HiQ

HiQ® User Manual

October 1996 Edition
Part Number 321063A-01

© Copyright 1996 National Instruments Corporation. All rights reserved.

Internet Support

support@natinst.com
E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 527 2321, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

HiQ®, HiQ-Script™, and natinst.com™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

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

Preface

Overview of the HiQ Documentation xi
Using Your Online Documentation xi
Accessing Online Help xii
Organization of HiQ Online Help xii
Organization of the User Manual xiii
Conventions Used in HiQ Documentation xiii
Customer Communication xiv

Chapter 1
Getting Started

System Requirements 1-1
Installing HiQ 1-1
Launching HiQ 1-2
Learning HiQ 1-2
The HiQ Notebook 1-3
An Organizational Tool 1-3
An Interactive Analysis Environment 1-3
Linear Algebra 1-4
Data Fitting and Interpolation 1-4
Optimization 1-5
Statistics 1-5
Differential Equations 1-5
Nonlinear Analysis 1-5
Integration 1-5
Differentiation 1-6
Polynomials 1-6
ActiveX Environment 1-6
Putting It All Together 1-7
Inspecting some Example Notebooks 1-7
Working with Multiple Windows 1-7
Interacting with a Notebook and Running Scripts 1-7
Accessing Online Help 1-8
Using the Getting Started Tutorial 1-8

Chapter 2
Using a Notebook

Understanding Objects 2-1
Notebook Example: Data Fitting 2-2
Opening a Notebook 2-2
Running the Script 2-3
Viewing the Object List 2-4
Notebook Example:
Expression Evaluator Problem Solver 2-5
Entering Data and Running Scripts 2-6

Chapter 3
Creating a Notebook

Getting Your Data into HiQ 3-1
Importing a Text File as a Text Object 3-3
Importing a Text File as a Numeric Object 3-4
Importing a Binary File as a Numeric Object 3-4
Using the Custom Import Mode 3-5
Visualizing Data in 2D and 3D Graphs 3-5
Visualizing Rainfall Data (2D Graph) 3-5
Creating a Graph 3-6
Plotting a HiQ Data Object 3-7
Modifying the Graph and Plot 3-8
Working with Multiple Plots in a Graph 3-8
Visualizing Seismic Data (3D Graph) 3-9
Modifying a 3D Graph and Plot 3-10
Rotating and Zooming a 3D Graph 3-11
Analyzing Data with HiQ-Script 3-11
Entering Your Script 3-12
Running the Script and Viewing the Output 3-14
Upgrading a Notebook to a Problem Solver 3-15
Providing Input to the Problem Solver 3-15
Connecting Your Script to Input Objects 3-19
Finishing Touches for My Expression Evaluator Notebook 3-21
Using Property Pages 3-22
Taking Advantage of ActiveX Instead 3-25
Adding an ActiveX Object to the Page 3-25
Deleting Objects and Object Views 3-25
Deleting the View of an Object from a Page 3-26
Deleting an Object from a Notebook 3-26

Chapter 4
Understanding HiQ-Script

Introduction to HiQ-Script 4-1
Objects in HiQ-Script 4-1
Naming Conventions 4-2
External Statements and Functions 4-2
Scope of Variables 4-3
Comments 4-3
Constants 4-3
Syntax Highlighting 4-3
Introduction to HiQ Object Types 4-4
Numeric Objects 4-5
Automatic Data Type Promotion 4-5
Complex Numeric Types 4-5
Scalar Object Types 4-6
Vector and Matrix Object Types 4-6
Polynomials 4-10
Text Objects 4-10
Graphical Objects 4-12
Graphing Data 4-12
Graph Objects 4-13
Plot Objects 4-13
Script Objects 4-14
Function Objects 4-14
Color 4-15
Font 4-16
Object Attributes 4-16
Expressions 4-16
Operator Precedence 4-16
Function Calls 4-17
Algebraic Expressions 4-17
Algebraic Operators 4-17
Matrix and Vector Algebra 4-17
Logical Expressions 4-18
Logical Operators 4-18
Statements 4-18
Keywords 4-19
Declaration Statements 4-20
Assignment Statements 4-20
Simple Assignment 4-21
Multiple Assignment 4-21

Iteration Statements 4-21
For 4-21
While 4-22
Repeat 4-23
Flow Control Statements 4-23
If 4-23
Select 4-24
Block Escape Statements 4-25
Next 4-25
Exit 4-26
Return 4-26
User-Defined Functions 4-27
Defining Functions 4-27
Programmatically Defining Functions in HiQ-Script 4-29
Call by Reference 4-29
Calling HiQ Built-In Functions 4-30
Using Function Name as a Parameter 4-30
Performance Issues 4-31
Interacting with the User 4-32
Prompting for Input 4-32
Displaying Error Messages and Warnings 4-33
Formatting Numbers 4-34

Chapter 5
HiQ-Script Reference

Expression Syntax Reference 5-1
Precedence Operator 5-1
Attribute Operator 5-2
Binary Algebraic Operators 5-3
Binary Logical Operators 5-5
Left Side Unary Algebraic Operators 5-7
Right Side Unary Algebraic Operators 5-8
Unary Logical Operators 5-9
Subscript Operator 5-10
Vector Initialization Operator 5-11
Matrix Initialization Operator 5-12
Polynomial Initialization Operator 5-13
Color Initialization Operator 5-13
Function Initialization Operator 5-14
Font Initialization Operator 5-15

Statement Syntax Reference 5-16
assume 5-16
exit 5-17
for 5-18
function 5-19
Function Call 5-19
if 5-20
local 5-22
next 5-22
project 5-23
repeat 5-24
repeat forever 5-24
return 5-25
select 5-26
Simple Assignment 5-27
while 5-27

Appendix
Customer Communication

Glossary

Index

Figures

Figure 2-1. An Object in Active Mode 2-1

Figure 3-1. Clicking and Dragging to Place a 2D Graph Object on a Notebook Page. 3-6
Figure 3-2. A 2D Graph Object in Place and in Active (Editing) Mode 3-7
Figure 3-3. Rename Object Dialog Box 3-19
Figure 3-4. Insert Object Dialog Box Showing Some of the ActiveX Documents That You Can Insert into HiQ. 3-25
Figure 3-5. The Delete View and Delete Object Menu Items 3-26

Tables

Table 4-1. Characteristics of Objects 4-4
Table 4-2. Characteristics of Numeric Objects 4-5

Table 5-1. Valid Operations between Different Numeric Object Types 5-5

Preface

This preface tells you about your HiQ documentation, which consists of online documentation and this *HiQ User Manual*.

Overview of the HiQ Documentation

HiQ's comprehensive online documentation puts reference information where you can access it quickly as you use HiQ. You will want to consult HiQ online help frequently. There you will find information not available in the user manual, such as the following resources:

- Answers to *How Do I...?* questions
- Context-sensitive help for buttons and many other items in the user interface
- Comprehensive, easy-to-use information regarding the HiQ built-in functions (For an overview of HiQ built-in functions, consult the section *An Interactive Analysis Environment* in Chapter 1, *Getting Started* or the topic *Built-In Functions* in online help.)

The *HiQ User Manual* describes how to install and begin using HiQ to solve your data analysis and visualization problems. It includes exercises to help you feel comfortable with the basic features of HiQ, such as graph objects or the HiQ script language. It also contains useful reference information to help you write scripts. The *HiQ User Manual* is designed to help you start using HiQ as quickly as possible.

Using Your Online Documentation

This section reviews the paths you can use to access online documentation for HiQ and describes the major types of information available.

Accessing Online Help

HiQ has tooltips, context-sensitive help, and online help that you can access in the following ways.

- You access *HiQ Online Help* by selecting **HiQ Help Topics** from the **Help** menu.
- Tooltips appear when you move the mouse cursor over a button in any toolbar. The text of the tooltip tells you what the button does.
- You can view context-sensitive help in two ways. For menus, a help topic appears in the status bar at the bottom of a HiQ Notebook as you move the mouse pointer over the menu items. Within other areas of the HiQ user interface, such as dialog boxes, you see the Help button, ?. After you click on the Help button, context-sensitive help appears for the next item that you click on.
- You can use the <F1> key to access context-sensitive help for a selected item or for an area where the insertion point is located. For example, when your insertion point is located in a script and within the name of a HiQ built-in function, help for that function appears when you press <F1>.

Organization of HiQ Online Help

HiQ Online Help is organized as follows:

- The topic *Getting Started with HiQ* contains a tutorial that guides you through the process of building your own Notebook.
- *How Do I . . .?* topics are a series of small help windows that describe fundamental operations in HiQ.
- The topic *HiQ User Manual* contains a complete online version of this manual.
- The topic *Built-In Functions* contains descriptions of each built-in function in HiQ. You can access the function descriptions through either an alphabetical list or a categorical list. This section also provides information on the function categories: analysis functions, graphical functions, and utility functions.

Organization of the User Manual

The *HiQ User Manual* is organized as follows:

- Chapter 1, *Getting Started*, tells you how to install and launch HiQ and suggests ways to learn exactly what you need to know to begin analyzing your data and visualizing solutions.
- Chapter 2, *Using a Notebook*, introduces the fundamental features of the HiQ Notebook interface. The Notebook is the starting point from which you access all the tools and capabilities of HiQ.
- Chapter 3, *Creating a Notebook*, shows you how to create a Notebook to visualize data and how to create a Notebook to analyze data. The chapter also introduces more features of the HiQ interface and aspects of the HiQ-Script programming language.
- Chapter 4, *Understanding HiQ-Script*, describes the HiQ-Script programming language in detail, focusing on HiQ objects and their attributes.
- Chapter 5, *HiQ-Script Reference*, contains descriptions of HiQ-Script expressions and statements.
- Appendix, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and descriptions of terms used in this manual, including abbreviations and acronyms.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in HiQ Documentation

The following conventions are used in this manual.

< >

Angle brackets enclose the name of a key on the keyboard—for example, <Option>. Angle brackets also enclose names of constants in the HiQ-Script language.

-

A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>.

»

The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **Notebook»Create»Polynomial» Complex** directs you to pull down the **Notebook** menu, select the **Create**

	item, select **Polynomial**, and finally select **Complex** to create a complex polynomial object in a Notebook.
bold	Bold text denotes the names of menus, menu items, parameters, dialog box buttons or options.
bold italic	Bold italic text denotes a note, caution, or warning.
`bold monospace`	This font emphasizes specific entities in HiQ-Script such as keywords and built-in functions.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text for which you supply the appropriate word or value.
`italic_monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions, and for statements and comments taken from scripts and program code.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.
	The *Glossary* lists abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms.

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. You can find these forms in the Appendix, *Customer Communication*, at the end of this manual.

Getting Started

Chapter 1

This chapter tells you how to install and launch HiQ and suggests ways to learn exactly what you need to know to begin analyzing your data and visualizing solutions.

System Requirements

HiQ requires the follwing minimum system configuration.

- Windows 95 or Windows NT
- 486 CPU with coprocessor
- 8 MB RAM with Windows 95, 16 MB RAM with Windows NT
- 256-color, 640 by 480 VGA display
- 20 MB free disk space

The following specifications are the recommended system configuration for HiQ.

- Windows 95 or Windows NT 4.0
- Pentium 90 or higher
- 16 MB RAM with Windows 95, 32 MB RAM with Windows NT
- 24-bit color, 1024 by 768 display
- 20 MB free disk space

Installing HiQ

Take the following steps to install HiQ.

Installing from Disks

1. Insert disk 1 in the 3.5 inch floppy drive of your computer.
2. Double-click on `Setup.exe` in a Desktop window or in Windows Explorer to run the installation routine.
3. Follow the *Setup* instructions you see on your screen.

Installing from CD-ROM

1. Insert the CD in the CD-ROM drive of your computer.
2. Follow the *Setup* instructions you see on your screen.

By default, the HiQ installation routine creates a new folder, `C:\Program Files\National Instruments\HiQ`, which contains the following items.

- `Program` folder—contains `HiQ.exe`, HiQ help files, and related files.
- `Examples` folder—contains example Notebooks demonstrating many of the analysis and visualization capabilities of HiQ, organized by category.
- `Readme.txt` file—contains late-breaking information on HiQ, known bugs, and corrections.

☞ **Note:** ***Be sure to read the*** `Readme.txt` ***file for the latest information on HiQ.***

Launching HiQ

To launch HiQ, choose one of the following options.

Option 1: Use the Start button

1. Click on the Windows **Start** button and select the **Programs** menu.
2. Select the **HiQ** submenu.
3. Click on the HiQ icon.

Option 2: Use Windows Explorer

1. Locate the HiQ executable, `hiq.exe`, in Windows Explorer.
2. Double-click on `hiq.exe`.

Learning HiQ

Use the following resources to quickly learn about HiQ.

- Explore the Welcome to HiQ Notebook in the `Examples` folder. This Notebook gives an overview of the features and capabilities of HiQ. It is recommended that you read through the Welcome to HiQ Notebook before proceeding further.

- Follow along with the examples in Chapter 2, *Using a Notebook*, and Chapter 3, *Creating a Notebook*.
- Build the HiQ Notebook described in the getting started tutorial in the online Help.

The HiQ Notebook

HiQ uses a notebook as the basic interface metaphor. The HiQ *Notebook* has the features that you would normally associate with a real engineering or scientific Notebook, namely, pages, sections, and tabs.

An Organizational Tool

On a single page you may arrange information in a free-form manner much like in a typical drawing program. Each piece of information that you place on a Notebook page is known as an *object*. As with a real notebook there are many types of things that you can put on a page: text, numerical objects, graphs, and more. HiQ has all of these types of objects as well as some that you do not find in a regular notebook. In fact, you are not limited to placing HiQ objects on your Notebook. You can also place any ActiveX control (formerly called OLE controls), greatly expanding the capabilities of HiQ.

You can interact with the objects in HiQ Notebooks in many ways. For example, if you have a three-dimensional graph on a Notebook page, you can rotate it, zoom in or out, and make any number of other changes to how that graph looks.

As your Notebooks grow, you can add more pages. You can also organize your work in sections that make sense for your project.

An Interactive Analysis Environment

The HiQ Notebook includes a special type of object known as a HiQ-Script object. *HiQ-Script* is the built-in programming language within HiQ that allows you to access a powerful library of analysis and visualization functions.

With a script object on a Notebook page, HiQ becomes not only a compelling documentation tool but also a highly interactive analysis environment. You can build a Notebook that documents what you have done and brings the analysis to life for your reader.

HiQ provides an extensive library of analysis and visualization functions that allows you to perform everything from 3D visualization to solving systems of ordinary differential equations. Many times these flexible, built-in functions are all you need to get the results you are looking for. However, when you need something more, you can create your own user-defined functions in HiQ-Script that perform exactly the kind of analysis you need for your project.

In addition to visualization capabilities, HiQ also includes a comprehensive set of functions from the mathematical disciplines listed in this section. Refer to HiQ's online help for a complete listing of all functions, their usages, and descriptions.

Linear Algebra

HiQ provides comprehensive linear algebra functionality including the following items.

- Complete vector and matrix algebra using the natural, intuitive linear algebra syntax in HiQ-Script
- Vector and matrix norms such as Euclidean, Lp, and infinity
- Eigenvalue and eigenvector analysis
- Linear system solvers
- Matrix decompositions such as LUD, SVD, QRD, Schur, and others
- Vector and matrix transformations

Data Fitting and Interpolation

Data fitting is an excellent tool for gaining insight into real-world data. HiQ offers a very flexible suite of data fitting and interpolation tools including the following items.

- Line, exponential, polynomial, and Gauss function data fitting
- Single- and multiple-dimension nonlinear data fitting
- Linear data fitting using a set of basis functions
- Polynomial interpolation
- Spline interpolation including cubic splines, b splines, and polynomial splines

Optimization

With HiQ's optimization routines, you can minimize the cost of producing a product, maximize the capital gains in a financial portfolio, or minimize time required to perform a task. HiQ's optimization capability includes the following items.

- Nonlinear single- and multiple-dimension optimization
- Linear programming

Statistics

HiQ offers a complete set of descriptive statistics you can use to analyze a set of data.

- Mean, standard deviation, variance, covariance, correlation, and more
- Cumulative distribution and probability density functions of several types
- Histogram, quartile, and range

Differential Equations

Dynamic systems such as the motion of an object, the results of a chemical reaction, and the reaction of complex economic systems can all be described by a set of differential equations. HiQ can solve several types of differential equations including the following types.

- Nonlinear initial-value problems including stiff systems
- Linear and nonlinear boundary-value problems

Nonlinear Analysis

HiQ's nonlinear analysis capability includes the following items.

- Roots of polynomials and nonlinear functions
- Solving a system of nonlinear equations

Integration

HiQ's integration capability includes the following items.

- Data integration
- Nonlinear function integration
- Polynomial integration

Differentiation

HiQ's differentiation capability includes the following items.

- Numeric multiple derivatives and partial derivatives
- Gradient, Laplacian, Jacobian, and Hessian
- Polynomial derivatives

Polynomials

HiQ has a native polynomial object that makes polynomial algebra as easy as linear algebra. With this object you can easily work with polynomials without having to worry about the degree. Polynomial functions return polynomial objects and include the following items.

- Complete polynomial algebra using the natural, intuitive polynomial syntax in HiQ-Script
- All polynomial math including inverse, greatest common divisor, and least common multiple, and more
- Integration
- Differentiation

ActiveX Environment

A HiQ Notebook is both an ActiveX container and an ActiveX object. This means that you can embed into HiQ any ActiveX object such as a Word document or an Excel Notebook. Or if you prefer, you can embed a HiQ document as an object in other ActiveX containers.

If analysis and visualization is your focus, prepare your technical report directly in HiQ, embedding Word or other ActiveX documents as necessary. On the other hand, if you are preparing a lengthy written technical report and prefer using Microsoft Word, you can embed a HiQ document directly in your Word document.

ActiveX lets you work in your favorite environments. If you prefer WordPad or Word, or prefer the charting capabilities of Excel, use objects from these applications within your HiQ Notebook. If you are a software developer and recognize a market niche that HiQ can help you fill, create your own ActiveX object and use HiQ as your container.

Putting It All Together

Imagine preparing a management study for a water reservoir. To report the results of your study you would use text to describe the problem,

tables of the data you collected, graphs of the data, analysis performed, and conclusions on how to better manage the reservoir in the future. HiQ can handle all these reporting, analysis, and visualization tasks well. In addition, a HiQ report could include an interactive script and graph where the audience for your report could observe the effect of changing variables on inflow water levels, and outflow. This interactive component of your report in HiQ might even become a tool that can be used in the future to manage the water reservoir.

Inspecting some Example Notebooks

Looking at some example Notebooks gives you an idea of what can be done and perhaps gives you some ideas about what you may want to do in HiQ. In the next two chapters you will learn how to use and create Notebooks to visualize and analyze data. Also the `Examples` folder contains a variety of Notebooks illustrating the analysis and visualization capabilities of HiQ. Keep in mind the following features when you work with Notebooks.

Working with Multiple Windows

A HiQ Notebook uses the standard Multiple Document Interface (MDI) that Microsoft Office products like Word and Excel use. A main window exists within which multiple HiQ Notebooks may reside. You can maximize, overlay, or tile the Notebook windows you open.

Interacting with a Notebook and Running Scripts

A well designed Notebook should be self documenting, describing how to interact with the Notebook. Interaction may be as simple as entering a start time and a stop time for simulation of a system of differential equations. Or a Notebook might instruct you to choose a data file to import in order to calculate and graph the amount of rainfall in a particular area.

Most Notebooks include scripts that you can run to solve the type of mathematical or visualization problem the Notebook was designed to handle.

☞ **Note:** ***To execute a script you right-click on it and select* Run.**

Accessing Online Help

HiQ offers tooltips, context-sensitive help, and online help that you can access in the following ways.

- You can access *HiQ Online Help* by selecting **HiQ Help Topics** from the **Help** menu. There you find information not available in this manual, such as answers to *How Do I...?* questions and comprehensive, easy-to-use reference information regarding HiQ's built-in functions.
- Tooltips appear when you move the mouse cursor over a button in any toolbar. The text of the tooltip tells you what the button does.
- You can view context-sensitive help in two ways. For menus, a help topic appears in the status bar at the bottom of a HiQ Notebook as you move the mouse pointer over the menu items. Within other areas of the HiQ user interface, such as dialog boxes, you see the Help button, ?. After you click on the Help button, context-sensitive help appears for the next item that you click on.

☞ **Note:** ***You can also use the <F1> key to access context-sensitive help for a selected item or for an area where the insertion point is located. For example, when your insertion point is located in a script within the name of a HiQ built-in function, help for that function appears when you press <F1>.***

Using the Getting Started Tutorial

A good way to learn about using and creating your own Notebooks is to complete the Getting Started tutorial. If you have never used HiQ, it is recommended that you work through the tutorial because it introduces several key interface elements.

The tutorial can be accessed by selecting **Help Topics** in the **Help** menu. Then in the help Contents tab select the topic *Getting Started With HiQ* and follow the instructions on your screen. A tutorial help window appears that guides you through the process of creating a simple HiQ Notebook.

Using a Notebook

Chapter 2

This chapter introduces the fundamental features of the HiQ Notebook interface. The Notebook is the starting point from which you access all the tools and capabilities of HiQ. If you are already familiar with the basics of a HiQ Notebook but want to learn more about creating your own, refer to Chapter 3, *Creating a Notebook*.

At this point you should have installed and launched HiQ. If you need to install and/or launch HiQ, refer to Chapter 1, *Getting Started*.

Understanding Objects

All the items on a Notebook page are called objects. Notice that any object you click on becomes active and displays a border with handles. You can interact with any activated object directly on the Notebook page.

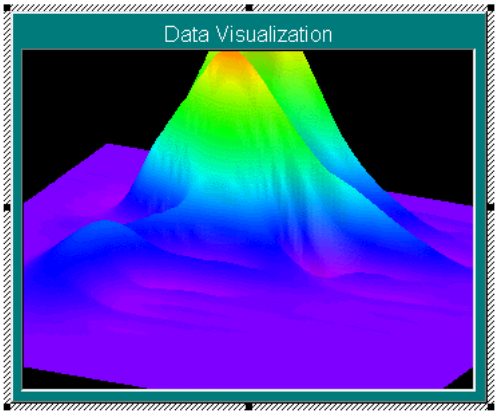


Figure 2-1. An Object in Active Mode

Objects are fundamental components of HiQ that are crucial to the use of a Notebook. Remember the following points about objects in HiQ.

