have a floating-point coprocessor (FPU). This option requires a 68020 or later processor and System 7 or later. Install this version only if your Macintosh doesn't have an FPU.

68040 processors have a built-in coprocessor and can run HiQ; however, the 68LC040 processor has the built-in coprocessor disabled. If you have a 68LC040 Macintosh, it does not have a floating-point coprocessor.

INSTALLING ON A POWER MACINTOSH

If you are installing HiQ on a Power Macintosh, the following dialog box appears during the installation procedure:

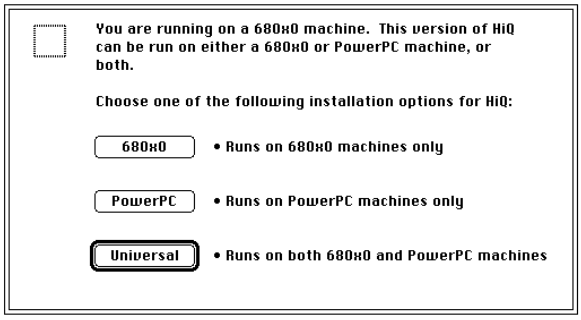


You have three options to choose from:

- **680*x*0—**This option installs HiQ as a native 680*x*0 Macintosh application. Choose this option only if you have purchased a multiple-user license for HiQ and your Power Macintosh is a server machine for other 680*x*0 machines. Do not choose this option if you plan to run HiQ from a Power Macintosh.
- **PowerPC—**This option installs HiQ as a native Power Macintosh application. This is the preferred option for the Power Macintosh.
- **Universal—**This option installs HiQ as a Fat Binary application that can run as a native application on either a 680*x*0 or Power Macintosh. The Fat Binary includes code that is appropriate for both 680*x*0 and the Power Macintosh. This type of installation requires approximately twice as much disk space as the processor specific options, but does not require any more memory to run. Choose this option only if you have purchased a multiple-user license for HiQ and your Power Macintosh is a server machine for other 680*x*0 and Power Macintosh computers.

INSTALLING ON A 680*X*0 MACINTOSH

If you are installing HiQ on a 680*x*0 Macintosh, the following dialog box appears during the installation procedure:



You have three options to choose from:

- **680*x*0—**This option installs HiQ as a native 680*x*0 Macintosh application. This is the preferred option for 680*x*0 Macintosh computers.
- **PowerPC—**This option installs HiQ as a native Power Macintosh application. Choose this option only if you have purchased a multiple-user license for HiQ and your 680*x*0 Macintosh is a server machine for other Power Macintosh computers. Do not choose this option if you plan to run HiQ from a 680*x*0 Macintosh machine.
- **Universal—**This option installs HiQ as a Fat Binary application that can run as a native application on either a 680*x*0 or a Power Macintosh. The Fat Binary includes code that is appropriate for both 680*x*0 and Power Macintosh computers. This type of installation requires approximately twice as much disk space as the processor specific options, but does not require any more memory to run. Choose this option only if you have purchased a multiple-user license for HiQ and your 680*x*0 Macintosh is a server for other 680*x*0 and Power Macintosh computers.

HiQ Files

The HiQ 2.2 Program Installer creates a `HiQ 2.2` folder on your hard disk. This folder contains the following items:

- The `HiQ 2.2` application

- `README`—A text file containing the latest information regarding this version of HiQ

- `HiQ Sample Worksheets`—A folder containing a set of sample HiQ Worksheets

- `HiQ Sample Data Files`—A folder containing a set of sample HiQ data files for use with some of the sample HiQ Worksheets

- `Worksheet Controls`—A folder containing Worksheets that implement controls and contain pictures for use in controls

- `LabSuite Tools & Examples`—A folder containing Worksheets and LabVIEW VIs that link HiQ and LabVIEW together

- `HiQ-Scripts`—A folder containing utility HiQ-Scripts

- `HiQ•Constants.68kMac`—A text file containing useful HiQ-Script constants for 680x0 Macintosh machines

- `HiQ•Constants.PowerMac`—A text file containing useful HiQ-Script constants for the Power Macintosh

- `HiQ•Conversions`—A text data file containing conversions used by the convertUnits built-in function (See the `convertUnits` function in the *Utility Functions* chapter of the *HiQ Reference Manual*)

- `HiQ•Errors`—The HiQ error file used by HiQ's error reporting system

- `HiQ•Help`—The HiQ help file used by HiQ's on-line help system

CHAPTER 2
NEW FEATURES AND IMPROVEMENTS

This chapter describes the many new features and improvements in HiQ 2.2.

NEW FEATURES

HIQ FOR MACINTOSHES WITHOUT AN FPU

Included with HiQ 2.2 is a version of HiQ that runs on a Macintosh without a Floating Point Unit (FPU). (An FPU is sometimes called a numeric coprocessor.) Several Powerbook, Performa, LC, and other Macintosh models do not come with an FPU and cannot run earlier versions of HiQ. With HiQ 2.2, many more Macintoshes can run HiQ. If you install on a 680*x*0 Macintosh, you have the option to install the non-FPU version of HiQ 2.2. The minimum system requirements for the non-FPU version of HiQ are 68020 or greater, color QuickDraw, and System 7 or greater.

You should only install the non-FPU version of HiQ 2.2 if you will be running HiQ on a Macintosh without an FPU. The non-FPU version of HiQ runs on a Macintosh that does have an FPU, but it runs more slowly than the FPU version. The non-FPU version also runs on a Power Macintosh, but this is discouraged. A native Power Macintosh version of HiQ is included on the installation disks.

If you are not sure if your Macintosh has an FPU, check the documentation that came with your computer. Alternatively, install and run the FPU version of HiQ 2.2. A message informs you if you *do not* have an FPU. If you see this message, remove the FPU version of HiQ 2.2 and install the non-FPU version.

DEFINING RESPONSE TO MOUSE ACTIONS

You can now define the response to single- or double-clicking on an object on the Worksheet. Two pull-down menus at the bottom of the Assign Properties… dialog box, accessed from the File menu, allow you to select these actions. The default actions are to Select on single-click and Edit on double-click. By choosing Run Script..., you can associate a script to either single- or double-click actions. When that action is detected, the script is executed. You can also choose Do Nothing as the action. When a Symbol has Do Nothing selected for both single- and double-click actions, the Symbol is visible but nothing can be done to it on the Worksheet. These responses are only activated by selecting Enable Click Actions from the Worksheet menu. They can be deactivated by selecting Disable Click Actions.

CONTROLS

With the click-action feature, you can create basic controls on the Worksheet. For an example of these controls, refer to the Example Controls Worksheet that came with your copy of HiQ 2.2. These controls are

implemented as picture objects that update when a script is run. For example, if you want to make a vertical toggle switch, you need three pictures—the picture that you are going to display on the Worksheet, a picture of a vertical toggle switch in the down position, and a picture of a vertical switch in the up position.

You would write a script resembling the following.

```
if toggleIsTrue then
   myControl = toggleUp;
   toggleIsTrue = <false>;
else
   myControl = toggleDown;
   toggleIsTrue = <true>;
end if;
```

The variable `toggleIsTrue` is an integer scalar and must exist before the script is run. Its value is 0 or 1. The variable `toggleUp` is a picture of a vertical toggle switch in the up position. The variable `toggleDown` is a picture of a vertical toggle switch in the down position.

Once you have created all of these Symbols, you compile the script. Place the picture myControl on the Worksheet. Choose Assign Properties... for myControl and choose the above script as the script to run on a single-click action. Select Enable Click Actions from the Worksheet menu and then click on the picture. It should toggle between the up and down position. You can then use the variable toggleIsTrue in another script to determine the state of the switch.

Refer to the Worksheet Controls folder that comes with HiQ for more examples of controls.

LOCKING SYMBOLS

There are now Lock and Unlock menu items in the Worksheet menu. The Lock option locks the position and size of the selected Symbol. You can still select and edit Symbols locked in this fashion. This option is particularly useful when you want to fix the position of a Symbol on the Worksheet but still allow editing. Refer to the earlier section, “Defining Response to Mouse Actions,” for more information on controlling access to Symbols.

APPLESCRIPT AND APPLEEVENTS

You can now use five AppleEvents in HiQ: the four required events—Run, Open, Print, and Quit—and a DoScript event for executing a script in a specified HiQ Worksheet. Any application that can send AppleEvents, such as AppleScript or LabVIEW, can now control HiQ automatically.

RUN

The Run event is set automatically when the application is launched.

OPEN

Use the Open event to open an existing Worksheet. The following example shows how to open a Worksheet named Example from AppleScript.

```
tell application "HiQ"
   open file "HardDisk:HiQ Worksheets:Example"
end tell
```

PRINT

Use the Print event to print an existing Worksheet. The following example shows how to print the Worksheet opened in the previous example:

```
tell application "HiQ"
   print file "HardDisk:HiQ Worksheets:Example"
end tell
```

QUIT

The Quit event tells HiQ to quit. The following script launches HiQ if it is not already running and then close the program.

```
tell application "HiQ"
   quit
end tell
```

DOSCRIPT

With the DoScript event, HiQ can execute a specified HiQScript in a Worksheet. The command has two forms. The first executes the specified script in the first Worksheet opened. The second executes the script in the specified Worksheet:

```
DoScript "scriptName"
DoScript "worksheetName:scriptName"
```

The following example launches HiQ, runs a script that creates a 3 x 3 matrix, places that matrix in the AppleScript variable, theResult, and then quits HiQ.

AppleScript:

```
tell application "HiQ"
   open file "HardDisk:HiQ Worksheets:Example"
   set theResult to DoScript "Example:Script"
   quit
end tell
```

HiQ-Script:

```
A={1,2,3;4,5,6;7,8,9};
return (A);
```

Matrices and vectors passed back to AppleScript using the `return` statement are converted to HiQ-Script tab-delimited text for easy transfer to other applications. The size of these matrices and vectors is limited to the value specified by Execution preferences found in File\Preferences for...\project. By default, the maximum number of rows and columns is 30.

