VEE Pro User's Guide

Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies Inc. shall not be liable for errors contained herein or for any direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this material.

U.S. Government Restricted Rights

The Software and Documentation have been developed entirely at private expense. They are delivered and licensed as “Commercial computer software” as defined in DFAR 252.227-7014 (June 1995), as a “commercial item” as defined in FAR 2.101(a), or as “Restricted computer software” as defined in FAR 52.227-19 (June 1987) (or any equivalent agency regulation or contract clause), whichever is applicable. Use, duplication, or disclosure of Software is subject to Agilent Technologies’ standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015(b)(2) (November 1995), as applicable in any technical data.

Trademark Information

Printing History

Edition 1 - March 2000 Reflects software version 6.0

Conventions Used in This Manual

This manual uses the following typographical conventions:

Getting Started	Italicized text is used for book titles and for emphasis.
Dialog Box	Bold text is used for the first instance of a word that is defined in the glossary.
`File`	Computer font represents text that you will see on the screen, including menu names, features, buttons, or text that you have to enter.
`dir` *filename*	In this context, the text in computer font represents an argument that you type exactly as shown, and the italicized text represents an argument that you must replace with an actual value.
`File` ⇒ `Open`	The "⇒" is used in a shorthand notation to show the location of Agilent VEE features in the menu. For example, "`File` ⇒ `Open`" means to select the `File` menu and then select `Open`.
`Sml` | `Med` | `Lrg`	Choices in computer font, separated with bars (|), indicate that you should choose one of the options.
Press **Enter**	In this context, bold represents a key to press on the keyboard.
Press **Ctrl + O**	Represents a combination of keys on the keyboard that you should press at the same time.

Contents

8. Using Agilent VEE Functions

9. Test Sequencing

10. Using Operator Interfaces

11. Optimizing Agilent VEE Programs

Figures

Tables

Introduction

Introduction

This chapter introduces Agilent VEE and its major features. You will also learn how to install and learn about VEE, and how to obtain VEE support.

Overview of Agilent VEE

Agilent VEE is a graphical programming language optimized for building test and measurement applications, and programs with operator interfaces. This release of the Agilent VEE product family includes VEE Pro 6.0 for groups of engineers that need to create complex test and measurement systems, and VEE OneLab 6.0 for individual engineers and scientists responsible for design and data acquisition.

Advantages of Using Agilent VEE for Test Development

VEE offers many advantages in test development:

- Increase your productivity dramatically. Customers report reducing their program development time up to 80%.
- Use VEE in a wide range of applications including functional test, design verification, calibration, and data acquisition and control.
- Gain instrument I/O flexibility controlling GPIB, VXI, Serial, GPIO, PC Plug-in cards, and LAN instruments. Use “panel” drivers, VXI*plug&play* drivers, ODAS drivers, “direct I/O” over standard interfaces, or imported libraries from multiple vendors.
- Use ActiveX Automation and Controls on PCs to control other applications such as MS Word, Excel, and Access that assist with generating reports, displaying and analyzing data, or putting test results into a database for future use.
- Increase throughput, build larger programs with ease, and become more flexible in instrument management. VEE has a compiler; a professional development environment suited for large, complex programs; and advanced instrument management capabilities.
- Leverage your investment in textual languages such as C/C++, Visual Basic, Pascal, Fortran, and Rocky Mountain Basic.

Creating Programs in Agilent VEE

VEE programs are created by selecting objects from menus and connecting them together. The result in VEE resembles a data flow diagram, which is easier to use and understand than traditional lines of code. There is no laborious edit-compile-link-execute cycle using VEE.

The following two figures compare a simple function programmed first in a textual language (ANSI C) and then in VEE. In both cases, the function creates an array of 10 random numbers, finds the maximum value, and displays the array and maximum value.

Figure I-1 shows the program, called “Random,” in the ANSI C textual language.

```
/* Program to find maximum element in array */
#include <math.h>
main( )
{
double num[10],max;
int i;
for (i=0;i<10,i++){
num[i]=(double) rand( )/pow(2.0,15.0);
printf("%f/n",num[i];
}
max=num[0];
for {i=1;i<10;i++){
if (num[i]>max)max=num[i];
}
printf("/nmax; %f/n",max);
}
```

Figure I-1. The "Random" Program in ANSI C

Figure I-2 shows the same program in VEE.

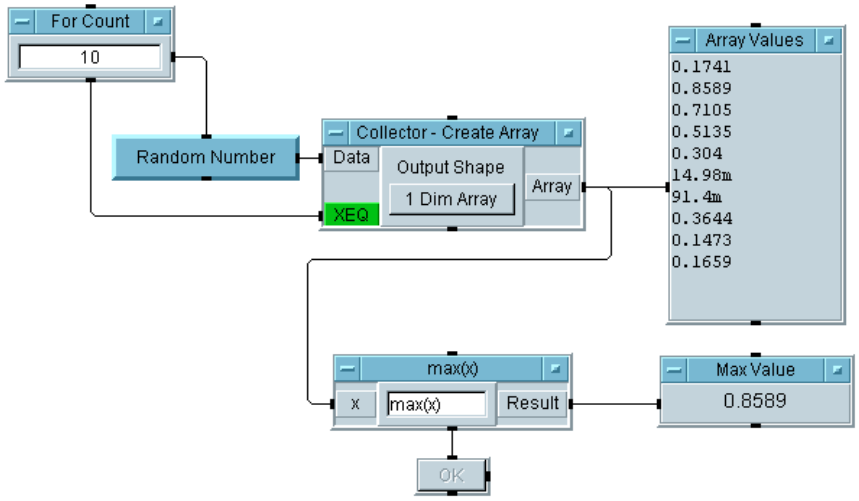


Figure I-2. The Same "Random" Program in VEE

In VEE, the program is built with program elements called **objects**. Objects are the building blocks of a VEE program. They perform various functions such as I/O operations, analysis, and display. When you view the objects with all of their connections, as shown in Figure I-2, this is called the **detail view**. The detail view is analogous to source code in a textual language.

In VEE, data moves from one object to the next object in a consistent way: data input on the left, data output on the right, and operational sequence pins on the top and bottom.

The objects are connected together to form a program. Follow the program from left to right. In the "Random" program shown in Figure I-2, a random number is added to the `Collector - Create Array` object ten times, creating an array. Then the program finds the maximum value in the array, and displays the `Max Value` and the `Array Values`.

Using VEE, with its modular programming approach, you can reduce the time it takes to create programs that control instruments, create customized data displays, and develop operator interfaces. This method of test development leads to productivity gains much greater than conventional techniques.

Note

In Figure I-2, some objects are displayed in detail, and some are displayed with only the name showing. The objects that are displayed in detail are shown in **open view**. Open view allows you to see the details of the object. To save space and increase program execution speed, you can **iconize** objects, or reduce them so that only the names are showing.

For example, in Figure I-2, the object labeled `Random Number` is shown as an icon. The object labeled `Create Array` is shown using an open view. The open view is larger and more detailed. Object views are discussed in more detail in "Changing Object Views" on page 31 of Chapter 1, "Using the Agilent VEE Development Environment."

Creating Operator Interfaces in Agilent VEE

An additional benefit of programming in VEE is that it only takes a few minutes to create an operator interface.

Using the "Random" program from Figure I-2, the objects that the operator needs to see are selected and put into a **panel view**. A panel view shows only the objects the operator needs to run the program and view the resulting data. Figure I-3 shows the panel view of the "Random" program in Figure I-2.

Note

The program and its operator interface are different views of the same VEE program. You can go back and forth from one view to the other by clicking the detail view and panel view buttons in the window title bar in VEE. Any edits or updates that you make to a program (detail view) are automatically made to the operator interface (panel view). You can also secure the operator interface from unwanted changes.

For more information about creating an operator interface, refer to "Creating a Panel View (Operator Interface)" on page 91.

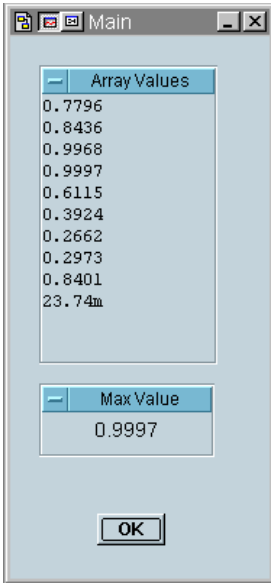


Figure I-3. Panel View (or Operator Interface) of VEE Program

With VEE, you can perform certain tasks in minutes that might take days in a textual language.

- Create colorful, intuitive front ends to programs.
- Create operator interfaces that can be used with a keyboard and mouse, or keyboard input only.
- Choose from a wide assortment of user input and data display features.
- Use pop-up panels to create focus and conserve screen space.
- Secure programs from unwanted tampering.
- Use labels, choose colors and fonts, and add beepers, notepads, buttons, and switches in a variety of formats.
- Use your own or standard off-the-shelf ActiveX Controls (PC only) for user input or displaying data.

Leveraging Existing Test Programs with Agilent VEE

On all supported operating systems, VEE provides mechanisms for linking conventional test programs as well as commercial applications. For example, you could use VEE to sequence existing tests in Rocky Mountain Basic, C, C++, Visual Basic, Fortran, or Pascal (or any compiled or interpreted language on your operating system). VEE also provides a number of interprocess communication features to share data with commercial applications such as databases or spreadsheets.

On PCs, VEE supports standard ties to ActiveX Automation and Controls, and DLLs. On HP-UX, VEE supports Named Pipes and Shared Libraries.

Controlling Instruments with Agilent VEE

VEE provides many options for controlling and communicating with instruments.

- Use panel drivers (instrument drivers) for over 450 instruments from different vendors plus all drivers available from various vendors that are VXI*plug&play* compatible in the Windows 95, Windows 98, Windows 2000, Windows NT 4.0, or HP-UX frameworks.
- Use VEE's Direct I/O to send instrument command strings over standard interfaces such as GPIB (IEEE - 488), GPIO, RS 232, VXI, or LAN-based instruments for remote testing.
- Control PC plug-in boards from any manufacturer that supplies a standard ODAS driver, or a Dynamic Link Library with the board.
- Use direct VXI backplane control using embedded PCs or workstations.
- Control a great variety of instrument types with an easy, organized instrument management capability.

Improving Test Capabilities with Agilent VEE

The VEE products offer the following features and benefits:

- Reduced development and maintenance time with graphical programming.
- Integration with conventional languages like C, C++, Visual Basic, Pascal, Fortran, and RMB.
- Convenient and flexible operator interface capabilities.
- Support for most popular test platforms.
- Use of ActiveX Automation and Controls.
- Agilent Technologies' excellent array of support options.
- Easy and powerful documentation tools.
- Ease of porting test data to standard spreadsheets and word processors for reports.
- Interprocess communication tools to link with other applications such as relational databases or statistical analysis packages (VEE Pro 6.0 only).
- Debugging tools that make the development and maintenance of large, complex programs efficient (VEE Pro 6.0 only).
- Powerful test executive tools included with the product (VEE Pro 6.0 only).
- Remote test capabilities with VEE's Web monitoring features (VEE Pro 6.0 only).
- Unlimited runtime for distribution of your programs (VEE Pro 6.0 only).
- Low cost site licenses (VEE Pro 6.0 only).

Installing and Learning About Agilent VEE

This section gives guidelines to install and learn about VEE, including installing VEE, learning about VEE, using VEE, and obtaining VEE support.

Installing Agilent VEE and I/O Libraries

For information on installing VEE Pro 6.0 and I/O Libraries, refer to the installation materials you received with VEE. (The I/O Libraries are used by VEE to communicate with instruments.)

Installing and Distributing VEE Pro 6.0 RunTime (in online Help) shows how to install and distribute the RunTime version of VEE Pro 6.0. The RunTime version is used to run VEE programs on PCs that do not have the VEE software installed. For more information about the runtime environment, refer to online `Help`. Select `Help`, `Contents and Index`, and `Installing and Distributing Agilent VEE Pro RunTime`. If desired, you can print the information.

Learning about Agilent VEE

To learn more about using VEE, you can watch the VEE multimedia tutorials, use online help, refer to manuals (including this one), and attend VEE classes.

- *VEE Multimedia Tutorials:* The VEE Multimedia Tutorials, located in the `Help` ⇒ `Welcome` menu of VEE, are video presentations that explain many key concepts of VEE. They demonstrate how to use VEE menus, edit objects, and run programs. Each presentation takes three or four minutes to complete, and you can watch them as many times as you like. You can pause the Tutorial, run VEE to try what you have learned, and then resume the Tutorial.

- *VEE Online Help:* One way to learn about the new features of VEE is to select `Help` ⇒ `Contents and Index` ⇒ `What's New in Agilent VEE 6.0`. Read `Help` ⇒ `Welcome` ⇒ `Introduction` for an overview of the VEE product.

There are many other features of online help as well. For more information, refer to “Getting Help” on page 25, and “Using Online Help” on page 99.

- *VEE Manuals:* The manual set for VEE includes this manual, *VEE Pro User’s Guide*, and the *VEE Pro Advanced Techniques* manual.
- *Agilent VEE Classes:* For information about VEE classes, check the Web site `http://www.agilent.com/comms/education`.

Note

The VEE programs for many of the lab exercises and programming examples in this manual are included in VEE, under `Help` ⇒ `Open Example...` ⇒ `Manual` ⇒ `UsersGuide`.

Ordering Free Evaluation Software

Free evaluation software is available on a CD or by downloading from the VEE website. To order the Agilent Technologies VEE Evaluation Kit CD, call (800) 829-4444 in the U.S. or contact Agilent Technologies offices worldwide at:

`http://www.agilent.com/tmo`

MATLAB Script Overview

MATLAB® Script is a subset of the standard, full-featured MATLAB from The MathWorks. It gives users direct access to the core set of MATLAB functionality, such as advanced mathematics, data analysis, and scientific and engineering graphics. The MATLAB Script object can be easily included in any Agilent VEE program.

MATLAB Script includes hundreds of functions for:

- Data analysis and visualization
- Numeric computation, including:
  - Linear algebra and matrix computation
  - Fourier and statistical analysis
  - Differential equation solving
  - Trigonometric and fundamental math operations
- Engineering and scientific graphics, such as:
  - 2-D and 3-D display, including triangulated and gridded data
  - Volume visualization of scalar and vector data
  - Quiver, ribbon, scatter, bar, pie, and stem plots

Signal Processing Toolbox

MATLAB Script for VEE also includes a subset of the MATLAB Signal Processing Toolbox, which is built on a solid foundation of filter design and spectral analysis techniques. Functions are included for:

- Signal and linear system models
- Analog filter design
- FIR and IIR digital filter design, analysis, and implementation
- Transforms such as FFT and DCT
- Spectrum estimation and statistical signal processing
- Parametric time-series modeling
- Waveform generation

About Full-Featured MATLAB

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. MATLAB includes hundreds of functions for:

- Data analysis and visualization
- Numeric and symbolic computation
- Engineering and scientific graphics
- Modeling, simulation, and prototyping
- Programming, application development, and GUI design

MATLAB is used in a variety of application areas including signal and image processing, control system design, financial engineering, and medical research. The open architecture makes it easy to use MATLAB and

companion products to explore data and create custom tools that provide early insights and competitive advantages.

As a VEE user, you can incorporate the full power of MATLAB and the Signal Processing Toolbox for applications involving data analysis, visualization and modeling. By upgrading to the full versions of these products, you can use a wide range of additional MATLAB features in VEE applications, such as creating user-defined functions (M-files), and access to the MATLAB command window, the MATLAB Editor/Debugger, and the Signal Processing GUI.

Note

For more information about using MATLAB Script objects in VEE programs, refer to “Using MATLAB Script in Agilent VEE” on page 187 of Chapter 4, “Analyzing and Displaying Test Data.”

Obtaining Agilent VEE Support

You can obtain VEE support via the Web or by telephone (for startup assistance).

Obtaining Information on the World Wide Web

The VEE website offers a variety of information, including application notes, user tips, technical information, and information about VEE partners, such as PC plug-in board vendors.

- *Main VEE Website*: `http://www.agilent.com/find/vee`.
- *For Current Support Information:* While connected to the network, in VEE, click `Help` ⇒ `Agilent VEE on the Web` ⇒ `Support`. Figure I-4 shows how to select support in VEE. Or, in the Web browser, select `http://www.agilent.com/find/vee` and click "Support".

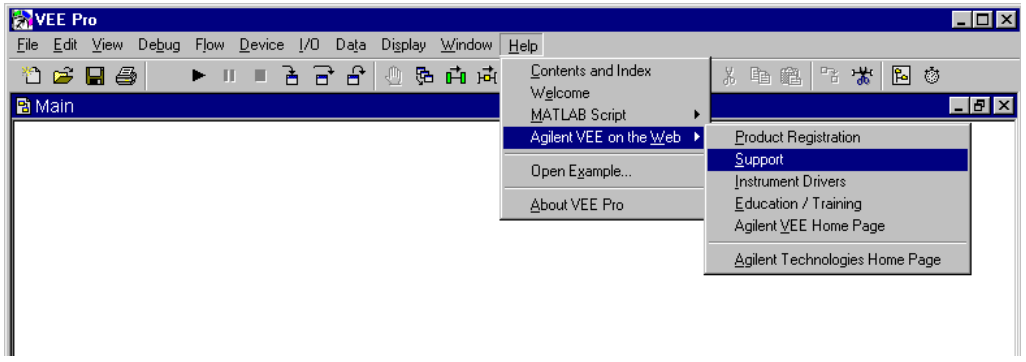


Figure I-4. Contacting Product Support in VEE Help Menu

- *For Complimentary Startup Assistance:* see phone support information in online help. In VEE, click `Help` ⇒ `Contents and Index` and choose `Agilent VEE Support`.

Sources of Additional Information for MATLAB

For complete, detailed information on using the MATLAB Script object, refer to the MATLAB Script Help Desk. In VEE, select `Help` ⇒ `MATLAB Script` ⇒ `Help Desk`. The `MATLAB Help Desk` will appear in a Web browser.

For further information about MATLAB, MATLAB Toolboxes, and other products from The MathWorks, visit `www.mathworks.com` or call 508-647-7000.

Other sources of information include:

- Complete MATLAB documentation: `www.mathworks.com/access/helpdesk/help/helpdesk.shtml`
- MATLAB Upgrade Offer: A special offer is available for VEE Pro 6.0 and VEE OneLab 6.0 users. To find out more, go to `www.mathworks.com/veeupgrade`
- MATLAB Product Information: `www.mathworks.com/products`
- MATLAB Technical Assistance: `www.mathworks.com/support`
- MathWorks Store: `www.mathworks.com/store`
- MathWorks Home Page: `www.mathworks.com`
- Usenet Newsgroup: The `comp.soft-sys.matlab` news group provides a forum for professionals and students who use MATLAB and have questions or comments about it and its associated products.

1

Using the Agilent VEE Development Environment

Using the Agilent VEE Development Environment

In this chapter you will learn about:

- Supported systems
- How to use the Help system
- Starting VEE
- The VEE window
- Working with objects
- Managing the workspace
- Selecting menu items
- Pins and terminals on VEE objects
- Connecting objects to make programs
- Creating, running, printing, saving, and opening programs
- How VEE programs work

Average time to complete: 1.5 hours

Overview

In this chapter, you will learn how to start VEE, how to use menus, and how to work with objects. You will learn about pins and terminals in VEE. You will connect objects together to build a simple VEE program, and learn how VEE programs work.

Interacting with Agilent VEE

This section explains how to use the VEE graphical programming language, including a list of systems supported, how the mouse and menus work, how to get help, how to start VEE, and how to work in the VEE window.

Supported Systems

This version of VEE, version 6.0, is supported on the following systems:

- Windows 95, Windows 98, Windows 2000, and Windows NT 4.0 on a PC.
- HP-UX Workstations (version 10.20 on Series 700). This version of VEE does not run on HP-UX version 11.x, or any version before 10.2.

The Mouse and the Menus

You are probably familiar with the computer's mouse- and menu-driven interface: the pull-down menus, toolbars, and dialog boxes that you control with the mouse and keyboard. VEE uses your computer's interface. In the instructions about using the mouse to operate menus, icons, buttons, and objects, the common techniques are as follows:

- To "click" an item, place the mouse pointer on the desired item and quickly press and release the *left* mouse button.
- To "double-click" an item, place the mouse pointer on the desired item and click the *left* mouse button twice, in rapid succession.
- To "drag" an item, place the mouse pointer on a desired item, *hold the left mouse button down*, and move the item to the appropriate location. Then, release the mouse button.

Note

The right mouse button is used less frequently. You will be advised if you are to click the right mouse button. If your mouse has a middle button, you will not use it for VEE.

Starting Agilent VEE

Windows	Click `Start` ⇒ `Programs` ⇒ `Agilent VEE Pro 6.0`.
HP-UX	From a shell prompt in an HP VUE or X11 window, type: `veetest` **Return**. (The `PATH` variable must include `/usr/bin`.)

The Agilent VEE Window

After you have installed and started VEE, you will see the VEE window shown in Figure 1-1.

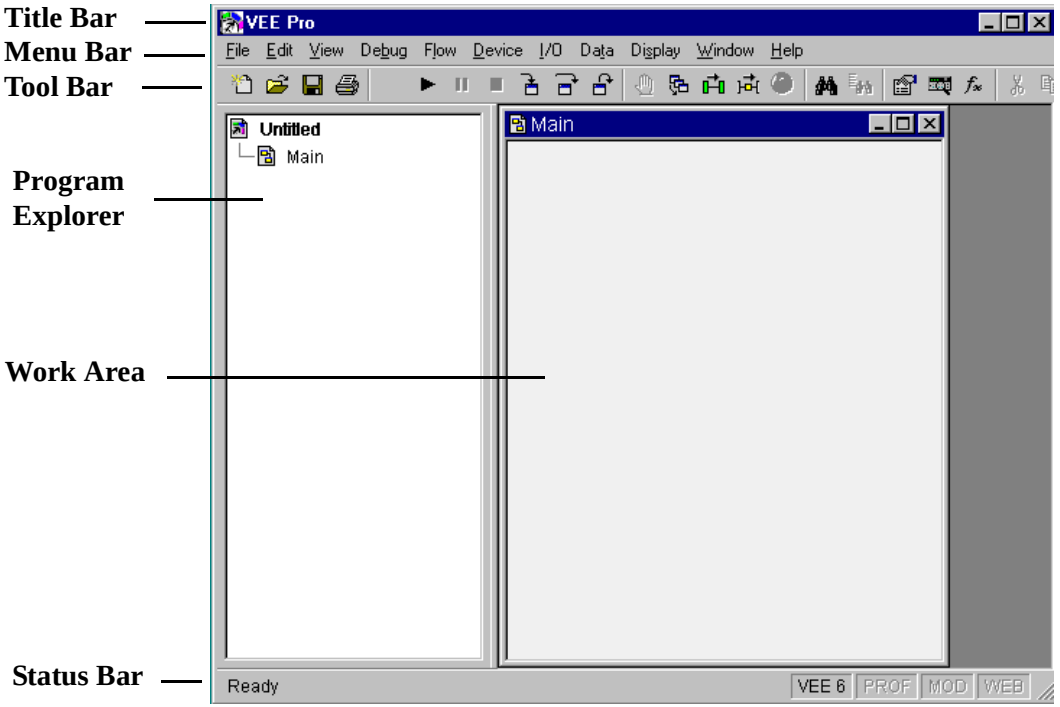


Figure 1-1. The VEE Development Environment

These items describe the parts of the VEE window.

Title bar	The top line in the window contains the VEE icon, the window name, and the minimize, maximize, and close buttons. Move the window by dragging the title bar. Click the VEE icon to get the window's menu.
Menu bar	The second line contains menu items, each of which provides VEE commands or objects.
Toolbar	The third line contains icons, or buttons, that provide direct access (or "shortcuts") to the most commonly used menu commands. (Place the mouse pointer over a button and VEE displays its function.)
Work area	A region in a programming (edit) window such as `Main`, `UserObject`, or `UserFunction` in which you place objects and wire them together.
Program Explorer	A region on the left side of the VEE window showing the structure of the VEE program. The upper corner shows the current program name, such as `myprog.vee`, or it displays `Untitled`. The Program Explorer lets you move among the programming windows. To resize the Program Explorer, move the normal pointer on the right boundary until it changes to a vertical splitter, click, and move.
Main window	A window that contains a work area in which you develop and edit VEE programs. There can be other programming/editing windows, such as `UserObject`.
Status bar	The bottom line displays messages about VEE status, including four status indicators in the right corner. The indicators (from left to right) show: The execution mode The state of the profiler MOD appears when the program has been modified Web server is enabled

Note

This book focuses on VEE version 6.0. If you have an earlier version of VEE (to check your version, click `Help` ⇒ `About VEE Pro`), inexpensive upgrades are available. If you have a support contract for software updates, you will receive the new version automatically.

Getting Help

VEE provides an online `Help` system for the VEE environment, and online `Help` for individual objects and topics. In addition, you can get help in the documentation that came with the computer and its operating system. The PC online `Help` includes information about topics such as:

- Choosing commands on the menu bar
- Selecting and dismissing menu items
- Using toolbars
- Understanding title bars and status bars
- Clicking icons and buttons
- Working with dialog boxes
- Working with various types of windows
- Using online help

To begin, you may want to start with the `Help` ⇒ `Welcome` screen, where you can access the VEE Multimedia Tutorials. The Welcome screen is shown in Figure 1-2.

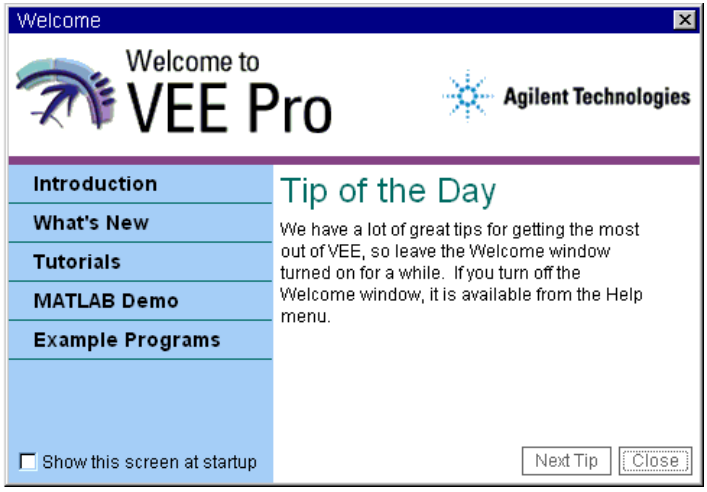


Figure 1-2. The VEE Welcome Screen in Help

VEE online `Help` is designed for your operating system. Click `Help` and the menu shown in Figure 1-3 appears. Help includes contents and index, the Welcome menu (where the Tutorials are located), instrument drivers, Web site information, examples, and version number.

Although you will not need to use VEE documentation to complete this self-paced training, consult the product documentation for more detailed information on any particular feature or concept. Use the Help system to search for VEE topics you need to locate. The Help system can “jump” to related topics.

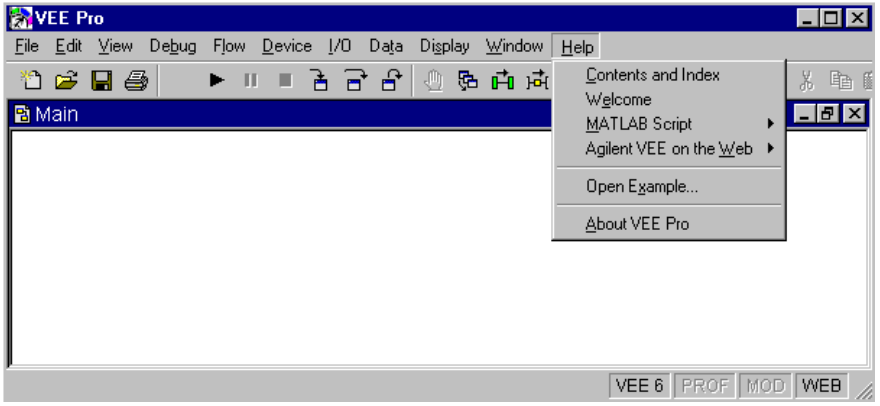


Figure 1-3. Using the Help Menu

Select `Contents and Index` to start VEE Help as shown in Figure 1-4. (The HP-UX screens are slightly different.)

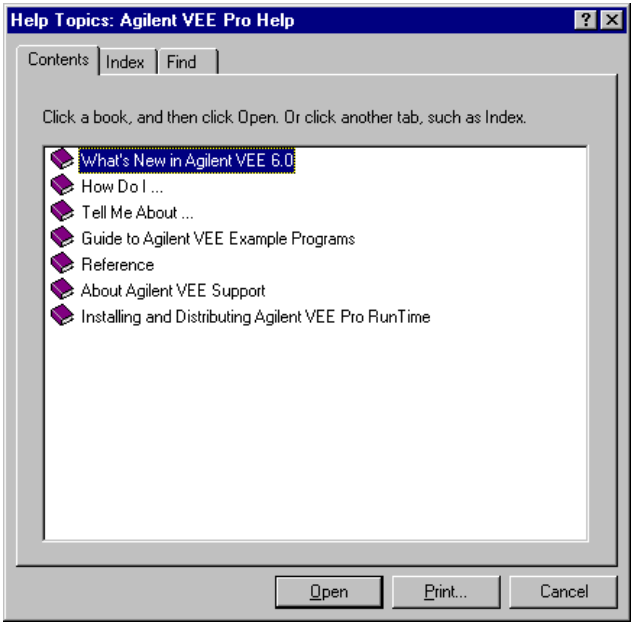


Figure 1-4. VEE Help Contents Tab

The `Help Contents` tab contains the following topics.

What's New in Agilent VEE 6.0	Explains new features.
How Do I...	Provides "how to" information for common tasks.
Tell Me About...	Explains VEE concepts.
Guide to Agilent VEE Example Programs	Summarizes example programs shipped with VEE.
Reference	Provides reference information for all functions and objects.
About Agilent VEE Support	Provides information about getting support for VEE.
Installing and Distributing Agilent VEE Pro RunTime	Explains how to distribute the VEE Pro RunTime environment.

Note

As a shortcut to get help on a selected object and on dialog boxes, press **F1** on your keyboard. In addition, Click `Help` in an object menu to get specific information on that object.

For more information about using specific online `Help` features as you develop programs, refer to "Using the Help Facility" on page 100.

Working with Objects

A VEE program consists of connected objects. To create a program, select *objects* from VEE menus, such as `Flow`, `Data`, and `Display`. Connect the objects via lines that attach to the object pins. (For more information about pins, refer to "Understanding Pins and Terminals" on page 46.). Create a program with a group of connected objects.

This section describes how to select and use objects in a program.

1. Start VEE. Click `Start` ⇒ `Programs` ⇒ `Agilent VEE Pro 6.0` in Windows, or type `veetest` and press **Return** from a shell prompt in HP VUE or X11 window in HP-UX (the PATH variable must include `/usr/bin`).

2. Follow the instructions in this section to experiment with objects.

Note Subsequent exercises assume you have started the VEE software. Refer back to this page or to the section called "Starting Agilent VEE" on page 23 for instructions on starting VEE.

Adding Objects to the Work Area

Pull down an appropriate menu, click the desired object, drag the object to an appropriate location in the work area, and click (the outline will disappear and the object will appear).

1. For example, to add a `Function Generator` object to the work area, select `Device` ⇒ `Virtual Source` ⇒ `Function Generator` in the menu bar as shown in Figure 1-5.

Note The arrow to the right of `Virtual Source` indicates a submenu. Three dots after a menu item indicate that one or more dialog boxes will follow. For example, `File` ⇒ `Save As...` operates this way.

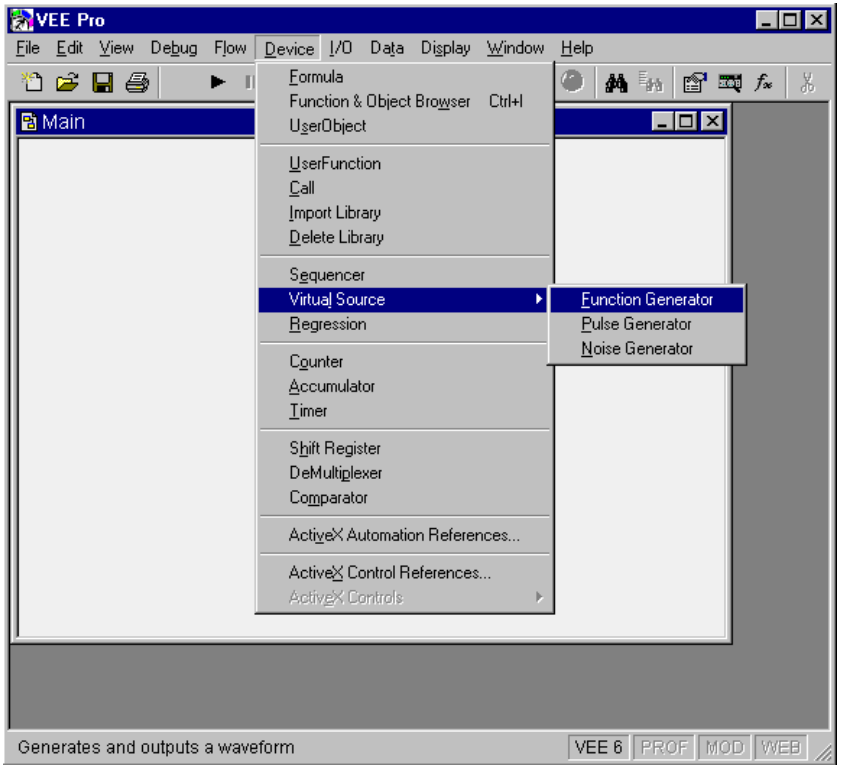


Figure 1-5. Adding Objects to the Work Area

An outline of the object appears in the work area.

2. Move the `Function Generator` to the center of the work area, and click to place the object. The `Function Generator` appears as shown in Figure 1-6.

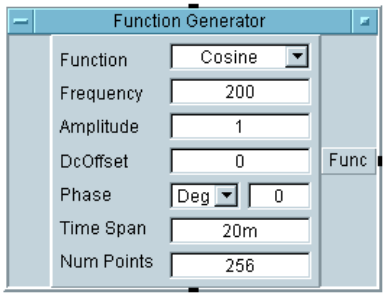


Figure 1-6. Adding a Function Generator Object

Having placed an object in the work area, you can move the object by dragging its title bar, just as you move a window.

Note

Throughout the rest of this manual, a shorthand notation is used to explain instructions. For example, selecting the `Function Generator` object is condensed into the following format:

`Device` ⇒ `Virtual Source` ⇒ `Function Generator`

Note

To give yourself more room on the screen, click `View` ⇒ `Program Explorer`. This will deselect it and remove it from the screen. Menu items are "selected" when there is a check mark displayed before them.

Changing Object Views

VEE displays objects either in "icon view" or "open view," as shown in Figure 1-7.

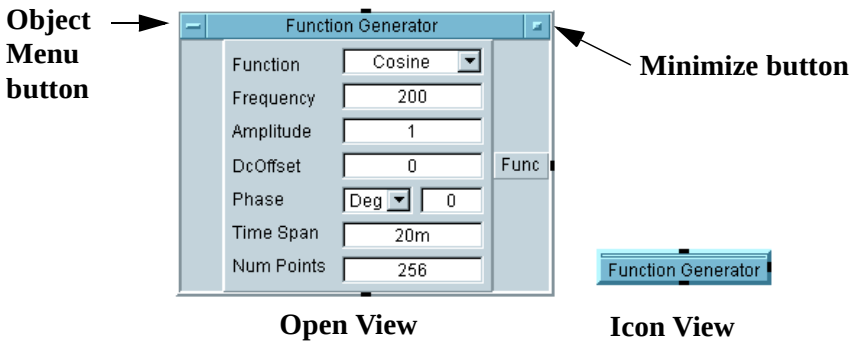


Figure 1-7. Object in Open View and Icon View

The iconic view conserves space in the work area and makes programs more readable. The open view provides more detail and allows you to edit the properties and settings of an object.

1. To switch from an open to iconic view, click the **Minimize** button (the box on the right end of the object's title bar).

2. To return to an open view, double-click the object icon view (anywhere on the object).

Note

The object menu also has `Minimize` and `Restore` selections. To display the object menu, click on the **Object Menu** button on the left end of the title bar, or right click anywhere on the object.

Not all objects have the same structure or parts, but you can edit objects in their open views and save space in their icon views.

Selecting an Object Menu

Each VEE object has an **object menu** that lets you perform actions on the object, such as `Clone`, `Size`, `Cut`, `Move`, and `Minimize`. Most objects have similar attributes, but there are differences, depending on the functionality of the object. See online help for the specific object from the object menu.

1. To select the object menu, click *once* on the object menu button. (All object menus open the same way.) The object menu appears, as shown in Figure 1-8. (Do not double-click the object menu button. That is the shortcut for deleting the object.)

2. Now you can click one of the object menu choices to perform the action you desire. Or, to dismiss the menu, click an empty area *outside* the menu.

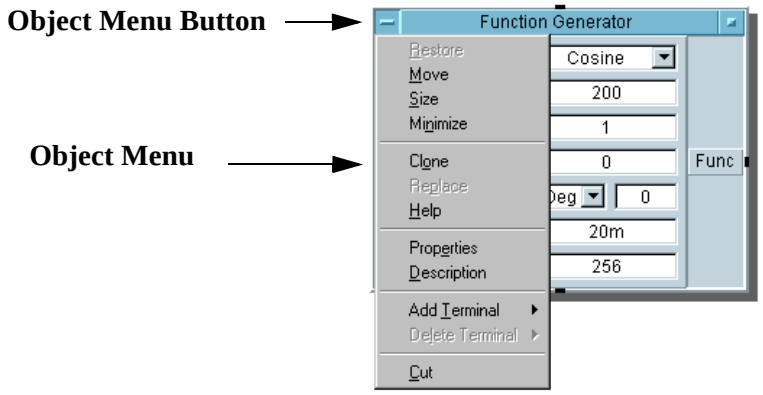


Figure 1-8. Selecting an Object Menu

Shortcut: You can also select the object menu by placing the mouse pointer anywhere on the object body and clicking the *right* mouse button. This works for both the open and icon views.

Moving an Object

1. To move the `Function Generator` object, click `Move` in the object menu, then click and hold the left mouse button. An outline of the object appears.

2. Move the outline to the new location while continuing to hold the mouse button down, as shown in Figure 1-9. Release the mouse button, and the object moves to the new location.

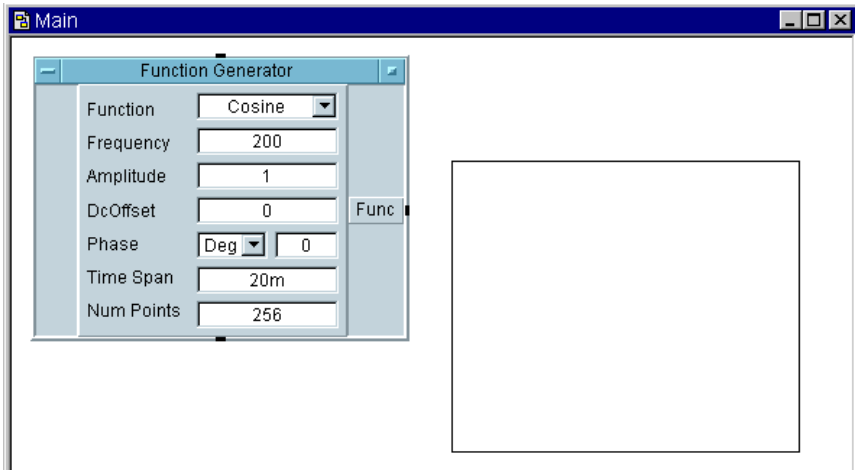


Figure 1-9. Moving an Object

You can also move objects as follows:

- Click the title area of the open view of an object and drag the object to a new location.
- *Except* for buttons, entry fields, pins, terminals, or the four corners (which resize the object), click any part of an open view object and drag the object to a new location.
- Click any part of an icon view object *except* near the four corners (which resize the object), and drag the icon to a new location.

Note

"Object Location Information," located on the status bar (at the bottom of the VEE window) gives the X and Y position (in pixels) of the upper-left corner of the outline relative to the upper-left corner of the workspace. To view an object's exact location, left click on an object to select it and hold the left mouse button down. The location is displayed in the status bar. Use this information when you need to place an object in an exact position.

Duplicating (or Cloning) an Object

The **Clone** operation creates a duplicate object exactly, including any changes you have made such as sizing or renaming. Cloning is a shortcut for cutting and pasting.

1. Open the object menu and select `Clone`. An outline of the duplicated object appears.

2. Move the outline to the desired location, and click to place the object. The cloned object appears, while the original object remains. In Figure 1-10, the `Function Generator` has already been cloned once, and the object menu has the command selected to clone it again.

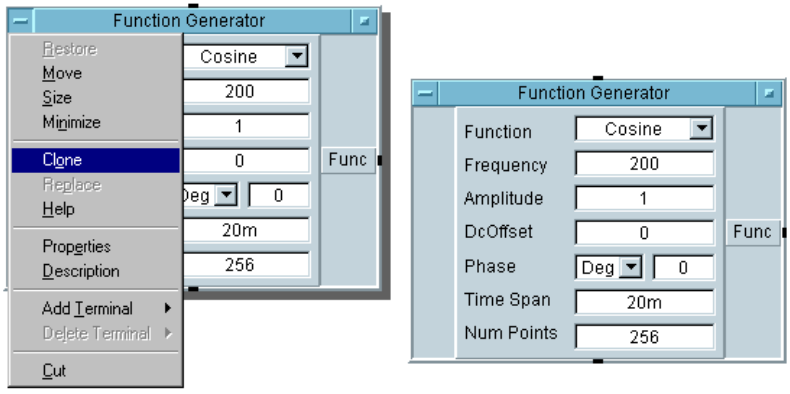


Figure 1-10. Cloning an Object

Copying an Object

This action copies an object to the clipboard, so you could Paste it to VEE or another application such as MS Paint or MS Word.

1. Click on an object to highlight it, then click `Edit` ⇒ `Copy`.

   -OR-

   Click on an object to highlight it, then press **Ctrl-C**.

Deleting (Cutting) an Object

To delete (or Cut) an object from the work area, go to the object menu for the object you want to delete and click `Cut`. For example, go to the object menu for the `Function Generator` and click `Cut`. The object disappears from the work area, but it is saved in the **cut buffer**.

1. Open the object menu, and select `Cut`.

   -*OR*-

   Select the object (click on it) and press **Ctrl-X**.

   -*OR*-

   Place the mouse cursor over the object menu and double-click.

Note Be careful, as it is easy to accidentally delete an object by *double-clicking* its object menu button. If you do delete an object by accident, use the `Paste` toolbar button (or `Edit` ⇒ `Paste`) to recover the object and all connections to it.

Pasting an Object ("Undoing" a Cut)

To paste a copied or deleted (cut) object back into the work area, follow these steps.

1. After an object has been copied or deleted, click `Edit` ⇒ `Paste`. An outline of the object appears. Place the object and click to release it.

   -*OR*-

   Press **Ctrl-V**.

Note If the object had lines attached, these connections will be maintained. This action operates like an "undo" in other programs. It's not called "undo" because it doesn't apply to all VEE programming actions. (It also works on groups of objects that have been deleted.)

Changing the Size of an Object

1. Place the mouse pointer over any of the four corners of the object until you see a sizing arrow, then click-and-drag to the desired size. Release to resize. Figure 1-11 shows an object being resized with the sizing arrow.

   -OR-

   Open the object menu and click `Size`. The mouse pointer becomes a "bottom-right-corner" bracket. Move the bracket to the desired position of the lower-right corner and click to resize.

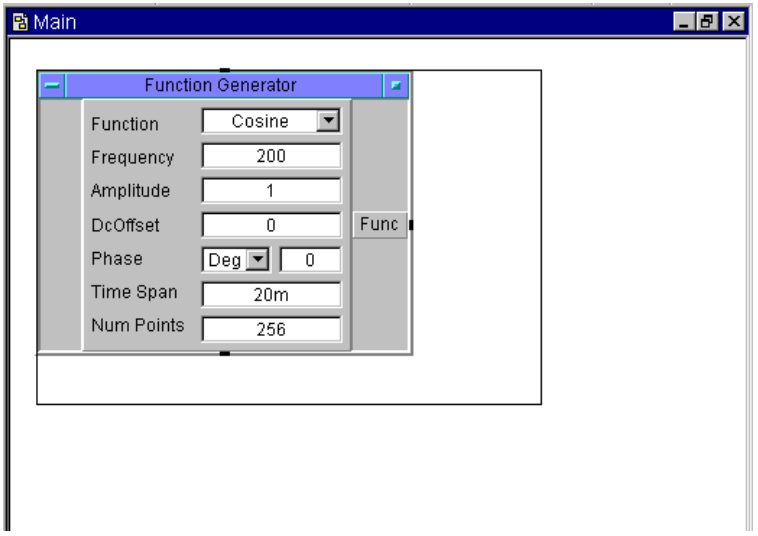


Figure 1-11. Changing the Size of an Object

Changing the Name (Title) of an Object

1. Open the object menu and select `Properties...` A Properties dialog box appears with the current title highlighted, as shown in Figure 1-12.

2. Type the new title and click `OK`. The new title appears in the title area. If you minimize the object, the new title appears in the icon.

-OR-

1. Double-click the object title bar to go directly to the Properties dialog box.

2. Type in the new title and click `OK`.

Note

You can save time by using standard keyboard and mouse editing techniques. For example, in the `Properties` dialog box `Title` field, if you click at the extreme left edge of the edit area the cursor will appear there. You can then add new text without deleting the existing title.

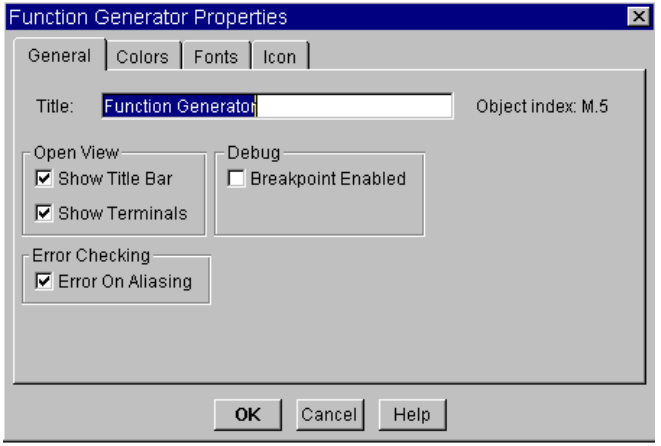


Figure 1-12. Changing the Title of an Object

Selecting or Deselecting Objects

1. To select an object, click on the object and a shadow appears behind it. For example, in Figure 1-13, the `For Count` object is selected.

2. To deselect an object, move the mouse pointer over any open area and click. The shadow disappears. For example, in Figure 1-13, the `Formula` object is not selected.

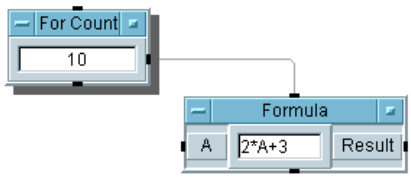


Figure 1-13. Selected and Deselected Objects

Note

The word “select” is also used to indicate choosing a menu item, but the context makes the meaning obvious.

Selecting Several Objects

If you click to select an object, only one object is selected. If you click again to select another object, the previous object is deselected and its shadow disappears. To select multiple objects when you want to perform an operation on all of them at once, such as `Cut`, follow these steps:

1. Press and hold down the **Ctrl** button as you click on different objects. Release the **Ctrl** button after you have highlighted all the objects you want to select.

   -OR-

   Press **Ctrl**, then click-and-drag a rectangle around the objects to be selected. The selected objects become shadowed.

Selecting/Deselecting All Objects

1. To select all objects, click `Edit` ⇒ `Select All`. (Or press **Ctrl-A**.)

2. To deselect all objects, click on an open area in the window.

Copying Multiple Objects

1. Copy the selected objects by placing the cursor on an object. Press and hold **Ctrl** while using the left mouse button to drag the multiple objects (outlines) to a desired location. A new instance of each object appears in the desired location.

   -OR-

   Or, use `Edit` ⇒ `Copy` to copy the selected objects to the cut buffer. Click `Paste` (in the `Edit` menu or on the toolbar), move the objects (outlines) to a desired location, and click the left mouse button. Figure 1-14 shows objects during copying.

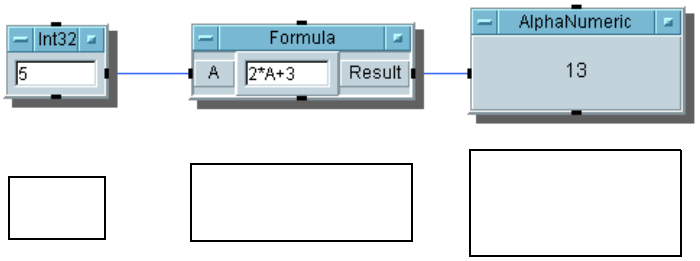


Figure 1-14. Multiple Objects during Copying

Note

In VEE for Windows, objects that you cut or copy are also placed on the Clipboard. You can paste them into other Windows applications that support the Windows Clipboard.

Editing Objects

There are several ways to edit objects in VEE. Different editing menus display different choices. Choose an editing menu or icon as follows:

1. Click `Edit` on the VEE menu bar to display the `Edit` menu, and select the operation you want. The commands in the `Edit` menu are the same for all of VEE.

   -OR-

   Click on an icon on the VEE toolbar. The VEE toolbar contains icons for frequently used editing commands such as `Cut`, `Copy`, and `Paste`.

   -OR-

   Open the object's object menu by clicking on it, and select the operation you want. Object menus include editing operations specific to an object, such as the `Properties` menu, that are not located in the main `Edit` menu. The commands in the object menu also vary depending on the type of object. For example, compare the object menus for the `Device` ⇒ `Formula` and `I/O` ⇒ `To` ⇒ `File` objects. The two menus contain different choices that are specific to the object.

   -OR-

   Place the mouse pointer anywhere on *blank* work area space and click the *right* mouse button. A pop-up `Edit menu` appears.

Note

Inactive menu items appear in a different shade than active items (they are "grayed out"). For instance, the `Cut`, `Copy`, and `Clone` operations in the `Edit` menu appear in a different shade from active menu items until an object is highlighted in the work area.

Creating Data Lines Between Objects

1. Click on or just outside the data output pin of one object, then click on the data input pin of another, as shown in Figure 1-15. (A line appears behind the pointer as you move from one pin to the other.)

2. Release the cursor and VEE draws a line between the two objects. Notice that if you reposition the objects, VEE maintains the line between them.

Note

For more information on pins, see "Understanding Pins and Terminals" on page 46.

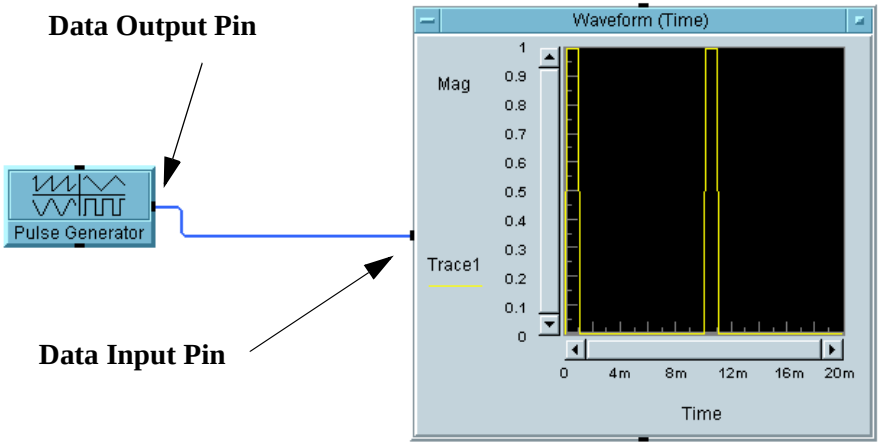


Figure 1-15. Creating Data Lines Between Objects

Deleting Data Lines Between Objects

1. Press **Shift-Ctrl** and click the line you want to delete.

   -OR-

   Select `Edit ⇒ Delete Line` and click the line you want to delete.

Moving the Entire Work Area

1. (Make sure there is at least one icon in the work area.) Place the mouse pointer anywhere on the background of the work area, press and hold the left mouse button, and move the work area in any direction.

Note Scroll bars appear if your program is larger than the work area, as shown in Figure 1-16.

Note If you click near a terminal, a line or "wire" may appear. If this happens, move the pointer to an open area and double-click.

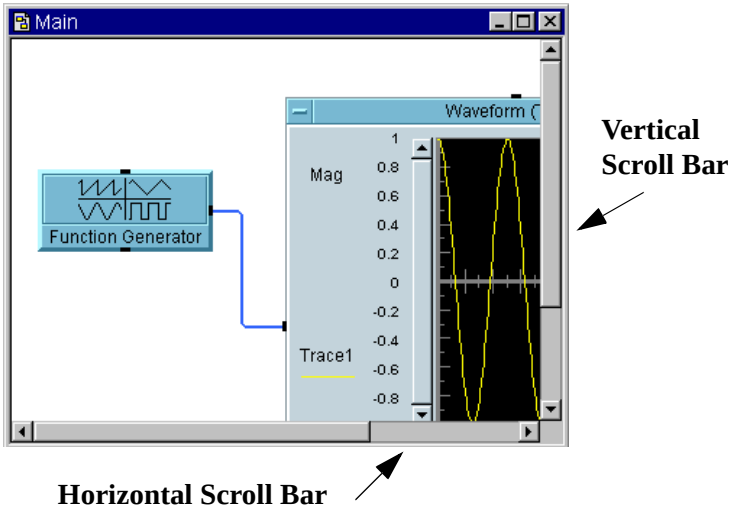


Figure 1-16. Scroll Bars in Work Area

Clearing the Work Area

1. Click `Edit` ⇒ `Select All` and then click the `Cut` button on the toolbar. This cuts all objects in the active window to the `Cut` buffer.

   -*OR*-

   Select `File` ⇒ `New`, or click the **New** button on the toolbar. VEE asks you if you want to save changes.

   -*OR*-

   Clear individual objects by clicking an object to make it active, and then clicking the `Cut` button on the toolbar.

Changing Default Preferences

The `Default Preferences` dialog box changes the default settings in the VEE environment.

1. Click the `Default Preferences` button on the toolbar.

   -*OR*-

   Click `File` ⇒ `Default Preferences`. The `Default Preferences` dialog box appears, as shown in Figure 1-17.

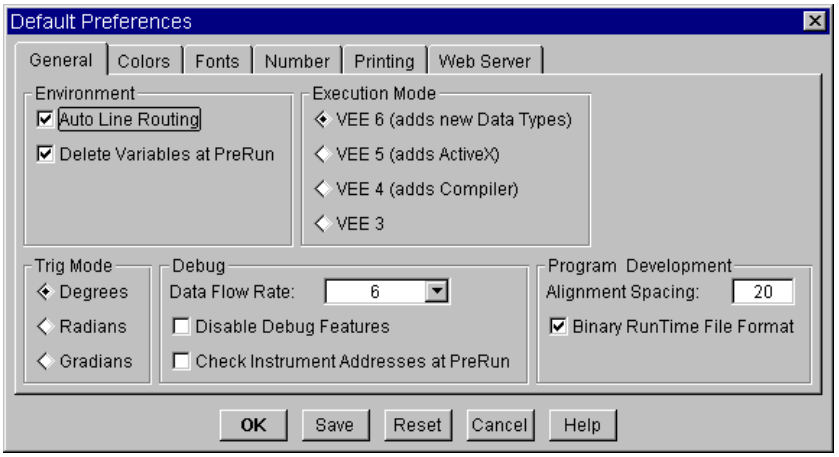


Figure 1-17. Default Preferences Dialog Box

This dialog box has *tabs* that let you select options to edit.

General	The default tab when the `Default Preferences` dialog box appears (shown previously). You can change the values of the displayed parameters; for example, `Environment` and `Execution Mode`.
Colors	Lets you customize the colors in the VEE environment.
Fonts	Lets you customize the fonts in the VEE environment.
Number	Lets you change the default number format.
Printing	Lets you set the values of the parameters for a printer.
Web Server	Lets you enable the built-in Web server to monitor and troubleshoot a program from a remote Web browser.

For more information, select `Help` ⇒ `Contents and Index` from the VEE menu bar. Then, browse `How Do I...`, `Tell Me About...`, or `Reference`.

Understanding Pins and Terminals

A VEE program consists of the objects in the work area *and* the lines that connect them. The lines that connect VEE objects are connected between object *pins*. Each object has several pins, as shown in Figure 1-18. Figure 1-18 uses the `Formula` object as an example. You can use any object.

.

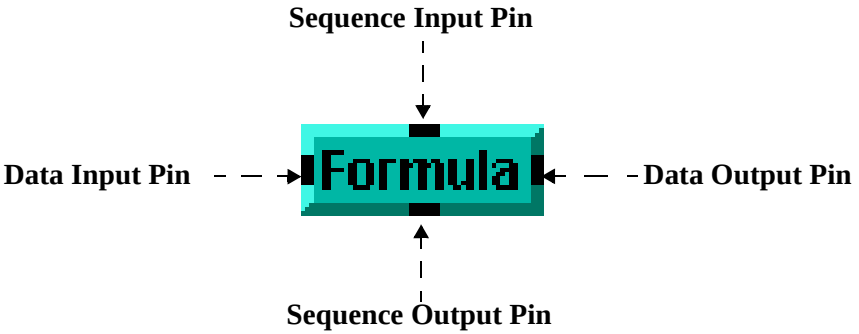


Figure 1-18. Data and Sequence Pins

Data Input Pin	The pin (or pins) on the left-hand side of an object.
Data Output Pin	The pin (or pins) on the right-hand side of an object.
Sequence Input Pin	The pin on the top of an object.
Sequence Output Pin	The pin on the bottom of an object.

Connect the data input and output pins to carry data between objects. By default, the pins execute from top to bottom. The sequence pin connections are optional. If connected, they will dictate an execution order.

Note

For more information, refer to “Following the Order of Events Inside an Object” on page 110.

In an object's open view, the data input and output pins appear as input and output **terminals**. (If the object is in icon view, double-click it to switch to open view.) The terminals carry detailed information such as the name of the terminal, and the type and value of the data being transmitted. The terminal labels are visible only in the open view, and only if the `Show Terminals` option is turned on for that object (see `Properties`... in the object's menu).

For example, Figure 1-19 includes two `Formula` objects. The `Formula` object on the left shows the terminal labels `A` and `Result`. The `Formula` object on the right has `Show Terminals` turned off, and the labels are not visible.

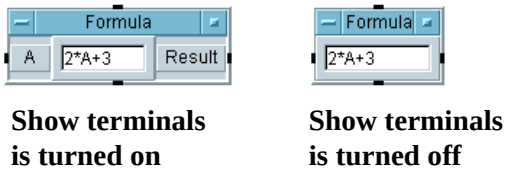


Figure 1-19. Show Terminals on an Object

To turn `Show Terminals` `ON` or `OFF`, select `Properties` from the object menu. The properties dialog box displays a checkbox in front of `Show Terminals` (see Figure 1-20).

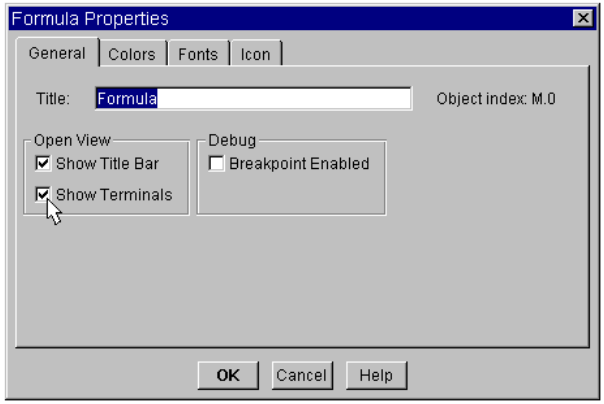


Figure 1-20. Using Show Terminals Checkbox

Click the checkbox to turn `Show Terminals` `OFF`. Click the checkbox again to turn `Show Terminals` back on. Click `OK` after you have made a selection.

Adding a Terminal

You can add terminals to an object. For example, you can add a second data input terminal to the `Formula` object.

1. Open the object menu and select `Add Terminal` ⇒ `Data Input`.

   -OR-

   With `Show Terminals` turned on, you can place the mouse pointer in the "terminal area" (the left margin of the open view object) and press **Ctrl+A** (press the **Ctrl** and **A** keys simultaneously).

Figure 1-21 shows the `Formula` object menu open to add a data input terminal, and another `Formula` object that has a second terminal already added. The new terminal is labeled `B`. If the data inputs are tied to particular functions, as with instrument drivers, you will be given a menu of these functions. Otherwise, the terminals will be named `A`, `B`, `C`... .

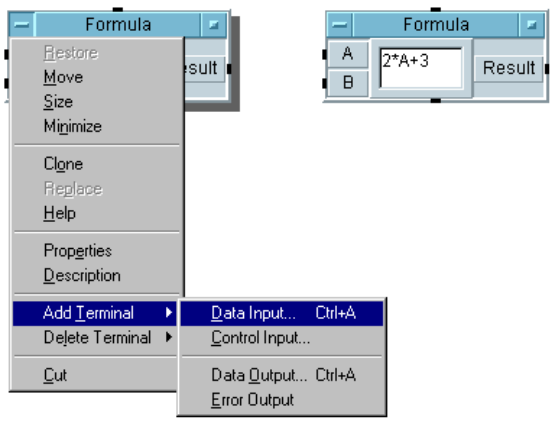


Figure 1-21. Adding a Terminal

Editing Terminal Information

To obtain information about a terminal, double-click the label area. For example, double-clicking `B` causes the dialog box in Figure 1-22 to appear.

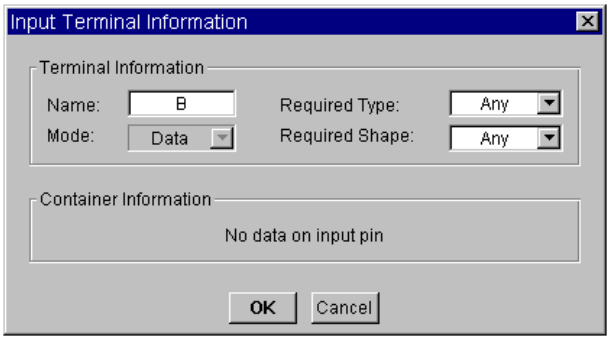


Figure 1-22. Obtaining Terminal Information

You can now edit the terminal. The dialog box has three kinds of fields:

entry field — A field with a white background, but no arrow. It becomes a *type-in* field when you click it. For example, you can click `B` in the `Name` field and rename the terminal.

status field — A field with a gray background that cannot be edited. For example, the `Mode` field cannot be edited.

selection field — A field with a white background that has an arrow on its right-hand side. Clicking the field or its arrow displays a **drop-down list**. For example, if you click `Any` (or the arrow) in the `Required Type` field, you can select another data type from the list by clicking the list as shown in Figure 1-23.

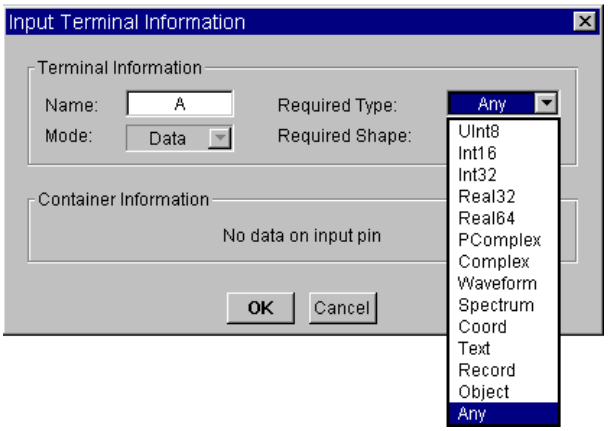


Figure 1-23. Using the Selection Field

If you select a data type other than `Any` for a data input terminal, only the specified type of data or data that can be converted to that type will be accepted by the terminal. Most of the time it is best to leave the `Required Type` and `Required Shape` fields set to `Any`. For more information, select `Help` ⇒ `Contents and Index` from the VEE menu bar. Then, browse `How Do I...`, `Tell Me About...`, or `Reference`.

Deleting a Terminal

1. Open the object menu and select `Delete Terminal ⇒ Input...` or `Delete Terminal ⇒ Output`, choose the input or output to delete, and click `OK`. For example, Figure 1-24 shows the dialog box that appears when you choose `Delete Terminal ⇒ Input...`.

   -OR-

   Place the mouse pointer over the terminal and press **CTRL-D**.

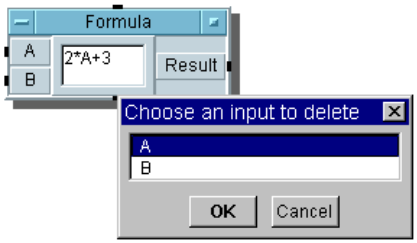


Figure 1-24. Delete Terminal Dialog Box

Connecting Objects to Make a Program

This section introduces VEE programs. In Lab 1-1, you create a VEE program, print the VEE screen, and save the program to a file.

Lab 1-1: Display Waveform Program

A VEE program consists of VEE objects connected in an executable *object diagram*. The following program displays a waveform.

(If VEE is running, clear the workspace by clicking the `New` button on the toolbar, or use `File` ⇒ `New`. Otherwise, start VEE and continue.)

1. Document the program. Select `Display` ⇒ `Note Pad` and place it at the top and center of the work area. Click on the editing area to get a cursor, then enter:

   ```
   Display Waveform generates a cosine waveform and sends
   it to a real time display.
   ```

   You may have to size the `Note Pad`, depending on the screen. To size an object, open the object menu, select `Size`, move the sizing arrow cursor to a corner of the object and drag. You can also click and drag any corner of the object.)

2. Add the `Function Generator` object. Select `Device` ⇒ `Virtual Source` ⇒ `Function Generator`, position the outline on the left side of the work area, and click to place the object. Edit the frequency to 100 by clicking in the `Frequency` field and typing `100`.

3. Add the `Waveform (Time)` object. Select `Display` ⇒ `Waveform (Time)` and place the object to the right side of the work area as shown in Figure 1-25.