- Anything contained within a Notebook is an object.
- An object is associated with a Notebook, but is not necessarily visible on the Notebook page until you place it there.
- All interactive tools of HiQ involve either the Notebook or its objects.
- HiQ uses several major categories of objects: numeric, graphic, HiQ-Script, text, and ActiveX. Each of the objects within these categories has a distinct purpose in HiQ.
- Additional aspects of HiQ objects are discussed in Chapter 3, *Creating a Notebook*, and Chapter 4, *Understanding HiQ-Script*.

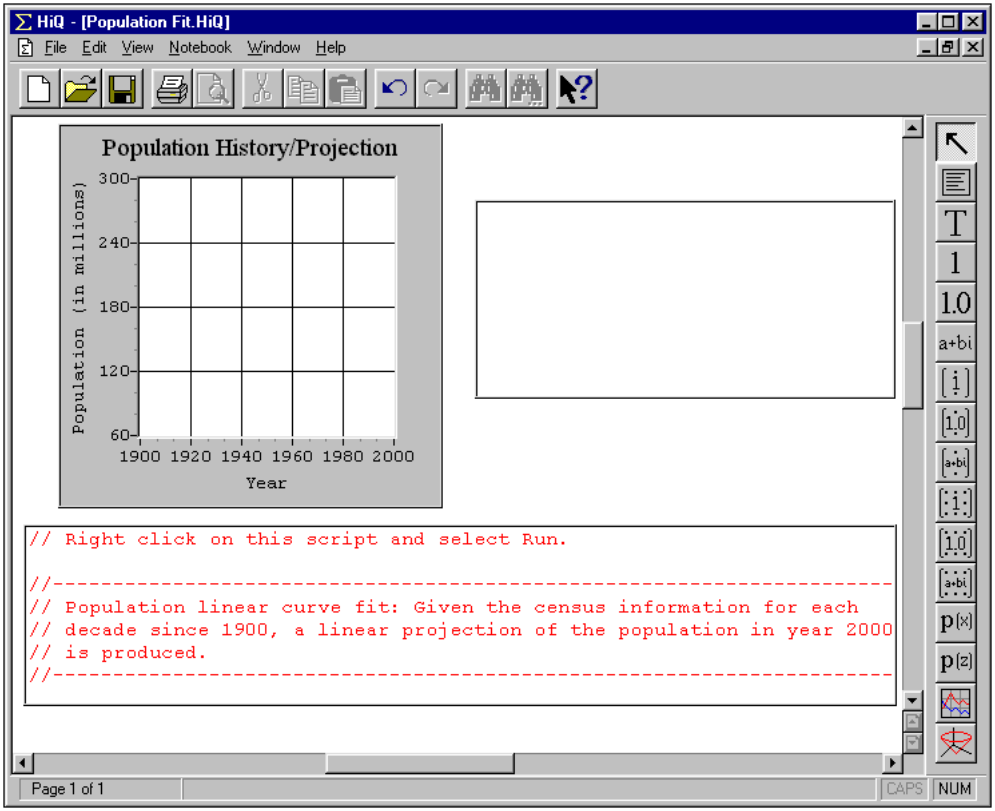
Notebook Example: Data Fitting

Now that you have had an introduction to the HiQ Notebook and objects, this section lets you work with an example Notebook that performs a data fitting analysis.

Opening a Notebook

1. Click on the Open tool, , located on the Standard toolbar at the top of the HiQ interface.
2. Open the `Examples` folder within the `HiQ` folder. Notice the many types of example Notebooks that come with HiQ. Later you can explore the different examples to get a feel for HiQ's capabilities.
3. Open the `Data Fitting` folder and select the file `Population Fit.HiQ`.

4. Click on **Open**. The Population Fit Notebook appears.

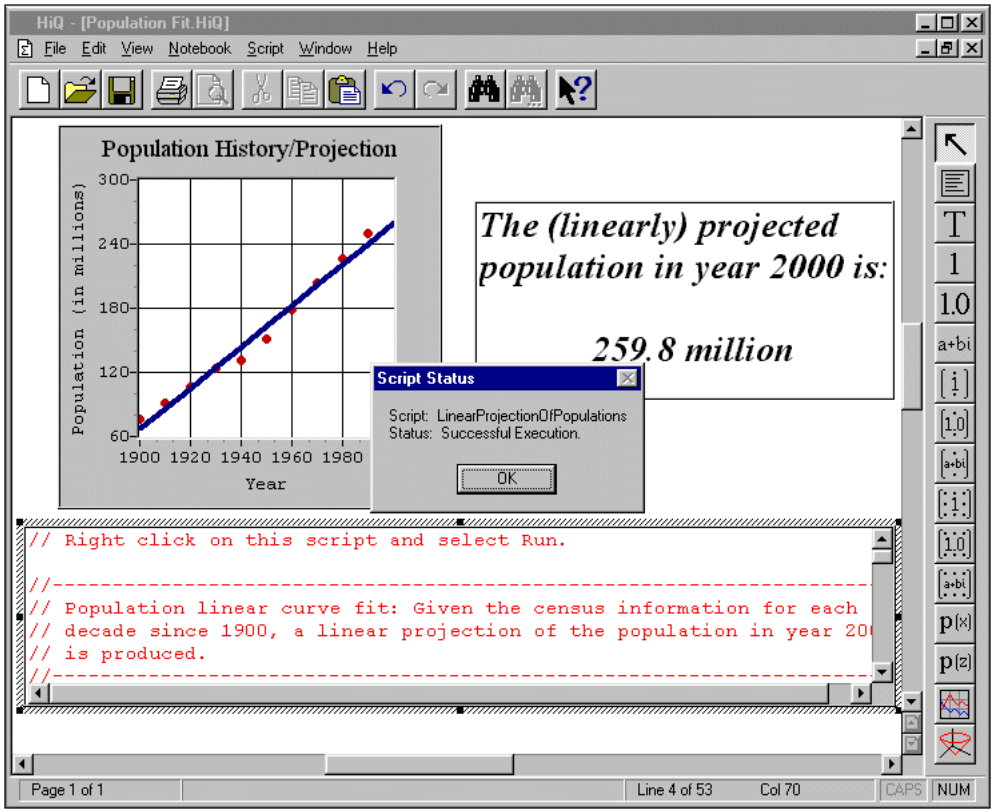


This Notebook demonstrates how HiQ can analyze a set of data, in this case population data, and graphically display it. More specifically, the Population Fit Notebook graphically presents the change in population of the U.S. over time.

This Notebook contains a script and a graph. The script object contains HiQ-Script code that performs the data fit, and the graph object displays the data fit after the script is executed.

Running the Script

Follow the instructions in the Notebook that tell you how to run the script. The output of the script appears as a plot within the graph. The plot represents the increase of U.S. population over time.



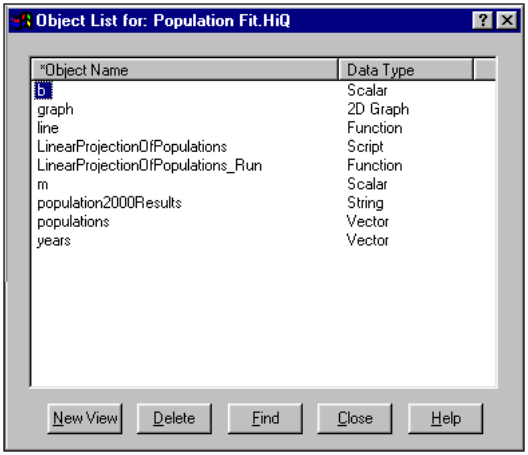
Two objects are visible on the Notebook page: the graph, the text box, and the script. However, these are not the only objects in this Notebook, as you will learn in the next section.

Scroll through the script to get a feel for how HiQ performs analysis. In this example, a single function `fit` performs the data fitting analysis. You can access most analysis capability in HiQ through calls to single functions like `fit`.

Viewing the Object List

Many, but not all, objects in a Notebook may appear on the Notebook page. To see all objects in any Notebook, perform the following steps.

1. Right-click on a blank portion of the Notebook.
2. Select **Object List...** from the pop-up menu.



The *Object list* contains all objects in the active Notebook. You may wonder where the objects that are not visible on the Notebook page came from. With the exception of the graph and HiQ-Script objects, all the objects in the list resulted from the execution of the script. As you can see, objects do not have to be visible to exist within a Notebook.

☞ **Note:** ***Any object created by a script exists in the Notebook and appears in the Object list after that script runs. The only exception is objects with local scope. See the*** *Scope of Variables* ***section of Chapter 4,*** *Understanding HiQ-Script* ***to learn about local scope.***

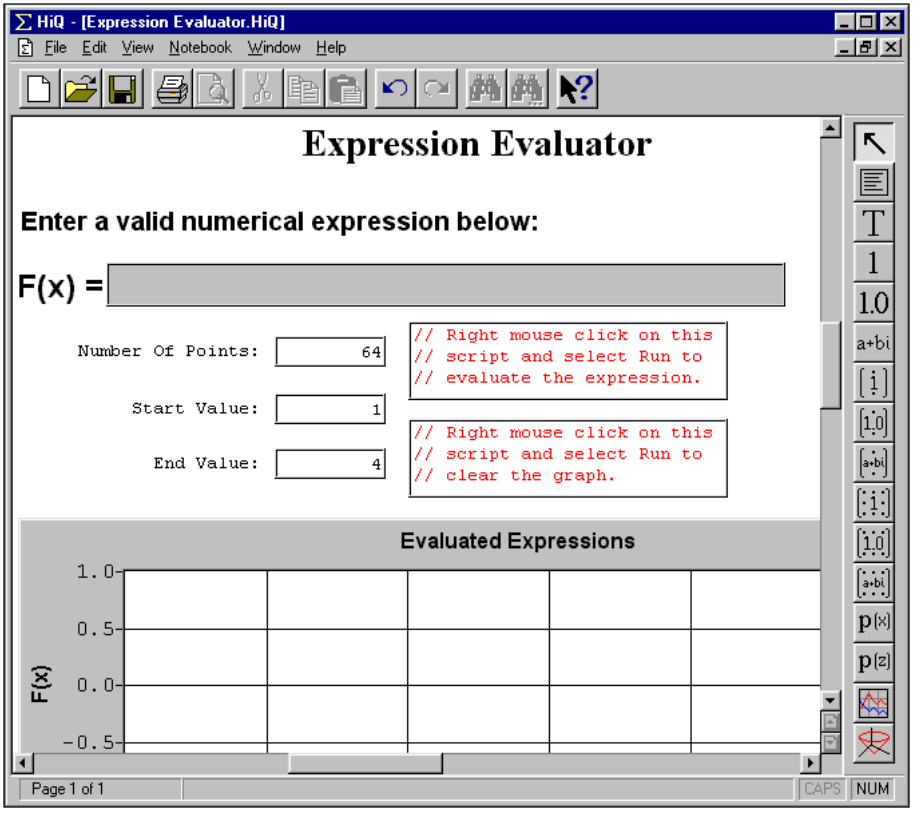
HiQ-Script is designed so that when output objects are generated, they appear only in the Object list unless placed on the Notebook page by the user. Chapter 3, *Creating a Notebook*, lets you practice placing a view of any object onto the Notebook page.

Notebook Example: Expression Evaluator Problem Solver

Problem solvers are HiQ Notebooks designed to interact with users to analyze or solve a wide range of problems belonging to a certain class. The previous example Notebook, Population Fit, does not qualify as a problem solver because it solves a single problem. For instance, Population Fit does not give you the option to fit populations from different countries. In order to analyze a different set of data you would need to modify the script. A problem solver allows the user to enter a different set of data or analysis problem without having to modify

script. The Expression Evaluator Notebook illustrates the problem solver features you can use in HiQ. Perform the following steps to learn more about the Expression Evaluator Notebook.

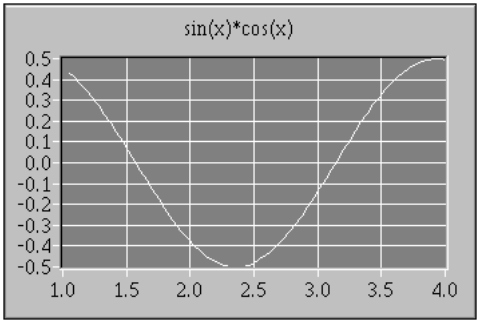
1. Click on the Open tool in the Standard toolbar.
2. Open the `Examples\Problem Solvers` folder within the `HiQ` folder.
3. Select the file `Expression Evaluator.HiQ` and click on **Open**.



Entering Data and Running Scripts

This Notebook can evaluate any mathematical expression and display its behavior graphically. Unlike the Population Fit Notebook, objects in the Expression Evaluator Notebook accept input from a user. The user enters a mathematical expression, the number of points to plot, and the interval of the domain.

1. Enter the mathematical expression `sin(x)*cos(x)` into the text object.
2. Follow the instructions in the Notebook that tell you how to run the script.



The expression you entered is fed into the numerical algorithm contained within the script. The plot that appears when you run the script is a graphical representation of the mathematical expression `sin(x)*cos(x)` that you entered. To observe the role of the numeric input of this Notebook, enter various values for the domain interval and number of plotted points and rerun the script.

Enter other mathematical expressions and run this Notebook to see how they display. Keep in mind that you must use valid operators and valid HiQ built-in functions. In this example, your expression must be in terms of *x*. Consult *HiQ Online Help* for complete descriptions of operators and built-in functions. Chapter 5, *HiQ-Script Reference* in this manual also gives information on valid operators.

You now have examined two example Notebooks that demonstrate the use of the HiQ Notebook. The Population Fit Notebook demonstrated how a Notebook performs a single data analysis, and the Expression Evaluator problem solver demonstrates how a Notebook can solve a class of problems. A problem solver Notebook accepts varied input from users without requiring them to alter the contents of the script.

The next chapter shows you how to create your own Notebook and problem solver.

Creating a Notebook

Chapter 3

After learning what you can do with a HiQ Notebook in Chapter 2, *Using a Notebook*, you are now ready to learn how to create one. This chapter shows you how to create a Notebook to visualize data and how to create a Notebook to analyze data. The chapter also introduces more features of the HiQ interface and aspects of the HiQ-Script programming language.

Getting Your Data into HiQ

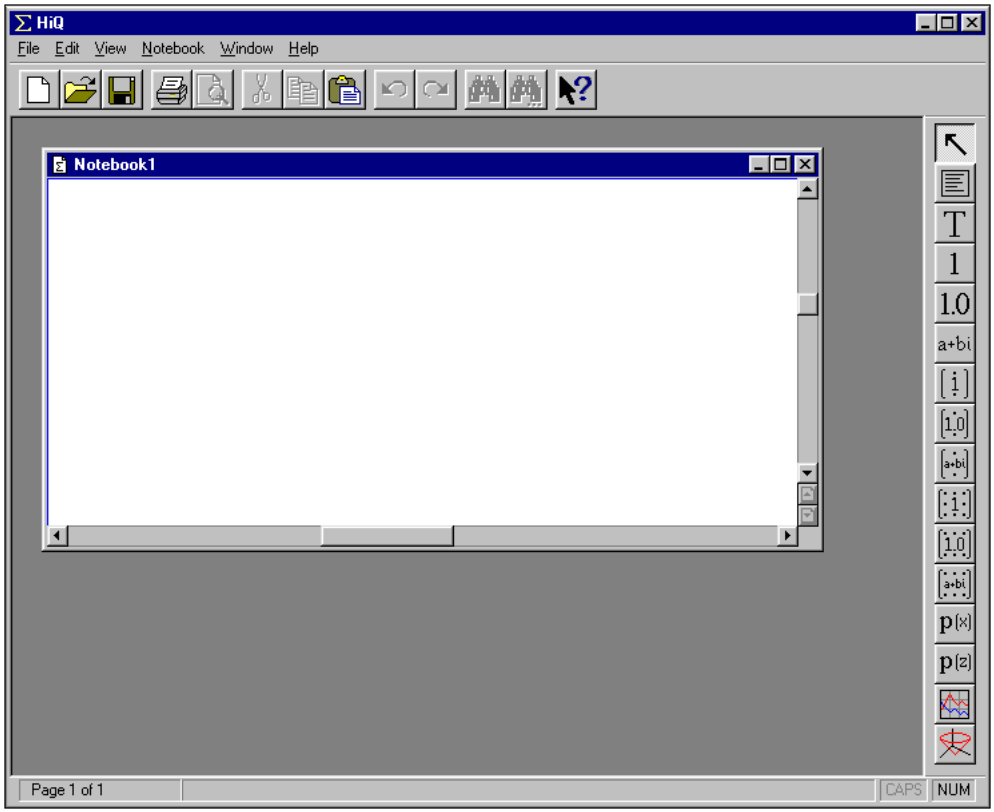
Many Notebooks you create with HiQ involve analyzing and visualizing data. Usually, this data exists in a file on your hard drive. When you need to import text or binary data from a file, you can take advantage of the HiQ Import wizard.

HiQ offers an interactive Import wizard that gives you maximum flexibility in defining the format of the data in your file. However, in most cases you can use the default import settings. When you change any of the import setting options in the Import wizard, the data preview window is updated automatically to give you immediate feedback about how the data will be imported.

Consider a meteorological data set that spans 11 years. To see how easy it is to import data into HiQ, input the file `rainfall.dat` by performing the following steps.

Open a new Notebook:

1. Click on the new Notebook tool, , in the Standard toolbar at the top of the HiQ interface. You see a new Notebook named Notebook1. Double-click on the title bar of the Notebook to maximize it.

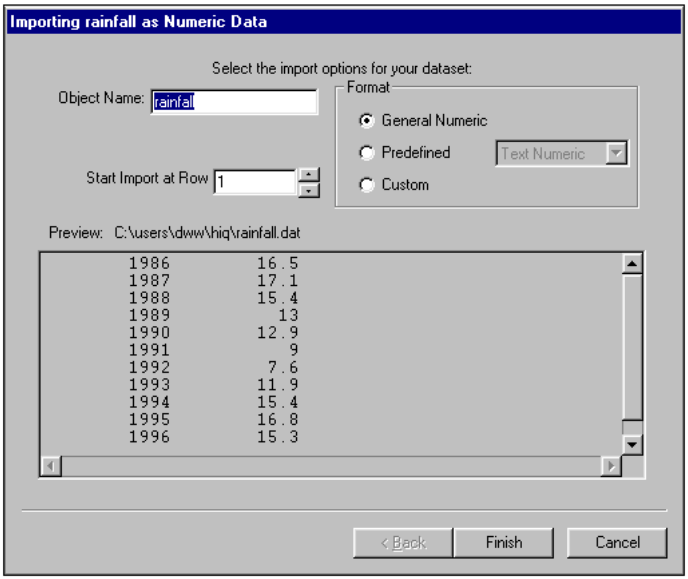


2. Pull down the **File** menu and select **Save As**.
3. Name the file `rainfall`.
4. Click on **Save**.

Import the data:

5. Select **Import Data...** from the **Notebook** menu.

6. Select the file `rainfall.dat` in the `Examples/Data` folder and click on **Open**. The Import wizard appears.



7. Click on **Finish** to import the data.

The rainfall data is now imported as a matrix object and is accessible through the Object list. You will learn how to display this data in a 2D graph in the *Visualizing Rainfall Data (2D Graph)* section.

To learn more about the Import wizard, follow the steps in the next three sections.

☞ **Note:** ***The Import wizard dialog box includes many advanced settings for data format which you can access by clicking on the* Next *and* Back *buttons. Click on the* Finish *button in the last option panel to accept all your settings.***

Importing a Text File as a Text Object

To import a text file into your Notebook, take the following steps.

1. Select **Import Data...** from the **Notebook** menu.
2. Navigate to the text file you want, select it, and click on **Open**.

3. From the Import wizard, select the appropriate import options.
   a. If the file has a `.txt` extension, click on **Finish** when you are ready to import the file.
   b. If the file is not a `.txt` file, click on the Predefined option and select **Text** from the list of predefined formats. Click on **Finish** when you are ready to import the file.

Your text file is now imported as a text object and is accessible through the Object list.

☞ **Note:** ***You can use the data preview window of the Import wizard to inspect how the data is imported in case you need to modify the import settings.***

Importing a Text File as a Numeric Object

To import a text file into your Notebook as a numeric object, take the following steps.

1. Select **Import Data...** from the **Notebook** menu.
2. Navigate to the text file you want, select it, and click on **Open**.
3. Use the Preview window to ensure that your data is going to import correctly. If your data appears correctly in the Preview window, click on **Finish**. Otherwise, take the remaining steps to customize the formatting of your data.
4. To customize the formatting of your text file, click on the Predefined option and select **Numeric (text)** from the list of predefined formats.
5. Click on **Next** and then select the appropriate delimiter method.
6. Click on **Next** again and select the appropriate type of object to create.
7. When your settings are complete, click on **Finish** in the last panel of the Import wizard.

Your text file is now imported as a numeric object and is accessible through the Object list.

Importing a Binary File as a Numeric Object

To import a binary file into your Notebook as a numeric object, take the following steps.

1. Select **Import Data...** from the **Notebook** menu.
2. Navigate to the binary file you want, select it, and click on **Open**.

3. Click on the Predefined option and select **Numeric (binary)** from the list of predefined formats.
4. Click on **Next** and select the type and byte ordering of your data.
5. Click on **Next** again and select the appropriate type of object to create.
6. When your settings are complete, click on **Finish** in the last panel of the Import wizard.

Your binary file is now imported as a numeric object and is accessible through the Object list.

Using the Custom Import Mode

The custom import option is useful when you know how to write import formats in HiQ-Script. Refer to the documentation on the `import` built-in function in the online help for details.

☞ **Note:** ***An Export wizard exists with similar capabilities for data that you wish to export from HiQ. You access the Export wizard by clicking on the object you wish to export and selecting* Export Data *from the* Notebook *menu.***

Visualizing Data in 2D and 3D Graphs

Data visualization is an indispensable tool for communicating results, interpreting your analysis, and gaining an intuitive understanding of what your data represents. The following examples show you how to use HiQ to graph two- and three-dimensional (2D and 3D) data.

Visualizing Rainfall Data (2D Graph)

You can graph most data in two dimensions, where the data is plotted against an independent variable. In the following example, you graph the rainfall data that you imported earlier in this chapter.

Creating a Graph

Start working with the `rainfall.hiq` Notebook you created earlier in this chapter. To add a 2D graph object to the Notebook, click on the 2D graph tool, , in the HiQ Tools toolbar, then click and drag on the Notebook page where you want the new graph to appear, as shown in the following figure.