PROGRAM-TO-PROGRAM COMMUNICATION

HiQ 2.2 supports Program-to-Program Communication (PPC) with a set of four PPC built-in functions. With these functions, you can exchange data between HiQ and any other application that supports PPC. A comprehensive library of LabVIEW VIs and examples comes with HiQ to illustrate the link between HiQ and LabVIEW using PPC. See Chapter 4, "PPC Functions," for more information on these functions.

HIGH-RESOLUTION PRINTING

NEW CAPABILITIES

HiQ 2.2 supports high-resolution printing. This means that the appearance of printed Worksheets is much improved. Lines in Graph Symbols now appear much sharper and there is a new hairline (as fine as your printer can produce) plot line width option in the Graph Editor. Notice that when you use the hairline option, the plot on screen appears the same as if you had chosen a line width of one pixel. When the graph is printed, however, the line will be as fine as possible for your printer.

The hairline option is also available for Worksheet annotations. The zero line-width setting in the Tool palette's Line Width tool has been replaced by the hairline setting.

FONT APPEARANCE

As a result of HiQ's new high-resolution capabilities, text on your existing HiQ Worksheets may now look different on screen. In particular, spacing between letters, words, and lines of text is now different. This may cause text on existing Worksheets to wrap in a different location. If this adversely affects the appearance of your existing Worksheets, try adjusting the width of text Symbols or change the font size.

Some fonts may be substituted by your printer when printed, resulting in a printout of lower quality that does not match what is on screen. To avoid this, choose fonts that are available on your printer. Fonts that are commonly substituted when printed include Chicago, Geneva, Monaco, and New York.

DETERMINATION OF PRINTER RESOLUTION

When HiQ prepares to print, it queries the printer driver for information on the resolutions supported by the printer. Some printers allow only a discrete set of possible resolutions, while others offer the ability to set the resolution to any value in a certain range. PostScript printers fall into this second category. HiQ tries to choose a printer resolution that gives the best quality output that also accurately represents the screen resolution. When possible, HiQ chooses a resolution that is an integer multiple of 72 dpi (dots per inch), the assumed screen resolution. HiQ also chooses a resolution with the same dpi in each direction.

Some printers cannot be set to an integer multiple of 72 dpi. While every attempt is made to remain faithful to the positions and sizes of objects, output from these printers may differ slightly. Text sizes and the locations of word wrapping may be slightly different than on screen. Printers that fall into this category are not recommended.

The following table shows how HiQ sets the resolution for the printer drivers included with System 7.5:

Printer	**Natural Resolution**	**HiQ Resolution**
AppleTalk ImageWriter	144	144
ImageWriter	144	144
LaserWriter	300	360
LaserWriter 300	300	300
LaserWriter 8	300	360
LW Select 310	300	360
Personal LaserWriter SC	300	300
StyleWriter II	360	360

The LaserWriter 300 and Personal Laser Writer SC are not recommended for high-resolution output because they cannot be set to an integer multiple of 72. ImageWriters are not recommended because they have a very low resolution.

If your printer is not in the preceding table, consult the documentation that came with your printer for more specifics. PostScript printers are always a good choice because they allow HiQ to pick an optimal printer resolution.

GRAPH EDITOR

Several new features in the Graph Editor allow more control over the appearance of your graphs.

SELECTION AND MOVEMENT

You can now select and move the following items, using the Arrow tool.

- graph frame (The Hand tool gives the ability to move the graph frame but not to select it.)
- graph title
- plot title
- plot legend

FONT SUPPORT

You can set font, size, and style of any graph text using two new menus, which are identical to the font menus in the Text Editor. To activate the menus, select either the plot, graph frame, or graph title with the Arrow tool. When the graph frame is selected, you can modify the font information for the axis labels and axis titles. When a plot is selected, you can modify the font information for the plot title and legend.

GRIDLINES

Gridlines are now displayed as solid and dashed lines for major and minor gridlines respectively, and are printed as hairlines. Due to the way dashed lines are drawn, you may notice that graphs with gridlines now print more slowly.

FRAME

The Graph Options... dialog box in the Graph menu now has a check box that allows you to place a frame around a 2-D graph.

COPYING GRAPHS AS PICTS

When you copy a graph in HiQ, a high-resolution PICT is created. These PICTs can be used in all applications that support PICTs. In applications that also support high-resolution printing, these PICTs should be printed in high resolution. The resolution of the PICT is set to the resolution of the connected printer or the next higher multiple of 72 dpi. If no printer is connected, a default resolution of 360 dpi is used.

If your system is not connected to the printer you will ultimately print to, select that printer from the Chooser before you do the copy. As long as you do not actually try to print while this printer is selected, the copy will be successful. Remember to reset the Chooser to your normal printer when you are done.

Some applications do not print the PICTs that HiQ creates at high resolution. If a high-resolution PICT is pasted into HiQ, it is printed at high resolution.

APPLE'S FAST MATH LIBRARY

If you are installing HiQ 2.2 on a Power Macintosh with System 7.1.2 or System 7.5.0, the Installer includes the MathLib and MathLib v2 Release Note files in the Extensions folder. Systems 7.5.1 and later either supply these files or do not require them. MathLib is a shared library that contains performance enhanced versions of many hardware math routines found in the ROM of Power Macintoshes. The library substantially improves the computational speed of HiQ built-in functions that use hyperbolic, trigonometric, power, exponential, and logarithmic functions. For example, some of the built-in functions affected are: xsinh, xsin, xexp, xln, and FFT. The ROM routines and the MathLib routines have identical numerical results. However, in a few exceptional cases, the sign of zero and generated NaN code may be different.

MathLib is a shared library. When dropped into any application folder, it enhances the performance of applications that use the ROM math functions found in that folder. If MathLib is placed into the System folder, it benefits all applications that use the ROM math functions.

Although MathLib was not written by National Instruments, the company distributes it with HiQ because of its obvious benefits for some HiQ users. For more information, see the MathLib v2 Release Note file.

OTHER NEW FEATURES

KEYBOARD KEY TO RUN A HIQ-SCRIPT

In the HiQ-Script Editor, there is now a keyboard equivalent to clicking on the Run button to run a script. Pressing <shift-return> in the Script Editor now compiles and runs the current HiQ-Script.

INTEGRATION PROBLEM SOLVER TO GENERATE A GRAPH

A new option has been added to the Numerical Integration Problem Solver. The Graph option at the bottom of that Problem Solver generates a graph of the function you are integrating. When the Save script option is also selected, HiQ-Script is generated containing the statements to create the graph.

SUPPORT FOR STATIONERY PAD FILES

HiQ now supports stationery pad files. When you open a HiQ stationery pad, you are asked to save it as a new document. You can create a stationery pad or change it back to a normal HiQ document by selecting File\Assign Properties… from the Finder and deselecting the stationery pad check box.

PROGRESS BOX UTILITY BUILT-IN FUNCTIONS

HiQ 2.2 has a new set of built-in functions for creating modifying, updating, and removing a progress box using HiQ-Script. Refer to Chapter 3, “Built-In Functions,” for more information.

IMPROVEMENTS

NEW UNTITLED WORKSHEET

In previous versions of HiQ, you were asked to name and save a Worksheet before a new Worksheet opened. In HiQ 2.2, a new Worksheet named untitled automatically opens when HiQ is launched in the foreground. An untitled Worksheet opens also when you select New Worksheet from the File menu.

New Worksheets are named untitled, untitled 2, untitled 3, and so on. For situations when HiQ is launched in the background (when AppleScript is running, for example), no new Worksheet opens.

It is recommended that you save the Worksheet as soon as possible. As always, frequent saves are good insurance against accidental data loss due to power failure or some other unforeseen circumstance. When you initiate any of the following actions, the standard Save dialog box appears, asking you to name the location where the Worksheet is to be saved.

- when choosing Save Worksheet, Save Worksheet As…, or Save Copy of Worksheet… anywhere in the application.
- when running a script when one of the Save Worksheet preferences is on.
- when running a script that contains a SymbolSave function call.

SAVING FROM EDITORS

In HiQ 2.2, when you save from an editor, the entire Worksheet is saved. When you close an editor, you are no longer asked to save the changes to the edited Symbol.

ON-PAGE TEXT EDITING

On-page text editing is more natural in HiQ 2.2. To select text on the Worksheet, you need only click on the text with the Arrow tool. To select a text object for moving or other Worksheet-related activities, click on the highlighted border with the Arrow tool. All objects now display this border when selected.

PLACEMENT OF SYMBOLS

When you place a new Symbol on the Worksheet, a single click places it in Expanded view instead of Icon view. You can still select the Icon view from the Worksheet menu. You still place Problem Solvers and HiQ-Script in Icon view.

Other modifications are that you can now place Symbols on top of existing Symbols, and you now select Symbols only with the Arrow tool.

GRAPH SIZING

The size of a graph is now determined by the size of its view on the Worksheet. The Graph Editor has been modified to have scrollbars so that it is more convenient to edit large graphs. The Fit to Contents command now more accurately fits the graph to the enclosing window.

IMPORTING AND EXPORTING SYMBOLS

FILE\IMPORT SYMBOL... AND FILE\EXPORT SYMBOL...

Importing and exporting data Symbols have been improved in HiQ 2.2 in the following ways.

- The File\Import Symbol… and File\Export Symbol… dialog boxes have a streamlined interface requiring fewer steps.
- New options in the File\Preferences for… dialog box allow you to set the default import and export method for numeric Symbols.
- The import and export operations are faster and more robust, and memory requirements for import have been greatly reduced in some cases.
- During an import or export operation, a progress box now appears, providing visual feedback about the progress of the operation and allowing you to stop it.
- The Import Symbol operation now allows you to overwrite an existing HiQ Symbol.
- The File\Import Symbol… and File\Export Symbol… dialog boxes now behave identically to the importNumeric and exportNumeric functions when importing numeric data. (In most cases, you should use these functions to import or export data in a HiQ-Script.)