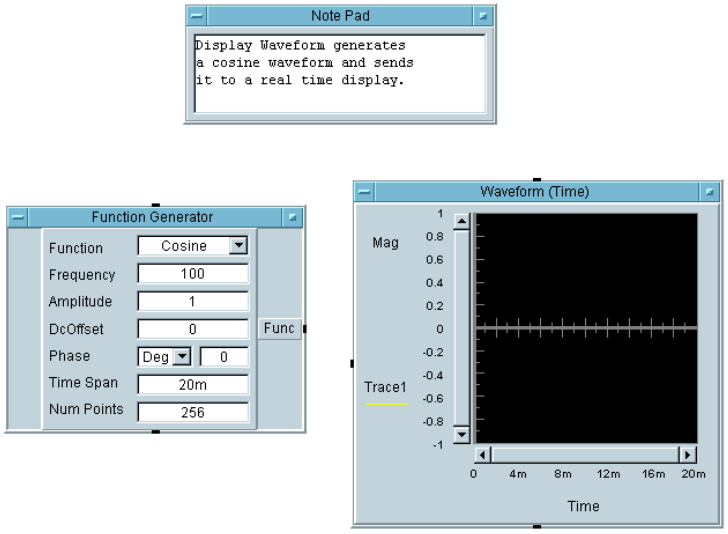


Figure 1-25. Creating a Program

In Figure 1-25, the `Func` label on the `Function Generator` object denotes a **data output pin**, and the `Trace1` label on the `Waveform(Time)` object denotes a **data input pin**. In VEE programs, you connect the data pins among the objects, and this determines the flow of the program.

4. Complete the program by connecting the data output pin on the `Function Generator` (next to `Func` on the right side) to the data input pin on the `Waveform (Time)` display (next to `Trace1` on the left side). To do the connecting, move the cursor to one of the pins.

   The cursor shape changes when it is near a pin where a connection is allowed. Click the left mouse button, move the mouse cursor to the other pin, and click again. A line is automatically routed between the two pins and the program is complete.

   Try moving one of the objects by dragging on its title bar. (Do not drag a pin or terminal, or a line will appear.) The line automatically reroutes to the logical path between the two objects.

If the lines appear to be scrambled, use `Edit` ⇒ `Clean Up Lines` to reroute the lines in the program.

Running a Program

5. Continuing with the same exercise, click the **Run** button on the toolbar to run the program, or use `Debug` ⇒ `Run`. The program displays a 100 Hz Cosine wave in the `Waveform (Time)` display as shown in Figure 1-26. (Your object might have a different frequency, which is not important to the example.)

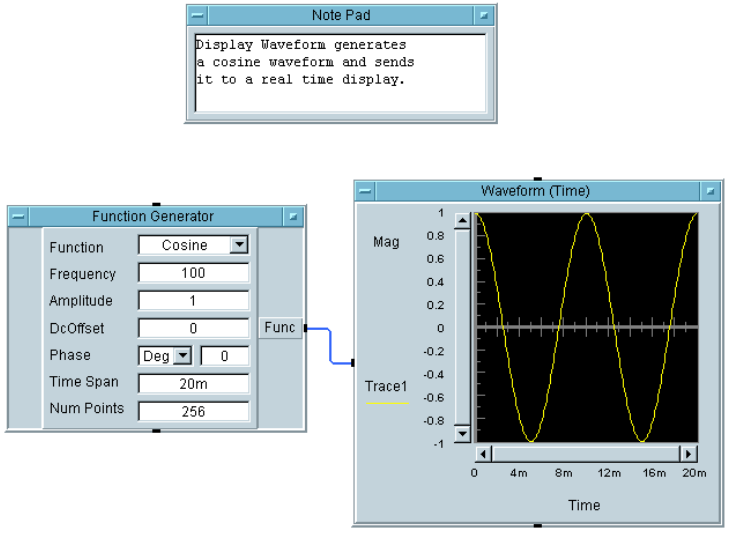


Figure 1-26. Running a Program

In addition to the **Run** button on the toolbar, you can use the **Stop**, **Pause**, and **Step** buttons on the toolbar to control the program. If you pause a running program, use the **Resume** button (same as the **Run** button) to resume. You can use the **Step Into** button on the toolbar to run a program one object at a time.

When instructed to run the program, click the **Run** button on the toolbar, or press **Ctrl**+**G**. Other keyboard shortcuts include the following:

Pause	**Ctrl**+**P**
Resume	**Ctrl**+**G**
Step Into	**Ctrl**+**T**

Changing Object Properties

You have seen how to change some properties of an object by selecting its object menu ⇒ `Properties`. You can also change the more common properties of an object directly in its open view. You may have noticed that the `Function Generator` object has two kinds of fields. A field with an arrow on its right-hand side is a *selection field*.

6. Continuing with the same example, click `Cosine` (or the arrow) in the `Function` field. A drop-down list of selections appears. Click `Sine` to select the Sine function as shown in Figure 1-27, noticing that the `Function` field has changed from `Cosine` to `Sine`.

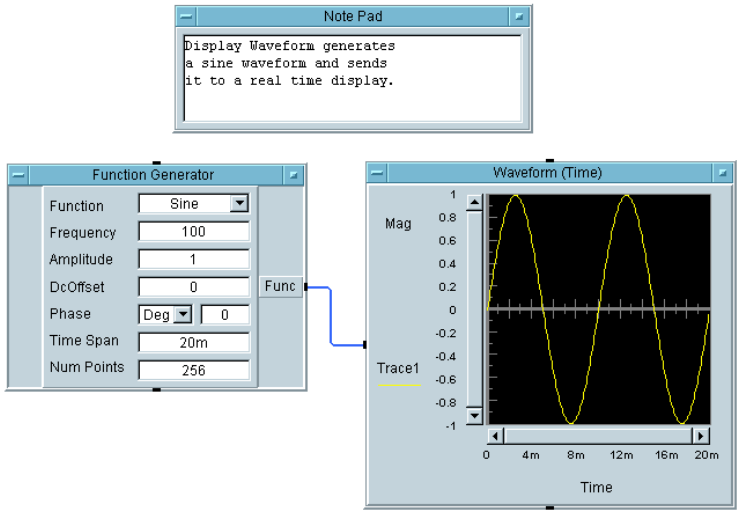


Figure 1-27. Changing the Function Field to Sine Wave

Some fields in dialog boxes do not have arrows. These are entry fields, which become *type-in* fields when you click them. Just click a field and a cursor appears. You can use standard keyboard and mouse editing techniques to move the cursor and enter a desired value.

7. Click the `Frequency` field to the right of the value `100`, and while holding the mouse button down, move the mouse to the left to highlight the last `0`, as shown in Figure 1-28.

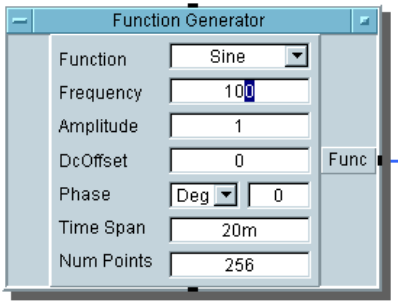


Figure 1-28. Highlighting a Frequency Field Number

8. Press **Delete** to delete the last `0`, changing the `Frequency` value to `10`. Run the program. It should look like Figure 1-29.

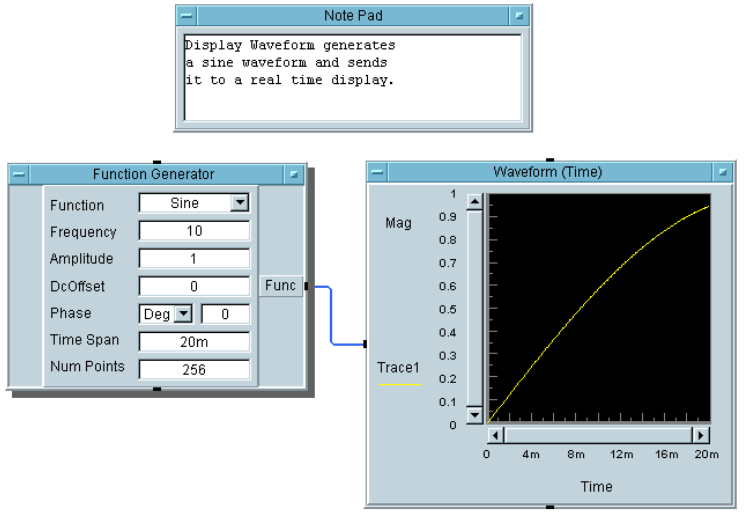


Figure 1-29. Example: Changing the Frequency Field to 10 Hz

The displayed waveform is now a 10 Hz sine wave. You may want to try changing a few object parameters as follows:

- Click `Deg` (or the arrow) in the `Function Generator` object and change the phase units to `Rad`. Next, click the `Phase` value field and enter the value `PI`. Run the program and note the phase shift in the displayed waveform. Then, change the `Phase` value back to `0` and the units back to `Deg`.

- The y-axis limits of the `Waveform (Time)` object are preset to `-1` through `1`. Click the y-axis name `Mag` to open a dialog that lets you change the settings. Click the fields for `Maximum` and `Minimum` to change the limits to `2` and `-2`. You will see the waveform displayed within the new limits. To change similar parameters for the x-axis scale, click `Time`.

Printing the Screen

9. Continuing with the same example, to print the screen, select `File` ⇒ `Print Screen`. On Windows, the dialog box in Figure 1-30 appears.

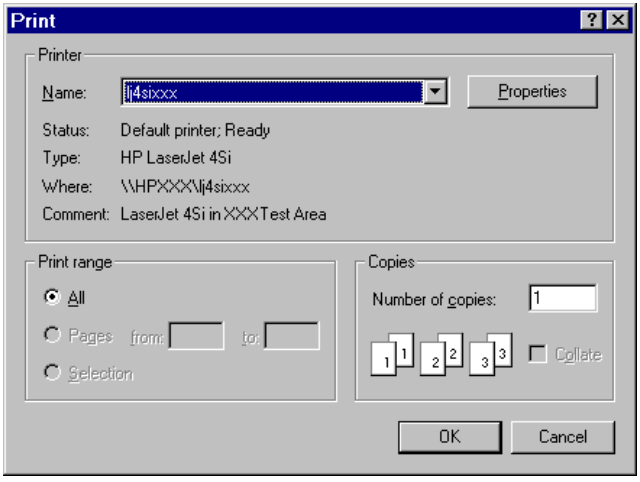


Figure 1-30. Printing the Screen

When you click `OK`, VEE prints the screen on the default printer named in the dialog box. You can select another printer, change the print range, and enter the number of copies. Click the `Properties` button for more selections. Different print drivers may use different dialog boxes. For further information about using Windows dialog boxes, see *Microsoft Windows Help*.

On HP-UX, the dialog box in Figure 1-31 appears.

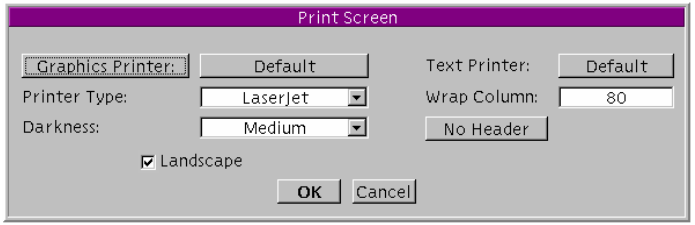


Figure 1-31. Print Screen Dialog Box

When you click `OK`, VEE prints the screen on the selected printer. This dialog box lets you select a graphics printer or text printer. It also lets you change the configuration of these devices before you print.

Shortcut: Click the `Print Screen` button on the toolbar to print the screen directly.

Saving a Program

You can save a program at any time. (You can save whatever is in the work area, whether it is a complete program or not).

10. Continuing with the same example, select `File` ⇒ `Save As...` and complete the dialog box.

    A dialog box entitled `Save File` appears. Figure 1-32 shows the PC format for this box.

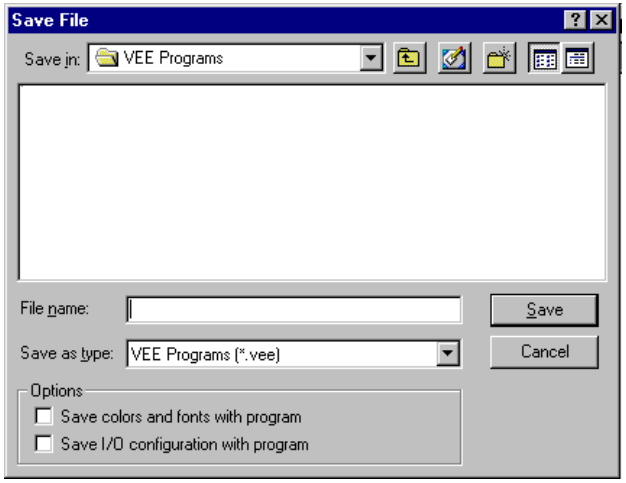


Figure 1-32. The Save File Dialog Box (PC)

11. By default, VEE for Windows saves files in the `VEE Programs` sub-directory in your `My Document` directory. To save the current program, type in the name `simple-program` in the `File name` field and click `Save`. If you do not type it in, VEE automatically adds the `.vee` extension to the file name.

Note

In VEE for Windows, you can use the long file names allowed by Windows 95, Windows 98, Windows 2000, and Windows NT 4.0.

In the PC `Save File` dialog box, you can make changes to the different fields as follows:

Save in	You can change the directory or drive by opening the drop-down menu. Double-click a folder to open it.
File name	Type in a file name of your choice.

Save as type	VEE programs are normally saved with the `.vee` extension, but you can change the file type if you wish. If you type a file name without the extension, the `.vee` is automatically added.
Save colors and fonts with program	(Optional) If you have changed program colors and fonts, using the `Default Preferences` menu, and you want others who load the program to get the colors and fonts you selected (rather than their defaults), click to check this item. When checked, VEE saves the changes you have made to the default configuration as part of the program.
Save I/O configuration with program	(Optional) If you have configured an instrument in the `Instrument Manager`, and you want others who load the program to get the instruments you configured rather than their defaults, it is recommended that you check this item. When checked, VEE saves the I/O configuration as part of the program.

Note

If you are using the evaluation kit software, VEE will only let you save programs to one file, `EVAL.VEE`, so just write over this file for the different examples.

For the HP-UX version of the `Save File` dialog box, see Figure 1-33.

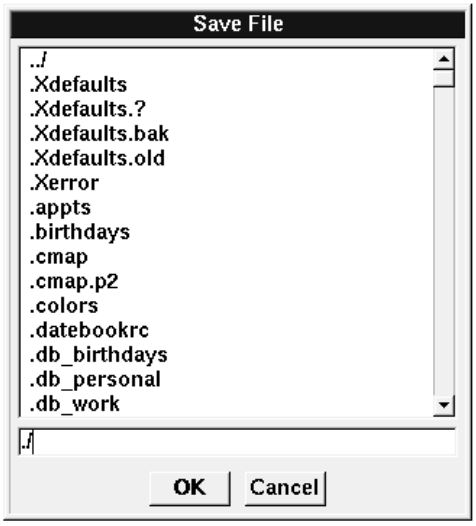


Figure 1-33. The Save File Dialog Box (UNIX)

By default, VEE for HP-UX saves files to the directory from which you started VEE. To save the current program, type in a name (for example, `simple-program.vee`) and click `OK`.

If you would like to save the file in another directory, use the **Backspace** key to delete the characters `./`, then type the file name with the complete path and click `OK`. In HP-UX, you need to add the `.vee` extension.

Tip: A handy way to replace a typed entry in a dialog box is to click and drag the mouse pointer over the entry to highlight it. Or you can highlight the entry by double-clicking the input field. Then you can type the correction and click `OK`.

Note

To re-save the program to the same file name, click the `Save` button or press **Ctrl**+**S** at any time (`File` ⇒ `Save`). It is a good idea to save files frequently while you are developing a program. To save a program that you have edited to a different file name, press **Ctrl**+**W** or `File` ⇒ `Save As`.

Exiting (Quitting) Agilent VEE

12. `Select File` ⇒ `Exit` to close the VEE application window.

Shortcut: Press **Ctrl-E** to exit VEE, or click on the x button at the right end of the title bar.

You will probably not need to use the following techniques, but if VEE stops responding to the mouse or keyboard, follow these instructions:

In Windows 95 and Windows 98	Press **Ctrl-Alt-Delete** and a window is displayed with various options. Follow the instructions in the window for MS Windows, or click `End Task`.
In Windows NT 4.0 and Windows 2000	Press **Ctrl-Alt-Delete** and click the `Task Manager` button. Select `VEE` in the Applications list and click `End Task`.
In HP-UX	You need to "kill" the process (that is the UNIX terminology). 1. Enter `ps -ef | grep vee` in HP-UX at a prompt to identify the process identification number. You will see a line with `veetest` on the end. The number following your login is the process identification number to enter. For example, it could read `johnj` *number* . . . `veetest`. 2. Enter `kill -9` *number* to stop the VEE application. Then you can enter `veetest` to start over again.

Re-Starting Agilent VEE and Running a Program

1. PC: In Windows, click `Start` ⇒ `Programs` ⇒ `Agilent VEE Pro 6.0`.

   HP-UX: From your home directory, enter `veetest`. (The executable is linked at install time to `/usr/bin/veetest`, which should be in your PATH. If it is not, you may have to change to that directory. If VEE has been installed in another directory, you may have to type the complete path.)

2. `Select File` ⇒ `Open` and complete the `Open File` dialog box.

   The format is the same as for the `Save File` dialog box. Note that in VEE for Windows, the default directory for user programs is the `VEE_USER` directory, unless you specified something else during installation. VEE opens the program in the Main window.

3. Click the **Run** button. It looks like a small arrowhead, and is located on the tool bar below the `Debug` menu as shown in Figure 1-34.

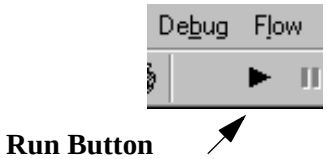


Figure 1-34. The Run button on the Tool Bar

Note

PC: The command `vee.exe -r` *filename* starts VEE in Windows and automatically run the program specified by *filename*. For example, you could create an icon on the Windows desktop and set its `Properties` ⇒ `Shortcut` to run a particular VEE program. An operator could then double-click an icon on the desktop to start VEE and run a program automatically. For more information, refer to the Windows Help information about commands and prompt paths.

HP-UX: The command `veetest -r` *filename* will start VEE and automatically run the program specified by *filename*. (If the VEE directory is not in your path, you need to enter the complete path, typically `/usr/bin/veetest -r` *filename*.)

Managing Multiple Windows in the Workspace

Most of the discussion so far has focused on the work area in the Main window. However, large VEE programs can contain multiple windows inside of the Main window. For example, a program may contain objects that you define, such as a `UserObjects` and `UserFunctions`. (You can think of `UserObjects` and `UserFunctions` as subroutines or subprograms to the main program. `UserObjects` and `UserFunctions` are discussed in more detail in the section “Creating a UserObject” on page 78 in Chapter 2, “Agilent VEE Programming Techniques.”) They are mentioned here to show how VEE helps you manage programs that have multiple windows.

Figure 1-35 shows a program with four windows. Each window has an icon (which provides menu commands), a title, and three buttons; minimize, maximize, and close. Maximizing a window makes it occupy the available area in the VEE workspace. Minimizing a window makes its icon appear along the bottom of the VEE workspace. Closing a window removes it from the workspace. VEE highlights the working window title bar.

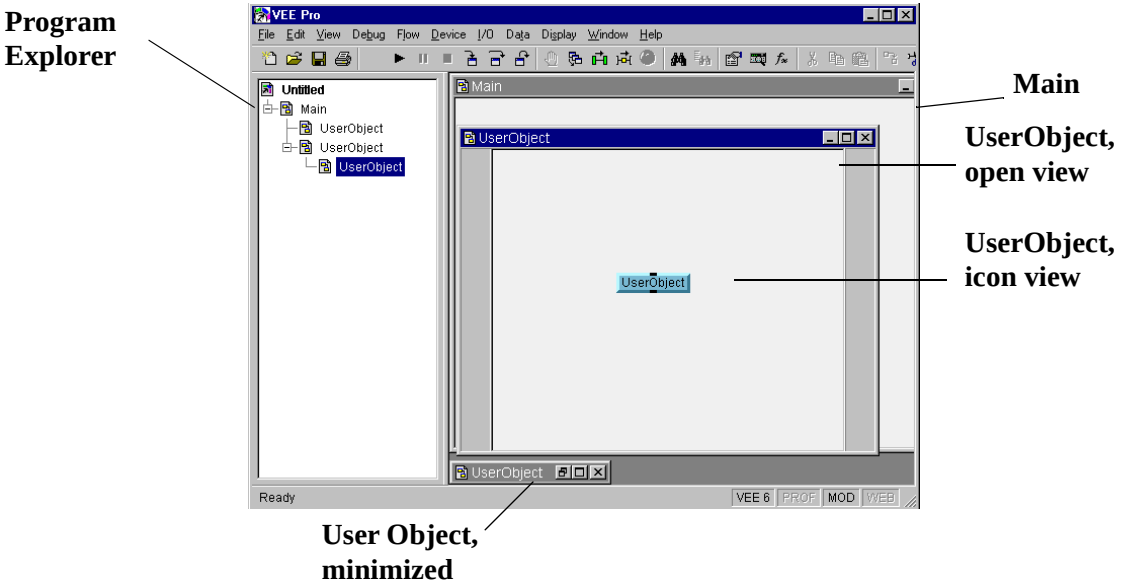


Figure 1-35. Multiple windows in the Work Area

As Figure 1-35 shows, the Program Explorer lists the hierarchy of the program. This built-in modular structure allows easy access to all parts of the program.

If the Program Explorer is not displayed, click `View` ⇒ `Program Explorer.` The default is for Program Explorer to appear. If you remove the check and click the `Save` button in `File` ⇒ `Default Preferences`, the Program Explorer will not appear the next time you start VEE.

To bring the Main window forward at any time, click on it or double-click its icon in the Program Explorer.

Note

If you close the Main window in VEE, you can display the Main window again by selecting `View` ⇒ `Main`.

How Agilent VEE Programs Work

In VEE, the general flow of execution through a program is called **propagation**. Propagation through a program is *not* determined by the geographic locations of the objects in the program, but rather by the way the objects are connected. Propagation is primarily determined by **data flow**, which, in turn, is determined by how the data input and output pins of the objects are connected.

Note

In other programming languages such as C, BASIC, or Pascal, the order in which program statements execute is determined by a set of sequence and selection rules. Generally, statements execute in their order of appearance in the program unless certain statements cause execution to branch to another statement or thread of code.

The rules of data flow in a VEE program are as follows:

- *Data flows from left to right through an object*. This means that on all objects with data pins, the left data pins are inputs and the right data pins are outputs.
- *All of the data input pins in an object must be connected*. Otherwise, an error occurs when the program runs.
- *An object will not execute until all of its data input pins have received new data*.
- *An object finishes executing only after all connected and appropriate data output pins have been activated*.

In VEE, you can change the order of execution by using sequence input and output pins. However, you do not normally need to use sequence pins except for special cases. *It is generally best to avoid using the sequence pins. If possible, let data flow control the execution of the program.*

Lab 1-2: Viewing Data Flow and Propagation

To see how data flow works, open the program you created earlier. Open the program `simple-program.vee` by clicking the `Open` button on the toolbar. (The program `simple-program.vee` is described in the section called “Display Waveform Program” on page 52.) Now run the program. It should appear as shown in Figure 1-36, although you may have different values for parameters.

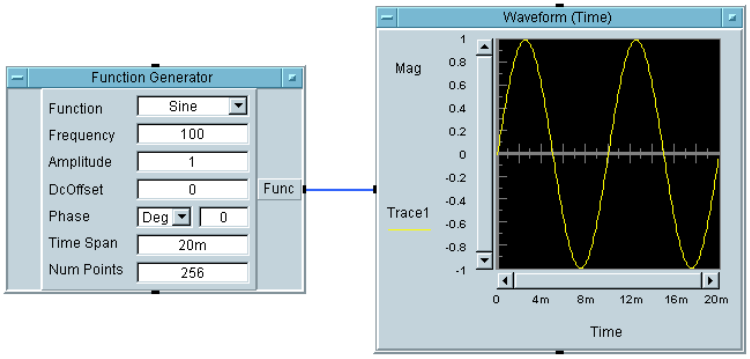


Figure 1-36. Typical simple-program.vee Display

The data output pin of the `Function Generator` object is connected to the data input pin of the `Waveform (Time)` object. When you run the program, the `Waveform (Time)` object will not execute until it receives data from the `Function Generator` object. This is a simple example of data flow.

Lab 1-3: Adding a Noise Generator

Add a “noisy sine wave” by adding a `Noise Generator` object to `simple-program.vee`, as shown in Figure 1-37.

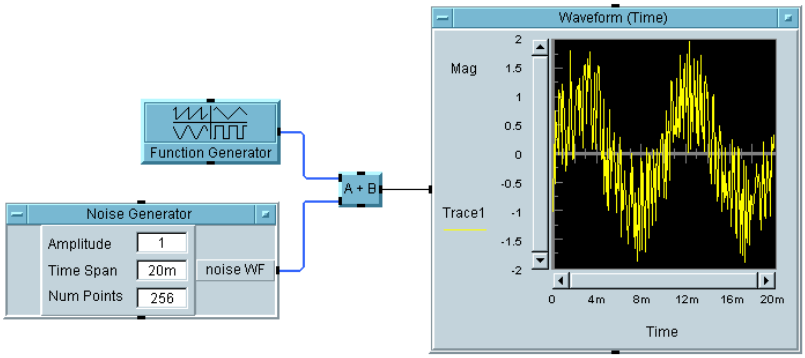


Figure 1-37. Example: Adding a Noise Generator Object

Note

The VEE programs for many of the lab exercises and programming examples in this manual are included in VEE, under `Help` ⇒ `Open Example...` ⇒ `Manual` ⇒ `UsersGuide`.

1. Delete the line connecting the `Function Generator` and `Waveform (Time)` objects in the original program. Click the **Delete Line** button on the toolbar and then click the line. Or, press and hold **Shift**+**Ctrl** and click the line.

2. Minimize the `Function Generator` to its icon.

3. Add the `Noise Generator` object (`Device` ⇒ `Virtual Source` ⇒ `Noise Generator`).

4. Add the `A+B` object, using `Device` ⇒ `Function & Object Browser`.

   The `Function & Object Browser` is shown in Figure 1-38. For `Type`, select `Operators`. For `Category`, select `Arithmetic`. For `Operators`, select +.) Click `Create Formula` and place the object in the work area between the `Function Generator` and the `Waveform (Time)` object. Minimize the `A+B` object.

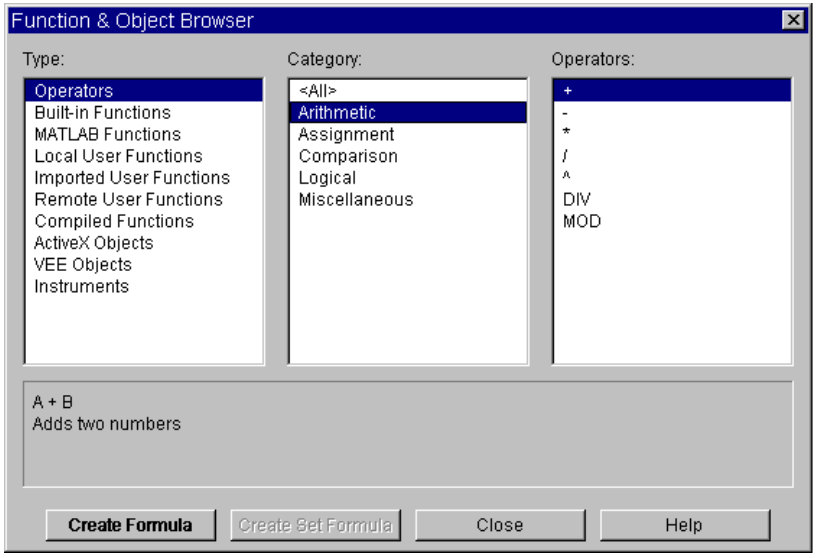


Figure 1-38. Function and Object Browser

5. Connect the input and output pins as shown in Figure 1-37.

6. Run the program.

   Notice that the `A+B` object does not execute until the `Function Generator` and the `Noise Generator` objects execute. However, it does not matter whether the `Function Generator` or the `Noise Generator` executes first, because the result is the same.

   Once both of the `A+B` input data pins receive data, the `A+B` object executes, summing the two signals and outputting the result to the `Waveform (Time)` object.

Note The data flow in a VEE program determines its execution.

To see the order of execution, turn on the `Debug` commands `Show Execution Flow` and `Show Data Flow`, or click their respective buttons on the toolbar. Run the program again. Each object highlights when it executes and a small, square marker moves down the lines to show data flow.

Note

`Show Execution Flow` and `Show Data Flow` can be enabled together or individually by clicking their toolbar buttons or their commands in the `Debug` menu. Normally, you should turn these commands off because they slow down the program.

Lab 1-4: Adding an Amplitude Input and Real64 Slider

Add an amplitude input and a Real64 slider to `simple-program.vee`.

1. Click on the object menu or press **Ctrl**+**A** with the mouse pointer in the "terminal area" at the left side of the `Noise Generator`. The dialog box appears for you to add an input, as shown in Figure 1-39.

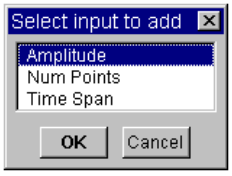


Figure 1-39. Example: Adding Input Terminals

2. Select `Amplitude` by clicking `OK`—an `Amplitude` input terminal appears.

   Now that the `Noise Generator` object has an amplitude input pin, you can input this data as a real number. VEE provides an object that makes this easy, called a `Real64 Slider`, which is located in the `Data` menu. (You could also use the `Real64 Constant` object or a `Real64 Knob`.)

3. Add a `Real64 Slider` object (`Data` ⇒ `Continuous` ⇒ `Real64 Slider`) and connect its data output pin to the `Amplitude` terminal, as shown in Figure 1-40. Run the program.

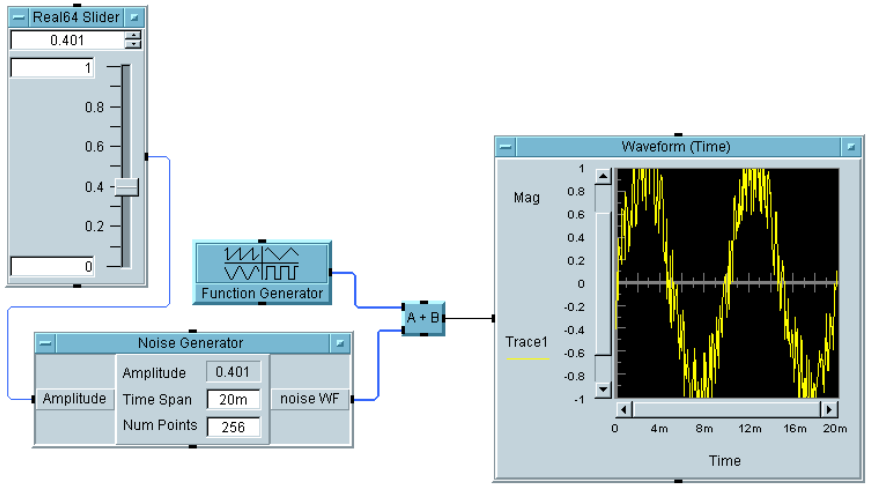


Figure 1-40. Example: Adding a Real64 Slider Object

Try changing the amplitude of the noise, by dragging the slide control on the `Real64 Slider` object. The amplitude of the noise does not change until you run the program. The noise component of the displayed waveform depends on the `Real64 Slider` output value.

Again, data flow determines the order of execution. The `Noise Generator` cannot execute until the `Real64 Slider` executes. The `A+B` object cannot execute until both the `Function Generator` and the `Noise Generator` execute, but it does not matter which one executes first. Finally, the `Waveform (Time)` object executes only after the `A+B` object has executed.

Note

You can display the value of an output by using the mouse to hover over the line. For example, hovering over the line from the `Real64 Slider` object to the `Noise Generator` displays a value of `0.401`. Notice that the value on the line (`0.401`) matches the value shown on the `Real64 Slider`, as shown in Figure 1-41. (Note that the objects are shown in iconized view.)

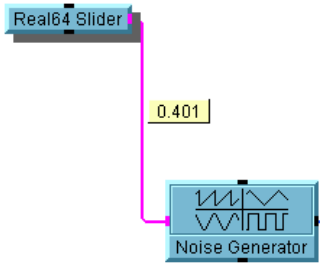


Figure 1-41. Displaying the Value on an Output Pin

4. Re-save the program to `simple-program.vee`. You will add some more features to it in the next chapter.

Chapter Checklist

You should now be able to do any of the following tasks. Review topics as needed, before going on to the next chapter.

- Look up on-line help documentation from the main menu bar and from the object menus.
- Start VEE.
- Identify the main menu bar, toolbar buttons, work area, and status bar.
- Explain the Program Explorer and its purpose.
- Select menu items from the main menu and object menus.
- Perform the following operations on an object: moving, renaming, iconizing, expanding, sizing, selecting, deselecting, deleting, cloning, etc.
- Move the work area, clear the work area, and manage multiple windows.
- Identify data and sequence pins on an object and explain their purpose.
- Examine terminals and change their names.
- Connect objects to create a program to simulate waveform data.
- Create, run, print and save a program.
- Exit VEE, and then reopen a program.
- Explain how data flows through a VEE program.

2

Agilent VEE Programming Techniques

Agilent VEE Programming Techniques

In this chapter you will learn about:

- Creating a `UserObject`
- Adding a dialog box for user input
- Using data files
- Creating panel views (an operator interface)
- Mathematically processing data
- Communicating with instruments
- Documenting a program
- Using debugging tools

Average time to complete: 2 hours

Overview

In this chapter, you will learn selected VEE programming techniques to help you build your own programs. For example, VEE allows you to create customized objects called **UserObjects**. You can also create interfaces for operators to use that show only the necessary parts of the program. These are displayed in the **Panel view** of the program.

You can write data from VEE to a file, and read data from a file into VEE. Data files and their associated I/O transactions can be used for many purposes, including communicating with instruments, files, strings, the operating system, interfaces, other programs, Rocky Mountain Basic, and printers.

VEE supports many data types and provides extensive mathematical processing capabilities. There are multiple ways for you to use VEE to communicate with instruments. VEE also provides powerful debugging tools to debug any problems in programs.

General Techniques

Inside the Main VEE program, you can create logical groups of objects, called `UserObjects`. A `UserObject` object (called `UserObject` hereafter) is created by placing a logical group of objects in a `UserObject` window. Inside the `UserObject` window, you connect inputs and outputs in the same way as the main program. The `UserObject` itself is connected to other objects in the main program with inputs and outputs, like any other object.

The idea in developing a `UserObject` is to create a unique context that performs a useful purpose within the main program. Besides conserving space in the main work area, you can make the program more understandable by giving it structure.

A VEE program can contain many `UserObjects` nested within the Main program. Each `UserObject` has an icon view which resides in the Main window. To associate the icon views of the `UserObjects` in the main program with their associated `UserObject` windows, name `UserObjects` in their edit windows, which also names them in their associated icon view. For example, if you name a `UserObject AddNoise`, its icon window in the Main program and the title bar on the `UserObject` will both read `AddNoise`. The following exercise teaches you how to create a `UserObject`.

Lab 2-1: Creating a UserObject

There are a couple of ways to create a `UserObject` in a VEE program:

- Select `Device` ⇒ `UserObject` from the menu bar to bring up an empty `UserObject` icon in the `Main` window, and add objects to it. If you double-click the `UserObject` icon, it is displayed in open view, as shown in Figure 2-1.

- Select objects within a program and then create a `UserObject` from them, by selecting the objects and clicking `Edit` ⇒ `Create UserObject`.

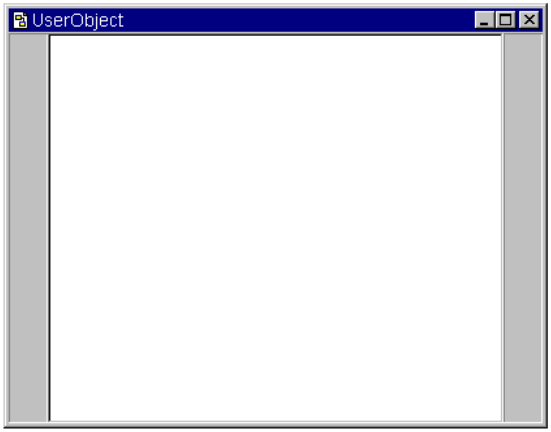


Figure 2-1. UserObject Window

Once you have created a `UserObject`, it is part of the main program. The `UserObject` window can be displayed as an icon, in open view, or minimized at the bottom of the screen as follows:

- Close the window by clicking the close button, and the `UserObject` is displayed as an icon in the main window.
- Maximize the window by clicking its maximize button, and the `UserObject` window will occupy the entire available area in the VEE workspace.
- Minimize the window by clicking its minimize button. The minimized `UserObject` is displayed along the bottom of the VEE workspace.

Note The icon view of the `UserObject` always resides in the Main window, and you can connect its pins to other objects in the Main window.

Note Before you begin, make sure `Program Explorer` in the `View` menu is deselected to give yourself more screen space in Main.

Now, you will create a `UserObject` for a program.

1. Open the program (`simple-program.vee`) you created in "Adding an Amplitude Input and Real64 Slider" on page 71. The program should appear in the main work area.

2. Remove the `Real64 Slider` from the program. (It is not used in this exercise.) Click to open the `Real64 Slider` Object Menu, and select `Cut`, or double-click on the `Real64 Slider` Object Menu button.

Note

If you run the program again now, with the `Real64 Slider` object removed and the input pin still on the `Noise Generator`, you will get a VEE error message that the input pin `Amplitude` on the `Noise Generator` is not connected. *Remember, all input pins must be connected for a VEE program to run*.

3. In the `Noise Generator` object, click the **Object Menu** button or click the right button over the object to open the object menu. Select `Delete Terminal` ⇒ `Input`, and in the dialog box for `Choose an input to delete` with `Amplitude` highlighted, click `OK`.

4. Rename the program by choosing `File` ⇒ `Save As...` and type in the new name `usrobj-program1.vee`.

5. Then, minimize the `Noise Generator` object and rearrange the objects as shown Figure 2-2.

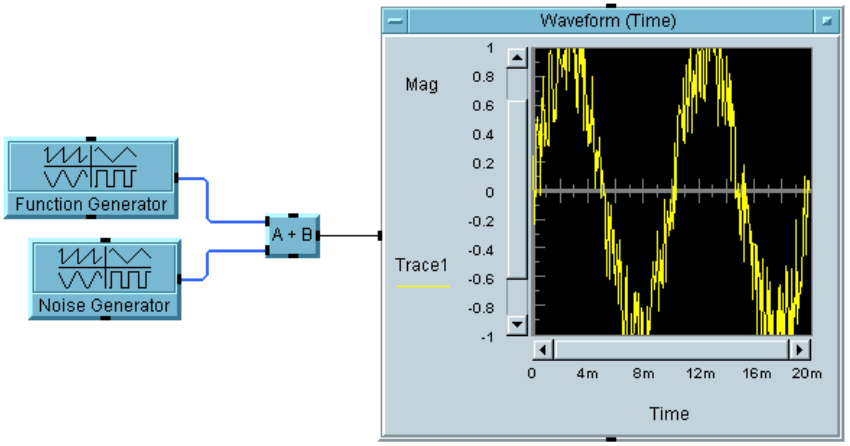


Figure 2-2. usrobj-program.vee at an Early Stage

6. Select the `Noise Generator` and `A+B` objects, using the shortcut **Ctrl**+*left mouse button*. Click `Edit` ⇒ `Create UserObject`. A dialog box appears labeled `Create UserObject`. (You could rename the object by typing in a new name if you wish. For now, click `OK` to create the `UserObject`.)

   The `UserObject` will contain the `Noise Generator` and `A+B` objects in the `UserObject` edit window, and will be automatically created in the `Main` window with the appropriate input and output pins and connections as shown in Figure 2-3.

   Tip: Position the icons in the upper left of the `UserObject` by simply pressing the **Home** button on the keyboard.

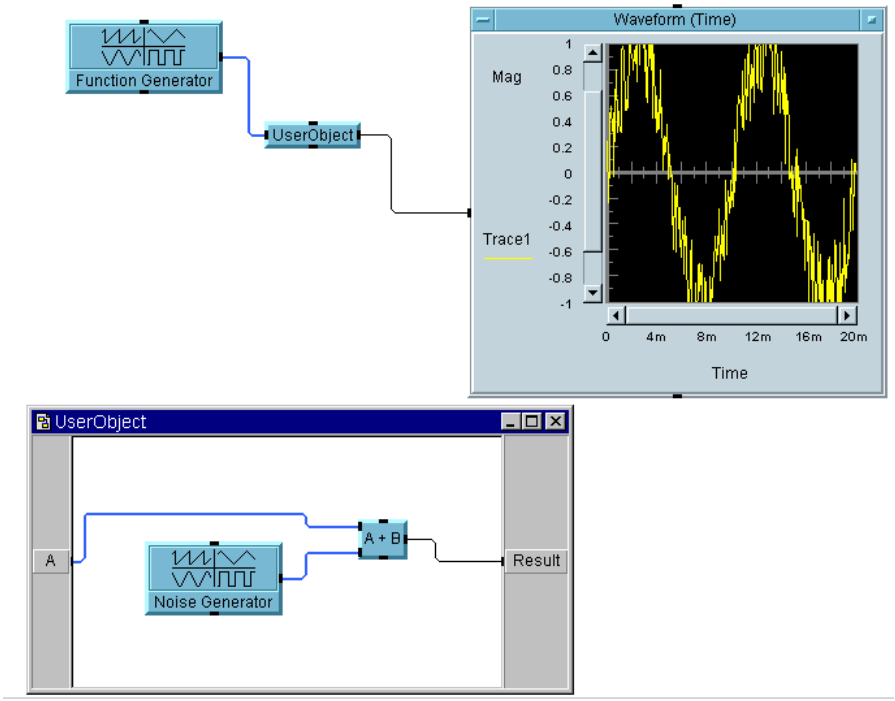


Figure 2-3. Creating a UserObject

Note

Rearranging the positions of the objects before executing `Create UserObject` is one of convenience. If you do not collect the objects to be included into one area, the `UserObject` will size itself to encompass all the selected objects. You can then rearrange and resize the work area of the `UserObject` and move the `UserObject` to an appropriate place in the work area. However, the cleanup is easier if you place the objects logically beforehand.

Note

You can use `Edit` ⇒ `Clean Up Lines` to clean up the line routing within a program. This command is context dependent. To clean up the lines for the `UserObject`, it must be the active window. Click the `UserObject` window, then, use `Edit` ⇒ `Clean Up Lines`.

Tip: Creating a `UserObject` in its edit window and then using the icon view of the `UserObject` lets you save screen space.

7. To help you keep track of the `UserObject`, change the title from `UserObject` to `AddNoise`. Double-click the title bar and enter the new title in the properties dialog box. Figure 2-4 shows how this makes the program easier to follow.

   Tip: To get to any object's Properties dialog box quickly, just double-click its title bar.

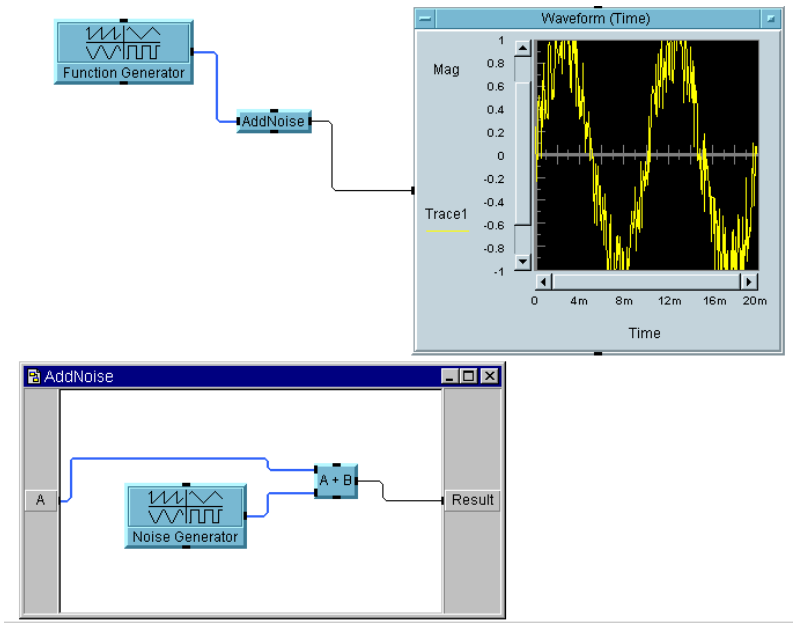


Figure 2-4. UserObject Renamed AddNoise

8. Click the **Run** button to display the noisy cosine wave as shown in Figure 2-5. Note that `AddNoise` is minimized, and appears in icon form at the bottom of the work space. To minimize `AddNoise`, click on the minimize button in its title bar, shown as the underline symbol (_).

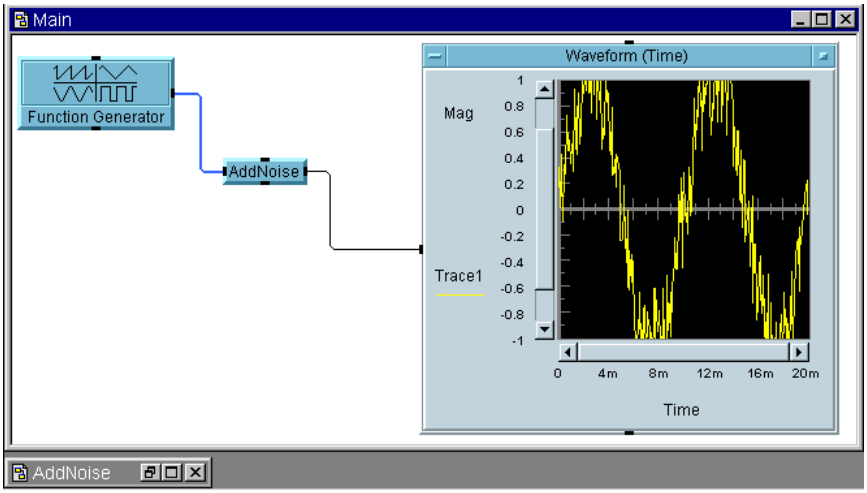


Figure 2-5. Noisy Cosine Wave

The key to effective `UserObjects` is to make sure they serve a logical purpose within the program. This unique object is not just a space saving device, but rather a way of structuring a program. `UserObjects` help you use "top-down" design in VEE programs. VEE also includes an object called a `UserFunction`, which is a re-usable code module. For more information about `UserObjects` and `UserFunctions`, refer to Chapter 8, "Using Agilent VEE Functions," on page 293.

For more information about `UserObjects`, select `Help` ⇒ `Contents and Index` from the VEE menu bar. Then, browse `How Do I...`, `Tell Me About...`, or `Reference`.

You will continue with this example in the following section. However, if you want to quit now, save the program as `usrobj-program3.vee`.

Lab 2-2: Creating a Dialog Box for User Input

If it is not already open, open the program `usrobj-program3.vee`.

In the `Data` ⇒ `Dialog Box` submenu are six choices for dialog: `Text Input`, `Int32 Input`, and `Real64 Input`, as well as `Message Box`, `List Box`, and `File Name Selection` boxes. In each case for text, integer, and real input, a dialog box helps you configure the prompt or label, default value, value constraints, and error message. Once you include one of these dialog boxes, a pop-up input box will appear when the program is run.

1. Select `Data` ⇒ `Dialog Box` ⇒ `Int32 Input` and place it to the left of the `Function Generator`. Change the `Prompt/Label` field to `Enter Frequency:`. (Remember to click and drag over the field to highlight it first.) Change the `Default Value` to `100`.

   Tip: You can also double-click an input field to highlight an entry.

2. Change the `Value Constraints` to `1` on the low end and to `193` on the high end. Change the error message to reflect these new values, as shown in Figure 2-6. Finally, iconize the `Int32 Input` object.

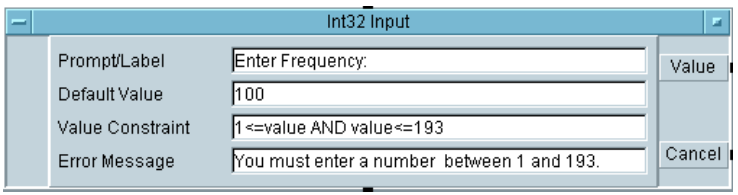


Figure 2-6. The Int32 Input Configuration Box

3. Open the `Object Menu` for the `Function Generator`, and choose `Add Terminal` ⇒ `Data Input`. In the dialog box for `Select input to add`, choose `Frequency` and click `OK`.

4. Connect the top output pin of the `Int32 Input` object to the input pin on the `Function Generator`. Notice that `Frequency` can only be changed through the input pin now, and you can no longer edit the `Frequency` input field. The program should look like Figure 2-7.

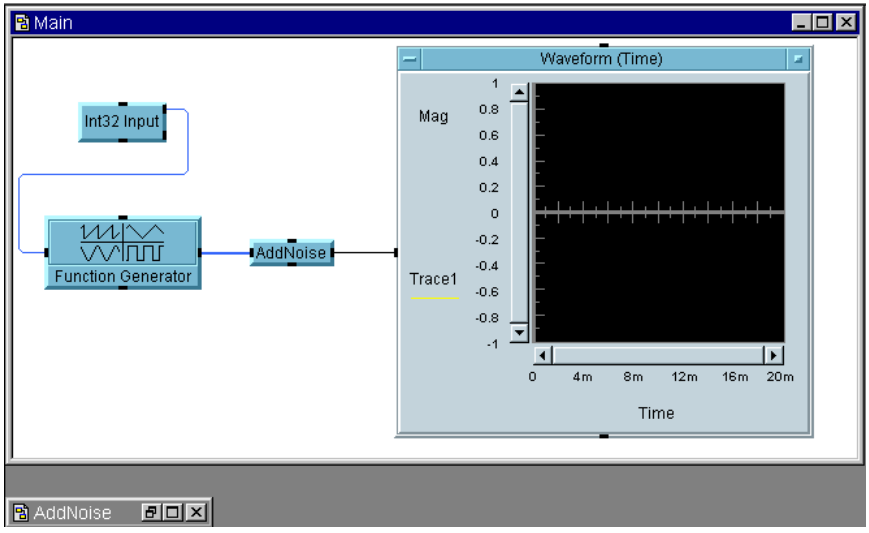


Figure 2-7. Int32 Input Added to usrobj-program.vee

5. Run the program. The input box for `Int32 Input` appears, with the instruction `Enter Frequency:`. Try running the program with different frequencies in the input box. See Figure 2-8, shown at run-time with the pop-up input box. Simply click and drag the pop-up box to control where it appears.

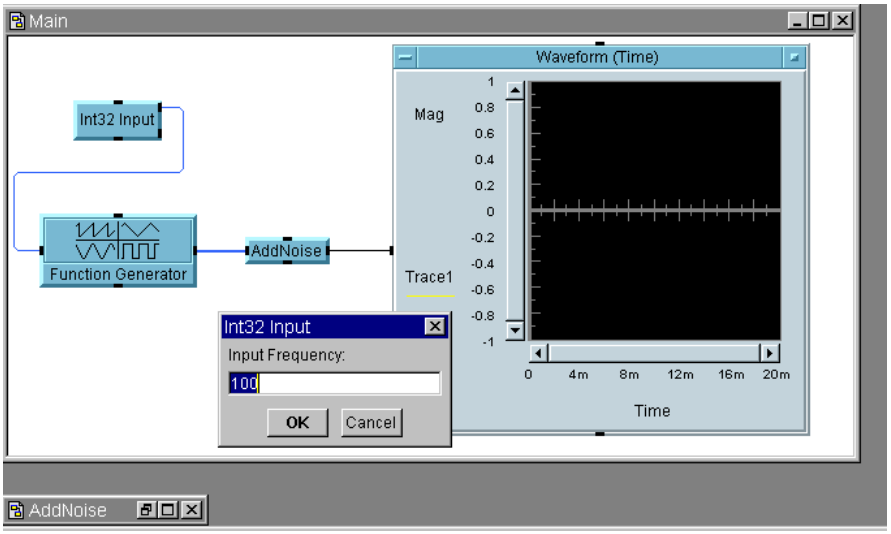


Figure 2-8. Runtime Pop-Up Input Box

You will get an error message box if you enter frequencies above 193. Notice that you get the exact error message that you configured.

You will continue with this example in the following section. However, if you want to quit now, save the program as `usrobj1-program4.vee`.

Note

The VEE programs for many of the lab exercises and programming examples in this manual are included in VEE, under `Help` ⇒ `Open Example...` ⇒ `Manual` ⇒ `UsersGuide`.

Lab 2-3: Using Data Files

You can write data from VEE to a data file and read the data in a file into VEE by including the `To File` and `From File` objects in the program. For example, add a `To File` object to the detail view of the program that you have been building.

If it is not already open, open the program `usrobj-program4.vee`.

1. Select `I/O` ⇒ `To` ⇒ `File` and place it in the Main work area.

2. Change the default filename, `myFile`, to `wavedata`.

   If there is no check mark to the left of `Clear File At PreRun & Open`, then click on the small input box. `To File` defaults to appending data to the existing file. In this case, however, you want to clear the file each time you run the program. The `To File` object should now look like Figure 2-9.

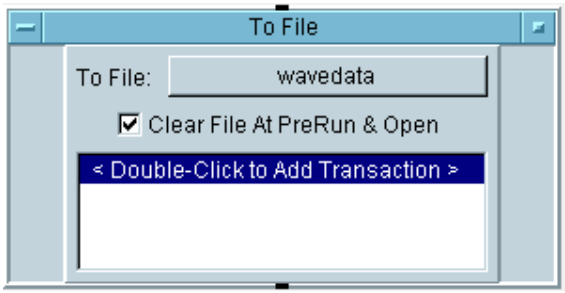


Figure 2-9. Adding a Data File

3. Double-click on the area labeled `Double-Click to Add Transaction` to write the data. The dialog box in Figure 2-10 appears. Click the `TEXT` field (or its arrow) to show the drop-down list of data types and click `CONTAINER`. Click `OK`. Notice that when you click `OK` in the `I/O Transaction` dialog, an input pin `a` is automatically added to the `To File` object.

   Examine `Help` in the `To File` object menu to see the other options for the transaction besides `WRITE CONTAINER`. Transactions are discussed in more detail in an appendix in the *VEE Pro Advanced Techniques* manual and in Chapter 5, “Storing and Retrieving Test Results.”

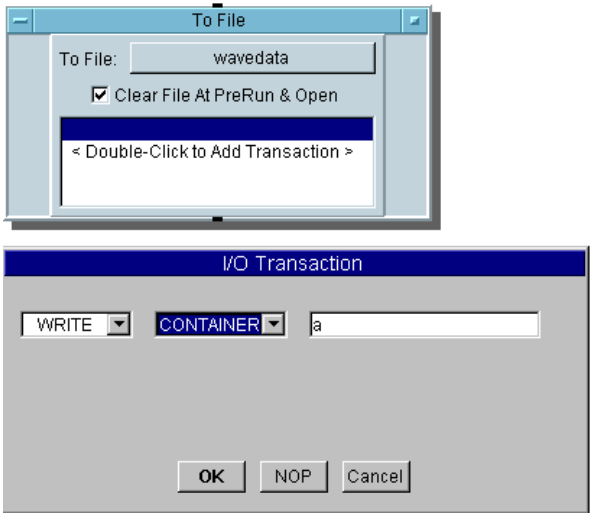


Figure 2-10. Choosing an I/O Transaction

4. Connect the data output pin of the `AddNoise UserObject` to the data input pin of `To File`. The program should now look like Figure 2-11.

Note You can connect one data output pin to several data input pins.

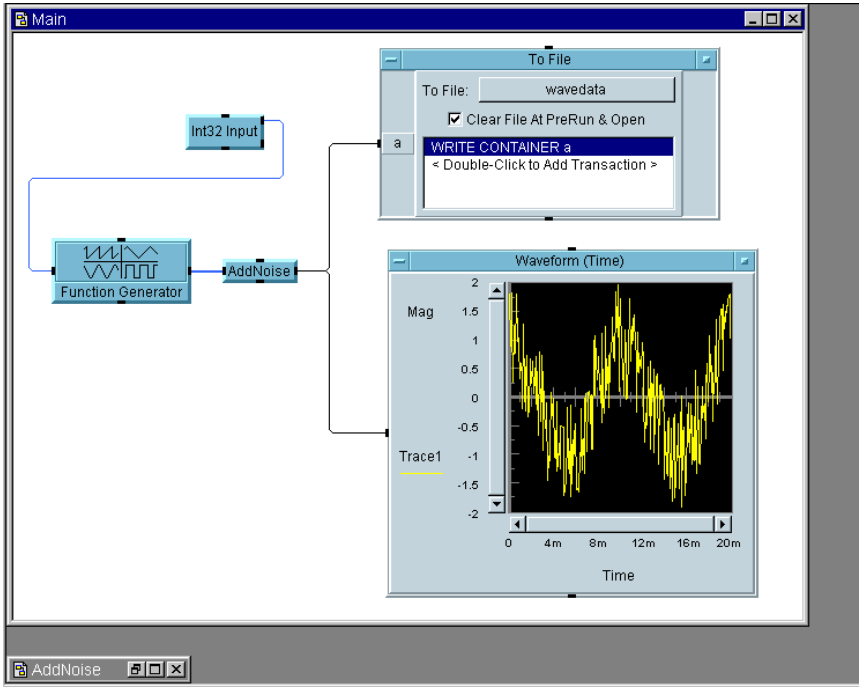


Figure 2-11. Adding a To File Object

5. Click the **Run** button on the tool bar again to test the program. The program now displays the noisy cosine wave output by the `AddNoise` UserObject and writes a container of waveform data to the file `wavedata`.

   Double-click the `To File` object to get the open view, then double-click the input terminal `a` to examine its contents. You should see an array of 256 points.

Add a `From File` object to the program to read the data back.

6. Select `I/O` ⇒ `From` ⇒ `File` and place it in the Main work area. Add a read transaction to `READ CONTAINER x` and change the file name to `wavedata` (the procedure is the same as for `To File`). Then, delete the line between `AddNoise` and the `Waveform (Time)` object, and connect

the objects as shown in Figure 2-12. The sequence line between `To File` and `From File` ensures the data is written to the file before it is read.

7. Run the program. It should similar to Figure 2-12. Save the program as `usrobj-program.vee`.

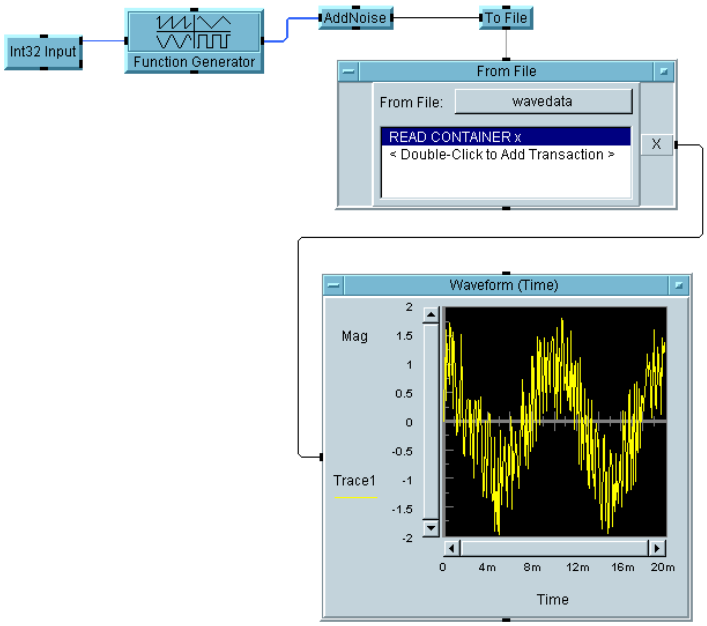


Figure 2-12. Adding a From File Object

Lab 2-4: Creating a Panel View (Operator Interface)

After you develop a program, you may want to create an operator interface. To do so, create a **panel view** of the program. This exercise uses the program you created in “Viewing Data Flow and Propagation” on page 68.

1. Open the program `simple-program.vee`. The program should look like Figure 2-13.

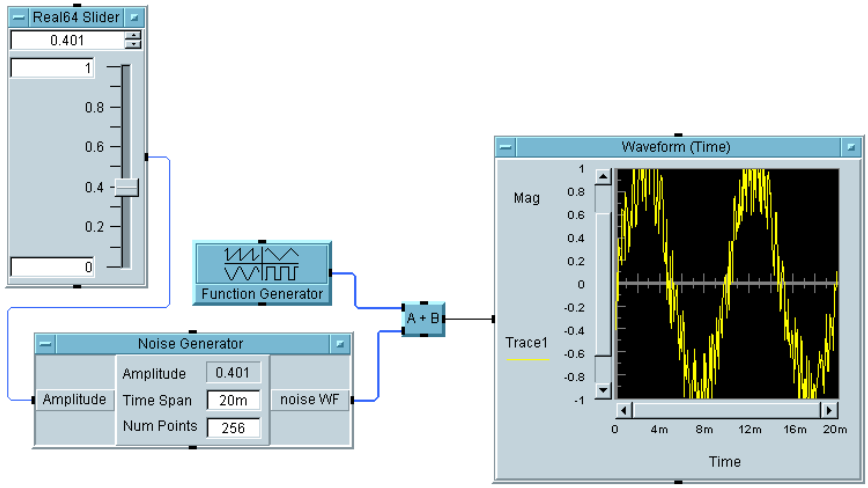


Figure 2-13. simple-program.vee

2. Select the objects that you want to appear in the `panel` view, which acts as the operator interface. Press and hold **Ctrl** while clicking on all the objects you want to select. (Make sure no object is accidentally selected.) In this case, select the `Real64 Slider` and `Waveform (Time)` objects. They will each now have a shadow to indicate they are selected.

3. Click the `Add to Panel` button on the toolbar to add the selected objects to the panel (or use `Edit` ⇒ `Add To Panel`). A panel view appears, showing the two objects that you added to the panel.

   You can size and move the objects in the panel view to appropriate locations to create a panel similar to the one shown Figure 2-14.

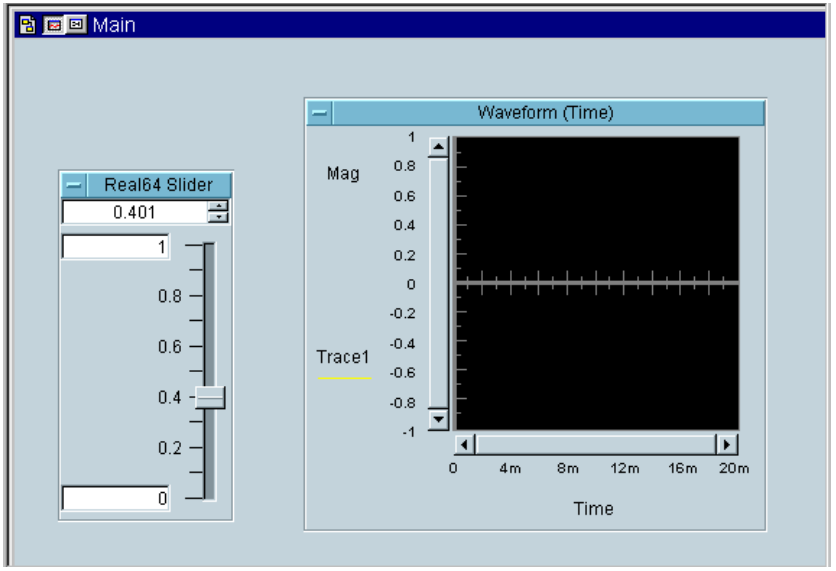


Figure 2-14. Example: Creating a Panel View

4. Press the **To Detail** button in the upper left `Main` window title bar to go to the detail view. Click the **To Panel** button to return to the panel view.

   The detail view is the normal window in which you edit a program. You can move, resize, or delete objects in the panel view independently from the detail view. The detail view is used to develop a program and the panel view is used to provide an operator interface.

5. Save the program as `simple-program_with_panel.vee`.

You can practice making some changes to the panel view as follows:

- To change colors on the panel, select `Properties` from the Main window object menu in panel view. Then choose `Colors`, click the `Panel View` ⇒ `Background:` button, and select the color you want.

- To change colors or fonts on any object, just double-click its title bar to get the `Properties` box. Then click either the `Colors` or `Fonts` tab and make the changes you want.

- To give a raised appearance to objects in the Panel view, open the `Properties` box for that object, open the `Appearance` folder by clicking on its tab, and select `Raised` under `Border`.
- To change the name of the Panel view, open the main `Properties` dialog box and name the panel view whatever you wish. The name you enter will be displayed when the program executes.

Lab 2-5: Mathematically Processing Data

VEE provides extensive built-in mathematical capabilities and data type support, as well as all the data and signal processing power of MATLAB. For more details, refer to the *VEE Pro Advanced Techniques* manual.

Using Data Types

VEE supports several data types, including text, integer and real numbers, and several types of complex and coordinate numbers. You have already seen how the `A+B` object can add two waveforms together in earlier examples. Mathematical operators, such as addition (`+`), can act on several data types and can even act on mixed data types.

For example, to create the following program clear the `Main` window, place the following objects in the `Main` window, and connect them as shown, noting the following information.

1. Select `File` ⇒ `New` to clear the work area.
2. Add a `Real64 Constant` object by selecting `Data` ⇒ `Constant` ⇒ `Real64`.
3. Add a `Complex Constant` object by selecting `Data` ⇒ `Constant` ⇒ `Complex`.
4. Add an `A+B` object. Select `Device` ⇒ `Function & Object Browser` to get the `Function & Object Browser`. Then, select `Type: Operators`; `Category: Arithmetic`; `Operators: +`. Click `Create Formula` to create the object.

5. Add an `AlphaNumeric` object by selecting `Display` ⇒ `AlphaNumeric`. Connect the objects as shown in Figure 2-15. Type in the value `1.53` in the data entry field of the `Real64 Constant` object and the complex value `(2,1)` in the `Complex` object. Run the program and you should get the result shown in Figure 2-15.