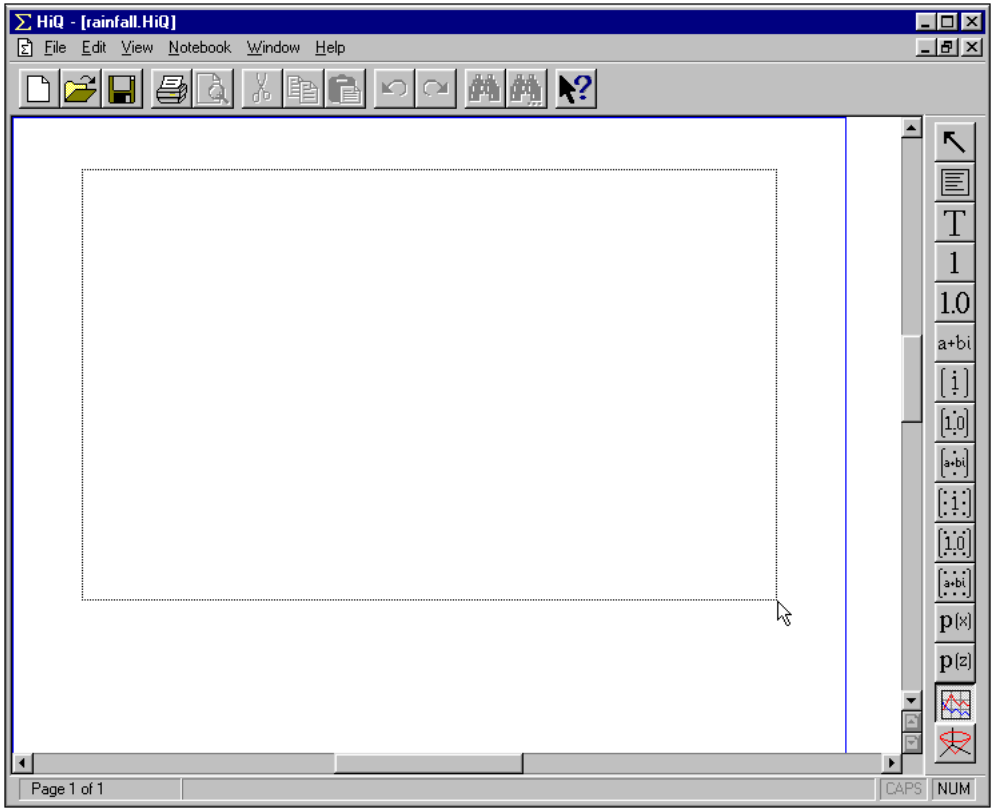


Figure 3-1. Clicking and Dragging to Place a 2D Graph Object on a Notebook Page.

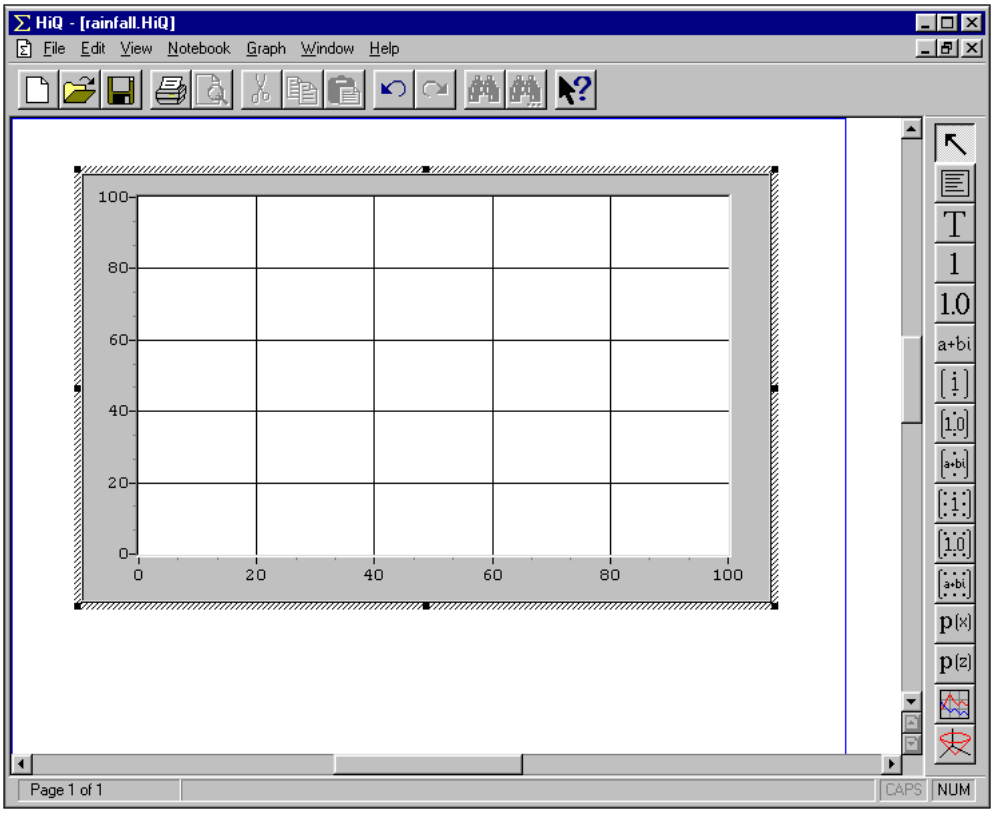


Figure 3-2. A 2D Graph Object in Place and in Active (Editing) Mode

Plotting a HiQ Data Object

Add a new plot to the graph as described in the following steps.

1. Right-click on the graph and select **New Plot...**.
2. Select Data as the range for your graph and click on the **Browse** button to the right.
3. Select the matrix `rainfall`, choose column two, and click on **OK**.
4. Select Data as the domain for your graph and click on the **Browse** button to the right.
5. Select the matrix `rainfall`, choose column one, and click on **OK**.
6. Click on **OK** to add the new plot.

The graph now contains a line plot representing the rainfall for 11 years.

Modifying the Graph and Plot

Many times the reports and presentations you create require graphs with special formatting. With HiQ, you can format your graphs to match your needs. In this example, rainfall data might better be represented by a bar graph. To modify the attributes of a plot and graph take the following steps.

1. Right-click on the graph and select **Properties...**.

To modify plot properties:

2. Click on the Plots tab.
3. In the Style area, change Plot Style to Vertical Bar.
4. In the Appearance area, change the Fill Color to red.

To modify graph properties:

5. Click on the Graph tab.
6. Enter `Yearly Rainfall` for the Graph Title.

To modify axis properties:

7. Click on the Axis tab.
8. Enter `Year` for the x-axis Title.
9. Set Axis to y axis.
10. Enter `Rainfall (inches)` for the y-axis Title.
11. Click on **OK**.

Now you have a graph that effectively communicates the trend in rainfall totals for eleven years. You can add multiple plots to this graph to view the same data in different forms or to compare different data sets. In this example, you may want to add a new plot using the same data in another form. The following section shows you how.

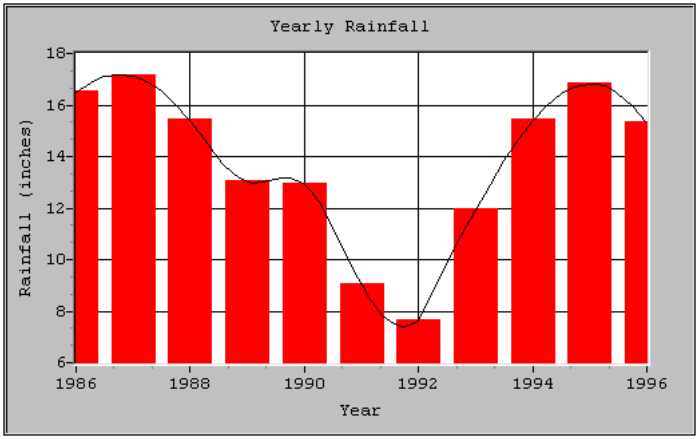
Working with Multiple Plots in a Graph

A HiQ graph can display multiple plots. In this section, you will add another plot of your rainfall data and display it in a different form.

1. Repeat the steps in the section *Plotting a HiQ Data Object* to add a second plot of the rainfall data.
2. Right-click on the graph object and select **Properties...**.
3. Click on the Plots tab. Select the rainfall plot, `(Handle)-2`, from the list of available plots in the Plots property page.

4. Change the Line Interpolation from linear to cubic spline and click on **OK**.

Your final graph should look like the one in the following illustration.



Visualizing Seismic Data (3D Graph)

For many real-world data sets—for example, terrain contours, the motion of an aircraft in three dimensions, the temperature distribution on a surface, and joint time-frequency analysis—you need to visualize data in three dimensions. The next example shows how to plot the results of a joint time-frequency analysis of seismographic data.

1. Open a new Notebook.
2. Place a 3D graph object on the Notebook page. (A preceding section, *Creating a Graph*, describes placement of an object.)
3. Right-click on the graph and select **New Plot...**.
4. Select File as the range and click on the **Browse** button to the right.
5. Select the file `seismic.dat` from the `Examples/Data` folder and click on **Open**.
6. Select Plot Against Indices.
7. Click on **OK** to add the new plot.

Note: ***Selecting Plot Against Indices plots the data using the row index for x and the column index for y.***

Modifying a 3D Graph and Plot

In addition to the properties of 2D graphs, 3D graphs have several other properties. You access all the properties from the Properties dialog box. For example, you may want to change the color of the surface plot and project the data to both the XZ and YZ planes.

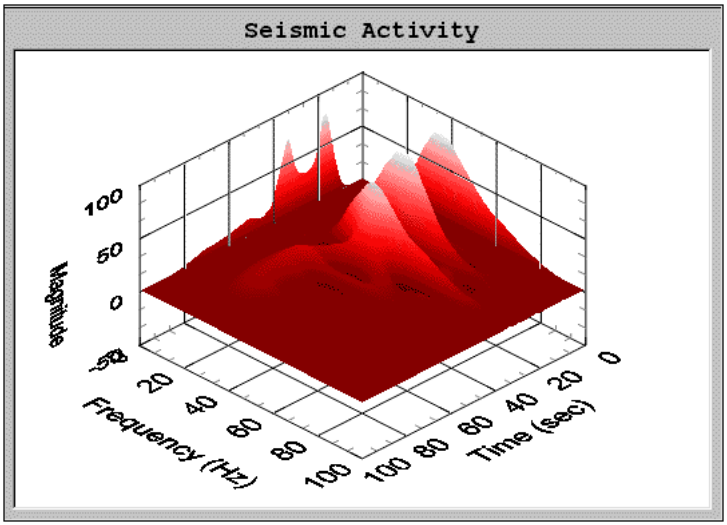
1. Right-click on the graph and select **Properties...**.

To modify the 3D Plot:

2. Click on the Plots tab.
3. Change the surface color to red.
4. Click on the **Pane** button on the lower left side to go to the second pane of the Plots property page.
5. Select Project to X-Z Plane and Project to Y-Z Plane.
6. Click on **OK**.

☞ **Note:** ***The Plots property page for 3D graphs has two panes. Click on the* Pane *buttons in the lower left of the page to move between the panes.***

The following illustration shows a graph like the one you have created that displays data regarding seismic activity as a function of time and frequency.



Experiment with the other 3D property pages, 3D and Lighting, to become more familiar with the flexibility of display formats for 3D

graphs. You can position the graph directly, using the mouse and the keyboard. The following section outlines how to use the mouse to position the graph.

Rotating and Zooming a 3D Graph

After you put a 3D graph into active mode, you can position it using the mouse and the <Ctrl> key.

☞ **Note:** ***You click on a graph object to put it into active mode (the mode for editing the graph); a shaded border appears around the object. If you do not see the shaded border, the object is probably in selected mode (the mode for sizing and moving the object as a whole). Deselect the object by clicking on a blank space in the Notebook page and then click on the object again to put it into active mode.***

To rotate the graph, click and hold anywhere in the graph area and drag the mouse up, down, left, or right to rotate the graph up, down, left, or right respectively.

To zoom the graph, click and hold anywhere in the graph area while holding down the <Crtl> key. Dragging the mouse down zooms the graph out and dragging the mouse up zooms the graph in.

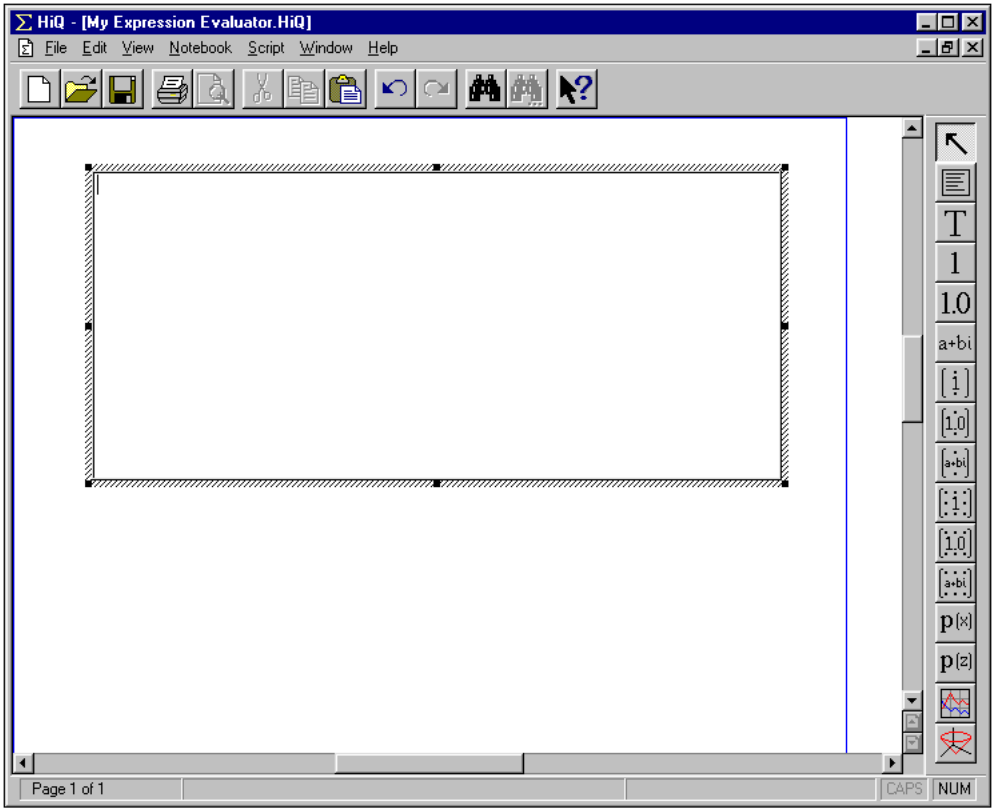
Analyzing Data with HiQ-Script

In this data analysis exercise, the objective is to create a Notebook similar to the Expression Evaluator Notebook discussed in Chapter 2. The Notebook you build in the following steps numerically evaluates mathematical expressions and graphically displays their behavior.

1. Click on the new Notebook tool, , in the Standard toolbar at the top of the HiQ interface. You see a new Notebook page.
2. Pull down the **File** menu and select **Save As**.
3. Name the file `My Expression Evaluator`.
4. Click on **Save**.

Continue building the My Expression Evaluator Notebook as follows.

1. Click on the script object, , in the HiQ Tools toolbar, which is located (by default) on the right edge of the HiQ interface.
2. On the Notebook page, click and drag to define the area for the new script object.



Entering Your Script

HiQ-Script is the programming language for HiQ you use to analyze a wide range of scientific or engineering problems. In this exercise, you will enter a block of code that numerically evaluates a mathematical expression and plots the calculated data on a graph. Unlike many other programming languages, HiQ-Script is relatively easy to use and learn.

☞ **Note:** ***HiQ automatically highlights selected elements of HiQ-Script. The*** *Syntax Highlighting* ***section of Chapter 4,*** *Understanding HiQ-Script****, tells you about highlighting.***

Enter the following script into the script object. A description follows each block of code.

Code Block 1:

```
s = “x*tan(x)”;
f = {funct:x:s};
```

This block of code converts the string `x*tan(x)` into a mathematical function (`f`).

Code Block 2:

```
x = seq(1,10,.02);
```

This block of code creates the `x` vector with elements in the interval [1,10] spaced every .02 units. All built-in functions, such as `seq`, have the default color of blue to differentiate them from other code.

Code Block 3:

```
for i = 1 to x.size step 1 do
   y[i] = f(x[i]);
endfor;
```

Here, you create the `y` vector using a **`for`** loop that calculates each element using the function `f` and the vector `x`. The size attribute of the vector x is used as the upper limit of the `for` loop. All loops and iteration statements are automatically bolded in HiQ-Script.

Code Block 4:

```
[graph1, plot1] = createGraph(x,y);
graph1.plot(plot1).line.color = <green>;
graph1.title = s;
```

This final block creates a graph to display the created plot of vectors x and y. Notice that when you enter the color constant the text color becomes magenta. All HiQ constants are magenta-colored by default in HiQ-Script. There are many HiQ constants available in HiQ-Script and not all pertain to mathematics. The constant used above specifies the plotted line color (`<green>`). Notice that on the second and third lines, the syntax for variable assignment is slightly different. In these cases, you are actually assigning a specified attribute of each variable. For

instance, on the second line you access the `line.color` attribute of `plot1` and set it to the constant `<green>`. Object attributes and constants are discussed in more detail in Chapter 4, *Understanding HiQ-Script*.

☞ **Note:** ***The* Edit *menu includes* Undo *and* Redo *commands that give you a chance to undo and repeat your actions. You can undo and repeat most of your actions in HiQ. Closing a Notebook clears the buffer file containing actions that you can undo.***

Running the Script and Viewing the Output

Each variable in HiQ-Script becomes an object in the Notebook when the script executes. To see this occur, take the following steps.

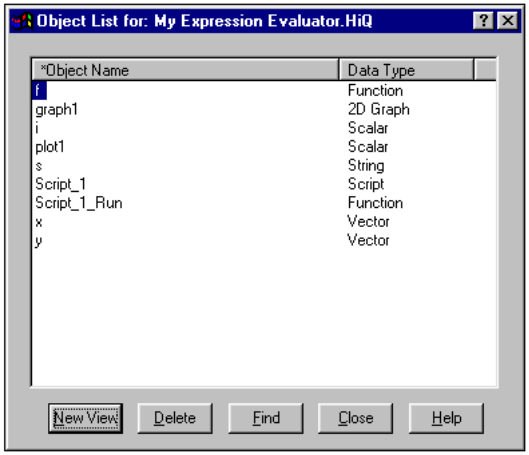
1. Right-click on the script object and select **Run** from the pop-up menu.

   If you get an error message, verify that your script exactly matches the script presented in the previous section, *Entering Your Script*.

2. Right-click on a blank area of the Notebook page and select **Object List**.

3. Compare each listed object with each assigned variable in the script. Each variable you assigned in your script now appears in the Object list, as you can observe in the following illustration.

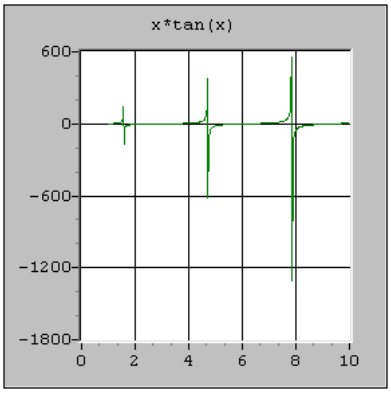
```
s = "x*tan(x)";
f = {funct:x:s};

x = seq(1,10,.02);

for i = 1 to x.size step 1 do
   y[i] = f(x[i]);
end for;

[graph1, plot1] = createGraph(x,y);
graph1.plot(plot1).line.color = <green>;
graph1.title = s;
```



4. Select `graph1` in the object list and click on **New View**. A view of the graph appears on the Notebook page.

5. Click on **Close** to dismiss the Object list.

Now that you have run your script, you see your graph on the Notebook page with the plot of the mathematical expression `x*tan(x)`.



At this point you have reached the objective of your Notebook. However, to use this Notebook to evaluate a different expression, you need to edit the HiQ-Script. Alternatively, you can provide input objects for users so that this Notebook becomes a *problem solver*, a HiQ Notebook that solves a broad class of problems without the user having to change the script. You can transform your Notebook to an interactive problem solver by performing the steps in the following section.

Upgrading a Notebook to a Problem Solver

To upgrade My Expression Evaluator Notebook to a problem solver, you need to address three main topics:

- How will the user provide input for the Notebook?
- What kind of analysis should the Notebook perform for the data?
- How do I want to present the results of the analysis?

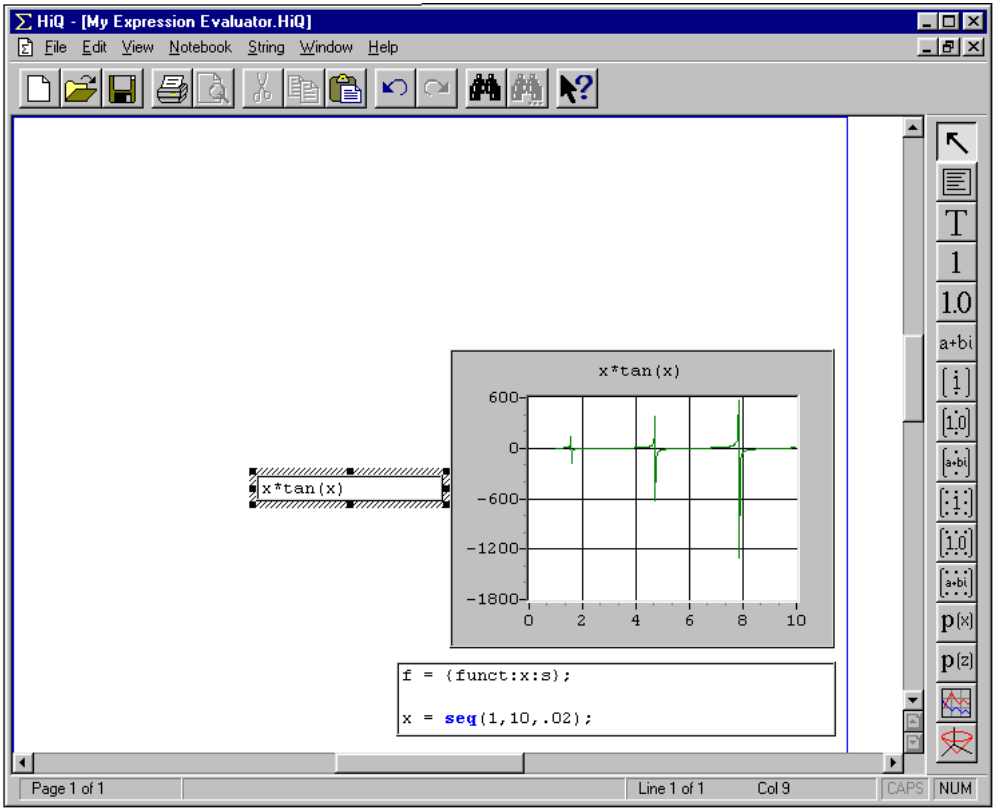
Providing Input to the Problem Solver

Consider what input is needed to execute the script. You probably already guessed that you need an object that accepts a mathematical expression, but you may also want to allow a user to input the number of plotted points and the interval of the domain. (By giving users the option to change the interval of the domain, you give them the ability to alter the smoothness of the plotted curve.) You can use a text object for input of the expression and scalar objects for the input of numeric values.