IMPORT/EXPORT BUILT-IN FUNCTIONS

The import and export functions have been modified to be more flexible. See Chapter 5, “Import/Export Functions,” for details.

CHAPTER 3
NEW BUILT-IN FUNCTIONS

This chapter discusses several new built-in graphical and utility functions for modifying plots, compiling HiQ-Script, returning mouse location, and displaying a progress dialog box.

GRAPHICAL FUNCTIONS

Two new graphical functions, change2DDataPlot and change3DDataPlot, have been added to facilitate the modification of plot data while maintaining all other plot attributes. These functions allow you to reuse plots with different sets of data. In effect, a plot can now be treated as a template to which data is added. You can interactively or programmatically set up a plot to look the way you like, then programmatically extract the plot from the graph and replace the data. You can then place the plot back in the same graph or place it in a new graph.

This methodology can also be applied to graphs. You can set up a graph interactively and then use it as a template for new graphs or as a container for new plots. This can significantly decrease the amount of programming required to get your graphs to look the way you want.

The following script demonstrates how to reuse graphs and plots:

```
// Create the graph or reuse an existing one.
if symbolGetType(myGraph) = <untyped> then
 myGraph = new2DGraph("Example");
end if;

// Progressively increase factor every time the script is executed.
if symbolGetType(factor) = <untyped> then
 factor = 1.0;
else
 factor = factor * 2.0;
end if;

// Generate the data to be used in this plot.
for i = 1 to 100 do
 y[i] = sin(i/factor);
 x[i] = i;
end for;

numberOfPlots = getNumberOfPlots(myGraph);

// In this example we assume that the graph will only contain 1 plot.
if numberOfPlots >= 1 then
```

```
 // Get the first plot.
 myPlot = getPlot(myGraph, 0);

 // Change the data in the plot.
 change2DDataPlot(myPlot, x, y);

 // Remove all plots from the graph.
 for i = 1 to numberOfPlots do
  removePlot(myGraph, 0);
 end for;
 else
 myPlot = new2DDataPlot("sin(x)", x, y);
end if;

// Add the plot.

addPlot(myGraph, myPlot);
```

The first time the above example is executed a graph and a plot are created. At that point the graph and plot can be modified interactively. Those modifications will be retained in subsequent executions of the script.

■ change2DDataPlot

FUNCTION

change2DDataPlot(thePlot, x, y)

PURPOSE

Updates the data associated with a 2-dimensional data plot without affecting other attributes of the plot like line style, line color, title, etc.

INPUT

thePlot (Plot): the plot to be modified.

x (Integer or Real Vector): vector specifying the domain (i.e., x-axis values) of the plot.

y(Integer or Real Vector): vector specifying the range (i.e., y-axis values) of the plot.

OUTPUT

No return value. The input plot will be modified.

EXAMPLE

```
aPlot = getPlot(aGraph, 0);
removePlot(aGraph, 0);
change2DDataPlot(aPlot, x, y);
addPlot(aGraph, aPlot);
```

ALGORITHM AND COMMENTS

Refer to new2DDataPlot for information on creating a new plot.

■ change3DDataPlot

FUNCTION

change3DDataPlot(thePlot, xVector, yVector, zMatrix)

change3DDataPlot(thePlot, xMatrix, yMatrix, zMatrix)

change3DDataPlot(thePlot, xVector, yVector, zVector)

PURPOSE

Updates the data associated with a 3-dimensional data plot without affecting other attributes of the plot like line style, line color, title, etc.

INPUT

thePlot (Plot): the plot to be modified.

Form 1: Surface plot.

xVector (Integer or Real Vector): vector specifying the first domain (i.e., x-axis values) of the plot.

yVector (Integer or Real Vector): vector specifying the second domain (i.e., y-axis values) of the plot.

zMatrix (Integer or Real Matrix): matrix specifying the range (i.e., z-axis values) of the plot.

Form 2: Parametric Surface.

xMatrix (Integer or Real Matrix): matrix specifying the first range (i.e., x-axis values) of the plot.

yMatrix (Integer or Real Matrix): matrix specifying the second range (i.e., y-axis values) of the plot.

zMatrix (Integer or Real Matrix): matrix specifying the third range (i.e., z-axis values) of the plot.

Form 3: Parametric Space Curve.

xVector (Integer or Real Vector): vector specifying the first range (i.e., x-axis values) of the plot.

yVector (Integer or Real Vector): vector specifying the second range (i.e., y-axis values) of the plot.

zVector (Integer or Real Vector): vector specifying the third range (i.e., z-axis values) of the plot.

OUTPUT

No return value. The input plot will be modified.

EXAMPLE

```
aPlot = getPlot(aGraph, 0);
removePlot(aGraph, 0);
change3DDataPlot(aPlot, x, y, z);
addPlot(aGraph, aPlot);
```

ALGORITHM AND COMMENTS

Refer to new3DDataPlot for information on creating a new plot.

UTILITY FUNCTIONS

■ compile

FUNCTION

compile(scriptText, scriptName)

PURPOSE

Creates a Compiled Script Symbol from a Text Symbol. Typically, you would use this function to define and create callable functions from within a HiQ-Script.

INPUT

scriptText (String): contains valid HiQ-Script text. This is a string. You can use any of the string manipulation functions to create this string.

scriptName (String): contains the name to use when compiling the function. There will be a compiled script named <scriptName>_Run generated by the compile as well as any other function definitions in the script.

OUTPUT

None. (The text is compiled.)

EXAMPLES

```
Expr = "sin(x)";
myString =
   "function myFn(x)
      return " + Expr + ";
      end function;";
compile(myString, "exampleFn");
z = myFn(1.2345);

// Result:
// The above script will generate a new function symbol called
   "myFn" and a
// Compiled Script symbol called "exampleFn_Run".
```

■ getUIInfo

FUNCTION

info = getUIInfo(flag)

PURPOSE

Returns detailed information about the mouse click that caused a script to be executed.

INPUT

flag (Language Constant): the user interface item to get information about.

<whoCalledMe>: the name of the symbol used to call the script. This is useful when a script is attached to several symbols and the desired behavior depends on which symbol invoked the script. You could, for example, use this to set a variable indicating which symbol caused the script to execute. If the script is run from within the Script Editor, the name of the compiled script is used.

<clickActionLocation>: the coordinates where the mouse was clicked relative to the upper left hand corner of the symbol which was used to call the script. This is useful when a different behavior is desired depending on where the mouse was clicked in the symbol. You could, for example, use a picture with up and down arrows and increment or decrement a counter variable depending on where in the picture the click took place. This value is {0,0} when executed from the Script Editor.

OUTPUT

info (String or Integer Vector): depending on the input flag, a string containing the name of the symbol used to call the script or a vector containing the x and y pixel coordinates where the mouse was clicked.

EXAMPLE

```
name = getUIInfo(<whoCalledMe>);
loc = getUIInfo(<clickActionLocation>);
```

name will contain the name of the symbol which caused the script to be run. loc will contain the horizontal and vertical location where the mouse was clicked relative to the upper left hand corner of the symbol that caused the script to be executed.

ALGORITHM AND COMMENTS

None.

■ progress

FUNCTION

status = progress(controlParameter)

PURPOSE

Displays, removes, and controls the behavior of a progress box, which is a graphical display that shows how much of a task has been completed.

INPUT

controlParameter (Integer): a HiQ-Script language constant that specifies the operation to be performed on the progress box, as shown in the following table:

controlParameter	Value	Effect
<startDisplay>	1	Displays the progress box. Turns autostop mode *on*.
<stopDisplay>	2	Removes progress box and resets all parameters. This is also done automatically when a script terminates.
<autoStop>	3	Turns autostop mode *on*. If autostop mode is *on*, pressing the Stop button in the progress box causes the script to terminate. This is the default setting for the progress box.
<scriptStop>	4	Turns autostop mode *off*. If autostop mode is off, the statement `progress(<checkForStop>)` returns the value1 if the Stop button has been pressed. This is useful if you want a script to perform another task after the Stop button has been pressed.
<checkForStop>	5	Causes the progress function to determine if the Stop button in the progress box has been pressed. This is only useful if autostop mode is *off*.

OUTPUT

status (Integer): when used with the language constant <checkForStop>, status gets the value 0 if the Stop button has not been pressed and 1 if it has been pressed.

EXAMPLES

Example for progress, setProgressTitle, setProgressText1, and setProgressText2 functions. This example creates a progress box that shows the progress of running a loop from 1 to 1000.

```
setProgressTitle("Progressing….");
setProgressText1("Counting…");
progress(<startDisplay>);

for i = 1 to 1000 do
   setProgressText2(numberToString(i, 0, <integer>));
   setprogressPercent(i/1000 * 100);
```

```
end for;

progress(<stopDisplay>);
```

The same example with autostop mode off:

```
setProgressTitle("Progressing….");
setProgressText1("Counting…");
progress(<startDisplay>);
progress(<scriptStop>); // Must appear after <startDisplay>

for i = 1 to 1000 do
   setProgressText2(numberToString(i, 0, <integer>));
   setprogressPercent(i/1000 * 100);
   // If you want to stop you must check.
   if(progress(<checkForStop>))then
      error("Script stopped!");
   end if;
end for;

progress(<stopDisplay>);
```

SEE ALSO

setProgressTitle, setProgressText1, setProgressText2, setProgressPercent

■ setProgressPercent

FUNCTION

setProgressPercent(percentDone)

PURPOSE

Adjusts the progress bar in a progress box. This is used to update the progress box's graphical display during a script execution.

INPUT

percentDone (Real): a number between 0 and 100 indicating the percentage completion of the operation being performed. Numbers less than 0 are treated as 0, and numbers greater than 100 are treated as 100.

OUTPUT

None. (The progress bar in a progress box is redrawn.)