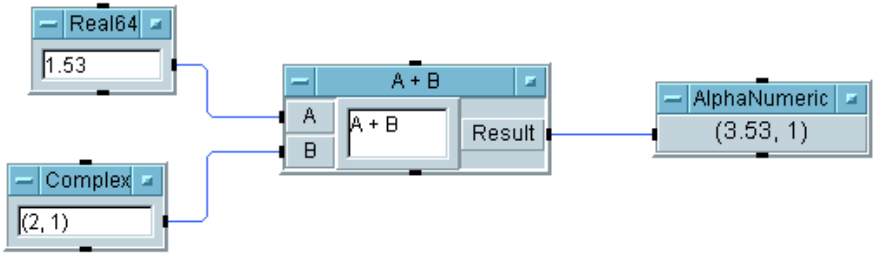


Figure 2-15. Using Data Types

VEE automatically converts the data as needed and then performs the addition in the `A+B` object. The real value `1.53` is converted to the complex value `(1.53,0)`, which is then added to the complex value `(2,1)`. The result, `(3.53,1)` (a complex number), is displayed in the `AlphaNumeric` object.

Note

Normally, VEE automatically handles all data type conversions. For more information, select `Help` ⇒ `Contents and Index` from the VEE menu bar. Then, browse `How Do I...`, `Tell Me About...`, or `Reference`.

Using Data Shapes

VEE supports a variety of data shapes, such as scalars and arrays. Unlike most programming languages, VEE objects can operate on an entire array, rather than on only one element.

The following program creates a one-dimensional, ten-element array, calculates the median of the 10 values, and then displays the median value.

1. Select `File` ⇒ `New` to clear the work area.

2. Add a `For Range` object, by selecting `Flow` ⇒ `Repeat` ⇒ `For Range`.

3. Add a `Sliding Collector` object, by selecting `Data` ⇒ `Sliding Collector`.

4. Add a `median(x)` object. Select `Device` ⇒ `Function & Object Browser`. Then, select `Type: Built-in Functions`; `Category: Probability & Statistics`; `Functions: median` and click `Create Formula`.

Shortcut: You can display the `Function & Object Browser` by clicking the ***fx*** button on the toolbar.

5. Add an `AlphaNumeric` object, by selecting `Display` ⇒ `AlphaNumeric`. Connect the objects as shown in Figure 2-16. Run the program. If you have not changed any of the inputs on the objects, you should see the result displayed in Figure 2-16.

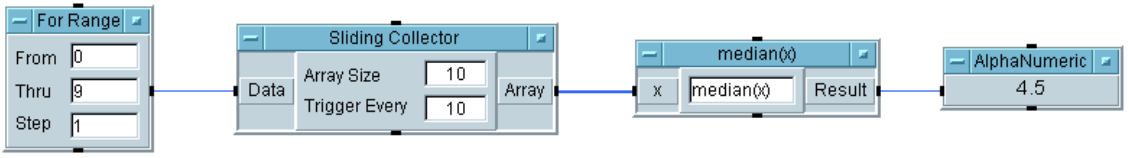


Figure 2-16. Connecting Data Objects

Using the Formula Object

VEE provides mathematical operators and functions which are documented in the `Reference` part of online help. Select `Help` ⇒ `Contents and Index`. Then, select `Reference` and browse the items as desired.

The predefined operator and function objects are available via `Device` ⇒ `Function & Object Browser` (or ***fx*** on the toolbar). You select them from the `Function & Object Browser` by clicking entities in three lists: `Type:`, `Category:`, and `Functions:`. Click `Create Formula` to create the object.

Besides using predefined operators and functions, you can create any valid VEE mathematical expression within the `Formula` object, which is found under the `Device` menu. In this section, you will create a program using a `Formula` object. To begin, clear the `Main` window and follow these steps.

1. Add the `Function Generator` object to the `Main` window and modify it to produce a `100` Hz sine wave. Select `Device` ⇒ `Virtual Source` ⇒ `Function Generator`.

2. Select `Device` ⇒ `Formula` to add the `Formula` object to the Main window. Add a second input (`B`) to the object by putting the mouse pointer in the input terminal area and clicking **Ctrl**+**A**.

3. Type the mathematical expression `abs(A)+B` in the entry field.

4. Select `Data` ⇒ `Constant` ⇒ `Real64` to add a `Real64 Constant` object to the `Main` window. Type in the value `0.5`.

5. Select `Display` ⇒ `Waveform (Time)` and set the `y-axis` scale to `-2` through `2`. Set `Automatic Scaling` to `Off`. To get the dialog box for these parameters, click `Mag`.

6. Connect the objects as shown in Figure 2-17. Run the program.

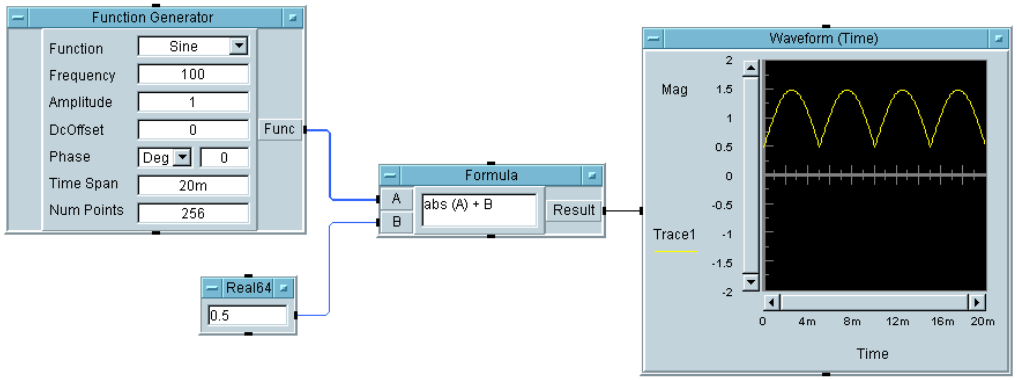


Figure 2-17. Creating a Formula Object Program

When you run the program, the `Formula` object takes the waveform input `A` and the real value `B`, and adds `B` to the absolute value of `A`. In effect, the expression `abs(A)+B` "rectifies" the sine wave and adds a "dc offset". You could have produced the same effect by using the `A+B` and `abs(x)` objects, but it is easier to read an expression in a `Formula` object. (This also saves space.)

Try double-clicking the input and output terminals of the `Formula` object. Note that the real scalar on input `B` is added to each element of the waveform data (a one-dimensional array) on input `A`, and the resulting waveform is output on the `Result` terminal.

Note

To augment VEE's extensive math capability, there are hundreds more mathematical functions available through MATLAB Script integration. Browse through these functions in the `Function & Object Browser`. For more information about using MATLAB functions, refer to "Using MATLAB Script in Agilent VEE" on page 187 of Chapter 4, "Analyzing and Displaying Test Data.".

Using Online Help

Now that you have created a few simple programs, here are some ways to teach yourself more about VEE.

1. First, run the Multimedia Tutorials located in the `Help` ⇒ `Welcome` menu. The tutorials demonstrate many of the main features of VEE. They will help bring you up to speed quickly. The tutorials display screen demonstrations of VEE programs being built and run, and describe what you are seeing. The tutorials also introduce key concepts for using VEE effectively.

2. Once you become familiar with VEE, look for more information in the `Help` entries in the object menus. You can experiment with the objects until you understand how they work. If you need to know more about an object, the object menus give you the most specific information. Consult them first.

3. To use the `Help` contents, index, or search capabilities, open `Help` on the main VEE menu bar.

Note

To review how to open the main `Help` facility and a listing of the `Help` contents, refer to “Getting Help” on page 25 of Chapter 1, “Using the Agilent VEE Development Environment.”

Using the Help Facility

Online `Help` provides information on the following topics:

- All menu items, as well as shortcuts for most of them
- Instrument driver information
- Frequently performed tasks and many example programs
- Definition of VEE terms
- Using the help facility
- VEE version

You can browse, use the keyword index, use hyperlinks to related topics, or even do a search. There are many `Help` features available in VEE that you can use as you develop programs.

Note

VEE also includes other helpful features for developing and debugging programs, such as line probe. For more information, refer to "Debugging Programs in Agilent VEE" on page 102.

Displaying Help about an Object

To get help on an object, click on the object menu button and select `Help`.

- Select `Flow` ⇒ `Repeat` ⇒ `For Count` to create a `For Count` object. Click on object menu and select `Help`. The `Help` topic appears describing `For Count`.
- Select `Device` ⇒ `Formula` to create a `Formula` object. Click on the object menu and select `Help`. The `Help` topic appears describing the particular formula displayed in the `Formula` object.
- Select `Device` ⇒ `Function & Object Browser`. Select any combination of choices and click on `Help`. The `Help` topic appears for the particular object that is selected.

Finding the Menu Location for an Object

To find the location for an object in the menus, and to display the information about that object, select `Help` ⇒ `Contents and Index`, click on the `Index` tab, type in the name of the object, and click `Display`.

For example, select `Help` ⇒ `Contents and Index`, click on the `Index` tab, and type in `Collector`. Click `Display` to display the `Help` topic for the `Collector` object.

Other Practice Exercises Using the Help Facility

- Look up the short-cut to delete an object.

  Select `Help` ⇒ `Contents and Index` ⇒ `How Do I...` ⇒ `Use the Keyboard Shortcuts` ⇒ `Editing Programs` ⇒ `To Cut an Object or Text`.

- Look up the word “terminal.”

  Select `Help` ⇒ `Contents` ⇒ `Reference` ⇒ `Glossary` ⇒ `Terminal`.

- Look up the VEE version number.

  Select `Help` ⇒ `About VEE Pro`.

- Find out what is new in this version of Agilent VEE.

  Select `Help` ⇒ `Contents and Index` ⇒ `What's New in Agilent VEE 6.0`.

Debugging Programs in Agilent VEE

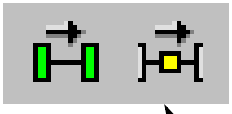
This exercise uses the program you created in "Creating a Panel View (Operator Interface)" on page 91. Select `File` ⇒ `Open`, highlight `simple-program_with_panel.vee`, and click `OK`.

VEE displays error messages during development and when a program runs, and can display caution, error, and informational messages as follows:

- When you run a program, VEE may display a yellow-titled `Caution` box.
- When you run a program, VEE may display a red-titled `Error` box.
- If you make a mistake while creating a program, such as typing an out of range value of `33000` into an `Int16 Constant`, VEE displays an `Error` message box with a dark blue title bar.
- VEE also displays information in the status bar about errors and cautions. The status bar is along the bottom of the VEE window.

Showing Data Flow

1. Click the **Show Data Flow** button on the center of the tool bar as shown in Figure 2-18. (Or you can click `Debug` ⇒ `Show Data Flow`.)



Show Data Flow button on toolbar

Figure 2-18. Show Data Flow

(To turn it off, click it again.) When you run the program, you will see small squares moving along the data lines to indicate the flow of data.

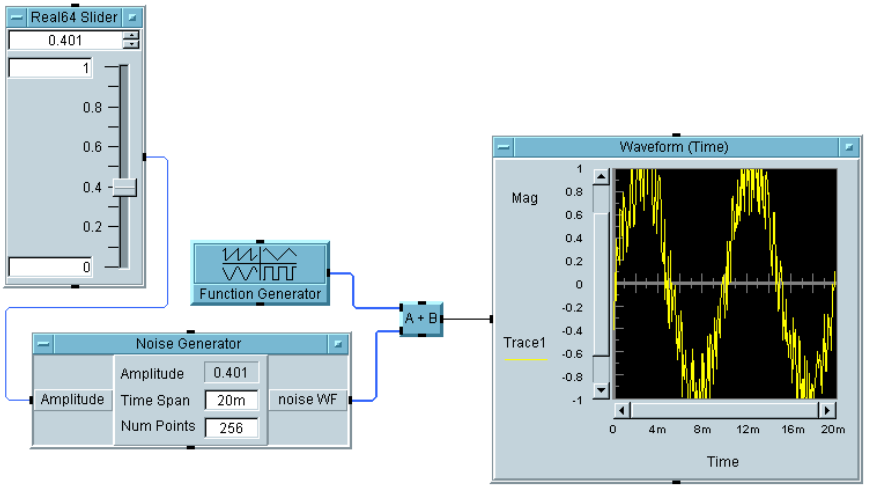
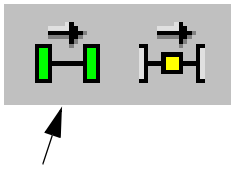


Figure 2-19. Data Flow in simple-program.vee

For example, in Figure 2-19, data moves from the `Real64 Slider` to the `Noise Generator`. The output from the `Noise Generator` and the `Function Generator` are input to the `A+B` object, and the results are displayed in the `Waveform (Time)` display.

Showing Execution Flow

1. Click the **Show Execution Flow** button on the tool bar as shown in Figure 2-20. (Or click `Debug` ⇒ `Show Execution Flow`.)



Show Execution Flow button on toolbar

Figure 2-20. Show Execution Flow

When you run the program, you will see a colored outline around the objects as they execute.

Use `Data Flow` and `Execution Flow` to understand how a program is operating, but turn them off to get higher performance. Combining these features with debugging tools such as **breakpoints** will help you understand how a VEE program works and where possible errors lie.

Examining Data on a Line

Checking the data at different points in your program is a fast, useful way to debug your program. The **Line Probe** is a way to view the data on a given line.

Place the mouse pointer over a data line in the detail view. The cursor becomes a graphic of a magnifying glass. The line and its connections are highlighted, and a box appears displaying the data value on the line. Click the magnifying glass cursor, and a dialog box appears with more information about the data line. (Or click `Debug` ⇒ `Line Probe` and click on a line.)

For example, Figure 2-21 shows part of a VEE program with the output displayed from the iconized `Function Generator`. The output shows the `Function Generator` generates a 256-point waveform array.

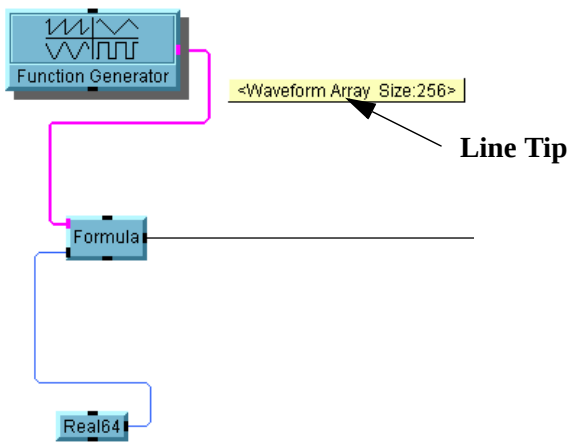


Figure 2-21. Displaying the Value on an Output Pin

If you click on a data line, a dialog box appears with all the information about the data on the line. For example, Figure 2-22 shows the dialog box that appears when you click on the output of the `Function Generator`.

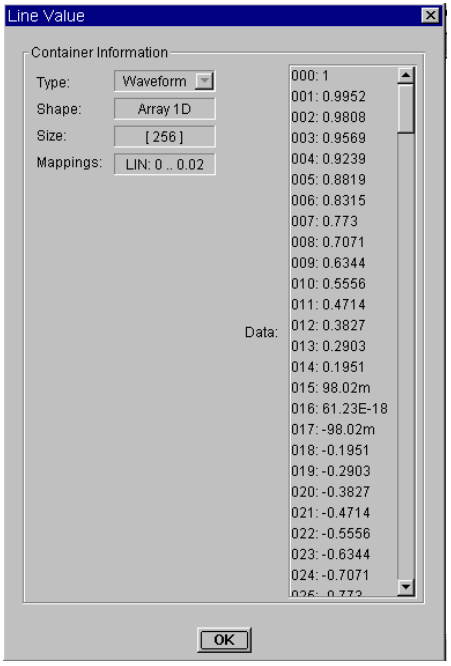


Figure 2-22. Displaying Information about a Line

Examining Terminals

To examine a terminal, double-click it in the open view as mentioned in “Understanding Pins and Terminals” on page 46. If an object is iconized, place the mouse pointer over the terminal, and VEE automatically pops up the name of the terminal.

Using the Alphanumeric Displays for Debugging

You can add the `Alphanumeric` or `Logging Alphanumeric` displays at different points in a program to track the flow of data. When the program is running correctly, delete them. `AlphaNumeric` displays a single data container (a `Scalar` value, an `Array 1D`, or `Array 2D`), and `Logging AlphaNumeric` (either a `Scalar` or `Array 1D`) displays consecutive input as a history of previous values. You can also use a `Counter` to see how many times an object ran.

Using Breakpoints

A **breakpoint** causes a program to pause before it executes a particular object. You can set breakpoints in a program to examine the data. When a breakpoint is set on an object, the object is highlighted with an orange colored outline. When the program runs, it will pause before executing that object.

1. Set a breakpoint on a single object. Double-click the title bar of an object to get the `Properties` dialog box, then select `Breakpoint Enabled` and click `OK`. Then select `Debug` ⇒ `Activate Breakpoints`. Run the program. It will pause at the object with the breakpoint.

2. Set additional breakpoints on several other objects. Select the objects. (Press **Ctrl** and click on each object.) Click the **Toggle Breakpoint(s)** button on the tool bar as shown in Figure 2-23. (You could also press **Ctrl-B**.) Run the program again. The program pauses at the first object with a breakpoint set.

Figure 2-23. Set Breakpoint(s)

3. Resume the program to continue and pause at the next object with a breakpoint set. Click the **Resume** button on the tool bar, shown in Figure 2-24. (Also in the `Debug` menu.)

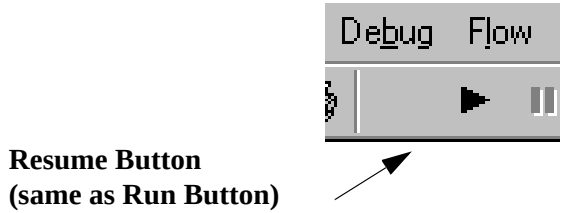


Figure 2-24. Resume Program (same as the Run Button)

4. Now clear breakpoints from the program. Select the objects with breakpoints. Click the **Toggle Breakpoint(s)** button on the tool bar, shown in Figure 2-25. You can also select `Debug ⇒ Clear All Breakpoints`.

Figure 2-25. Clear Breakpoint(s)

5. To pause or stop the program, click the **Pause** or **Stop** buttons on the tool bar, shown in Figure 2-26. (Also located in the `Debug` menu.)

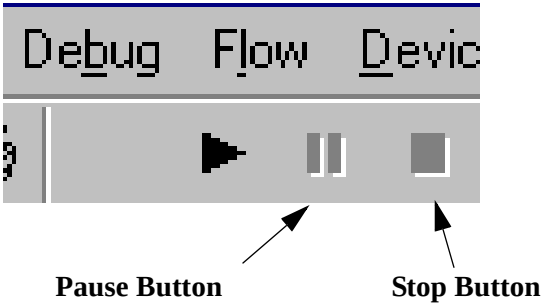


Figure 2-26. Pause or Stop a Program

Resolving Errors

If you get an error message when you run a program, VEE automatically puts a red outline around the object where the error was found.

You can either correct the error and the outline will disappear, or you can click the **Stop** button, which will remove the red outline, and then fix the error. If you click **Stop**, you can look at the error again before resuming, with `View ⇒ Last Error`.

Using the Go To Button to Locate an Error

Figure 2-27 shows an example runtime error message. When this program runs, VEE displays a Run Time error and shows a red outline around the `UserObject AddNoise`. When the `Go To` button is pressed, VEE opens the `UserObject AddNoise` and shows a red outline around the `A + B` object, which is missing a connection on the `A` input pin. In a large program, the `Go To` feature can help you locate the source of an error quickly.

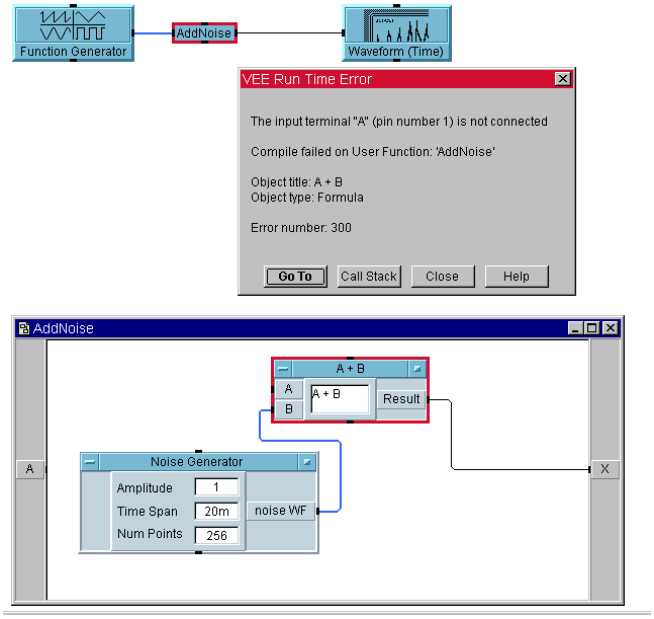


Figure 2-27. Example Runtime Error Message using Go To

Using the Call Stack

If an error is in the Main program, it may be easy to see. However, in a large program, the `Call Stack` helps locate errors that are nested several levels deep.

1. Press the **Pause** button on the tool bar (next to the **Run** button).

2. Press the `Call Stack` button on the error dialog box, or select `View` ⇒ `Call Stack`. `Call Stack` lists the hierarchy of the execution of the program.

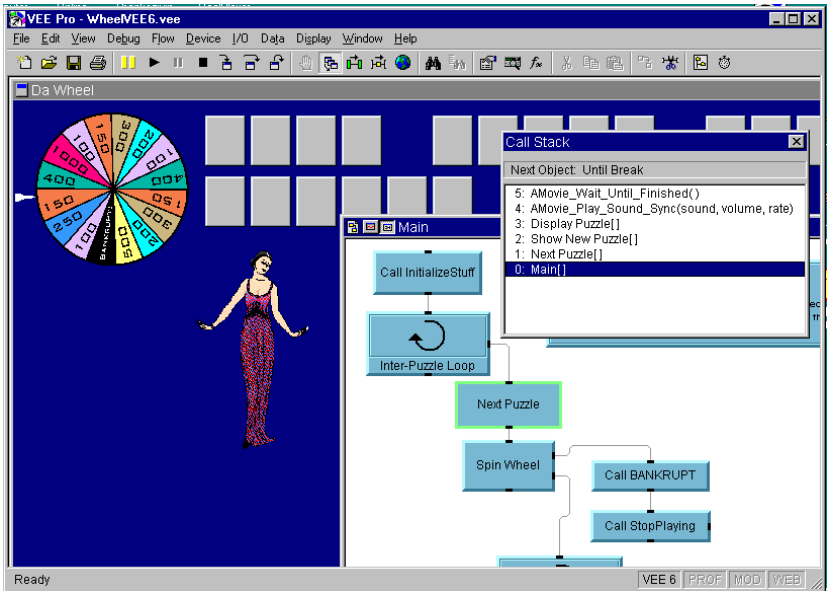


Figure 2-28. Using the Call Stack in Wheel.exe

The `Call Stack` shows the hierarchy of the execution of the program. Figure 2-28 shows an example program that is shipped with VEE: the `Wheel.exe` program in `Examples/Games`. In Figure 2-28, the program is currently executing `AMovie_Wait_Until_Finished()` user function which was called by `AMovie_Play_Sound_Sync` which was called by `...Next_Puzzle` in Main. You can double-click on any of the items in the Call Stack listing to have VEE locate and show the function.

Following the Order of Events Inside an Object

Figure 2-29 shows the order of events inside an object.

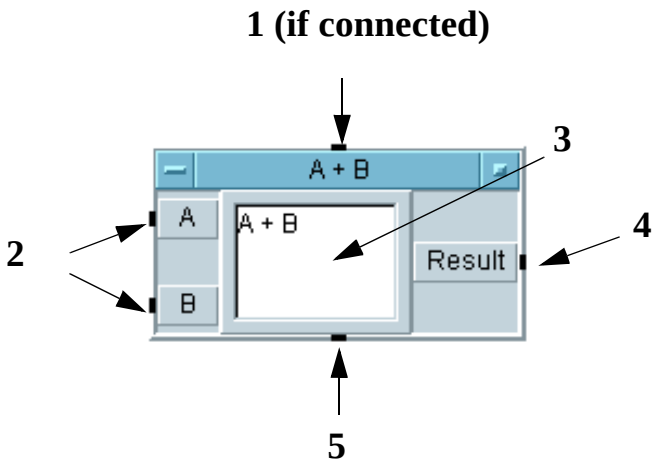


Figure 2-29. The Order of Events in an Object

In Figure 2-29, the pins operate as follows:

1 If the sequence input pin is connected, the object will not operate until it receives a message to execute (a “ping” in VEE terms). However, the sequence input pin *does not* have to be connected.

2 All data input pins must have data before the object operates. (You can add data input/output pins to most objects. Click on the `Add/Delete Terminal` menu in any object menu to find out the pins that can be added.)

3 The object performs its task. In this case, `A` is added to `B` and the result is placed on the output pin.

4 The data output pin fires. The object waits for a signal from the next object that the data is received before its operation is completed. Therefore, a given object does not fire its sequence output pin until all objects connected to its data output pin have received data.

5 The sequence output pin fires.

There are exceptions to this sequence of events:

- You can add error output pins to trap errors inside an object. The error output pins override the standard object behavior. If an error occurs when the object executes, the error pin will send out a message and the data output pins will not fire.

- You can add control input pins to some objects, and they may cause the object to perform some immediate action. For example, an object sub-function such as `Title` or `Autoscale` in the `Waveform (Time)` display can be performed with control pins. Control lines to an object are shown in VEE programs as dashed lines

  For example, Figure 2-30 shows a control line that sets a custom title for the waveform display. Note that the object is not required to have data on a control pin to perform this action. The object does not execute, only the action such as setting the title is performed. You can click `Show Data Flow` to see how the control line to the `Title` control input will carry data first.

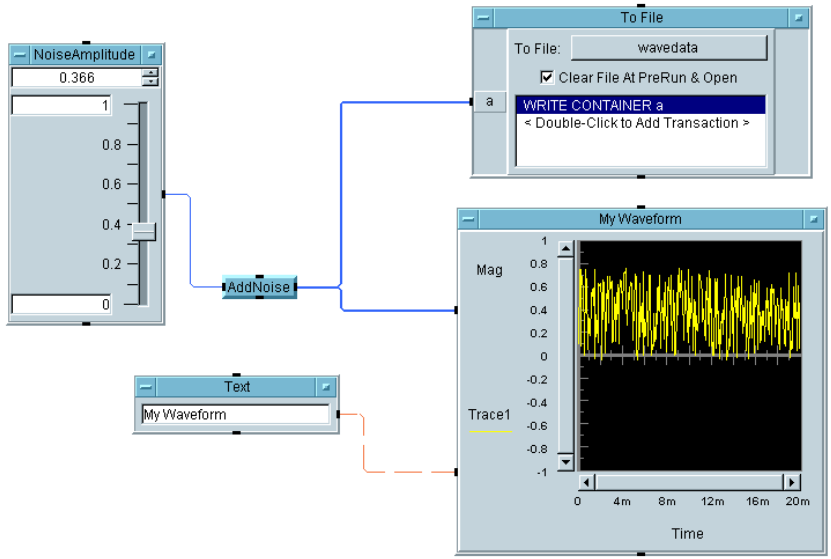


Figure 2-30. Control Line Used to Execute Custom Title

Following the Execution Order of Objects in a Program

As a VEE program runs, the objects execute in the following order:

1. `Start` objects operate first.

   Figure 2-31 shows a VEE program with two **threads**, which are sets of objects connected by solid lines in a VEE program. The `Start` objects, located under `Flow` ⇒ `Start`, are used to operate the individual threads in a program. If a program includes `Start` object(s), they execute first.

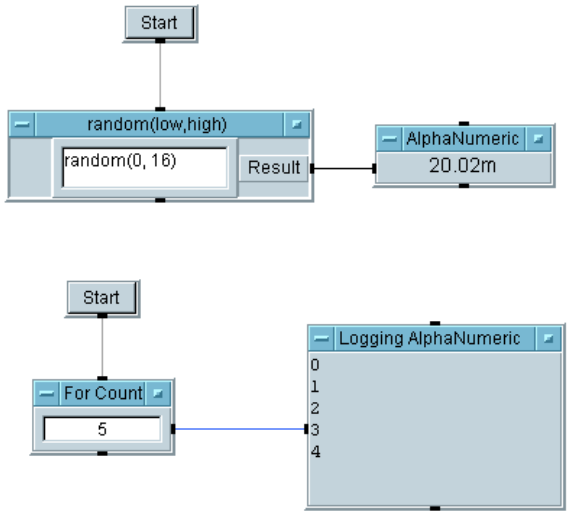


Figure 2-31. Start Objects Executing Separate Threads

2. Objects with no data input pins operate next. `Data` ⇒ `Constant` objects are often in this category.

3. Objects with input pins will only operate when all connected inputs are satisfied. (Recall that connecting sequence inputs is optional.)

Stepping Through a Program

Stepping through a program is a very effective debugging tool. VEE has functions to `Step Into`, `Step Over`, and `Step Out` of objects.

To activate stepping, click the **Step Into**, **Step Over**, or **Step Out** buttons on the tool bar, shown in Figure 2-32.

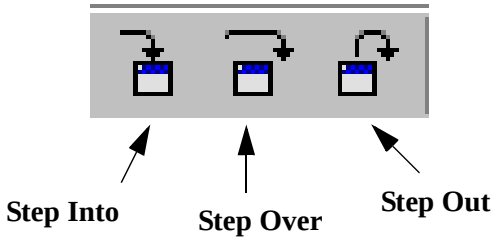


Figure 2-32. Step Into, Step Over, and Step Out Buttons on the Toolbar

- **Step Into** executes a program one object at a time. If the program reaches a `UserObject` or `UserFunction`, VEE puts the `UserObject` or `UserFunction` into detail view and executes each of the objects inside it.

- **Step Over** and **Step Out** execute a program one object at a time, without opening `UserObjects` or `UserFunctions`. If the program reaches a `UserObject` or `UserFunction`, VEE executes the `UserObject` or `UserFunction` in its entirety.

For example, to step through a program:

1. Open the `simple-program_with_panel.vee` program.

2. Click the **Step Into** button on the tool bar.

3. As you keep clicking **Step Into**, the colored outlines around the objects guide you through the program sequentially.

When stepping, VEE puts the `Panel View` behind the `Detail View` to show you the order of objects as they execute. Within `Main` the input boxes have no input pins connected, so they execute first in no defined order. If you wanted them to execute in a particular order, you could control this by connecting their sequence pins.

Data flows left to right, so you see the data generators executing next in no particular order. The addition (`A+B`) object cannot execute until both inputs are satisfied. Then the `Waveform (Time)` object executes. Again, you could mandate execution order anywhere in the program by using the sequence pins or the `Flow` ⇒ `Do` object. (To learn more about the `Do` object, consult `Help`.)

Note

For more information about the step functions, refer to online `Help`. For more information about `UserFunctions`, refer to Chapter 8, "Using Agilent VEE Functions," on page 293.

Finding an Object in a Complex Program

To find a particular object, especially in a large program, select `Edit` ⇒ `Find`. Type in the object or function name in the pop-up dialog box, and VEE displays all instances and locations of that object or function in the program. (See "Finding Functions in Large Programs" on page 321 for more details.)

Practice Programs

The practice programs in this section illustrate more VEE features.

Lab 2-6: Generate a Random Number

1. Document the program:

   a. Select `Display ⇒ Note Pad` and place it at the top center of the work area. Click on the editing area to get a cursor, then enter:

   ```
   This program, Random, generates a real number
   between 0 and 1, then displays the results.
   ```

2. Select `Device ⇒ Function & Object Browser`. Select `Type ⇒ Built-in` function, `Category ⇒ All`, and `Functions ⇒ random`. Click `Create Formula`. Place the object in the work area and click to place the object.

3. Click `Data ⇒ Constant ⇒ Int32` and place it to the left of `random`. Open the `Int32` object menu, click `Clone`, and put this below the other `Int32` object. Double click the `0` to get a cursor, then enter `1`. Connect the constant object with `0` to the low input pin of `random`, and connect the constant `1` to the high input pin.

4. Select `Display ⇒ AlphaNumeric` and place it to the right of the `random` object. Open the object menus and select `Help` to understand the objects better.

5. Connect the `random` object output pin to the `AlphaNumeric` input pin. A data line appears, connecting the two objects.

Note

As you move the mouse pointer with the line attached near the target pin, a box highlights the pin. Then you click again to complete the connection.

Note

If for some reason you want to terminate the line connecting operation before you have completed the link, double-click the mouse and the line will disappear.

6. Click the **Run** button on the tool bar, and you will see a random number displayed as shown in Figure 2-33.

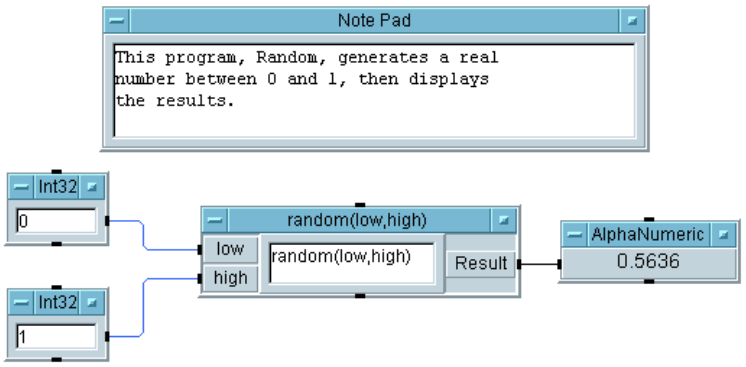


Figure 2-33. The Random Program

7. Select `File` ⇒ `Save As...`, type `Random.VEE`, and click `OK`. (Or save to `EVAL.VEE`, if you are using the evaluation kit software.) This name will appear next to VEE in the title bar when you open it in the future.

Lab 2-7: Setting and Getting a Global Variable

This program gives you more practice in the basic mechanics of building a VEE program, and introduces global variables. You can use the `Set Variable` object to create a variable that can be retrieved later in the program using a `Get Variable` object. You can use any VEE data type. This example uses a number of type `Real64`. (For more information about VEE data types, see Chapter 4, “Analyzing and Displaying Test Data.”)

1. Select `Display ⇒ Note Pad` and place it at the top-center of the work area. Click on the upper left-hand corner of the editing area to get a cursor, then enter the following information:

   ```
   Set and Get a Global Variable prompts the user to
   enter a real number. The variable, num, is set to this
   real number. Then num is recalled and displayed.
   ```

2. Select `Data ⇒ Constant ⇒ Real64` and place it on the left side of the work area. Open the object menu and examine the `Help` entry.

3. Open the `Real64` object menu and select `Properties`. Change the title to the prompt, `Enter a Real Number:`, then click `OK`.

Note

This exercise uses one of the Constant objects for an input dialog box by simply changing its title to a prompt. This is a common technique for getting user input. You could use `Data ⇒ Dialog Box ⇒ Real64 Input`. Also, you can double-click on the title bar to get the `Constant Properties` dialog box.

4. Select `Data ⇒ Variable ⇒ Set Variable` and place it to the right of the `Real64` object. Double-click `globalA` to highlight it, then enter `num`. Notice the name of the object changes to `Set num`.

   This means that the user will enter a real number in the `Real64` object. When the user clicks the **Run** button, the number will be set to the global variable, `num`.

5. Connect the data output pin of the `Real` object to the data input pin of the `Set num` object.

6. Select `Data ⇒ Variable ⇒ Get Variable` and place it below the `Set num` object. Change the variable name to `num`. Notice the name of the object changes to `Get num`.

7. Connect the `Set num` sequence output pin to the `Get num` sequence input pin.

Note

A global variable has to be set before you can use it. Therefore, you need to use the sequence pins in this case to make sure that the variable num has been set, before you retrieve it with `Get num`.

8. Select `Display` ⇒ `AlphaNumeric` and place it to the right of the `Get num` object.

9. Connect the `Get num` data output pin to the `AlphaNumeric` data input pin.

10. Enter a real number and click the run button on the tool bar. The program should look similar to Figure 2-34.

11. `Select File` ⇒ `Save As...` and name the program `global.vee`. (If you are using the evaluation kit software, save the program to `EVAL.VEE`.)

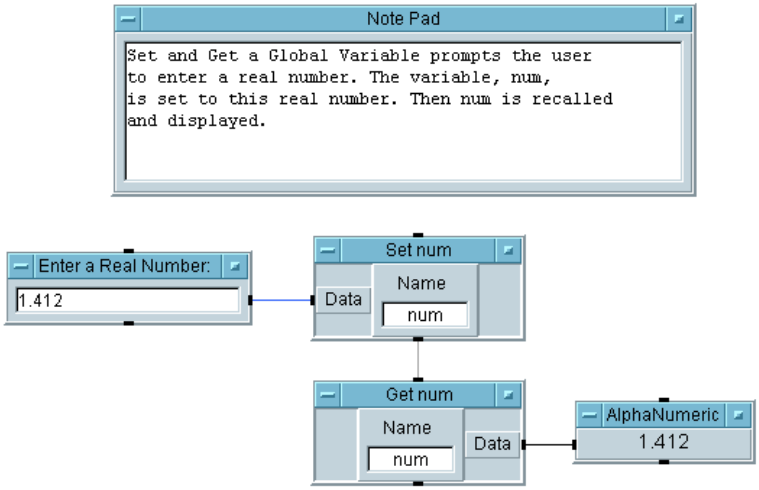


Figure 2-34. Set and Get a Global Variable

Documenting Agilent VEE Programs

By using the `File ⇒ Save Documentation...` command, you can automatically generate program documentation. VEE lists all objects, with key settings, the default and user names, the Description entries, and any "nesting." For example, objects within a `UserObject` are nested one level from the main VEE environment, and these levels are indicated by numbers.

You can also document individual objects using a Description. First, this exercise describes how to document an individual object, and then how to generate the program documentation.

Documenting Objects with Description Dialog Boxes

All objects have a `Description` item in their object menus, which provides a dialog box to accept documentation on that particular object. This documentation file also provides a way to correlate the documentation with screen dumps. In this section, you will add an entry to the `Description` dialog box.

1. Open the `Random.vee` program.

2. In the Main object menu, click `Description`. Type text in the dialog box as shown in Figure 2-35: Click `OK` when you are done.

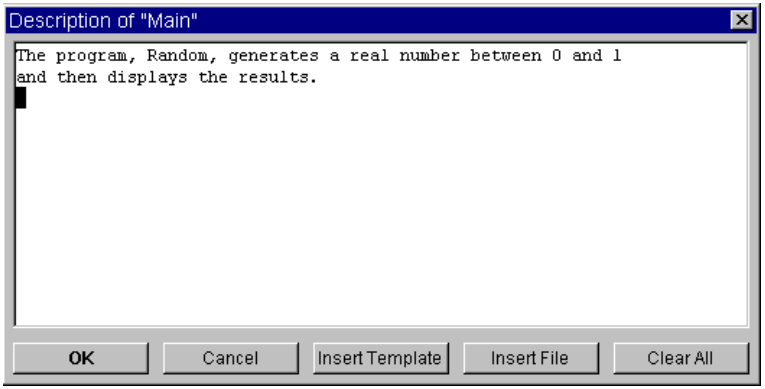


Figure 2-35. The Description Dialog Box

Note

The entries in the `Description` dialog box will not be visible to users unless they access them through the object menu. Also, notice that you can insert a file or a template in this dialog box.

Generating Documentation Automatically

Follow these steps to generate a file of program documentation:

1. Open `Random.vee`. Click `File` ⇒ `Save Documentation...`. Enter the file name using a `*.txt` suffix (`Random.txt`, for example), then click `Save`. By default, the file is saved on a PC in the folder `C:\My Documents\VEE Programs`.

2. Open the file in any text editor to view or print. Figure 2-36, Figure 2-37, and Figure 2-38 show the documentation file using the Notepad program in MS Windows98.

   Figure 2-36 shows the beginning of the file, with information on the file, revision dates, and system I/O configuration.

```
Source file: "C:\\My Documents\\VEE Programs\\Random.vee"
File last revised: Mon Jan 03 15:29:02 2000
Date documented:  Mon Feb 28 14:43:27 2000
VEE revision: 6.0
Execution mode: VEE 6
Convert Infinity on Binary Read: no

I/O Configuration
  My Configuration (C:\WINDOWS\Local Settings\Application
Data\Agilent VEE\vee.io)
```

Figure 2-36. The Beginning of the Documentation File

```
M: Main
 Device Type              : Main
 Description              :
   1. The program, Random, generates a real number between 0 and 1
   2. and then displays the results.
 Context is secured       : off
 Trig mode                : Degrees
 Popup Panel Title Text   : Untitled
 Show Popup Panel Title   : on
 Show Popup Panel Border  : on
 Popup Moveable           : on
 Popup Panel Title Text Color         : Object Title Text
 Popup Panel Title Background Color   : Object Title
 Popup Panel Title Text Font          : Object Title Text
 Delete Globals at Prerun : on

M.0: Main/Note Pad
 Device Type              : Note Pad
 Note Contents            :
   1. This program, Random, generates a real
   2. number between 0 and 1, then displays
   3. the results.
   4.

M.1: Main/random(low,high)
 Device Type              : Formula
  Input pin  1            : low (Any, Any)
  Input pin  2            : high (Any, Any)
  Output pin 1            : Result
 Formula                  : random(low,high)
```

Figure 2-37. The Middle of the Documentation File

In Figure 2-37, the VEE objects are described along with their settings. The number before each object indicates where the object is located. For example, the first object in Main is listed as `M1`. Figure 2-38 shows the remainder of this documentation file for your reference.

```
M.2: Main/Int32
 Device Type              : Constant
  Output pin 1            : Int32
 Wait For Event           : off
 Auto execute             : off
 Initialize At Prerun     : off
 Initialize at Activate   : off
 Constant size fixed      : off
 Password masking         : off
 Indices Enabled          : on
 Int32 Value              : 0
M.4: Main/Int32
 Device Type              : Constant
  Output pin 1            : Int32
 Wait For Event           : off
 Auto execute             : off
 Initialize At Prerun     : off
 Initialize at Activate   : off
 Constant size fixed      : off
 Password masking         : off
 Indices Enabled          : on
 Int32 Value              : 1
M.5: Main/AlphaNumeric
 Device Type              : AlphaNumeric
  Input pin  1            : Data (Any, Any)
 Clear At Prerun          : on
 Clear at Activate        : on
 Indices Enabled          : on
```

Figure 2-38. The Remainder of the Documentation File

Note

After you run the `Save Documentation` command, run a `File` ⇒ `Print Program` command to put identification numbers on the objects, so you can match the text documentation to the printer output.

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary, before proceeding to the next chapter.

- Create a `UserObject`, and explain how `UserObjects` give programs structure and save space on screen.
- Create pop-up dialog boxes and sliders (or knobs) for user input.
- Use data files to save data to a file and load data from a file.
- Create an operator interface, using a Panel view of the program.
- Use different data types and data shapes.
- Use mathematical operators and functions.
- Use online `Help`.
- Show the data flow and the execution flow in a program.
- Debug programs by examining data on a line, terminals, and alphanumeric displays.
- Use breakpoints.
- Resolve errors with the `GoTo` command.
- Resolve errors using the Call Stack.
- Use Step Into, Step Over, and Step Out to trace and debug a program.
- Use the Find feature.
- Document objects with description dialog boxes.
- Generate a documentation file.

3

Easy Ways to Control Instruments

Easy Ways To Control Instruments

In this chapter you will learn about:

- Configuring an instrument
- Using a panel driver
- Using the Direct I/O object
- Controlling PC plug-in boards
- Using a VXI*plug&play* driver

Average time to complete: 1 hour

Overview

In this chapter, you will learn how to use VEE to control instruments. With VEE, you can control instruments in several ways:

- **"Panel" drivers** give you a simple user interface (or "front panel") to control an instrument from your computer screen. When you change parameters in the VEE panel driver, the corresponding state of the instrument is changed. Panel drivers are provided by Agilent Technologies with VEE and cover over 450 instruments from different vendors.

- **The Direct I/O object** allows you to transmit commands and receive data over many supported interfaces. This technique is equivalent to using command strings with a textual language like Rocky Mountain Basic.

- **Open Data Acquisition Standard Drivers (ODAS drivers)** use ActiveX Automation technology to control PC Plug-in cards (PCPI cards). An ODAS driver has a standard format, and therefore can be supplied by the vendor of the PC Plug-in card or another third party.

- **I/O libraries** can be imported to control PC Plug-in boards and then call functions from that library using the `Call` object. These libraries, usually shipped as Dynamic Link Libraries (DLLs), are similar to ODAS drivers, but ODAS drivers are standard and easier to use.

- **VXI*plug&play* drivers** can be used to call C functions to control instruments. These are provided by Agilent Technologies and other vendors with their supported instruments.

This chapter is designed to give you the fundamentals of controlling instruments to cover most situations. For more complete information, refer to the *VEE Pro Advanced Techniques* manual.

Panel Drivers

Agilent VEE includes over 450 panel drivers for different instrument vendors. A panel driver works by using a display in the VEE program that controls the settings in the corresponding physical instrument. Panel drivers provide maximum ease-of-use and save the most development time. Figure 3-1 shows an example panel driver.

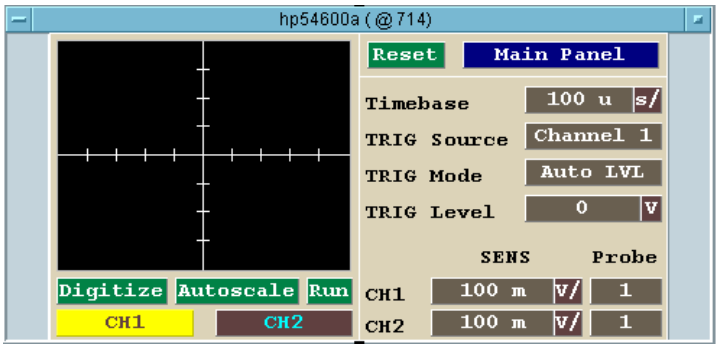


Figure 3-1. The HP54600A Scope Panel Driver

Direct I/O Object

VEE's Direct I/O object allows you to communicate with any instrument from any vendor over standard interfaces (whether or not there is a driver available for the instrument). The Direct I/O object works by transmitting commands to the instrument and receiving data back from the instrument. Using Direct I/O generally yields faster execution speeds. Choosing the best method of instrument control will depend on driver availability, the need for fast test development, and the performance requirements. Figure 3-2 shows an example using Direct I/O to control a function generator.

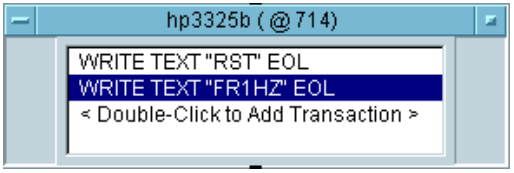


Figure 3-2. A Function Generator Direct I/O Object

PC Plug-in Boards with ODAS Driver

ODAS drivers are supplied by the vendor of the PC Plug-in card or can also be supplied by a third party, since they are standard drivers. VEE enables you to control a PC Plug-in board with an ODAS driver by choosing PC Plug-in board functions in a `Formula` object.

ODAS drivers give you a more standard way to control a PC Plug-in board than proprietary DLLs, and port better from one PC to another. Figure 3-3 shows an example of a `Formula` object in VEE used to control a PC Plug-in board with an ODAS driver.

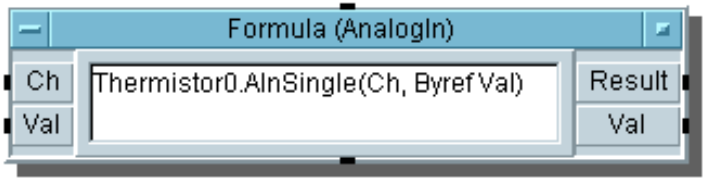


Figure 3-3. ODAS Driver Object in a VEE Program

PC Plug-in Boards with I/O Library

I/O libraries, usually shipped as Dynamically Linked Libraries (or DLLs) for PC Plug-in boards, are supplied by the vendor of the PC Plug-in board. VEE enables you to control the PC Plug-in board by calling library functions with the `Call` object. Figure 3-4 shows an example of the `Import Library` object that makes the functions available in VEE.

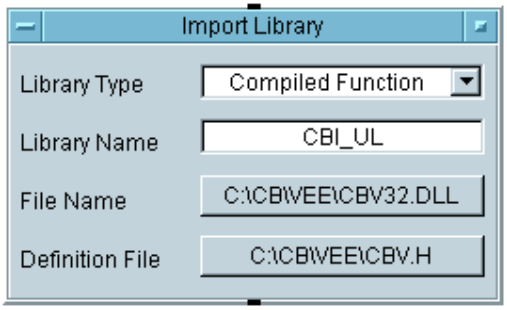


Figure 3-4. Importing a PC Plug-In Library

VXI*plug&play* Drivers

VXI*plug&play* drivers are supplied by the instrument vendor or by Agilent Technologies. (For a list of VXI*plug&play* drivers available from Agilent Technologies, refer to the VEE literature or the *VEE Pro Advanced Techniques* manual. Contact your instrument vendor for other VXI*plug&play* drivers.) VEE enables you to control an instrument with a VXI*plug&play* driver by making calls to the driver. Figure 3-5 shows an example of calls to a VXI*plug&play* driver from VEE.

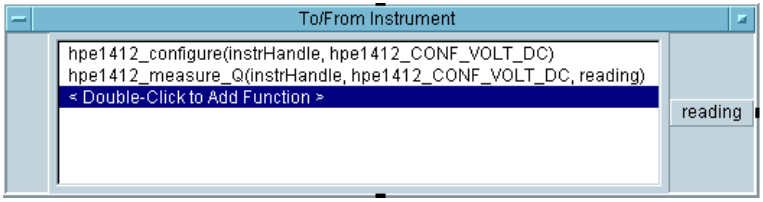


Figure 3-5. Calls to a VXI*plug&play* Driver from VEE

Configuring an Instrument

With VEE you can develop programs without the instruments present. In this exercise, you will configure an oscilloscope for use with a panel driver. Then you will add the physical instrument to the configuration.

Lab 3-1: Configuring an Instrument without the Instrument Present

1. Select `I/O` ⇒ `Instrument Manager...`. Move the dialog box to the upper-left work area by clicking and dragging its title bar, as shown in Figure 3-6.

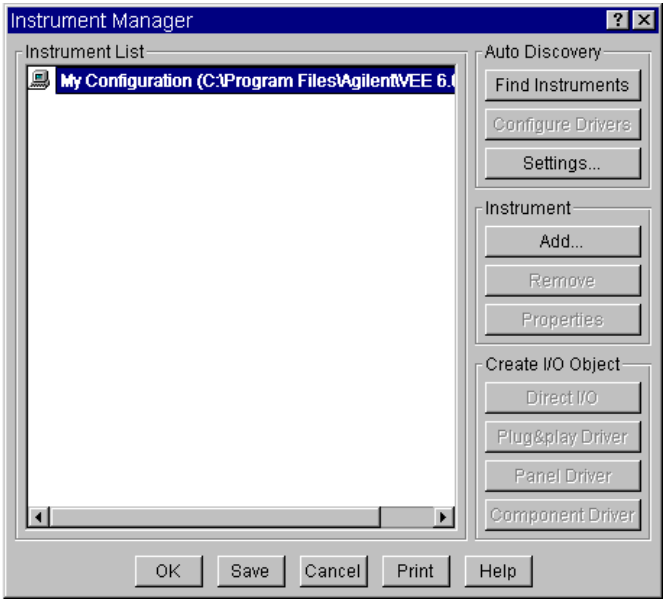


Figure 3-6. The Instrument Manager Box

Note

If you have any instruments connected and powered on, VEE can find the instruments and automatically find the drivers for them. For more information about automatically finding and configuring instruments, refer to the online Tutorials under `Help` ⇒ `Welcome` ⇒ `Tutorials` in the main VEE screen.

By default, there are no instruments configured, and this example assumes that no instruments appear in the `Instrument Manager` list.

2. In the `Instrument Manager` dialog box, make sure `My Configuration` is highlighted, and click `Add...` under `Instrument`. The `Instrument Properties` dialog appears, as shown in Figure 3-7.

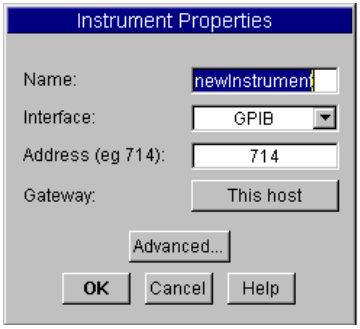


Figure 3-7. Instrument Properties Dialog Box

The entries in the `Instrument Properties` dialog box are as follows:

Name — The name the instrument will be called in the program. Choose a name that follows these syntax guidelines:

- Instrument names must start with an alphabetic character, followed by alphanumeric characters or underscore characters.
- You cannot use embedded blanks in instrument names.

Interface — Type of interface. Choose from GPIB, Serial, GPIO, or VXI.

Address — The logical unit of the interface (GPIB is usually 7) plus the local bus address of the instrument (a number from 0 to 31). If you leave the address at 0, it means that you are developing without an instrument present.

Gateway — Specifies whether instruments are controlled locally or remotely. Use the default entry `This host` to control instruments locally, or enter a `Gateway` for remote control. (For more information, refer to the *VEE Pro Advanced Techniques* manual.)

3. Change the name to `scope`, leave all the other defaults as they are, and click `Advanced...`. The `Advanced Instrument Properties` dialog box appears as shown in Figure 3-8.

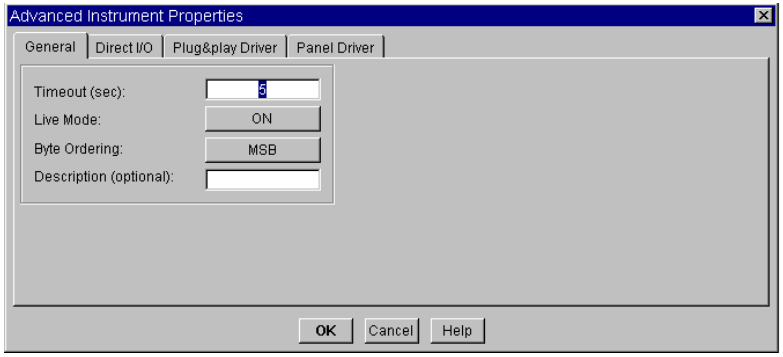


Figure 3-8. The Advanced Instrument Properties Dialog

The entries in the `General` folder are as follows:

Timeout	The maximum number of seconds allowed for an I/O transaction to complete before you get an error message.
Live Mode	Specifies whether there is live communication with the instrument. Set this to `OFF` unless you have an instrument present. VEE defaults to the `ON` setting.
Byte Ordering	Specifies the order the device uses for reading and writing binary data. The field toggles between Most Significant Byte (MSB) first or Least Significant Byte first. All IEEE488.2-compliant devices must default to MSB order.
Description	Enter any description here. For example, if you want the instrument number on the title bar, enter the number.

4. Toggle `Live Mode` to `OFF`. Then click the `Panel Driver` folder, and the dialog box appears as shown in Figure 3-9.

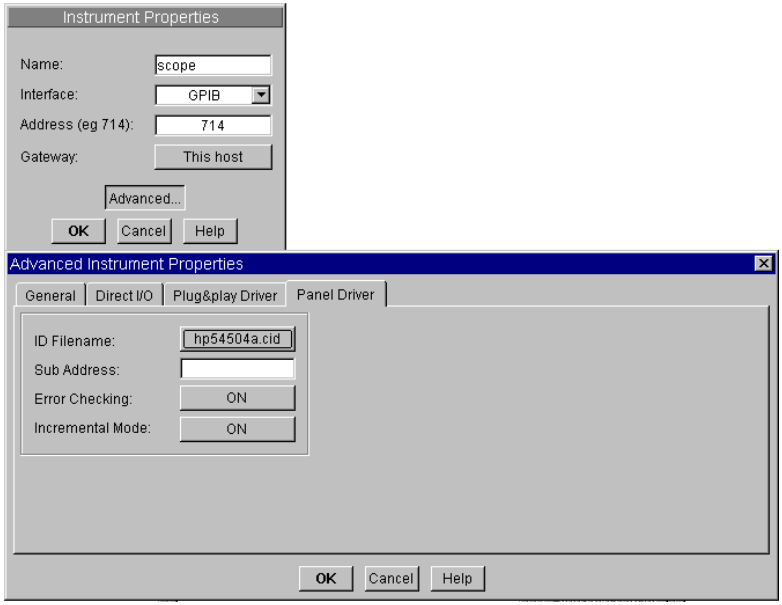


Figure 3-9. The Panel Driver Folder

5. Click the field to the right of `ID Filename` to obtain a list box entitled `Read from what Instrument Driver?`. This list includes all of the panel driver files loaded with your revision of VEE in the directory specified.

Note

You will need to have installed the panel drivers from the VEE CD-ROM in order to complete the example. The `*.cid` files signify the compiled instrument driver files.

6. Scroll down the list to highlight `hp54504a.cid`, then click `Open`. Figure 3-9 shows this instrument already selected. You can also double-click on a highlighted file to select it.

The other entries in the Panel Driver folder are as follows:

Sub Address	Leave this field blank. Sub Address is used only by non-VXI cardcage instruments for identifying plug-in modules.
Error Checking	Leave the default setting `ON`. `Error Checking` can be turned off for extra throughput, but then it does not check for I/O errors.
Incremental Mode	Leave the default setting `ON`. `Incremental Mode` can also be turned off, which sends the entire instrument command string for the instrument state each time you change a setting.

7. Click `OK` to return to the `Instrument Properties` box. Click `OK`.

   The list of available instruments should now include an instrument configuration named `scope`, using the driver file `hp54504a.cid`, as shown in Figure 3-10. The instrument does not have a bus address specified, because it is not live at present. You can develop the program in this mode, and add an address later, when you are ready to connect the instrument to your computer.

   Tip: Press the **Tab** key after typing in a field to move to the next field, and press **Shift-Tab** to move to the previous field. Pressing **Enter** is equivalent to clicking `OK`. VEE closes the dialog box.

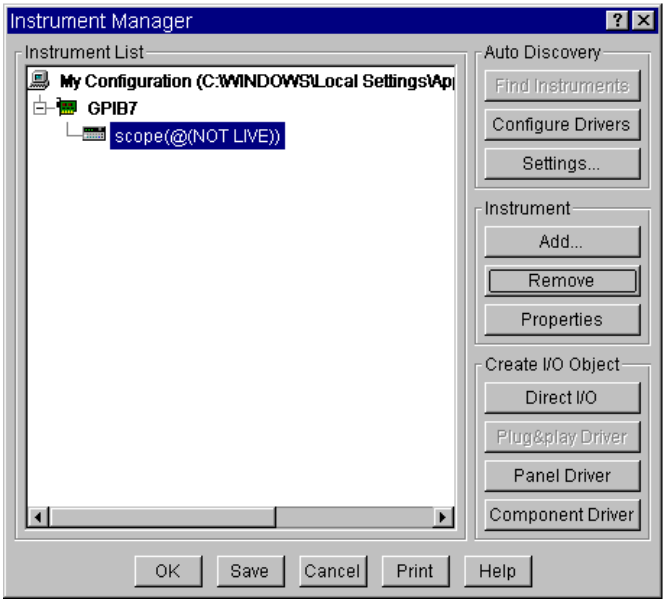


Figure 3-10. Scope Added to List of Instruments

8. Click `Save` to close the `Instrument Manager` box. (You could also click `Panel Driver` under `Create I/O Object` to put it in the program immediately, and VEE would save the configuration automatically.)

You have now added the HP 54504A oscilloscope named `scope` to the instrument list. You can use this driver while programming, even though the actual instrument is not present.

Selecting an Instrument to Use in a Program

1. Select `I/O` ⇒ `Instrument Manager...`.

2. Highlight the selection `scope(@(NOT LIVE))`, then click `Panel Driver` under `Create I/O Object`.

Note

In the `Instrument Manager`, you can often create different types of objects under `Create I/O Object`, depending on the type of instrument configured. For example, if you had chosen `Direct I/O` rather than `Panel Driver` for this exercise, you would get a `Direct I/O` object with the name `scope(@(NOT LIVE))`.VEE also provides a `Component Driver`, which uses a subset of the functions provided by a `Panel Driver`. For more information, refer to the *VEE Pro Advanced Techniques* manual.

3. Place the outline of the `scope` panel and click to place it. The display should look similar to Figure 3-11.

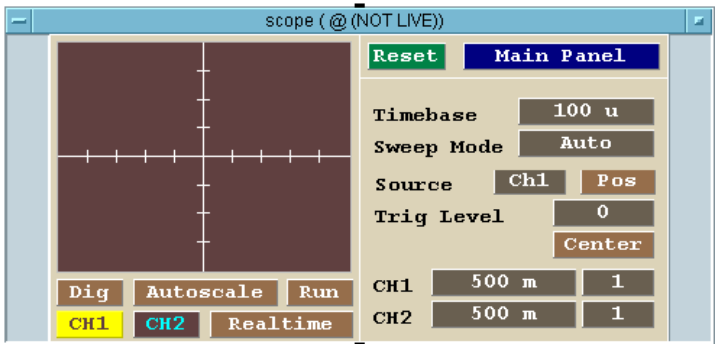


Figure 3-11. Selecting scope(@(NOT LIVE))

You may now use the panel driver in the program like any other VEE object.

Adding the Physical Instrument to the Configuration

1. Select `I/O` ⇒ `Instrument Manager...`, and highlight `scope`. Under `Instrument...` click `Properties`.

2. Double-click the `Address` field to highlight the current entry and type `709`. The `7` in `709` is the logical unit. (If the GPIB (HP-IB) logical unit is not 7, replace 7 with the actual logical unit number.) The `9` in `709` is the default address for scopes.

3. Click `Advanced:` and toggle `Live Mode` to `ON`, then click `OK`. Click `OK` to close the `Instrument Properties` box.

4. Click `Save` to save the changes.

Using a Panel Driver

These exercises use the HP 3325B Function Generator as the example. The principles are the same in using any VEE panel driver. By using a panel driver instead of programming an instrument directly, you save time developing and modifying programs. Changes in the instrument settings are made through menu selections or by editing fields in dialog boxes. If the instrument is connected and `Live Mode` is `ON`, the changes you make will register on the instrument.

To use a panel driver in a program, add inputs and/or outputs as needed and connect the panel driver to other objects. You can use several instances of the same driver in a program to set the instrument to different states. In VEE, you can iconize a panel driver to save space, or use the open view to display the instrument settings. You can also change settings while a program is running.

Lab 3-2: Changing Settings on a Panel Driver

1. Select `I/O` ⇒ `Instrument Manager...`. Select `My Configuration`, then click `Add...` under `Instrument` to display the `Instrument Properties` dialog box, and edit the information as follows:

Name	Edit to `fgen` and press the **Tab** key twice to move to the `Address` field.
Address	Change to `713`, or the address you want on the bus.

2. Click on `Advanced`. In the `General` folder and toggle `Live Mode` to `OFF`.

3. Click on the `Panel Driver` folder and set `ID Filename:` to `hp3325b.cid`. Click `OK` twice to return to the `Instrument Manager`.

4. Under `Create I/O Object`, click `Panel Driver`. Place the object on the left side of the workspace. (This process would be the same regardless of the instrument, as long as the instrument had been configured and added to the list.)

Note

You are programming without the instrument attached. If the instrument were attached, you would edit the configuration to the proper address.

5. Click `Sine` in the `Function field` to get a pop-up menu, and then select `Triangle` as shown in Figure 3-12.

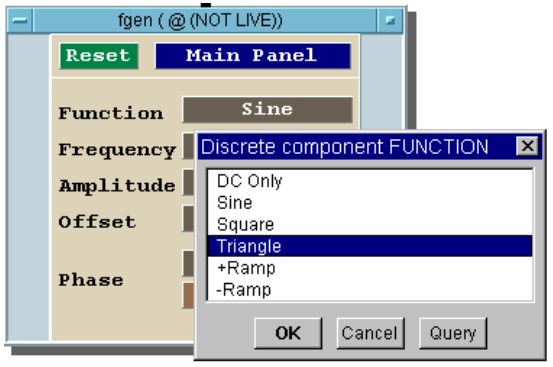


Figure 3-12. The Function Pop-up Menu on fgen

6. Click the field to the right of `Frequency`.

7. Type `100` in the Continuous component `FREQUENCY` dialog box that appears, and click `OK`. Note that the `Frequency` setting has now changed.

You can use the same methods to change the instrument settings on any driver. If the instrument is configured with an address and `Live Mode` is `ON`, every change you make in the driver panel is reflected by the instrument.

Moving to Other Panels on the Same Driver

Most drivers have more than one panel to simplify the user interface. To move to a different panel, click `Main Panel` in the object to get a menu of panels.

1. In the `Panel Driver` object, click `Main Panel` and select `Sweep` in the `Discrete Component MENU` presented as shown in Figure 3-13.

2. Click `OK` to display the `Sweep Panel`. You can also look at the other panels to see what is available.

3. Click `OK` to return to the `Main Panel`.

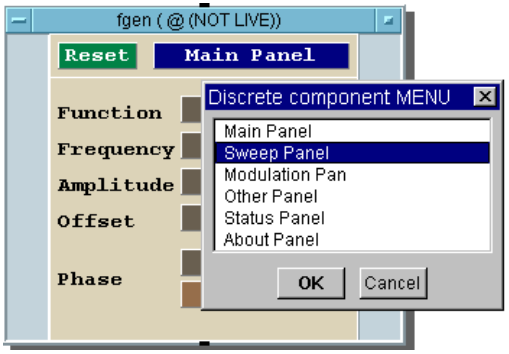


Figure 3-13. Sweep Panel in Discrete Component Menu

Adding Inputs and/or Outputs to a Panel Driver

In addition to interacting with the panel directly, you can control settings or read data from an instrument in a program by adding data inputs and/or outputs to the driver. The input and output areas are shown in Figure 3-14.

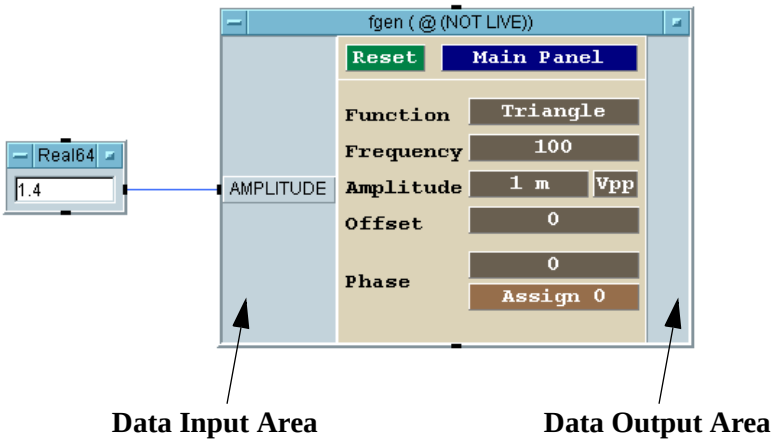


Figure 3-14. The Data Input and Output Areas on a Driver

1. Place the mouse pointer over the data input area of the function generator instrument panel, and press **CTRL-A** to add a data input terminal. A list box of the instrument components appears.