A text object already exists in the Object list to contain the mathematical expression. Although it currently contains the expression `x*tan(x)`, you can establish this object as an input object where users can enter a new expression to be evaluated.

Start with the Notebook you created in the *Analyzing Data with HiQ-Script* section.

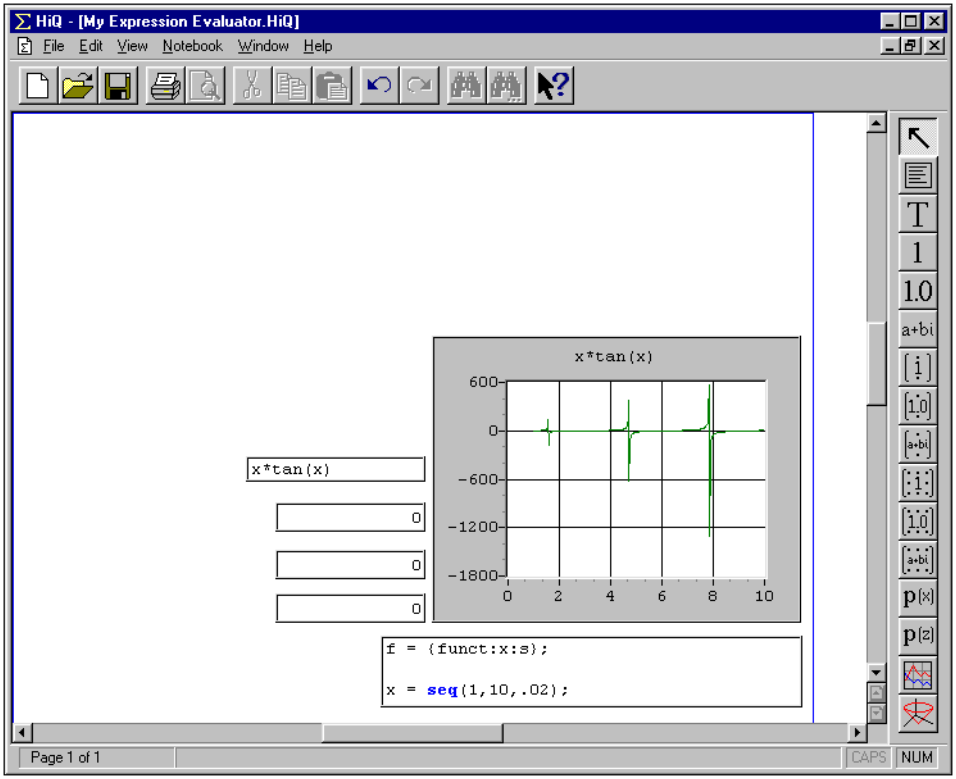
1. Right-click on a blank portion of the Notebook page and select **Object List**.
2. Select the text object called `s`.
3. Click on **New View**. A new view of the object appears on the Notebook page.
4. Click on **Close**.



You have placed a text object on the Notebook page to accept input of an expression. Now create the required scalar objects for the start, stop, and number of steps settings.

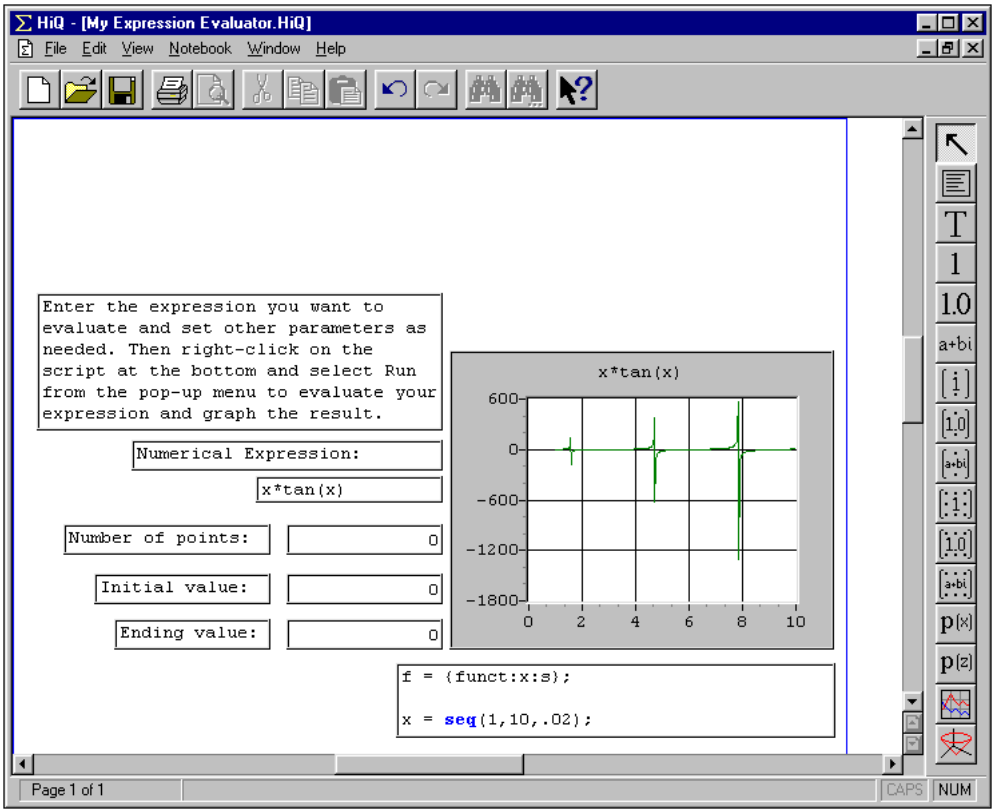
1. Click on the Real Scalar tool, 1.0, in the HiQ Tools toolbar.
2. Click and drag an area for the object on the Notebook page.
3. Repeat steps 1 and 2 to create two more scalar objects.
4. Position and size these objects as shown in the following figure.

Note: ***You can select one or more objects by clicking and dragging a box that touches the items you want to select. A border and resizing handles appear for the object(s). You can then reposition the object(s) by clicking and dragging on one of the selected objects. When you use the resizing handles of one object while others are selected, all selected objects resize.***



Note: ***You can easily align and size groups of objects by selecting them, right-clicking on one, and choosing* Align.**

5. Click on the Text tool, T, in the HiQ Tools toolbar. Then click and drag an area on the Notebook page. Repeat these actions so that you have a total of five new text objects. You will use these objects as a prompt for the user.
6. Place four of the text objects to the left of each of the four input objects.
7. Place your cursor in each of the text objects and enter your prompt text. The text objects should prompt the user to enter the numerical expression, number of points, initial value, and ending value, respectively.
8. Place the remaining text object above the four input sites and enter instructions telling users to input an expression and run the script, as suggested in the following illustration.



Though it is not mandatory, you may want to rename your scalar objects so they are easier to identify in your script.

9. Right-click on the top-most scalar object and select **Rename**.
10. Enter the name `numpts` in the New Name field and click on **OK**.

Note: ***To rename an object, you right-click on the object and select* Rename.**

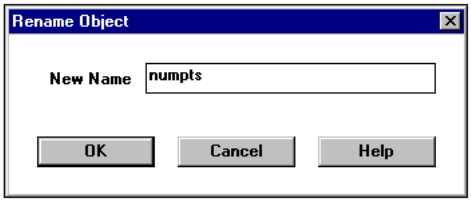


Figure 3-3. Rename Object Dialog Box

11. Rename the next two scalar objects, `initial` and `ending`, respectively.

Now your problem solver has all the input objects required for easy interaction with users. Your next task is to modify your script.

Connecting Your Script to Input Objects

The script you have written performs most of the analysis you need. However, you must revise the script so that it uses the new input objects your created in the previous section.

The changes you must make in your script are relatively minor. First, incorporate the scalar objects into the script. As you recall, you use the scalar objects on the Notebook to enter the number of points, initial value, and ending value. You need to replace specific numerical values currently in the script with the names of the input objects you created, as described in the following steps.

1. Modify the following line.

   ```
   x = seq (1,10,.02);
   ```

   Change it to read as follows.

   ```
   x = seq(initial,ending,(ending-initial)/numpts);
   ```

2. On the Notebook, change the values of the scalars `initial`, `ending`, and `numpts` to `1`, `10`, and `450` respectively.

3. Delete the lines that assign to variables having the same name as objects that appear on the Notebook. There is one such line that you need to delete from this script:

   ```
   s = “x*tan(x)”;
   ```

4. Modify the following line that creates a graph.

   ```
   [graph1,plot1] = createGraph(x,y);
   ```

   Change it to plot new data to an existing plot as follows.

   ```
   plot1 = addPlot(graph1,plot1,x,y);
   ```

5. It is customary to include comments when entering any programming code. You may want to include some comments to describe sections of your script. A comment must appear in the following format and is automatically colored red.

   ```
   //comments
   ```

Your updated script complete with comments, should look like the following script.

```
//Right-click on this script
//and select Run.

//Converts a string into a function.
f = {funct:x:s};

//Creates the x vector.
x = seq(initial,ending,(ending - initial)/numpts);

//Calculates the y vector.
for i = 1 to x.size step 1 do
   y[i] = f(x[i]);
endfor;

//Adds the data to the existing
//plot, plot1.
plot1 = addPlot(graph1,plot1,x,y);
graph1.plot(plot1).line.color = <green>;
graph1.title = s;
```

Now your script is ready to execute. Try it before going further with this exercise. Right-click on the script object and select **Run** from the pop-up menu. If you get an error message, verify that your script exactly matches the script above.

☞ **Note:** ***Watch the status bar at the bottom of the Notebook window for the following information.***

- ***The current Notebook page number and page count***
- ***The current line and column number when a script or text object is active***
- ***A red, diamond-shaped icon when a script is running (one icon for each running script)***
- ***The mode of the keyboard (one of several possible states)***
- ***Information about the currently selected object***

Finishing Touches for My Expression Evaluator Notebook

You might want to spice up your Notebook to make it more user-friendly and to make it look better as a technical report. Here are some design recommendations to introduce you to more HiQ features.

- Use the text object to form a title at the top of your Notebook.
- Use the text object to display a brief description of your problem solver.
- Change the color, size, and location of the objects currently on the Notebook.

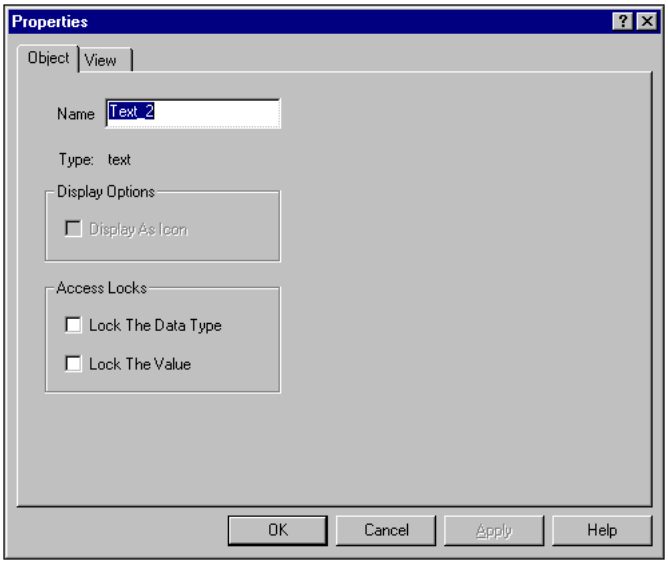
You use the Properties dialog box of each HiQ object to set these and other properties.

☞ **Note:** ***All HiQ objects including the Notebook itself have properties (or attributes) that are easy to set. You can right-click on any object and select*** **Properties** ***from the pop-up menu to see the Properties dialog box for that object. From the Properties dialog box, you can then choose the settings you want for that object.***

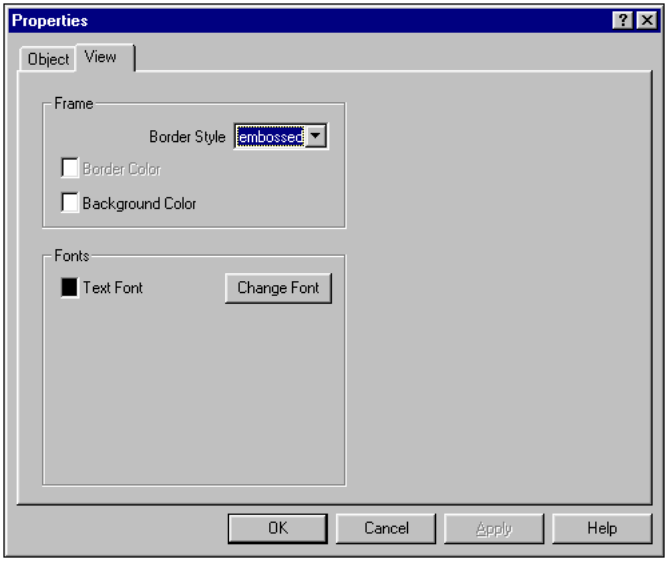
Using Property Pages

Take the following steps to learn how to use Properties dialog boxes to change colors and fonts.

1. Access the Properties dialog box of the text object that reads “Numerical Expression:” by right-clicking on the object and selecting **Properties** from the pop-up menu.



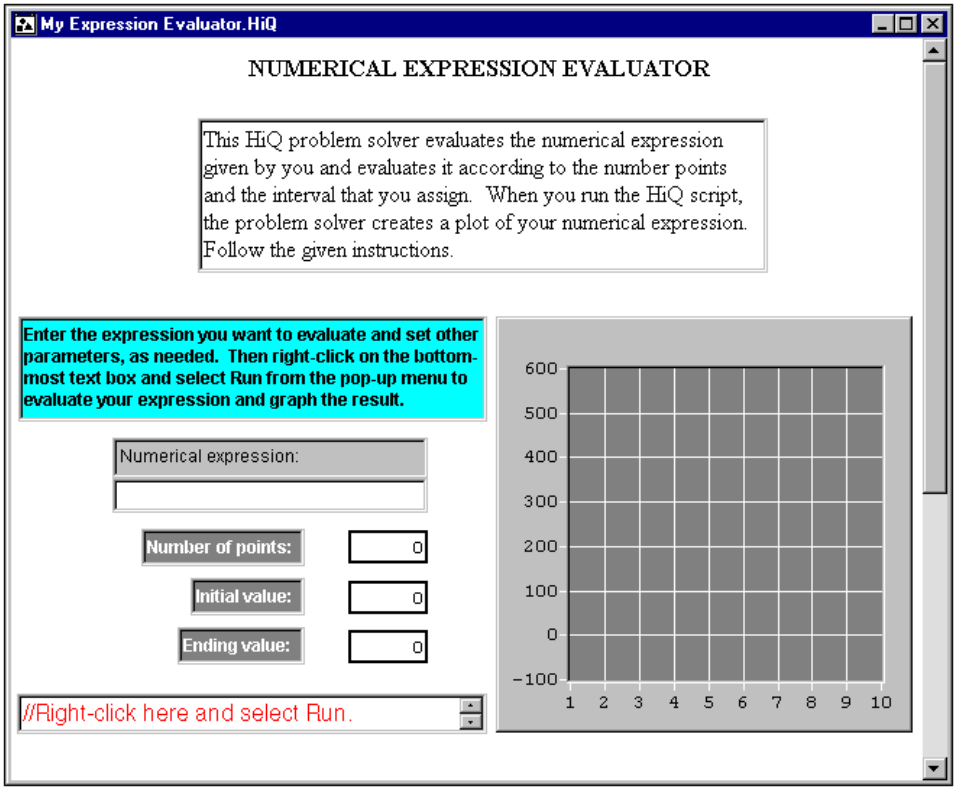
2. Click on the View tab in the Properties dialog box.



3. Click on the Background Color button. A color palette appears.
4. Select a gray from the Basic colors section of the **Color** dialog box.
5. Click on **OK**.
6. Click on **Change Font**. The font dialog box appears.
7. Change the font style to bold and the point size to 12. Then click on **OK**.
8. Click on **Apply** and then **OK** to apply and save your changes to the text object.

Fonts and color properties are among the many properties that you can change in HiQ. You may want to explore the properties of other objects to become familiar with the HiQ Properties dialog boxes, and learn how a Notebook can be touched up to become an appealing user interface or

technical report. The final draft of My Expression Evaluator Notebook might look like this one:



Now that you have finished touching up My Expression Evaluator Notebook, print out a hard copy.

1. Select **Print** from the **File** menu.
2. Choose your settings.
3. Click on **OK**.

You now have a printed version of your new Notebook. For examples of how to create other Notebooks, including problem solvers, be sure to refer to your `Examples` folder in the HiQ folder. These examples demonstrate the power and flexibility of HiQ. The remaining pages in this chapter present tips about working with objects in HiQ.

Taking Advantage of ActiveX Instead...

Instead of using a text object for the problem description above, you might want to use a Microsoft Word document, which is an ActiveX object.

☞ **Note:** ***There are two types of objects in HiQ: native objects and ActiveX objects. Native object types appear in the HiQ Tools toolbar. Unlike ActiveX objects, you can create, access, and modify regular HiQ objects in your scripts.***

Adding an ActiveX Object to the Page

To place an ActiveX object in a Notebook, you can use the **Edit»Insert Object** menu item. When you insert an ActiveX object, you can chose to either link the object to the original file, or you can embed the object directly into HiQ. In both cases, the objects appear the same on the Notebook page, but a linked object does not store its data in a Notebook file and an embedded object does. When a linked object file changes, the object displayed in the HiQ Notebook automatically updates. However, if the linked object file outside of the Notebook is moved or deleted, the object within the Notebook is no longer visible in HiQ.

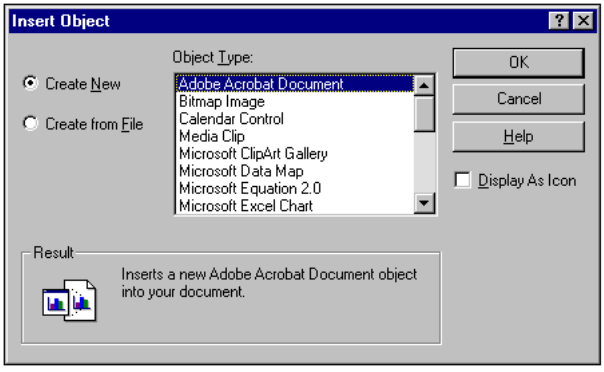


Figure 3-4. Insert Object Dialog Box Showing Some of the ActiveX Documents That You Can Insert into HiQ.

Deleting Objects and Object Views

The objects in your HiQ Notebooks can exist without displaying on the Notebook page. For example, you might want to display the plot of a data set without displaying the numeric object that is the source of the data. You should remember this distinction when you want to delete an

object from the Notebook or when you want to delete only a view of that object.

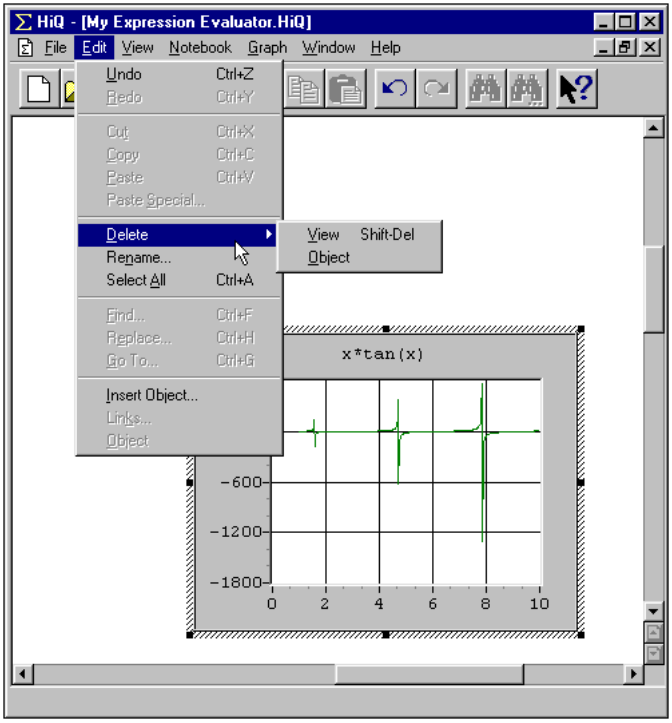


Figure 3-5. The Delete View and Delete Object Menu Items

Deleting the View of an Object from a Page

Deleting the view of an object from the Notebook page is not the same as deleting the object. When you delete a view, the object remains in the Notebook. You can still access the object through a script or through the Object list.

To delete a view, select the object by clicking on it and then select **Edit»Delete»View**. The selected view is removed from the Notebook page, but the object remains in the Notebook.

Deleting an Object from a Notebook

To delete an object entirely from a Notebook, select the object by clicking on it and then select **Edit»Delete»Object**. The object and all of its views are removed completely from the Notebook.

The exercises in this and the preceding chapter prepare you to create your own HiQ Notebooks. The remaining chapters describe the HiQ-Script language in depth.

Remember to consult the HiQ online help. There you will find information not available in this manual, such as answers to *How Do I...?* questions and comprehensive, easy-to-use reference information regarding the HiQ built-in functions.

Understanding HiQ-Script

Chapter 4

This chapter describes the HiQ-Script programming language in detail, focusing on HiQ objects and their attributes.