EXAMPLE

Refer to the progress function for examples.

SEE ALSO

progress, setProgressTitle, setProgressText1, setProgressText2

■ setProgressText1

FUNCTION

setProgressText1(text)

PURPOSE

Sets the first line of text in the progress box. This is typically used to indicate the type of operation being performed.

INPUT

text (String): the first line of text in the progress box.

OUTPUT

None. (The first line of text in a progress box is set.)

EXAMPLE

Refer to the progress function for examples.

SEE ALSO

progress, setProgressTitle, setProgressText2, setProgressPercent

■ setProgressText2

FUNCTION

setProgressText2(text)

PURPOSE

Sets the second line of text in the progress box. This is typically used to indicate the item affected by the current operation.

INPUT

text (String): the second line of text in the progress box.

OUTPUT

None. (The second line of text in a progress box is set.)

EXAMPLE

Refer to the progress function for examples.

SEE ALSO

progress, setProgressTitle, setProgressText1, setProgressPercent

■ setProgressTitle

FUNCTION

setProgressTitle(title)

PURPOSE

Sets the title of the progress box. The default title is Progress.

INPUT

title (String): the title of the progress box.

OUTPUT

None. (The title of the next progress box created is set.)

EXAMPLE

Refer to the progress function for examples.

SEE ALSO

progress, setProgressText1, setProgressText2, setProgressPercent

CHAPTER 4
PPC FUNCTIONS

Program-to-Program Communication (PPC) is a low-level protocol available on Macintosh computers. In a manner similar to a file I/O operation, a program can open a port that you send data to or receive data from using PPC. Because it requires another application with which to communicate, it is a little more complicated than performing file I/O.

When two applications communicate, one of them is the server and the other is the client. The server must present a port to the system before the client can connect to it. The steps that create a connection are as follows:

1. The server application opens a port, giving it a name and type.
2. The client application opens a port, giving it a name and type, then tries to connect to the server's port. It can also give it information about the location of the server's port. If all the information is correct, a connection is established. If the information about the location or name of the server's port is insufficient or incorrect, a dialog box prompts the user to select the desired port.
3. Once communication is established between the client port and the server port, the two applications can talk to each other. If the client sends data, the server is expected to receive it. If the server sends data, the client is expected to read it.
4. When the server and client are finished communicating, both ports are closed.

Because the information being sent and received can be arbitrary, both client and server must agree on a protocol. This protocol is determined by the user of the PPC software.

In HiQ, PPC is established using several built-in functions. These functions are PPCOpen, PPCClose, PPCRead, and PPCWrite and are described later in this chapter.

PPC: AN EXAMPLE

The following example shows how to establish a connection between two versions of HiQ. The server application accepts a string, compiles it, and then runs it. The string must be a valid HiQ-Script. If the compilation of the script sent to the server fails, the server shuts down.

As mentioned earlier, you must first define a protocol. In this example, the data protocol specifies a four-byte integer (representing the number of text bytes to follow), followed by the text bytes (representing the HiQ-Script). The server must first read four bytes, convert these bytes to an integer, then use this integer to read that number of text bytes, and close the port.

A server script follows.

```
// Open up a PPC Port.
thePPCID = PPCOpen("", "HiQ Script Server", "Script Server", <server>);
theScript = "";
UpdateDisplay(theString);
if thePPCID > 0 then
   // An IEEE 32 bit integer is 4 bytes.
   numberOfBytes = IEEEInt32ToInteger(PPCRead(thePPCID,4));
   theScript = PPCRead(thePPCID, numberOfBytes);
   UpdateDisplay(theScript);
   PPCClose(thePPCID);

   // Now we compile the script.
   compile(theScript, "doit");

   // Now run the script.
   doit_Run();
end if;
```

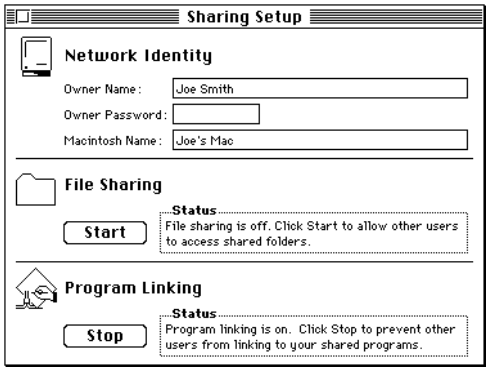
A client script follows.

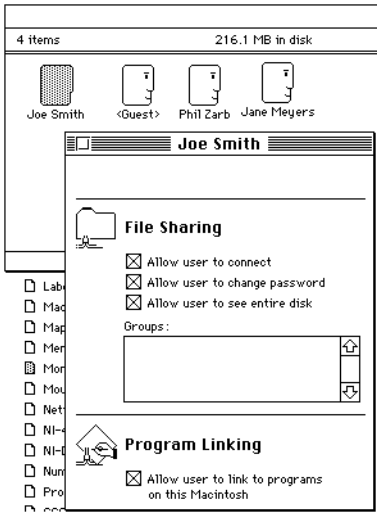
```
// Open the PPC Port.
thePPCID = PPCOpen("", "HiQ Script Server", "Script Server", <client>);
if thePPCID > 0 then
   // Send the length of the script to be compiled and run.
   // Send it as a 4 byte IEEE integer.
   PPCWrite(thePPCID, integerToIEEEInt32(StringLength(theScript)));

   // Now send the script.
   PPCWrite(thePPCID, theScript);

   PPCClose(thePPCID);
end if;
```

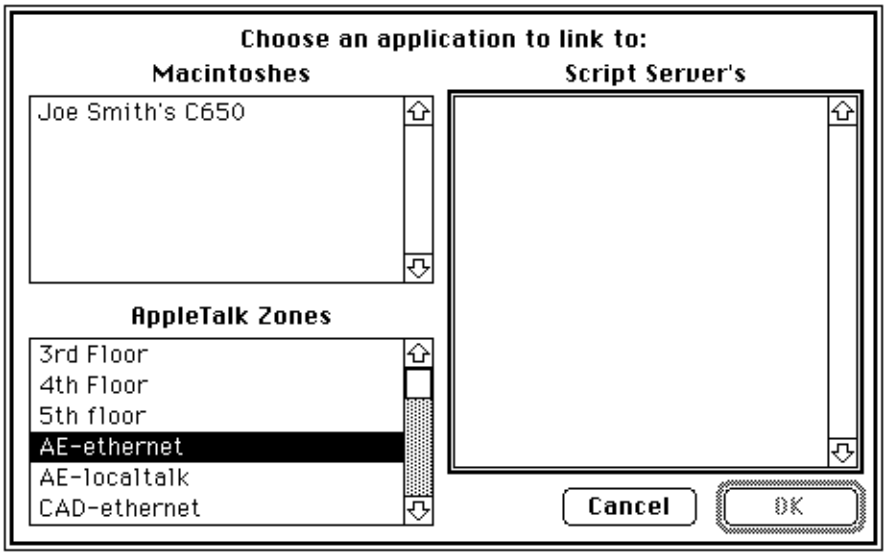
Notice that the PPCOpen function is basically identical in both cases. Before establishing a connection, Program Linking must be turned on in the Sharing Setup Control Panel on the server system, and Program Linking must also be enabled for your user name or for <Guest> in the Users & Groups Control Panel on the server machine. The Sharing Setup dialog box and a User dialog box are shown in the following illustrations.





When establishing a connection, there are two dialog boxes that might be presented. The first is the normal login dialog box that you see when mounting a file server, as shown in the following illustration. Use this to log in to the server system.

The second dialog box that might be presented is the PPC Browser dialog box, which you use to select the port and system to connect to. In the above example, the LocationName argument is passed in as an empty string. The PPC dialog box, shown in the following illustration, is then presented. This dialog box is only presented if the location name is blank or cannot be found.



A location name is composed of three parts—the computer name, the location type, and the zone. The

following is an example of a location name:

```
Joe Smith's Mac:HiQPPCToolbox@twilight
```

When initiating a server port connection, the location name and zone name are ignored. The zone name can be replaced with *, which indicates the current zone. For networks that don't have zones, always use *. The default computer name is the empty string. The default location type name is HiQPPCToolbox. The default zone name is *.

If the port cannot be opened, a -1 is returned as the port ID. After a port is opened, another port of the same name and type cannot be opened on the same computer until the first one is closed. HiQ tries to ensure against this possibility by closing all ports opened during a given script run when the script finishes. If for some reason a port is left open, you can reboot your machine to get rid of it.

■ PPCOpen

FUNCTION

thePPCPortID = PPCOpen(LocationName, PortName, PortType, mode)

PURPOSE

Open a PPC port.

INPUT

LocationName (String): a Name-Binding Protocol (NBP) location string. For mode = <server>, the location and zone are ignored. For mode = <client>, if a port of the proper name and type are found at the specified location, a connection is made. If not, the PPC Browser dialog box is presented.

PortName (String): for mode = <server> the name of the port to present to the system. For mode = <client>, the name of the port to which you want to connect.

PortType (String): the type of the port. This can be any string.

mode (Integer): <server> = 1; <client> = 2. <server> mode creates a port and presents it to the system. <client> mode creates a port and attempts to connect to the port specified by PortName and PortType at the location specified by LocationName. If not found, the PPC Browser is presented.

OUTPUT

thePPCPortID (Integer): a unique number that identifies the port that was opened. A value of -1 means that the port was not opened. Port IDs are in the range of 1000-1019. File IDs are in the range 0-19. This property can be used to distinguish between file IDs and port IDs.

EXAMPLES

See the examples at the beginning of this section.

ALGORITHM AND COMMENTS

PPC is similar to file I/O. In fact, certain import /export routines have been modified to support connections to ports. The only difference at the user level between the file I/O built-in functions in HiQ and the PPC I/O functions is the open function. The file I/O functions provide some additional functionality, such as seek and EOF, that cannot be provided for PPC.