2. Select the desired component from the menu presented.

Note

You could also open the object menu and select `Add Terminal` by `Component` ⇒ `Select Input Component`. Then select the desired component field on the driver.

Follow the same process to add a data output, by placing the mouse pointer in the data output area.

Deleting Data Input or Output Terminals

Place the mouse pointer over the terminal and press **CTRL-D**.

Note

You could also open the object menu and select `Delete Terminal ⇒ Input...` from the object menu and choose the appropriate input from the menu presented.

On Your Own

Set a state on the HP 3325B Function Generator, or any other function generator available. Change the `Function` setting to a `Square` wave. Add input components for `Amplitude` and `Frequency`. Create input dialog boxes for the amplitude and frequency and modify the titles to prompt the operator. Enter different values for the amplitude and frequency, and run the program to see if the settings have changed after operator inputs. (If an instrument is attached, then its settings will change if `Live Mode` is `ON`.)

Using Direct I/O

If there is not a driver available for a particular instrument, or you want higher throughput, use the Direct I/O object.

Lab 3-3: Using Direct I/O

In this exercise, you will configure the HP 3325B function generator using Direct I/O.

1. Select `I/O ⇒ Instrument Manager...`.
2. Highlight the `fgen(@(NOT LIVE))` entry and select `Instrument ⇒ Properties`.
3. Click on `Advanced`. Select the `Direct I/O` folder as shown in Figure 3-15. Look through the options available, then click `OK` to return to `Instrument Properties`, then `OK` again to return to the `Instrument Manager`.

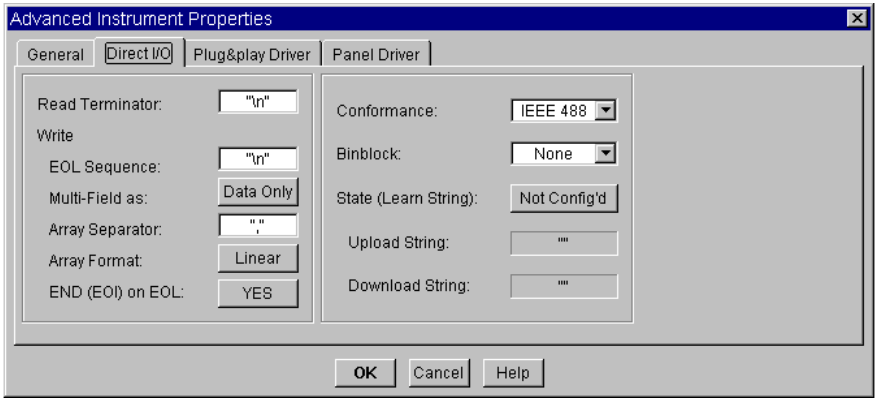


Figure 3-15. The Direct I/O Configuration Folder

Note

This example uses the GPIB interface (IEEE488). To configure Serial, GPIO, or VXI instruments, refer to the *VEE Pro Advanced Techniques* manual.

4. To place the object on the screen, make sure that `fgen(@(NOT LIVE))` is still highlighted, and click `Create I/O Object` ⇒ `Direct I/O`. Figure 3-16 shows the Direct I/O object.

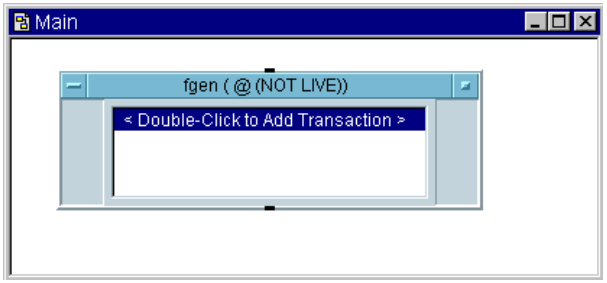


Figure 3-16. A Direct I/O Object

To use a `Direct I/O` object in a program, you have to configure I/O transactions. The next section explains writing text commands, reading data, and uploading/downloading instrument states.

Sending a Single Text Command to an Instrument

To send a single text command to an instrument, type in the appropriate string. Most GPIB instruments use alphanumeric strings for commands sent to the instrument. For example, to send a command to the HP3325B Function Generator to set the amplitude to 5 volts, you would enter the command string `"AM 5 VO"`.

This exercise uses the HP 3325B function generator configured in the previous section. If necessary, go back to “Using Direct I/O” on page 147 and configure the instrument before you continue.

1. In the `fgen(@(NOT LIVE))` object, double-click the transaction bar to get the `I/O Transaction` dialog box, as shown in Figure 3-17.

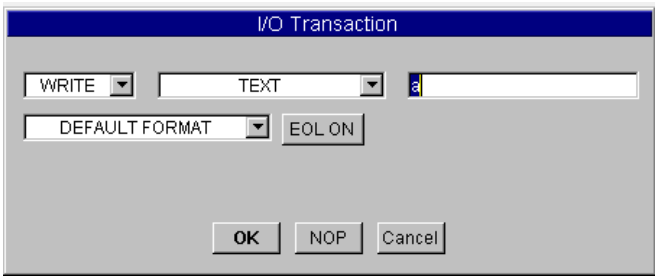


Figure 3-17. The I/O Transaction Dialog Box

The down arrow next to `WRITE` shows a menu of transactions: `READ`, `WRITE`, `EXECUTE`, and `WAIT`. To write data to an instrument, use the default selection. Open the object menu and consult `Help` to find out about each action.

2. Use the default selections `WRITE`, `TEXT`, `DEFAULT FORMAT`, and `EOL ON`. Click the input field labeled `a`, type `"AM 5 VO"` (including the quotes), and click `OK`.

You should see the transaction `WRITE TEXT "AM 5 VO" EOL` as shown in Figure 3-18. The text in quotation marks is the command that will be sent to the HP3325B when the program runs.

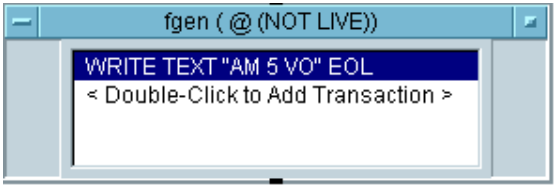


Figure 3-18. A Direct I/O Transaction

In most cases, the process will be the same for sending text commands to instruments. However, there are instruments that specify characters sent at the end of each command or at the end of a group of commands. You need to get this information from the instrument documentation, then include it in the `Direct I/O Configuration` dialog box.

Sending an Expression List to an Instrument

In some cases, you may want to send an expression list to an instrument. For example, you may want to loop through a number of frequencies in the `Function Generator`. To do so using a Direct I/O Transaction, you would use a variable for the frequency in an expression list, and add a data input for that variable to the Direct I/O object. The following steps describe how to send an expression list to an instrument.

1. Place a second `Direct I/O` object for the HP3325B in the Main window. Double-click in the transaction area to get the `I/O Transaction` dialog box.

   You can use all of the defaults except for the command string. In this case, use the format `"FR"`, <*frequency*>, `"HZ"`. This is an Expression List, each expression being separated by commas. The frequency is represented by variable `A`, which will be a data input to the `Direct I/O` object.

2. Click on the input field for command strings and type `"FR",A,"HZ"`. (For example, if `A` were `100`, VEE would send the string `"FR100HZ"`.) Click `OK`. Notice that VEE automatically adds a data input pin labeled A.

3. Select `Flow` ⇒ `Repeat` ⇒ `For Range` and place it to the left of the `Direct I/O` object.

4. Connect the `For Range` data output pin to the `Direct I/O` data input pin.

5. Edit the fields in `For Range` to: `From 10`, `Thru 1.8M`, and `Step 50k`.

   `For Range` will now send out numbers ranging from 10 to 1.8 million in steps of 50,000. As the numbers are received by the Direct I/O object, the command string causes the function generator to output the frequencies. The Direct I/O setup should look like Figure 3-19.

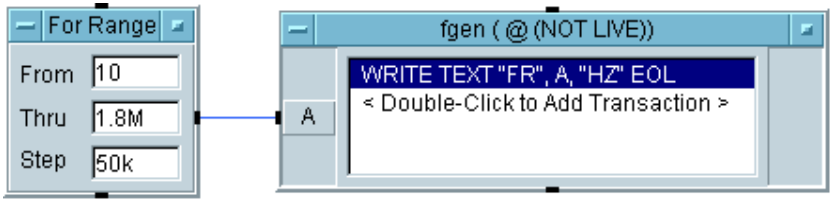


Figure 3-19. Direct I/O Setup Using an Input Variable

6. (Optional) Connect an HP3325B to your computer, if you have one, and edit the configuration of this Direct I/O object to include the address of the instrument. Run the program and you will see the instrument generating these frequencies.

Reading Data From an Instrument

Instruments send data to a computer in many different formats. To read data from an instrument, you must know the datatype you want to read, and whether the data is returned as a single value (scalar) or an array. You must also know if the instrument returns data as text (ASCII) or binary.

You can find this information in the instrument documentation, or you can use the VEE `Bus I/O Monitor` in the `I/O` menu to examine the data being returned. This information determines how to configure the I/O transaction.

In this example, an HP3478A Multimeter is connected to the HP3325B Function Generator described in the last exercise. When the generator sends out a certain frequency, the multimeter triggers a reading and sends the results back to VEE. The following steps describe how to configure the transactions for the multimeter.

Note

This example describes a `READ TEXT` transaction. Other choices for `READ` include `BINARY`, `BINBLOCK`, and `CONTAINER`, which are discussed in detail in the *VEE Pro Advanced Techniques* manual.

1. Select `I/O ⇒ Instrument Manager...`. Click `Add...`. Change the name to `dvm`. Click on `Advanced...` and set `Live Mode:` to `OFF`. Assuming that you do not have an HP3478A connected, click `OK` to return to the Instrument Manager. (If you *do* have an HP3478A, modify the address and the instrument will track the commands.)

2. Highlight `dvm(@(NOT LIVE))` and click `Direct I/O` under `Create I/O Object`.

3. Double-click the `<Double-Click to Add Transaction>` bar to display the `I/O Transaction` dialog box.

4. Highlight the input field and type `"T5"`, then click `OK`. This will write the `"T"` command to the instrument. `T5` is the command for a single trigger to the multimeter.

5. Open the object menu and click `Add Trans...` to add another transaction bar, or use `<Double-Click to Add Transaction>` to add a transaction and display the `I/O Transaction` dialog box.

6. Click the down arrow beside `WRITE` to get a drop-down menu, then select `READ`. When you select `READ`, new buttons appear in the `I/O Transaction` box.

7. Check the `ExpressionList` input field to verify that it contains an `x`. Press **Tab** to move to the next field. Data returned from an instrument is sent to data output pins. In this case, data will be read from the instrument and put into a data output named `x`.

Note Names are *not* case sensitive.

8. Leave the `REAL64 FORMAT` default. The multimeter returns single readings as real numbers.

9. Leave `DEFAULT NUM CHARS` as is.

   The default for the number of characters is 20. If you want to change the number, click on `DEFAULT NUM CHARS` to toggle to `MAX NUM CHARS` and change the number `20` to the desired number.

10. Leave `SCALAR` as is and click `OK`.

    You will see the transaction displayed on the bar as `READ TEXT X REAL64`. Notice that VEE automatically adds a data output named `x`.

Note

If the instrument is returning an array of values, click on the `SCALAR` menu in the `I/O Transaction` dialog box to get the menu for different dimensions, as shown in Figure 3-20. Once you have selected the array dimension, you will also need to specify a size for the array.

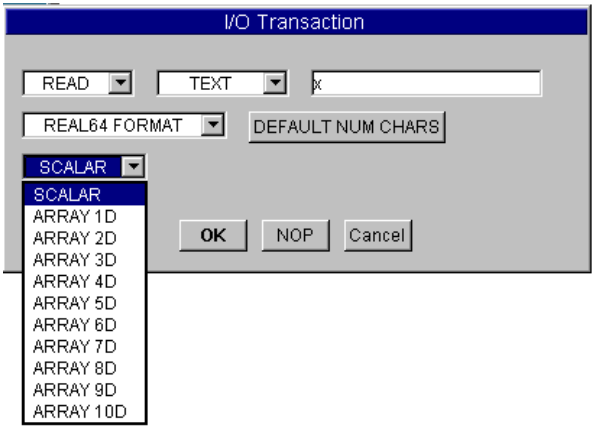


Figure 3-20. Configuring a READ Transaction

11. Add a `Display` ⇒ `AlphaNumeric` to the right and connect its input to the `Direct I/O` output labeled `X`.

    The two Direct I/O transactions should look like Figure 3-21.

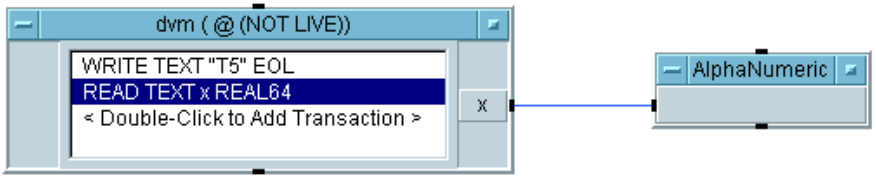


Figure 3-21. Direct I/O Configured to Read a Measurement

The process to configure a transaction is similar, regardless of the data format for the `READ TEXT` transaction. You can explore the other formats available. For a more detailed information about each item, refer to the *VEE Pro Advanced Techniques* manual.

To create a complete test program, this multimeter object and a function generator object could be combined with VEE data and display objects. Fully functional test programs are easy to create in VEE. However, it is beyond the scope of this introductory chapter to show specific details for all the various instruments you might be using. For more complex examples, refer to the *VEE Pro Advanced Techniques* manual.

Uploading and Downloading Instrument States

Some instruments offer a “learn string” capability. The learn string embodies all the function settings that compose an instrument state. Direct I/O will upload this learn string, save it with that particular Direct I/O object, and later allow you to download it to the instrument in the program. To upload an instrument state, follow these steps:

1. Set the instrument to the desired state manually.

2. Open the `Direct I/O` object menu and click `Upload State`.

   Now this state is associated with this particular instance of the Direct I/O object.

3. Open an `I/O Transaction` dialog box by double-clicking in the transaction area.

4. Click `TEXT`, select `STATE (LEARN STRING)`, then click `OK` to close the `I/O Transaction` box. The previously captured state is sent to the instrument when this `WRITE` transaction is executed.

Uploading and downloading are controlled by the settings in the Direct I/O Configuration dialog box. If Conformance is IEEE 488.2, then VEE will automatically handle learn strings using the 488.2 *LRN? definition. If Conformance is IEEE 488, then `Upload String` specifies the command used to query the state, and `Download String` specifies the command that precedes the state string when downloaded. Figure 3-22 shows an example.

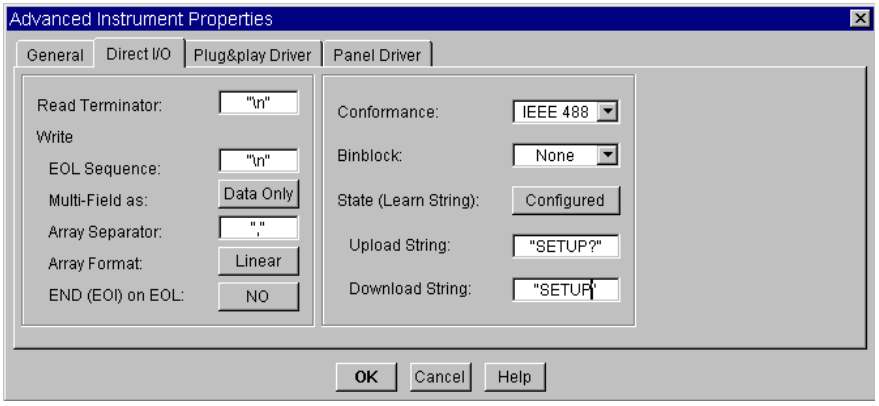


Figure 3-22. Learn String Configuration for HP54100A

Conformance can support IEEE 488 or IEEE 488.2. This example uses the HP 54100A Digitizing Oscilloscope, which conforms to IEEE 488 and requires a `"SETUP?"` to query the learn string and `"SETUP"` to precede the learn string when downloading. When you select `Configured for State (Learn String)` two more fields appear, labeled `Upload String` and `Download String`. The proper strings have been entered in their input fields.

Using PC Plug-in Boards

VEE provides three ways to control PC plug-in boards or cards:

1. ODAS drivers supplied by the PC Plug-in card vendor.

2. Data Translation's Visual Programming Interface. (Order the VPI application directly through Data Translation.)

3. Dynamic link libraries supplied by the PC board manufacturer, such as ComputerBoards or Meilhaus. (See "Using Dynamic Link Libraries" on page 417 for information on using dynamic link libraries.)

Using ODAS Drivers

Follow the manufacturer's instructions to install the PC Plug-in board, install the ODAS driver software, and run the ODAS configuration utility. Then configure the driver in VEE.

1. Select `I/O ⇒ Instrument Manager...`. Select `Find Instruments`. The `Instrument Manager` displays entries similar to those shown in Figure 3-23.

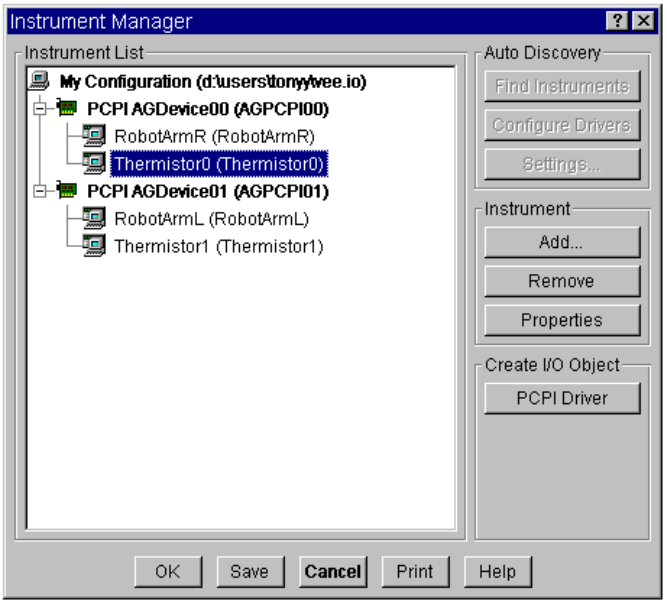


Figure 3-23. ODAS Driver Entries in Instrument Manager

2. Select one of the sub-entries such as `Thermistor0`, and under `Create I/O Object`, select `PCPI Driver`. Click to place the object. It appears as a `Formula` object in VEE as shown in Figure 3-24.

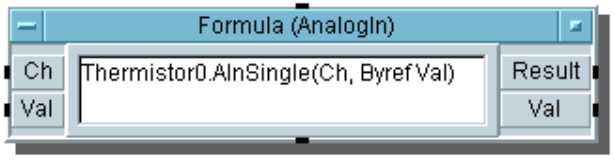


Figure 3-24. PC Plug-in Card with ODAS Driver as Formula Object

For more detailed information about using PC Plug-in Cards with ODAS drivers, refer to the *VEE Pro Advanced Techniques* manual.

Data Translation's Visual Programming Interface (VPI)

Data Translation's VPI works with VEE to create seamless data acquisition performance for PC plug-ins. By leveraging the flexibility of Data Translation's Open Layers standards, you have access to over 50 data acquisition boards.

The VPI works directly with plug-in ISA, PCI, and USB-based data acquisition cards that require low channel count. The VPI adds a menu selection and specific PC plug-in data acquisition icons to VEE. These drive the Data Translation hardware functionality.

Amplicon

Amplicon has a wide range of analog and digital I/O PC plug-in boards within the 200 Series, all with VEE support.

The software interface is part of Amplicon's AmpDIO driver package, a 32-bit API with a multithreaded DLL for Windows and support for interrupt driven acquisition. The API contains over 100 calls for efficient and flexible programming as a Compiled Function using a VEE-specific definition file and the facility to utilize up to eight boards in one program.

In addition to Amplicon's own range of plug-in boards, which includes serial communication devices, Amplicon can supply boards from a wide range of other manufacturers for data acquisition, serial communication, and GPIB applications.

Figure 3-25 shows the VEE runtime software (provided free with Amplicon analog output boards PCI224 and PCI234 and analog input boards PCI230 and PCI260) providing concurrent input and output signals on a PC.

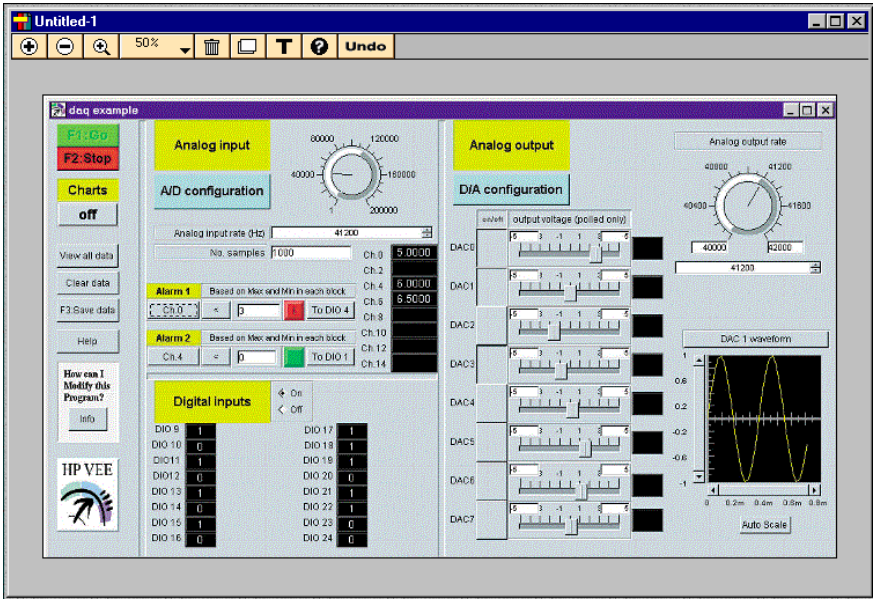


Figure 3-25. Amplicon Data Acquisition Example

ComputerBoards PC Plug-ins

ComputerBoards offers low cost, powerful PC plug-in boards that are compatible with VEE. (For a complete list of supported PC plug-in vendors, see VEE literature or *VEE Pro Advanced Techniques*.)

You simply install the board and its I/O library, and configure the board using a program supplied by the manufacturer. Follow the instructions to connect the board to the device. In VEE, import the library, and you are ready to call the measurement functions in the ComputerBoards I/O library. See the figures below from a demonstration program supplied by the manufacturer.

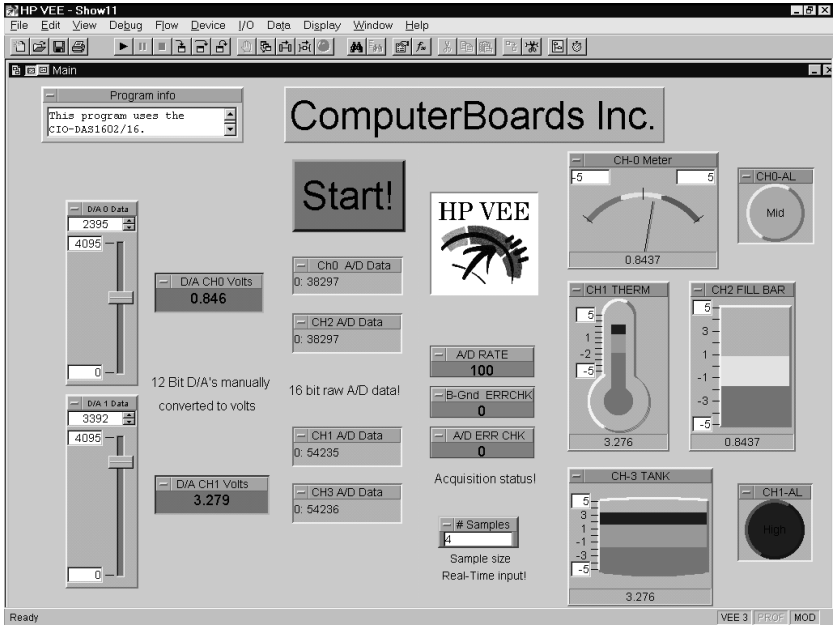


Figure 3-26. VEE Using a ComputerBoards 100 KHz Board

Figure 3-26 shows the panel view of the demonstration program using this 100 KHz A/D board. Figure 3-27 shows VEE importing the ComputerBoards I/O library that made these data acquisition function calls possible.

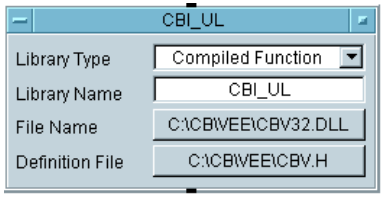


Figure 3-27. Importing the ComputerBoards I/O Library

Meilhaus Electronic ME-DriverSystem

Meilhaus Electronic is one of the leading European designers, producers and sales companies for PC-based data acquisition and interface technology. The ME-DriverSystem for Windows on CD-ROM is included with all data acquisition boards made by Meilhaus Electronic (i.e. ME series). The ME-DriverSystem is also integrated into the VEE menu structure.

After the ME-DriverSystem for VEE is installed, the driver functions appear in a VEE pull-down menu. Figure 3-28 shows the ME Board menu in VEE.

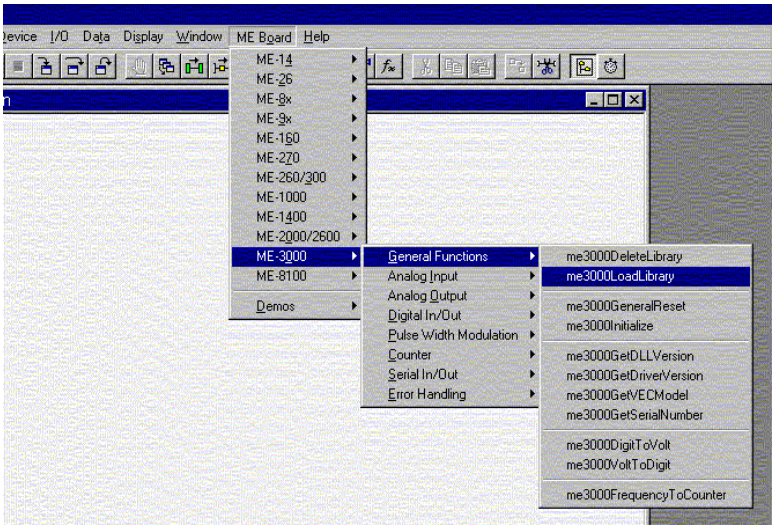


Figure 3-28. ME Board Menu in VEE

The second menu level presents functional groups such as Analog Input and Output, Digital I/O, and special functions of certain boards. Figure 3-29 shows the user panel for data acquisition board ME-3000.

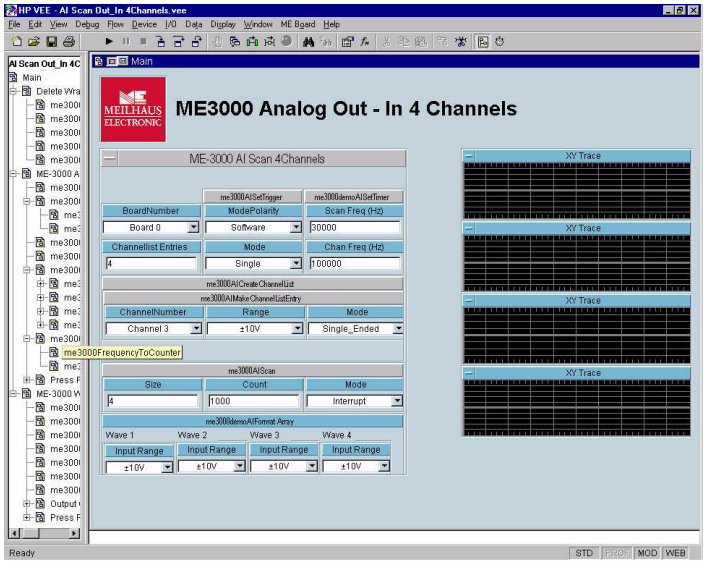


Figure 3-29. User Panel for Data Acquisition Board ME-3000

Finally, in the third menu, the actual functions are located, such as me3000AISingle. Figure 3-30 shows the function panel.

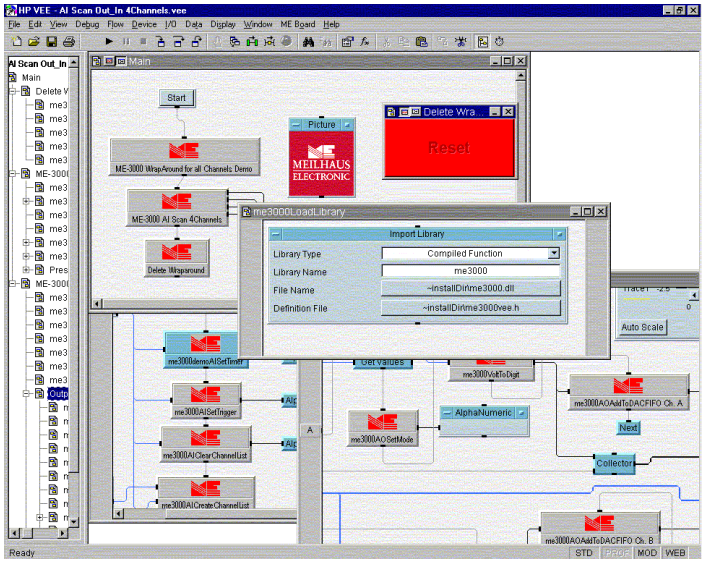


Figure 3-30. Function Panel for ME-DriverSystem

Using a VXI*plug&play* Driver

VXI*plug&play* drivers are issued and supported by the various instrument vendors. These are C-based drivers and are designed for the maximum performance and ease of use.

Agilent VEE is fully VXI*plug&play* compatible. All available VXI*plug&play* drivers from Agilent Technologies ship as a separate product, and are also available on the Web at `http://www.agilent.com/find/inst_drivers`. These same drivers are also included with VEE along with all Agilent Technologies panel drivers. To get VXI*plug&play* drivers for other instruments, contact the instrument vendor.

Lab 3-4: Configuring a VXI*Plug&play* Driver

This example describes how to configure an HPE1412 driver.

1. Select `I/O` ⇒ `Instrument Manager...`.

2. Highlight `My configuration`, then click `Add...` under `Instrument` to get the `Instrument Properties` dialog box. Enter a name, such as `Instrument`, and click `Advanced...` to display the `Advanced Instrument Properties` dialog box.

3. In the `Advanced Instrument Properties` dialog box, toggle `Live Mode:` to `OFF` and select the `Plug&play Driver` folder. Click the `Plug&play Driver Name:` field to display the drop-down menu which lists all the drivers installed on the computer. This example uses the HPE1412 driver, as shown in Figure 3-31.

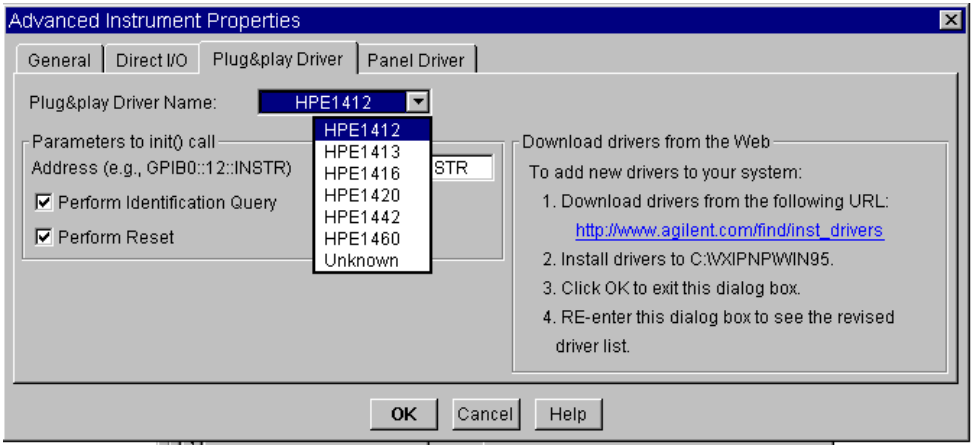


Figure 3-31. Selecting a VXI*plug&play* Driver

4. Select the `HPE1412` driver, click `OK` to return to the `Instrument Properties` dialog box, and click `OK` to return to the `Instrument Manager`. There should now be an entry for `Instrument(@(NOT LIVE))`.

5. Highlight `Instrument(@(NOT LIVE))`, and under `Create I/O Object`, select `Plug&play Driver`. Click to place the object.

Note

In VEE, a VXI*plug&play* driver resembles a Direct I/O object.

To make measurements with the instrument, you need to configure I/O transactions that use C functions in the VXI*plug&play* driver. The driver provides you with panels to pick the right functions to use.

6. Double-click on the transaction bar labeled `<Double-click to Add Function>`, and `Select a Function Panel` is displayed as shown in Figure 3-32. Figure 3-33 shows the hierarchy of functions in the function panel. Notice that `Help` for the item selected is displayed in the dialog box.

Note VEE automatically initializes the instrument. You do not have to use an `init` function, as you would in other languages.

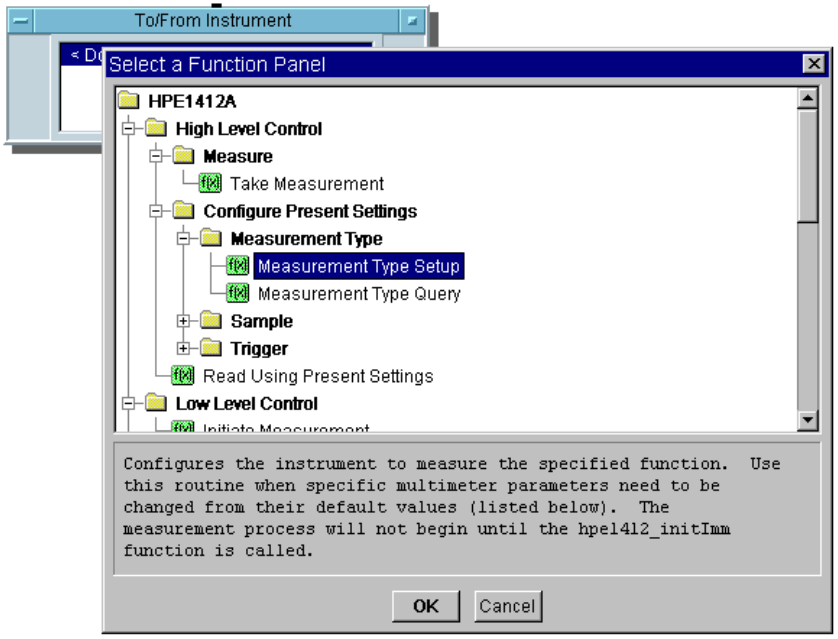


Figure 3-32. Selecting a Function for a VXI*plug&play* Driver

7. Click `Configure Present Settings` ⇒ `Measurement Type` ⇒ `Measurement Type Setup`. The `Edit Function Panel` is displayed. Under `func`, click to display the drop-down list. Select the default `DC Voltage`, as shown in Figure 3-33.

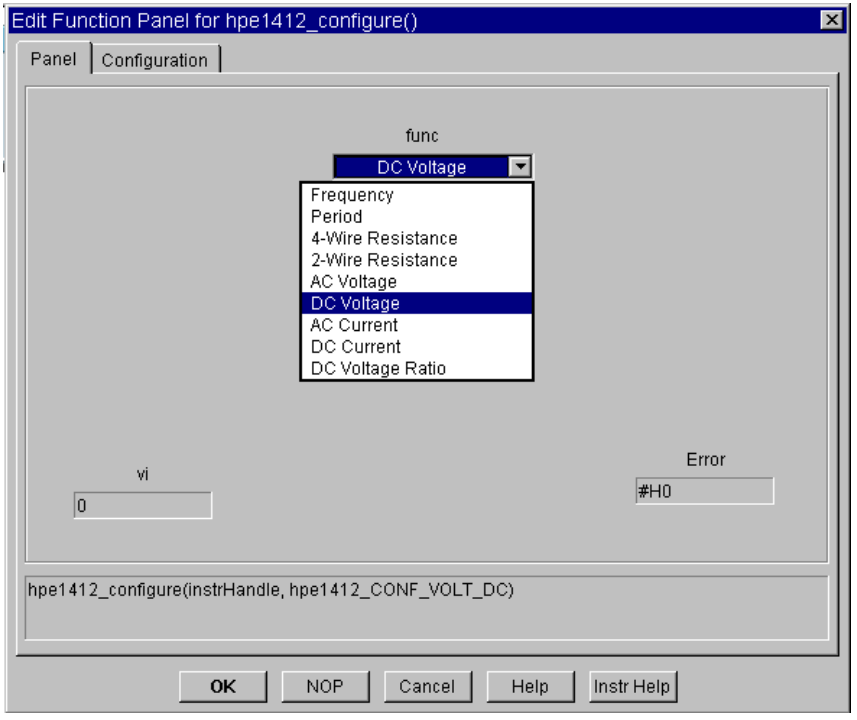


Figure 3-33. The HPE1412 Edit Function Panel

8. Click `OK`. The `To/From Instrument` object now contains an entry for `hpe1412_configure(instruHandle,hpe1412_CONF_VOLT_DC)`, as shown in Figure 3-34.

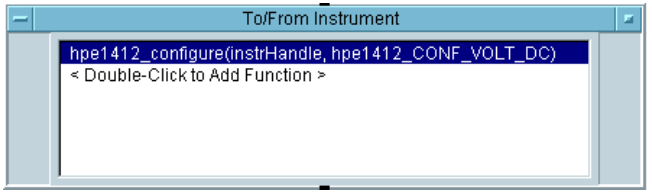


Figure 3-34. DC Voltage Function in VXI*plug&play* Object

9. In the `To/From Instrument` object, double-click to add a function and select `Take Measurement` under `Measure`. Click on the `Configuration` folder to display the dialog box shown in Figure 3-35.

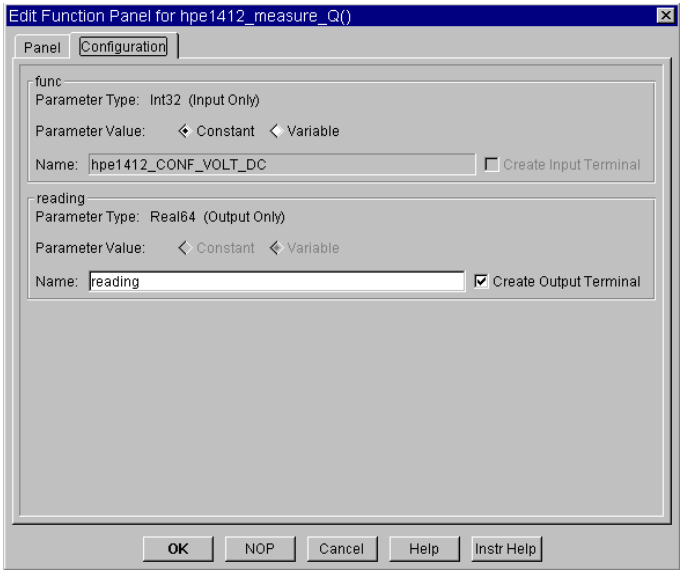


Figure 3-35. Configuration Folder in Edit Function Panel

10. Click `OK`. A second function call is listed in the `To/From Instrument` object as shown in Figure 3-36.

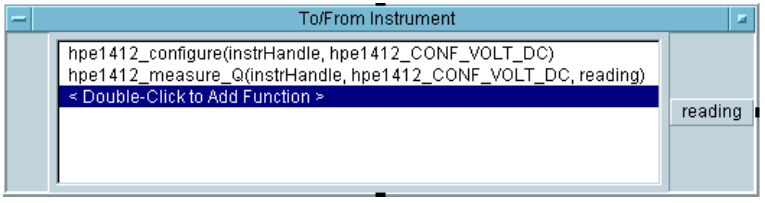


Figure 3-36. HPE1412 Driver Ready for a DC Reading

Other I/O Features

- Explore the full power of VEE's I/O capabilities in the `I/O ⇒ Advanced I/O` submenu: `Interface Operations`, `Instrument Event`, `Interface Event`, and `MultiInstrument Direct I/O`.
- You can display, print, or store bus activity for debugging with the Bus I/O Monitor in the I/O menu.
- VEE includes an ActiveX Automation server to programmatically find instruments. For further information, see the *VEE Pro Advanced Techniques* manual.
- You can also change I/O configurations programmatically at run time. For further information, see the *VEE Pro Advanced Techniques* manual.

Chapter Checklist

You should now be able to perform the following tasks. Review the appropriate topics, if necessary, before going on to the next chapter.

- Explain the benefits of using instrument drivers and Direct I/O.
- Explain the process for controlling instruments.
- Configure an instrument for a state driver.
- Configure an instrument for Direct I/O.
- Change settings on an instrument driver.
- Add and delete component inputs and outputs.
- Move to different panels on an instrument driver.
- Use Direct I/O to write commands to an instrument.
- Use Direct I/O to read data from an instrument.
- Upload and download instrument states using learn strings.
- Use VXI*plug&play* drivers to communicate with an instrument.
- Explain three methods for controlling PC plug-in boards.

4

Analyzing and Displaying Test Data

Analyzing and Displaying Test Data

In this chapter you will learn about:

- VEE data types
- VEE analysis capabilities
- Using math objects
- Using the Formula object
- Using the MATLAB Script object
- VEE display capabilities
- Customizing displays

Average Time to Complete: 1.5 hours

Overview

In this chapter, you will learn about VEE analytical and display capabilities. You will learn how to locate the right math objects for your applications and how to display test results, so that you can turn data into useful information easily and quickly.

You can also use other familiar applications such as MS Excel to analyze the data using ActiveX Automation. (For more information, refer to Chapter 6, “Creating Reports Easily Using ActiveX,” on page 247.) You can use display capabilities external to VEE using ActiveX controls. (For more information, refer to “Using an ActiveX Control” on page 396). This chapter focuses on VEE's own core set of tools and the MATLAB Script object included with VEE.

Agilent VEE Data Shapes and Data Types

In a VEE program, data is transmitted across the lines between objects and is then processed by subsequent objects. In order to specify a set of data, VEE packages it into a container that has both a *data shape* (*scalar* or *array*) and a *data type* (such as `Int32`, `Real64`, or `Text`).

Data Shape: A *scalar* is a single number including numbers expressed as two or more components such as complex numbers, and an *array* contains a group of data items that can be specified as one dimensional (Array 1D), two dimensional (Array 2D), etc.

Data Types: The VEE data types are described in Table 4-1.

In general, you will not be concerned with data types or shapes, because most objects operate on any VEE data type and will automatically convert data to the type required for that object. For example, if a Magnitude Spectrum display receives a Waveform data type, VEE automatically performs a Fast Fourier Transform to convert it from the time domain into the frequency domain.

Occasionally, however, an object requires a particular data type so it is good to be aware of them. You will also want to be aware of the differences in supported data types between VEE and MATLAB. (For more information, refer to the section “Working with Data Types” on page 191.)

The following are brief descriptions of VEE data types that you can read through quickly. Issues involving using these data types is explained in subsequent chapters.

Table 4-1. Agilent VEE Data Types

Data Type	Description
UInt8	Unsigned byte 0 to 255.
Int16	A 16-bit two's complement integer (-32768 to 32767).
Int32	A 32-bit two's complement integer (-2147483648 to 2147483647).
Real32	A 32-bit floating point number that conforms to the IEEE 754 standard (+/-3.40282347E+/-38).
Real64	A 64-bit floating point number that conforms to the IEEE 754 standard (+/- 1.797693138623157 E308).
PComplex	A magnitude and phase component in the form (mag, @phase). Phase is set by default to degrees, but can be set to radians or gradians with the File ⇒ Default Preferences ⇒ Trig Mode setting.
Complex	A rectangular or Cartesian complex number having a real and imaginary component in the form (real, imag). Each component is Real64. For example, the complex number 1 + 2i is represented as (1,2).
Waveform	A composite data type of time domain values that contains the Real64 values of evenly-spaced, linear points and the total time span of the waveform. The data shape of a Waveform must be a one-dimensional array (Array 1D).
Spectrum	A composite data type of frequency domain values that contains the PComplex values of points and the minimum and maximum frequency values. The domain data can be mapped as log or linear. The data shape of a Spectrum must be a one-dimensional array (Array 1D).

Table 4-1. Agilent VEE Data Types (Continued)

Data Type	Description
Coord	A composite data type that contains at least two components in the form (x,y,...). Each component is Real64. The data shape of a coord must be a Scalar or an Array 1D.
Enum	A text string that has an associated integer value. You can access the integer value with the ordinal(x) function.
Text	A string of alphanumeric characters.
Record	A composite data type with a field for each data type. Each field has a name and a container, which can be of any type and shape (including Record).
Object	Used only for ActiveX Automation and Controls, a reference to an ActiveX control or a reference returned from an Automation call. Literally, this is a reference to an IDispatch or IUnknown interface.
Variant	Used only for ActiveX Automation and Controls, a data type that is required for some ActiveX method calls as a By Ref parameter type.

Note

Investigate `I/O ⇒ To/From Socket` for sharing data in mixed environments.

Agilent VEE Analysis Capabilities

VEE supports common math operations and hundreds of other functions. In addition, VEE also includes the MATLAB Script feature. The MATLAB Script feature is a subset of the standard full-featured MATLAB from The MathWorks. It provides additional mathematical capabilities in VEE including Signal Processing, advanced mathematics, data analysis, and scientific and engineering graphics. The MATLAB Script feature is fully integrated with VEE, and you can include MATLAB Script objects in any VEE program.

If neither VEE nor MATLAB have a math function you need, you still have several options available. You can create the function with the `Formula` object, which is discussed later in this chapter, you can write the function in a compiled language such as C and link it to VEE, or you can communicate with another software application from VEE.

Using Built-In Math Objects

In the VEE `Device ⇒ Function & Object Browser`, you can access built-in (preprogrammed) mathematical expressions for both VEE and MATLAB.

Accessing a Built-in Operator or Function

To access VEE mathematical operators and functions, select `Device ⇒ Function & Object Browser`. For example, to create a formula that returns a random number in a specified range, select `Type: Built-in Functions`, `Category: Probability & Statistics`, and `Functions: random`, as shown in Figure 4-1.

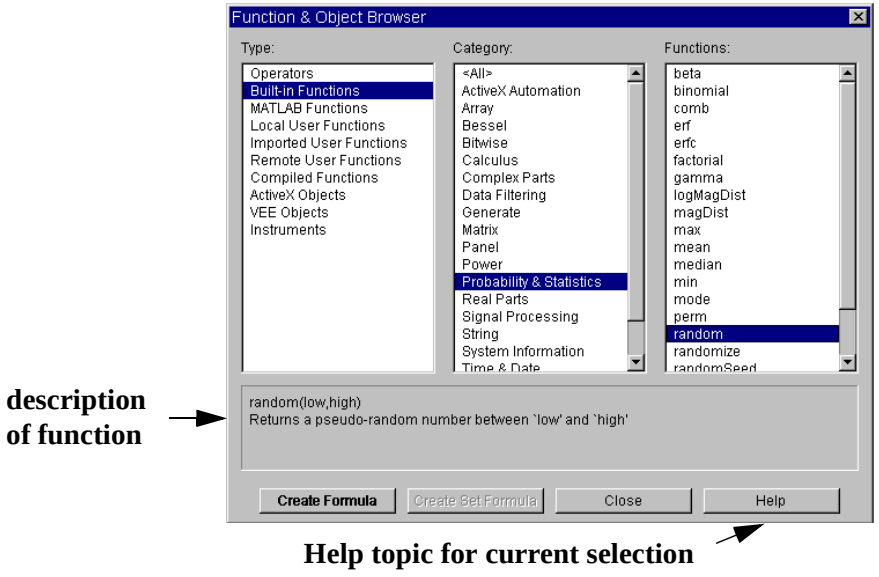


Figure 4-1. A VEE Function in the Function & Object Browser

Notice that the `Function & Object Browser` displays a brief description of the current selection, as shown in Figure 4-1. You can also click on the `Help` button for a more detailed description of the current selection and get information such as the definition, use, syntax and examples.

To access MATLAB operators and functions, select `Device ⇒ Function & Object Browser`, and under `Type:` select `MATLAB Functions`. For example, to convert roots to polynomials, select `Type: MATLAB Functions`, `Category: Interpolation & Polynomials`, and `Functions: poly` as shown in Figure 4-2.

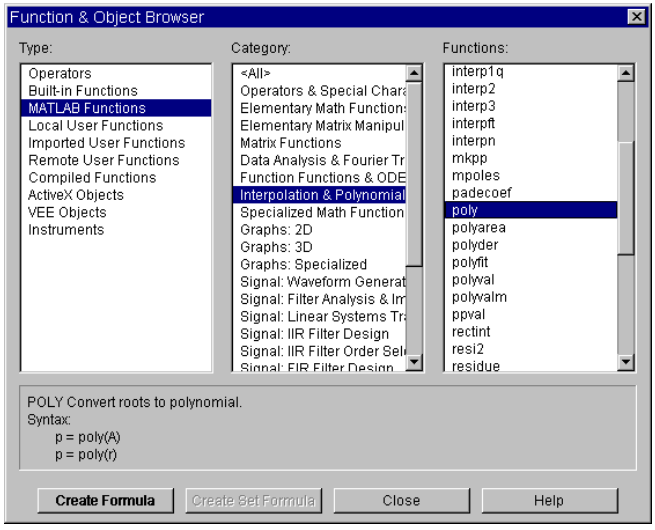


Figure 4-2. A MATLAB Function in the Function & Object Browser

Again, a brief description of the current selection is displayed in the `Function & Object Browser`, and clicking on `Help` will display a more detailed description about the current selection. The MATLAB Runtime Engine and Script is discussed more in the section “Using MATLAB Script in Agilent VEE” on page 187.

Lab 4-1: Calculating Standard Deviation

Generate a cosine waveform of at a frequency of 1 kHz, amplitude of 1 V, a time span of 20 ms, represented by 256 points. Calculate its standard deviation and display it.

1. Select `Device ⇒ Virtual Source ⇒ Function Generator`. Set the `Frequency` appropriately and iconize it.

2. Select `Device` ⇒ `Function & Object Browser`, then select `Built-in Functions`, `Probability & Statistics`, and `sdev`. Click `Create Formula`.

Note You can go directly to the Function & Object Browser dialog box by pressing the ***fx*** icon on the tool bar, shown in Figure 4-3, or by pressing **Ctrl-I**.

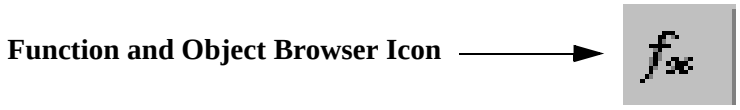


Figure 4-3. Opening Function and Object Browser from fx Icon

3. Open the object menu for `sdev()` to consult `Help`.

Note The `sdev(x)` object is defined as the square root of the variance of `x`, and `x` may be of the type UInt8, Int16, Int32, Real32, Real64, Coord, or Waveform. The Function Generator outputs a Waveform data type.

4. Connect the `Function Generator` to `sdev(x)`.

5. Select `Display` ⇒ `AlphaNumeric` and connect it to the `sdev(x)` data output pin.

6. Run the program. It should look like Figure 4-4.

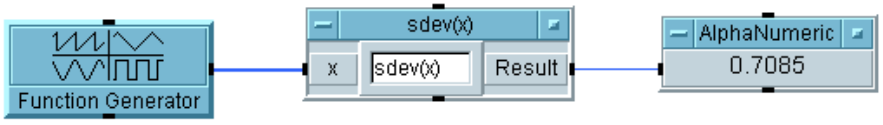


Figure 4-4. Calculating Standard Deviation

Creating Expressions with the Formula Object

The `Formula` object can be used to write mathematical expressions in VEE. The variables in the expression are the data input pin names or global variables. The result of the evaluation of the expression will be put on the data output pin.

Figure 4-5 shows a `Formula` object. The input field for the expression is in the center of the object. A default expression `(2*A+3)` indicates where to enter the formula. Just double-click the field to type in a different expression.

Note

You can type in a `Formula` expression on more than one line. If a `Formula` expression contains a **Return**, it is interpreted as a multi-line single expression. If a `Formula` contains statements separated by semi-colons (`;`), they are interpreted as multiple expressions in the `Formula`.

You can also use standard editing commands to edit expressions in a `Formula`. For example, you can drag the mouse to highlight characters, use **Ctrl-C** for copying the characters, **Ctrl-V** for pasting, and **Ctrl-X** for deleting.

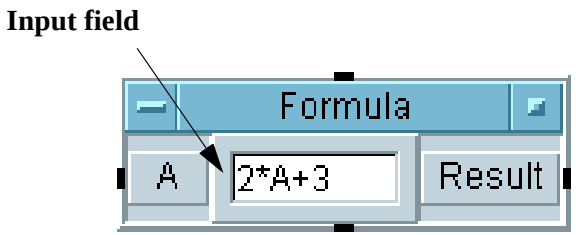


Figure 4-5. The Formula Object

Note All the functions created from the `Devices ⇒ Function & Object Browser Built-in` type are simply Formulas that already have their expressions set appropriately. They can be modified to combine functions and add (or delete) inputs. You can also do multiple-line entry in the `Formula` object, and assign values to output terminals.

Evaluating an Expression with the Formula Object

In this example, you will evaluate the expression, `2*A^6-B`, where `A=2` and `B=1`. (Notice the `^` sign for exponentiation.)

Note The variable names are *not* case-sensitive.

1. Select `Device ⇒ Formula`. Click the `Formula` input field and type `2*A^6-B`.

2. Place the mouse pointer over the data input area (but not right over the `A` input) and press **Ctrl-A** to add an input pin.

Note It will be labeled `B` by default, but you can rename it.

3. Select `Data ⇒ Constant ⇒ Int32`, clone it by selecting `Clone` from the object menu, and connect the two Int32 objects to the Formula inputs `A` and `B`.

4. Enter `2` in the `A Int32` input box and `1` in the `B Int32` input box.

5. Select `Display ⇒ AlphaNumeric` and connect it to the output of Formula, and run the program. It should display the result 127, as shown in Figure 4-6.

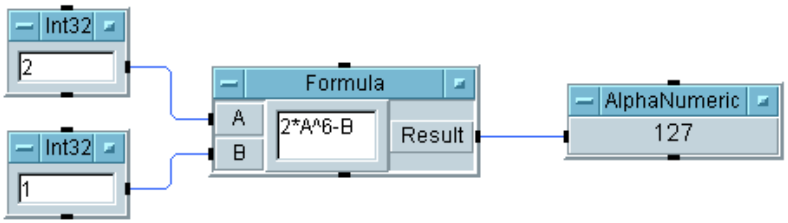


Figure 4-6. Evaluating an Expression

Using an Agilent VEE Function in the Formula Object

This example generates a cosine wave and calculates the standard deviation and root mean square using the `Formula` object.

1. Select the `Function Generator`, `Formula`, and `AlphaNumeric` objects and connect them together using their data pins.

2. Clone the `Formula` object by opening the object menu and selecting `Clone`, and place it just below the first one. Connect the `Function Generator` data output pin to the second `Formula` object.

3. Clone another AlphaNumeric display and connect it to the second `Formula` object.

4. Enter `sdev(A)` in the first `Formula` object, and `rms (A)` in the second `Formula` object.

   `sdev(A)` and `rms (A)` are the two math functions from the `Device` ⇒ `Function & Object Browser` dialog box. Notice that they can be called as functions or independent objects, and they will perform in the same way.

5. Run the program. The program displays the same answers when these functions are put into the `Formula` object as it did when they were used as independent objects, as shown in Figure 4-7.

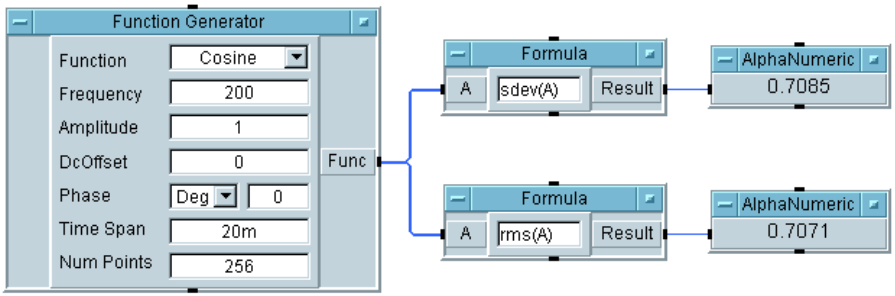


Figure 4-7. Formula Examples Using VEE Functions

Now calculate the standard deviation and root mean square using only one `Formula` object. Formulas can have multiple output terminals with values assigned to them.

6. Double-click the object menu to `Cut` one of the `Formula` objects.

7. In the remaining `Formula` object, change the expression to
   ```
   B=sdev(A);
   C=rms(A)
   ```

Note

When a `Formula` object contains multiple expressions, you must put a semicolon at the end of an expression to distinguish it from the next expression. For example, in the formula `B=sdev(A);` the semi-colon indicates the end of the expression.

Note

You can put *line breaks* at any point in a `Formula` object. The formula is read as one expression as long as there are no semi-colons. For example, you could enter a single expression as

```
B=sdev
(A)
```

You can also add spaces in the formula to improve readability.

8. In the `Formula` object, add an output terminal. Rename the output terminals `B` and `C`. Connect output terminal `B` to one of the `Alphanumeric` objects, and output terminal `C` to the other `Alphanumeric` object.

9. Run the program. It should look like Figure 4-8.

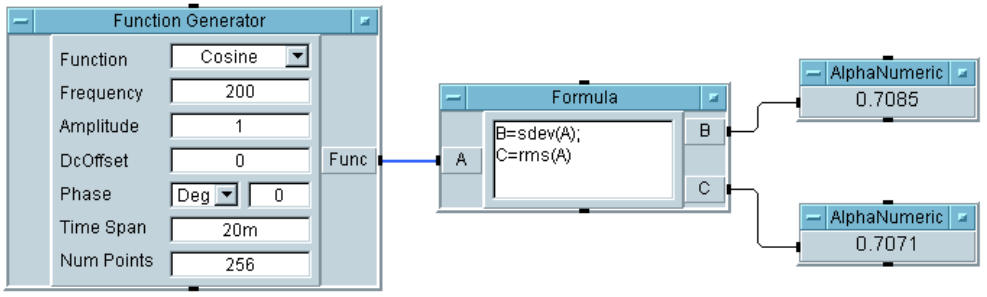


Figure 4-8. VEE Functions Using One Formula Object

On Your Own

Complete the following exercises and check the results as shown in Figure 4-9.

1. Create an array of numbers from 1 to 2048 using the `ramp` object in the `Generate` category of `Built-in Functions`. Calculate the standard deviation of this array and display it.

2. Do the same exercise described in the previous step, using the `ramp()` function in a `Formula` object instead of the `ramp` object.

3. Do the same exercise described in the previous step by nesting the functions. Use only two objects.

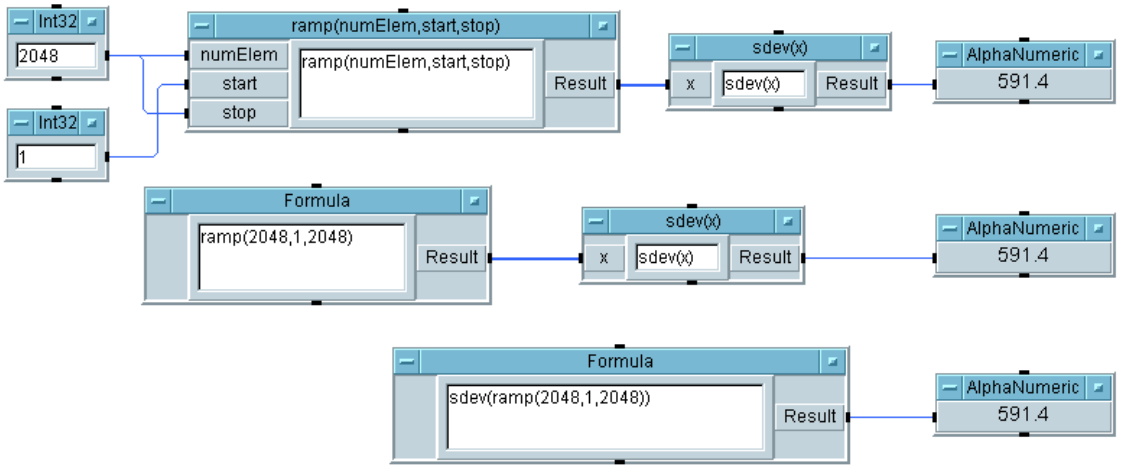


Figure 4-9. On Your Own Solution: Ramp and SDEV

For the second and third exercises, you have to delete the input terminal `A` on the `Formula` object to avoid an error message, because all data input pins must be connected and have data before an object can operate.

Using MATLAB Script in Agilent VEE

VEE includes the MATLAB Script object, which gives you access to the functionality of MATLAB. VEE can pass data to the MATLAB Script Engine and receive data back, enabling you to include MATLAB mathematical functions in VEE programs.

Note

If you already have MATLAB installed, VEE will use your installed MATLAB to process MATLAB Script. However, if you do not have the Signal Processing Toolbox, you will not be able to use those functions from VEE unless the MATLAB Script Engine that ships with VEE is registered. To register MATLAB, change directory (CD) to <*VEE_installation_dir*>`\MATLAB\bin` and execute `MATLAB.exe /regserver`.

Some uses of the MATLAB Script object include:

- Letting MATLAB operate on VEE-generated data.
- Returning results from the MATLAB Script object and using the results in other parts of the VEE program.
- Performing sophisticated filter design and implementation in the MATLAB Script object by using MATLAB's Signal Processing Toolbox functionality.
- Visualizing data using 2-D or 3-D graphs.

Figure 4-10 shows how the MATLAB Script object appears in a VEE program. When the MATLAB Script program executes, it generates the data shown in the `Alphanumeric` object.

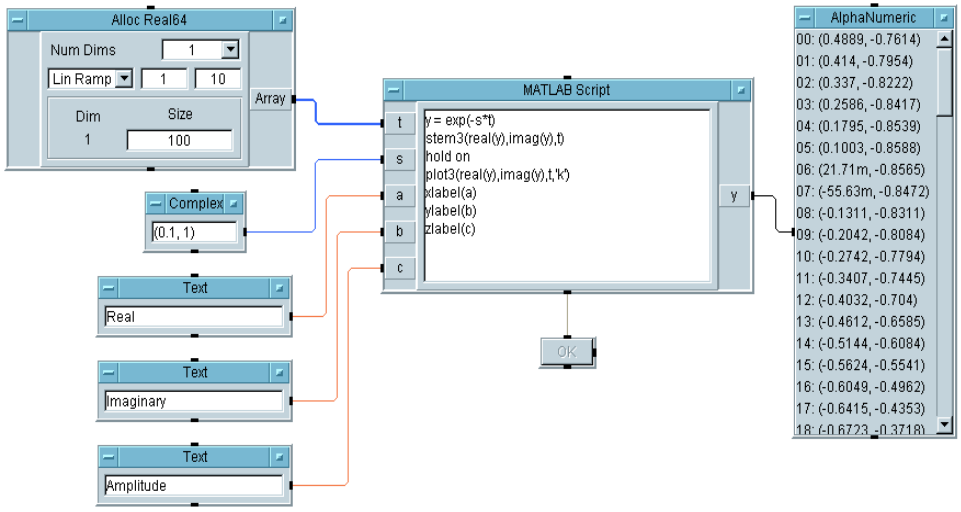


Figure 4-10. MATLAB Script Object in a VEE Program

Figure 4-11 shows the graph that is produced when the program runs.

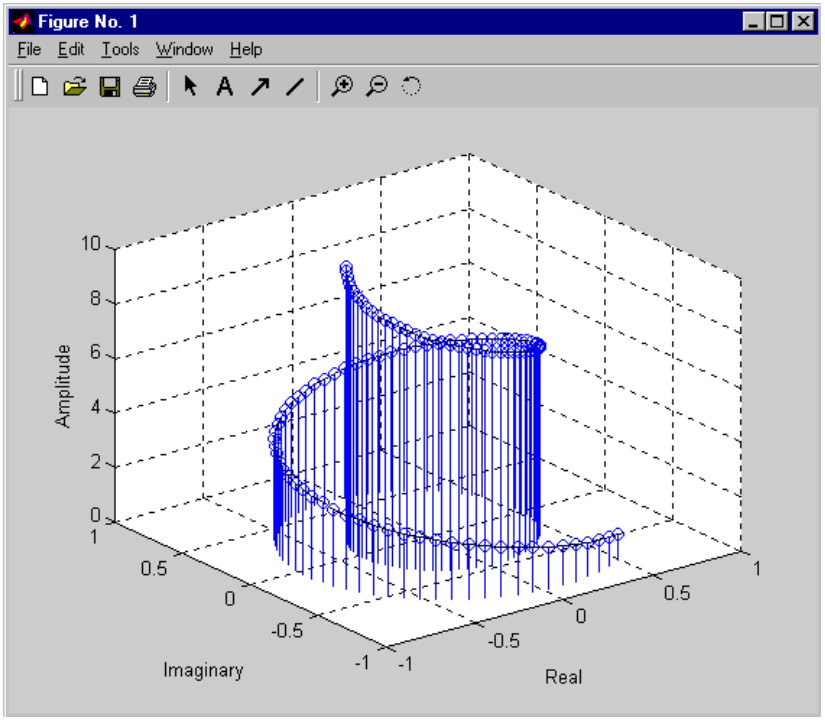


Figure 4-11. Graph Generated by the Program

When you include MATLAB Script objects in a VEE program, VEE calls the MATLAB Script Engine to perform the operations in the MATLAB Script objects. Information is passed from VEE to MATLAB and back again. Some notes about MATLAB:

- The first MATLAB Script object that executes in a program opens a single MATLAB session. All other instances of MATLAB Script objects share the session. MATLAB Script objects can therefore share global variables in the MATLAB workspace.