Introduction to HiQ-Script

No single analysis software package has all the algorithms that you need. That is why HiQ has a built-in scripting language that you can use to build the algorithm you need to solve your problem. Previous experience with programming languages is useful, but is not necessary. HiQ-Script is powerful and flexible because it draws from the most useful features of FORTRAN, Pascal, and C, and adds access to the depths of the HiQ computation engine. HiQ-Script is intuitive because it uses words rather than special symbols to describe programming constructs as much as possible. In some ways it is similar to pseudo-code commonly used in algorithm descriptions, yet you can compile and run HiQ-Script.

The exercises in Chapter 3, *Creating a Notebook*, can help you become familiar with writing a HiQ script. If you have not already done so, you may want to complete the exercises in that chapter.

To create a script, select the script tool in the HiQ Tools toolbar and click and drag on a Notebook page where you want the object to appear. You can then immediately begin typing. You find standard text editing features in the **Edit** menu, including **Cut**, **Copy**, and **Paste**.

Objects in HiQ-Script

An object is a fundamental unit of many modern programming languages, including HiQ-Script. You can think of an object as a data container that has a name. Examples of HiQ objects include text, script, numeric, and graphic objects. Because HiQ is an ActiveX container, it can contain any ActiveX object, including objects from Microsoft Word and Excel.

Naming Conventions

To name a variable in HiQ-Script, you can use all letters of the alphabet (both uppercase and lowercase), digits (0–9), and the underscore (_). Variable names must start with a letter or an underscore. In addition, all names with two or more consecutive underscores are reserved for HiQ itself.

All names of objects and functions must be unique. All names are case-sensitive in HiQ-Script (except for names of built-in functions, keywords, and constants). For example, the variable `a` and the variable `A` are two unique variables. HiQ's library of built-in functions generally avoids single-letter function names. This way, single-letter names are available when you need to create variables such as `f(x)` or `g(x)`.

The names of built-in HiQ functions, keywords, and constants are *not* case sensitive. For example, whether your HiQ script calls `sin`, `Sin`, `SIN`, or even `sIn`, HiQ recognizes a call to the `sin` built-in function.

External Statements and Functions

A HiQ script consists of a series of “external” statements and any number of user-defined functions. An *external statement* is a HiQ-Script statement outside a user-defined function. These statements are executed when you run a HiQ script. The first three lines in the following example are external statements within a script called `myScript`. The function `myfunct` is defined on the last three lines of the following example. `myfunct` executes only because it is referenced in the external statement on the third line.

```
x = 5;
y = 6;
a = myfunct(x,y);

function myfunct(x,y)
   return 2*x + y;
end function;
```

HiQ treats the external statements in a script as a special user-defined function that has no input parameters. When you compile a script, HiQ creates this function and names it by appending `_Run` to the script name. In the preceding example, HiQ names the function that results from the compiling of the script `myScript_Run`. You can call this new function from any other script.

Scope of Variables

HiQ has two scopes for variables, local and project. A *local variable* exists only inside a function definition and only while the script is executing that function. When the function exits, the variable no longer exists. On the other hand, a *project variable* is visible to the Notebook and thus continues to exist after a function exits. Project variables appear in the Object list. The default scope is local inside a function; otherwise, the default scope is project.

Comments

You can insert comments into a HiQ script. Comments are ignored and do not increase the execution time of a script.

The comment notation is a double slash, `//`. The compiler ignores any text following these characters on a line.

```
// This is a comment that begins a line.
a = 1; // This is a comment that follows the statement.
```

Constants

HiQ-Script contains many useful constants. While mathematical constants can be used in any statement, other constants only make sense when used with specific functions or object attributes. All HiQ constants are enclosed in angle brackets, <>. For example, you would insert the constant pi (π) in a line of script as follows.

```
x = 2*<pi>*freq;
```

In the following example, the constant `<L2>` specifies a type of norm for the `norm` function.

```
y = norm(x, <L2>);
```

A complete list of constants appears in the *HiQ Constant Reference* section of *HiQ Online Help*.

Syntax Highlighting

The script object offers a variety of syntax highlighting options you can set to make your script more readable. You access these options in the Properties dialog box of any script object, under the View tab. You can change the font and font characteristics of the main script, comments, keywords, built-in functions, and constants.

Introduction to HiQ Object Types

All objects in HiQ have a specific object type. For example, a Notebook may contain a graph object, a plot object, a script object, and several matrix objects. HiQ automatically manages the object type of an object, as necessary. This means you do not have to explicitly declare an object type for an object, such as integer scalar, real matrix, or 3D graph. For example, if you wish to evaluate `y = 2*x`, where `x` is an integer scalar, HiQ automatically gives `y` the object type integer scalar. If `x` is a real number, HiQ makes `y` a real number. If `y` already exists as a real matrix, it is automatically changed to a real scalar. For more details, see the *Automatic Data Type Promotion* section that follows in this chapter.

Table 4-1. Characteristics of Objects

Object Type	Description
Numeric	A scalar, vector, or matrix composed of integer, real or complex numbers. See Table 4-2.
Text	A text string of any length (`"This is a text string."`).
Plot	A graphical representation of a single function or data set.
Graph	A graphical representation of a collection of plots. Can be 2D or 3D graphs.
Script	An executable script.
Polynomial	A special function of one variable, x, in multiple powers, for example, $3x^2 + 2x + 10$. A polynomial can be real or complex.
Font	Contains a string for the font name and an integer for the font size.
Color	Contains three integers representing the red, green and blue components of a color.

☞ **Note:** ***A logical object type is not available in HiQ. You can implement the logical type by representing false as an integer zero, and true as any non-zero integer. In addition, keep in mind that the keywords*** `true` ***and*** `false` ***evaluate to 1 and 0, respectively.***

Numeric Objects

Numeric object types are divided into two categories: numeric type (integer, real, or complex) and object type (scalar, vector, or matrix).

HiQ has nine numeric object types (integer, real, and complex scalars, vectors, and matrices) and two polynomial types (real and complex polynomials).

Table 4-2. Characteristics of Numeric Objects

Numeric Types	Integer	32-bit signed integer. Range: – 2147483648 to 2147483647
	Real	64-bit floating point numbers. Range: – 1.7976931348623159e+308 to 1.7976931348623159e+308
	Complex	Two 64-bit floating point numbers.
Object Types	Scalar	A single numeric value.
	Vector	A 1D array of numeric values.
	Matrix	A 2D array of numeric values.

Automatic Data Type Promotion

Within the numeric object group HiQ promotes the data type of an object automatically. HiQ promotes a data type when not promoting would result in a loss of precision. This promotion makes expressions such as `1/3` result in a real number rather than an integer, unlike most programming languages. If you really did want an integer result you can force the expression back to integer by using the `int` function. For example, `int(1/3)` would result in the value 0.

☞ **Note:** ***Automatic data type promotion makes it easier to program with HiQ-Script, but also increases execution time. To learn how to improve performance of your HiQ scripts, read the*** *Performance Issues* ***section later in this chapter.***

Complex Numeric Types

You can reference the real and complex parts of a complex object using object attribute notation (the “dot” syntax). For example, you set a variable to the real part of a complex scalar `z` as follows.

```
realPartofZ = z.r;
```

Conversely, to set the real part of the complex scalar `z` to the current value of `a`:

```
z.r = a;
```

The same syntax applies to the imaginary part:

```
imagPartofZ = z.i;
z.i = b;
```

Scalar Object Types

The scalar types (integer, real and complex) behave in HiQ as they do in most other programming languages, with the additional feature of automatic data type promotion. For details, see the *Automatic Data Type Promotion* section earlier in this chapter.

Vector and Matrix Object Types

Vectors are 1D arrays of numeric data and matrices are 2D arrays of numeric data. Defining these objects allows HiQ to perform vector and matrix arithmetic using the familiar syntax of linear algebra. For example, if **A** and **B** are matrices, then their product is expressed in a HiQ-Script statement as it is in linear algebra:

```
C = A*B;
```

Likewise, the inverse of A is expressed in HiQ-Script as follows.

```
D = A^-1;
```

Consider some other examples:

```
y = A*x;
y = A';        //transpose
y = x'*A*x;
```

☞ **Note:** ***The function and variable names you create in HiQ are case-sensitive, so you can adhere to the mathematics convention of capitalizing names of matrices and putting the names of vectors in lowercase letters.***

Initializing

You may want to initialize all or part of a vector or matrix within a script. To initialize a vector or matrix, enclose the element values in curly braces, `{}`. Separate the elements in a row with a comma, and separate each row with a semicolon, as in the following examples.

```
A = {1,2;3,4}; // is a 2x2 matrix
B = {1,2,3,4}; // is a 1x4 matrix
```

To create a vector, add the `vector` keyword (or the letter `v`) and a colon after the left curly brace and before the data, as in the following examples.

```
a = {vector:1,2,3,4};        or
a = {v:1,2,3,4};     // a four-element vector
```

Refer to Chapter 5, *HiQ-Script Reference* for a detailed discussion of these issues in the *Vector Initialization Operator* and *Matrix Initialization Operator* sections.

☞ **Note:** ***The default keyword in the initializer syntax is*** `matrix`**.**

As with any other statement, you can split the expression across multiple lines. For example, a matrix assignment would be more readable if it were written in the following form.

```
A = {matrix: 1,1;
             1,1};
```

When you initialize a matrix within a for loop, HiQ automatically dimensions the object. Instead of relying on this automatic dimensioning, you can set the dimensions of your matrix, as in the following statements.

```
A = {matrix: 0}; // Makes A an integer matrix.
A.rows = m;
A.cols = n;
```

`A` is the name of your matrix and `m` is the number of rows and `n` is the number of columns in your object.

Subscripting

Often you need to access a single element or a range of elements in a vector or matrix. In HiQ-Script you use square brackets, `[]`, containing an index (for vectors) or two indices (for matrices), to reference any element or range of elements. For example, you identify an element in row 3, column 4 of a matrix **A** as `A[3,4]`.

You may use that element in an expression as follows.

```
z = 4*A[3,4] + sin(theta);
```

Also, you may assign a value to that element:

```
A[3,4] = 2.5;
```

You can access elements of a vector **v** similarly, as shown in following expression which multiplies an element from a matrix and an element from a vector.

```
y = A[3,4]*v[2];
```

You refer to the entire dimension of a vector or matrix by using an asterisk (`*`) for an index. For example, you refer to all the elements of the third column of a matrix **A** as `A[*,3]`.

A range of elements in a vector or matrix is accessed using a range for an index. A range is defined using the colon (`:`), and an optional starting index and ending index. For example, the range `1:5` refers to elements one through five. You can also use the asterisk (`*`) to mean "first" when you place it to the left of the colon and "last" when you place it to the right of the colon. For example, the range `5:*` refers to elements five through the last element, whereas the range `*:6` refers to the first element through element six. If you omit a starting or ending index, then "first element" and "last element" are assumed, respectively. Thus, when you refer to a range of `:` (colon), you designate all elements of an index.

Consider the 5 x 5 matrix, **A**.

```
 1   2   3   4   5
 6   7   8   9  10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
```

To refer to a submatrix consisting of the first two rows and columns of **A**, use the following syntax.

```
A[1:2,1:2]
```

The following submatrix results.

```
1   2
6   7
```

To refer to a submatrix consisting of the last two rows and columns of **A**, use the following syntax.

```
A[4:5,4:5]      or
A[4:*,4:*]      or
A[4:,4:]
```

The following matrix results.

```
19  20
24  25
```

To refer to the third column of **A**, use the following syntax.

```
A[:,3]
```

The following vector results.

```
 3
 8
13
18
23
```

You can use expressions anywhere in an index. For example, the following expressions are valid: `A[i,j]` and `b[i*3:j*4]`.

☞ **Note:** ***If an expression evaluates to a non-integer value or a value outside the valid range of the data, a run-time error results.***

Automatic Typing and Dimensioning

You can explore automatic typing and matrix dimensioning with the following simple statement regarding matrices **A** and **B**.

```
A = B;
```

A takes on the dimensions of **B**, whether **A** is a new matrix or was previously defined.

If **b** is a vector with as many elements as **A** has columns, a row of **A** may take on the values of **b** as follows.

```
A[4,:] = b;
```

Row four of **A** now equals vector **b**.

HiQ promotes numeric data types as necessary, as in the case of the following script.

```
a = 5;
a[2,3] = 2;
```

When HiQ runs this script it generates the following value for `a`.

```
0 0 0
0 0 2
```

The following script represents a different case.

```
a[2,3] = 2;
a = 0;
```

When HiQ runs this script it generates the value 0 for a.

Polynomials

HiQ has real and complex polynomial objects. These objects make polynomial algebra in HiQ-Script as easy as linear algebra. You can declare a polynomial using one of the following syntaxes.

```
myPoly = {poly: 1,2,3};     or
myPoly = {poly: "x^2 + 2x + 3"};
```

Either of these declarations generate the polynomial $x^2 + 2x + 3$. To declare a complex polynomial, insert any complex scalar expression in the polynomial initialization syntax, as in the following example.

```
pz1 = {poly: (1,2), 1 + 2*<i>};
pz2 = {poly: "(1,2)x + (1,2)"};
```

You can operate on a polynomial as you would operate on other data types, as in the following example.

```
p2 = myPoly + myPoly;
p3 = p2 * myPoly;
```

To evaluate a polynomial at a particular value treat the polynomial as you would a single input function, as in the following example.

```
y = p4(3);
```

Furthermore, anywhere that you can use a function in HiQ you can use a polynomial. For example, you can perform numerical integration of a polynomial as follows.

```
p = {poly: 3,2,1};
result = integrate(p, -1, 1);
```

Text Objects

You can set text objects, also called strings, to any combination of ASCII characters.

The following statement sets the variable `text` to the value "HiQ".

```
text = "HiQ";
```

☞ **Note:** ***Unlike statements, a single text value enclosed in quotation marks (“”) cannot extend across multiple lines of HiQ-Script. The following syntax is not valid in HiQ Script because it extends across multiple lines:***

```
text = “HiQ Numerical
Analysis”;
```

You can concatenate (add together) text strings:

```
text1 = "Numerical";
text2 = "Analysis";
text3 = text1 + text2;
```

In the preceding example, `text3` would become "NumericalAnalysis." To add a space between the words, you must explicitly place it there, as follows.

```
text3 = text1 + " " + text2;
```

Several HiQ constants such as `<CR>`, `<LF>`, and `<CRLF>` can be used to insert special characters in a text object. For example, the following statement creates a text object with two lines.

```
text = text1 + <CRLF> + text2;
```

You can compare strings using the standard ASCII sequence and the HiQ-Script comparison operators. Chapter 5, *HiQ-Script Reference* lists and describes all operators.

```
text1 = "abc";
text2 = "def";
text3 = "ABC";
```

In a comparison, `text2` would be greater than `text1` because "`d`" is greater than "`a`." In keeping with the standard ASCII sequence, `text1` would be greater than `text3`.

You can access substrings using the standard HiQ-Script vector notation. (See the *Vector and Matrix Object Types* section earlier in this chapter.)

```
text = "abcdef";
text1 = text[2:4];
text2 = text[3:*];
```

`text1` is set to `bcd` and `text2` is set to `cdef`.

Graphical Objects

Data visualization in HiQ centers around the graph object. You graph data by adding your data directly to the graph or by creating a plot object and adding the plot object to the graph. You can graph your data quickly with a single call to `createGraph` or `addPlot`. These functions embed your data directly into the graph. When you need to place a single plot in multiple graphs, you can create a separate plot object using `createPlot` and add the plot to several graphs using `addPlot`. These functions link a plot object to your graph. If the plot object changes (for example, when you change the data or modify an attribute) the change is reflected in all the graphs where the plot appears.

Graphing Data

You can easily graph data in HiQ using the functions `createGraph` (for a new graph) or `addPlot` (for an existing graph). These functions return a unique plot identification number called a *plot handle* for each data set you add to a graph. The following example script creates a new graph with a new plot and then adds a second plot to the same graph. The first line creates a two-dimensional plot (myPlot1) of the vectors `x` and `y` in a new graph (`myGraph`). The second line adds a two-dimensional plot (`myPlot2`) of the vectors `x` and `z` to the existing graph, `myGraph`.

```
[myGraph, myPlot1] = createGraph(x, y);
myPlot2 = addPlot(myGraph, x, z);
```

The previous script returns the plot handles `myPlot1` and `myPlot2`. You can use these plot handles to access the attributes of the plots. Because embedded plots are not HiQ objects, your script must access embedded plot attributes through the graph object, using the object attribute notation (the “dot” syntax). The following example changes the line color of `myPlot1` to red and the line style of `myPlot2` to dashed.

```
myGraph.plot(myPlot1).line.color = <red>;
myGraph.plot(myPlot2).line.style = <dashline>;
```

You can also change the data of an existing embedded plot using the plot handle and `addPlot`. The following example changes the data plotted in `myPlot1`.

```
myPlot1 = addPlot(myGraph, myPlot1, x, z);
```

Graph Objects

The `createGraph` function creates 2D or 3D graph objects and optionally embeds a new plot in the graph. The following code creates an empty 2D graph, `MyGraph`.

```
MyGraph = createGraph(<graph2D>);
```

You can think of a graph object as a container for any number of plots. A graph has many attributes that you can set. For example, to set the title you enter the following statement.

```
MyGraph.title = “My Graph”;
```

You can also set attributes for the axes on a graph. To do this use `.axis` in front of the axis attribute. For a 2D graph you can set the axis titles as in the following examples.

```
MyGraph.axis.x.title = “x Axis”;
MyGraph.axis.y.title = “y Axis”;
```

You can also place these attributes on the right side of an assignment as in this example:

```
theTitle = MyGraph.title;
```

The object `theTitle` now contains the text of the graph title.

Plot Objects

The `createPlot` function creates a 2D or 3D plot object, given a set of data. The following code creates a 2D line plot of the vectors `x` and `y`.

```
myPlot = createPlot(x, y);
```

Like other objects, plots have many attributes that you can alter using the object attribute notation (the “dot” syntax). See the table of plot attributes in the *Object Attribute Reference* section of *HiQ Online Help* for a complete list. For example, if you want to set the coordinate system and title of a plot you can add the following statements.

```
myPlot.coordinateSystem = <polar>;
myPlot.title = “My Data”;
```

You can learn more about using plots and graphs by reviewing the example Notebooks in the `Examples` folder.

Script Objects

Inside HiQ-Script, a script object can only be initialized, converted to text, or saved to a file. The `import` function can initialize a script object with an assignment statement. The `toText` function can convert a script object to a text object. The `export` function can save a script object to a file.

Function Objects

To create (or initialize) a function in HiQ, you can use a special initialization syntax. Refer to the *Function Initialization Operator* section in Chapter 5, *HiQ-Script Reference*, for more information. The syntax of this operator joins a function name, a parameter list, and a string, as follows.

```
{function_name: parameter_list: body};
```

This operator is useful for generating more generic scripts that are possibly based on a user changing a string, as in the following example:

```
myText = "x^2";
myFunc = {function: x: myText};
myFunc(x);
```

The following script is equivalent to the preceding script. Both create a single-statement function.

```
function myFunc(x)
   return x^2;
end function;
myFunc(x);
```

The utility of this function becomes more apparent when the input string is a text object containing user input.

As in the example above (`myText = "x^2";`), a text object used to create a single-statement function must not contain a semicolon (`;`). A text object used to create a multiple-statement function must contain semicolons at the end of each statement and a return statement. In the following script, `myText1` is a valid text object that you can use to create a multiple-statement function, `myFunc1`.

```
myText1 = "a = 1; b = 2; return a + b + x;";
myFunc1 = {function: x: myText1};
```

The following script, `myText2`, is also a valid text object.

```
myText2 = "a = 1;" + <CRLF> + "b = 2;" +
<CRLF> + "return a + b + x;";
myFunc2 = {function: x: myText2};
```

The resulting functions `myFunc1` and `myFunc2` are identical.

You can exploit many interesting features by using this syntax. For example, to create a new object with a name that you generate from a script, write a function such as the following.

```
function copyObject(object, name)
   renameScript = "project " + name + " = x;";
   f = {func: x: renameScript};
   f(object);
end function;
```

You can also write a script to generate and dynamically name several objects. The following example generates 20 scalar objects named x1, x2, …, x20.

```
for i = 1 to 20 do
   name = "x" + toText(i);
copyObject(i, name);
end for;
```

Color

Colors are used mainly for graph and plot attributes. To create a color, use the color initialization syntax and specify the amount of red, green, and blue. This creates a 24-bit RGB color. For example, you can create a medium red with the following settings.

```
redColor = {color: 200,0,0};
```

A medium green and blue would require the following settings.

```
greenColor = {color: 0, 200, 0};
blueColor = {color: 0, 0, 200};
```

The valid range of colors is from 0 to 255. Values outside this range are automatically rounded up or down to the range boundary.

☞ **Note:** ***If you are running in 256-color mode, the color you request is matched to the nearest available color.***

Font

You use a font object mainly to set the font attributes of graphs and plots. To create a font, use the font initialization syntax, specifying the font name as a string, and the font size as an integer.

```
Courier10 = {font: “courier”, 10};
```

Object Attributes

You can use the attribute operator in HiQ-Script to access most attributes of objects. The attribute operator is a period (`.`) followed by the attribute name. For example, if you want to find the type of an object, use the `.type` attribute:

```
if a.type = <integer> then
   doSomething();
else
   doSomethingElse();
end if;
```

To get the number of rows in a matrix, use the `.rows` attribute:

```
for i = 1 to A.rows do
end for;
```

Refer to the *Object Attribute Reference* section in *HiQ Online Help* for a complete list.

Expressions

Expressions are an integral part of statements. Every statement uses expressions of one form or another. An expression consists of a combination of objects, constants, and operators. There are two major types of expressions: algebraic and logical.

`a + b/c` is an example of an algebraic expression.

`a < b` is an example of a logical expression.

Operator Precedence

Operators in expressions have a defined precedence. When precedence is equal, evaluation is from left to right. To override the default precedence, use parentheses, ( ), as in the following example.

```
(a + b)*c
```

In the preceding expression, the addition takes place first. Without the parentheses the multiplication takes place first because it has a higher precedence than addition. For more information, see the *Precedence Operator* section in Chapter 5, *HiQ-Script Reference*.

Function Calls

There are two types of function calls: built-in functions and user-defined functions. You use the syntax shown in the following example to call either type of function.

```
a = sin(x);
```

`sin` is a HiQ built-in function.

```
function f(x)
   return x^2;
end function;
a = f(x);
```

`f` is a user-defined function which returns x^2 (equivalent to `x^2` in a script) as the result.