■ PPCClose

FUNCTION

PPCClose(thePPCPortID)

PURPOSE

Close a PPC port.

INPUT

thePPCPortID (Integer): the unique number returned by PPCOpen that identifies the port being closed. After a port is closed, the port number cannot be used in any subsequent calls to PPC functions.

OUTPUT

None.

EXAMPLES

See the examples at the beginning of this section.

ALGORITHM AND COMMENTS

None.

■ PPCRead

FUNCTION

aByteString = PPCRead(thePPCPortID, theMaxNumberOfBytes)

PURPOSE

Read data from a PPC port.

INPUT

thePPCPortID (Integer): the unique number returned by PPCOpen that identifies the port being used.

theMaxNumberOfBytes (Integer): the maximum number of bytes that are to be read from the port.

OUTPUT

aByteString (String): a string containing the bytes being read. A byte string can contain characters that do not show up in the Text Editor. Unless you know that the string being read is plain text, it is not advisable to attempt to open this string in the Text Editor.

EXAMPLES

See the examples at the beginning of this section.

ALGORITHM AND COMMENTS

This function initiates a read and then waits for the read to be completed. While it is waiting, processing of background and update events is allowed. The function does not return until the read is complete. You can interrupt this with Command-period, just like any other script.

■ PPCWrite

FUNCTION

numberOfBytesWritten = PPCWrite(thePPCPortID, aByteString)

PURPOSE

Write data to a PPC port.

INPUT

thePPCPortID (Integer): the unique number returned by PPCOpen that identifies the port being used.

aByteString (String): a string containing the bytes being read. A byte string can contain characters that do not show up in the Text Editor. Unless you know that the string being read is plain text, it is not advisable to attempt to open this string in the Text Editor.

OUTPUT

numberOfBytesWritten (Integer): the number of bytes that were actually written to the port. Unless there was some kind of I/O error, this is the length of the string.

EXAMPLES

See the examples at the beginning of this section.

ALGORITHM AND COMMENTS

This function initiates a write and then waits for the write to be completed. While it is waiting, processing of background and update events is allowed. The function does not return until the write is complete. You can interrupt this with Command-period just like any other script.

CHAPTER 5
IMPORT/EXPORT FUNCTIONS

The import and export functions have been modified to use a more flexible import/export methodology. In previous versions of HiQ, these import/export built-in functions took only a filename as their first argument. This release of HiQ introduces the idea of passing either a filename, a file ID, or a port ID, thus providing additional flexibility for import. When a file ID is passed in, the import or export proceeds from the current location in the file. Therefore, you can easily import a file that has header information by reading the header information and then proceeding with the import. The following example reads the first three lines of the file and then uses the general import method to import the remaining data.

```
theFileName = GetFileName("");
theFileID = open(theFileName, "rb");
header = "";
header = header + readLine(theFileID, 1024);
header = header + readLine(theFileID, 1024);
header = header + readLine(theFileID, 1024);
theData = ImportNumeric(theFileID, "");
close(theFileID);
```

WRITING YOUR OWN IMPORT AND EXPORT FUNCTIONS

While HiQ attempts to provide tools that can handle the majority of file formats, you may need a specialized format. With this version of HiQ, a more generic approach for specifying the target (filename, fileID, or portID) has been introduced. You may want to write your own set of routines using this same method. This section presents an example of this method as well as the passing of a function as an argument to a powerful shell routine that you can easily plug in your own set of import or export filter functions. This method assumes that you want to import or export a file with an arbitrary header. The example presented here shows how to write a simple filter function that assumes a three-line header on import and an arbitrary header on export. You can easily modify these routines for any number of other methods.

Here is the shell function for import:

```
// Generic Import Shell
function importNumericWithHeader(theLocation, theMethod, theHeader,
theData)
   // Is it a filename?
   if symbolGetType(theLocation) = <string> then
      theID = open(theLocation, "rb");
      closeFile = true;
   else
      theID = theLocation;
      closeFile = false;
```

```
    end if;

    theMethod(theID, theHeader, theData);

    // If it was a file then close it.
    if closeFile then
       close(theID);
    end if;
end function;
```

Here is the shell function for export:

```
// Generic Export Shell
function exportNumericWithHeader(theLocation, theMethod, theHeader,
theData)
    // Is it a filename?
    if symbolGetType(theLocation) = <string> then
       theID = open(theLocation, "wb");
       closeFile = true;
    else
       theID = theLocation;
       closeFile = false;
    end if;

    theMethod(theID, theHeader, theData);

    // If it was a file then close it.
    if closeFile then
       close(theID);
    end if;
end function;
```

Notice the similarity between the two functions. You can use the following two filter functions as theMethod parameter in the above shell functions:

```
// An example import method.
function import3LineHeader(theID, theHeader, theData)
    theHeader = "";

    if theID >= 1000 then // PPCPort ID's start at 1000
       theHeader = PPCRead(theID, 1024);
    else
       theHeader = theHeader + readLine(theID, 1024);
       theHeader = theHeader + readLine(theID, 1024);
       theHeader = theHeader + readLine(theID, 1024);
    end if;

    // now import the remainder using the general method.
    theData = importNumeric(theID, "");

end function;
```

```
// An example export method.
function exportWithHeader(theID, theHeader, theData)

   if theID >= 1000 then // PPCPort ID's start at 1000
      PPCWrite(theID, theHeader);
   else
      write(theID, theHeader);
   end if;

   // now export the remainder using comma delimited format.
   exportNumeric(theID, theData, ",", 8);
end function;
```

Notice that for the PPC version, only one read is employed. This is because there is no equivalent to readLine in PPC. As long as the total number of bytes in the header is less than 1,024 (in this case), only one read is required.

The subsequent examples use the following to generate their data:

```
theHeader =
"Line one.
Line two.
Line three.
";
theData = {1.1, 1.2, 1.3; 2.1, 2.2, 2.3; 3.1, 3.2, 3.3};
```

To use the above shell functions by specifying a filename:

```
exportNumericWithHeader("myfile", exportWithHeader, theHeader,
   theData);
importNumericWithHeader("myfile", import3LineHeader, theHeader2,
   theData2);
```

To use the above shell functions by specifying a file ID:

```
theFileID = open("myfile", "wb+");
exportNumericWithHeader(theFileID, exportWithHeader, theHeader,
   theData);
seek(theFileID, 0, <seekFromStart>);
importNumericWithHeader(theFileID, import3LineHeader, theHeader2,
   theData2);
close(theFileID);
```

To use the above shell functions by specifying a port ID on the server side:

```
thePortID = PPCOpen("", "myfile", "example", <server>);
exportNumericWithHeader(thePortID, exportWithHeader, theHeader,
   theData);
importNumericWithHeader(thePortID, import3LineHeader, theHeaderPPC,
   theDataPPC);
PPCClose(thePortID);
```

To use the above shell function by specifying a port ID on the client side:

```
thePortID = PPCOpen("", "myfile", "example", <client>);
importNumericWithHeader(thePortID, import3LineHeader, theHeaderPPC,
   theDataPPC);
exportNumericWithHeader(thePortID, exportWithHeader, theHeader,
   theData);
PPCClose(thePortID);
```

■ importSymbol

FUNCTION

x = importSymbol(filename, type)

x = importSymbol(fileID, type)

x = importSymbol(portID, type)

PURPOSE

To import a numeric or text Symbol from an external file or a Program-to-Program Communication (PPC) port.

INPUT

filename (String): the name of the file to be imported. Enclose in quotation marks (""), if passing a string constant.

fileID (Integer): the ID of the file to import from. The file ID is returned from the open function. The import proceeds from the current read location in the file. This option is useful when a file has a header that you want to read separately. See the example above.

portID (Integer): the ID of the PPC port to import from. The port ID is returned from the PPCOpen function.

type (Integer): Defines the new Symbol type to be created (<importNumeric> or <importText>). If <importNumeric> is chosen, the general method is used. See importNumeric for more information on the general method.

OUTPUT

x (Numeric or String): contains the imported data.

EXAMPLES

```
radius = importSymbol("Radius", <importNumeric>);
```

ALGORITHM AND COMMENTS

Refer to importNumeric for more information on the general numeric import algorithm.

■ importNumeric

FUNCTION

x = importNumeric (filename, delimiterList)

x = importNumeric (fileID, delimiterList)

x = importNumeric (portID, delimiterList)

PURPOSE

To import a numeric Symbol from an external text file or Program-to-Program Communication (PPC) port.

INPUT

filename (String): The name of the file to be imported. Enclose in quotation marks (""), if passing a string constant.

fileID (Integer): The ID of the file to import from. The file ID is returned from the open function. The import proceeds from the current read location in the file. This option is useful when a file has a header that you want to read separately. See the example above.

portID (Integer): The ID of the PPC port to import from. The port ID is returned from the PPCOpen function.

delimiterList (String): Characters in this list are treated as delimiters (delimiters separate row elements; each row must end with a carriage return). Enclose in quotation marks. Each delimiter denotes a position; thus if the delimiter is a comma, importing "1,,2" results in "1 0 2." If the empty string ("") is specified, the general import method is used. See the algorithm and comments section that follows for more information on the general import method.

The following language constants have been added for your convenience for defining common delimiters:

```
<generalMethod> - Invokes the general import method.
<tabMethod>     - Uses the tab character as the delimiter.
<commaMethod>   - Uses the comma "," character as the delimiter.
<spaceMethod>   - Uses the blank character as the delimiter character.
```

OUTPUT

x (Numeric Symbol): contains the imported data.

EXAMPLES

The following file called “theData”:

```
1 2,3
4 5,6
```

can be imported using:

```
anArray = importNumeric("theData", " ,");

// Row elements may be separated by " ", or ","
```

ALGORITHM AND COMMENTS

The functions importSymbol and importNumeric use the same internal code that the Import Symbol... dialog box uses for importing numeric data.