- VEE does *not* perform any syntax checking of MATLAB commands before the MATLAB Script Engine is called. Errors and warnings generated by MATLAB are shown in the regular VEE dialog boxes, just like any other VEE error or caution.

- Unlike VEE, MATLAB is case sensitive. If you name a MATLAB Script object input or output terminal with a capital `X`, be sure to use a capital `X` in MATLAB, not lower-case `x`.

- Only some VEE data types are allowed as MATLAB script inputs. This is discussed in more detail in a following section.

Including a MATLAB Script Object in Agilent VEE

When you use a MATLAB object in a VEE program, it looks like a VEE `Formula` object. There are two ways to add a MATLAB Script object to a program:

1. Select `Device` ⇒ `MATLAB Script` and click to place the object in the program. This creates a default MATLAB Script object that you can edit for your purposes.

   -*OR*-

   Select Device ⇒ `Function & Object Browser`, and select `Type: MATLAB Functions`. Choose a predefined MATLAB function and click `Create Formula`. Click to place the object in the program. Figure 4-12 shows some predefined MATLAB functions that could be added to a VEE program.

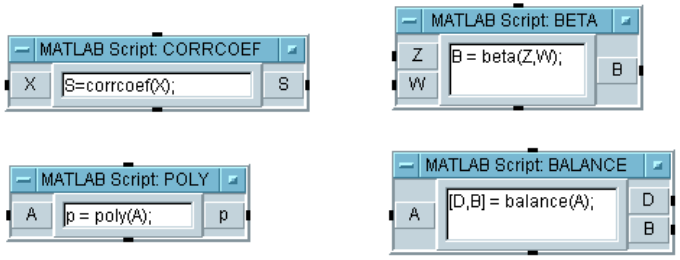


Figure 4-12. Adding Predefined MATLAB Objects to a VEE Program

Notice that each object is named `MATLAB Script<`*function name*`>` to help you distinguish it from other VEE formula objects. Each object already includes the function it will perform, and the input and output pins that are likely to be needed, just like built-in VEE formula objects. You can also edit MATLAB Script objects exactly as you can edit any other VEE object.

Note

For more information about MATLAB functions, from the main VEE window, select `Help` ⇒ `MATLAB Script` ⇒ `Help Desk`.

Working with Data Types

Only a subset of the VEE data types are supported as inputs and outputs of MATLAB objects.

VEE automatically converts some one-dimensional arrays to make it more convenient for programs that contain both VEE and MATLAB functions. For example, a VEE one-dimensional text array will automatically convert to a two-dimensional character array when it is input to a MATLAB Script object, and a character one-dimensional array from a MATLAB Script object will automatically convert to a Text Scalar when it is output from the MATLAB Script object.

Note

For a complete listing and description of the automatic conversions between VEE data types and MATLAB data types, refer to the VEE online `Help`.

You can also use input terminal data type constraints to ensure that the data input from another object is converted to a supported type, as shown in the following example.

1. Select `Data` ⇒ `Constant` ⇒ `Int32` and click to place the object. Change the value to 7. `Clone` the object and place the second `Int32` under the first. Change its value to 20.

2. Select `Device` ⇒ `MATLAB Script` and place the object to the right of the constant objects.

3. Select `Display Alphanumeric` and place it to the right of the `MATLAB Script` object.

4. Connect the output pin from the top `Int32` object to the input pin `A` of the `MATLAB Script` object. Connect the output pin from the bottom `Int32` object to the input pin `B` of the `MATLAB Script` object. Connect the output pin from the `MATLAB Script` object to the input pin of the `Alphanumeric` object.

   Run the program. It generates a VEE Runtime Error stating the expected input was a `Real64`, `Complex`, `Waveform`, or `Text`, and `Int32` input was received instead.

To avoid errors like this, change the input terminal data type on the MATLAB Script object.

5. Double-click on terminal `A` to open the `Input Terminal Information` dialog box. Click on `Required Type:` to display the drop-down menu, select `Real64`, and click `OK`. Double-click on terminal `B` and change it to a `Real64` as well, as shown in Figure 4-13.

6. Run the program. Now the `Int32` data is automatically converted to `Real64` on the input pin before it is passed to MATLAB.

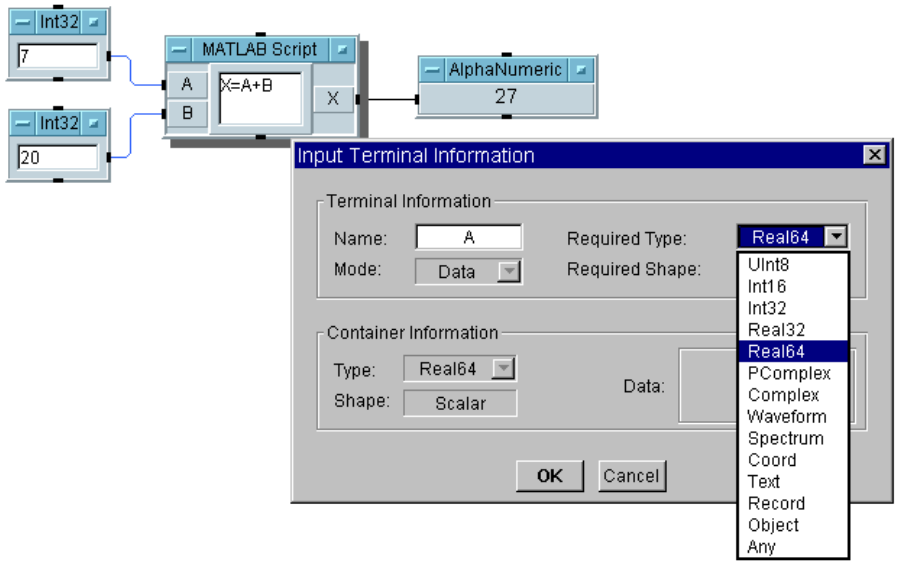


Figure 4-13. Changing Input Terminal Data Type

Displaying Test Data

Table 4-2 describes the display capabilities for the different VEE objects.

Table 4-2. Displays

Display	Description
Alphanumeric	Display values as text or numbers. Requires SCALAR, ARRAY 1D, or ARRAY 2D.
Beep	Gives an audible tone to highlight a place in your program.
Complex Plane	Displays Complex, Polar Complex (PComplex), or Coord data values on a Real vs. Imaginary axis.
Indicator=>> Meter, Thermometer, Fill Bar, Tank, Color Alarm	All of these indicators display numbers with a graphical representation suggested by their names. They all have color-coded ranges - usually three, but the meter has five. The Color Alarm can simulate an LED with a text message flashing up on the alarm in each range.
Label	An object used to put a text label on the Panel View. The colors and fonts may be easily adjusted through Properties... in the object menu while in the Panel View.
Logging Alphanumeric	Displays values as text or numbers when repeatedly logged. Requires SCALAR or ARRAY 1D.
Note Pad	Uses a text note to clarify a program.
Picture (PC)	An object used to put a graphic image on the Panel View. The formats supported are: *.BMP (bitmaps), *.GIF (GIF87a and GIF89), *.ICN (X11 bitmap), *.JPEG, *.PNG, and *.WMF (Windows Meta File)
Picture (UNIX)	*.GIF (GIF87a) and *.xwd (X11 Window Dump)

Table 4-2. Displays (Continued)

Display	Description
Polar Plot	Graphically displays data on a polar scale when separate information is available for radius and angle data.
Spectrum (Freq)	A menu that contains frequency domain displays: Magnitude Spectrum, Phase Spectrum, Magnitude vs. Phase (Polar), and Magnitude vs. Phase (Smith). Inputs must be Waveform, Spectrum, or an array of Coords. Waveform inputs are automatically changed to the frequency domain with a Fast Fourier Transform (fft).
Strip Chart	Graphically displays the recent history of data that is continuously generated while the program runs. For each y input value, the x value is incremented by a specified Step size. When new data runs off the right side of the display, the display automatically scrolls to show you the latest data.
Waveform (Time)	Graphically displays Waveforms or Spectrums in the real time domain. Spectrums are automatically converted to the time domain using an Inverse Fast Fourier Transform (ifft). The x axis is the sampling units of the input waveform.
X vs. Y Plot	Graphically displays values when separate data information is available for X and Y data.
XY Trace	Graphically displays mapped arrays or a set of values when y data is generated with evenly-spaced x values. The x value that is automatically generated depends on the data type of the trace data. For example, a Real trace would generate evenly-spaced Real x values; whereas, a Waveform trace would generate x values for time.

Customizing Test Data Displays

Displays may be customized in a variety of ways. Not only can you label, move and size displays like all VEE objects, but you can also change the x/y scales, modify the traces, add markers, or zoom in on parts of the graphical display.

The following example illustrates some of these features. It uses the `Noise Generator` to generate a waveform, and then displays it with the `Waveform (Time)` display. The example also describes how to change the `X` scale, zoom in on a wave segment, and use the markers to measure the distances between points on the waveform. The same principles may be applied to all the graphical displays.

Displaying a Waveform

1. Select `Device` ⇒ `Virtual Source` ⇒ `Noise Generator`.
2. Select `Display` ⇒ `Waveform (Time)`.
3. Connect the data output of the `Noise Generator` to the data input of `Waveform (Time)` and run the program. It should look like Figure 4-14.

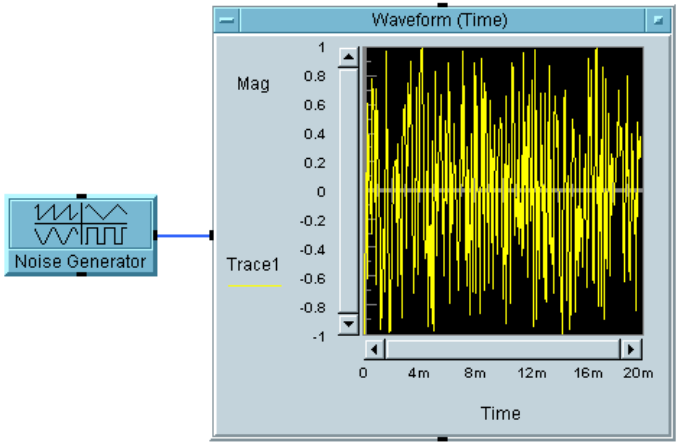


Figure 4-14. Displaying a Waveform

Changing the X and Y Scales

1. Double-click the `Waveform (Time)` title bar to get the `Y Plot Properties` box, select the `Scales` folder, select `20m` for the `X Maximum` and enter `1m`.

   This alters the time span of the display from 20 milliseconds to 1 millisecond.

2. Double-click the `Minimum` field on the `Y` axis where it says `-1`, and enter `-.5`. Click `OK`.

Zooming in on Part of the Waveform

1. Open the `Waveform (Time)` object menu and click `Zoom` ⇒ `In`.

   The cursor becomes a small right angle. By clicking and dragging, you can draw a square on the graph outlining the area you want to enlarge.

2. Outline an area of the waveform including several peaks, and release the mouse button.

The display zooms in to this selected area of the waveform. Notice the `x` and `y` scales change automatically.

Adding Delta Markers to the Display

1. Move to the open view on the `Noise Generator`.

   a. Change the `Num Points` setting to `16`. Run the program again.

   b. Open the `Waveform (Time)` object menu and select `Properties` (or just double-click on the title bar), then under `Markers`, click `Delta`. Then click `OK`.

Note

You can get and set the values of the markers at runtime. See the online `Help` topic under `Contents and Index` ⇒ `How Do I...` ⇒ `Display Data` for more information.

You will see two white arrows pointing up and down at one of the data points on the waveform. Also, notice that the display records the `x` and `y` coordinates of these markers at the bottom of the display. To measure the `x` or `y` distance between two peaks, click-and-drag the arrows to the peaks you want to measure. You will see one of the markers move to those new peaks with the new coordinates recorded at the bottom of the display, as shown in Figure 4-15.

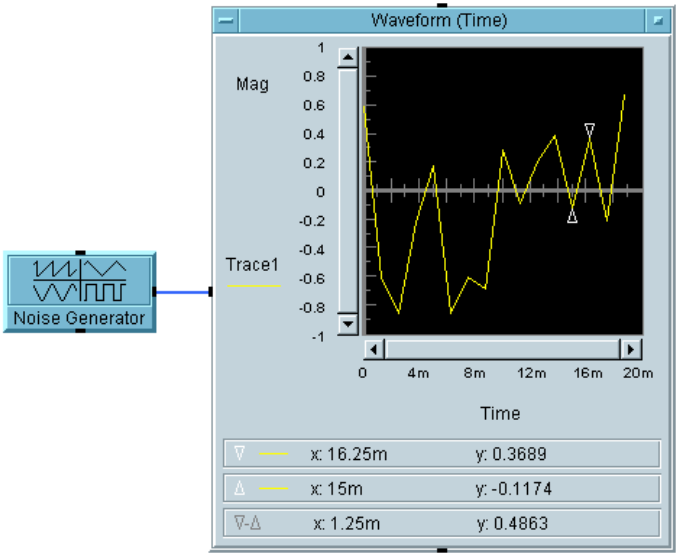


Figure 4-15. Delta Markers on a Waveform Display

VEE will automatically interpolate between waveform data points. Open the object menu, select `Properties`, then under `Markers`, click `Interpolate`.

Changing the Color of the Trace

1. Double-click the title bar to get to `Properties`, then click the `Traces` folder tab.

   You can select the color, line type, and point type for the Trace selected in this folder.

Note

You can also change these values at run time by using the `Traces` or `Scales` control inputs. For more information, see the *VEE Pro Advanced Techniques* manual.

2. Click `OK` for the selected color. Then click `OK` to exit the `Properties` box.

The trace will now be displayed in the new color. Other display characteristics such as Panel Layout, Grid Type, Clear Control, and Add Right Scale may be customized in a similar fashion as the features in the exercise above.

Note

VEE also includes `Plot` in the display object menus, which allows you to plot test results on the display without printing out the rest of the program.

For Additional Practice

To learn about other VEE objects and gain more practice, do the exercises in Appendix A, “Additional Lab Exercises,” on page 467. Solutions are provided with a discussion of key points.

Chapter Checklist

You should now be able to do the following tasks. Review topics as needed, before going on to the next chapter.

- Describe the main data types in VEE.
- Describe some of the main areas of analytical capabilities in VEE.
- Find an online `Help` explanation for any object in the `Function & Browser` dialog box.
- Describe the relationship between input pins and variables in a VEE math object.
- Evaluate a mathematical expression using the `Formula` object, and then evaluate two expressions using the `Formula` object. (Remember to use a semicolon after the first line.)
- Use a VEE function in a mathematical expression in the Formula object.
- Use the MATLAB Script object.
- Describe major display capabilities in VEE.
- Customize a graphical display in terms of the scales used, the part of the waveform seen, the markers used, and the color of the trace.

5

Storing and Retrieving Test Results

Storing and Retrieving Test Results

In this chapter you will learn about:

- Putting test data into arrays
- Using the `Collector` object
- Using the `To/From File` objects
- Creating mixed data types using `Records`
- Performing search and sort operations using `DataSets`
- Creating simple test databases using the `Dataset` objects

Average Time to Complete: 2 hours

Overview

In this chapter, you will learn the fundamentals of storing and retrieving test data. You will create arrays of the right data type and size to hold your test results, and then access the data or part of the data for analysis or display.

This chapter also describes the `To/From File` objects, the Record data type, and Dataset files. The `To File` and `From File` objects read data to and from files based on I/O transactions. The `Record` data type can be used to store several types of data in a single structure. You can use the `Dataset` to store one or more records in a file, and perform search and sort operations on datasets.

Note

The `To File` object is also described in "Using Data Files" on page 87 of Chapter 2, "Agilent VEE Programming Techniques."

Using Arrays to Store Test Results

Data types can be stored in two ways:

- Scalar values (that is, a single number such as 9 or (32, @10))

-OR-

- Arrays from 1 to 10 dimensions.

Note The overview of VEE data types is described in Chapter 4, "Analyzing and Displaying Test Data."

Indexing for arrays is zero-based in VEE, and brackets are used to indicate the position of the array element. For example, if the array `A` holds the elements `[4, 5, 6]`, then

`A[0] = 4`, `A[1] = 5`, and `A[2] = 6`

The syntax for arrays is as follows:

colon	Used to indicate a range of elements. For instance, `A[0:2]` = `[4, 5, 6]` in the array above.
asterisk (*)	a wildcard to specify all elements from a particular array dimension. `A[*]` returns all elements of array `A`.
commas	In the subarray syntax, commas are used to separate array dimensions. If `B` is a two-dimensional array with three elements in each dimension, `B[1,0]` returns the first element in the second row of `B`.

The syntax to access elements of an array can be used in the `Formula` object or any expression field, such as those in the `To/From File` object.

Lab 5-1: Creating an Array for Test Results

The easiest way to create an array is to use the `Collector` object.

This exercise uses the `For Count` object to simulate four readings from an instrument. The readings are put into an array and the results are printed. The principles will be the same regardless of the data type or the size of the array, since the `Collector` will take any data type and create the array size automatically depending on the number of elements sent.

1. Select `Flow` ⇒ `Repeat` ⇒ `For Count`, `Data` ⇒ `Collector`, and `Display` ⇒ `AlphaNumeric`.

   about the For Count object

   `For Count` outputs increasing integer values starting at 0 depending on the number of iterations you specify in the input field. Highlight the default number `10` by double-clicking, then type `4`. `For Count` will output `0`, `1`, `2`, and `3`.

   about the Collector object

   The `Collector` receives data values through its Data input terminal. When you finish collecting data, you "ping" the `XEQ` terminal to tell the `Collector` to construct the array and output it. You can use the `For Count` sequence output pin to ping the `Collector XEQ` pin. The `Collector` displays a button that toggles between a `1 Dim Array` and *n+1* `Dim Array`.

   Double-click the `Collector` to get the open view, and read through `Help` in the object menu to understand the object.

2. Click `n+1 Dim` in the `Collector` to change the selection to `1 Dim Array`.

3. Connect the `For Count` data output pin to the `Data` input pin on the `Collector`.

4. Connect the `For Count` sequence output pin to the `XEQ` input pin on the `Collector`.

The XEQ pin, a special trigger pin that exists on several different objects, determines when the object executes. In this case, you want the object to fire after all of the data for the array has been collected.

5. Connect the `Collector` data output pin to the `AlphaNumeric` data input pin.

6. Enlarge `AlphaNumeric` to accommodate the array by clicking and dragging on any corner of the object. (You could also have enlarged `AlphaNumeric` when you first selected it by using “click and drag” on the object outline.)

7. Run the program. It should look like Figure 5-1.

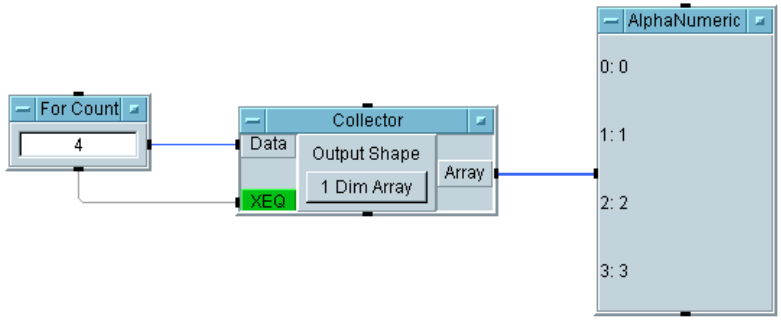


Figure 5-1. The Collector Creating an Array

Lab 5-2: Extracting Values from an Array

To extract values from an array, you can either use the bracket notation in an expression, or use the `Access Array ⇒ Get Values` object. The following example uses expressions in the `Formula` object. You will add several objects to the program in this exercise.

1. Delete the data line between the `Collector` and `AlphaNumeric` by placing the mouse pointer over the line, pressing **Shift-Ctrl**, and then clicking the left mouse button. Then iconize the `Collector`.

2. Select `Device` ⇒ `Formula` and clone it. Move `AlphaNumeric` to the right, and put both `Formula` objects to the right of the `Collector`.

3. Connect the `Collector` data output to the data inputs of the `Formula` objects. Enter `A[2]` in the upper Formula input field, and `A[1:3]` in the lower Formula input field.

   `A[2]` will extract the third element of the array as a Scalar; `A[1:3]` will return a sub-array of three elements holding the second, third, and fourth elements of `A` (meaning the array on the `A` input terminal).

4. Clone `AlphaNumeric` and connect a display to each `Formula` object.

5. Run the program. It should look like Figure 5-2.

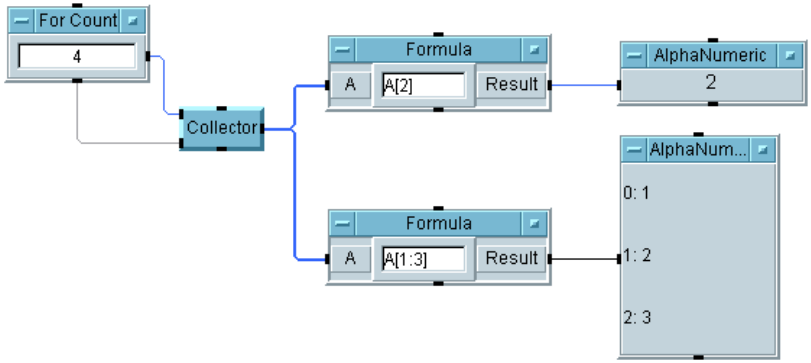


Figure 5-2. Extracting Array Elements with Expressions

Using the To/From File Objects

The `To File` and `From File` objects read data to and from files based on I/O transactions. They have the following characteristics:

- A data file is opened on the first `READ` or `WRITE` transaction. When the program ends, VEE closes any open files automatically.
- VEE maintains one read pointer and one write pointer per file regardless of how many objects are accessing the file. The read pointer identifies the next data item to be read, and the write pointer indicates where the next data item should be written.
- The `To/From File` objects can append data to existing files or overwrite them. If the `Clear File at PreRun & Open` setting is checked in the open view of the `To File` object, then the write pointer starts at the beginning of the file. If not, the pointer is positioned at the end of the existing file. Each `WRITE` transaction appends information to the file at the location of the write pointer. If an `EXECUTE CLEAR` transaction is performed, the write pointer moves to the beginning of the file and erases its contents.
- A read pointer starts at the beginning of a file, and advances through the data depending on the `READ` transactions. You may perform an `EXECUTE REWIND` in the `From File` object to move the pointer back to the beginning of the file without affecting any data.

Note

The `To File` object is also described in “Using Data Files” on page 87 of Chapter 2, “Agilent VEE Programming Techniques.”

Understanding I/O Transactions

I/O transactions are used by VEE to communicate with instruments, files, strings, the operating system, interfaces, other programs, Rocky Mountain Basic, and printers. For example, look at the `To File` object in Figure 5-3.

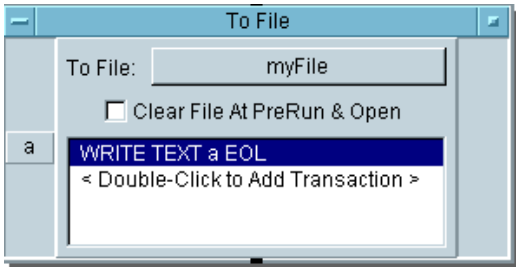


Figure 5-3. The To File Object

The `To File` object shown in Figure 5-3 sends data to the specified file `myFile`. It can include inputs, called transactions, to accept data from a program. For example, this `To File` object includes the transaction `WRITE TEXT a EOL`. When you double-click the transaction, an `I/O Transaction` dialog box appears as shown in Figure 5-4, which configures the specific transaction statement.

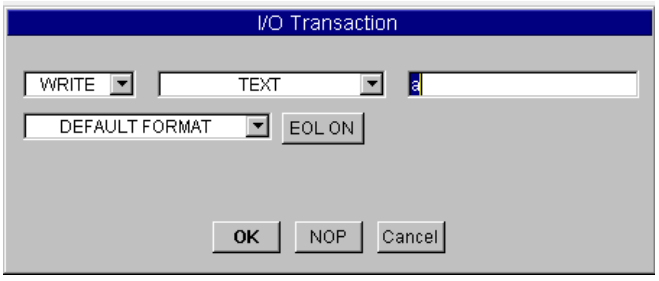


Figure 5-4. An I/O Transaction Dialog Box

There are different forms of this dialog box depending on the object, but all contain certain common elements, including the “actions”, the “encoding”, the “expression list”, the “format”, and the “end-of-line” (EOL) sequence.

I/O Transaction Format

An I/O transaction to write data is usually in the following format:

<action> <encoding> <expression list> <format> <EOL>

Table 5-1 describes the most common actions: READ, WRITE, EXECUTE, and WAIT.

Table 5-1. Types of I/O Transactions

Action	Explanation
READ	Reads data from the specified source using the specified encoding and format.
WRITE	Writes data to the specified target using the specified encoding format.
EXECUTE	Executes a specific command. For example, EXECUTE REWIND repositions a file read or write pointer to the beginning of the file without erasing the contents. EXECUTE CLOSE closes an open file.
WAIT	Waits the specified number of seconds before the next transaction.

Note

There are also a number of actions for `I/O ⇒ Advanced I/O Operations` that you can examine by exploring the objects in the menu.

Encodings and formats refer to the way data is packaged and sent. For instance, a TEXT encoding sends data as ASCII characters. The TEXT encoding could be formatted in a number of ways. For example, to send a string of letters and numbers to a file, a `WRITE TEXT STRING` transaction would send the entire string represented by ASCII characters. A `WRITE TEXT REAL` transaction would only extract the Real numbers from the same string and send them using ASCII characters for the individual digits. Table 5-2 provides brief explanations of encodings.

Table 5-2. I/O Transaction Encoding

Encoding	Explanations
TEXT	Reads or writes all data types in a human-readable form (ASCII) that can easily be edited or ported to other software applications. VEE numeric data is automatically converted to text.
BYTE	Converts numeric data to binary integer and sends or receives the least significant byte.
CASE	Maps an enumerated value or an integer to a string and reads/writes that string. For example, you could use CASE to accept error numbers and write error messages.
BINARY	Handles all data types in a machine-specific binary format.
BINBLOCK	Uses IEEE488.2 definite length block headers with all VEE data types in binary files.
CONTAINER	Uses VEE specific text format with all data types.

In a write transaction, an “expression list” is simply a comma-separated list of expressions that need to be evaluated to yield the data sent. The expression may be composed of a mathematical expression, a data input terminal name, a string constant, a VEE function, a `UserFunction`, or a global variable. In a read transaction, the expression list should consist of a comma-separated list of output terminal names that indicate where to store the data when it is read.

In conjunction with reading data from instruments, data formats are described in Chapter 3, “Easy Ways to Control Instruments,” on page 127. Most of these formats apply to all I/O transactions.

EOL (end-of-line sequence of characters) may be turned on or off, and you can specify the EOL sequence by opening the object menu of most of the `I/O ⇒ To` objects and selecting `Properties...`, then select `Data Format`, and make the changes under `Separator Sequence`.

Lab 5-3: Using the To/From File Objects

This lab exercise describes the process of getting test data to and from files. In this exercise, you will store and retrieve three common test result items: a test name, a time stamp, and a one-dimensional array of Real values. The same process will apply to all VEE data types.

Sending a Text String to a File

1. Select `I/O` ⇒ `To` ⇒ `File`. Set the entries as follows:

filename	Use the default file `myFile`. The default file can be changed by clicking the `To File` input field to get a list box of files in the home directory.
Clear File At PreRun & Open	Check this box. By default, VEE appends new data to the end of an existing file. Checking the box clears the file before new data is written.

2. Double-click in the transaction area to display the `I/O Transaction` dialog box. (Refer to Figure 5-3 and Figure 5-4, if necessary.)

   `WRITE TEXT a EOL` is the default transaction. It writes the data on pin a using TEXT encoding and a specified end-of-line sequence. VEE is not case-sensitive. You can use lower-case or upper-case strings for data input and data output terminal names.

   Set the entries as follows:

a (expression field)	The expression list field is highlighted and contains the default a. Type `"Test1"`, then click OK. (You need the quotation marks to indicate a `Text` string. If you typed `Test1` without the quotation marks, VEE would interpret this as a terminal name or global variable name.)
WRITE	Use the default `WRITE`.

TEXT	Use the default `TEXT`. The encoding `TEXT` will send the data using ASCII characters.
DEFAULT FORMAT	Use `DEFAULT FORMAT`. The `DEFAULT FORMAT` will choose an appropriate VEE format such as `STRING`.
EOL ON	Use the default. The default `EOL` sequence is the escape character for a new line `\n`.

3. Click `OK` to return to the `To File` object. The transaction bar should now contain the statement `WRITE TEXT "Test1" EOL`. This transaction sends the string `Test1` to the specified file.

Sending a Time Stamp to a File

The function `now()` in the `Device ⇒ Function & Object Browser ⇒ Time & Date` category gives the current time expressed as a `Real64 Scalar`. The value of the Real is the number of seconds since 00:00 hours on Jan. 1, 0001 AD.

Therefore, `now()` returns a value about `63G`. VEE provides this format because it is easier to manipulate mathematically and conserves storage space. If you want to store the time stamp in a more readable format, use the `TIME STAMP FORMAT` in the `To File` object. Follow these steps to send a time stamp to a file.

1. In the same `To File` object, double-click in the transaction area to display the `I/O Transaction` box.

2. Double-click the expression list input field to highlight the `a` and type `now()`. The `now()` function sends the current time from the computer clock in a `Real` format.

3. Change the `Real` format to the `Time Stamp Format`.Click the arrow next to `DEFAULT FORMAT` to display the drop-down menu and select `TIME STAMP FORMAT`. The `I/O Transaction` dialog box now displays additional entries. Set the entries as follows:

Date & Time	Select `Time` in the drop-down menu.

HH:MM:SS	Click and toggle to `HH:MM` (from the hour, minute, and second format to the hour, minute format).
24 HOUR	Click and toggle to `12 HOUR` (from 24-hour format to a.m. and p.m. format).

The I/O Transaction dialog box should now look like Figure 5-5.

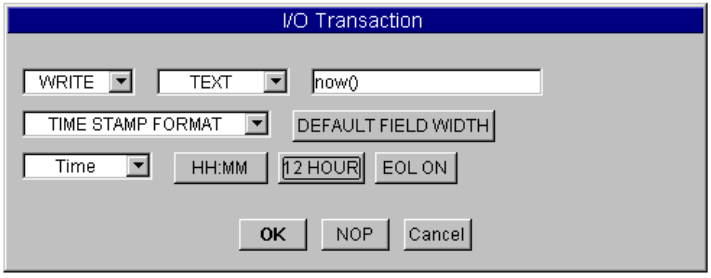


Figure 5-5. The TIME STAMP I/O Transaction Box

4. Click `OK` to return to the `To File` box. The second transaction bar should now contain the statement `WRITE TEXT now() TIME:HM:H12 EOL`.

Sending a Real Array to a File

Create a one-dimensional array of four elements using the `For Count` and `Collector` objects, and append it to `myFile`.

1. Select `Flow` ⇒ `Repeat` ⇒ `For Count`. Change the default value in `For Count` to `4`.

2. Select `Data` ⇒ `Collector`. Double-click the `Collector` to switch to Open view. Connect the data output of `For Count` to the data input of the `Collector` (the top input pin). Connect the `For Count` sequence output pin to the `XEQ` pin (the bottom input pin) on the `Collector`. Iconize the `Collector`.

The Collector will now create the array [0, 1, 2, 3], which you can send to the data file.

3. Using the same To File object, double-click in the transaction area. In the I/O Transaction dialog box, open the DEFAULT FORMAT menu, and select REAL64 FORMAT.

   The I/O Transaction dialog box displays additional buttons for the REAL64 FORMAT selection. You can leave all of the default choices, but you might want to investigate the options for future reference.

4. Click OK to close the I/O Transaction box. The transaction bar in the To File object should now contain the statement WRITE TEXT a REAL64 STD EOL. Notice that VEE also automatically adds an input terminal a.

5. Connect the output from the Collector to the input a of To File. The program should now look like Figure 5-6. (The configured I/O Transaction box is also displayed.)

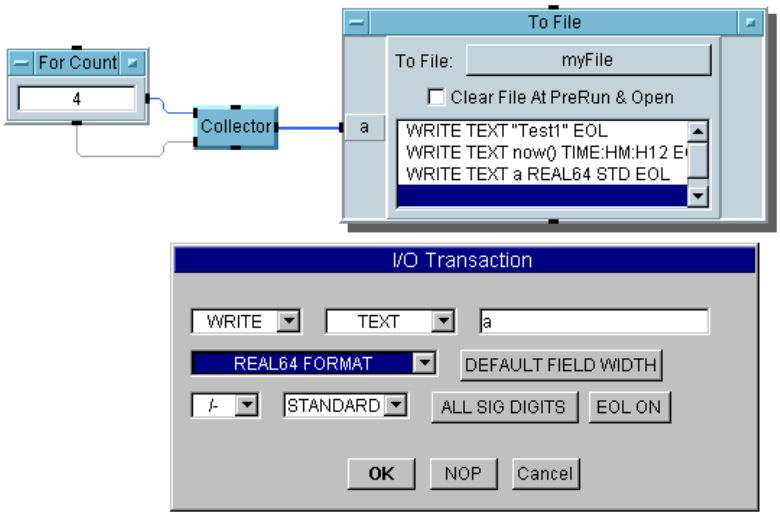


Figure 5-6. Storing Data Using the To File Object

Retrieving Data with the From File Object

To retrieve data using a `From File` object, you must know how the data was stored.

Note

You can also store and retrieve data using `To DataSet` or `From DataSet`, which does not require you to know the type of data in the file. Datasets are described in the section "Using DataSets to Store and Retrieve Records" on page 232.

In this example, the name of a test is stored in a `String Format`, followed by a time stamp in `Time Stamp Format` and an array of `Real64` numbers. You will create three transactions in `From File` to read the data back into VEE.

1. Select `I/O` ⇒ `From` ⇒ `File` and place it below the `To File` object.

2. Connect the sequence output pin of the `To File` object to the sequence input pin of the `From File` object.

   This sequence connection ensures the `To File` object has completed sending data to `myFile` before `From File` begins to extract data.

3. In the `From File` object, leave the default data file `myFile`. Double-click the transaction bar to get the `I/O Transaction` dialog box. Click `REAL64 FORMAT` and change it to `STRING FORMAT`, as shown in Figure 5-7.

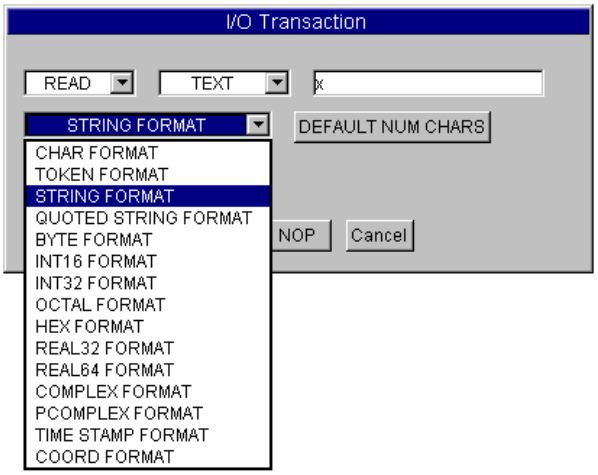


Figure 5-7. Selecting String Format

4. All of the other defaults are correct, so click `OK` to close the `I/O Transaction` box. The transaction bar in the `From File` object should now contain the statement `READ TEXT x STR`.

Now add two more transactions to read back the time stamp and the real array.

5. In the same `From File` object, double-click below the first transaction bar. The `I/O Transaction` dialog box appears. Double-click on the expression list input field to highlight `x` and type `y`, for the second transaction to read data back to pin `y`. (If this pin were left as "`x`" then the second transaction would overwrite the data that the first transaction put into "`x`," instead of appending it.) Change `REAL64 FORMAT` to `STRING FORMAT`, then click `OK`.

Note

To read the time stamp back as a text string, use the `STRING FORMAT`. The `TIME STAMP FORMAT` converts the time stamp data back to a `Real` number.

6. In the same `From File` object, double-click below the second transaction bar to display to the `I/O Transaction` dialog box. Set entries as follows:

(expression field)	Edit `x` to `z`, so that the `Real` array is read back to the `Z` output terminal.
SCALAR	Change `SCALAR` to `ARRAY 1D`.
SIZE:	Now the `I/O Transaction` box adds a `SIZE` button. In this case, the array has four elements. Replace `10` with `4` and click `OK`.

Note

If you do not know the size of an array, you may toggle `SIZE` to `TO END`. This will read data to the end of the file without VEE knowing its exact size. For example, you could use this feature to read the entire contents of a file as a string array to examine the file contents.

The transaction bar in the `From File` object should now contain the statements `READ TEXT y STR` and `READ TEXT z REAL64 ARRAY:4`. Notice that VEE automatically adds the data output terminals for x, y, and z. You can also manually add or delete input and output terminals under object menu ⇒ `Add Terminal`, `Delete Terminal`, or using the shortcuts **Ctrl-A** and **Ctrl-D**.

7. Select `Display ⇒ AlphaNumeric` and clone it twice to get three displays. Connect the `AlphaNumeric` objects to the three data output pins on `From File`. Enlarge the array display by clicking and dragging the object by any corner.

   Tip: You can also size the `AlphaNumeric` displays by clicking and dragging the object outlines when you first select them from the menu.

8. Run the program. It should look like Figure 5-8.

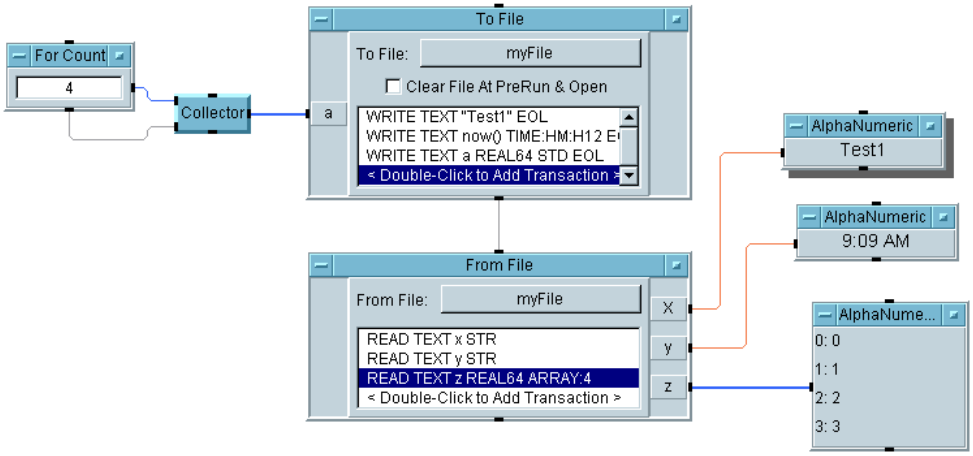


Figure 5-8. Retrieving Data Using the From File Object

Notice that the first Alphanumeric displays the title, the second displays the time of the test, and the third lists the numbers in the array.

Using Records to Store Mixed Data Types

The Record data type can store different data types in a single data container. Record can include any VEE data type. The data can be in the shape of a Scalar or an Array. You can store a test name, a time stamp, and a real array in a single data structure.

The individual elements in a Record are stored as fields and are accessed using a dot notation. For example, `Rec.Name` accesses the field called `Name` within a Record called `Rec`. In an array of records, `Rec[2].Name` signifies the `Name` field in the third record in the array. All arrays start indexing at zero.

There are several benefits to structuring test data using the Record data type:

- You can create logical groupings of mixed data types in a single container, which makes a program easier to develop and maintain. For example, you might use the following fields for a record storing test data: test name, value returned, pass or fail indicator, time stamp, nominal value expected, upper pass limit, lower pass limit, and a description of the test.

- You can manipulate a single data container rather than eight separate data containers. This makes the program simpler and more readable.

- You can store and retrieve Records from DataSets in VEE. A **DataSet** is a special file created to store records. When you retrieve records from a DataSet, you do not have to know the data types. VEE provides objects to retrieve, sort, and search the information stored in DataSets.

Lab 5-4: Using Records

This exercise describes how to use the Record datatype. You will learn how to build a record, how to retrieve a particular field in that record, how to set a chosen field, and how to unbuild the entire record in a single step. This exercise also uses the time stamp function `now()` in a different way.

Building a Record

Build a Record with three fields: the name of a test stored as a `String`, a time stamp stored as a `Real Scalar`, and simulated test results stored as a four element `Array of Reals`. When you retrieve these fields in the next exercise, you will see that you can convert the time stamp into a number of different formats for display.

1. Create the test name by selecting `Data ⇒ Constant ⇒ Text` and entering `Test1` in the input field. Rename the object `Text Constant`. Iconize `Text Constant`.

2. Select `Device ⇒ Function & Object Browser`. Click `Built-in Functions` under `Type`, `Time & Date` under `Category`, select `now` under `Functions`, and click `Create Formula`. Place the object below `Text Constant`.

3. Select `Data ⇒ Constant ⇒ Real64` and place it below `now()`.

   You can turn this `Scalar Real64` into an `Array 1D` by clicking `Properties...` in the `Real64` object menu and choosing `1D Array`.

4. Open the `Constant Properties` box by double-clicking on the `Real64` title bar. Select `1D Array` under `Configuration`, change the `Size` to `4`, then click `OK`.

   Enter four values into this array by double-clicking next to element `0000` to highlight the first entry, then input the values `2.2`, `3.3`, `4.4`, `5.5` using the **Tab** key between each entry. Iconize `Real64`.

5. Select `Data ⇒ Build Data ⇒ Record` and place it to the right of the three other objects. Add a third data input terminal so you can input three fields. Open each terminal by double-clicking over the terminal and rename the three input terminals to `testname`, `time`, and `data`.

   The `Output Shape` on the `Build Record` object toggles between `Scalar` and `Array`. The `Scalar` default will be the correct choice for the majority of situations. (For more information, see the *VEE Pro Advanced Techniques* manual.)

6. Connect the `Text Constant` object to the `testname` terminal, the `now()` object to the `time` terminal, and the `Real64` object to the data terminal on the `Build Record` object.

7. Run the program. Double-click on the `Record` data output terminal to examine the record. It should look like Figure 5-9.

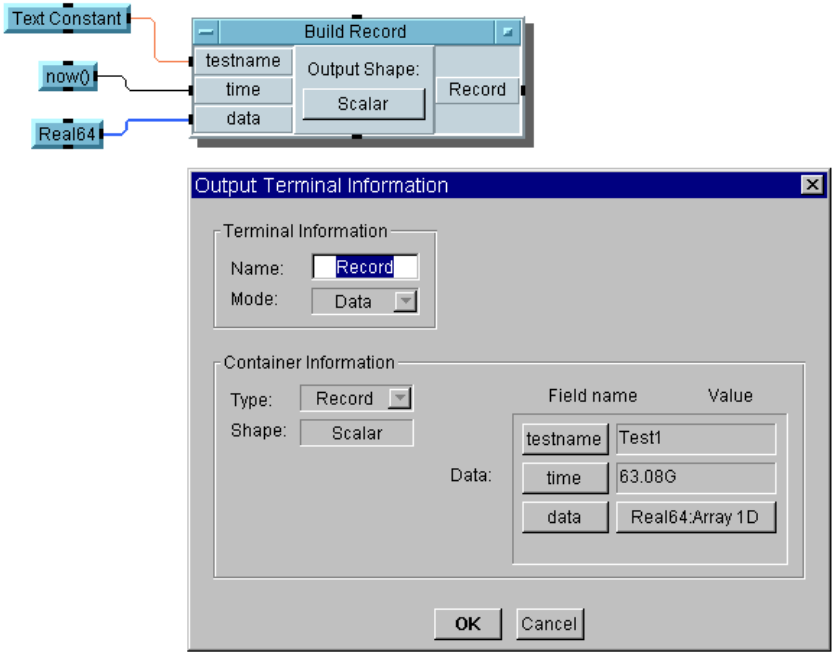


Figure 5-9. Output Terminal Information on a Record

You can see the three fields and their values. If you click on the `Real64: Array 1D` button, a list box shows the actual values. Notice that the time stamp has been stored as a `Real64 Scalar`. In the next exercise, you will convert it to a more readable form. Click `OK` to close the `Output Terminal Information` dialog box. Save the program as `records.vee`.

Getting a Field From a Record

Use the `Get Field` object to extract each of the three fields from the record, then display the values for each.

1. Open the `records.vee` program.

2. Select `Data` ⇒ `Access Record` ⇒ `Get Field`. The object appears with `rec.field` for a title.

   The data input labeled `rec` will take any record regardless of the number and type of fields. `Rec.field` is the default selection in the input field, but this can edited to retrieve any field. `Rec` refers to the record at the data input terminal by the same name. (Remember that VEE is *not* case sensitive.)

Note

The `Get Field` object is a `Formula` configured with inputs and an expression, like the formulas in the `Function & Object Browser`.

3. Clone `rec.field` twice and place the objects to the right of `Build Record`.

4. Connect `Build Record` data output to all three `rec.field` objects.

   Since the three fields are stored as `testname`, `time`, and `data`, you will have to edit the `rec.field` objects to get the appropriate field.

5. Edit the three `rec.field` object expression fields to `rec.testname`, `rec.time`, and `rec.data`.

6. Select `Display` ⇒ `AlphaNumeric` and clone it twice. Connect the three displays to the three `rec.field` objects. Resize the third display to accommodate the real array, about three times longer than the other objects.

7. Open the second `AlphaNumeric` display object menu and select `Properties`, then select the `Number` folder. Click to the left of `Global Format` to remove the check mark.

Set the display format. Open the `Standard` menu in the `Real` section. Select `Time Stamp` and click `OK`.

8. Click `HH:MM:SS` to toggle to `HH:MM`. Click `24 HOUR` to toggle to `12 HOUR`. See Figure 5-10.

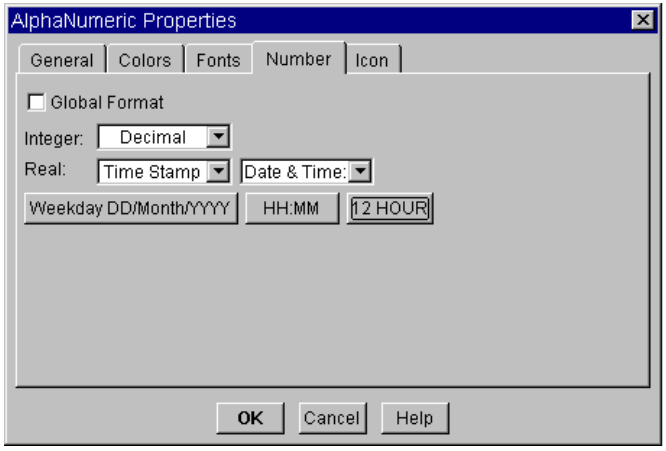


Figure 5-10. The AlphaNumeric Properties Box

9. Run the program and save it as `getfield.vee`. The program should look like Figure 5-11.

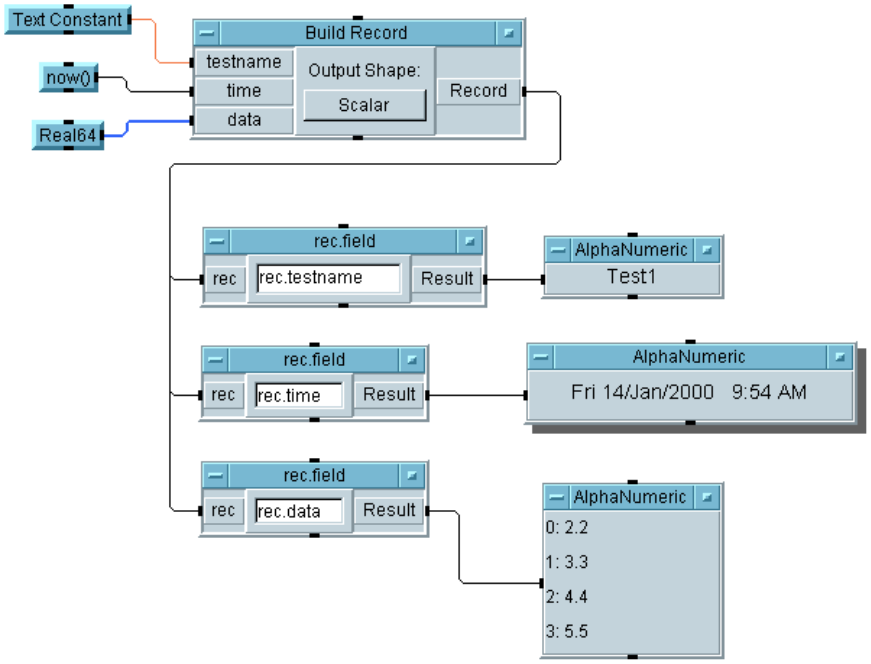


Figure 5-11. Using the Get Field Object

Notice that the second display lists the weekday, the date, and the time expressed in hours, minutes, and an a.m. or p.m. designation.

Setting a Field in a Record

This exercise shows how to change data in specific fields of a record.

Note

You can re-use the same `Record` with different tests.

1. Open the `getfield.vee` program.
2. Delete all objects after `Build Record`, by selecting objects and pressing **Ctrl-X**.

1. Select Data ⇒ Access Record ⇒ Set Field and place it to the right of Build Record. Connect the output from Build Record to the rec input of Set Field. The title will be rec.field = b.

   Set Field works by assigning the expression on the right side of the assignment symbol (=) to the left hand side. Therefore, the specified field of rec is modified to contain the value(s) from the right hand side. The rest of the record is unchanged. You connect the incoming record to rec and the incoming new value to b. The modified record will be put on the data output terminal labeled rec.

Note

The Set Field object is a Formula configured with inputs and an expression, like the formulas in the Function & Object Browser.

2. Edit the expression to rec.data[*]=b to change the value of the four element array in the data field. (You need to use the array [*] notation, because you are changing the whole array in the field of this record.) You will put the new values for the array on the input terminal b.

3. Select Data ⇒ Constant ⇒ Real64 and place it under the Build Record object. Open the object menu, and select Properties. Select 1D Array under Configuration, then edit the Size to 4, and click OK.

   If the new values for the record field are contained in an array, it must have the same size as the current array.

   Enter the values 1, 2, 3, 4 into Real64 by highlighting the first entry and using the **Tab** key to move to subsequent entries. (Do not press the **Tab** key after the last entry.) Connect it to the Set Field (titled rec.field=b) input labeled b.

Now use the Get Field object to extract the field rec.data from the record and display the results.

4. Select Data ⇒ Access Record ⇒ Get Field and place the object under the Set Field (rec.field=b) object. Edit the Get Field object expression from rec.field to rec.data. Connect the data output of rec.field = b to the data input of rec.field.

Note You could also use a `Formula` object with `A.data` in the expression field.

5. Select an `AlphaNumeric` display, size it to accommodate an array, and connect it to the `rec.field` output pin.

6. Run the program and save it as `setfield.vee`. The program should look like Figure 5-12.

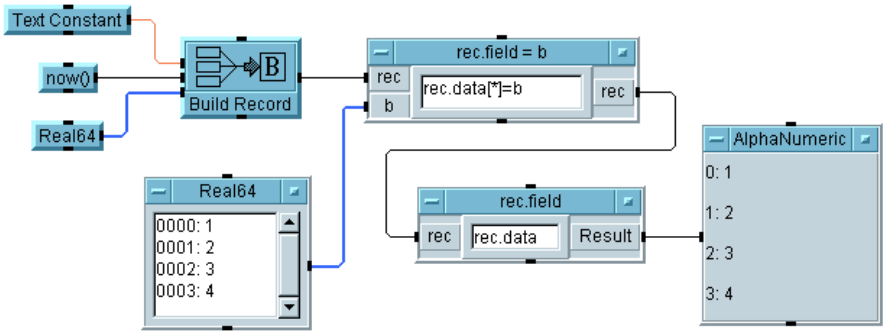


Figure 5-12. Using the Set Field Object

You can modify any `Record` fields as shown in this example. You could also modify part of the field. For example, try changing the expression in `rec.field = b` to `rec.data[1]=20`. Then delete the `rec.field = b` input `b`. Run the program again and you should see the array: `2.2`, `20`, `4.4`, `5.5`.

Unbuilding a Record in a Single Step

To extract all record fields and get a list of the field names and their types, use the `UnBuild Record` object.

1. Open the `setfield.vee` program. Delete all objects after `Build Record`.

2. Select `Data` ⇒ `UnBuild Data` ⇒ `Record` and place it under `Build Record`, switch the open view, and connect the output of `Build Record` to the input of `UnBuild Record`. Add another data output pin to `UnBuild Record` and rename the `A`, `B`, and `C` outputs to the field names: `testname`, `time`, and `data`.

3. Select an `AlphaNumeric` display and clone it four times. Connect the five displays to the five output terminals on `UnBuild Record`. You will have to enlarge the displays for `Name List`, `Type List`, and `data` to accommodate arrays. Also, reconfigure the time display to present time in day/month/year and hours, minutes using a 12 hour format.

4. Run the program and save it as `unbuild.vee`. It should look like Figure 5-13.

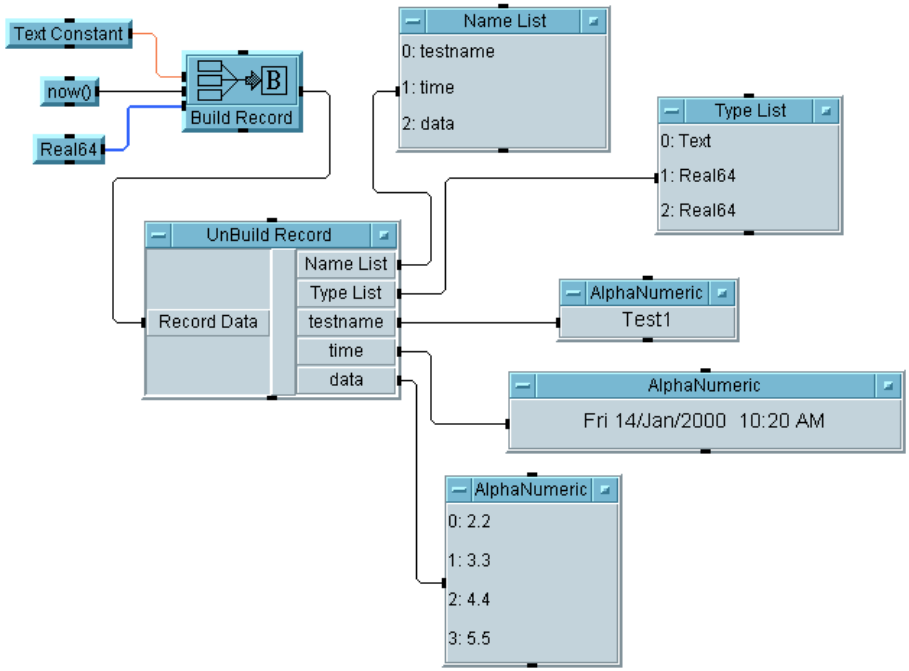


Figure 5-13. Using the UnBuild Record Object

Notice that the `Name List` pin gives the names `testname`, `time`, and `data` of the three fields in the record, just as the `Type List` identifies `testname` as `Text`, and `time` and `data` as `Real64` types.

Using DataSets to Store and Retrieve Records

DataSets can retrieve one or more records. VEE objects unpack the records. Therefore, by storing records to DataSets instead of files, you do not have to remember the data types. You can also perform sort and search operations on the data, creating your own customized test database.

Lab 5-5: Using DataSets

A DataSet is simply an array of Records stored in a file. This exercise shows how to get data into and out of a DataSet.

Storing and Retrieving a Record from a DataSet

This exercise creates an array of ten Records, each containing three fields with a test name, a Real64 Scalar, and an array of Reals. It stores the array of Records in a DataSet, and retrieves the records and displays them.

1. Select Flow ⇒ Start. Select Flow ⇒ Repeat ⇒ For Count and place the object under Start. Select Device ⇒ Formula and place the object to the right of For Count. Connect Start to the sequence input pin on For Count; connect the For Count data output pin to Formula data input pin.

2. Double-click the Formula expression field to highlight the default expression, and then type "test" + a.

   When you click Start, the For Count object outputs integers zero through nine sequentially to the A pin of Formula. In the Formula object, the integers are added to the word "test" and output as Text Scalars: test0, test1, test2,...,test9. These values will fill the first fields in the ten Records.

3. Select `Data ⇒ Build Data ⇒ Record`, and place the object to the right of `Formula`. Add a data input pin. Connect the data output of `Formula` to the `A` input of `Build Record`.

4. Select the **Function & Object Browser** icon from the tool bar.

   a. Choose `Built-in Functions`, `Probability & Statistics`, and `random` to create the `random (low, high)` object. Place the object below the `Formula` object.

   b. Delete the input terminals, and change the input parameters from `low` to `0`, and from `high` to `1`.

   c. Rename the object `Random Number`, and connect its data output to the `B` terminal of `Build Record`.

5. Connect the `Formula` sequence output pin to the sequence input pin of `Random Number`. Connecting the sequence pins ensures that each iteration of the program puts a new random number into the `B` field of the particular record.

6. Select `Data ⇒ Constant ⇒ Real64`. Place the `Real64` object below the `Formula` object.

   a. Open the object menu and click `Properties`. Type `Real Array` for the title, under `Configuration` click `1D Array`, and change the `Size` to `3`. Click `OK`.

   b. Highlight each entry in the array by double-clicking and typing in the numbers `1`, `2`, and `3`.

   c. Connect the `Real Array` data output to the `C` terminal on `Build Record`.

7. Select `I/O ⇒ To ⇒ DataSet` and place the object under `Build Record`. Connect the data output of `Build Record` to its data input. Leave the default file name `myfile`, and check `Clear File At PreRun`.

8. Run the program. It should put an array of ten records into the DataSet called `myFile`, as shown in Figure 5-14.

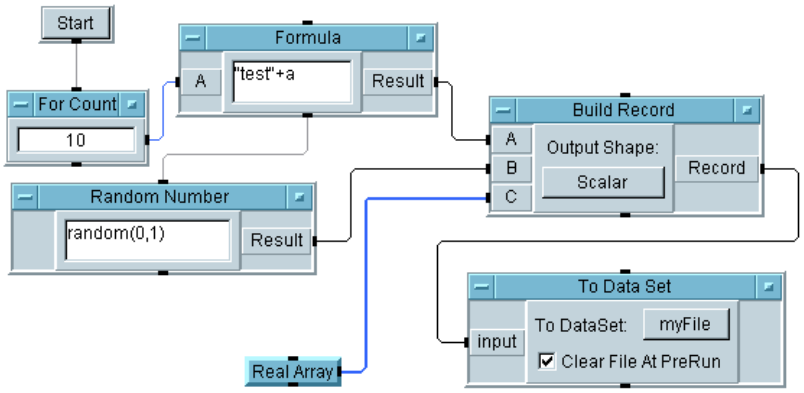


Figure 5-14. Storing an Array of Records in a DataSet

Now retrieve the array of records and display it using the `From DataSet` and `Record Constant` objects.

9. Select `I/O` ⇒ `From` ⇒ `DataSet` and place the object below `For Count`. Leave the default file name, `myFile`. Click the `Get Records` field to toggle from `One` to `All`. Finally, leave the default of `1` in the expression field at the bottom.

   With these settings, VEE looks at the DataSet in `myFile` and finds all the records that fit the criterion in the expression field. If you set `Get Records` to `One`, VEE would output the first record that met the criterion in the expression field. The `1` signifies a `TRUE` condition meaning that all of the records fit the criterion, so the entire array of records in the file will be put on the output pin labeled `Rec`. Other uses of the expression field are explained in other exercises. Consult `Help` in the object menu for more information.

   Connect the `For Count` sequence output pin to the sequence input on the `From Data Set` object. This ensures the part of the program that sends data to `myFile` executes before the data is read from the file. You can turn on `Show Data Flow` to show the order of events.

10. Select `Data ⇒ Constant ⇒ Record` and place the object below `To Data Set`. Open the object menu and select `Add Terminal ⇒ Control Input`. Click `Default Value` from the list box presented, then click `OK`. Resize the `Record` object to be larger, so you can see the results when you run the program.

    The record received will become the default value. In this case, `Record` will receive an array of records from the `From Data Set` object, and it will format itself to display that array of records.

11. Connect the `From Data Set` output pin `Rec` to the `Default Value` pin on `Record`. If you would like to see this terminal, open the object menu and select `Properties`, then `Show Terminals`, then `OK`. A dotted line appears between `From Data Set` and `Record`.

Note A dotted line between two objects indicates a control line.

12. Run the program and save it as `dataset1.vee`. The program should look like Figure 5-15.

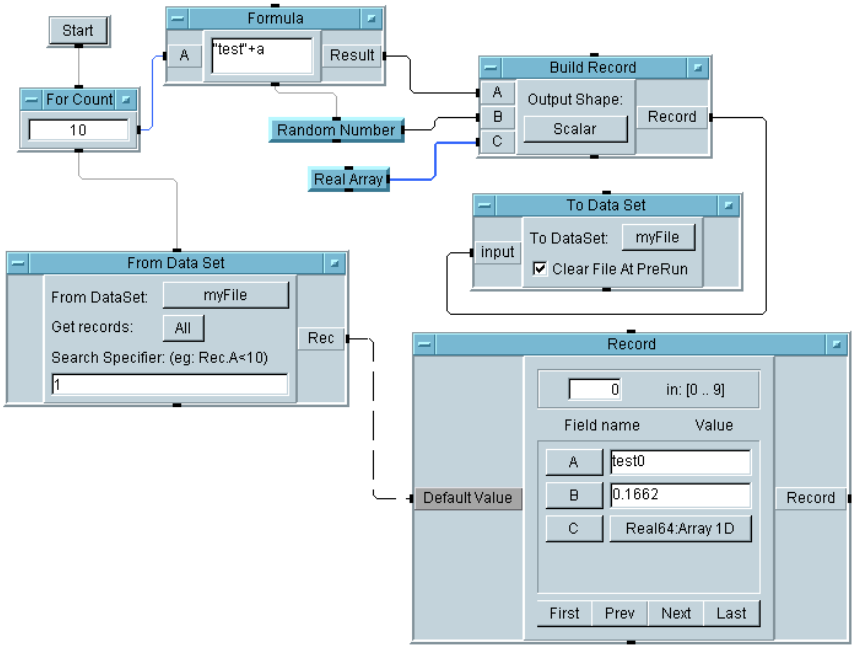


Figure 5-15. Storing and Retrieving Data Using DataSets

Note

A `From Data Set` object must include at least one record that meets the criterion, or VEE issues an error message. To avoid an error, add an `EOF` (end-of-file) output pin to the object which activates if no records meet the criterion. You can then add actions to the program when the `EOF` results.

Customizing a Simple Test Database

You can search and sort a `DataSet` for information, such as test name, time stampls, test parameters, test values, pass or fail indicators, and test descriptions. Therefore, `DataSet` records can act as a test database. To search for information, you can use the `From Data Set` object as follows:

- The expression field in the `From Data Set` object is used for search operations.
- The function `sort()` can be used to sort records using a specified field.

Lab 5-6: Using Search and Sort Operations with DataSets

In this exercise, you will learn how to search a `DataSet` for information, create an operator interface for the search operation, and program a sort operation.

Performing a Search Operation With DataSets

1. Open the `dataset1.vee` program.
2. Double-click on the expression field at the bottom of the `From Data Set` object to highlight the current expression, `1`. Enter `Rec.B>=0.5`. The object will now output all records, where field `B` (the random number in our code) is greater or equal to `0.5`.
3. Add an `EOF` pin that will fire if no records match the criterion in the expression field. Place the cursor over the data output area of the `From Data Set` object, and press **Ctrl-A**. An `EOF` output pin is added to the `From Data Set` object, as shown in Figure 5-16.

Note

To add an `EOF` pin, you could also open the object menu, and click `Add Terminal` ⇒ `Data Output...`.

4. Run the program and save it as `dataset2.vee`.

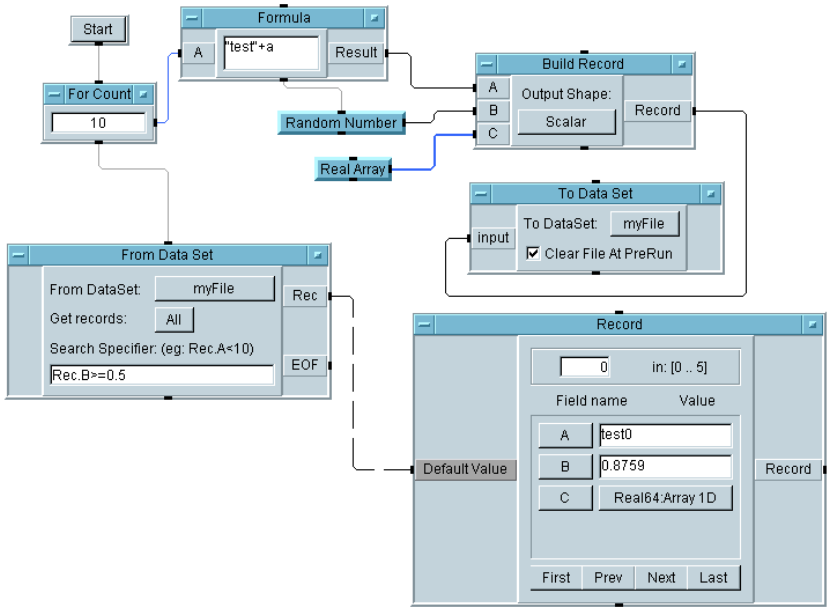


Figure 5-16. A Search Operation with DataSets

Creating an Operator Interface for a Search Operation

This exercise adds a menu for an operator to extract data from the test results database. The operator interface is secured to avoid accidental modifications to the program.

The specifications of the program are as follows:

- Provide a test menu that will allow operators to select a particular test from `test0` through `test9`, from which they want all related test data.
- Display the specified test results with the fields and values labeled. The operator should be able to interact with the display to gain more detailed information.
- Include clear operating instructions.

Follow these steps to create the program.

1. Open the `dataset2.vee` program.

Add a control input that will allow you to input the expression in the `From Data Set` object programmatically.

2. Open the `From Data Set` object menu and select `Add Terminal...` ⇒ `Control Input...`. Select `Formula` from the menu presented. A `Formula` input terminal appears. Click the `Get records` field to toggle from `All` to `One` to access one test record at a time.