Algebraic Expressions

Algebraic expressions are used in most places that take expressions. An algebraic expression is any expression that uses any of the algebraic operators.

Algebraic Operators

Algebraic operators are symbols like `+`, `-`, `*`, `/` and `^`. For a complete list refer to the *Expression Syntax Reference* section, in Chapter 5, *HiQ-Script Reference*. Any of the algebraic operators or precedence operators may be used in an algebraic expression.

Matrix and Vector Algebra

Arithmetic operators for matrices include `+`, `-`, `*`, and `/`. Matrices that you add or subtract must have the same dimensions.

```
C = A + B;
D = A - B;
```

Remember that matrix **C** in the preceding example is the element-by-element summation of matrices **A** and **B**; Matrix **D** is the element-by-element difference of matrices **A** and **B**.

Matrix multiplication follows the standard rules: if **A** is an m × n matrix, and **B** is an n × p matrix, then **C** has the dimension m × p.

```
C = A * B;
```

Multiplying a matrix by a scalar results in the multiplication of each element by the scalar.

```
D = A*s; // where s is a scalar
```

The division operator (/) denotes multiplication by an inverse:

```
F = A/B;
```

The following expression performs the same operation as the preceding expression.

```
F = A * inv(B);
```

If the divisor is a scalar, then the resulting matrix consists of each element of the matrix divided by the scalar.

Logical Expressions

Logical expressions are used by flow control statements. A logical expression results in a scalar value of 0 (false) or nonzero (true). Logical expressions include valid algebraic expressions that result in a scalar with the use of any logical operator.

Logical Operators

There are both binary and unary logical operators. Some examples of logical operators are `not` (unary), `and`, `>`, `<`, `>=`, `<=`, `=`. For a complete list of this type of operator refer to the *Expression Syntax Reference* section in Chapter 5, *HiQ-Script Reference*. The *Precedence Operator* section in Chapter 5, *HiQ-Script Reference* explains all precedence rules for operators and how to control the precedence of operations.

Statements

HiQ-Script is a statement-oriented language. Every statement must end with a semicolon (`;`). For example, the following line of code is a complete statement.

```
a = 1;
```

As with most modern languages, HiQ-Script is essentially free of line-oriented restrictions. Because of the language's combination of keywords and the semicolon statement terminator, statements may start

anywhere on a line and span multiple lines. Furthermore, you can insert as many spaces as you want between words and leave empty lines within a statement. One type of statement, called a compound statement, can itself contain statements. In the following example, `a = 1` is a statement within a compound `if` statement.

```
if a = 0 then
   a = 1;
end if;
```

A compound statement can be used anywhere a statement can be used.

In addition to statements, HiQ has function definitions. Unlike statements, function definitions cannot be nested inside a compound statement or inside another function definition.

☞ **Note:** ***You can add line breaks to statements and function definitions. However, line breaks are not valid in comments you place in a script. Each line of a comment must begin with the comment operator, which is a double slash mark (//).***

Keywords

HiQ-Script has a number of keywords such as `if`, `then`, and `for`. These words are reserved and cannot be used as variable or function names anywhere in the language.

There are also a number of compound keywords such as `end if` and `end for`. The space in the keyword is optional so `endif` and `endfor` are acceptable alternatives for `end if` and `end for`. However, compound keywords *must* appear together on the same line.

The following is a complete list of all the keywords used in HiQ-Script.

`all`	`exitif`	`nextrepeat`
`and`	`exitrepeat`	`nextrepeatiteration`
`assume`	`exitselect`	`nextwhile`
`block`	`exitwhile`	`nextwhileiteration`
`blocks`	`false`	`not`
`case`	`for`	`or`
`default`	`forever`	`project`
`do`	`foriteration`	`repeat`
`else`	`fors`	`repeatiteration`
`elseif`	`from`	`repeats`
`end`	`function`	`return`
`endfor`	`if`	`select`
`endfunction`	`ifs`	`selects`
`endif`	`iteration`	`step`
`endrepeat`	`local`	`then`
`endselect`	`mod`	`to`
`endwhile`	`next`	`true`
`exit`	`nextcase`	`when`
`exitall`	`nextfor`	`while`
`exitblock`	`nextforiteration`	`whileiteration`
`exitfor`	`nextiteration`	`whiles`

Declaration Statements

Declaration statements are used to define the scope of objects. Declaration statements are not used to declare the type of an object. Variables can have local or project scope. Local variables have effect only while their function runs. Refer to the *assume*, *project*, and *local* statement descriptions in the *Statement Syntax Reference* section of Chapter 5, *HiQ-Script Reference*, for more information.

Assignment Statements

Use assignment statements to transfer the result of an expression to one or more variables.

Simple Assignment

A simple assignment consists of a variable and an expression where the result of the expression is copied into the variable. In the following example, the result of evaluating the expression `2*3` is assigned to the variable `a`.

```
a = 2 * 3;
```

Any valid algebraic expression may appear on the right side of an assignment. The left side may be a variable, a variable with a subscript operator, or a variable with an attribute operator.

Multiple Assignment

A multiple assignment statement is used to retrieve data from a built-in function that returns more than one item. The right side of the assignment can only be a single function call. The left side is a comma-delimited list of variables to hold the results surrounded by brackets, `[]`, as in the following example.

```
[E,V] = eigen(A);
```

Iteration Statements

Three looping constructs are available:

- `for  ... end for;`
- `while ... do ... end while;`
- `repeat ... end repeat when ;`

For

The most powerful and probably most used looping construct is the for statement. The for statement takes the following general form.

```
for identifier = beginning to ending step step_size do
    statements
end for;
```

The expressions defining the beginning, ending, and step conditions (`beginning`, `ending`, and `step_size`) may be any valid integer or real HiQ-Script expression. The `ending` and `step_size` expressions are evaluated one time at the beginning of the loop.

Loops may proceed in a negative direction by assigning start, end, and step values appropriately. To run the loop from `i = 10` to `i = 5`, the for statement would take the following form.

```
for i = 10 to 5 step -1 do
   statements
end for;
```

You cannot modify the value of the loop index variable inside the loop.

The for loop in HiQ-Script is optimized for speed. An equivalent while or repeat loop can be significantly slower depending on the number of statements in the loop.

See the *Block Escape Statements* section later in this chapter for methods of breaking out of a for loop.

While

The while statement takes the following general form.

```
while logical_expression do
   statements
end while;
```

In the preceding statement, *`logical_expression`* is any valid HiQ-Script logical expression. A simple example is as follows:

```
while i < 5 do
     x = sin(theta);
     i = i + 1;
end while;
```

Notice that `while` checks the logical expression before executing the statements within the loop. If the logical expression tests false initially, the statements inside the loop are never executed.

See the *Block Escape Statements* section later in this chapter for methods of breaking out of the while loop.

Repeat

A repeat loop works similarly to the while construct, except that it checks the validity of the logical expression *after* executing the statements within the loop. A repeat statement always executes at least once and requires the following general form.

```
repeat
    statements
end repeat when logical_expression;
```

In the preceding statement, `logical_expression` is any valid HiQ-Script logical expression. The following item is a simple example.

```
repeat
    x = sin(theta);
    i = i + 1;
end repeat when i > 5;
```

In addition, you can use an alternative construct:

```
repeat forever
    statements
end repeat;
```

There must be an `exit` or `return` statement inside a `repeat forever` loop.

The *Block Escape Statements* section, found later in the chapter, explains methods of breaking out of a repeat loop.

Flow Control Statements

The `if` and `select` statements help you control flow in your HiQ script.

If

The if statement takes the following general form. As shown in the following examples, you must terminate any `if` statement with `end if`.

```
if logical_expression then
    statements
else if logical_expression then
    statements
else
    statements
end if;
```

Notice the syntax of the following simple example:

```
if i < 3 then
   a = sin(x);
end if;
```

Notice the syntax of this more complex example:

```
if i < 3 then
   a = sin(x);
   b = cos(y);
else if i > 5 then
   a = cos(x);
   b = sin(y);
else
   a = 0;
   b = 0;
end if;
```

Select

Use the select statement to select a value from a list of values and to execute a block of code associated with that value. The path taken depends upon the current value of the select expression. A select statement takes the following general form.

```
select algebraic_expression from
   case algebraic_expression :
      statements
   case algebraic_expression :
      statements
         .
         .
         .
   default:
      statements
end select;
```

Notice the syntax of the following simple example:

```
select i from
   case 1:
      z = sin(theta);
   case 2:
      z = cos(theta);
```

```
    case 3:
        z = sin(theta)*cos(theta);
    default:
        z = 0;
end select;
```

Depending on the current value of `i`, one of the case blocks executes, and then control transfers to the statement following `end select`. If `i = 2`, `z` is calculated as `cos(theta)`. If `i` is not 1, 2, or 3, the default block executes, and `z` is set to zero.

Block Escape Statements

You use the next and exit statements to break from a loop or select statement. You use a return statement to escape from a user-defined function.

Next

You use a next statement to move to the next iteration of a loop structure. All statements following the next statement are ignored for that iteration. You can use a `next` statement within while, repeat, and for loops. Also, you can use a `next case` statement to skip to the next case in select statements. The next statement takes the following general form.

```
next statement_keyword;
```

In a for loop, you can execute a specified section of code for only certain values by using the following code strucuture.

```
for i = 1 to 1000 do
    if (sin(i/1000) > 0.5) then
        next for;
    end if;
    statement;
    statement;
        .
        .
        .
    statement;
end for;
```

Exit

You use the exit statement to escape entirely from a specified loop. Furthermore, you may escape from more than one loop with a single invocation of the exit statement. The exit statement takes the following general forms.

```
exit statement_keyword;
exit count plural_statement_keyword;
exit all plural_statement_keyword;
```

For example, the following script contains three nested `for` loops with an `exit` statement which breaks to the outer `for` loop.

```
for i = 1 to 100 step 2 do
   for j = 1 to 10 do
      for k = 1 to 20 do
      theta = .005 * j * k * r;
      if theta > 0.5 then
          exit 2 fors; //breaks to the outer for loop
      end if;
          .
          .
          .
      end for;
   end for;
end for;
```

You can break out of all specific types of iteration statements, such as `for` or `repeat` loops with commands like the following.

```
exit all fors;
```

or

```
exit all repeats;
```

You can break out from all types of blocks with statements like the following.

```
exit 2 blocks
```

or

```
exit all blocks
```

Return

The `return` statement causes an exit from a user-defined function and optionally returns the result of the expression. This is useful for defining functions that work like `sin(x)`. The following example code creates a function `f` which works exactly like the built-in function `sin`.

```
function f(x)
   return sin(x);
end function;
```

The return statement does not require an expression. For example, you can use a return statement to exit from the middle of a function that does not have a return value.

```
function f(x)
    if x = 1 then
        x = 2;
        return;
    end if;
    for i = 1 to 20 do
        x = x + 1;
    end for;
end function;
```

In the following example, the `return` statement (which does not include an expression in this case) is not necessary, because it is implied by the `end function` statement.

```
function f(x)
    return;
end function
```

The following example code has the same return behavior as the preceding example.

```
function f(x)
end function;
```

User-Defined Functions

In HiQ-Script you can create user-defined functions. After you compile a script that contains user-defined functions, these functions become objects available for use in any script in the Notebook. For example, if you compile a script `aScript` in which you define two functions, `f` and `g` , three new function objects result: `f`, `g`, and `aScript_Run`.

Remember, unlike other objects, you cannot edit or place function objects on a Notebook page. To modify these objects, you must edit the script that defines them, then recompile the script.

Defining Functions

A function may be defined in the same script in which it is called, or in any other script. There is no limit to the number of functions that can be defined in any particular script.

A function definition takes the following general form.

```
function fname(x,y,z)
   statements
        .
        .
        .
end function;
```

☞ **Note:** ***You may not define a function inside another function.***

A function you define may return only a single value: only HiQ built-in functions can return multiple values. (See *Calling HiQ Built-in Functions* later in this chapter.) If you need to define a function that returns more than one variable, you can achieve this by allowing the function to return values through some of its input parameters. The *Call by Reference* section later in this chapter describes how to do this. It is not necessary for a function to return a value.

☞ **Note:** ***You can add a*** `return` ***statement to the statements in the preceding function definition prototype, but it is not necessary. The*** `return` ***statement exists implicitly at the end of any function call.***

A function call takes the following form.

```
t = fname(a,b,c);
```

Consider the following example. An aerospace engineer calculating aerodynamic loads on an expendable booster during atmospheric flight needs to find dynamic pressure, $\bar{q}$, which is defined as,

$$\bar{q} = \frac{1}{2}\rho V^2,$$

where ρ is local atmospheric density and V is the booster's relative velocity.

The engineer might write the following function.

```
function dynpress (rho,v)
   q = .5 * rho * v^2;
   return q;
end function;
```

In this example, the engineer could have put the defining expression on the `return` line:

```
function dynpress (rho,v)
   return .5 * rho * v^2;
end function;
```

The following script shows what a call to this function would look like.

```
Q = dynpress(density, velocity);
```

Programmatically Defining Functions in HiQ-Script

Using HiQ's initializer syntax, you can define a new function at run-time in HiQ-Script. Function initialization syntax takes the following general form.

```
myFct = {function:parameter_list:body};
```

The following example creates and uses a new function that takes one parameter and returns a value.

```
body = "cos(x)*sinh(x)";
myFct = {function:x:body};
y = myFct(x);
```

The definition of the function `myFct` is not required before running the script as it is in the following example.

```
function myFct(x)
return cos(x)*sinh(y);
end function;
y = myFct(x);
```

This syntax gives you the flexibility of creating new functions using text objects to define the script code. These text objects can then be placed on the notebook page and changed by the user without having to edit a script object. The problem solver Notebook in Chapter 3, *Using a Notebook* uses this syntax to hide the main script from the user.

If the body of the function is more than one line, then you must include a return statement in the body as in the following example.

```
body = "y[1] = -10*x[1] + 10*t;" + <CRLF>;
body = body + "y[2] = -5*x[2] + 5*t;" + <CRLF>;
body = body + "return y;";
myFct = {function:x,t:body};
```

Call by Reference

Unlike functions in some other languages, functions you define in HiQ-Script can modify the parameters passed to them; that is, parameters passed to user functions are called by reference. Consider the following example.

```
x = 2;
y = f(x);
function f(a)
   a = a + 1;
   return a^2;
end function;
```

The variable `x` is set to 2 and passed to the function `f`. The function first increments its parameter by 1, then returns the square of the modified parameter. After execution, `x` has been incremented to 3 and `y` is set to the value returned by the function, 9.

☞ **Note:** ***Unlike most parameters, which are passed by reference, subranges and attributes are passed by value in HiQ-Script.***

Calling HiQ Built-In Functions

Calling a HiQ built-in function uses the same syntax as calling a user-defined function. Refer to the function reference section of *HiQ Online Help* for a complete list of all HiQ built-in functions.

Using Function Name as a Parameter

Many of the HiQ built-in functions accept as a parameter the name of another function. For instance, the function `derivative` has the following general form.

```
derivative(function,x,n,h);
```

The parameter *function* must specify the name of a HiQ built-in function or the name of a function that a user has defined. If you want to pass the name of a user-defined function as a parameter, you must have a compiled script containing the function definition. The function name is recognized if it appears in the Object list.

Using the example of the `derivative` function, assume you have previously compiled a script containing the following function definition.

```
function myfunc(x)
   return 3*x^3 - 2*x^2;
end function;
```

It is now possible to write a script calling the function `derivative`, passing `myfunc` as a parameter:

```
y = derivative(myfunc, 4.0, 2, 0.01);
```

If you use the preceding line of script within a function, you must declare the parameter `myfunc` as `project`, as in the following example. This completed example, building on the previous two, shows the correct code for referring to a user-defined function within another user-defined function:

```
function calls_myfunc(y)
   project myfunc;
   return derivative(myfunc, y, 2, 0.01);
end function;
```

If you omit the `project` statement, the call to function `myfunc` causes a run-time error, stating that the parameters to function `derivative` are not of the appropriate type.

Performance Issues

HiQ-Script may not run as fast as code compiled to machine language by a normal compiler.

Consider the factors that can decrease the efficiency of a HiQ script. Improper use of looping constructs is the most likely cause for slow scripts. The following paragraphs describe how to avoid this and other inefficiencies in your scripts. Another factor that can slow your scripts is HiQ's capability of dynamically assigning data types in response to the operations your are performing. This feature makes HiQ code easy to use, but slows performance because the compiler generates a set of instructions that require an additional level of interpretation.

You can take various steps to make your scripts perform better. The following example illustrates the way most users would first attempt to take the `sin` of every element of a vector.

```
for i = 1 to x.rows do
   y[i] = sin(x[i]);
end for;
```

For small dimensions, this loop runs so quickly that you would not notice any decrease in speed. However, for larger dimensions the slow execution rate is quite noticeable. The following line of code performs the same work much faster.

```
y = eval(sin, x);
```

The preceding line of code illustrates the concept of vectorization. While vectorization is more often seen in the world of super-computers, it is also an important tool in HiQ to add speed to your scripts.

When you write scripts, it is best to use vectors or matrices when appropriate. The following example represents an opportunity to eliminate a for loop by using the vector math capability of HiQ-Script.

```
m = 1.;
b = 10.;
for i = 1 to x.rows do
   y[i] = m*x[i] + b;
end for;
```

You can replace the preceding `for` loop with the following code.

```
m = 1.;
b = 10.;
y = m*x + b;
```

This construct runs much faster than the earlier loop.

You should try to replace loops in HiQ-Script with equivalent built-in function calls as much as possible. Pay particular attention to nested loops. Replacing the inner loop with a built-in function can give you radical improvements in execution speed.

Interacting with the User

Several functions let you present message boxes and prompts to your users during a script execution. Avoid having your script present a series of dialog boxes asking for user input to solve a problem. A better practice is to collect user input in objects that are visible on the Notebook page and then to perform some validity checks with appropriate error messages in the script to validate the input.

Prompting for Input

There are four functions that prompt a user for input: `getText`, `getNumeric`, `getFileName`, and `putFileName`.

☞ **Note:** ***Three other display functions,*** `warning`, `message`***, and*** `error`***, prompt a user for responses, but not for text or numeric input. Refer to the*** *Displaying Error Messages and Warnings* ***section later in this chapter to learn about these functions.***

getText and getNumeric are very similar. In fact, getNumeric can be written as the following script.

```
function getNumeric(prompt, defaultInput, title)
   s = getText(prompt, defaultInput, title);
   return toNumeric(s);
end function;
```

The following example prompts the user to enter a name and then asks for his or her age.

```
name = getText("Please enter your name", "");
age = getNumeric("How old are you " + name + "?", "");
```

Another important area of user interaction is asking for the name of a file. The getFileName and putFileName functions make this possible. These functions are similar but provide two distinct capabilities. You use getFileName to ask the user for a file that already exists on disk. You use putFileName to ask the user to specify the name of a file to be created. Neither of these functions actually creates or opens a file. You need to use them in conjunction with the open function to get the best use of them. You can use the following script to copy a file.

```
fileIn = getFileName("", "All files (*.*)|*.*|", 1,
"Choose A File");
if fileIn <> "" then
   fileOut = putFileName("", fileIn+" copy",
   "All files (*.*)|*.*|", 1, "Save Copy As");
   inID = open(fileIn, "rb");
   data = import(inID, "::%string");
   outID = open(fileOut, "wb");
   export(outID, data, "");
   close(inID);
   close(outID);
end if;
```

Displaying Error Messages and Warnings

While it is important not to create too many error messages, you should keep the needs of your users foremost. Sometimes a script cannot generate a correct solution without user input. Your error messages and warnings can help users stay on track.

Consider a case where a user gives erroneous input for a function in a script, and you want to inform the user of the error.

```
function invSqrt(x)
  if x <= 0 then
    error("Input must be greater than zero in invSqrt.");
  else
    return 1/sqrt(x);
  end if;
end function;
```

If the input is less than or equal to zero, the built-in function `error` is called, which displays a message and terminates the function.

Sometimes you may want to display an informational message without terminating the script. This is particularly useful for debugging scripts. Consider the following script which displays the current time.

```
message("The time is " + time());
```

Sometimes it is useful to allow the user to decide what to do. The following script looks for a particular file. The loop exits when it finds the file. Every 30 seconds, the script prompts the user to ask if he or she wants to continue waiting:

```
start = timer();
last = start;
repeat forever
    fileID = open("c:\temp\key.dat", "r");
    if fileID <> -1 then
        exit repeat;
    end if;
    now = timer();
    if(now - last) > 30 then
        if warning("Continue waiting?") = 0 then
            exit repeat;
        end if;
        last = now;
    end if;
    wait(1);
end repeat;
```

Formatting Numbers

Under certain conditions you may want to format a number into a string for use in a dialog box or other situation. The built-in function `toText` performs these conversions. The `toText` function takes two parameters. The first parameter is the number to format and the second

parameter is an optional format string. The syntax of the optional format string is identical to that used for the built-in functions `import` and `export`.

The following examples show `toText` in action.

Formatting an integer. (`myIntString` will be “10”.)

```
myIntString = toText(10);
```

Formatting a real. (`myRealString` will be “1.2”.)

```
myRealString = toText(1.2);
```

Formatting a real specifying precision. (`myRealString` will be “1.200”.)

```
 myRealString = toText(1.2, ":%[p3]:");
```

Formatting specifying scientific format. (`myRealString` will be “1.200e0”.)

```
myRealString = toText(1.2, ":%e[p3]:");
```

Formatting a complex number. (`myComplexString` will be “1.2-3.4i”.)

```
myComplexString = toText((1.2,-3.4), ":%cs:");
```

Formatting a complex number using scientific format. (`myComplexString` will be “1.20e00+3.13e03i”.)

```
myComplexString = toText((1.2,3125),
    ":%cs[%e[p2]]:");
```

HiQ-Script Reference

Chapter 5

This chapter contains descriptions of HiQ-Script expressions and statements.

Expression Syntax Reference

This section shows you proper syntax for expressions in HiQ-Script. Remember that an *expression* consists of mathematical operators and operands.

Precedence Operator

Purpose

Change the default precedence of an expression.

Syntax

`(`*`expression`*`)`

Syntax Descriptions

Name	Description
expression	Any valid expression placed within the parenthesis. Can be a logical or algebraic expression depending on context.