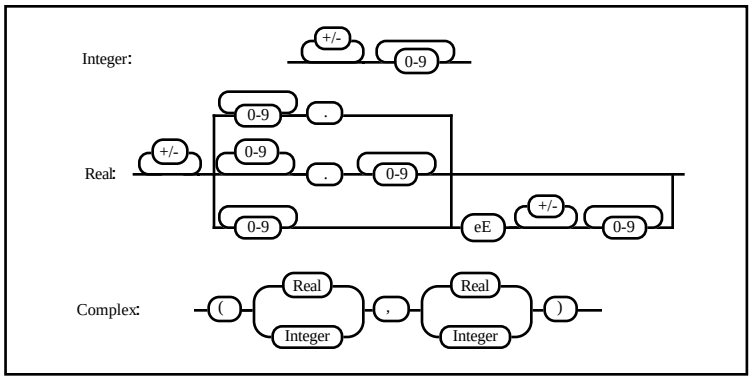
The algorithm employed dynamically determines the size and type of the data being imported. It first assumes that the item being imported is an integer scalar. As it processes the data, it creates tokens based on the parameter, delimiterList. The algorithm then attempts to decode the data, until the results are error-free—interpreting it in turn as an integer, a real, and then a complex data type. The type of decoding that succeeds determines the resulting data type. If unsuccessful, the algorithm assigns NaN (Not a Number) as the numeric value and real or complex as the data type; there is no equivalent to a NaN for integer numbers.

The dimension of the data is determined from the source by the number of items on each line and the number of lines. The maximum column dimension is determined by the line with the most numbers. For rows that have fewer columns, the trailing numbers are set to zero.

When reading the source, the text is broken up into tokens using two different methods. Each token is a sequence of numbers that represents a valid number.

The first method is the delimited text method. The list of delimiters entered determines where each token ends. If there are two tokens in a row, a zero data value is implied (for example, "1,,2" results in "1 0 2"). There is one exception to this rule. If the blank character is specified as a delimiter, multiple blanks are treated as a single blank (for example, "1 2" results in "1 2"). A carriage return determines the end of each row of data.

The second method is the general import method. This method is used by ImportSymbol when importing numeric data and can be set in importNumeric by specifying the empty string ("") as the delimiter list. This method looks for sequences of characters that form a pattern recognizable as a number. Any sequence of characters that does not form a recognizable pattern is treated as a single delimiter. This method recognizes the following numeric formats:



The algorithm does not recognize NaNs.

■ exportSymbol

FUNCTION

exportSymbol(filename, Symbol)

exportSymbol(fileID, Symbol)

exportSymbol(portID, Symbol)

PURPOSE

Export a numeric or text Symbol to an external file or Program-to-Program Communication (PPC) port.

INPUT

filename (String): the name of the file to export to. Enclose in quotation marks (""), if passing a string constant.

fileID (Integer): the ID of the file to export to. The file ID is returned from the open function. The export proceeds from the current write location in the file. This option is useful when you want to write a header as the first part of a file.

portID (Integer): the ID of the PPC port to export to. The port ID is returned from the PPCOpen function.

Symbol (numeric or text Symbol): the Symbol being exported.

OUTPUT

None.

EXAMPLES

```
radius = 1.5;
exportSymbol("radius_File", radius);
```

ALGORITHM AND COMMENTS

When exporting numeric vectors, the orientation (row or column) determines whether the vector is exported on a single line, or multiple lines with a single value on each line. When exporting matrices, each row of the matrix occupies a single line.

Tabs are used as the delimiter when exporting numeric data. To specify a different delimiter, use the exportNumeric function.

When exporting to a PPC stream, a single null byte is sent to signal the end of the export.

■ exportNumeric

FUNCTION

exportNumeric(filename, Symbol, delimiter, digits)

exportNumeric(fileID, Symbol, delimiter, digits)

exportNumeric(portID, Symbol, delimiter, digits)

PURPOSE

Export a numeric Symbol to an external file or Program-to-Program Communication (PPC) port.

INPUT

filename (String): the name of the file to export to. Enclose in quotation marks (""), if passing a string constant.

fileID (Integer): the ID of the file to export to. The file ID is returned from the open function. The export proceeds from the current write location in the file. This option is useful when you want to write a header as the first part of a file.

portID (Integer): the ID of the PPC port to export to. The port ID is returned from the PPCOpen function.

Symbol (numeric or text Symbol): the Symbol being exported.

delimiter (String): the character which separates row elements. Enclose in quotation marks. If more than one character is specified only the first character is used.

digits (Integer): for real or complex numbers, the number of digits to the right of the decimal point; ignored for integers.

The following language constants have been added for your convenience for defining common delimiters:

```
<tabMethod>     - Uses the tab character as the delimiter.
<commaMethod>   - Uses the comma "," character as the delimiter.
<spaceMethod>   - Uses the blank character as the delimiter character.
```

OUTPUT

None.

EXAMPLE

```
a = {1.5, 2.1, 3.4};
exportSymbol("theData", a, ",", 2);
```

The output file, theData, will look like:

```
1.50,2.10,3.40
```

ALGORITHM AND COMMENTS

When exporting numeric vectors, the orientation (row or column) determines whether the vector is exported on a single line, or multiple lines with a single value on each line. When exporting matrices, each row of the matrix occupies a single line.

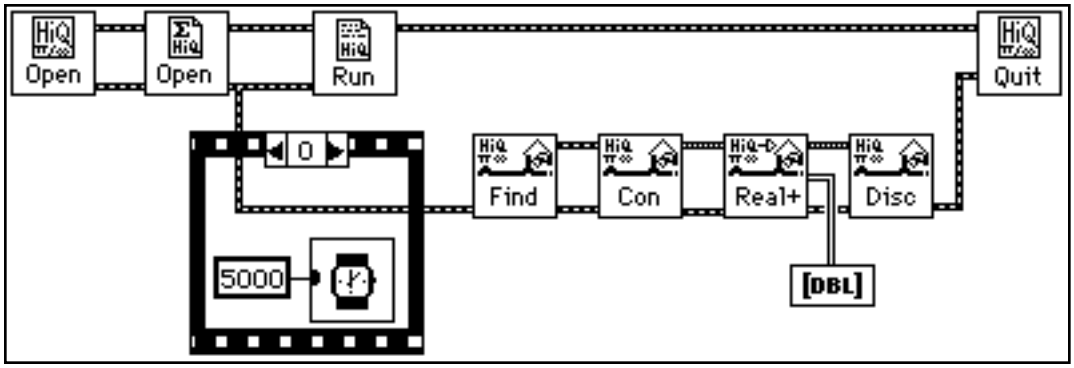
When exporting to a PPC stream, a single null byte is sent to signal the end of the export.

CHAPTER 6
LABSUITE UTILITIES

The LabSuite environment (HiQ and LabVIEW) can be tightly integrated using AppleEvents and Program-to-Program communication (PPC) to help you automate your data acquisition and numerical analysis project. LabVIEW can control HiQ using AppleEvents and can write data to or read data from HiQ using PPC, even across a network. Getting started with LabSuite automation is easy with the AppleEvents and PPC utility libraries described in this chapter. You can also get a head start on your application by using the examples in the LabSuite Examples folder. For more information on these examples, refer to the LabSuite README file in the LabSuite Tools & Examples folder.

LABSUITE: AN EXAMPLE

The figure below illustrates a LabVIEW block diagram that controls HiQ using AppleEvents and reads a matrix of real data from HiQ using PPC. In this example, you launch HiQ, open a Worksheet, then execute a HiQ-Script. That script calls the HiQ function PPC_HiQ_LV_WriteData to open a PPC port and send a matrix of data to LabVIEW.



After waiting for the script to establish a valid PPC server port, the VI prompts you to find the port, establishes a connection to that port, reads the data, disconnects from the port, and closes HiQ.

This example can be modified to write data to HiQ by replacing the HiQ PPC Read Real+ VI with HiQ PPC Write Real+. Once you have completed the initial process of launching HiQ, opening a Worksheet, executing a HiQ-Script, and establishing a PPC connection to the HiQ server, you can write to or read from HiQ as required. You do not need to go through the initializing process again. Remember that for every read in LabVIEW, there must be a corresponding write in the HiQ-Script and vice versa.

This chapter describes three methods of linking HiQ and LabVIEW together and documents the utility virtual instruments (VIs) and HiQ-Scripts used in each method. All utility VIs and HiQ-Scripts are located in the LabSuite Tools folder.

LINKING HIQ AND LABVIEW

The three methods of linking HiQ and LabVIEW include:

- transferring data using File I/O
- transferring data using PPC
- controlling HiQ from LabVIEW using AppleEvents.

FILE I/O

Even though AppleEvents and PPC are a better choice for linking HiQ and LabVIEW, you can still link them using file I/O. The HiQ and LabVIEW utilities used to implement this link are described in the LabSuite README document in the LabSuite Tools & Examples folder.

PPC

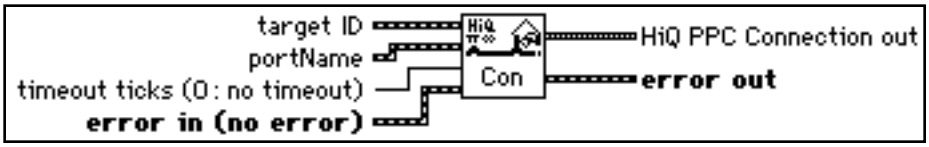
Program-to-Program communication is an efficient and reliable way to transfer data between HiQ and LabVIEW. PPC follows the client/server model of communication. The PPC server must first establish a PPC port that the PPC client can then connect with. Once this connection is made, the data flow can be bidirectional and is defined by your VI and Worksheet. Both HiQ and LabVIEW can be either a PPC server or a PPC client. When HiQ or LabVIEW writes a number of data bytes to a PPC port, the other application is expected to read exactly that number of data bytes.