   You want the user to select a particular test name. The test names are located in field `A` of all records. Add the expression:

   `Rec.A==`<*test name in quotation marks*>

   `Rec.A` outputs the record where field `A` matches the test name the operator selects. For example, if the operator selects `test6`, the expression should read `Rec.A=="test6"`. The object extracts the test record, which can then be displayed.

Create a menu that allows the operator to click a button next to the desired selection.

3. Select `Data` ⇒ `Selection Control` ⇒ `Radio Buttons` and place the object to the left of `For Count`.

   a. Open the object menu and select `Edit Enum Values...`. Highlight `0000: Item 1` and type `test0`. Press the **Tab** key to move to `0001: Item2 and enter test1`. When you press the **Tab** key after the third entry (`test2`), another entry automatically appears. Continue to enter values until you reach `test9`. Click `OK` and all ten entries should be displayed, from `test0` to `test9`.

   b. Click the `Properties` selection in the object menu, change the object name from `Radio Buttons` to `Test Menu`, select `Auto Execute` under `Execution`, select `Open View` ⇒ `Show Terminals`, and click `OK`.

4. The program can now execute whenever the operator makes a menu selection, so delete the `Start` object. Press the right mouse button over the `Start` object and select `Cut`.

5. The program should only execute when a menu selection is made, so connect the `Test Menu` data output pin `Enum` to the `For Count` sequence input pin. The program should look like Figure 5-17.

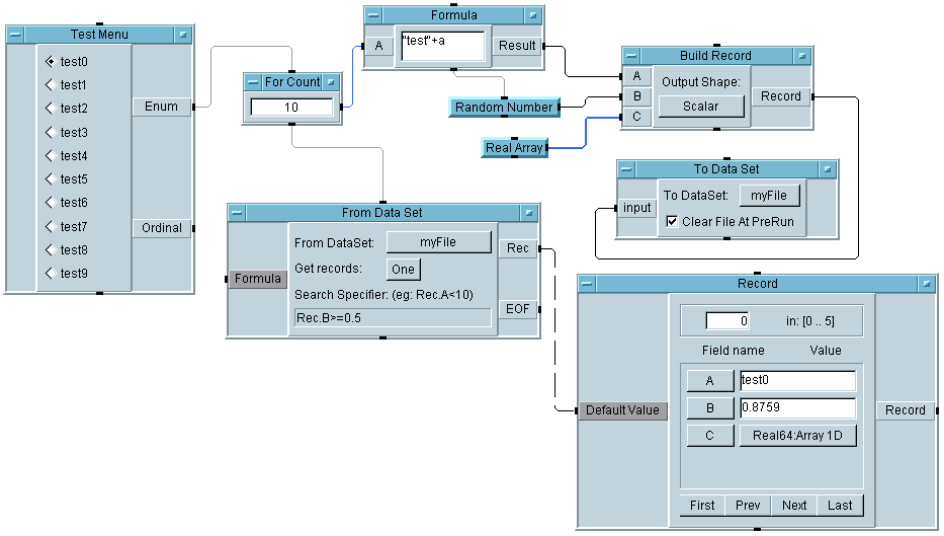


Figure 5-17. Adding the Test Menu object

6. The output of the `Test Menu` goes into a `Formula` object, which then sends the correct formula to the `From Data Set` object.

   Select `Device` ⇒ `Formula`, and place the object below `Test Menu`. (You may want to rearrange and/or resize objects as you add items during the exercise.) In the new `Formula` object, enter the following expression:

   ```
   "Rec.A==" + "\"" + A + "\""
   ```

"Rec.A=="	`"Rec.A=="` sends a `Text` data type to the `From Data Formula` expression input. (The quotation marks indicate a text string.)
A	`VEE` looks at the first field `A` of all records in the `DataSet` file, and selects the first record that equals the selected test name.
"\""	The escape character for a quotation mark is `\"`. The escape character is then put into quotes to indicate a text string. The *test name* comes from the `Test Menu` as an `Enum` data type. Quotes are required to put the correct formula into the `From DataSet` object.

   For example, if `test6` is selected, then the final formula will read `Rec.A=="test6"`. The `From Data Set` object then outputs the first record it finds, whose `"A"` field is equal to `"test6"`.

7. Connect the `Test Menu Enum` data output pin to the data input pin on the `Formula` object. Iconize the `Formula` object.

8. Connect the `Formula` data output pin to the control input pin on the `From Data Set` object labeled `Formula`.

9. To ensure that the old data from `Formula` is not reused, delete the sequence line between `For Count` and `From Data Set.` Connect the `For Count` sequence output pin to the `Formula` sequence input pin.

10. Connect the `Formula` sequence output pin to the `From Data Set` sequence input pin. This ensures the right data from `Formula` is being used.

11. Create a box displaying instructions for the operator. Select `Display` ⇒ `Note Pad`. Change the title to `Test Results Database Instructions`. Click on the `Note Pad` input area and type: `Select the test results you want from the Test Menu.`

12. Rename the `Record Constant` object `Test Results`.

13. The program should look like Figure 5-18. Run the program a few times to verify that it works. Since the `Test Menu` object has `AutoExecute` turned on, make a menu selection to run the program.

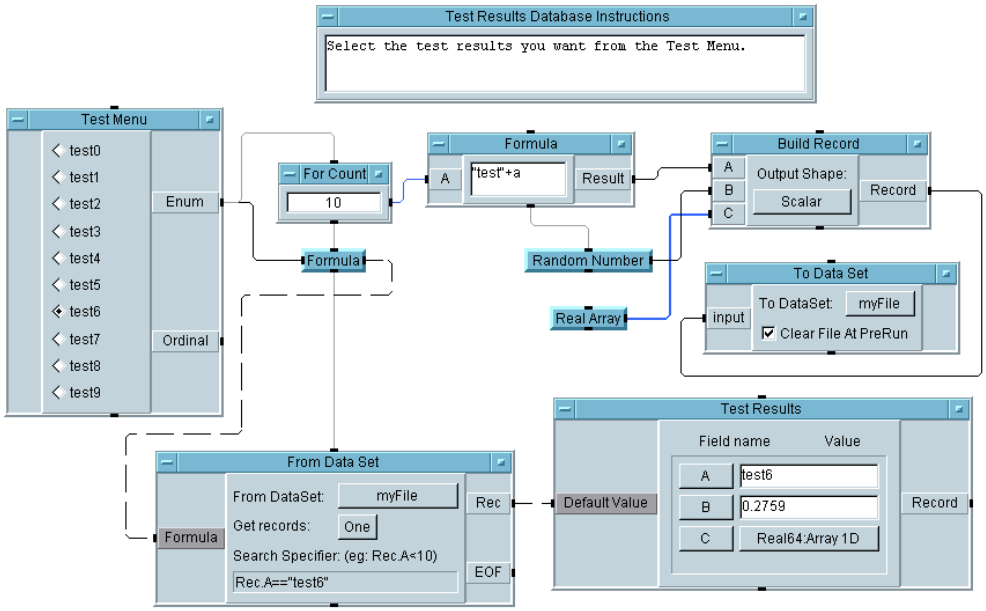


Figure 5-18. Adding a Menu to the Search Operation

Next, create the operator interface.

14. Press **Ctrl** and click these objects: `Test Menu`, `Test Results Database Instructions`, and `Test Results`.

    All objects selected show a shadow. Verify no other objects are selected.

    Then select `Edit` ⇒ `Add to Panel`, and the operator interface appears as a panel view. You can then move and size the objects. One layout is shown in Figure 5-19.

Note

If the `Add to Panel` selection is grayed out, it means that you do not have any objects selected in the work area.

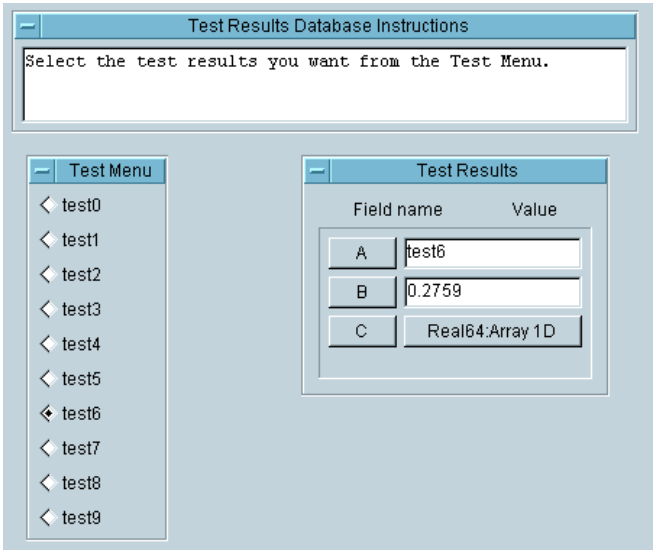


Figure 5-19. The Operator Interface for the Database

15. Run the program a few times by making selections in `Test Menu`. Save the program as `database.vee`.

    Notice that you can get more detailed information on any given record by clicking the field names or the values in the `Record Constant` object (named `Test Results`).

Note

To secure the operator interface and program from changes, select `File` ⇒ `Create RunTime Version...`. Enter a name for the program and VEE will automatically add a `*.vxe` extension to separate it from unsecured versions.

Performing a Sort Operation on a Record Field

This exercise uses the `dataset2.vee` program from a previous exercise. The `dataset2.vee` program sets a condition in the `From DataSet` object such as `Rec.B>=0.5`, and VEE extracts all the records that meet the requirement. The array of resulting records is displayed in the `Record Constant` object.

In this exercise, `dataset2.vee` is modified to sort the resulting records to determine which tests are failing by the greatest margin. The tests are sorted by the second field in descending order.

1. Open the `dataset2.vee` program.

2. Select `Device` ⇒ `Formula` and connect the `From Data Set` data output pin `Rec` to the `Formula` object data input pin. Double-click the `Formula` expression field to highlight the default formula, then enter `sort(a, 1, "B").`

   The `Sort` object is located in the `Function & Object Browser`, `Array Category` functions. You can read detailed information on its capabilities in the object menu `Help` entry. The `sort ()` function is called from the `Formula` object.

   The first parameter sorts the data on the `Formula` object `A` pin, which is in an array of records. The second parameter indicates the direction of the sort: any non-zero number indicates an ascending direction, a zero indicates descending. The default direction is ascending. The third parameter, in the case of a `Record` data type, indicates the name of the field to sort. Therefore, this performs an ascending sort on the `B` field in the array of records.

3. Select `Display` ⇒ `AlphaNumeric` and connect it to the data output pin of the `Formula` object.

4. Run the program a few times. It should look similar to Figure 5-20. Notice that the program sorts all of the records returned from the `DataSet` file in ascending order by field `B`.

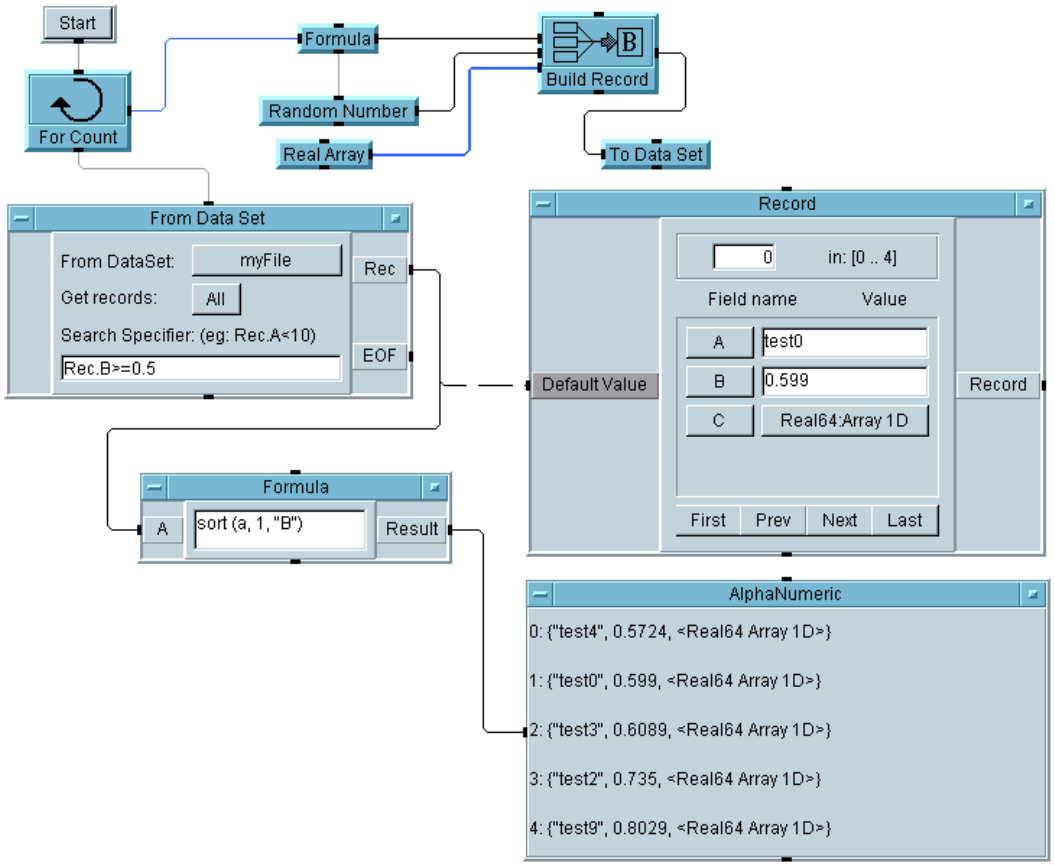


Figure 5-20. A Sort Operation on a Record Field

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary, before proceeding to the next chapter.

- Explain the basic notation for using arrays.
- Create an array using the `Collector` object.
- Extract elements from an array using the `Formula` object.
- Send a string, time stamp, and real array to a file.
- Retrieve a string, time stamp, and real array from a file.
- Use the function `now()` for a time stamp.
- Format time stamps in a variety of ways for display.
- Build and unbuild a record.
- Get and set fields in a record.
- Store a record to a `DataSet`.
- Retrieve a record from a `DataSet`.
- Perform a search operation on a `DataSet`.
- Perform a sort operation on a `Record` field.
- Combine VEE tools to create a simple test database.

6

Creating Reports Easily Using ActiveX

Creating Reports Easily Using ActiveX

In this chapter you will learn about:

- ActiveX Automation in VEE
- Using ActiveX for reports with MS Excel
- Using ActiveX for reports with MS Word

Average time to complete: 1.5 hours

Overview

In this chapter, you will learn how to generate reports in other applications, such as MS Excel, by sending data from the VEE program to the MS Excel program. VEE uses ActiveX Automation to control other applications, which provides a fast process for creating detailed and effective reports.

The first lab exercise describes how to send data to an MS Excel spreadsheet automatically using ActiveX Automation. The second exercise describes generic template for generating reports, and how to expand on the functionality of the basic template. The final exercise uses ActiveX in VEE to send a screen dump and test data to an MS Word document. (The principles are the same for other spreadsheet and word processing programs that support ActiveX Automation.)

Note

ActiveX replaces the use of DDE in VEE. However, DDE is still supported in VEE. To use DDE in legacy applications, refer to the second edition of *Visual Programming with HP VEE*.

ActiveX Automation in Agilent VEE

In this chapter, the term ActiveX Automation refers to VEE's ability to act as an Automation Controller of Automation Server applications such as MS Excel, MS Word, and MS Access. The exercises focus on the practical application of Microsoft's ActiveX technology to generate test and measurement program reports.

Note There are also other related lab exercises in this manual: "Using an ActiveX Control" on page 396, and "The Callable VEE ActiveX Automation Server" on page 446. For more detailed information about Automation terminology and concepts, refer to the *VEE Pro Advanced Techniques* manual.

Listing ActiveX Automation Type Libraries

To find the automation objects installed on your computer, click `Devices ⇒ ActiveX Automation References`.

Note For information about ActiveX Control References, refer to Chapter 10, "Using Operator Interfaces." Refer also to Chapter 12, "Platform Specifics and Web Monitoring."

`Devices ⇒ ActiveX Automation References` lists the Type Libraries that are installed on your PC. Each application and ActiveX Component that can be an Automation Server registers a Type Library. VEE displays what is available on your PC. These libraries include information about the functionality of the application or component that is exposed to ActiveX clients.

Type libraries typically consist of a set of classes. Some classes can be created by the programmer. Other classes are always created by the application or component. Classes consist of properties, methods, and events, although not all have to be present. The Type Library provides both the programmer and the VEE environment with information necessary to utilize the application or component using ActiveX interfaces.

When you put a check next to a Type Library in the ActiveX Automation References box, the library objects become available for use in a VEE program. For example, in Figure 6-1, Microsoft Excel 9.0 is checked.

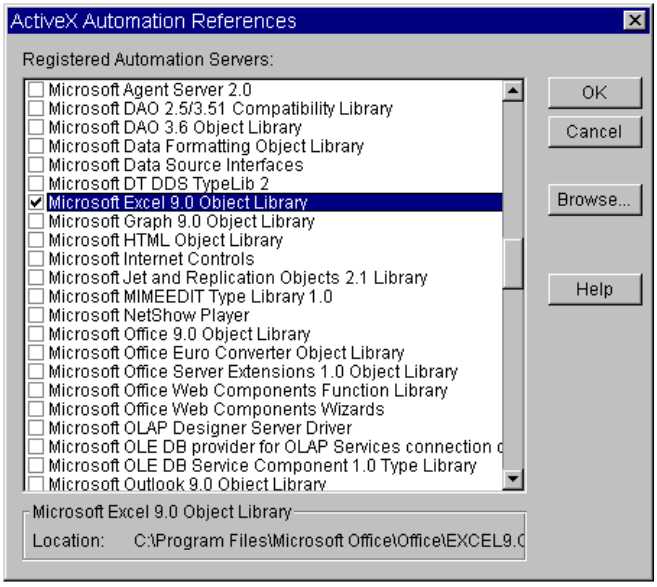


Figure 6-1. The ActiveX Automation Reference Box

Creating and Using ActiveX Programs with Agilent VEE

VEE includes a data type called `Object` for ActiveX programs. A VEE object with the data type specified as `Object` is a pointer to something or some data held by the Automation Server. For example, an `Object` could point to a worksheet inside MS Excel, or to a cell inside that worksheet. (Technically, an `Object` is a pointer to an IDispatch interface returned by MS Excel or the Server.)

For example, if you select `Data` ⇒ `Variable` ⇒ `Declare Variable`, set the `Name` to `App`, and set the data type as `Object`, you can use the variable `App` to point to an ActiveX Automation object such as the Excel Automation Server. Figure 6-2 shows an example of a data type `Object`.

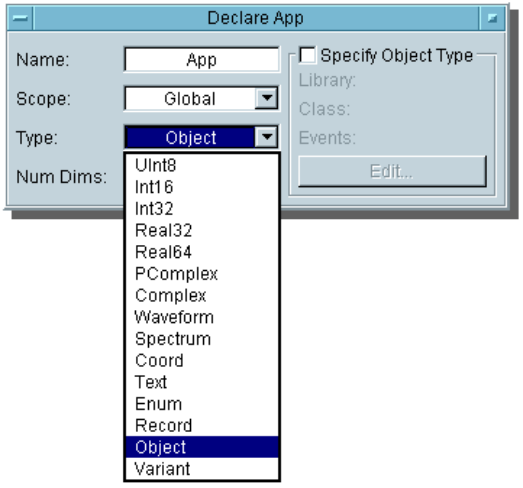


Figure 6-2. Example of Data Type "Object"

Performing Operations Using ActiveX Statements

To communicate with an ActiveX Automation server, such as the Excel Automation Server, enter ActiveX commands in a VEE `Formula` object. For example, Figure 6-3 shows a VEE `Formula` object that has been named `Set Up Excel Worksheet`. It contains a list of commands to set up an Excel worksheet to display the results of a test.

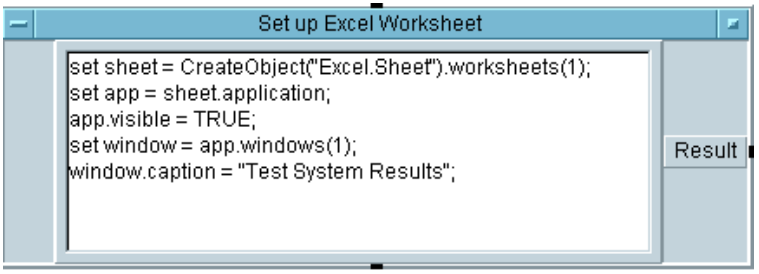


Figure 6-3. Commands to Set Up Excel Worksheet to Display Test Results

VEE uses standard Microsoft Visual Basic syntax to create the commands or statements like those shown in Figure 6-3. The commands or statements perform three types of operations: *get properties*, *set properties*, or *call methods*.

- *Get property* statements usually refer to getting some type of data. The syntax is <*object*> . <*property*>. For example, `sheet.application` gets the `application` property of the `sheet` object.
- *Set property* statements usually refer to setting some type of data equal to something. The syntax is <*object*> . <*property*> `=` <*property type*>. For example, `object.property = MaxSize` sets a property.
- *Call methods* call a `method`. A `method` requests the object to perform an action. `Methods` have parameters that allow data to be passed in and returned. The syntax is <*object*> . <*method*>*(parameters)*.

Note

The syntax for data type `Objects` looks similar to the VEE syntax for getting a Record field, `rec.field`, and calling a `UserFunction`, `myLib.func()`, so it is important to assign descriptive names to variables.

Using CreateObject and GetObject

Notice that one of the statements in `Set Up Excel Worksheet` in Figure 6-3 contains the `CreateObject()` function call. `CreateObject()` and `GetObject()` are functions in the VEE `Function & Object Browser`, and they are designed specifically to return a pointer to an ActiveX object in VEE.

For example, `CreateObject("Excel.Sheet")` starts up Excel and returns a reference to a workbook in it. (The Microsoft statement “sheet” returns a workbook.) Use `GetObject()` to get something or some data that already exists in a running Excel, or to load a file into a running Excel.

`CreateObject` and `GetObject` are located under `Device` ⇒ `Function & Object Browser`, `Type: Built-in Functions`, `Category: ActiveX Automation`. Figure 6-4 shows an example `CreateObject` and `GetObject`.

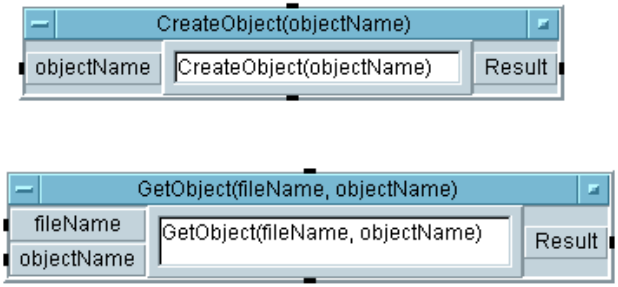


Figure 6-4. CreateObject and GetObject

Sending Agilent VEE Data to MS Excel

This section introduces the VEE objects and MS Excel function calls for generating reports.

Lab 6-1: Sending Agilent VEE Data to MS Excel

In this lab, you will generate virtual test data for MS Excel. (The example uses MS Office 2000 and the MS Excel 9.0 Object Library, and should also work with MS Office 97 and the MS Excel 8.0 Object Library.) After referencing the right Automation Type Library, you will declare some global variables of the `Object` type and put them in a `UserFunction` called `globals`. The global variables simplify the program and make it easier to understand.

Note

The VEE programs for many of the lab exercises and programming examples in this manual are included in VEE, under `Help` ⇒ `Open Example...` ⇒ `Manual` ⇒ `UsersGuide`.

1. Reference the Automation Library. Click `Device` ⇒ `ActiveX Automation References...`, select `Microsoft Excel 9.0 Object Library`, and click `OK`.

2. Create a `UserFunction` to store the global variables. Click `Device` ⇒ `UserFunction`. Rename it `globals`. (For more information about `UserFunctions` refer to Chapter 8, “Using Agilent VEE Functions,” on page 293.)

3. Click `Data` ⇒ `Variable` ⇒ `Declare Variable` and place it to the left inside `globals`. Change the `Name` to `sheet`. Change the `Type` to `Object`. Other items appear in the dialog box. For this exercise, you do not need to specify the `Object Type` and `Class`. (The `Type` and `Class` are specified in another example in this chapter.)

4. Clone this object three times, and rename the other objects as follows: `app`, `range`, and `window`. Size and move the `globals UserFunction` below `Main`. It should look like Figure 6-5.

5. After you have compared the entries to Figure 6-5, iconize the four objects.

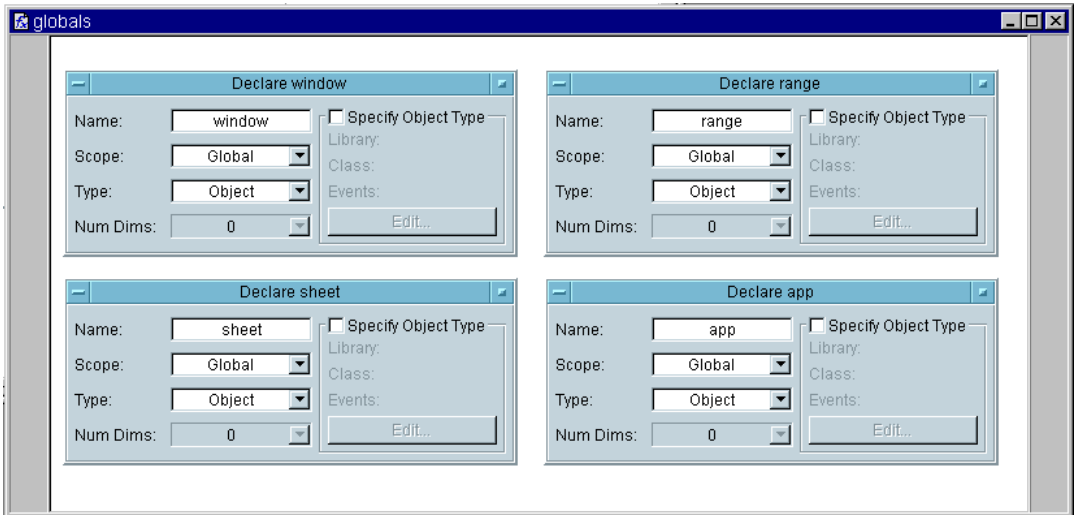


Figure 6-5. The Globals UserFunction

Notice that by using the datatype `Objects` in the `globals UserFunction`, you could specify the `Object Type and Class`. There are two reasons to specify `Object Type` and `Class`: more specific type checking, and catching events.

More Specific Type Checking: For example, if you specify an `Object app` as being of type `Excel.Application`, then only an `Object` of type `Excel.Application` can be assigned to `app`. Assigning an `Object` of type `Excel.worksheet` or `Word.bookmark` will cause an error.

Catching Events: You could also use a VEE `UserFunction` to catch various *events* that could occur in the application, such as a right-button-down in the MS Excel worksheet. For any of these types of events, you can specify a VEE `UserFunction` to handle the event and pass information back to MS Excel.

Events are useful for ActiveX Controls, where you need a way for the control to communicate back to VEE. For more information, refer to the *VEE Pro Advanced Techniques* manual.

6. Open the `UserFunction globals` object menu and click `Generate ⇒ Call`. This generates a `Call globals` object configured correctly. Place it to the left in the `Main` window and iconize the `globals UserFunction` window.

7. Click `Device ⇒ Formula` and place it in the upper center of the Main window. Rename it `Set up Excel Worksheet`. Connect the `globals` sequence out pin to the `Formula` sequence in pin. Delete the input terminal `A` from `Set Up Excel Worksheet` (open the `Object` menu and select `Delete Terminal ⇒ Input`.)

8. Inside `Set up Excel Worksheet`, enter the lines shown in Figure 6-6. Notice that semicolons are used for line separators, just as in ANSI C.

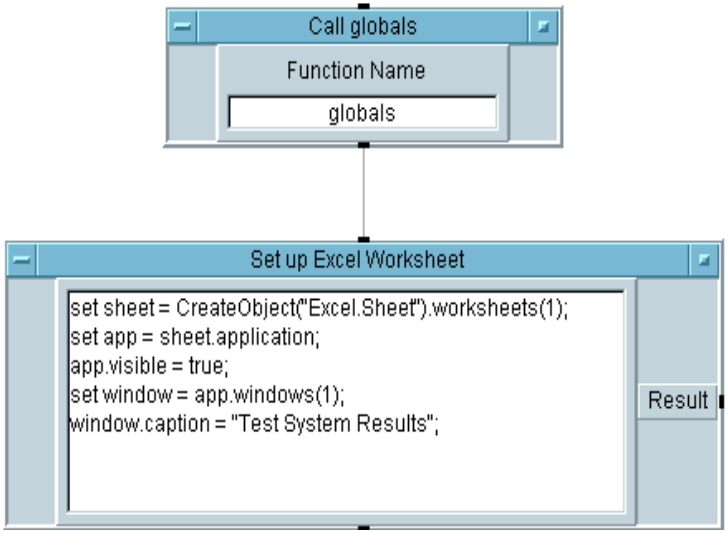


Figure 6-6. Setting Up the MS Excel Worksheet

The commands in the `Formula` object `Set Up Excel Worksheet` are as follows:

set sheet =

The keyword `set` is used to assign or set whatever is on the right-hand side of the assignment operator (in this case, the equal sign) to the variable on the left-hand side of the expression. For example, `set app` sets the application `sheet.application`, which has been defined as an Excel worksheet.

CreateObject ("Excel.Sheet").

Creates a new instance of the Automation Server (in this case, MS Excel) and creates an empty sheet (Excel terminology for a new workbook). Each Automation Server has its own terminology, but the syntax is the same. For example, to create a report in MS Word, you would enter `CreateObject("Word.Document")` to run Word and create a blank document.

If the `set` keyword is used, the right-hand side object pointer itself is assigned to the left-hand side variable. If `set` is not used, then the default property (often the name) of the right-hand side is assigned to the left-hand side. For more information, refer to the *VEE Pro Advanced Techniques* manual.

worksheets(1);

Now that Excel is running with a new workbook in it, with `CreateObject("Excel.Sheet")`, you want to address the first worksheet in it. Add `worksheets(1)` to the statement, so the entire statement reads:

```
setsheet =
CreateObject("Excel.Sheet").worksheets(1);
```

This sets `sheet` to `Sheet 1` of the report. (To see an example, open MS Excel and select `File` ⇒ `New` to create a new workbook. You will notice there are several sheets in it labeled `Sheet1`, `Sheet2`, and so on. You want `Sheet1`.)

set app = sheet.application;	Asks Excel for a pointer to the entire application, and not just the worksheet itself, by asking the worksheet for its property `Application` and setting it to the variable `app`.
app.visible = true;	Sets the `app`'s visible property to `true` in order to display Excel on screen (so that you can see it).
set window = app.windows(1);	References the first window.
window.caption = "Test System Results";	Sets the first window's caption to "Test System Results."

Note

For more information about the Application Server libraries, refer to the many books available about ActiveX Automation and MS Visual Basic. You can probably find information on the World Wide Web about ordering books such as Office 2000 or the Office 97 Visual Basic Programmer's Guide. The books will help you with VEE as well, since VEE syntax is very similar to MS Visual Basic.

9. Create a `Formula` object (under `Device` ⇒ `Formula`). Clone the `Formula` object to create a second `Formula` object. Create a `For Range` object (under `Flow` ⇒ `Repeat` ⇒ `For Range`). Rename the objects, connect them, and configure them as shown in Figure 6-7. (Be sure to delete the input terminal on the `Formula` object `Fill in Title`.)

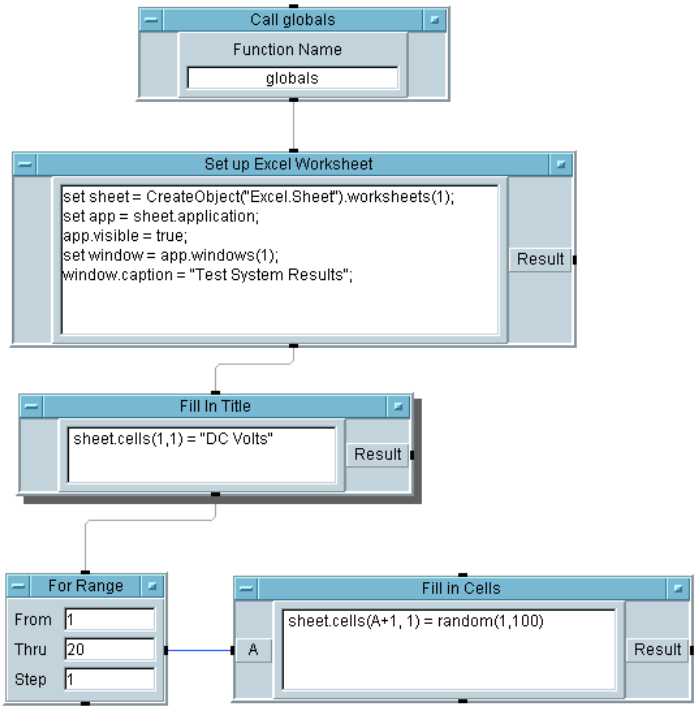


Figure 6-7. Adding the Title and Data to the Sheet

The instructions in the `Formula` objects and the `For Range` object are described as follows:

sheet.cells(1,1) = "DC Volts"

Refers to the first row and column in the Excel worksheet. The text DC Volts will be placed there. This sets the default property (which is value) of cell `(1,1)`, to `"DC Volts"`.

sheet.cells(A+1,1) = random(1,100)

This statement is shorthand for `sheet.cells(A+1,1).value=random(1,100)`. The worksheet cell at row `A+1, col 1` gets the row number by adding `1` to the input pin `A` value but stays in column `1`. The value between `1` and `100` returned by `random` is assigned to the specified cell in the worksheet.

from 1 thru 20, step 1 (the For Range object) As the `For Range` object outputs the integers from `1` to `20`, `Fill in Cells` puts the random number in the specified cell.

10. Create a `Formula` object and an `AlphaNumeric` object, rename, configure, and connect them as shown in Figure 6-8.

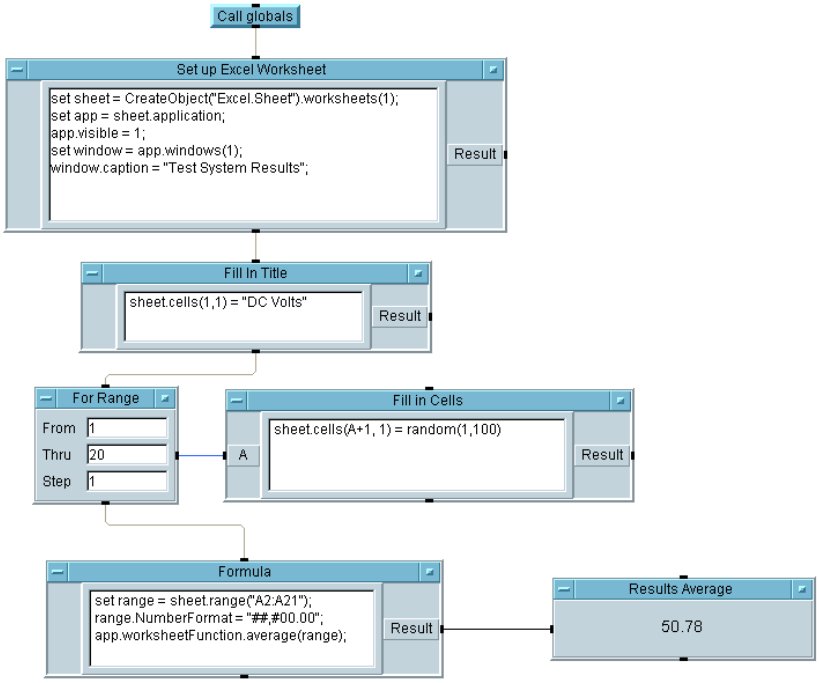


Figure 6-8. The Results Average Program

The entries in the `Formula` object are as follows:

set range = sheet.range("A2:A21");	Sets the VEE variable range to reference the range `A2` to `A21` on the Excel worksheet. (`A` refers to the first column in a worksheet.)
range.NumberFormat = "##,#00.00";	Assigns the format to each of those cells with the pound signs (`#`) allowing for larger numbers, if necessary.
app.worksheetFunction.average (range);	Calls an Excel method `average()` that returns the average value of the designated `range` of values, which is displayed in `Results Average`.

11. Save the program as `results_average.vee`. Run the program. MS Excel will launch with a worksheet like the one shown in Figure 6-9.

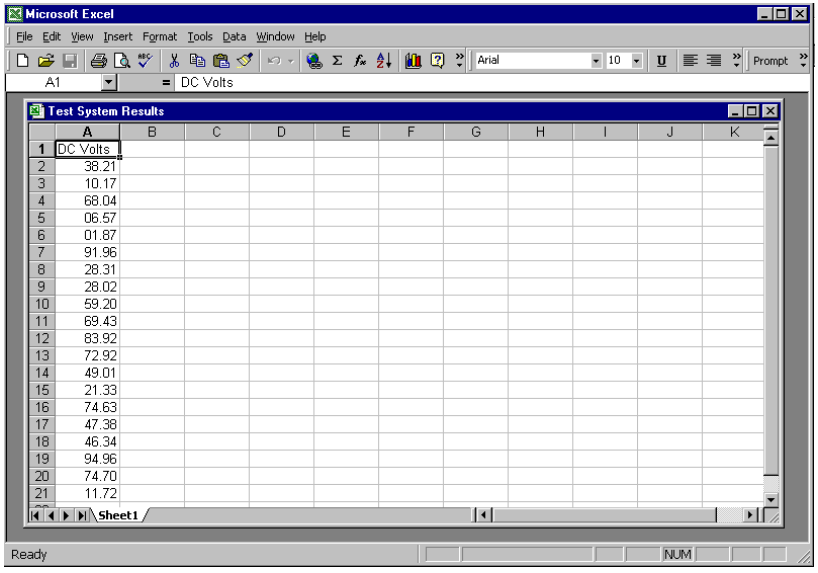


Figure 6-9. Excel Worksheet for "Results Average" Program

Creating an Agilent VEE to MS Excel Template

In this exercise, you will create a program to display an array of VEE test data in MS Excel. You can use this program as a template for displaying the results of other tests in MS Excel spreadsheets.

Lab 6-2: Creating an Agilent VEE to MS Excel Template

1. Open `results_average.vee`.

2. Change the `For Range` object to loop `10` times.

3. Add the input `B` to `Fill in Cells` and change the statement inside to read: `sheet.cells(A+1,1) = B[A-1].`

   Click `Device` ⇒ `Formula`, rename it to `Array of Test Data`, and enter the embedded functions `randomize(ramp(20), 4.5, 5.5)` to create a random array of `20` elements with values from `4.5` to `5.5`. Delete the input pin and connect the data output pin to the `B` input of `Fill in Cells`.

4. Change the range in the `Formula` box on the bottom of the screen from `A21` to `A11`. The statement should now read:
   `set range = sheet.range("A2:A11");`

5. Save the program as `report_template.vee` and run it. Compare it to the Excel worksheet as shown in Figure 6-10 and the complete program as shown in Figure 6-11.

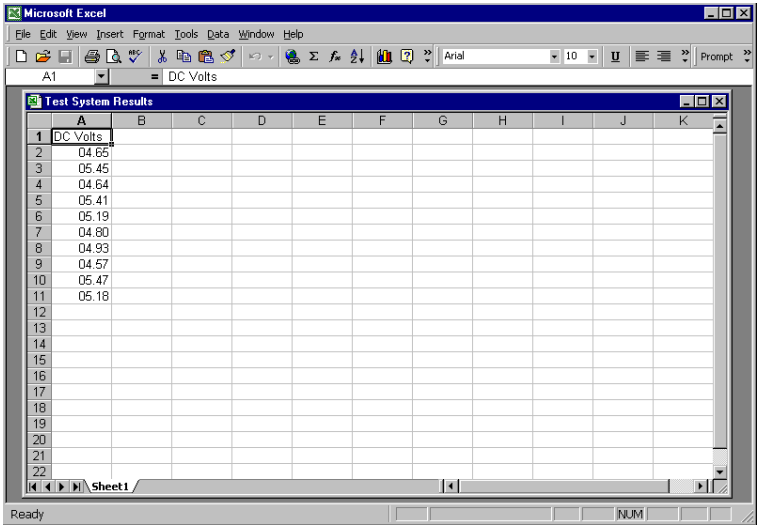


Figure 6-10. Excel Worksheet for Array of Test Data

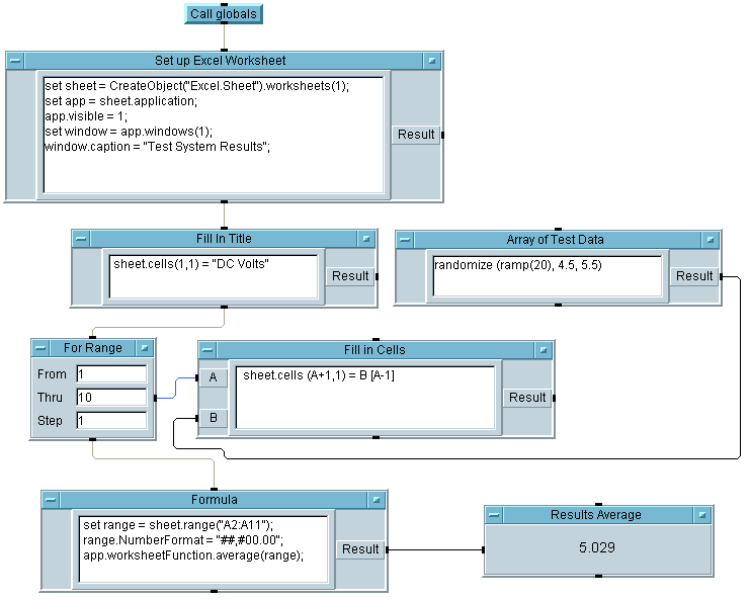


Figure 6-11. Program for Array of Test Data

You can re-use this program as a template for displaying test results in MS Excel. You simply put the test data into arrays and modify the rest of the template to fill in the appropriate cells in the right format.

To find additional methods and properties in the MS Excel library, look at the `Function & Object Browser` and choose the `ActiveX Objects` under `Type` and `Excel` under `Library`. You can choose a `Class` or `Member` and press `Help` to get Help provided by the Automation Server author (in this case, Microsoft). For more complete information on these libraries, consult Microsoft documentation.

On Your Own

Generate a waveform and strip out the `Time Span` to get an array. Create a VEE object for MS Excel with a worksheet and set it to an `Object` variable. Make the application visible. Then put the 256 point array into the worksheet range `"A1:A256"` in one step, instead of one cell at a time.

HINTS: Use an `Unbuild Waveform` object. Use the `[a]` build array syntax to create a 2D array from a 1D array. Then call the function `transpose()` to make it a 256 x 1 array instead of a 1 x 256 array for Excel to accept it in one step, as shown in Figure 6-12.

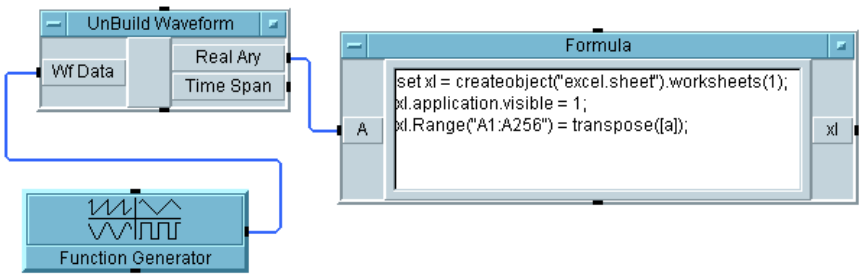


Figure 6-12. Program for On Your Own Exercise

Extending Capabilities With MS Excel

Figure 6-13 shows a more elaborate example of a program to display test results in MS Excel. You can see how knowledge of a few more calls in the MS Excel library can expand the template for displaying VEE data in MS Excel.

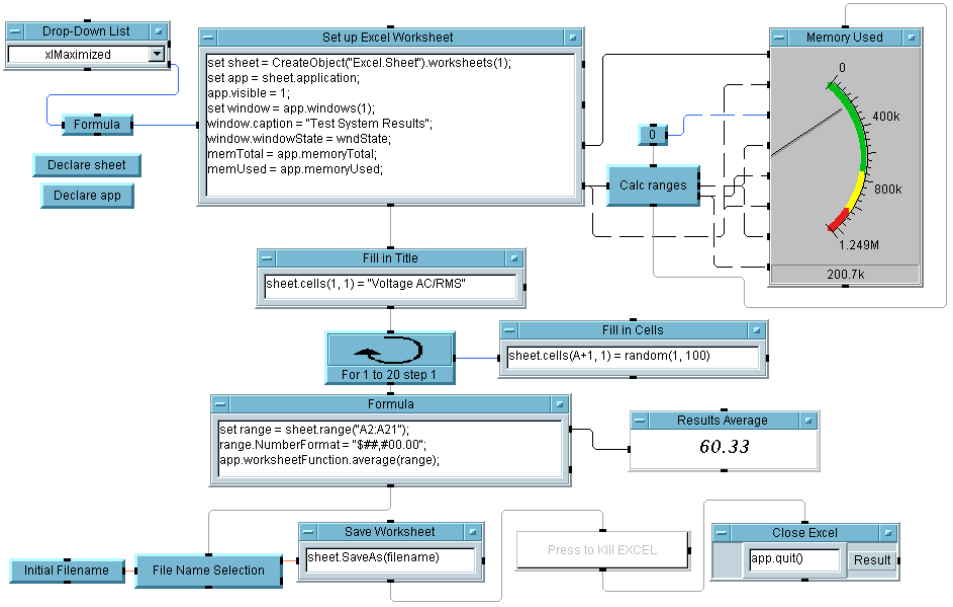


Figure 6-13. A VEE to MS Excel Program Example

The entries in Figure 6-13 are as follows:

MS Excel Window Size

Notice the Drop-Down List in the upper right work area. This allows you to choose one of three options `xlMaximized`, `xlMinimized`, `xlNormal` to select the size of the worksheet window inside Excel when it comes up. Each window size is associated with a number, which VEE calculates and puts in the `wndState` variable. This value is then assigned to the `windowState` property in the Excel library.

Memory Tracking	(Click `Show Terminals` in the `Properties` boxes on the `Formula` and `Meter` objects.) Notice the `memoryTotal` and `memoryUsed` properties in the Excel library that are assigned to the VEE variables `memTotal` and `memUsed`. These values are then used to calculate the ranges to configure a VEE meter before it displays the memory being used by MS Excel.
Number Format	Notice how easy it is to add a dollar sign to the number format.
sheet.SaveAs (filename)	The `SaveAs()` method is being called from the Excel library to automatically save the worksheet. Notice that a `File Name Selection` box (from the `Data` ⇒ `Dialog Box` menu) is used to display the pop-up `Save As` box from VEE. The file name you select is then used as a parameter in the `Excel SaveAs()` method call.
Press to Kill Excel	The Confirm (`OK`) button has been used to signal when you want to close Excel.
Close Excel	The `quit()` method is called to tell MS Excel to exit.

Using MS Word for Agilent VEE Reports

This lab describes how to display VEE test information in an MS Word document, including text, a time stamp, and a screen dump of a VEE pop-up panel with an XY Display. Consult Microsoft documentation to find out more elaborate ways of controlling MS Word from other applications using ActiveX Automation.

Lab 6-3: Using MS Word for Agilent VEE Reports

To begin, follow the steps to declare five variables as type `Object`.

1. Click `Device` ⇒ `ActiveX Automation References...` and select `Microsoft Word 9.0 Object Library`.

2. Click `Data` ⇒ `Variable` ⇒ `Declare Variable`.

   a. Change the Type field to `Object`. Clone it four times.

   b. Name the five object variables `App`, `Doc`, `Wnd`, `Sel`, and `Bmp`.

   c. Select `Specify Object Type` on all of them. The advantages of declaring the particular `Class` within a `Library` are as follows: VEE can do type checking for program errors, and you can catch events from the Automation Server.

   d. Then click the `Edit...` button and select `Word for Library` in each case. Select the following `Classes`:

      `App` will use `Application`
      `Sel` will use `Selection`
      `Wnd` will use `Window`
      `Doc` will use `Document`
      `Bmp` will use `Shape`

   e. Select `Enable Events` where the class permits it. Iconize these five icons. See Figure 6-14 for the open view of these variables.

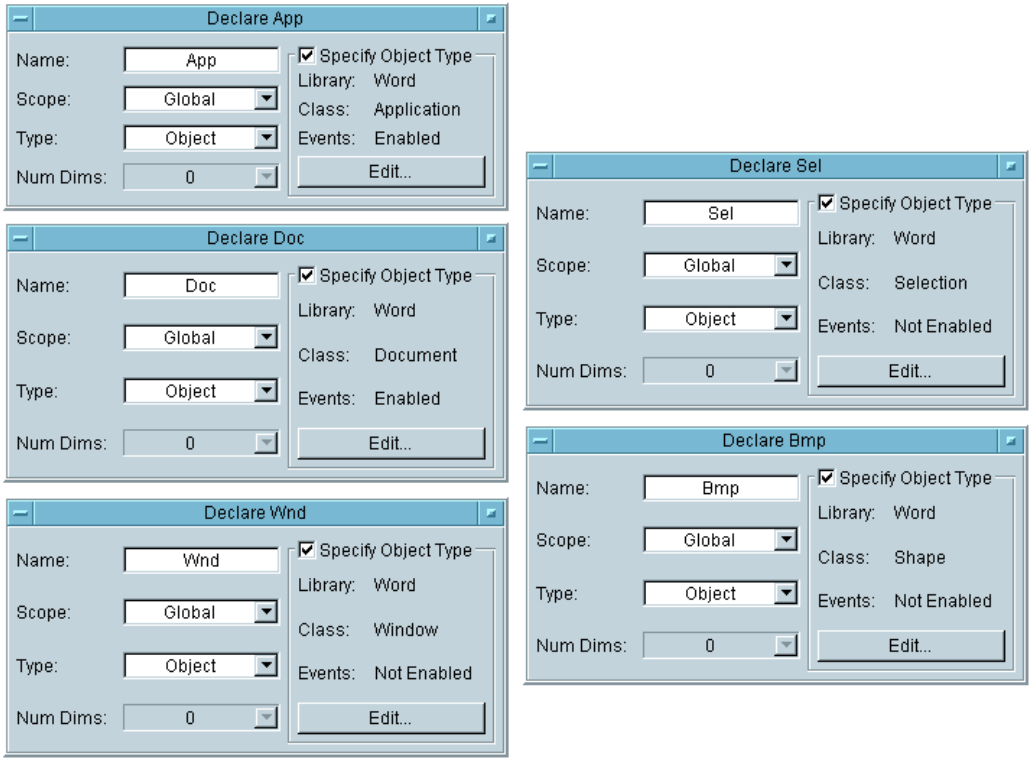


Figure 6-14. Object Variables

3. Create a `UserFunction` called `Graph`, which uses a `Function Generator` virtual source to send a sine wave to a `Waveform (Time)` display. Create a panel view of the display only. Then generate a `Call Graph` object in the Main window. (Recall that the `UserFunction` object menu includes an easy way to generate a call.)

Now create a bitmap file of the Panel with the Waveform display to use in the report in MS Word.

4. To create a file name for the bitmap, click `Device` ⇒ `Formula`. Rename it `Image Filename`. Enter `installDir() +`

`"\\panel.bmp"` in the `Formula` input field. (Use the escape sequence `\\` to specify the ASCII character `\`.) Delete input terminal `A`.

If you installed in `c:\Program Files\Agilent\` for example, you would then generate the following text string on the `Result` output pin: `C:\Program Files\Agilent\VEE Pro 6.0\panel.bmp`.

5. Create another `Formula` object and enter `savePanelImage("Graph", FileName, 256)`. Rename the input terminal to `FileName`.

   This saves the screen dump from the `UserFunction Graph` in the `panel.bmp` file in the installation directory at a color depth per pixel of 256.

6. Create another `Formula` object and enter the statement:
   `Set App = CreateObject("Word.Application")`
   This launches MS Word and assigns the object variable `app` to refer to this instance of the application. Delete input terminal `A`. Connect `Call Graph`, `ImageFileName`, and `savePanelImage` as shown in Figure 6-15.

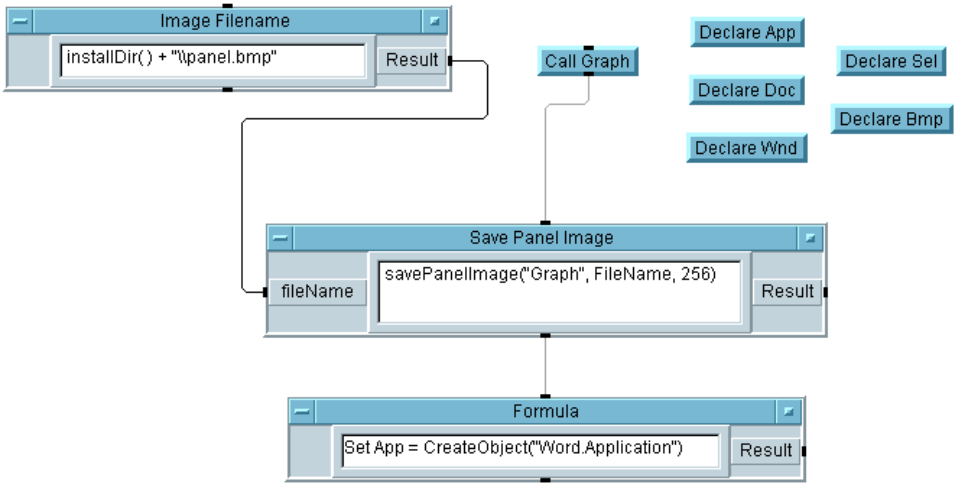


Figure 6-15. Beginning of Lab 6-3 Program

7. Click `Device` ⇒ `Formula` and enter the statements shown in Figure 6-16, which are also described below. Rename input terminal `A` to `FileName`. Connect the data input and sequence input pins as shown in Figure 6-16.

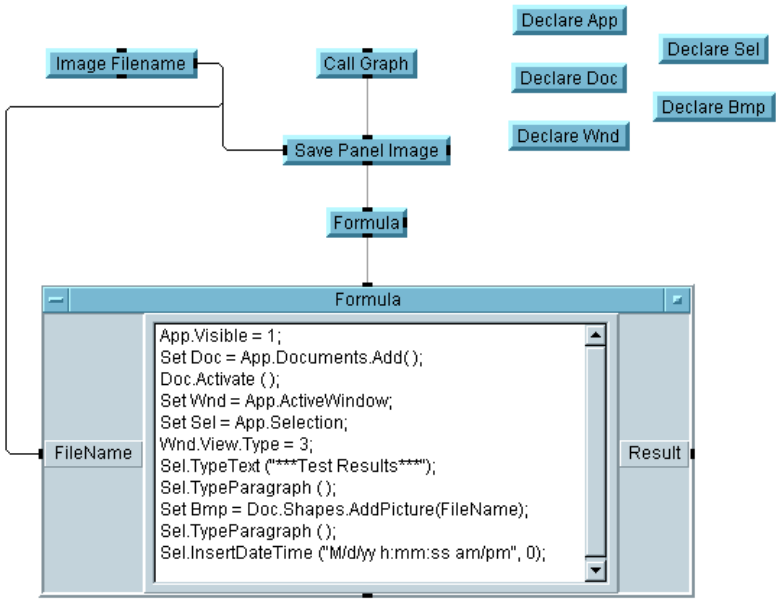


Figure 6-16. Adding the ActiveX Statements

In Figure 6-16, notice that you can nest property and method calls together with the `Object`'s dot notation. Refer to ActiveX documentation to find the right properties in the target applications. You can use the properties and methods described in this chapter to begin generating test and measurement reports. The entries in the `Formula` object are as follows:

App.Visible = 1; Makes MS Word visible on the screen.

Set Doc = App.Documents. Add();	Adds a Document in MS Word and assigns it to the `Object` variable `Doc`. Note: In the Excel example, Excel was started with a blank worksheet using `CreateObject(Excel.Sheet)`. In this example, Word is started and the method `Add()` adds an empty Document to it. Either application can be created either way.
Doc.Activate();	Activates the Document above.
Set Wnd = App.Active Window;	Takes the document in the active window and assigns it to the `Object` variable `Wnd`.
Set Sel = App.Selection;	Puts focus (selection) into the document and assigns this to the `Object` variable `Sel`. This allows you to insert text.
Wnd.View.Type = 3;	Specifies the type of window for displaying the document. The 3 indicates a normal window size. A 1 would iconize the window. Note: The 3 is used here instead of the constant wdPageView because the constant is missing from the Office 2000 Type Library.
Sel.TypeText(* Test Results ***), Sel.TypeParagraph();**	Puts the title `*** Test Results ***` in the document and issue a carriage return/line feed.
Set Bmp = Doc.Shapes. AddPicture(FileName);	Puts the `panel.bmp` bitmap into the document and assigns this call in the `Shapes Class` to the `Object` variable `Bmp`.
Sel.TypeParagraph(); Sel.InsertDateTime (M/d/yy h:mm:ss am/pm, 0);	Puts a time stamp in the document.

8. Add three more `Formula` objects and one `If/Then/Else` object, configure, and connect them as shown in Figure 6-17.

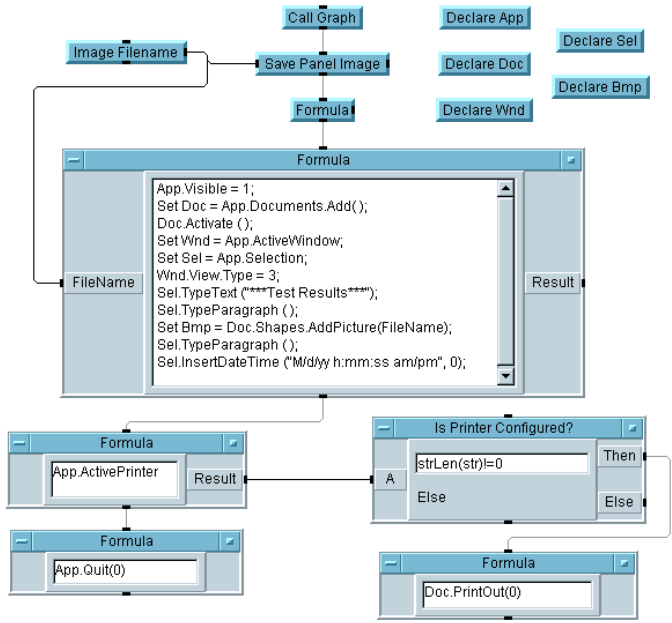


Figure 6-17. The Complete Program for Report in MS Word

The entries in the additional objects are as follows:

App.ActivePrinter	Requests the default printer in a string including its port.
strLen(str) != 0	Makes sure that `ActivePrinter` has located a configured printer (if the string on the input is not null, then...), then outputs a 1 `(=TRUE)` on the `Then` pin, which pings the `Formula` object containing the `PrintOut` call.
DocPrintOut(0)	Prints the document.
App.Quit(0)	Closes the MS Word application.

9. Run the program. It should look like Figure 6-18. (If the colors look strange in the screen dump, iconize any open applications, so the PC has a full palette of colors to work with.)

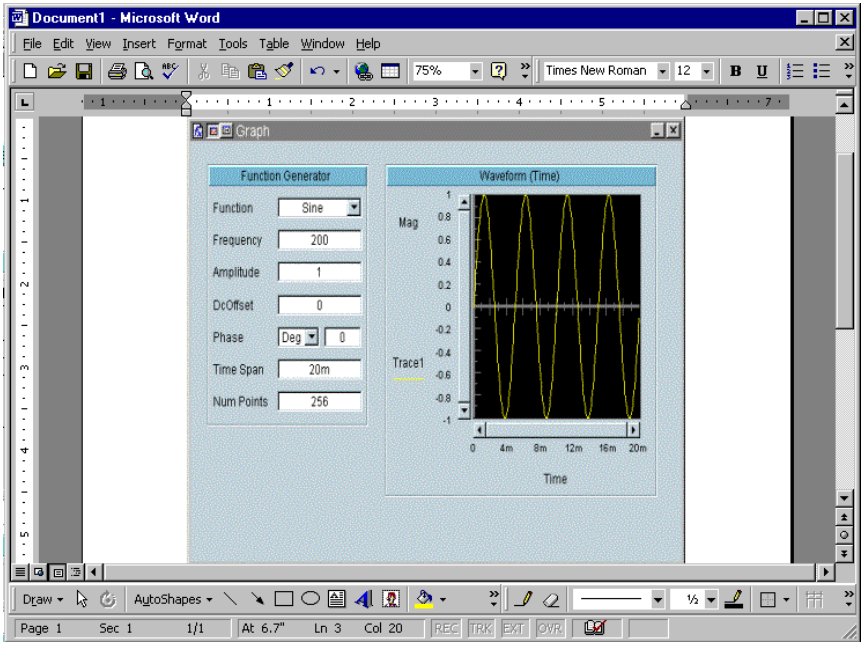


Figure 6-18. The MS Word Document Created by Lab 6-3

For more information about controlling MS Excel and MS Word using ActiveX Automation, refer to Microsoft documentation. Remember that you can also control other Server applications that support ActiveX Automation, sometimes just called Automation, or OLE Automation.

For more information about using ActiveX controls, refer to Chapter 10, “Using Operator Interfaces.” For more information about using ActiveX from a MS Visual Basic program to control VEE, refer to Chapter 12, “Platform Specifics and Web Monitoring.”

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary, before going on to the next chapter.

- Describe the basic concept behind ActiveX Automation in VEE.
- Send data from VEE to MS Excel.
- Use a generic template to send arrays of test data to an MS Excel worksheet. (Make sure you know how to send an array to the spreadsheet in one step.)
- Employ some of the extended capabilities of the MS Excel library, such as finding out the memory used by a program.
- Send text, a time stamp, and a display bitmap to MS Word from VEE.

7

Integrating Programs In Other Languages

Integrating Programs In Other Languages

In this chapter you will learn about:

- The `Execute Program` object
- Using operating system commands from VEE
- Making VEE programs portable across platforms

Average time to complete: 1 hour

Overview

In this chapter, you will learn the easiest way to integrate compiled programs and operating system commands with VEE. One of the great advantages of VEE is that it integrates well with other applications and programs. Furthermore, by using ActiveX, you can use components from other programs. (For more information, refer to Chapter 6, “Creating Reports Easily Using ActiveX.”)

In VEE, the `Execute Program` object specifies programs and parameters and uses operating system commands. There is an `Execute Program` object for the PC, and another for HP-UX. This chapter includes a lab exercise with the `Execute Program` object for PCs and another with the `Execute Program` object for HP-UX.

Understanding the Execute Program Object

In addition to ActiveX Automation, there are three ways to run programs in other languages from VEE:

1. Use the `Execute Program` object to escape VEE and run another program, application, or operating system command. This method is the most versatile and easy to use.

2. Link compiled functions in other languages to VEE, either through Shared Libraries in UNIX operating systems or Dynamic Link Libraries on the PC. Although this is slightly more difficult to execute, it gives you significant performance gains. For more information on Shared Libraries, refer to "Overview of Compiled Functions" on page 414. For more information about Dynamic Link Libraries, refer to "Using Dynamic Link Libraries" on page 417.

3. Use a method specifically designed for Rocky Mountain Basic programs. For more information, refer to "Communicating with Rocky Mountain Basic Programs" on page 442.

The `Execute Program` object is located in the `I/O` menu. There is one object for the PC and one for HP-UX, as shown in Figure 7-1 and Figure 7-2. Notice that in the PC version the `Execute Program` object does not use transaction I/O to communicate with programs, so you do not add data input and output pins to pass data to the compiled program.

The HP-UX version uses transaction I/O, so the example includes input and output pins and a program to illustrate how the object is used.

Using the Execute Program Object (PC)

Figure 7-1 shows the `Execute Program` Object on the PC.

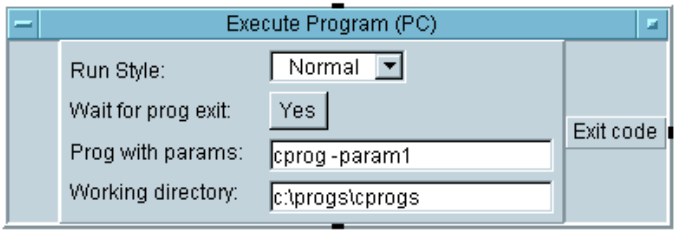


Figure 7-1. The Execute Program Object (PC)

Use the `Execute Program` object to run the following from VEE:

- Compiled programs written in other languages
- `*.BAT` or `*.COM` files
- MS DOS system commands, such as `dir`
- Any document or URL with a recognized extension. The “open” action is invoked in the files. If an “open” action does not exist, the default action is invoked with the file. An example of a URL would be `http://www.agilent.com/find/vee`.

The fields in the `Execute Program (PC) Object` are as follows:

Run Style Determines the window size. `Normal` specifies a standard window, `Minimized` specifies an icon, and `Maximized` specifies the maximum window size. The `Working directory` is the directory that holds any files related to the program.

Wait for prog exit

Specifies when to fire the sequence pin.

- When set to Yes, the sequence pin is not fired until the program finishes executing.
- When set to No, the sequence out pin fires before the specified program is done executing. Note that when launching documents or URLs, if the document or web site is loaded into an application that is already running, VEE does not wait until the application exits.

Prog with params

(Program with parameters) This field holds the same words you would type at a DOS prompt. For example, to run a program in C, enter the executable file name `- myprog.exe`. (You can omit the `.exe` extension.)

- If the program has parameters, they would follow the executable file name preceded by a hyphen, such as `myprog -param1 -param2`.
- To run a DOS system command, first run the DOS command interpreter with the `/c` option. For example, for Windows 95 and Windows 98, enter the command `command.com /c` <*system command*>.
- For Windows NT 4.0 and Windows 2000, enter the command `cmd /c` <*system command*>
- This option tells the command interpreter to read the string following the `/c` as a system command.

Using the Execute Program Object (HP-UX)

Figure 7-2 shows the `Execute Program` Object for HP-UX.

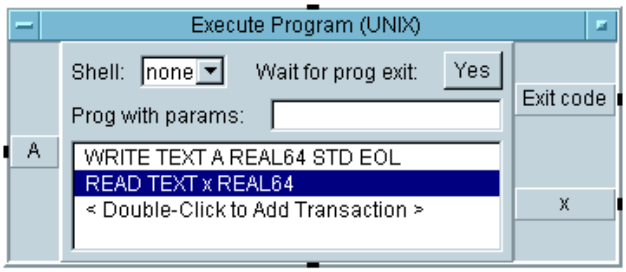


Figure 7-2. The Execute Program Object (UNIX)

HP-UX is designed to run a number of programs (called processes) concurrently. If VEE initiates another program, VEE is called the *parent* process and the program initiated is called the *child* process. The `Execute Program` object spawns a child process, either directly or through a command shell. The fields in the `Execute Program (UNIX)` object are as follows:

Shell Field

The `Shell` field opens a menu with the following choices: `none`, `sh`, `csh`, and `ksh`.