Comments

The result of this operation is an expression. HiQ evaluates expression(s) within the parentheses, `()`, first, then evaluates the operations surrounding the parentheses.

The following precedence rules apply when you do not use the precedence operator (a parenthesis).

Operator Precedence Higher to Lower	Direction of Evaluation
.	Left to right.
[]	Left to right.
' + – (unary) not	Right to left.
^ **	Left to right.
* / \ %	Left to right.
+ – (binary)	Left to right.
< <= > >= != <> = ==	Left to right.
and &&	Left to right.
or \\	Left to right.

Attribute Operator

Purpose

Access an attribute of a variable.

Syntax

variable.operator

Syntax Descriptions

Name	Description
variable	Name of variable to access.
operator	Name of attribute to access.

Comments

Refer to the *Object Attribute Reference* section in *HiQ Online Help* for a list of attributes for each object type.

Binary Algebraic Operators

Purpose

Combines two objects into one, using an algebraic rule.

Syntax

```
left_operand  operator  right_operand
left_operand .operator  right_operand
```

Syntax Descriptions

Name	Description
left_operand	Any valid algebraic expression.
operator	Operator from the following list. addition + subtraction – multiplication * division / left division \ exponentiation ^ or ** remainder %
right_operand	Any valid algebraic expression.

Comments

Not all operators are valid for all object types, as indicated in the following table.

Object Type	+	–	*	/	\	^	%
Integer Scalar	x	x	x	x	x	x	x
Real Scalar	x	x	x	x	x	x	x
Complex Scalar	x	x	x	x	x	x	
Integer Vector	x	x	x**				
Real Vector	x	x	x**				
Complex Vector	x	x	x**				
Integer Matrix	x	x*	x		x*		
Real Matrix	x	x	x		x		
Complex Matrix	x	x	x		x		
Polynomial	x	x	x		x	x	x
Text	x						

* Promoted to a real matrix before the operation is performed.
** Operation valid only when used in conjunction with the transpose (`'`) operator.

If the operation is not valid for the data type, HiQ promotes less complex data types to data types that allow the operation. If no data type is valid, HiQ terminates the script and displays an error message. When you combine vectors and matrices, the dimensions must be valid.

If preceded by a single period (`.`), the operators perform element-wise operations on vectors and matrices.

Remember the following points about matrix-vector operations.

- For matrix division, the matrices must be square. `A/B` is equivalent to `A * inv(B)`. `A\B` is `inv(A) * B`.
- Matrix-vector left division `A\b` is equivalent to `solve(A, b)`.
- Vector-matrix division `b/A` is equivalent to `b * inv(A)`.

Table 5-1. Valid Operations between Different Numeric Object Types

Left Operand	Right Operand									
	Int Scalar	Real Scalar	Cmplx Scalar	Int Vector	Real Vector	Cmplx Vector	Int Matrix	Real Matrix	Cmplx Matrix	Poly-nomial
Integer Scalar	+ – * / \ ^ %	^		+ - *			+ - *			
Real Scalar		+ – * / \ ^ %	*		+ - *	+ - *		+ - *	+ - *	+ - * /
Complex Scalar		* /	+ – * / \ ^			+ - *			+ - *	+ - * /
Integer Vector	+ - *						*			
Real Vector		+ – * /			+ – *			* /		
Complex Vector		+ – * /	+ – * /			+ – *		*	* /	
Integer Matrix	+ - * ^			*			+ *			
Real Matrix	^	+ - * /			* \	*		+ – * /	* /	
Complex Matrix	^	+ - * /	+ - * /			* \		* /	+ – * /	
Poly-nomial	^	+ - * /	+ - * /							+ - * / %

Binary Logical Operators

Purpose

Logically compares two objects using a specified rule and returns the result of the comparison as `1` for true and `0` for false.

Syntax

left_operand operator right_operand

Syntax Descriptions

Name	Description
left_operand	Any valid logical expression.
operator	Operator from the following list. logical and — and (or &&) logical or — or (or ||) less than — < greater than — > equal — = (or ==) not equal — != (or <>) less than or equal — <= greater than or equal — >=
right_operand	Any valid logical expression.

Comments

Not all operators are valid for all object types. The following table shows object types and valid operators.

Object Type	and &&	or ||	<	>	= ==	!= <>	<=	>=
Integer Scalar	x	x	x	x	x	x	x	x
Real Scalar	x	x	x	x	x	x	x	x
Complex Scalar					x	x		
Integer Vector					x	x		
Real Vector					x	x		
Complex Vector					x	x		
Integer Matrix					x	x		

Object Type	and &&	or ||	<	>	= ==	!= <>	<=	>=
Real Matrix					x	x		
Complex Matrix					x	x		
Polynomial					x	x		
Text*			x	x	x	x	x	x

* Comparison of text objects is case sensitive.

Left Side Unary Algebraic Operators

Purpose

Modifies the expression located to the right of the operator.

Syntax

operator algebraic_expression

Syntax Descriptions

Name	Description
operator	Operator from the following list. unary minus – unary plus + The unary plus essentially has no effect. You can use this operator to create the expression `x = +1`, which is the same as the expression `x = 1`.
algebraic_expression	Any valid algebraic expression.

Comments

The following table shows object types and valid operators.

Object Type	+	–
Integer Scalar	x	x
Real Scalar	x	x
Complex Scalar	x	x
Integer Vector	x	x
Real Vector	x	x
Complex Vector	x	x
Integer Matrix	x	x
Real Matrix	x	x
Complex Matrix	x	x
Polynomial	x	x

Right Side Unary Algebraic Operators

Purpose

Modifies the expression located to the left of the operator.

Syntax

```
algebraic_expression operator
```

Syntax Descriptions

Name	Description
algebraic_ expression	Any valid algebraic expression.
operator	Operator from the following list. Matrix or vector conjugate transpose ' (an apostrophe)

Comments

Not all operators are valid for all object types. The following table shows object types and valid operators.

Object Type	'
Integer Vector	x
Real Vector	x
Complex Vector	x
Integer Matrix	x
Real Matrix	x
Complex Matrix	x

Unary Logical Operators

Purpose

Inverts the logical expression located to the right of the operator.

Syntax

not logical_expression

Syntax Descriptions

Name	Description
logical_expression	Any valid logical expression.

Comments

Not all operators are valid for all object types. The following table shows valid operators for each object type.

Object Type	not
Integer Scalar	x

Subscript Operator

Purpose

Defines a subscript range. On the left-hand side of an assignment this is the range into which the right-hand expression is to be placed. When on the right-hand side of an assignment, a sub-array is generated.

Syntax

vector[*subscript_expression*]
matrix[*subscript_expression1, subscript_expression2*]

Syntax Descriptions

Name	Description
vector or *matrix*	Name of variable to be subscripted.
subscript_expression	A subscript expression. See *Comments* for a complete description.

Comments

This section describes your options in forming subscript expressions that access elements in a vector or matrix.

You can use a subscript expression to refer to a single element or to a range of elements in an index. To refer to a single element of an index use a single algebraic expression as in the following examples.

- `vector[i];`
- `vector[2*i+3];`
- `matrix[i,j];`
- `matrix[2*i,3*j];`

To refer to a range of elements in an index, use two optional algebraic expressions separated by a colon (`:`) as in the following general example.

```
start_expression:end_expression
```

The `start_expression` and `end_expression` are optional and you can replace them with an asterisk (`*`). When you omit the expressions or replace them with an asterisk, `start_expression` takes the value of 'first element' by default and `end_expression` takes the value of 'last element' by default.

The following four examples show four different ways to refer to the entire range of a row index.

- `matrix[*:*,2]`
- `matrix[:*,2]`
- `matrix[*:,2]`
- `matrix[:,2]`

Note: ***You can use the wild-card character (*) for all references in any subscript expression. When you do so, you select the entire range of elements for that index.***

See Also

Simple Assignment, Multiple Assignment

Vector Initialization Operator

Purpose

Creates a vector from a set of expressions.

Syntax

`{vector: list}`

Syntax Descriptions

Name	Description
list	A comma-delimited list of algebraic expressions. Vectors and matrices with either a single row or column are allowed.

Comments

Your statement must include `vector:`. If you omit that string, HiQ assumes that you are creating a matrix. You can abbreviate the word vector with `v`.

See Also

Matrix Initialization Operator, Polynomial Initialization Operator, Color Initialization Operator, Function Initialization Operator, Font Initialization Operator.

Matrix Initialization Operator

Purpose

Creates a matrix from a set of expressions.

Syntax

{*row_list_1; row_list_2; … row_list_n*}
{matrix:*row_list_1; row_list_2; … row_list_n*}

Syntax Descriptions

Name	Description
row_list	Comma-delimited list of algebraic expressions. Vector or matrix expressions are allowed. Each row provided must have the same number of columns.

Comments

You can omit `matrix:` from the syntax. When you omit that string, HiQ assumes you are creating a matrix, even if the expressions form only one row or column.

See Also

Vector Initialization Operator, Polynomial Initialization Operator, Color Initialization Operator, Function Initialization Operator, Font Initialization Operator.

Polynomial Initialization Operator

Purpose

Creates a polynomial from a set of expressions.

Syntax

`{polynomial: list}`

Syntax Descriptions

Name	Description
list	**List option 1**—A comma-delimited list of algebraic expressions. Vectors and matrices of single rows or columns are allowed. **List option 2**—A string of the form shown in the following example. `"x^3 + 3x^2 + 3"`

Comments

Your statement must include `polynomial:`. If you omit that string, HiQ assumes that you are creating a matrix. You can abbreviate the word polynomial with `p`.

In the first option, coefficients are taken from the list highest order first, for example, `{p:1,0,2}` is the polynomial $x^2 + 2$.

See Also

Matrix Initialization Operator, *Vector Initialization Operator*, *Color Initialization Operator*, *Function Initialization Operator*, *Font Initialization Operator*.

Color Initialization Operator

Purpose

Creates a color from a set of expressions.

Syntax

`{color: red, green, blue}`

Syntax Descriptions

Name	Description
red	Integer value of the red component of an RGB color.
green	Integer value of the green component of an RGB color.
blue	Integer value of the blue component of an RGB color.

Comments

Your statement must include `color:`. If you omit that string, HiQ assumes that you are creating a matrix. You can abbreviate the word color with `c`.

The values provided should be within the range [0,255]. Values outside the range are legal, but are constrained to the range.

See Also

Matrix Initialization Operator, Vector Initialization Operator, Polynomial Initialization Operator, Function Initialization Operator, Font Initialization Operator.

Function Initialization Operator

Purpose

Creates a function from a set of expressions.

Syntax

`{function: arglist: body}`

Syntax Descriptions

Name	Description
arglist	A comma-delimited list of arguments to be passed into the generated function. This parameter is optional; if omitted, you must also omit the colon ":" that follows.
body	A string containing a valid expression which doesn't contain a semi-colon ";" or a complete function body. In the first case, the expression is placed into a function as follows. **function** *name*(*arglist*) **assume project**; **return** *body*; **end function**; In the second case the function is generated as: **function** *name*(*arglist*) **assume** project; *body* **end function**;

Comments

Your statement must include `function:`. If you omit that string, HiQ assumes you are creating a matrix. You can abbreviate the word function with `f`.

See Also

Matrix Initialization Operator, Vector Initialization Operator, Polynomial Initialization Operator, Color Initialization Operator, Font Initialization Operator.

Font Initialization Operator

Purpose

Creates a font from a set of expressions.

Syntax

{font: *name*, *size*}

Syntax Descriptions

Name	Description
name	A text expression containing the font name.
size	An integer expression containing the point size of the font.

Comments

Your statement must include `font:`. If you omit that string, HiQ assumes you are creating a matrix. You can abbreviate the word font with `fo`.

See Also

Matrix Initialization Operator, Vector Initialization Operator, Polynomial Initialization Operator, Color Initialization Operator, Function Initialization Operator.

Statement Syntax Reference

This section shows you proper syntax for statements in HiQ. Remember, a semicolon (`;`) must follow all statements in HiQ.

assume

Purpose

Sets the scope for variables.

Syntax

`assume` *`scope_keyword`*`;`

Syntax Descriptions

Name	Description
scope_keyword	`project` or `local`.

Comments

Without a contravening `assume` statement, the default scope for variables is `project` for external statements. Within functions, the default scope for variables is `local`.

See Also

project, *local*, and *Scope of Variables* section in Chapter 4, *Understanding HiQ-Script*

exit

Purpose

Terminates processing of a block of statements. Execution begins at the statement following the block you exit.

Syntax

```
exit statement_keyword;
exit count plural_statement_keyword;
exit all plural_statement_keyword;
```

Syntax Descriptions

Name	Description

statement_keyword	A keyword from the following list.
	`if`
	`for`
	`while`
	`repeat`
	`select`
	`block`

count	Number of blocks to exit.

plural_statement_keyword	A keyword from the following list.
	`ifs`
	`fors`
	`whiles`
	`repeats`
	`selects`
	`blocks`

Comments

The `block` keyword refers to any statement block.

for

Purpose

Repeatedly executes a block of statements. A counter variable updates at each iteration.

Syntax

```
for counter = start to finish do
statements
end for;

for counter = start to finish step step_size do
statements
end for;
```

Syntax Descriptions

Name	Description
counter	Name of variable to modify on each iteration.
start	Expression that evaluates to an integer or real scalar. *counter* is set to the result of the expression.
finish	Expression that evaluates to an integer or real scalar. *counter* is incremented by 1 or *step_size*, if specified, until it reaches the result of the expression.
step_size	Expression that evaluates to an integer or real scalar. *counter* is incremented by the result of the expression on each iteration.
statements	Zero or more statements.

Comments

The algorithm ensures that round-off error does not propagate while HiQ performs the iteration. The counter in a for loop cannot be changed by the statements inside the for-loop block. In particular, you must avoid passing the loop counter as a parameter in a function call to a function that modifies this value (See *Call by Reference*.).

See Also

while, repeat

function

Purpose

Defines a function.

Syntax

```
function name(argument_list)
statements
end function;
```

Syntax Descriptions

Name	Description
name	Name of the function.
argument_list	Comma-delimited list of variable names. Inside the function these arguments are aliases for the variables or constants passed in from the calling routine.
statements	One or more valid statements.

Comments

Function blocks are not statements; that is, they cannot appear inside any statement block. They can only appear in the scope of external statements. The position in which a function call appears in relation to its definition is not important. It may appear before, after, or within another script.

See Also

return, *Function Initialization Operator*

Function Call

Purpose

Calls a built-in or user-defined function.

Syntax

```
function_name(argument_list);
variable = function_name(argument_list);
[variable_list] = function_name(argument_list);
```

Syntax Descriptions

Name	Description
function_name	Name of built-in or user-defined function. Built-in function names are not case sensitive. User-defined function names are case sensitive.
argument_list	List of values or variable names to pass to the function.

variable_list	Comma-delimited list of variable names. You may omit variable names. If you want only the first value, then you may omit the brackets, `[]`. If you want to omit a variable at the beginning or middle of the list, you may leave out the name but keep the commas as in the following samples, the first of which omits `b` and the second of which omits `a`.
	`[a,,c] = …`
	`[,b,c] = …`
	Multiple return value syntax is allowed only for calls to HiQ built-in functions.

See Also

function

if

Purpose

Executes a block of statements only when a condition is true.

Syntax

```
if condition then
statements
end if;

if condition then
statements
else
statements
end if;

if condition then
statements
else if condition then
statements
       .
       .
       .
else
statements
end if;
```

Syntax Descriptions

Name	Description
condition	Logical expression. The statements in the block are executed only if the expression evaluates to a non-zero value.
statements	Zero or more statements.

Comments

You may specify the `else if` block repeatedly. You can omit the `else` block. HiQ evaluates conditions in the order that they appear.

See Also

while, *select*

local

Purpose

Defines the scope of a list of variables to be `local`.

Syntax

`local` *`variable_list`*;

Syntax Descriptions

Name	Description
`variable_list`	Comma-delimited list of variable names.

Comments

If a variable is not specifically declared to be `local` or `project`, then the variable is `local` if used in a user-defined function block, and is `project` if outside a user-defined function block.

See Also

assume, *project*, and *Scope of Variables* section in Chapter 4, *Understanding HiQ-Script*

next

Purpose

In an iteration statement, causes execution to advance to the beginning of the next iteration. In a select statement, causes execution to advance to the next case.

Syntax

`next` *`statement_keyword`*;

Syntax Descriptions

Name	Description

statement_keyword	A keyword from the following list.
	`case`
	`for`
	`while`
	`repeat`

See Also

select, *for*, *while*, *repeat*

project

Purpose

Defines the scope of a list of variables to be `project`.

Syntax

`project` *`variable_list`*`;`

Syntax Descriptions

Name	Description
variable_list	Comma-delimited list of variable names.

Comments

If a variable is not specifically declared to be `local` or `project`, then the variable is local if used in a user-defined function block, and is project if outside a user-defined function block.

See Also

assume, *local*, and *Scope of Variables* section in Chapter 4, *Understanding HiQ-Script*

repeat

Purpose

Repeatedly executes a block of statements until a condition is true. Evaluation of the condition is done after the statements are executed.

Syntax

```
repeat
statements
end repeat when condition;
```

Syntax Descriptions

Name	Description
condition	Logical expression. Until this expression evaluates to a non-zero result, the block of statements is repeatedly executed.
statements	Zero or more statements.

See Also

for, while, repeat forever

repeat forever

Purpose

Repeatedly executes a block of statements. Exit from this block of statements only occurs as a result of a `return` or `exit` statement.

Syntax

```
repeat forever
      statements
end repeat;
```

Syntax Descriptions

Name	Description
statements	One or more statements.

Comments

The only way to exit this loop is to have an exit or return statement inside it. A compilation error results if there is no `exit` or `return` statement.

See Also

for, *while*, *repeat*

return

Purpose

Exits a function. If an expression is specified, then that expression is returned to the calling function.

Syntax

```
return;
return algebraic_expression;
```

Syntax Descriptions

Name	Description
algebraic_ expression	Any valid algebraic expression. The expression is evaluated before the function returns.

See Also

function

select

Purpose

Selects a group of statements to be executed based on the evaluation of an expression.

Syntax

```
select selector from
case item :
statements
        .
        .
        .
default:
statements
end select;
```

Syntax Descriptions

Name	Description
selector	Algebraic expression. This determines which case is chosen.
item	Algebraic expression. Used in choosing the case. The first case that evaluates to the same result as item is the one chosen.
statements	One or more valid statements.

Comments

The expression *selector* is evaluated then compared, in order, to each *item* expression. When a match is found that block of statements is executed. If no match is found, the default block, if present, is executed. Unlike C and C++, HiQ-Script does not allow control to “fall through” at the end of execution. To force a “fall through” use a `next case` statement.

See Also

next, *exit*

Simple Assignment

Purpose

Evaluates an expression and places the result into a variable.

Syntax

variable = expression;

Syntax Descriptions

Name	Description
variable	Name of a variable or a variable with specified subscript.
expression	Any valid expression.

See Also

Multiple Assignment

while

Purpose

Repeatedly executes a block of statements while a particular condition is true. Evaluation of the condition is done before the statements are executed.

Syntax

```
while condition do
      statements
end while;
```

Syntax Descriptions

Name	Description
`condition`	Logical expression. While this expression evaluates to a non-zero result, the block of statements is repeatedly executed.
`statements`	Zero or more statements.

See Also

for, repeat, repeat forever

Customer Communication

Appendix

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 Fax-on-Demand 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. 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
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: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

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

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate 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.

`support@natinst.com`

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.

	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 527 2321	09 502 2930
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
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	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
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:_____

_____

_____

_____

_____

_____

HiQ Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use 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.

National Instruments Products

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

HiQ, NI-DAQ, LabVIEW, or LabWindows version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

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 User Manual*

Edition Date: October 1996

Part Number: 321063A-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 (___)_____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

A

ActiveX (Microsoft ActiveX)	A programming system and user interface that lets you work with interactive objects. Formerly called OLE.
ActiveX embedded object	An object placed into a container and unconnected to any other object or application. *See also* embedded.
ActiveX linked object	An object placed into a container and connected to another object or application in the same container or in a separate container. *See also* linked.
argument	*See* parameter.
assignment	A script statement that sets a value to a variable.
attribute	*See* property.

B

built-in function	One of many programming utilities in HiQ-Script that perform analysis, graphical, or utility operations.

C

caret	*See* insertion point.
cell	In a matrix or vector, the intersection of a row and column that contains a numerical value.
color object	A HiQ object that sets the color attributes of graphs and plots.
comment	An explanatory line or portion of a line in HiQ-Script.
compiled script	A HiQ object containing HiQ-Script language that has been converted to code that a computer uses to execute the program.

constant — A predefined value in HiQ-Script.

cursor — The pointer or other image that displays on screen to show the location of your mouse, trackball, or other pointing device.

D

data type — *See* numeric type.

debug — To check and correct invalid code in a HiQ script in order to eliminate errors during the compilation or execution of the script.

declaration — A HiQ-Script statement that defines the scope of variables. Can be either project or local.

dialog box — A window containing a message, interactive options, and buttons.

drag and drop — To move an object to a specific location, using the mouse to click on, drag, and release the object. Depending on the location of release, a specific action can occur.

E

embedded — Inserted into a container object and unconnected to any other object or application. Compare this term to linked. *See also* ActiveX embedded object.

error message — An information box that appears when HiQ cannot complete an action due to an internal error, compile error, run-time error, or user error.

expression — A mathematical operator and its operands.

F

font object — A HiQ object type used to set the font attributes of graphs and plots.

function — A block of code that performs a specific task in HiQ-Script; can be a HiQ built-in function or a user-defined function.

function call — Specific HiQ-Script syntax that calls a user or built-in function with given parameters.

G

grab handle — A site on a selected object that you click on and drag to move, size, or reshape the object.

graph — The HiQ object that contains a two-dimensional or three-dimensional collection of plots. *See* plot.

H

handle — *See* grab handle.

HiQ script — A block of programming code that can perform mathematical analysis and display its output textually, graphically, and numerically.

HiQ-Script — An intuitive programming language for mathematics.

HiQ Tools toolbar — Part of the HiQ user interface that contains icons for objects you can place in a Notebook. Depending on your preference, the toolbar can appear on any edge of the interface, or as a floating palette.

I

initialization — In HiQ-Script, a designated syntax which creates a particular type of object, for example, vector, matrix, color, and others.

insertion point — The location where text will be inserted (also referred to as the *caret*).