Several examples illustrating a PPC link between LabVIEW and HiQ can be found in the LabSuite Examples folder. The VIs used to link LabVIEW and HiQ via PPC are found in HiQ PPC Utilities.llb in the LabSuite Tools folder and are described below.

Two HiQ utilities (PPC_HiQ_LV_ReadData and PPC_HiQ_LV_WriteData) for reading data to and writing data from LabVIEW are located in the LabSuite Tools folder and are described below.

■ (LabVIEW) HiQ PPC Connect

Establishes a connection with a HiQ PPC server.



target ID is a cluster of information describing the location of HiQ PPC server and is a complex cluster of information defined by Apple Computer, Inc. This cluster is created by the Open HiQ and the Find an Open HiQ VIs.

portName is a cluster containing the following parameters in the order listed below.

I32 **nameScript** is a 32-bit integer used in international localization that specifies the language system you are using. Use a **nameScript** value of 0 for Roman language systems (for example, English); consult *Inside Macintosh, Volume VI* for a list of available script codes.

I32 **selector** describes the format of the **type string** parameter.

1: (creator/type) Signifies that **type string** is an 8-character string; the first four characters are the creator (for example, LBVW), and the last four characters define the port type.
2: (port type string) Signifies that **type string** is a 32-character (or less) description of the service provided by the port

abc **name** is the name you give to the port. The value of **name**, which can be no more than 32 characters, is displayed in the PPC Browser dialog box list of port names. The Get Target ID VI uses **name** to identify the port.

abc **port type string** is an 8-character string; the first four characters are the creator (for example, LBVW), and the last four characters define the port type, when **selector** has a value of 1. The **type string** is a 32-character (or less) description of the service that the port provides when **selector** has a value of 2. (In almost all cases, you should specify a value of 2 for **selector**, and use a description of the service provided by the port for **type string**. Consult *Inside Macintosh, Volume VI*, for more information about other cases.)

timeout ticks (0: no timeout) if non-zero specifies the number of ticks PPC Inform Session waits for LabVIEW to establish a session before returning an error. One tick equals 1/60 of a second.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the error in cluster in error out. The error in cluster contains the following parameters.

status is TRUE if an error occurred. If status is TRUE, this VI does not perform any operations.

code is the error code number identifying an error. A value of 0 generally means no error, a negative value means a fatal error, and a positive value is a warning.

source identifies where an error occurred. The source string is usually the name of the VI that produced the error.

HiQ PPC Connection out is a cluster of information describing the active PPC session. You use this cluster as an input to other HiQ PPC VIs to identify the active PPC session. The HiQ PPC Connection out cluster contains the following parameters.

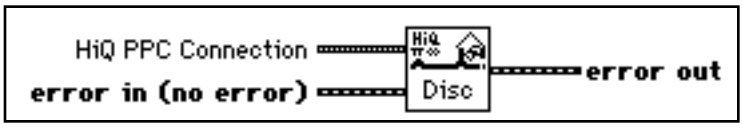
port refnum is a port reference number describing the local port associated with the current PPC session.

session refnum is a session reference number describing the current HiQ PPC session.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI.

■ (LabVIEW) HiQ PPC Disconnect

Closes the active PPC port and ends the active PPC session.



HiQ PPC Connection in is a cluster of information describing the active PPC session.

port refnum is a port reference number describing the local port associated with the current PPC session.

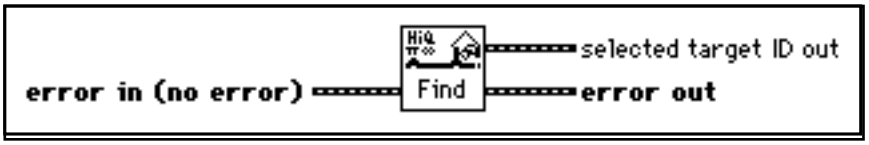
session refnum is a session reference number describing the current HiQ PPC session.

error in (no error) describes error conditions that occur prior to the execution of this VI.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) Find an Open HiQ PPC Port

Displays the PPC Browser dialog box for selecting an open HiQ PPC port on a network or on the same computer.



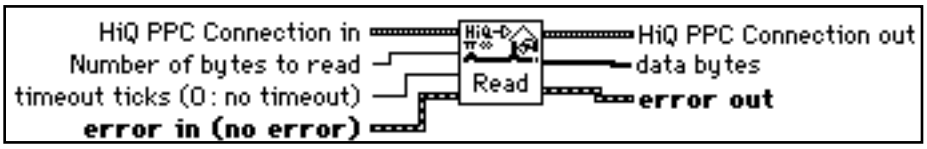
error in (no error) describes error conditions that occur prior to the execution of this VI.

target ID out is a cluster of information describing the location HiQ PPC server.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Read

Reads the specified number of bytes from the server HiQ. A HiQ-Script must be executing on the server that writes the specified number bytes.



HiQ PPC Connection in is a cluster of information describing the active PPC session.

Number of bytes to read specifies how many bytes to read from HiQ.

timeout ticks (0: no timeout) If non-zero specifies the number of ticks PPC Inform Session waits for LabVIEW to establish a session before returning an error. One tick equals 1/60 of a second.

error in (no error) describes error conditions that occur prior to the execution of this VI.

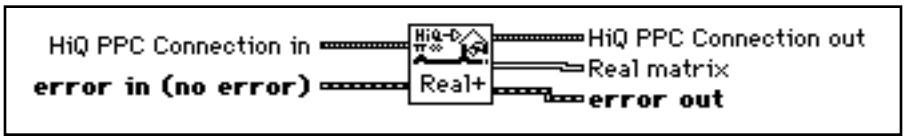
HiQ PPC Connection out is a cluster of information describing the active PPC session.

data bytes is a one dimensional array containing the data bytes received from HiQ.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Read Real+

Reads real data from the HiQ server. This VI works with the HiQ PPC_HiQ_LV_WriteData function.



HiQ PPC Connection in is a cluster of information describing the active PPC session.

error in (no error) describes error conditions that occur prior to the execution of this VI.

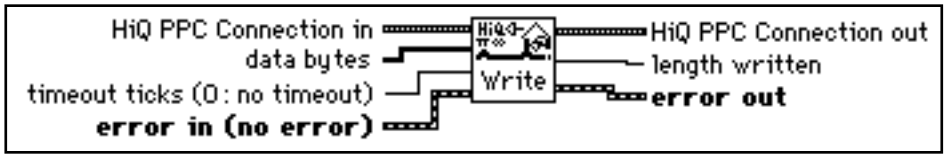
HiQ PPC Connection out is a cluster of information describing the active PPC session.

Real matrix is a two dimensional array containing the data received from HiQ.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Write

Writes a one dimensional array of one-byte data to the HiQ server. A HiQ-Script must be executing on the server to read the data.



HiQ PPC Connection in is a cluster of information describing the active PPC session.

data bytes is a one dimensional array containing the data bytes to write to HiQ.

timeout ticks (0: no timeout) if non-zero specifies the number of ticks PPC Inform Session waits for LabVIEW to establish a session before returning an error. One tick equals 1/60 of a second.

error in (no error) describes error conditions that occur prior to the execution of this VI.

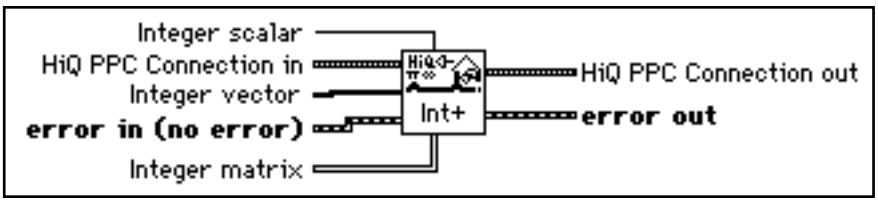
HiQ PPC Connection out is a cluster of information describing the active PPC session.

length written contains the actual number of bytes written to HiQ.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Write Integer+

Writes either scalar, vector, or matrix integer data to the HiQ server. This VI works with the PPC_HiQ_LV_ReadData function.



Integer scalar contains the scalar data to be written to HiQ.

HiQ PPC Connection in is a cluster of information describing the active PPC session.

Integer vector is a one dimensional array containing the data to be written to HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

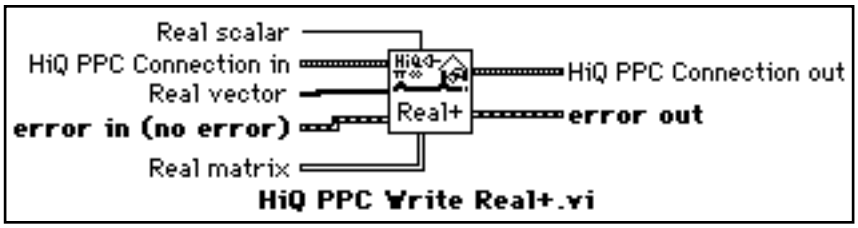
Integer matrix is a two dimensional array containing the data to be written to HiQ.

HiQ PPC Connection out is a cluster of information describing the active PPC session.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Write Real+

Writes either scalar, vector, or matrix real data to the HiQ server. This VI works with the PPC_HiQ_LV_ReadData function.



HiQ PPC Connection in is a cluster of information describing the active PPC session.

Real scalar contains the scalar real data to be written to HiQ.

Real vector is a one dimensional array containing the real data to be written to HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

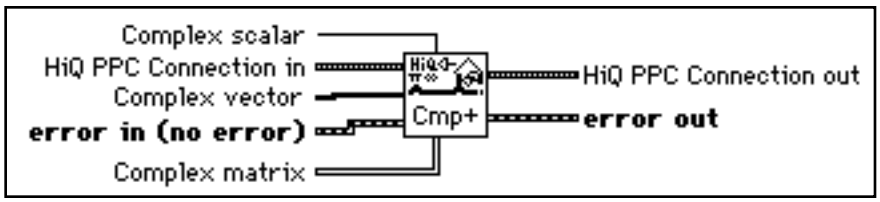
Real matrix is a two demesnial array containing the real data to be written to HiQ.