The first token in the `Prog with params` field is interpreted as the name of an executable program, and the following tokens are assumed to be parameters.

If you have shell-dependent features in the `Prog with params` field, such as standard input and output redirection (`<<` and `>>`), wildcards (`*`, `?`, `[a-z]`), or pipes (`|`), you need to specify a shell; otherwise, select `none` because it yields a faster execution speed.

Wait for prog exit field	The `Wait for prog exit` field toggles between `Yes` and `No`. Regardless of the setting, VEE spawns a child process, if one is not already active. All transactions specified in the `Execute Program object` are completed. ■ If set to `Yes`, the child process must terminate before the data output pins are fired. ■ If set to `No`, the child process fires the data output pins and remains active. The performance of your program is greater with this setting.
Prog with params	The name of an executable program file and command line parameters *-OR-* a command that will be sent to a shell for interpretation.

You may add input or output terminals to the `Execute Program` object. Data is received from a VEE program on an input pin, and then you perform a `WRITE TEXT` transaction to send the data to the child process. A `READ TEXT` transaction reads data from the child process, and places it on a data output pin for use by the VEE program.

You may also send the name of the program or shell command to the `Execute Program` object by adding the data input terminal labeled `Command`, which is available by adding data inputs.

Using a System Command

To call a compiled program in another language, you can type in the executable file and any parameters in the `Execute Program` object.

However, to execute an MS DOS system command, you must first run the DOS command interpreter. In this exercise, you will run the DOS command interpreter and execute an MS DOS system command.

Lab 7-1: Using a System Command (PC)

1. Select `I/O` ⇒ `Execute Program (PC)`. Click the `Prog with params` field to get a cursor, then type:

   ```
   command.com /c dir >> c:\bob
   ```

Note

Replace `command.com` with `cmd` for Windows NT 4.0 or Windows 2000. If the drive letter is different from `c:`, then substitute that drive letter in these instructions. On NT, you may have to specify a directory for which you have write permissions.

   (You may need to include the complete path of the `command.com` executable.) The command runs the DOS command interpreter, which runs the system command to display the current directory, and redirects the output (`>`) to the `bob` file instead of the computer screen.

   Leave `Yes` for the `Wait for prog exit` selection. Leave `Normal` for `Run Style`, and enter `c:\` for the Working directory.

2. Select `I/O` ⇒ `From` ⇒ `File` and place it below `Execute Program`. Connect the sequence out pin of `Execute Program` to the sequence in pin of the `From File` object.

   Click the `From File:` input field labeled `myFile` to get a list box, enter `c:\bob`, then click `OK`. (The program creates the file `bob` for you.)

3. Double-click the transaction bar to get the `I/O Transaction` box.

   a. Change `REAL64 FORMAT` to `STRING FORMAT`.

   b. Change `SCALAR` to `ARRAY 1D`.

   c. Click on the `SIZE: (10)` field to toggle it to `TO END: (*)`, then click `OK`. The transaction bar should now read: `READ TEXT x STR ARRAY:*` . This transaction will read the contents of the `bob` file.

4. Select `Display` ⇒ `Logging AlphaNumeric` and connect its data input pin to the `From File` data output.

5. Run the program. It should look like Figure 7-3.

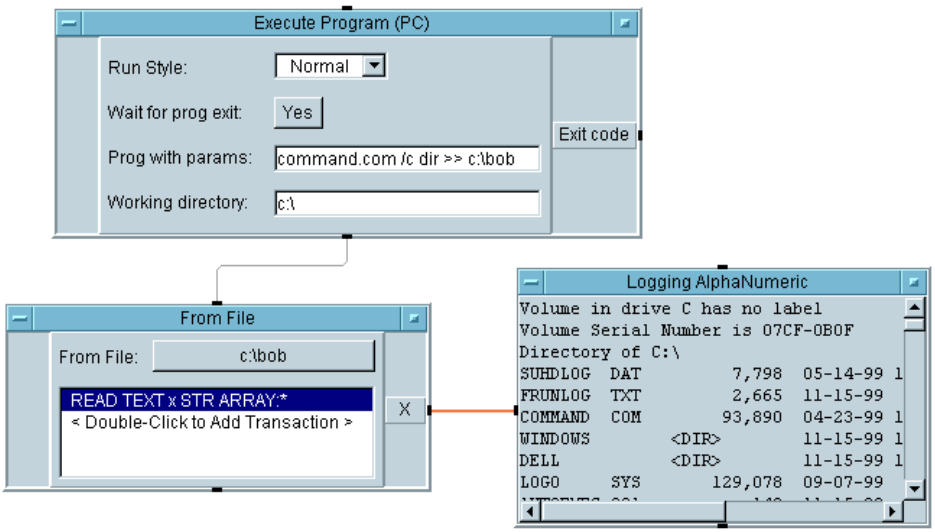


Figure 7-3. Listing the Files in a Directory (PC)

Lab 7-2: Listing the Files in a Directory (UNIX)

In this exercise, you will use the operating system command `ls`, which lists the filenames in a directory. Since this is not a shell-dependent command, you can set Shell to `none`. Then you will program a variation of this exercise using a shell-dependent feature, the pipe (`|`).

To list the files in an HP-UX program, use the `ls` command as follows.

1. Select `I/O` ⇒ `Execute Program (UNIX)` and place it in the upper-left work area.

2. Make sure the `Shell` field is set to `none` and the `Wait for prog exit` field is set to `Yes`. This ensures the `ls` command terminates before VEE continues with the program.

3. Click the `Prog with params` field and enter: `ls /tmp`. (You could specify any directory.)

4. Add a data output terminal. The default will be named `X`. There is no exit code from the program, so disregard the terminal labeled `Exit code`.

5. Double-click the transaction bar to get the `I/O Transaction` box. Edit the default variable a to an `X`, since data from the program will be read into that output terminal.

   a. Change `WRITE` to `READ`, to specify a `READ TEXT` transaction.

   b. Change `REAL64 FORMAT` to `STRING FORMAT`.

   c. Change the shape of the data from `SCALAR` to `ARRAY 1D`, to specify a one-dimensional array.

   d. Toggle the `SIZE:` button to `TO END: (*)`, since you do not know how many files are in the directory. Click `OK`. The transaction bar should now read `READ TEXT X STR ARRAY:*`.

6. Select `Display` ⇒ `Logging AlphaNumeric` and connect its data input pin to the `X` terminal on the `Execute Program` object.

7. Run the program. It should look like Figure 7-4.

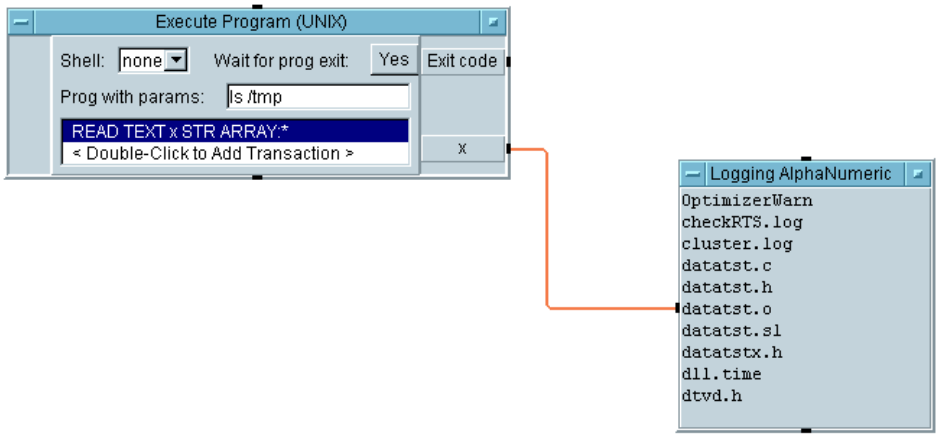


Figure 7-4. Listing the Files in a Directory (UNIX)

Listing the Files in a Directory Using a Shell

This variation of the last exercise uses a shell-dependent feature, a pipe (|), to send the output of one operating system command to another. The second command will be `wc`, which stands for `word count`. The `wc` command counts lines, words, and characters in the named files. The command `wc -l` filename counts the number of lines in the specified file. This example describes how to count the number of files in a directory, then display the number and the files.

1. Select `I/O` ⇒ `Execute Program (UNIX)`.

   a. Set the `Shell` field to `sh` to use the shell features, "|" and ";".

   b. Enter the command, `ls /tmp|wc -l;ls /tmp`, in the `Shell` command field.

   c. Add two data output terminals, one labeled `x` and the other labeled `Lines`.

d. Configure the first transaction as `READ TEXT Lines INT32`. Lines replaces the default variable `a`.

e. Configure a second transaction as `READ TEXT X STR ARRAY:Lines`. Enter `Lines` in the `SIZE` field to specify the length of the array.

2. Select `Display` ⇒ `AlphaNumeric` and Clone it. Connect one `AlphaNumeric` to the `Lines` output and another one to the `X` output pin.

3. Run the program. It should look like Figure 7-5.

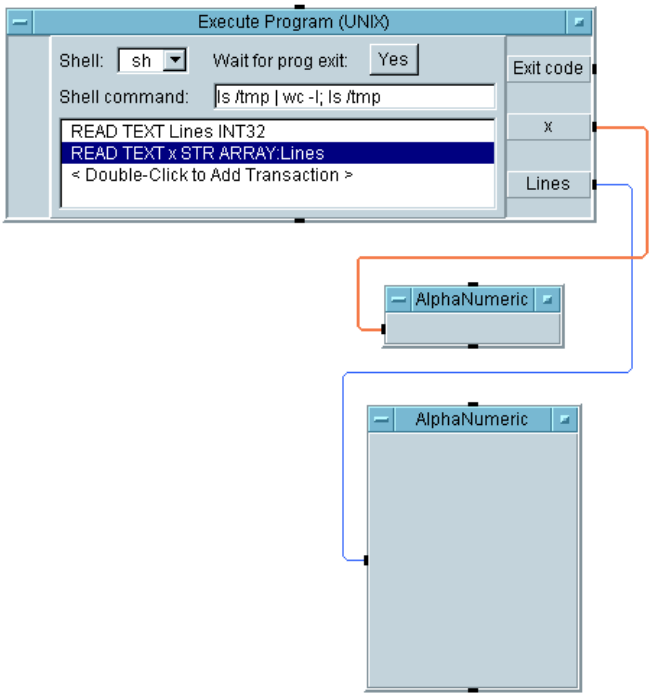


Figure 7-5. Using a Shell Command with a Pipe

Writing Programs That Port Easily

If you plan to integrate programs in other languages, write the VEE programs so that they ports easily between platforms. VEE includes system information objects in `Function & Object Browser` ⇒ `System Information`, as shown in Figure 7-6. These objects can also be used as functions.

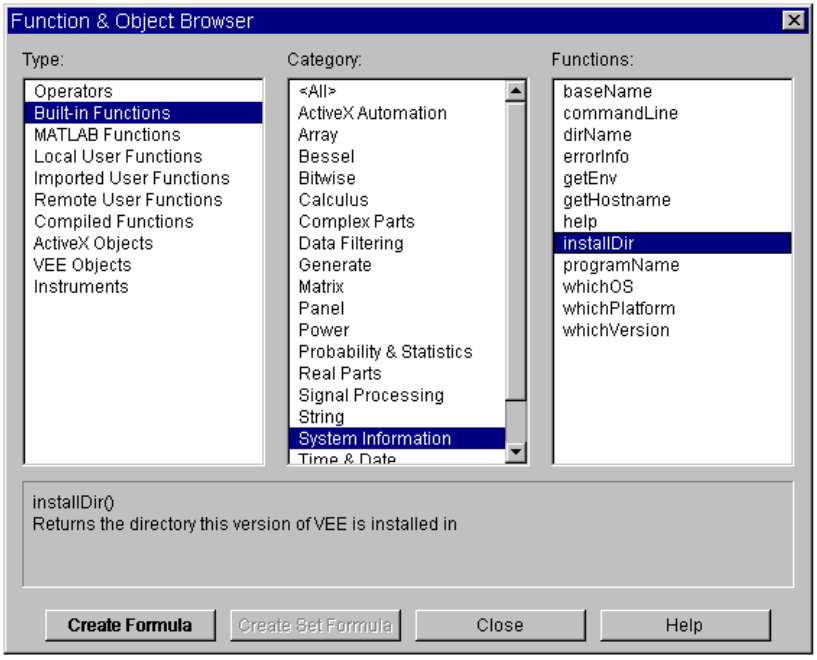


Figure 7-6. System Information Functions

The system information functions in the `Function &Object Browser` that are commonly used to enhance program portability are as follows:

installDir Specifies the VEE installation directory.

whichOS	Determines the operating system and sends out one of the following strings: `Windows_95`, `Windows_98`, `Windows_2000`, `Windows_NT`, `HP-UX`. The program can branch based on these results when incorporating programs in other languages. For example, look at `manual49.vee` in the `examples\manual` directory to see a program that uses `whichOS()` to make sure it imports the right type of library. On HP-UX, it would import a shared library. On a PC, it would import a Dynamic Link Library.
whichplatform	Determines the hardware system on which VEE is running, then returns a string indicating that platform.
whichVersion	Specifies the VEE version, which is useful for program maintenance and debugging.

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary, before going on to the next chapter.

- Explain the purpose of the `Execute Program` object.
- Give an overview of the configuration settings on the `Execute Program` object.
- Explain the general process of how the `Execute Program` object sends data to/from a program on a PC platform.
- Explain how the HP-UX platform differs.
- Run operating system commands from VEE.
- Create a program that will use the `whichOS()`, `whichPlatform()`, or `whichVersion()` object so that it will run on different operating systems.

8

Using Agilent VEE Functions

Using Agilent VEE Functions

In this chapter you will learn about:

- Defining a user function
- Creating, calling, and editing functions
- Creating, merging, importing, and deleting function libraries
- Finding functions in large programs
- Merging existing VEE programs with tests

Average Time to Complete: 1 hour

Overview

In this chapter, you will learn about VEE `UserFunctions`, compiled functions, and remote functions. Functions are re-usable, modular code that can help you significantly reduce the time it takes to develop tests. By re-using functions that have been created in previous programs, you can leverage existing work, reduce the code size of programs, and make it easier to maintain test programs. You can also use functions in groups, as libraries, which you can create and then merge into new programs. You can share functions among multiple programs and multiple developers.

Using Functions

Like many programming languages, VEE uses functions to create subprograms that perform specific tasks. The lab exercises in this chapter describe how to create, call, and edit VEE user-defined functions. You will also learn how to create libraries of functions, which can be merged into programs in the development phase or imported at runtime.

Defining an Agilent VEE Function

There are three types of user-defined functions in VEE. The overview of each type of function is as follows:

1. `UserFunctions`

   - To *create* a `UserFunction`, you select `Device` ⇒ `UserFunction`, or click `Edit` ⇒ `Create UserFunction` with several objects selected.

   - To *call* a `UserFunction` from different places in a program, you use the `Call myFunction` (Device ⇒ Call) object or use an expression within an object (from `Formula`, for example). You can also generate call objects in the Main program from the `UserFunction`, using the `UserFunction` object menu and selecting choices such as `Generate` ⇒ `Call`.

   - To *edit* a `UserFunction`, you click on `Edit` ⇒ `Edit UserFunction...` and select the appropriate `UserFunction` from the list box presented.

   - To *transfer* `UserFunctions` from one program to another, you merge the `UserFunctions` during program development or import them at runtime (`Device` ⇒ `Import Library`).

2. Compiled Functions

   - To *create* a compiled function, you work outside of VEE using a compiled language. You then put the functions into a library, such as a DLL.

   - To *link* a compiled function to a program, you use the `Import Library` object, which links the library to VEE at run time. (For a more detailed discussion, refer to Chapter 11, "Optimizing Agilent VEE Programs.").

   - To *call* a compiled function, you use the `Call myFunction` object or write an expression within a VEE object.

3. Remote Functions

   - Similar to `UserFunctions`, except that they run on a remote host computer connected on your network.

The Differences Between UserObjects and UserFunctions

In previous chapters, you have already created and used `UserObjects`. The reason that VEE provides both `UserObject` and `UserFunction` is because the two have different characteristics and can therefore be used for different purposes. Here are the differences between a `UserObject` and a `UserFunction`:

A `UserObject` (located in `Device` ⇒ `UserObject`) is an object you define that may be used just like any other object in VEE. You program a `UserObject` like a subprogram but it graphically remains on the screen. If you want to use it elsewhere in a program, you must clone it and maintain all copies. Note that if you clone a `UserObject` many times, it makes the program larger and slower to load. If you add a feature to one `UserObject`, you would need to add the same feature to all the other `UserObjects` if you want them to remain identical.

With a `UserFunction` (located in `Device` ⇒ `UserFunction`), there is just one copy of the subroutine in memory, and it is only displayed graphically in the workspace in its own window if you want it to be.

Otherwise, it is stored to be called from the `Call` object or any other expression field. Changes to a `UserFunction` will be inherited by all instances in the program that calls that `UserFunction`. You can also create libraries of `UserFunctions` for more code re-use.

Lab 8-1: UserFunction Operations

This exercise describes how to create a `UserFunction` named `ArrayStats`, which will accept an array, calculate its maximum value, minimum value, mean, and standard deviation, and put the results on its output pins.

Creating a UserFunction

1. Select `Device` ⇒ `Formula`, delete its default input pin, and change its default expression to `ramp(1024,1,1024)`.

   This will create a 1024 element array with values from 1 to 1024.

2. Select `Device` ⇒ `UserFunction`. Rename it `ArrayStats`.

   a. Add one data input terminal for the array

   b. Add four data output terminals for the results.

   c. Rename the output terminals: `Max`, `Min`, `Mean`, and `Sdev`. Select `max`, `min`, `mean`, and `sdev` from the `Probability & Statistics` category in the `Function & Object Browser` box.

   d. Place them in `ArrayStats`, and connect their data inputs to `A` and their data outputs to the appropriate output terminals. Make the `ArrayStats` window smaller to see both of the Main and `ArrayStats` windows. See Figure 8-1.

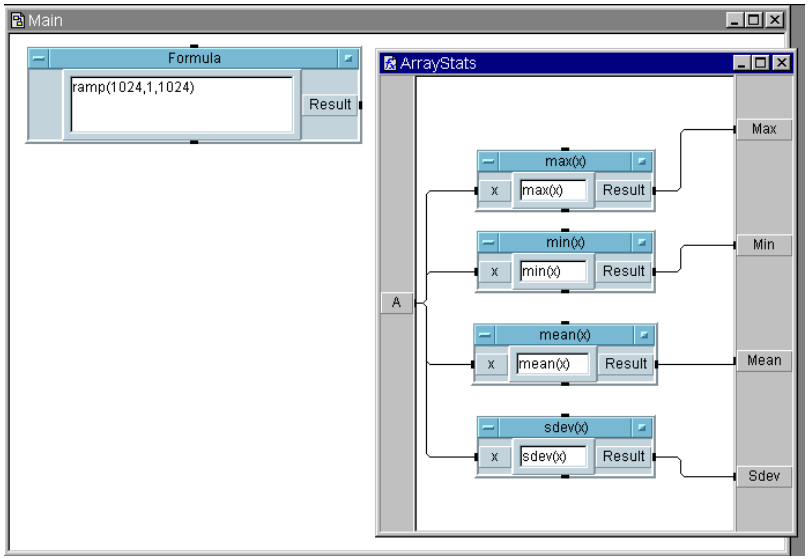


Figure 8-1. The Main and ArrayStats Windows

3. Iconize `ArrayStats`. It appears as an icon at the bottom of the workspace.

4. Click `Device` ⇒ `Call`, open the object menu, and click `Select Function` as shown in Figure 8-2. Then click `OK`. Notice that VEE renames the object automatically and adds the correct pins.

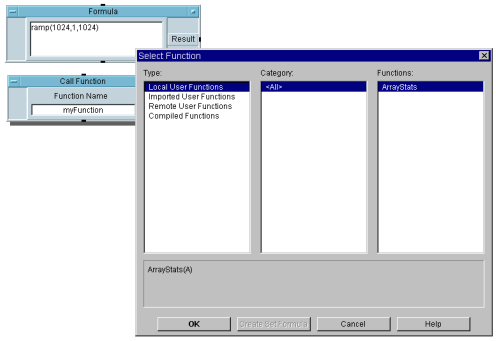


Figure 8-2. Configuring the Pins for Call myFunction

5. Connect the output of `Formula` to the `Call ArrayStats` input. Select `Display` ⇒ `AlphaNumeric`, clone it three times, and connect the displays to the `Call ArrayStats` output pins. Rename the displays.

6. Run the program. It should look like Figure 8-3. Save the program as `array_stats.vee`.

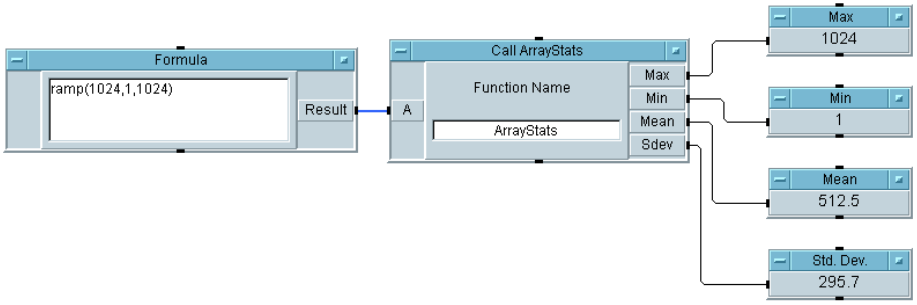


Figure 8-3. Calling the User Function ArrayStats

To use `ArrayStats` elsewhere in the program, you would click on `Device` ⇒ `Call`, open the `Select Function` box from the object menu, and choose `ArrayStats`. VEE would automatically rename the object `Call ArrayStats`, and add the necessary input and output terminals.

Shortcut: From the `UserFunction` object menu, select `Generate` ⇒ `Call` to bring up the `Call ArrayStats` object. (Make sure that the `UserFunction` is not expanded to the whole workspace when doing this.)

Editing a UserFunction

In this exercise, edit `ArrayStats` to deliver a record with four fields giving the array statistics.

1. Delete the four `AlphaNumeric` displays.

2. Select `Edit` ⇒ `Edit UserFunction...` and select `ArrayStats` from the `Edit UserFunction` list box. All of the `UserFunctions` in the program are displayed.

3. Open the `ArrayStats` object menu, click on `size`, and enlarge the editing window. If you need to resize objects, click and drag any corner of the object.

4. Delete the four lines going to the output terminals. (Press **Ctrl-Shift** and click on the line you want to delete.)

5. Select `Data` ⇒ `Build Data` ⇒ `Record` and place it to the right side of the `ArrayStats` window.

   a. Add two data input terminals.

   b. Label the four terminals after the statistical functions: `max`, `min`, `mean`, and `sdev`.

   c. Connect the four `Formula` object outputs to the inputs on `Build Record`.

   d. Rename the `Max` output terminal `X` by double-clicking `Max`, typing the new name, and clicking `OK`.

   e. Delete the other `ArrayStats` data output terminals.

f. Connect the `Build Record` output to the `X` output terminal on the `User Function` editing window. The program should look like Figure 8-4. Then click the iconize button on the window.

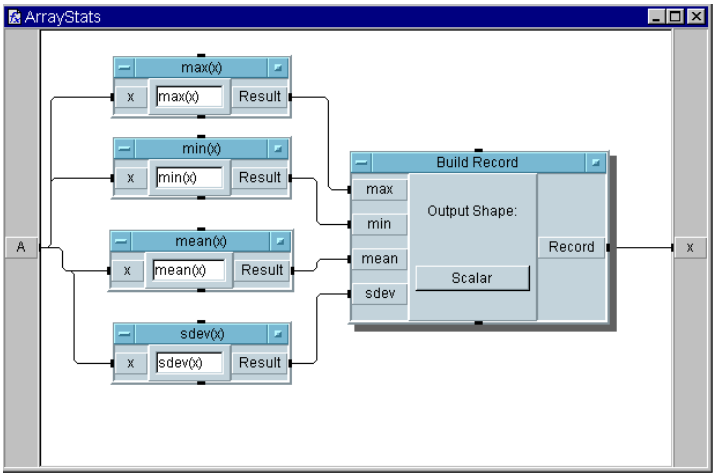


Figure 8-4. Editing the UserFunction ArrayStats

6. Open the `Call ArrayStats` object menu and click `Configure Pinout`. This will adjust the number of pins to match the recent edits.

Note

In order to update the number of pins, you must open the object and click `Configure Pinout` whenever you change the number of inputs or outputs in a `UserFunction`. Or you can manually update the `Call` object's input and output pins, but using `Configure Pinout` is much simpler. You can use `Find` to find all the `Call` objects and expressions that call a `UserFunction`. For more information, refer to "Finding Functions in Large Programs" on page 321.

Now display a record using the `Record Constant` object. Use the `Default Value` control input to accept a record from `ArrayStats`. VEE automatically configures the `Record Constant` to hold the incoming record.

7. Select `Data` ⇒ `Constant` ⇒ `Record` and place it to the right of the `Call Function` object.

   a. Open the `Record` object menu and click `Add Terminal` ⇒ `Control Input...`. Select `Default Value` from the list box presented. You can open the `Properties` menu to `Show Terminals`, if you wish.

   b. Now connect the `Call Function` data output to the control input pin on the `Record` object. Notice that control lines are indicated by dashed lines to differentiate them from data lines.

8. Run the program. It should look like Figure 8-5.

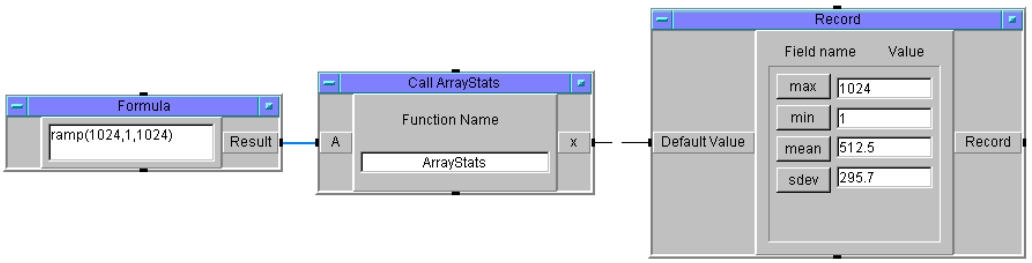


Figure 8-5. After Editing ArrayStats Output to a Record

Calling a UserFunction from an Expression

In this exercise, you will learn how to call `ArrayStats` from an expression in the `Formula` object.

1. Select `Device` ⇒ `Formula` and replace the default formula with `ArrayStats(A)`. Click `Replace` in the `Call ArrayStats` object menu.

The Status Bar at the bottom of the VEE screen prompts you to select the replacement object. Click on the `Formula` object that calls the `ArrayStats` function. VEE automatically replaces the `Call ArrayStats` object with the new `Formula` object and retains the wiring of the data lines.

The `Formula` object takes the input at terminal `A` and sends it to the `UserFunction ArrayStats`. `ArrayStats` delivers the record of statistics to its terminal `X`. The first output value from the `UserFunction (X)` is returned to the `Formula` object and delivered to its `Result` output.

2. Run the program. It should look like Figure 8-6.

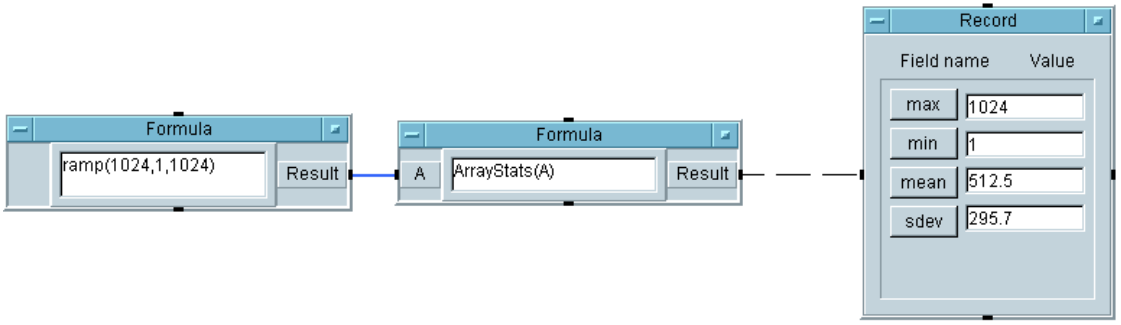


Figure 8-6. Calling the ArrayStats User Function

Notice that the functionality of `ArrayStats` in the `Formula` object is exactly the same as it was in the `Call ArrayStats` object. This example uses a `Formula` object, but you could call `ArrayStats` from *any* input field that accepts expressions, such as the `To File` object.

Note

When you call a `UserFunction` from an expression, the `UserFunction` will only deliver a single output (the uppermost data output pin). If you need all of the outputs, or they cannot be put into a `Record`, then use the `Call Function` object.

Note

When you call a `UserFunction` from an expression, input terminals are used as function parameters to pass to the function. If no data is passed to the function, you must still include empty parentheses after the function name. Otherwise, VEE assumes you are referring to a `Global` variable or input terminal. For example, if the `UserFunction` called `MyFunction` has no input parameters, you must write `MyFunction()` in an expression. The `Call` object does not require the parentheses, because VEE knows you are referring to a function.

Generating a Call to a UserFunction

To generate and place a call object in the Main program from a `UserFunction`, use the `UserFunction` object menu `Generate` menu. The `Generate` menu contains most of the common objects that call a `UserFunction`. When you select a calling object, it can be placed in the calling window, such as the Main program, properly configured with the correct name and pins.

In this exercise, you will learn how to generate the `ArrayStats` object in the Main program from the `ArrayStats UserFunction`.

1. In the same example used in Figure 8-6, double-click the `Formula` object `ArrayStats` to delete the object. (You could also select the object menu and select Cut.)

2. In the `UserFunction ArrayStats`, select the object menu and select `Generate` ⇒ `Formula Call`. Figure 8-7 shows the `Generate` menu in a `UserFunction` object menu.

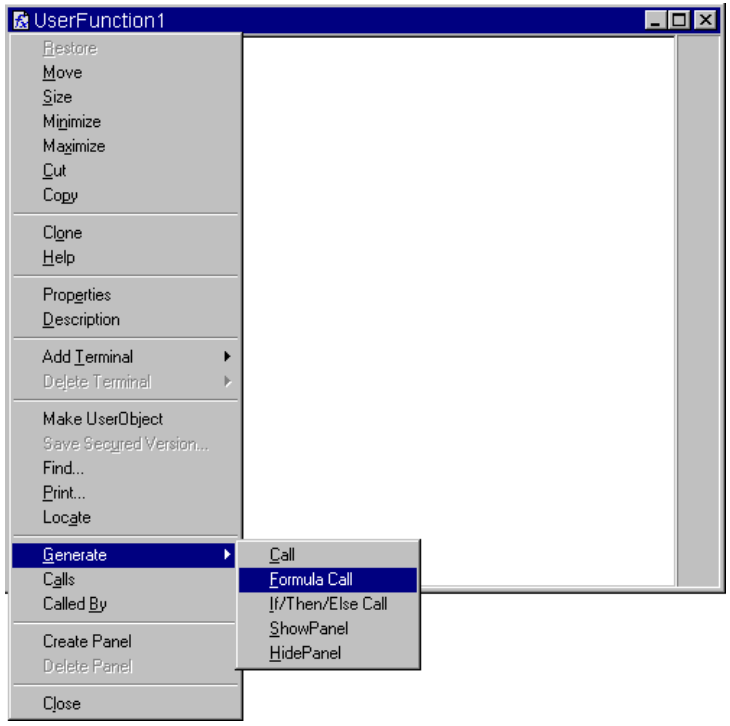


Figure 8-7. The Generate Menu in a UserFunction

3. Place the object in Main. Notice that VEE automatically names the new object `ArrayStats(A)` and includes the expression `ArrayStats(A)` to call the `UserFunction ArrayStats`.

4. Connect the output from the `Formula` object to `ArrayStats(A)`, and connect the output from `ArrayStats(A)` to `Record`.

5. Run the program. It should look like Figure 8-8.

Open a `UserFunction` object menu and select the `Generate` menu to review the other objects that can be placed into a program to call a `UserFunction`. They include `Call`, `Formula Call` (used in this example), `If/Then/Else Call`, `ShowPanel`, and `HidePanel` objects.

Figure 8-8. Generating a Call Object ArrayStats(A) from a UserFunction

UserFunctions and the Program Explorer

`UserFunctions` and `UserObjects` make VEE programs more modular and easy to understand. The Program Explorer is a valuable tool to navigating through complex programs. For example, the hierarchy of the program in Figure 8-9 is shown in the Program Explorer. To display the Program Explorer, click `View` ⇒ `Program Explorer` or click the **Program Explorer** icon on the toolbar.

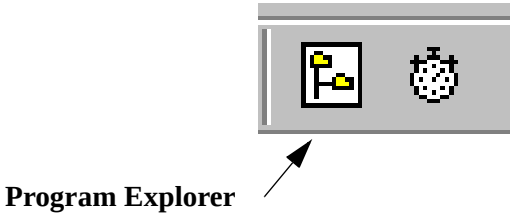


Figure 8-9. Program Explorer Icon on the Toolbar

Figure 8-10 shows the Program Explorer being used.

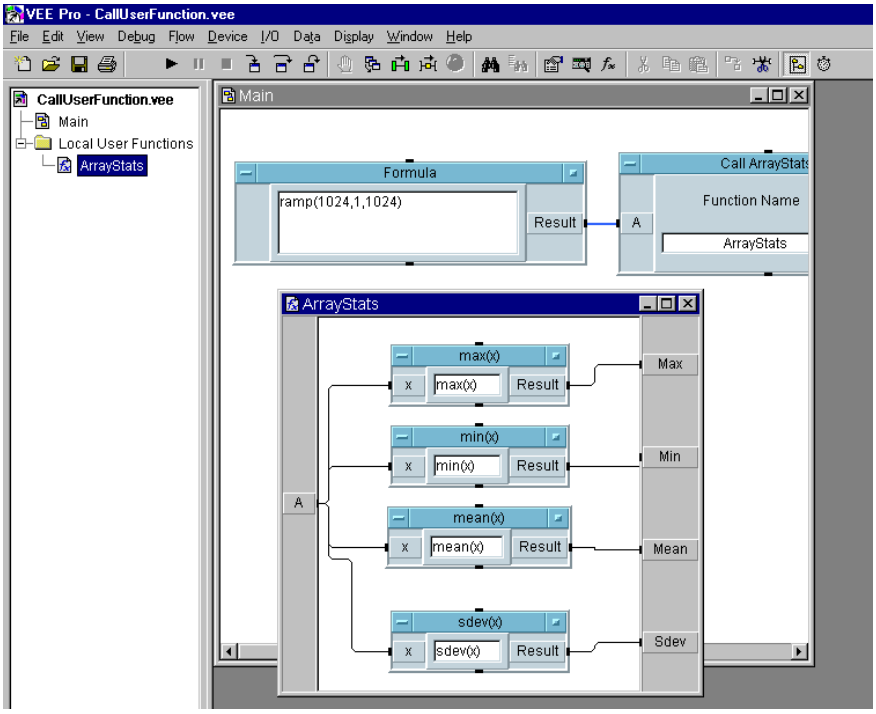


Figure 8-10. Using the Program Explorer with UserFunctions

Using Libraries With Agilent VEE UserFunctions

To leverage existing VEE test programs, you can re-use `UserFunctions`. When you save a program, the `UserFunctions` are automatically saved as well. A `UserFunction` can hold a VEE program or a library of logically related `UserFunctions`.

There are two ways to put existing `UserFunctions` into a new program:

1. Put a copy of the original `UserFunctions` into the current program, using the `File ⇒ Merge Library...` command (where you now maintain the separate copy of each `UserFunction`). These merged `UserFunctions` can be edited, so use the `File ⇒ Merge Library...` command when you plan to modify the `UserFunctions`.

-OR-

2. Access the original `UserFunctions` using the `Device ⇒ Import Library` object, which accesses the original functions in another file without making a copy. These `UserFunctions` are imported at run time. This spreads out the load times, conserves disk space, and saves memory. Imported `UserFunctions` can be viewed (such as for debugging purposes) but cannot be edited. Instead, you can edit their original files. You can also delete imported `UserFunctions` programmatically, using the `Device ⇒ Delete Library` object.

Therefore, *merge* `UserFunctions` when you need a new copy of the function to modify or you need a standalone program, and *import* `UserFunctions` when you want a single source for the function or you want to save space.

Lab 8-2: Creating and Merging a Library of UserFunctions

In this exercise, you will create a report generation program that includes a VEE library of `UserFunctions`. Then you will create a new program that merges the library of `UserFunctions`.

Creating a Library of UserFunctions

1. Create the top level program (without programming the details of the `UserFunctions`).

   a. Create four `UserFunctions`: `BuildRecAry` with one output pin, `ReportHeader`, `ReportBody` with one input pin, and `ReportDisplay`. Iconize all the `UserFunctions`.

   b. In `Main`, create four `Device` ⇒ `Call` objects configured and connected as shown in Figure 8-11. Save the program as `Report.vee`.

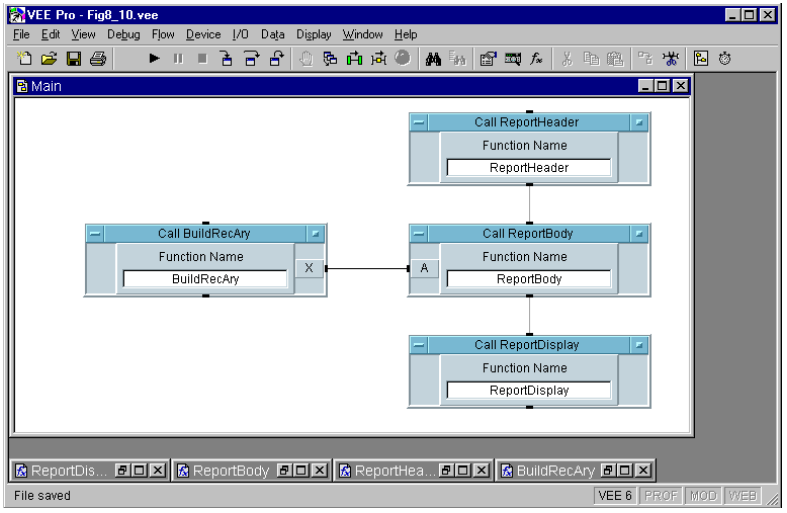


Figure 8-11. Report.vee from the Top Level

Note

The `Call` object does not require parentheses when referring to a `UserFunction`. If you were calling the function from a `Formula` object, you would need to include parentheses whether or not the function used parameters.

The four `UserFunctions` are a library. You can see them listed by clicking `Edit` ⇒ `Edit UserFunction...`. Click `Cancel` to close the list box.

2. Program the four `UserFunctions` as shown in the following figures.

   Figure 8-12 shows the `BuildRecAry UserFunction`. It includes a `Formula` object with the triadic expression testing if `A<=5`. If the expression evaluates to `TRUE`, then the object outputs `"PASS"`. Otherwise, the object outputs `"FAIL"`. (Note that the parentheses are required.)

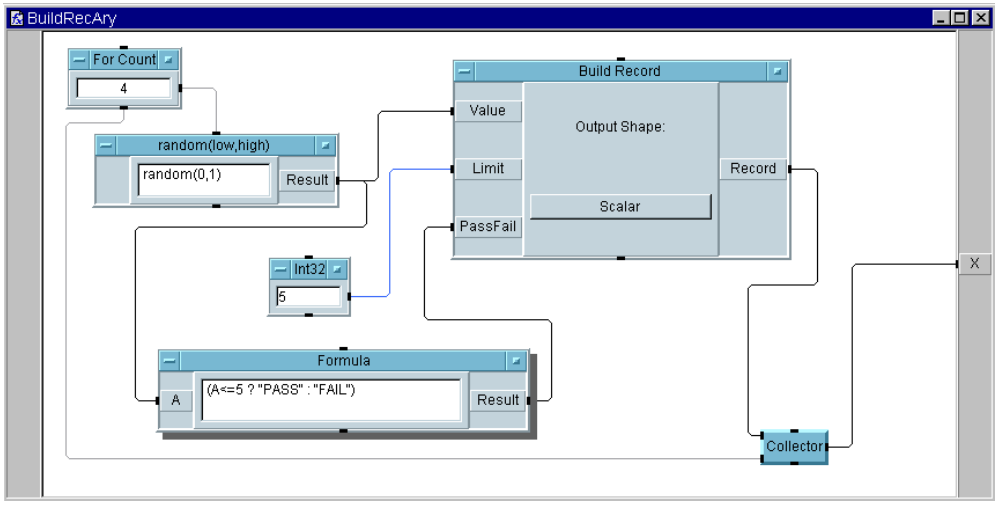


Figure 8-12. The BuildRecAry UserFunction

Figure 8-13 shows the `ReportHeader UserFunction`.

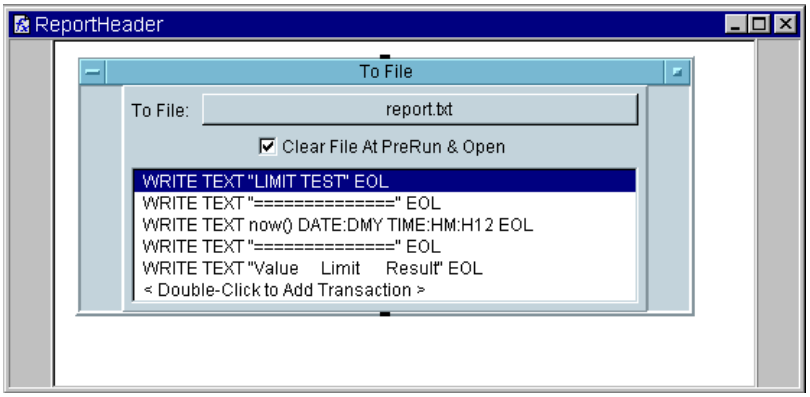


Figure 8-13. The ReportHeader UserFunction

Figure 8-14 shows the `ReportBody UserFunction`. Note the array of records `A[B]`. As the `Value` of `B` changes from `0` to `1` to `2`, you can access the particular field in that `Record`, including `Value`, `Limit`, and `PassFail`, using the <*record*>.<*field*> notation. Note that the `For Count` outputs start with zero. Note also that the first transaction has `EOL off`.

The vertical bar in quotes, `"|"`, represents a constant string character for a vertical bar. `FW:5` stands for a string field width of `5`. `RJ` stands for right justified.

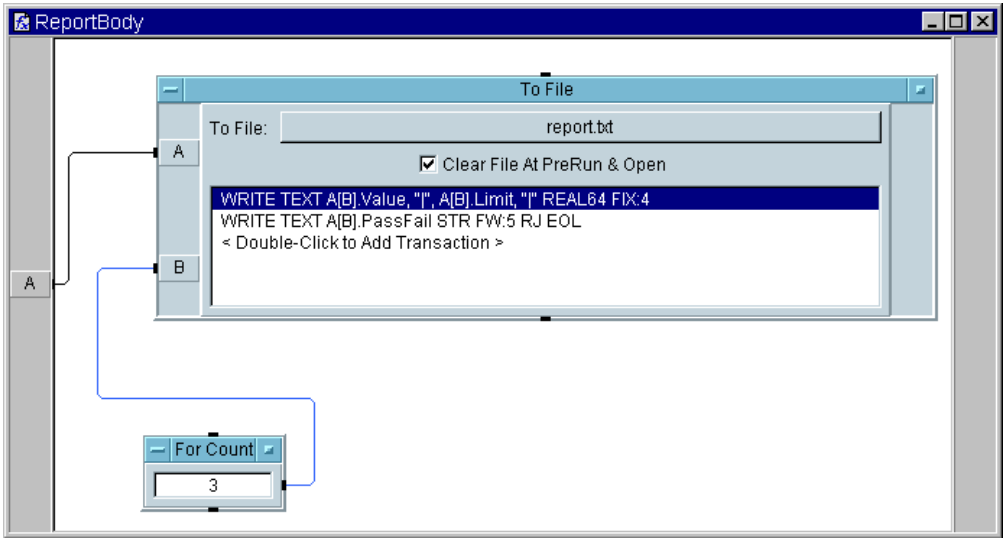


Figure 8-14. The ReportBody UserFunction

Figure 8-15 shows the `ReportDisplay UserFunction` in detail view. Note that it reads a string array to the end of the file, specified by the asterisk sign (`*`) after the `STR ARRAY` format.

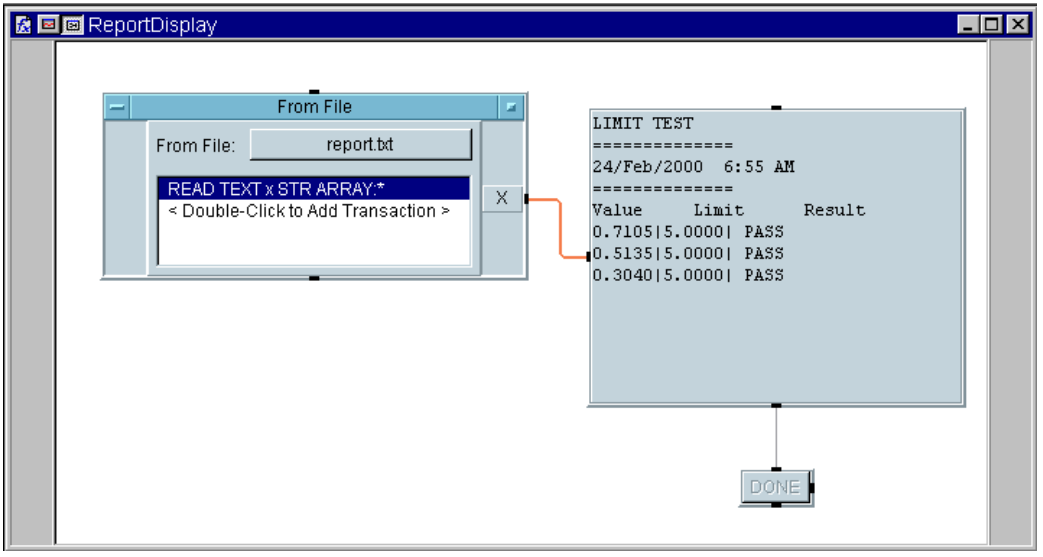


Figure 8-15. The ReportDisplay Detail View

Figure 8-16 shows the panel view of the `ReportDisplay UserFunction` with `Show Panel on Execute` selected in the `Properties` box. In the `Properties` box, the `Pop-up Panel Title` has also been changed to `ReportDisplay`. To create the panel, select the `Confirm (OK)` and `Logging AlphaNumeric` objects, and click `Edit` ⇒ `Add to Panel`. Note that the `Logging AlphaNumeric` display has `Show Title Bar` deselected. Note also that `Confirm (OK)` button has been renamed `DONE`. The `Confirm (OK)` button is included to keep the display on the screen until the user is done viewing it.

3. Run the program and the `ReportDisplay` panel should pop up and display the values as shown in Figure 8-16. Click on `DONE` to complete execution. Then save the program as `Report.vee`.

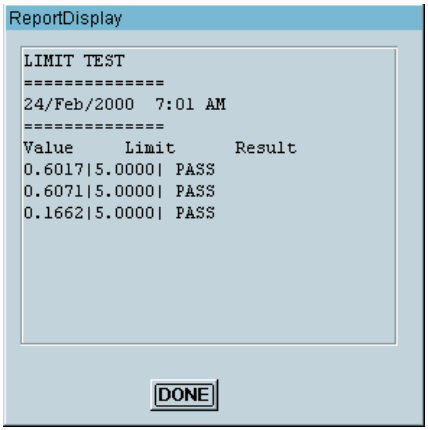


Figure 8-16. The ReportDisplay Panel View

Creating Another Program and Merging in the Library

In this exercise, you will create a new program and merge the library into it. This exercise builds a library of functions for generating reports. The new program contains a `Note Pad` object explaining each function in the library. It will be named `RepGen`.

You could re-use `RepGen` by creating new report generation `User Functions`, merging them with the program, and updating the `Note Pad` object to keep track of them. Then you could use the `Merge Library...` command to leverage all the functions from `RepGen`.

1. Select `File` ⇒ `New`.

2. Select `File` ⇒ `Merge Library...`. Select `Report.vee` from the `Merge Library` list box. (If you are in a different directory, type the whole file path.)

   Select `Edit` ⇒ `Edit UserFunction` (or look at the Program Explorer) to make sure the library from `Report.vee` transferred to the new program. When you use the `Merge Library...` command, you can edit merged functions just like local functions.

3. Select `Display ⇒ Note Pad` and type the `UserFunction` descriptions similar to the ones shown in Figure 8-17. Then save the program as `RepGen.vee`.

Note

You can save a "program" of `UserFunctions` for the purpose of creating a library, even though there is no actual VEE program calling the functions.

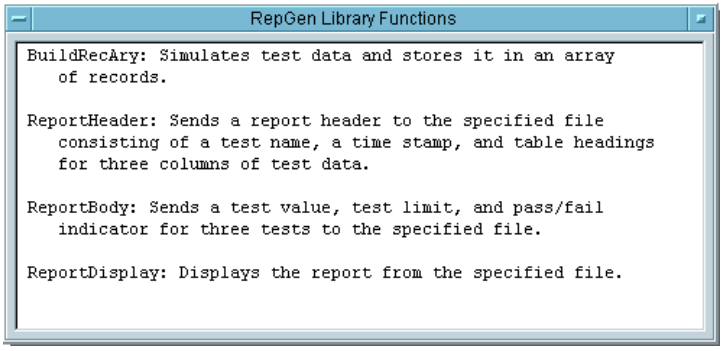


Figure 8-17. The RepGen.vee Library of UserFunctions

Lab 8-3: Importing and Deleting Libraries

Once you have created a library of `UserFunctions`, you may not want to merge them into every program. You might like to bring in the library at run time, use some of the functions, and then delete the library to conserve memory. The `Import Library` and `Delete Library` objects are designed for this situation.

In this exercise, you will import functions from the `RepGen` program. Then you will call the `BuildRecAry` function to simulate some test data, display it, and finally delete the library to free up memory and swap space.

1. Select `File` ⇒ `New`.

2. Select `Device` ⇒ `Import Library` and place it in `Main`. Set the fields in the `Import Library` object as follows:

Library Type The menu in the `Library Type` field allows you to select a `UserFunction`, a Compiled Function, or a Remote Function. In this case you want a UserFunction library, so leave the default.

Library Name The `Library Name` shows `myLib` as a default. This name is used as a "handle" by the VEE program to distinguish between different libraries being imported. The `Delete Library` object uses this name to identify the library to be deleted. You can use the default name.

File Name The `File Name` field shows a dialog box for the user program directory by default on a PC. (HP-UX systems access the directory you were in when you started VEE.) Specify the file that holds the library of functions.

Click the default name `myFile` to get the list box. Select `RepGen.vee` (from "Creating and Merging a Library of UserFunctions" on page 310). This file will be in the directory you specified for your programs during installation.

3. Open the object menu and select `Load Lib` to import the library immediately instead of waiting until runtime. This command is very useful in the development stage. (In the next step, you will notice that when choosing `Select Function` in the `Call` object, the functions are designated with the library handle first, such as `myLib.BuildRecAry`.)

   To display the library of imported functions, use the `Program Explorer`.

Note

Because you have imported the library, you can only view the `UserFunction` and set breakpoints for it. You cannot edit the `UserFunction`. To add a `UserFunction` to a program that can be edited, use the `Merge Library...` command.

4. Select `Device` ⇒ `Call` and place it below the `Import Library` object. Connect the output sequence pin from `Import Library` to input sequence pin on the `Call` object.

5. Open the `Call Function` object menu and click `Select Function` to show a list of the functions imported with the `Load Lib` command. Select `myLib.BuildRecAry` as shown in Figure 8-18.

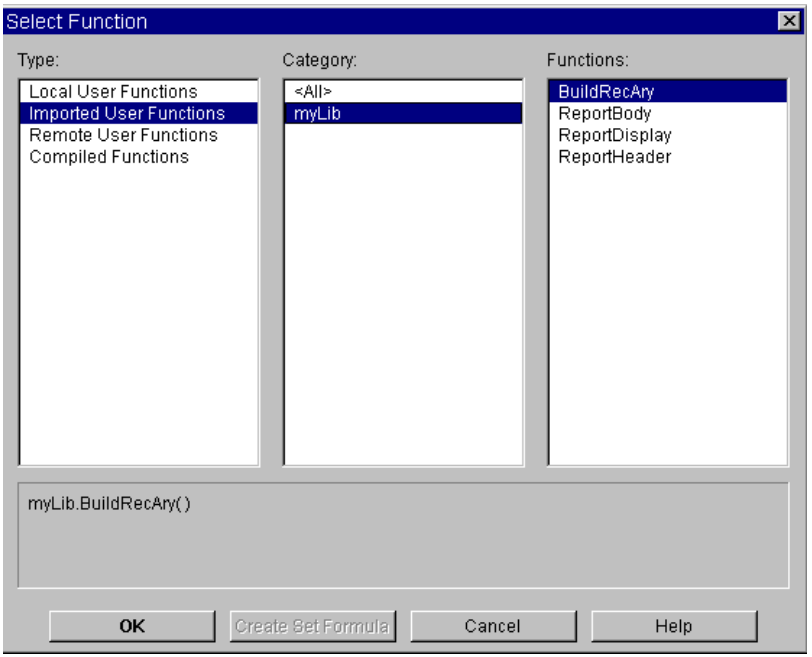


Figure 8-18. Selecting a Function from an Imported Library

VEE automatically inserts the function in the `Function Name` field and adds the required output terminal. You could also have entered `myLib.BuildRecAry` in the `Function Name` field to accomplish the same results. Use `Select Function` when you need to list the names of the functions in the library.

6. Select an `AlphaNumeric` display, enlarge it, and connect it to the `Call` data output.

7. Select `Device` ⇒ `Delete Library` and place it below the `Call` object. Connect the sequence pins, so the library is deleted after the `BuildRecAry` function has been called. You can leave the default `Library Name`, since this is the same name you used with the `Import Library` object.

8. Run the program. It should look like Figure 8-19. Save the program as `libraries.vee`.

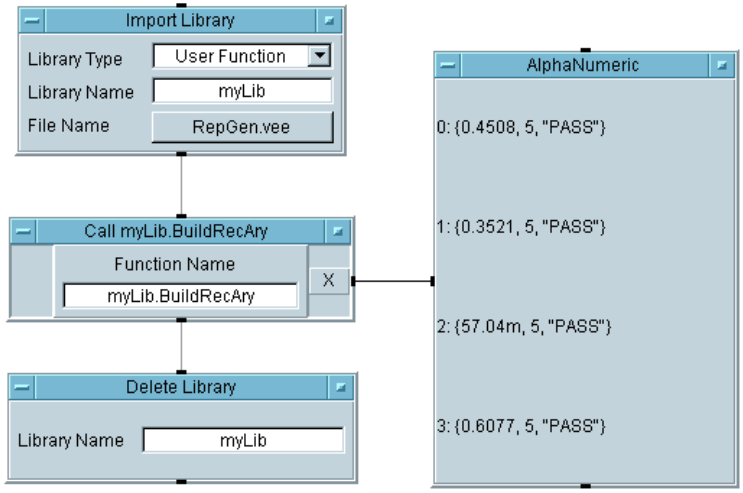


Figure 8-19. Calling a Function from a Library

Note the following about names of merged and imported functions:

- If a merged function has the same name as a local function, VEE displays an error.
- If an imported function has the same name as a local function, it is allowed, but the local functions are called if only the function name is used. You can explicitly call the imported function with the `MyLib_func()` syntax as in the `Call` object in Figure 8-19.
- If two imported libraries have the same function names, the results will be indeterminate. Notice that the `Call` object uses the `Library` name `myLib.BuildRecAry`, so there is no confusion. Even if there were another local function or other imported function with the same name, this specifies the name and location of `BuildRecAry`.

Finding Functions in Large Programs

VEE provides a `Find` feature located in the `Edit` menu to help you locate objects and text in a large program. For example, open the `Solitaire.vee` program in the `Examples/Games` directory. Go to the `detail` view and click `Edit` ⇒ `Find...` to display the dialog box shown in Figure 8-20.

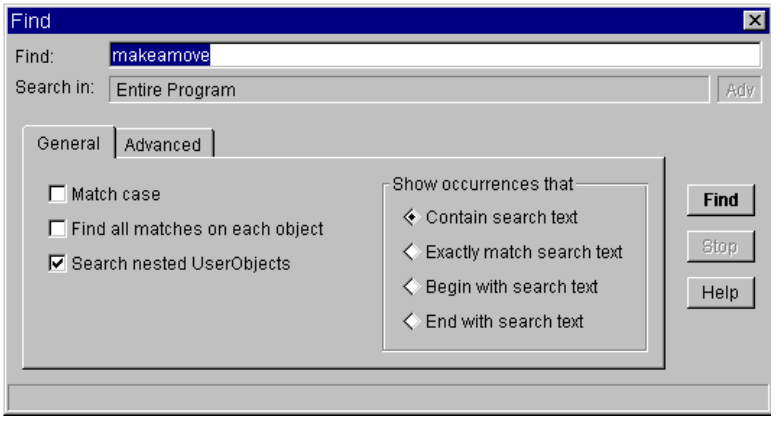


Figure 8-20. The Find Dialog Box

Type `makeamove` (a `UserFunction` in this program), as shown in the figure, and click `Find`. VEE automatically locates the `UserFunction` named `makeamove` and shows the part of the program from which it was called, as shown in Figure 8-21.

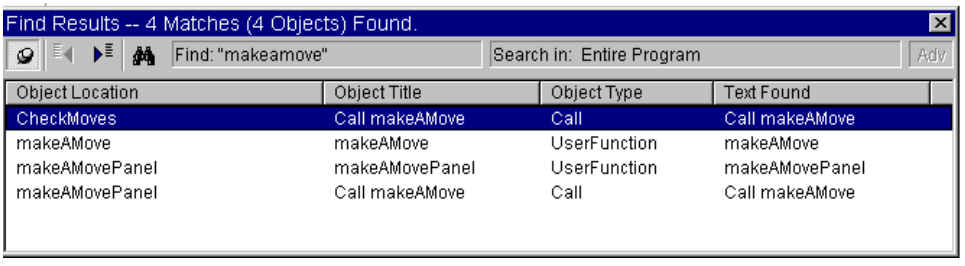


Figure 8-21. The Find Results Dialog Box

You can use `Find` to locate any object or text such as variables, titles, settings on objects, etc. Double-click on any line in the `Find Results` box to locate an object.

Note

`Find` can also be used by placing the mouse pointer over objects in the `Program Explorer` and clicking the right button. This will limit the scope of the search to the particular `UserFunction` or `UserObject`.

Merging Agilent VEE Programs

The easiest way to leverage existing programs is to merge a past program with the current test. You can re-use programs by merging them and then editing them to suit your current needs.

The `File ⇒ Merge...` command adds the contents of a program or set of saved objects into the work area while keeping the existing contents of the work area. By default, `File ⇒ Merge...` displays a directory of programs that are shipped with VEE. They include commonly needed programs such as a bar chart display and a data entry keypad for user input (such as ID numbers).

Lab 8-4: Merging a Bar Chart Display Program

In this exercise, you will merge an existing program with a new program. Although the example uses a program from the VEE `Lib` directory, you could use any program. You will create an array with five values from 1 to 5 using the `ramp()` function. Instead of displaying the array with one of the internal VEE displays, you will merge the `BarChart` program with the program you are creating.

1. Select ⇒ `Formula` and place it in the left work area.

2. Delete the data input terminal.

3. Change the default formula to `ramp(5,1,5)`.

   The first parameter is the number of elements desired in the ramp array. The second parameter is the starting number, and the third is the last number. For more information on this function, select `Help` in the `Formula` object menu now that it has the `ramp` call in it. (Or try `Help ⇒ Contents & Index`, then use the `Search` feature in the `Index` folder.)

4. Click on `File ⇒ Merge...` to get the `Merge File` list box. Select `BarChart.vee` and place it to the right of the `Formula` object. Connect the two objects.

5. Run the program. It should look like Figure 8-22.

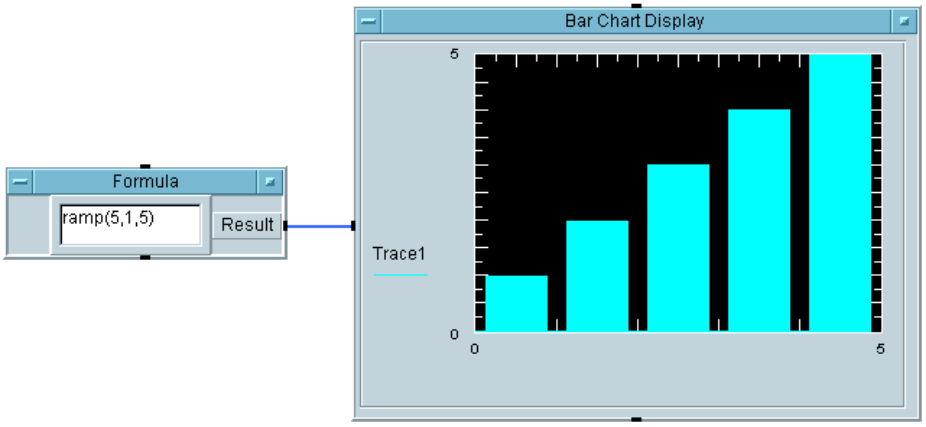


Figure 8-22. Merging the BarChart Program

Notice that the Bar Chart Display takes a one-dimensional array and displays the values as vertical bars. It uses the number of bars necessary to display the values in the array. To see how the program is created, open the detail view of the display. You can look at examples in the library directory for more ideas about programs.

Note

The `File ⇒ Merge` command is used to merge in `UserObjects` and objects. The `File ⇒ Merge Library` command is used to merge `UserFunctions`.

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary, before going on to the next chapter.

- Define a `UserFunction` and compare it to a Compiled Function and a Remote Function.
- Create, call, and edit a `UserFunction`.
- Create, merge, import, and delete `UserFunction` libraries.
- Use the `Find` feature in one of the game programs.
- Merge an entire VEE program with the current program.

9

Test Sequencing

Test Sequencing

In this chapter you will learn about:

- The `Sequencer` object
- Configuring a test for the `Sequencer`
- Creating a test execution order based on run time results
- Accessing data logged by the `Sequencer`
- Ways to pass data to or from `Sequencer` tests
- Performing analysis on logged data from the `Sequencer`
- Storing `Sequencer` test data

Average time to complete: 2 hours

Overview

In this chapter, you will learn the fundamentals of using the `Sequencer` object. The `Sequencer` object can execute a series of sequence transactions, each of which may call a `UserFunction`, Compiled Function, or Remote Function. Typically, the `Sequencer` is used to perform a series of tests.

Some of the benefits of using the `Sequencer` include:

- Easy development of a test plan
- Wide array of branching capabilities between tests
- Major component for building a customized test executive
- Ability to call tests in VEE and other languages
- Automatic logging of test results

Note The `Sequencer` is one of VEE's most powerful features. For more information about the `Sequencer`, refer to online `Help`.

The first lab shows you how to configure a test for the `Sequencer` object, to add or insert or delete a test in the test execution flow, and to access the test data that has been logged by the `Sequencer`. The lab simulates test results with the `random()` function.

In the second lab, you will learn how to structure data passed to tests using global variables, to call `UserFunctions` from the `Sequencer`, and to log `Sequencer` data to files. Finally, you will analyze parts of the data.

Note To use a status panel that updates through a sequence of tests, see "Creating a Status Panel" on page 398.

Note In addition to the lab exercises in this chapter, you can get more practice in using the `Sequencer` by completing the exercise in "Test Sequencing" on page 514 in Appendix A, "Additional Lab Exercises."

Using the Sequencer Object

The `Sequencer` object executes tests in a specified order based on runtime results. Each test may be a VEE `UserFunction`, a Compiled Function, a Remote Function, or any other expression which returns a single result. That result is compared to a test specification to determine whether or not it passes. The `Sequencer` then uses a pass or fail indicator to determine the next test it should perform.

There are six different options for branching to the next test. These options include executing the next test, repeating the same test, or jumping back to an earlier test. Lab 9-1 explains branching options in detail. The `Sequencer` can even ask for user input to decide what course of action to take.

After the specified tests have been executed, the `Sequencer` automatically logs the test data to an output terminal. From this point the data can be analyzed and displayed, or stored to a file for future investigation.

Creating a Test Execution Order

In this lab you will simulate test results using the `random()` function, establish a test execution order, learn how to modify that order, and retrieve specific data from the logged results.

Lab 9-1: Configuring a Test

Note

The example explains how to implement the `random()` function with a certain range of test results expected, but you can use the same principles to configure other tests.

1. Select `Device` ⇒ `Sequencer` and place it in the upper-left work area.
2. Select `Display` ⇒ `AlphaNumeric` display, place it below the `Sequencer`, increase its width, and connect the `Sequencer Log` output terminal to the `Alphanumeric` data input.
3. Double-click the `Sequencer` transaction bar to get the `Sequence Transaction` dialog box. Set the fields as follows.

Note

As you edit the fields, remember to click on new fields to modify them or use the **Tab** key to move forward to different fields. Use the **Shift-Tab** keys to move the cursor backward. Press the **Enter** key only when you are done editing the dialog box.

TEST:	The default name is `test1`, which you can use. This is just the label for the test in the `Sequencer`. It is not the test function itself.
SPEC NOMINAL:	Represents the expected test value. The default is `.5`. Change this to `.25`, and then alter the upper `RANGE` field (on the far right) from `1` to `.5`.

FUNCTION: The default entry `testFunc(a)` holds the actual function that performs the test. In this case, replace the default field with the `random()` function. `Random()` will return a `Real64` value between `0` and `1` simulating a test result. This result will be compared to the test specification.

The `random(low,high)` object is located in the `Probability & Statistics` category in the `Function & Object Browser` box. Remember that you can call this math function from any expression field without actually using a `Formula` object. If you do not provide the parameters low and high, as shown in this example, the function will use the default parameters `0` and `1`.