K

keyword — A reserved word in HiQ Script, such as `if`, `then`, or `while`, that is used for constructing specific types of programming statements.

L

linked — Inserted into a container object and connected with another object or application. Compare this term to embedded. *See also* ActiveX linked object.

loop — A statement in HiQ-Script consisting of keywords and nested statements that performs a repetitive function. Also known as an iteration statement.

M

matrix — A HiQ numeric object containing an array of elements with rows and columns. *See* vector.

message box — A secondary window that appears containing information about the status of a HiQ operation. *See* dialog box.

N

Notebook — The workspace in HiQ that stores, organizes, and displays all the components of an analysis and visualization problem.

numeric object — A HiQ object type that is defined in terms of a numeric type (integer, real, or complex) and an object type (matrix, vector, polynomial, or scalar).

numeric type — One of three types of a numeric object (integer, real, or complex).

O

object — An entity in a HiQ Notebook that contains data of a specific type, for example, numeric, graphic, text, or HiQ-Script. Objects work together in a Notebook to generate and display solutions to analysis and visualization problems. Objects are always stored in a Notebook, but are not always visible on a Notebook page. *See* object view.

Object list — A window in HiQ that displays all the objects in a HiQ Notebook.

object view — A HiQ object that is visible on a Notebook page.

OLE (Microsoft OLE) — Object Linking and Embedding. A programming system and user interface that lets you work with interactive objects. *See* ActiveX.

operand — An object (or objects) modified by an operator.

operator — Code in HiQ-Script that is specific to basic mathematical operations and structure of HiQ-Script language. Often represented as a symbol, for example, "/" represents division.

P

parameter — An independent variable passed to a user-defined or built-in function call in a parameter list.

plot — A HiQ object type that graphically repesents a two-dimensional or three-dimensional function or data set used in conjunction with a graph object. *See* graph.

polynomial — A HiQ numeric object represented by an equation in the polynomial form $ax^n + bx^{n-1}+cx^{n-2} + ...$

pop-up menu — A context-sensitive menu that you can access by right-clicking with your mouse on an object or on the Notebook.

Problem Solver — A HiQ Notebook containing objects, including HiQ-Script, that allows you to interactively perform analysis of data and display results for a broad class of problems. For example, the expression evaluator problem solver can display the results of any expression.

Properties dialog box — A window in HiQ with tabbed pages (property pages) where you can quickly set a wide variety of attributes for a given object.

property — Attributes of a HiQ object. Examples include the color of a plot, the size of a matrix, and the type of any object.

property page — A tabbed subsection of a property dialog box containing a collection of object attributes.

S

scalar — A HiQ numeric object represented as a number. *See* vector and matrix.

scope — In HiQ-Script, a HiQ-object declaration that specifies whether the object is available to the entire Notebook (project) or is temporarily available (local).

script — A block of code that performs a certain task. *See* HiQ script and compiled script.

script object — A HiQ object type containing HiQ-Script and from which you compile and execute your program.

section tabs — An organizing tool in a HiQ Notebook that lets you label and quickly access different parts of your Notebook.

selection handle — A graphical control point of an object that provides direct manipulation support for operations of that object, such as moving, sizing, or scaling.

selection tool — The mouse cursor in HiQ shaped like a standard pointer arrow that lets you select, move, and manipulate objects on the Notebook.

shortcut key — A key or combination of keys that you press to invoke a command.

Standard toolbar — Site on the user interface of HiQ that contains basic utility tools, for example, Save, Open, Cut, Paste, and, Print. Depending on your preference, the toolbar can appear on any edge of the interface, or as a floating window.

statement — In HiQ-Script, a line of code consisting of various keywords, functions, and/or operators that performs a certain task.

status bar — A region, usually the bottom of a window, containing information about HiQ and any selected object.

symbol — The name for *objects* in HiQ for the Macintosh.

T

text object — A HiQ object type where you can enter and edit text.

toolbar — Site on an application interface that contains various buttons and other controls. Depending on your preference, a toolbar can appear on any edge of the interface, or a floating window.

tooltip — A small, descriptive pop-up window that appears when you position the mouse cursor over a toolbar icon.

type — A classification of an object based on its characteristics, behavior, and attributes.

U

user-defined function — A functions that a user creates in HiQ-Script to perform customized analysis, graphical, or utility operations.

V

vector — A HiQ numeric object containing an array with one row or column. Compare with matrix.

Index

Numbers and Symbols

. (period). *See* attribute operator.
2D graphs. *See* graphical objects.
3D graphs. *See* graphical objects.

A

accessing online help, *xi-xii*, 1-8
ActiveX objects, 3-25 to 3-26
    adding object to page, 3-25
    Delete View and Delete Object menu items (figure), 3-26
    deleting, 3-25 to 3-26
        object from notebook, 3-26
        views of objects, 3-26
    native objects vs. ActiveX objects (note), 3-25
    overview, 1-6
algebraic expressions
    algebraic operators, 4-17
    matrix and vector algebra, 4-17 to 4-18
algebraic operators
    left side unary algebraic operators, 5-7 to 5-8
    right side unary algebraic operators, 5-8 to 5-9
assignment statements, 4-20 to 4-21
    multiple assignment, 4-21
    simple assignment, 4-21, 5-27
assume statement, 5-16 to 5-17
attribute operator
    purpose and use, 4-16
    syntax and description, 5-2 to 5-3
automatic data type promotion, 4-5
automatic typing and dimensioning of vector and matrix object types, 4-9 to 4-10

B

binary algebraic operators, 5-3 to 5-5
binary files, importing, 3-4 to 3-5
binary logical operators, 5-5 to 5-7
block escape statements, 4-25 to 4-27
    exit, 4-26, 5-17
    next, 4-25, 5-22 to 5-23
    return, 4-26 to 4-27, 5-25
bulletin board support, A-1

C

call by reference, 4-29 to 4-30
color for Notebook. *See* property pages.
color initialization operator
    purpose and use, 4-15
    syntax and description, 5-13 to 5-14
comments in scripts, 4-3
complex numeric types, 4-5 to 4-6
constants, 4-3
creating Notebooks. *See* Notebook creation.
customer communication, *xiv*, A-1 to A-2

D

data fitting
    example, 2-2 to 2-5
        object list, 2-4 to 2-5
        opening Notebook, 2-2 to 2-3
        running scripts, 2-3 to 2-4
    overview, 1-4

declaration statements, 4-20
Delete View and Delete Object menu items (figure), 3-26
deleting ActiveX objects, 3-25 to 3-26
    object from notebook, 3-26
    views of objects, 3-26
differential equations, 1-5
differentiation capability, 1-6
documentation
    accessing online help, *xi-xii*
    conventions used in HiQ documentation, *xiii-xiv*
    organization of HiQ online help, *xii*
    organization of user manual, *xiii*
    overview of HiQ documentation, *xi*

E

electronic support services, A-1 to A-2
e-mail support, A-2
embedded plots, 4-12
error messages, displaying, 4-33 to 4-34
example notebooks. *See* Notebook use.
exit statement
    purpose and use, 4-26
    syntax and description, 5-17
Export wizard (note), 3-5
expression evaluator problem solver, 2-5 to 2-7
    entering data and running script, 2-6 to 2-7
    illustration, 2-6
expression syntax reference, 5-1 to 5-16
    attribute operator, 5-2 to 5-3
    binary algebraic operators, 5-3 to 5-5
    binary logical operators, 5-5 to 5-7
    color initialization operator, 5-13 to 5-14
    font initialization operator, 5-15 to 5-16
    function initialization operator, 5-14 to 5-15
    left side unary algebraic operators, 5-7 to 5-8
    matrix initialization operator, 5-12
    polynomial initialization operator, 5-13
    precedence operator, 5-1 to 5-2
    right side unary algebraic operators, 5-8 to 5-9
    subscript operator, 5-10 to 5-11
    unary logical operators, 5-9 to 5-10
    vector initialization operator, 5-11 to 5-12
expressions, 4-16 to 4-18
    algebraic expressions, 4-17
    function calls, 4-17
    logical expressions, 4-18
    operator precedence, 4-16 to 4-17
external statements and functions, 4-2

F

fax and telephone support, A-2
FaxBack support, A-2
flow control statements, 4-23 to 4-25
    if, 4-23 to 4-24
    select, 4-24 to 4-25
font initialization operator, 5-15 to 5-16
font objects, 4-16. *See also* property pages.
for statement
    purpose and use, 4-21 to 4-22
    syntax and description, 5-18
formatting numbers, 4-34 to 4-35
FTP support, A-1
function call statement, 5-19 to 5-20
function calls, 4-17
function initialization operator
    purpose and use, 4-14 to 4-15
    syntax and description, 5-14 to 5-15
function statement, 5-19
functions, user-defined. *See* user-defined functions.

G

getting started with HiQ. *See also* Notebook use.
    Getting Started Tutorial, 1-8
    installing HiQ, 1-1 to 1-2

launching HiQ, 1-2
learning HiQ, 1-2 to 1-3
system requirements, 1-1
graphical objects, 4-12 to 4-13
embedded plots, 4-12
graph objects, 4-13
plot objects, 4-13
visualizing rainfall data (2D graph), 3-5 to 3-9
creating a graph, 3-6 to 3-7
modifying graph and plot, 3-8
plotting HiQ data object, 3-7
working with multiple plots, 3-8 to 3-9
visualizing seismic data (3D graph), 3-9 to 3-11
modifying 3D graph and plot, 3-10 to 3-11
rotating and zooming, 3-11
steps, 3-9

H

help. *See* online help.
Help menu, 1-8
highlighting, syntax, 4-3
HiQ
installing, 1-1 to 1-2
launching, 1-2
learning to use, 1-2 to 1-3
system requirements, 1-1
HiQ Notebook. *See* Notebook creation; Notebook use.
HiQ-Script. *See also* objects.
analyzing data with HiQ-Script, 3-11 to 3-15
entering scripts, 3-12 to 3-14
running script and viewing output, 3-14 to 3-15
case sensitivity of function and variable names (note), 4-6
comments, 4-3
connecting script to input objects, 3-19 to 3-21
activating objects (note), 3-19
example of script, 3-20
modifying the script, 3-19 to 3-20
running the script, 3-20
status bar information (note), 3-21
constants, 4-3
expression syntax reference, 5-1 to 5-16
attribute operator, 5-2 to 5-3
binary algebraic operators, 5-3 to 5-5
binary logical operators, 5-5 to 5-7
color initialization operator, 5-13 to 5-14
font initialization operator, 5-15 to 5-16
function initialization operator, 5-14 to 5-15
left side unary algebraic operators, 5-7 to 5-8
matrix initialization operator, 5-12
polynomial initialization operator, 5-13
precedence operator, 5-1 to 5-2
right side unary algebraic operators, 5-8 to 5-9
subscript operator, 5-10 to 5-11
unary logical operators, 5-9 to 5-10
vector initialization operator, 5-11 to 5-12
expressions, 4-16 to 4-18
algebraic expressions, 4-17
function calls, 4-17
logical expressions, 4-18
operator precedence, 4-16 to 4-17
external statements and functions, 4-2
line breaks (note), 4-19
naming conventions, 4-2
overview, 4-1
performance issues, 4-31 to 4-32
scope of variables, 4-3

statement syntax reference, 5-16 to 5-28
assume, 5-16 to 5-17
exit, 5-17
for, 5-18
function, 5-19
function call, 5-19 to 5-20
if, 5-20 to 5-21
local, 5-22
next, 5-22 to 5-23
project, 5-23
repeat, 5-24
repeat forever, 5-24 to 5-25
return, 5-25
select, 5-26
simple assignment, 5-27
while, 5-27 to 5-28
statements, 4-18 to 4-28
assignment statements, 4-20 to 4-21
block escape statements, 4-25 to 4-27
declaration statements, 4-20
flow control statements, 4-23 to 4-25
iteration statements, 4-21 to 4-23
keywords, 4-19 to 4-20
syntax highlighting, 4-3
user interaction, 4-32 to 4-35
displaying error messages and warnings, 4-33 to 4-34
formatting numbers, 4-34 to 4-35
prompting for input, 4-32 to 4-33
user-defined functions, 4-27 to 4-31
call by reference, 4-29 to 4-30
calling built-in functions, 4-30
defining, 4-27 to 4-29
defining programmatically, 4-29
function names used as parameters, 4-30 to 4-31

I

if statement
purpose and use, 4-23 to 4-24
syntax and description, 5-20 to 5-21
Import wizard, 3-1
Import wizard dialog box (note), 3-3
importing data into HiQ, 3-1 to 3-5
binary file as numeric object, 3-4 to 3-5
custom import mode, 3-5
procedure, 3-2 to 3-3
text file as numeric object, 3-4
text file as text object, 3-3 to 3-4
initializing vector and matrix object types, 4-6 to 4-7
input for problem solver, 3-15 to 3-19
prompts for users, 3-18
renaming objects (note), 3-19
scalar object creation, 3-17
selecting one or more objects (note), 3-17
sizing objects (note), 3-17
text object creation, 3-16 to 3-17
installing HiQ, 1-1 to 1-2
integration capability, 1-5
interpolation, 1-4
iteration statements, 4-21 to 4-23
for, 4-21 to 4-22, 5-18
repeat, 4-23, 5-24
while, 4-22, 5-27 to 5-28

K

keywords, 4-19 to 4-20

L

launching HiQ, 1-2
learning to use HiQ
Getting Started Tutorial, 1-8
resources, 1-2 to 1-3
left side unary algebraic operators, 5-7 to 5-8
linear algebra functionality, 1-4
local statement, 5-22
logical expressions, 4-18
logical objects not available (note), 4-4
logical operators, 4-18
binary, 5-5 to 5-7
unary (not), 5-9 to 5-10

M

manual. *See* documentation.
matrix initialization operator, 5-12
matrix object types. *See* vector and matrix object types.
multiple assignment, 4-21

N

naming conventions for objects, 4-2
next statement
purpose and use, 4-25
syntax and description, 5-22 to 5-23
nonlinear analysis capability, 1-5
Notebook creation. *See also* problem solver.
analyzing data with HiQ-Script, 3-11 to 3-15
entering scripts, 3-12 to 3-14
running script and viewing output, 3-14 to 3-15
Export wizard (note), 3-5
importing data into HiQ, 3-1 to 3-5
binary file as numeric object, 3-4 to 3-5
custom import mode, 3-5
Import wizard, 3-1
Import wizard dialog box (note), 3-3
procedure, 3-2 to 3-3
text file as numeric object, 3-4
text file as text object, 3-3 to 3-4
visualizing rainfall data (2D graph), 3-5 to 3-9
creating a graph, 3-6 to 3-7
modifying graph and plot, 3-8
plotting HiQ data object, 3-7
working with multiple plots, 3-8 to 3-9
visualizing seismic data (3D graph), 3-9 to 3-11
modifying 3D graph and plot, 3-10 to 3-11
rotating and zooming 3-11
steps, 3-9
Notebook use. *See also* problem solver.
ActiveX environment, 1-6
data fitting example, 2-2 to 2-5
object list, 2-4 to 2-5
opening Notebook, 2-2 to 2-3
running scripts, 2-3 to 2-4
example notebooks, 1-7
interacting with Notebooks, 1-7
running scripts, 1-7
working with multiple windows, 1-7
expression evaluator problem solver, 2-5 to 2-7
entering data and running script, 2-6 to 2-7
illustration, 2-6
interactive analysis environment, 1-3 to 1-6
data fitting and interpolation, 1-4
differential equations, 1-5
differentiation, 1-6
integration, 1-5
linear algebra functionality, 1-4
nonlinear analysis, 1-5
optimization routines, 1-5
polynomials, 1-6
statistics, 1-5
objects. *See also* objects.
important points, 2-2
interacting with objects, 2-1
organizational tool, 1-3
numbers, formatting, 4-34 to 4-35
numeric objects, 4-5 to 4-10
automatic data type promotion, 4-5
case sensitivity of function and variable names (note), 4-6
characteristics (table), 4-5
complex numeric types, 4-5 to 4-6
importing files
binary files, 3-4 to 3-5
text files, 3-4
polynomials, 4-10
scalar object types, 4-6

vector and matrix object types, 4-6 to 4-10
automatic typing and dimensioning, 4-9 to 4-10
default keyword in initializer syntax (note), 4-7
initializing, 4-6 to 4-7
subscripting, 4-7 to 4-9

O

Object list
illustration, 2-5
objects with local scope (note), 2-5
selecting, 2-4
objects. *See also* ActiveX objects; HiQ-Script.
attribute operator, 4-16
characteristics (table), 4-4
colors, 4-15
definition, 1-3
font objects, 4-16
function objects, 4-14 to 4-15
graphical objects, 4-12 to 4-13
embedded plots, 4-12
graph objects, 4-13
plot objects, 4-13
important points, 2-2
interacting with objects, 2-1
logical objects not available (note), 4-4
naming conventions, 4-2
numeric objects, 4-5 to 4-10
automatic data type promotion, 4-5
case sensitivity of function and variable names (note), 4-6
characteristics (table), 4-5
complex numeric types, 4-5 to 4-6
importing binary files, 3-4 to 3-5
importing text files, 3-4
polynomials, 4-10
scalar object types, 4-6
vector and matrix object types, 4-6 to 4-10
overview, 4-1
script objects, 4-14
text objects
importing text files, 3-3 to 3-4
purpose and use, 4-10 to 4-11
online help
accessing, *xi-xii*, 1-8
organization of HiQ online help, *xii*
operator precedence, 4-16 to 4-17. *See also* precedence operator.
optimization routines, 1-5

P

performance issues, 4-31 to 4-32
period (.). *See* attribute operator.
plot objects, 4-13. *See also* graphical objects.
plots
embedded plots, 4-12
plot handle, 4-12
polynomial initialization operator, 5-13
polynomials
overview, 1-6
using in scripts, 4-10
precedence operator
overriding default precedence, 4-16 to 4-17
precedence rules (table), 5-2
syntax and description, 5-1 to 5-2
problem solver
ActiveX objects, 3-25 to 3-26
adding object to page, 3-25
deleting objects and object views, 3-25 to 3-26
native objects vs. ActiveX objects (note), 3-25
connecting script to input objects, 3-19 to 3-21
activating objects (note), 3-19
example of script, 3-20
modifying the script, 3-19 to 3-20
running the script, 3-20
status bar information (note), 3-21

expression evaluator example, 2-5 to 2-7
entering data and running script, 2-6 to 2-7
illustration, 2-6
finishing touches, 3-21
properties, setting (note), 3-21
property pages, 3-22 to 3-24
accessing Properties dialog box, 3-22
changing properties, 3-23 to 3-24
providing input, 3-15 to 3-19
prompts for users, 3-18
renaming objects (note), 3-19
scalar object creation, 3-17
selecting one or more objects (note), 3-17
sizing objects (note), 3-17
text object creation, 3-16 to 3-17
project statement, 5-23
properties, setting (note), 3-21
property pages, 3-22 to 3-24
accessing Properties dialog box, 3-22
changing properties, 3-23 to 3-24

R

repeat forever statement, 5-24 to 5-25
repeat statement
purpose and use, 4-23
syntax and description, 5-24
return statement
purpose and use, 4-26 to 4-27
syntax and description, 5-25
right side unary algebraic operators, 5-8 to 5-9

S

scalar object types, 4-6
scope of variables, 4-3
script objects, 4-14
scripts. *See* HiQ-Script.
select statement
purpose and use, 4-24 to 4-25
syntax and description, 5-26
simple assignment, 4-21, 5-27
statement syntax reference, 5-16 to 5-28
assume, 5-16 to 5-17
exit, 5-17
for, 5-18
function, 5-19
function call, 5-19 to 5-20
if, 5-20 to 5-21
local, 5-22
next, 5-22 to 5-23
project, 5-23
repeat, 5-24
repeat forever, 5-24 to 5-25
return, 5-25
select, 5-26
simple assignment, 5-27
while, 5-27 to 5-28
statements, 4-18 to 4-28
assignment statements, 4-20 to 4-21
block escape statements, 4-25 to 4-27
declaration statements, 4-20
flow control statements, 4-23 to 4-25
iteration statements, 4-21 to 4-23
keywords, 4-19 to 4-20
statistics, 1-5
strings. *See* text objects.
subscript operator, 5-10 to 5-11
subscripting vector and matrix object types, 4-7 to 4-9
syntax highlighting, 4-3
system requirements, 1-1

T

technical support, A-1 to A-2
text files, importing
as numeric object, 3-4
as text object, 3-3 to 3-4
text objects
importing text files, 3-3 to 3-4
purpose and use, 4-10 to 4-11

three dimensional (3D) graphs. *See* visualizing seismic data (3D graph).
Tooltips, 1-8
two dimensional (2D) graphs. *See* visualizing rainfall data (2D graph).
typing and dimensioning of vector and matrix object types, automatic, 4-9 to 4-10

U

unary algebraic operators
    left side, 5-7 to 5-8
    right side, 5-8 to 5-9
unary logical operators, 5-9 to 5-10
user interaction, 4-32 to 4-35
    displaying error messages and warnings, 4-33 to 4-34
    formatting numbers, 4-34 to 4-35
    prompting for input, 3-18, 4-32 to 4-33
    providing input to problem solver, 3-15 to 3-19
        connecting script to input objects, 3-19 to 3-21
        creating scalar objects, 3-17
        text objects as user prompts, 3-18
        using text object for input site, 3-16
user-defined functions, 4-27 to 4-31
    call by reference, 4-29 to 4-30
    calling built-in functions, 4-30
    defining, 4-27 to 4-29
    defining programmatically, 4-29
    function names used as parameters, 4-30 to 4-31
    general form, 4-28

V

vector and matrix object types, 4-6 to 4-10
    automatic typing and dimensioning, 4-9 to 4-10
    default keyword in initializer syntax (note), 4-7
    initializing, 4-6 to 4-7
    subscripting, 4-7 to 4-9
vector initialization operator, 5-11 to 5-12
visualizing rainfall data (2D graph), 3-5 to 3-9
    creating a graph, 3-6 to 3-7
    modifying graph and plot, 3-8
    plotting HiQ data object, 3-7
    working with multiple plots, 3-8 to 3-9
visualizing seismic data (3D graph), 3-9 to 3-11
    modifying 3D graph and plot, 3-10 to 3-11
    rotating and zooming, 3-11
    steps, 3-9

W

warnings, displaying, 4-33 to 4-34
while statement
    purpose and use, 4-22
    syntax and description, 5-27 to 5-28