HiQ PPC Connection out is a cluster of information describing the active PPC session.

error out describes error conditions that occurred in this VI.

■ (LabVIEW) HiQ PPC Write Complex+

Writes either scalar, vector, or matrix complex data to the HiQ server. This VI works with the PPC_HiQ_LV_ReadData function.



Complex scalar contains the scalar complex data to be written to HiQ.

HiQ PPC Connection in is a cluster of information describing the active PPC session.

Complex vector is a one dimensional array containing the complex data to be written to HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

Complex matrix is a two dimensional array containing the complex data to be written to HiQ.

HiQ PPC Connection out is a cluster of information describing the active PPC session.

error out describes error conditions that occurred in this VI.

■ (HiQ) PPC_HiQ_LV_ReadData

FUNCTION

error = PPC_HiQ_LV_ReadData(PPCID, values)

PURPOSE

Read a matrix from LabVIEW. The data can be integer, real, or complex. The HiQ PPC Write Integer+, HiQ PPC Write Real+, or HiQ PPC Write Complex+ VI should be called to write the data.

INPUT

PPCID (Integer): a unique number that identifies the open PPC port to read the data from.

values (Integer Matrix, Real Matrix, or Complex Matrix): the data that is read from LabVIEW.

OUTPUT

error (Integer): 0 if no error occurred, -1 if an error occurred.

EXAMPLE

```
thePPCID = PPCOpen("", "HiQ Script Server", "Script Server", <server>);
error = PPC_HiQ_LV_ReadData(thePPCID, A);
PPCClose(thePPCID);
```

ALGORITHM AND COMMENTS

This utility reads the data format, representation, type, size, row dimension, column dimension, and data from the HiQ PPC Write+ VIs. Using this utility removes the need to handle the lower level details inside your HiQ-Script.

■ (HiQ) PPC_HiQ_LV_WriteData

FUNCTION

error = PPC_HiQ_LV_WriteData(PPCID, values)

PURPOSE

Write a real matrix to LabVIEW. The HiQ PPC Read Real+ VI should be called to read the data.

INPUT

PPCID (Integer): a unique number that identifies the open PPC port to write the data through.

values (Real Matrix): the data that is to be written to LabVIEW.

OUTPUT

error (Integer): 0 if no error occurred, -1 if an error occurred.

EXAMPLE

```
A = {1.4, 5.2, 4.5;19.3, 2.2, 0.4;1.1, 89.2, 36.4};
thePPCID = PPCOpen("", "HiQ Script Server", "Script Server", <server>);
PPC_HiQ_LV_WriteData(thePPCID, A);
PPCClose(thePPCID);
```

ALGORITHM AND COMMENTS

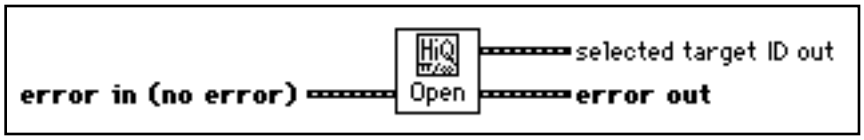
This utility writes the data type, row dimension, column dimension, and data to the HiQ PPC Read Real+ VI. Using this utility removes the need to handle the lower level details inside your HiQ-Script.

APPLEEVENTS

HiQ supports five AppleEvents: the four required events—Run, Open, Print, and Quit—and a DoScript event for executing a script in a specified HiQ Worksheet. The LabVIEW VIs that control HiQ using these events are described below.

■ Open HiQ

Displays the file dialog box to allow you to choose the HiQ executable you want to open. If you want to open a HiQ on the network, you must first mount the network drive using Chooser under the Apple menu.



error in (no error) describes error conditions that occur prior to the execution of this VI.

selected target ID out is a cluster of information describing the location of the target HiQ.

error out describes error conditions that occurred in this VI.

■ Find an Open HiQ

Displays the PPC dialog box to allow you to choose an open HiQ. HiQ must already be executing.

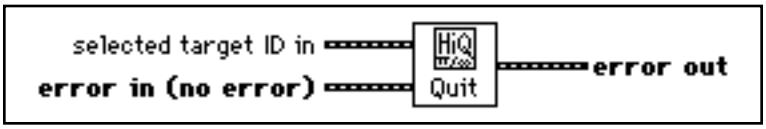
error in (no error) describes error conditions that occur prior to the execution of this VI.

selected target ID out is a cluster of information describing the location of the target HiQ.

error out describes error conditions that occurred in this VI.

■ Quit HiQ

Closes the selected HiQ.



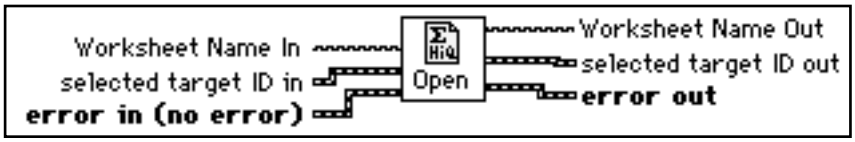
selected target ID in is a cluster of information describing the location of the target HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

error out describes error conditions that occurred in this VI.

■ Open Worksheet

Tells the selected HiQ to open a specific Worksheet. If Worksheet Name In is empty, this VI prompts for a Worksheet name using a file dialog box.



Worksheet Name In is the name of the HiQ Worksheet to print. You can specify the full pathname if required.

selected target ID in is a cluster of information describing the location of the target HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

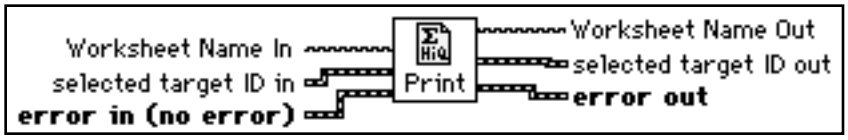
Worksheet Name Out is the name of the HiQ Worksheet that was printed.

s**elected target ID out** is a cluster of information describing the location of the target HiQ.

error out describes error conditions that occurred in this VI.

■ Print Worksheet

Tells the selected HiQ to print a specific Worksheet. If Worksheet Name In is empty, this VI prompts for a Worksheet name using a file dialog box.



Worksheet Name In is the name of the HiQ Worksheet to open. You can specify the full pathname if required.

selected target ID in is a cluster of information describing the location of the target HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

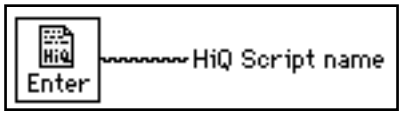
Worksheet Name out is the name of the HiQ Worksheet that was printed.

selected target ID out is a cluster of information describing the location of the target HiQ.

error out describes error conditions that occurred in this VI.

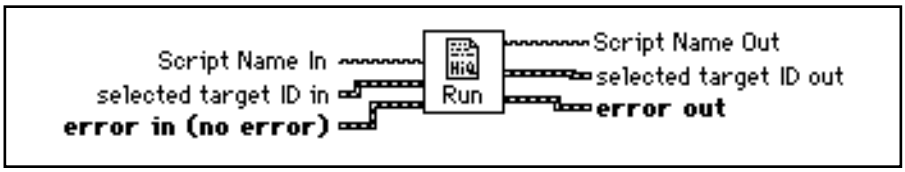
■ Enter HiQ Script

Prompts you to enter a HiQ-Script name.



■ Execute Script

Tells the selected HiQ to execute a specific HiQ-Script. If Script Name In is empty, this VI prompts for a HiQ-Script name using the Enter HiQ Script VI.



Script Name In is the name of the HiQ-Script to execute.

selected target ID in is a cluster of information describing the location of the target HiQ.

error in (no error) describes error conditions that occur prior to the execution of this VI.

Script Name Out is the name of the HiQ-Script that was executed.

selected target ID out is a cluster of information describing the location of the target HiQ.

error out describes error conditions that occurred in this VI.

APPENDIX
CUSTOMER COMMUNICATION

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a FaxBack system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

ELECTRONIC SERVICES

BULLETIN BOARD SUPPORT

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions, including a collection of example Worksheets, HiQ Demo software, and a list of common questions. From these sites, you can also download the latest instrument drivers, updates, and example programs.

For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422 or (800) 327-3077
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 1 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

INTERNET SUPPORT

You can submit technical support questions to the HiQ applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

e-mail: `hiq.support@natinst.com`

To access our FTP site, log on to our Internet host at the following address as `anonymous`, using your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

World-Wide FTP site: `ftp.natinst.com`

You can also browse our World-Wide Web site for more information on HiQ and any other National Instruments product.

World-Wide Web site: `http://www.natinst.com`

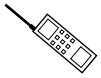
FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at the following numbers:

(512) 418-1111 or (800) 329-7177

Fax and Telephone Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.



	Fax	Telephone
Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	519 622 9311
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 71 11
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 48301892	02 48301915
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 202 2544	5 520 3282
Netherlands	03480 33466	03480 30673
Norway	32 84 84 00	32 84 86 00

Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 20 51 51	056 20 51 55
Taiwan	02 377 1200	02 737 4644
United States	512 794 5678 or 800 328 2203	512 795 8248
U.K.	01635 523545	01635 523154

TECHNICAL SUPPORT FORM

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___) _____

Computer brand _____ Model _____ Processor _____

Operating system (include version number) _____

Clock speed _____ MHz RAM _____ MB Display adapter _____

Mouse yes ___ no ___ Other adapters installed_____

Hard disk capacity _____ MB Brand _____

Instruments used _____

_____

National Instruments hardware product model ___ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

List any error messages:_____

_____

_____

The following steps reproduce the problem: _____

_____

_____

DOCUMENTATION COMMENT FORM

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

TITLE: *HiQ 2.2 Release Notes for Macintosh and Power Macintosh*

EDITION DATE: *June 1995*

PART NUMBER: *320949A-01*

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone ( ) _____

MAIL TO: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

FAX TO: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678