You may leave the other defaults. This configuration will lead to a `PASS` result in approximately half of the runs. The dialog box should look like Figure 9-1.

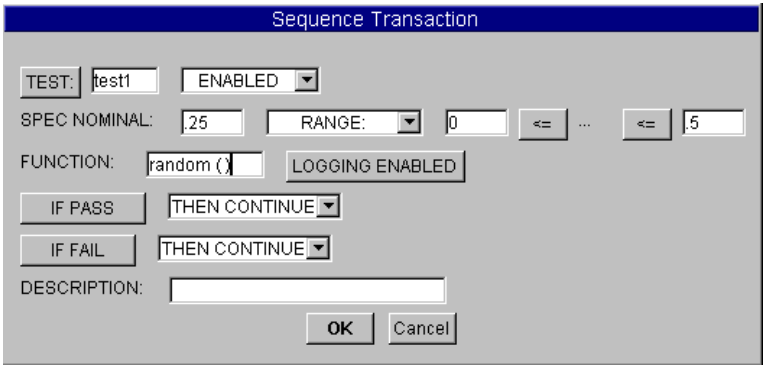


Figure 9-1. The Sequence Transaction Dialog Box

Click `OK` to close the dialog box. You will see the transaction
`test1 0 <= (.25) <= .5`
on the first transaction bar. This means that `test1` will pass if the returned value is in the range from `0` to `.5` with the end points included. The expected result is about `.25`.

4. Run the program. It should display the name of the test, the test result, and the pass-fail indicator (`1` for `PASS`, `0` for `FAIL`) in the display, as shown in Figure 9-2.

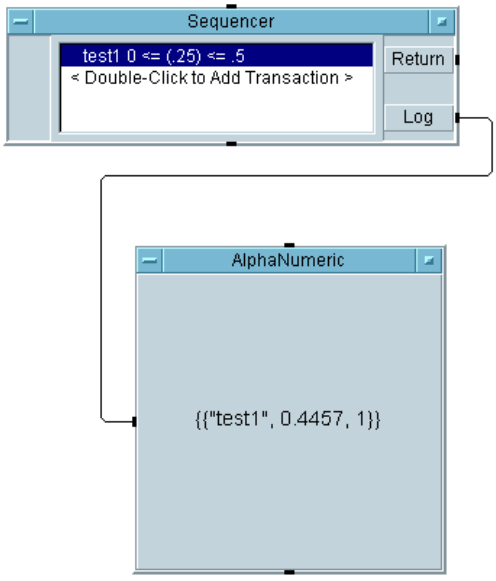


Figure 9-2. Configuring a Test

Before proceeding, study Table 9-1 to understand the various choices in the `Sequence Transaction` dialog box. Open the dialog box again by double-clicking on the transaction bar. Open the various menus and make read about the different options.

Table 9-1. Sequence Transaction Dialog Box

Sequence Transaction Field	Explanation
TEST:	Unique name used to reference the test in the `Sequencer`. The default names start with `test1` and increment with each test. Choosing `TEST:` means that a test result will be compared to specifications and branching will occur to the next test based on the configuration. The `TEST:` button toggles to `EXEC:`. If you toggle `TEST:` to `EXEC:`, the test will execute without a comparison between a test result and specifications. For example, you might choose `EXEC:` when the `UserFunction` is setting up global variables. Selecting `EXEC:` will also disable logging for the test.
ENABLED	Determines when to execute a test. This menu displays four menu choices: ■ `ENABLED` executes the test under all conditions. ■ `ENABLED IF:` executes the test if the stated expression evaluates to `TRUE`. For example, the test might be enabled if the input pin `A` holds the value `1 (A == 1)`. You can use `ENABLED IF:` for audit test control. You might want a particular test to execute every ten runs, for instance. ■ `DISABLED` is the opposite of `ENABLED`. ■ `DISABLED IF:` is the opposite of `ENABLED IF:`.
SPEC NOMINAL:	Expected value from the test.

Table 9-1. Sequence Transaction Dialog Box (Continued)

Sequence Transaction Field	Explanation
RANGE:	Specifies the range of test values. This menu displays four menu choices: ■ `RANGE` signifies the range of test values that signify a `PASS` condition. You may also choose from the usual comparisons: `>`, `>=`, `<`, `<=`, `==`, `!=` . ■ `LIMIT` uses just one value for a comparison of test data. ■ `TOLERANCE` states the passing range of values by adding or subtracting the specified tolerance to the `SPEC NOMINAL` value. ■ `%TOLERANCE` states the passing range of values by adding and subtracting a percent tolerance of the `SPEC NOMINAL` value to the nominal specification.
FUNCTION:	Specifies the test to run. You can call `UserFunctions`, Compiled Functions, Remote Functions, or you can write in an expression to be evaluated. The result of the function you call (or expression you evaluate) is tested against the specifications. If a `UserFunction` returns more than one value, VEE assumes the top output pin holds the result to be tested. Functions can also be combined and nested: `(random(0,myfunc()+3,100)*2)`, for example.

Table 9-1. Sequence Transaction Dialog Box (Continued)

Sequence Transaction Field	Explanation
LOGGING ENABLED	Logs test data. To specify logging options, open the `Sequencer` object menu, choose `Properties`, click on the `Logging` folder, and choose from the list. By default, `Name`, `Result`, and `Pass` are checked. There is also a field to choose between `Log to Output Pin Only` and `Log Each Transaction To:`. If logging is enabled, each test logs a record. This button toggles to `LOGGING DISABLED`.
IF PASS	Determines branching instructions. If the test passes, VEE goes to this line for branching instructions. `IF PASS` tells VEE to branch according to the selection in the drop-down menu. This button also toggles to `IF PASS CALL:`. `IF PASS CALL:` tells VEE to call the stated function, then go to the branching menu selection. (Refer also to `THEN CONTINUE`, which is the next item in this table.)

Table 9-1. Sequence Transaction Dialog Box (Continued)

Sequence Transaction Field	Explanation
THEN CONTINUE	Determines test branching.The drop-down menu `THEN CONTINUE` (for `IF PASS` and `IF FAIL`) contains six branching options: ■ `THEN CONTINUE` executes the next test configured in the `Sequencer`. ■ `THEN RETURN:` tells VEE to stop executing tests and put the specified expression on the `Return` output pin of the `Sequencer`. ■ `THEN GOTO:` jumps to the test named in its field. ■ `THEN REPEAT` repeats the current test up to the number of times specified in the `MAX TIMES:` field. If the `PASS/FAIL` condition still exists after the maximum number of repeats, then VEE continues with the next test. ■ `THEN ERROR:` stops execution by generating an error condition with the given error number. An error can be trapped with the `Error` output pin on the `Sequencer`. No other output pins will send data. ■ `THEN EVALUATE:` calls the specified `UserFunction`, which must return a string that states a branching menu option. Valid string results from the `UserFunction` are: `"Continue"`, `"Return <`*expr*`>"`, `"Goto <`*name*`>"`, `"Repeat <`*expr*`>"`, `"Error <expr>"`, where <*expr*> is any valid VEE expression and <*name*> is the name of a test in the sequence. This option allows you to ask the user what to do next.

Table 9-1. Sequence Transaction Dialog Box (Continued)

Sequence Transaction Field	Explanation
IF FAIL	Branching instructions. If the test fails, VEE goes to this line for branching instructions. `IF FAIL` toggles to `IF FAIL CALL:`. Options are the same as for `IF PASS`.
DESCRIPTION:	Text comments on the test. They will show on the `Sequencer` transaction bar and can be stored with the test record by using the `Logging` folder in the `Properties` dialog box.

Adding or Inserting or Deleting a Test

In this section, you add another test transaction to the `Sequencer` object. You can use the same `random()` function to simulate a test result, but this time you will compare the result to a limit instead of a range of values.

1. Double-click below the first `Sequencer` transaction bar to get the `Sequence Transaction` dialog box. Fill in the fields as follows:

test2	Use the default.
SPEC NOMINAL	Change the settings from `.5` to `.25`.
RANGE	In the drop-down menu, select `LIMIT:`. Choose `<` for the operator. Change `1` to `.5` for the limit.
FUNCTION	Change the field from `testFunc(a)` to `random()`.

   Leave the other default selections and click `OK` to return to the `Sequencer`.

Note You could also add a transaction after the highlighted one by selecting `Add Trans...` from the object menu.

The `Sequencer` test plan should now have a second transaction that reads: `test2 (.25) < .5`. Now insert a transaction between these two tests.

2. Make sure the second transaction bar is highlighted. Then open the object menu and select `Insert Trans...`. Fill in the fields as follows:

TEST	Change the name field to `Insert`.
FUNCTION	Change to `random()`.

   Click `OK`. You will now see `Insert 0 <= (.5) <= 1` on the second transaction bar. Run the program to see the three records from the three tests. (You may have to enlarge the display to see all the entries.)

3. Now delete `Insert` by clicking the `Insert` transaction bar, placing the mouse pointer over the `Insert` transaction bar, and pressing **Ctrl-K**.

Note

You could also click the target transaction bar and select `Cut Trans` from the object menu. You can also paste a transaction that has been cut by choosing `Paste Trans` from the object menu (**Ctrl-Y** is the shortcut). And in a similar fashion, you can copy a transaction with the `Copy Trans` selection.

4. Run the program and note the two records of data from the two tests. Save the program as `seq1.vee`. The program should look like Figure 9-3.

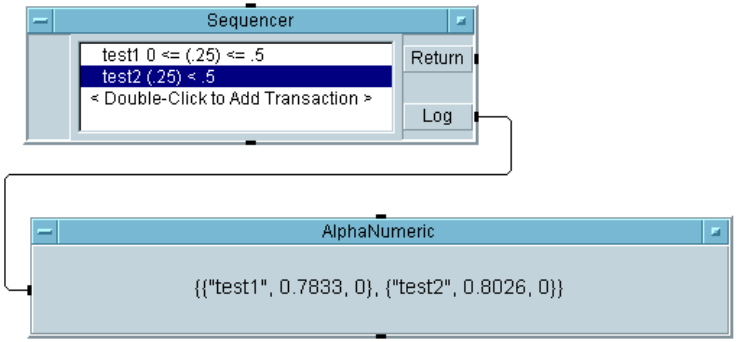


Figure 9-3. A Simple Sequencer Example

The braces indicate a `Record` data type. The `Sequencer` outputs a `Record of Records`, as shown in the `AlphaNumeric` display. This means you could put the sequencer in a loop and run the same sequence of tests several times yielding an array of `Records of Records`.

Accessing Logged Test Data

The `Sequencer` outputs a `Record of Records`. Each test uses the test name as its field name in the `Sequencer` record. The fields within each test are named according to the logging configuration. Using the default configuration with the fields `Name`, `Result`, and `Pass`, you could access the result in `test1` with the notation `Log.Test1.Result`, as shown in Figure 9-4.

.

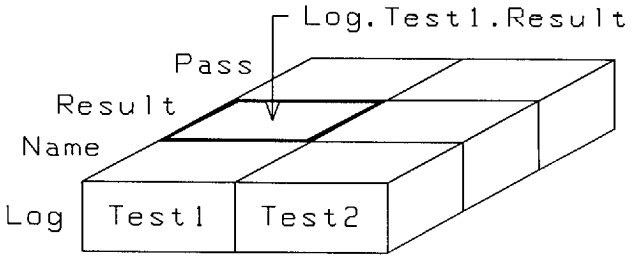


Figure 9-4. A Logged Record or Records

The following steps access test results.

1. Open `seq1.vee`.

2. Select `Device` ⇒ `Formula` and place it below the display. Change the expression to `Log.Test1.Result`. (Remember that VEE is not case sensitive and the capitals in the names are for clarity in documentation.)

   Change the input terminal name from `A` to `Log`. (You could leave the default name `A`. The formula would then read `A.Test1.Result`.) Connect the `Sequencer` output terminal `Log` to the `Formula` input terminal `Log`.

3. Select `Display` ⇒ `AlphaNumeric` display and connect it to the `Formula` output.

4. Run the program and it should access the `Result` field in `Test1`. Save the program as `seq2.vee`. The program should look like Figure 9-5.

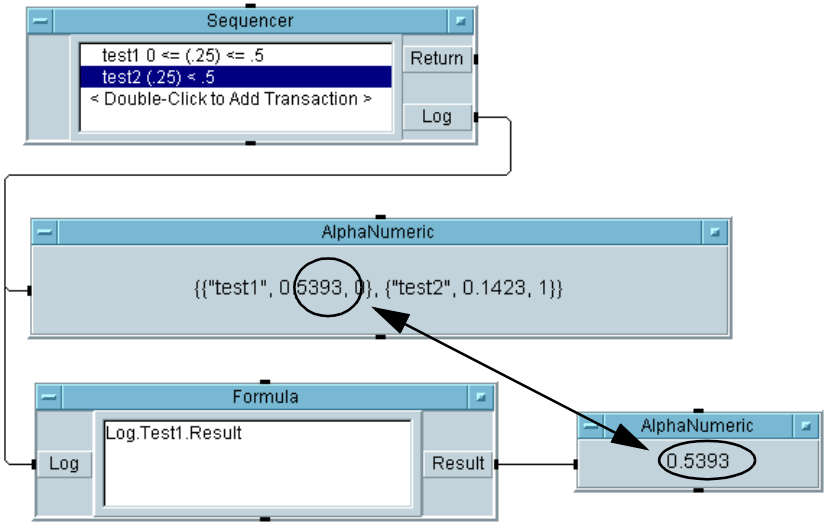


Figure 9-5. Accessing Logged Data

Note

Each test creates a record, named with the test name, as it executes within the `Sequencer`. This record can be used in subsequent tests. For example, you could enable `test2` if `test1` passed (`ENABLED IF: test1.pass == 1`). If you need to access test data in an expression field while the test is still running, test data is stored in the temporary record `thistest`.

5. Change the formula to read `Log.test1` and run the program again. It should retrieve the entire record for `test1`, which is indicated by the braces around the three values in the display.

6. By changing the formula, you can access the `result`, `pass`, `name`, and other fields in the `test1` and `test2` records. Select `Logging` tab in the `Properties` box and add the `Nominal` and `Time Stamp` fields to the logged records. Access these new fields with the `Formula` object.

Passing Data in the Sequencer

In this lab, you will create a `UserFunction` and call it from three different tests. In the first part, you will pass data to the `UserFunctions` through an input terminal on the `Sequencer`. In the second part, you will modify the program to use a global variable instead of an input terminal. This will give you a chance to call a function in `EXEC` mode rather than `TEST` mode. In the third part, you will learn how to test a waveform output against a mask.

Lab 9-2: Passing Data Using an Input Terminal

First, follow the steps to create the `UserFunction Rand`, which will simulate a measurement procedure. `Rand()` will add an input parameter to the output of the `random(low,high)` object, and put this result on the output pin. `Rand()` will be called from three different tests.

1. Select `Device` ⇒ `UserFunction`. Change the name from `UserFunction1` to `Rand`.

2. Get the `random(low,high)` function, delete the input terminals, delete the parameters, and place it in `Rand`. (Recall that without parameters, the defaults will be `0` and `1`.) Place an `A+B` object to the right of `random(low,high)`. Connect the output of `random(low,high)` to the upper left input of the `A+B` object.

3. Add a data input terminal to `Rand`. Connect the input terminal `A` to the lower left input terminal of the `A+B` object.

4. Add a data output terminal to `Rand`. Connect the output of the `A+B` object to the `Rand` output terminal.

   The `UserFunction Rand` should look like Figure 9-6.

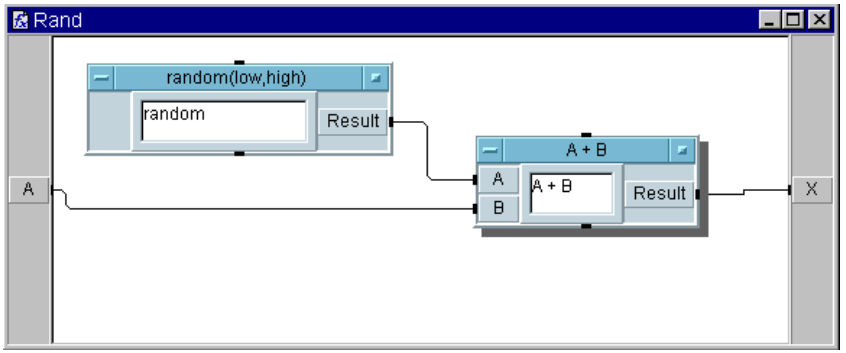


Figure 9-6. The Rand UserFunction

5. Save the program as `seqdat1.vee`. Close the `Rand` window using the **x** button on its top right corner.

Note

Closing the window does not remove the `UserFunction`. If you want to check this, just click on `Edit` ⇒ `Edit UserFunction` and you will see `Rand` come up in a list box of `UserFunctions` to edit. Or you can iconize the `Rand` function and you will see the icon for it in the bottom of the VEE screen.

Now set up three tests in the `Sequencer` to call `Rand` using a `Sequencer` input pin to feed the input parameter to `Rand`.

6. Select `Device` ⇒ `Sequencer` and place it in Main. Add an input terminal to the `Sequencer`. Click the transaction bar to get the `Sequence Transaction` dialog box. Change the `FUNCTION` field from `testFunc(a)` to `rand(a)`. This will call the `UserFunction Rand()` and send it the value on the `Sequencer` input terminal `A`. Click `OK` to get back to the `Sequencer` open view.

Note

You could also use a `Sequencer` input terminal name, such as `A`, to pass data to any of the expression fields within the `Sequence Transaction` box. For example, you could use `A` to pass data to `RANGE:` and `SPEC NOMINAL:`.

Make sure the transaction is highlighted, place the cursor on the transaction bar, press **Ctrl-K** to cut the test, then press **Ctrl-Y** three times to paste the test back into the `Sequencer`. (You could also cut and paste using the object menus.)

The default test names will be `test1x2`, `test1x1`, and `test1`. Open the three `Sequence Transaction` dialog boxes and change these names to `test1`, `test2`, and `test3` for clarity.

1. Select `Data` ⇒ `Continuous` ⇒ `Real64 Slider` and place it to the left of the `Sequencer`. Change the name to the prompt `Select Num:`, size the object to be smaller, and connect it to the `Sequencer` input terminal.

   Tip: You can size an object as you place it by clicking and dragging the object corners using the left mouse button.

2. Select an `AlphaNumeric` display, place it below the `Sequencer`, enlarge it to be wider, and connect it to the `Log` output terminal on the `Sequencer`.

3. Save the program again as `seqdat1`. Select a number on the `Real64 Slider` object and run `seqdat1`. It should look like Figure 9-7.

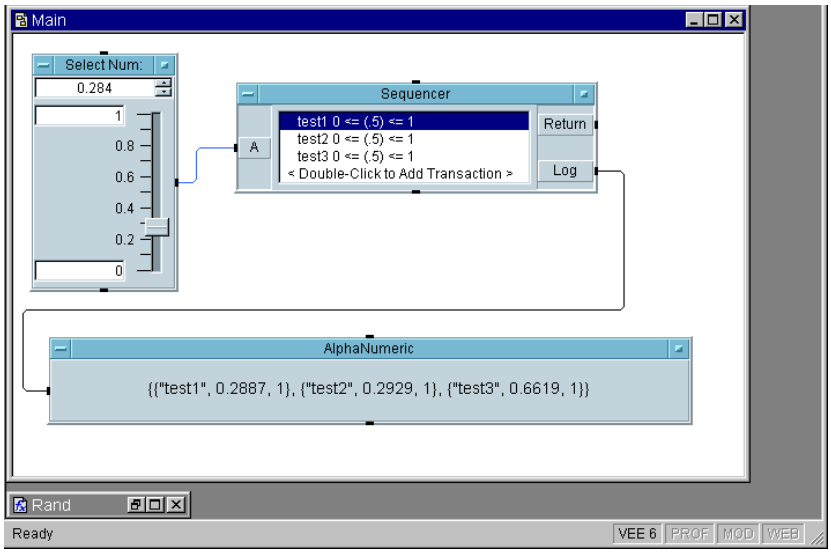


Figure 9-7. Passing Data Using an Input Terminal

As the number of tests increases, passing data using an input terminal requires more and more input pins. To reduce the input pins, you could pass records to input terminals and use individual fields in the records for the separate tests. You could also use a separate `UserFunction` to set up global variables, which can then be called by other `UserFunctions` or any expression field within the program. The next exercise illustrates this.

Passing Data Using a Global Variable

This exercise modifies the `seqdat1` program by adding a global variable to pass the parameter a to the `UserFunction Rand`.

1. Delete the `Real64 Slider` object labeled `Select Num`. Delete the `A` input terminal on the `Sequencer`.

2. Highlight the `test1` transaction bar, open the object menu, and click `Insert Trans...`. When the `Sequence Transaction` box appears, click `TEST` to toggle the selection to `EXEC` and change the name to `Setup`.

You will use EXEC mode, since the User Function will only set up a global variable and will not yield a result that needs to be tested against a specification.

3. Change the FUNCTION field to global() and click OK to close the dialog box.

   You will now create the UserFunction global().

4. Select Device ⇒ UserFunction. Change the name UserFunction1 to global.

   Select Data ⇒ Continuous ⇒ Real64 Slider and put it in the UserFunction, change the name to Select Num:, and size it to be smaller vertically.

   Select Data ⇒ Variable ⇒ Set Variable and place it to the right of the Real64 Slider.

   Change the global variable name from globalA to a. Connect the Real64 Slider to the Set Variable object.

To display the function on the screen for an operator to select a number, add a pop-up panel view. Include a Confirm (OK) button, so that the panel remains on the screen until the operator has made a selection. (It is also possible to do these tasks with a Real Input Dialog Box inside the global() UserFunction.)

5. Select Flow ⇒ Confirm(OK) and place it above the Real64 Slider object. Connect the OK data output pin to the Real64 Slider sequence input pin.

Note

If you place the OK button below the Set Variable object it causes a logic error. That is because VEE sends the old value on the Slider to the Set Variable object and pauses until the OK button is pressed. Any new value you entered on the pop-up panel is ignored. When OK is connected *above* the Real64 Slider, VEE waits to set the global variable until after the OK is pressed, and therefore uses the new Slider value. You can turn on Show Data Flow to watch the execution order.

6. Select `Display ⇒ Note Pad` and place it to the right of the `OK` button. Enter the following user prompt in the `Note Pad`:

   ```
   Please select a number for this run of tests 1, 2, and
   3.
   ```

7. Select the `Note Pad`, the `Real64 Slider`, and the `OK` button by pressing **Ctrl** and clicking on those objects. Each object will now have a shadow to indicate it is selected. Click `Edit ⇒ Add To Panel`. (Remember the `Edit` menu is also available via the right button on an open area of the VEE screen or the detail view of a `UserObject` or `UserFunction`.) In Panel view, size the panel to be smaller, and position the `Note Pad` on top, the `Real64 Slider` in the middle, and the `OK` button on the bottom.

Note

When you reposition the objects in Panel view, it does *not* affect the layout of the objects in the Detail view.

Open the `UserFunction Properties` window. In the `General` folder under `Pop-up Panel`, click to select to `Show Panel on Execute`. Figure 9-8 shows the `UserFunction` in detail view, and Figure 9-9 shows the panel view.

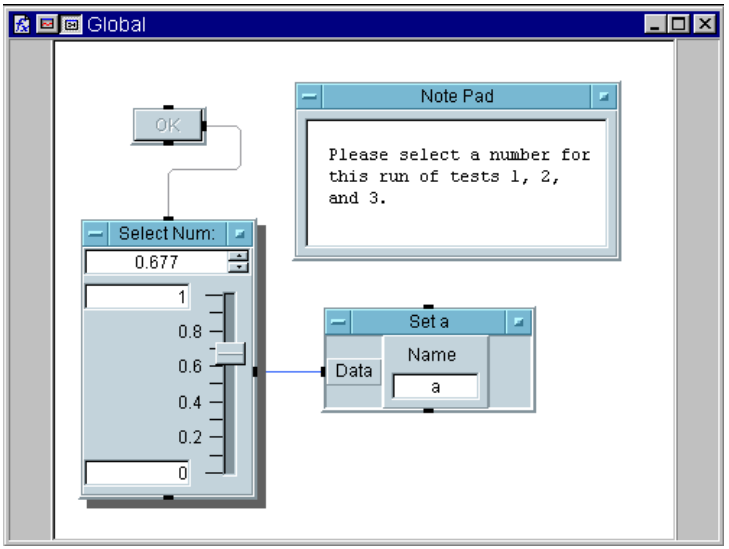


Figure 9-8. The Global UserFunction (Detail)

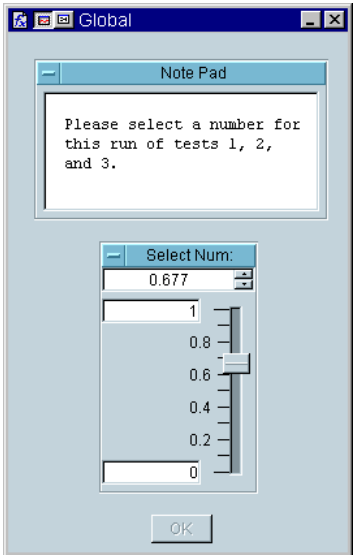


Figure 9-9. The Global UserFunction (Panel)

8. Save the program as `seqdat2` and run it. When the pop-up panel appears, select a value and press `OK`. It should look like Figure 9-10.

Note

The pop-up panel will appear in the center of the screen by default. To move it, click and drag the title bar.

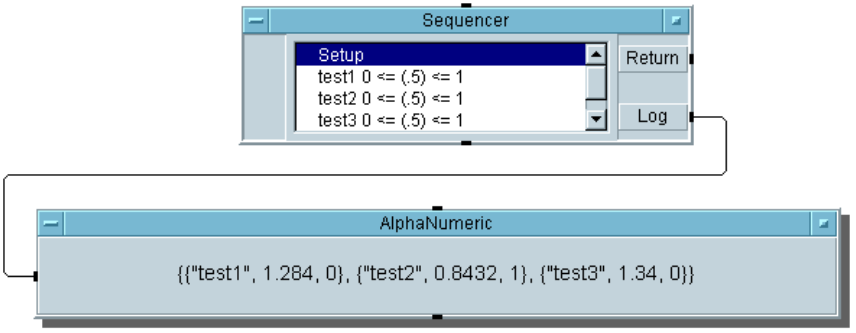


Figure 9-10. Passing data Using a Global Variable

Comparing a Waveform Output with a Mask

In this exercise, you will create a `UserFunction` called `noisyWv` and call it from a single transaction bar in the `Sequencer`. The operator will be able to vary the amplitude of the wave from `0` to `1`. This function simulates a test result that returns a noisy waveform. You will use the `Coord` object in the `Data` ⇒ `Constant` menu to create a straight line mask at `0.6`, which the `Sequencer` will use to test the noisy waveform.

1. Create the `UserFunction` called `noisyWv`, as shown in Figure 9-11 in the `Detail` view.

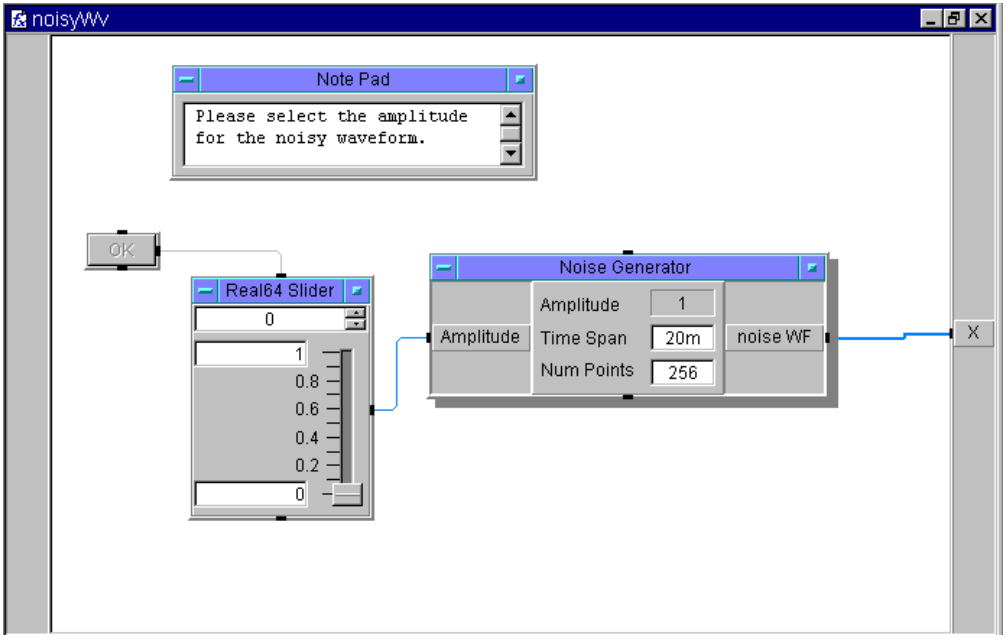


Figure 9-11. The noisyWv UserFunction (Detail)

2. Press **Ctrl** and click on the `OK` button, the `Real64 Slider`, and the `Note Pad` to highlight them for creating a Panel view. Select `Edit` ⇒ `Add To Panel`.

   When the `Panel` view displays, rearrange the objects to your taste, and size the window.

   Open the object menu, click `Properties`, and under `Pop-up Panel` click next to `Show Panel on Execute`.

   The `Panel` view should look like Figure 9-12.

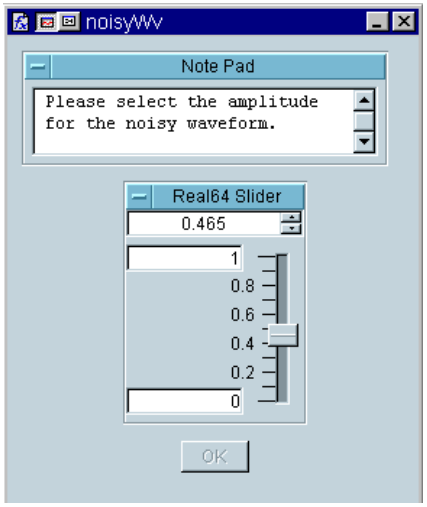


Figure 9-12. The noisyWv UserObject (Panel)

3. Select `Device` ⇒ `Sequencer` and place it left-center of Main. Add a data input terminal and name it `mask`.

4. Click the transaction bar to get the `Sequence Transaction` dialog box. Change fields as follows:

FUNCTION	Type in `noisyWv()`.
RANGE	Click and select `LIMIT` from the pop-up menu. Leave the `<=`, and the `LIMIT` terminal name field type `mask`. All of the other defaults are fine, so click `OK`.

`test1` will get a result from `noisyWv()` and test it against the limit value at the input terminal named `mask`. If the noisy wave is less than or equal to the mask at all points, it will pass. Otherwise, it will fail.

5. Select `Data` ⇒ `Constant` ⇒ `Coord` and place it above the `Sequencer`. Connect its output to the `Sequencer` input terminal `mask`.

   Open the `Coord` object menu, click `Properties`, and set fields as follows:

Configuration	Set to 1D Array.
Size	Type in 2 (you only need two pairs of coordinates to specify a straight line.)

   Click `OK`.

6. In the `Coord` object, you now see two indices for pairs of coordinates. Double-click the first index, `0000:` and a cursor will appear. Type in the coordinates separated by a comma, and VEE adds the parentheses automatically. Type `0, 0.6` **Tab** `20m, 0.6` and then click on the work area outside the object. The entries are as follows:

   - The `x` axis (time axis) for the `Noise Generator` in `noisyWv()` goes from 0 to 20 milliseconds; hence, the two `x` values of `0` and `20m`.
   - The two `y` values are both `0.6`, since you want a straight line mask.

Note You can create any mask waveform by configuring the proper number of coordinate pairs and filling them in.

   The `Sequencer` comparison mechanism operates just like the `Comparator` object, which accepts the `Coord` data type to test waveforms. Of course, you could also compare two Waveform data types. Press **Tab** to move between coordinate pairs, and click on the work area when you are done.

7. Select an `AlphaNumeric` display, increase its width, and connect it to the `Log` output from the `Sequencer`.

8. Save the program as `seqdat3` and run it. It should look like Figure 9-13.

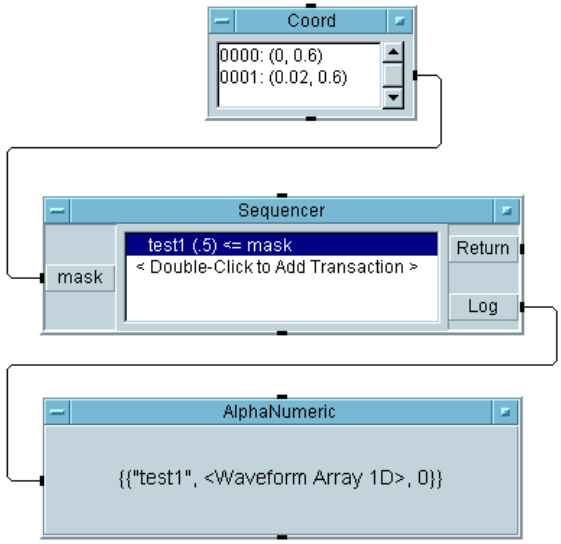


Figure 9-13. Comparing a Waveform to a Mask

This completes the exercises about passing data with the `Sequencer`. In the next exercise, you will learn how to access and analyze data from several iterations of the `Sequencer`.

Analyzing Data from the Sequencer

As mentioned earlier, `Sequencer` data comes out as a record of records. In many cases, however, the `Sequencer` may run through a series of tests several times. This generates an array of records. Each record represents one run through the `Sequencer` and holds other records, representing each test within a run. The easiest way to visualize this is to imagine a cube of data in memory, as shown in Figure 9-14.

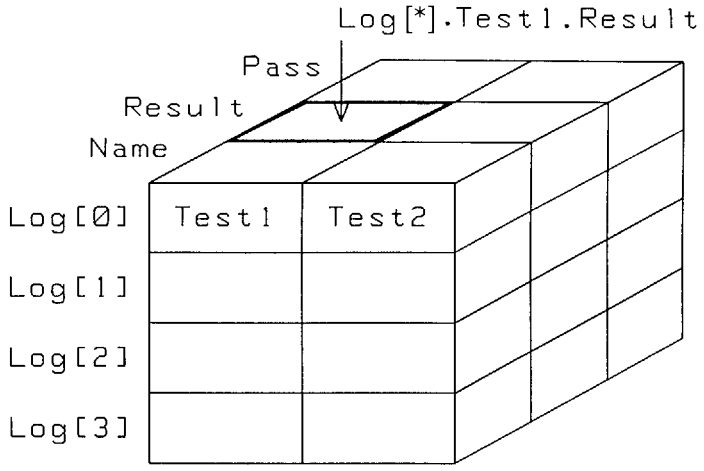


Figure 9-14. A Logged Array of Records of Records

The array of records is called `Log`, because that is the name associated with the `Sequencer` output pin. To access a particular run, use array indexing with the bracket notation.

- `Log[0]` is the first run through the `Sequencer`, `Log[1]` is the second run, and so forth.
- The main record for each run has two fields, `Test1` and `Test2`.

- Within the record `Test1` there are three fields: `Name`, `Result`, and `Pass`. The same holds for the record `Test2`.

- Therefore, `Log.Test1.Result` gives an array of four values, each representing one of the four runs. `Log[0].Test1.Result` outputs a scalar value, the `Result` of `Test1` in the first run (`Log[0]`).

The logged array of records simplifies analyzing and investigating the data. For example, you might want to know how many tests passed in a particular run. Or you might want to average the results of `Test2` in all runs. Or you might want to see all of the data on `Test1` in the fourth run. All of these queries are possible with this data structure. The next exercise performs some analysis operations on data.

Lab 9-3: Analyzing Several Runs of Data from the Sequencer

1. Clear the screen and open the `seqdat1.vee` program.

Modify the `seqdat1.vee` program to run through the `Sequencer` three times. Then perform some analysis operations on the data.

2. Select `Flow` ⇒ `Repeat` ⇒ `For Count` and place it above the `Real64 Slider` object. Change the number of iterations to `3`, and connect the data output pin to the sequence input pin of the Sequencer.

3. Delete the data line between the `Sequencer Log` pin and the display. Select `Data` ⇒ `Collector` and place it to the right of the `Sequencer`. Connect its upper left data input pin to the `Sequencer Log` pin and its `XEQ` pin (lower left) to the sequence output pin on the `For Count` object. Connect the `Collector` data output pin to the `AlphaNumeric` display. Enlarge the display vertically somewhat to accommodate an array with three elements.

   The `Sequencer` will now run through `test1` and `test2` three times and collect the data into an array with three elements, each one holding a record of records for each run. (Refer to the cube of data in Figure 9-14 to visualize this.)

Run the program at this point to see the display of the `Sequencer` data.

Now use the `Formula` object to extract part of the data to analyze. This exercise uses the results of `test1` for all three runs as an example, and finds the mean of that array.

4. Select `Device` ⇒ `Formula` and place it below the display. Connect the `Formula` input pin to the output of the `Collector`. Change the `Formula` input field to read: `a[*].test1.result`. Connect a `mean(x)` object to `Formula`, and an `AlphaNumeric` display to `mean(x)`.

   The a refers to the array on the input terminal `A`. `Test1.result` accesses the proper field. All runs will be shown in an array. (`A[0].test1.result` would refer to the first run only, for example.)

5. Run the program. It should look like Figure 9-15.

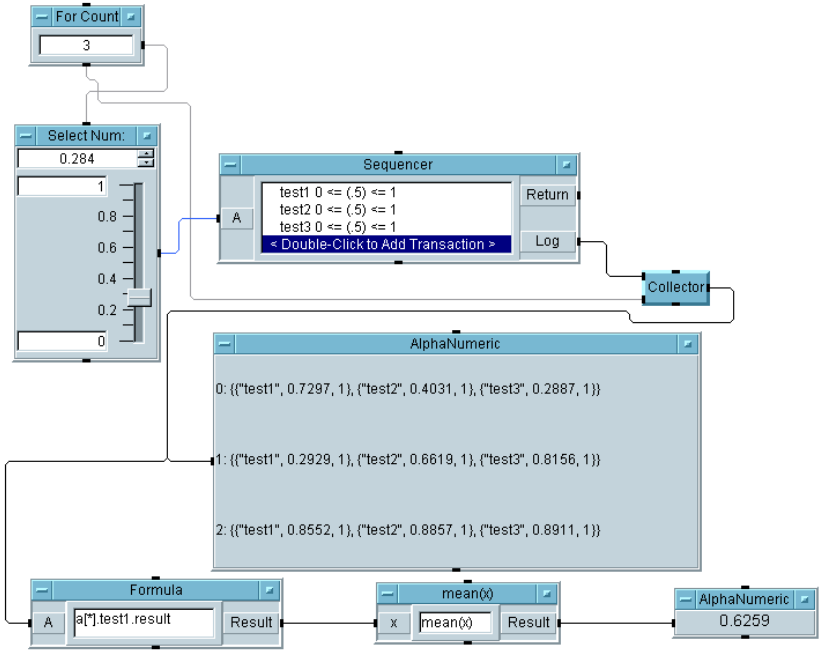


Figure 9-15. Analyzing Several Runs of Sequencer Data

Although this exercise accesses a single array, the principle is the same for extracting other arrays of data from the `Sequencer` output. Note that you can easily change which fields are saved for each by opening the `Logging folder` in the `Sequencer Properties` dialog box.

Storing and Retrieving Logged Data

This exercise shows how to use the `To/From File` objects and `To/From DataSet` objects.

Lab 9-4: Using the To/From File Objects with Logged Data

1. Open the `seqdat2` file and delete the data line to the display.

2. Select `Flow` ⇒ `Repeat` ⇒ `For Count` and place it to the left of the `Sequencer`. Change the `For Count` number to `3`, and connect its data output pin to the sequence input pin on the `Sequencer`.

3. Enlarge the work area vertically and place the `AlphaNumeric` display near the bottom. Select `Data` ⇒ `Collector` and place it in the left work area. Connect the `Sequencer Log` pin to the `Collector` data input pin. Connect the `For Count` sequence output pin to the `Collector XEQ` pin.

   The `Collector` will create an array of records of records from the `Sequencer`. Using the `WRITE CONTAINER` transaction in the `To File` object you can write any VEE data container to a file quite easily.

4. Select `I/O` ⇒ `To` ⇒ `File` and place it to the right of the `Collector`. Select `I/O` ⇒ `From` ⇒ `File` and place it below the `To File` object. Add an input terminal to the `To File` object and connect the `Collector` output to it. Connect the `To File` sequence output to the `From File` sequence input pin. Connect the `From File` data output to the display.

   Check `Clear File At PreRun & Open` in `To File`, and configure a `WRITE CONTAINER a` transaction. Configure a transaction in the `From File` object like the following: `READ CONTAINER x`.

   You can use the default data file for storage.

5. Run the program. It should look like Figure 9-16.

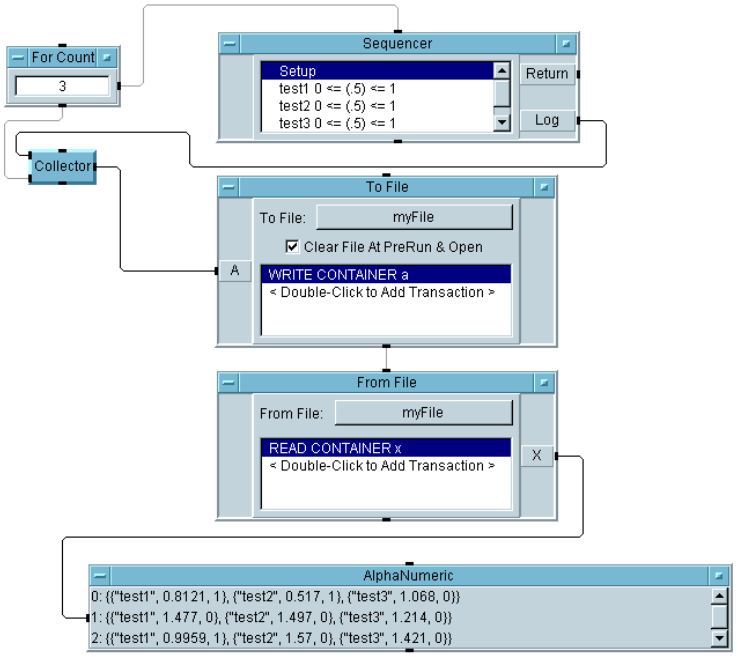


Figure 9-16. Storing Logged Data with To/From File

Using the To/From DataSet Objects with Logged Data

Since you are storing test data as records, you may prefer to use the `To/From DataSet` objects. In this case you do not need a `Collector`, because you can append each run of the `Sequencer` to the end of the `DataSet`.

Modify the last program to look like Figure 9-17. The `To/From DataSet` objects are in the `I/O` menu. Notice the sequence line going into `From DataSet`. The sequence line is included because you want to wait for all three runs to be appended to the `DataSet` before you trigger `From DataSet`.

One reason you might use the `To/From DataSet` objects to collect data instead of the `To/From File` objects is because you can convert the data to useful information with the `Search Specifier` feature in the `From DataSet` object.

Note

Remember to change the `Get records` field in `From DataSet` from `One` to `All`.

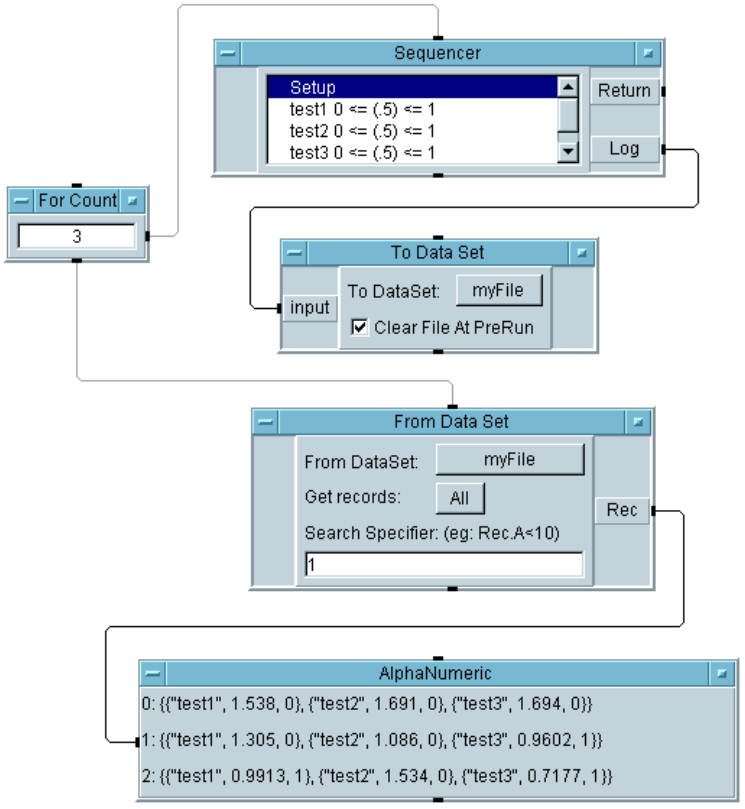


Figure 9-17. Storing Logged Data with To/From DataSet

Chapter Checklist

Use the following checklist to determine whether there are topics you need to review before going on to the next chapter.

- Describe the `Sequencer` object conceptually.
- Configure a test for the `Sequencer`.
- Add, insert, and delete operations for a `Sequencer` test.
- Access logged data from the `Sequencer`.
- Use `Sequencer` input terminals to pass data to tests.
- Use `Global variables` to pass data to tests.
- Compare a waveform output to a mask.
- Analyze several runs of data from the `Sequencer`.
- Store data using the `To/From File` objects.
- Store data using the `To/From DataSet` objects.

10

Using Operator Interfaces

Using Operator Interfaces

In this chapter you will learn about:

- Building operator interfaces
- Using menus for an operator
- Importing bitmaps to add clarity
- Securing test programs
- Operator interface features
- Using ActiveX Controls to extend capabilities of VEE

Average Time To Complete: 2 hours

Overview

In this chapter, you will learn more about operator interfaces, including adding menus, customizing interfaces, adding warning signals, and importing bitmaps. This chapter expands on the exercises in previous chapters, where you created operator interfaces and pop-up panels.

Some benefits of using VEE operator interface features are:

- Maximum ease of use for the operator
- Improved program performance
- Security from unauthorized changes
- Clarity through visual aids

Key Points Concerning Operator Interfaces

This section is an overview of how to create an operator interface in VEE.

Creating an Operator Interface

VEE includes a wide range of selection controls, pop-up dialog boxes, indicators, and displays to create operator interfaces. Selection controls include items such as buttons, switches, check boxes, drop-down menus, and list boxes. Indicators include items such as tanks, thermometers, fill bars, vu meters, and color alarms.

In addition to the operator interface elements provided within VEE, you can add elements from other sources. There are thousands of operator interface elements that can be downloaded from the World Wide Web. There are operator interfaces that are available through ActiveX controls. (Some items that you download may be free and some may charge a fee.)

Whether you use operator interface objects that are all provided in VEE or add outside elements of your own, the process for creating an operator interface is the same.

To create an operator interface for a VEE program, you create a Panel view of the program.

1. Select the object or objects that you want in the panel view, by holding down the **Ctrl** key and clicking each object to select it.

2. Select `Edit` ⇒ `Add To Panel`. The screen switches to Panel view, shown by default in blue, that includes the objects you highlighted from the Detail view.

You now have a view of the VEE program that you can customize to show only what the operator needs to see.

Moving Between Panel View and Detail View

To move between the Panel view and the Detail view of a VEE program, click the panel or detail icon on the *title bar* of the window as shown in Figure 10-1.

Note

You must create a Panel view of the program to have the panel view button displayed in a window title bar.

Typically, you develop the program in Detail view and then create a Panel view for the operator interface. The Panel view button can be on the title bar of a `UserObject` window, `UserFunction` window, or Main window.

.

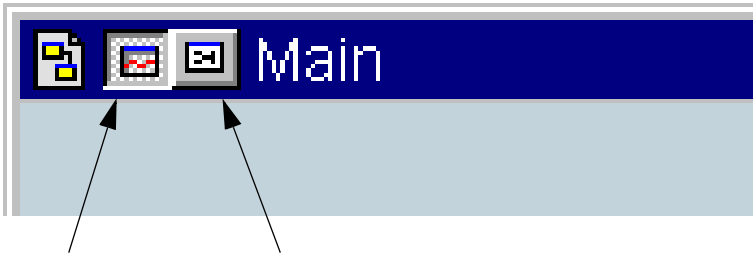


Figure 10-1. Panel View Button and Detail View Button in Title Bar

Customizing an Operator Interface

In the Panel view of a VEE program, you can change the size of objects, rearrange objects, and change the way the objects are displayed *without affecting the same objects in the detail view.* For example, you could remove the title bar and the scales from the Panel view of a `Waveform (Time)` display without affecting the `detail` view of the same `Waveform (Time)` display. However, if you delete an object in the detail view, it will also be deleted in the panel view.

In panel view, you can choose different colors and fonts to add emphasis, and scalable bitmaps to add clarity. You can also document the panel view for the operator by editing title bars, using the `Note Pad` and `Label` objects, and using the `Description` option in the object menus.

Figure 10-2 shows some of the VEE indicators available.

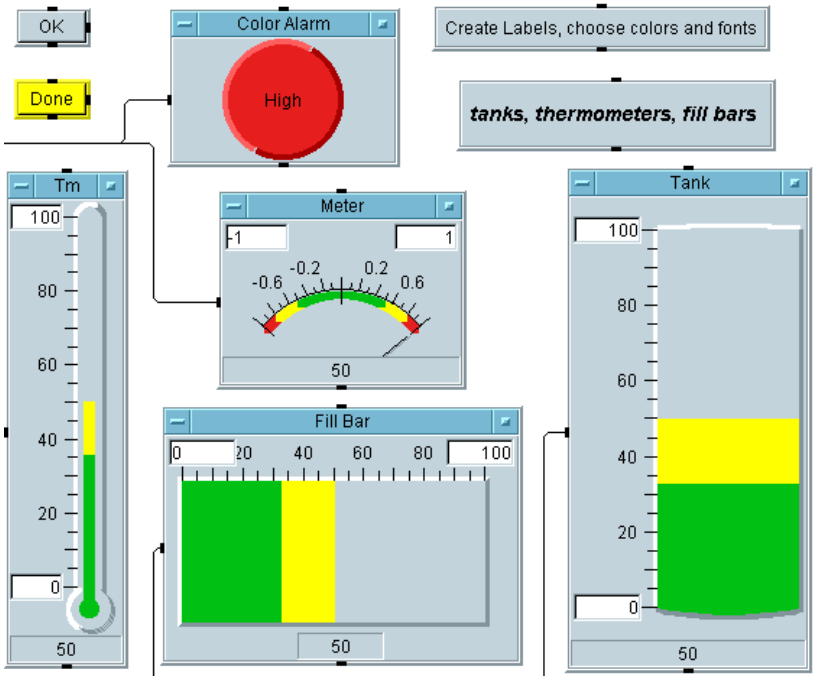


Figure 10-2. A Selection of VEE Indicators

Using Operator Interface Objects

This section introduces the operator interface objects and options that are available in VEE. You can skim through this section to get an idea of the items you can choose to create operator interfaces for programs, and how you can customize them. Then do the lab exercises to see how to set up operator interfaces for some common tasks.

Colors, Fonts, and Indicators

- **Colors and Fonts** You can configure colors and fonts using the `File ⇒ Default Preferences` selection or through the `Properties` selection in each object menu. The choice of colors and fonts depends on the operating system and the fonts you have installed.

- **Color Alarms** Color alarm objects are located in the `Display ⇒ Indicator` menu. They can be configured for three different ranges with color and a text message, and as squares or circles. Alarms are often used to simulate an "LED," or to warn operators of a situation that demands their attention.

- **Tanks, Thermometers, Fill Bars, Meters** These objects are in the `Display ⇒ Indicator` submenu. They can be customized with colors and labels. These indicators can be set to horizontal or vertical formats, and have three default ranges, which can be configured under `Properties` in the object menus.

Graphic Images

You can import bitmaps into the panel view by setting the `Background Picture` in the `Panel` folder of the `Properties` box. VEE imports `*.jpeg`, `*.png`, `*.wmf`, `*.xwd`, `*.GIF`, `*.bmp`, and `*.icn` files to serve as the background for your Main, `UserObject`, or `UserFunction` panel.

When a bitmap is set as the background picture, other VEE objects will appear on top of the picture. (For more information about how to do this, refer to "Importing Bitmaps for Panel Backgrounds" on page 389.) Images

may be scaled, tiled, cropped, or centered. Figure 10-3 shows a VEE logo sized and used as a background image.

Figure 10-3. Logo Used as a Background Picture

Figure 10-4 shows a background picture that has been tiled.

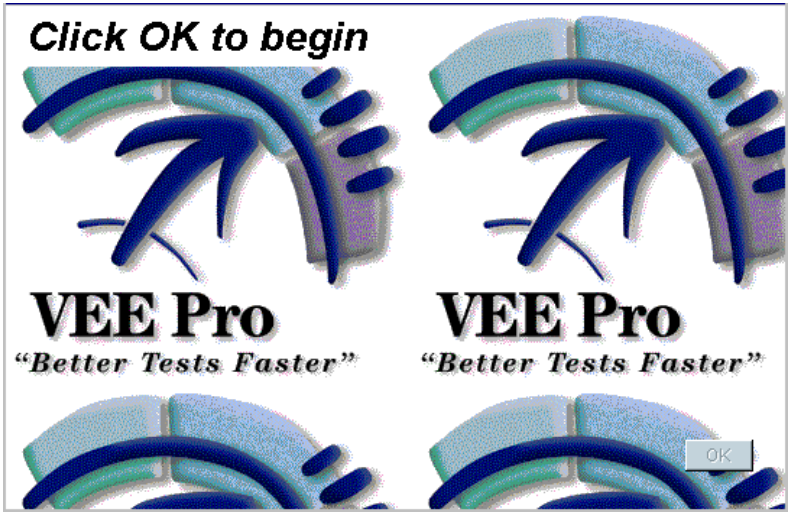


Figure 10-4. Background Picture Used as Tile

There is also a `Picture` object in the `Display` menu, if you want to place a bitmap in a program. Figure 10-5 shows a picture that has been included with `Display` ⇒ `Picture`, and then cropped in VEE.

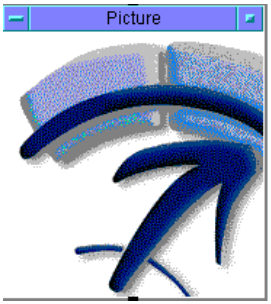


Figure 10-5. A Cropped Image in VEE

Note

You can also change bitmaps for any icon using the `Properties` ⇒ `Icon` tab.

Displaying a Control for Operator Input

There are various ways to set up a program so that an operator can control it by entering input. You can get user input from pop-up dialog boxes, any data constant, sliders, and knobs. To choose a control, look in menus such as `Data` ⇒ `Selection Control`, `Data` ⇒ `Toggle Control`, and `Data` ⇒ `Continuous`. Figure 10-6 shows a collection of the objects you can use to clarify programs for the operator.

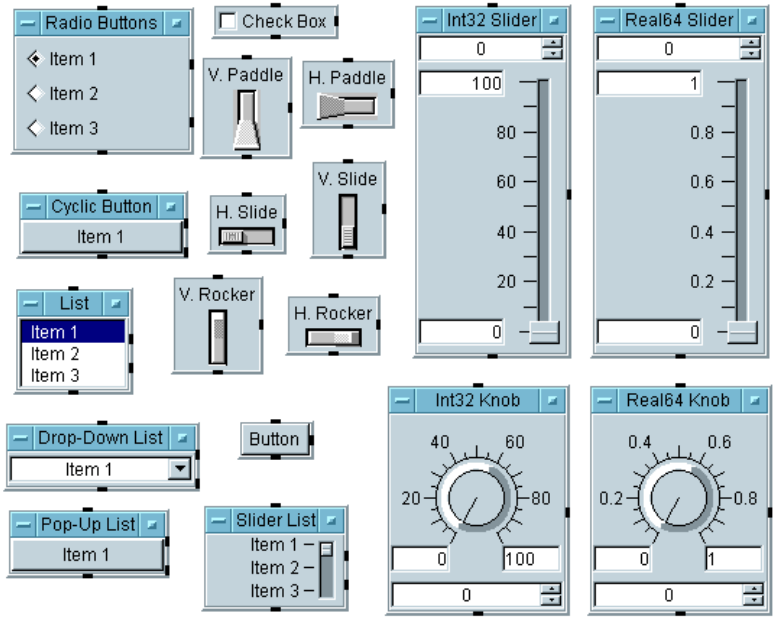


Figure 10-6. Controls from Various Data Submenus

For each of the objects shown in Figure 10-6, you can also customize the object's look and feel. For example, see the `Real64 Knob Properties` dialog box in Figure 10-7. To configure the object, choose a folder such as `Colors` and make selections.

Note

With ActiveX you can also use controls and displays from other applications, as shown in the example "Using an ActiveX Control" on page 396.

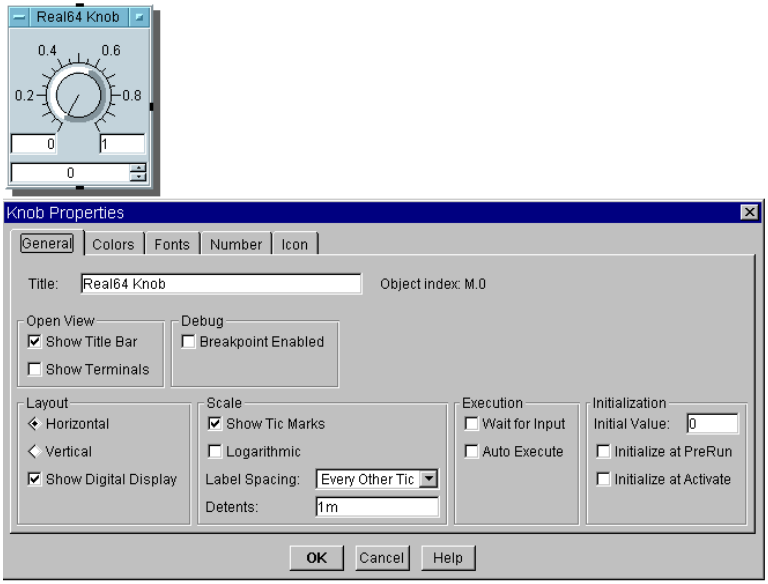


Figure 10-7. The Properties Dialog Box

Displaying a Dialog Box for Operator Input

VEE includes built-in pop-up dialog boxes with automatic error checking, prompts, and error messages. They are located under `Data ⇒ Dialog Box`.

For example, a program could require the operator to enter a real number when the program runs. You can include a `Real64 Input` object in the program that automatically displays a `Real64 Input` box for the operator when the program runs. The `Real64 Input` box also automatically displays an error message if the operator does not enter the correct information at the prompt. Figure 10-8 shows the object to include in the program, and the `Real64 Input` box that appears when the program runs.

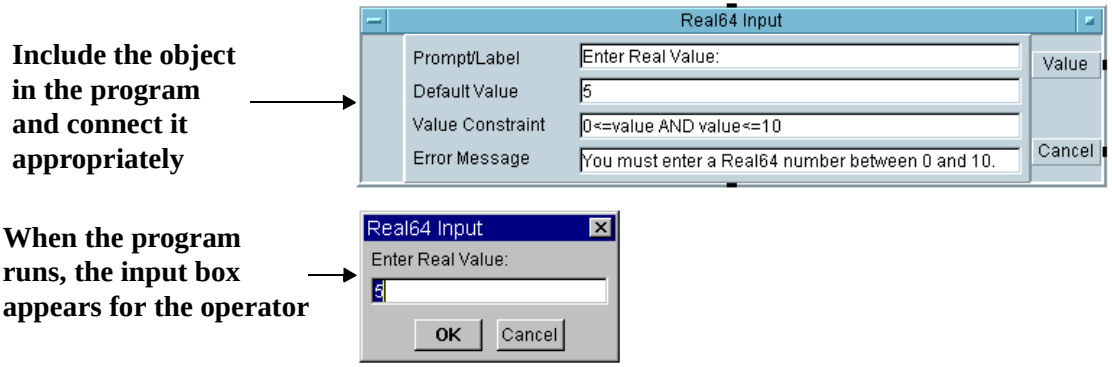


Figure 10-8. A Text Input Box

Figure 10-9 shows the configurable error message that appears if the program runs and the operator presses `OK` without entering correct information into the `Real64 Input` box.

Figure 10-9. An Example of Automatic Error Checking

The input boxes for `Int32` and `Text`, which are also located in `Data` ⇒ `Dialog Box`, are similar to the `Real64 Input`. In addition, the `Data` ⇒ `Dialog Box` menu includes choices for `Message Box`, `List Box`, and `File Name Selection`.

Figure 10-10 shows a dialog box that pops up to display a message.

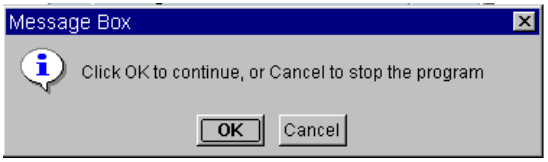


Figure 10-10. A Pop-Up Message Box

Figure 10-11 shows a dialog box that pops up for an operator to enter a list.

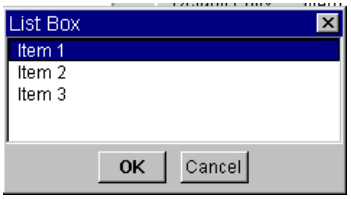


Figure 10-11. The List Selection Box

Figure 10-12 shows a dialog box that pops up for an operator to select a file name.

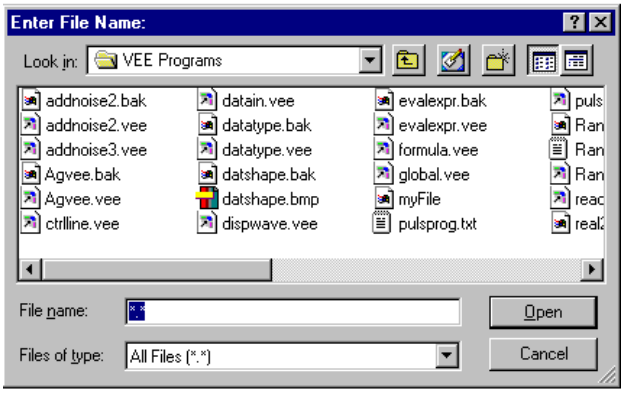


Figure 10-12. A Pop-Up File Selection Box

Displaying a Toggle Control for the Operator

VEE includes built-in toggle controls that can be used to sent out a 0 or a 1. To use a toggle control, set the initial state, and execute a subprogram when the toggle is activated. You can also put custom bitmaps on a `Toggle`.

For example, if you have a program where the operator needs to set switches or alarms, you can use toggle controls. Figure 10-13 shows a panel for the operator to set the switches.

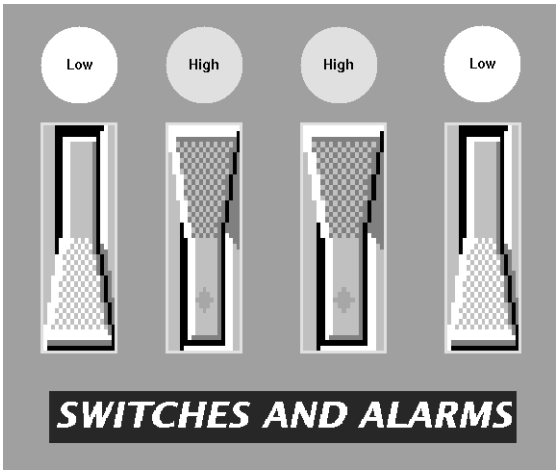


Figure 10-13. Switches and Alarms Combined

Aligning Objects in the Operator Interface

In the panel view, there is a “snap-to-grid” feature to help align objects. You can change the grid size from 10 to 1 (10 is the default) to make very accurate alignments, as shown in Figure 10-14. You can use this feature to give the program a professional look. The “snap-to-grid” feature is located in the `Panel` folder under the `Properties` selection of the `UserObject` or `UserFunction` menu. (Remember, you must have created a panel view for the `Panel` folder selection to display in the dialog box.)

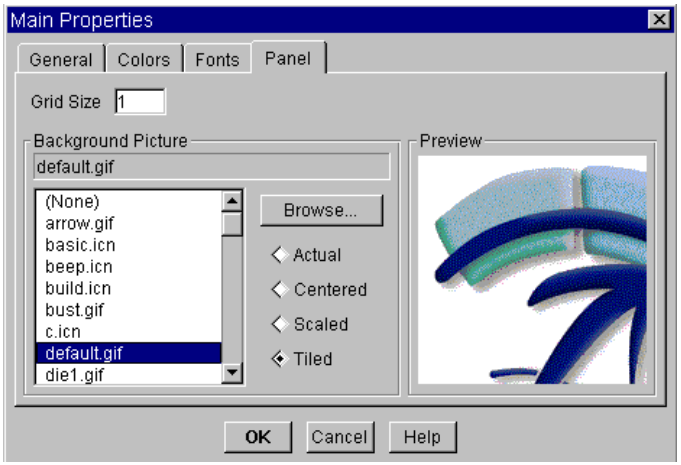


Figure 10-14. Configuring Panel Properties

Creating an Operator Interface for the Keyboard Only

You can also use VEE to create interfaces that the operator can control using the keyboard only. They do not require a mouse.

For example, you can configure the `OK` object to act as a softkey. Typically you configure it so that it is attached to one of the **F**-keys. The operator can then press **F**-keys to control the program, as shown in Figure 10-15.

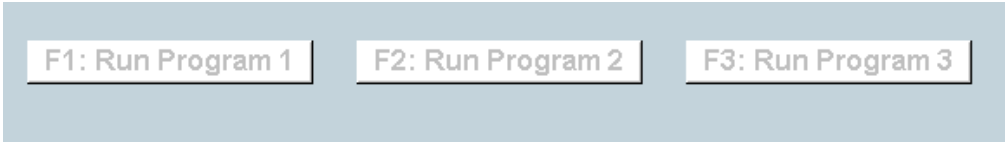


Figure 10-15. A Softkey Executing a UserFunction

Figure 10-16 shows how to configure an `OK` object using the `Properties...` dialog box to connect to a function key, **Enter**, or **Esc** keys.

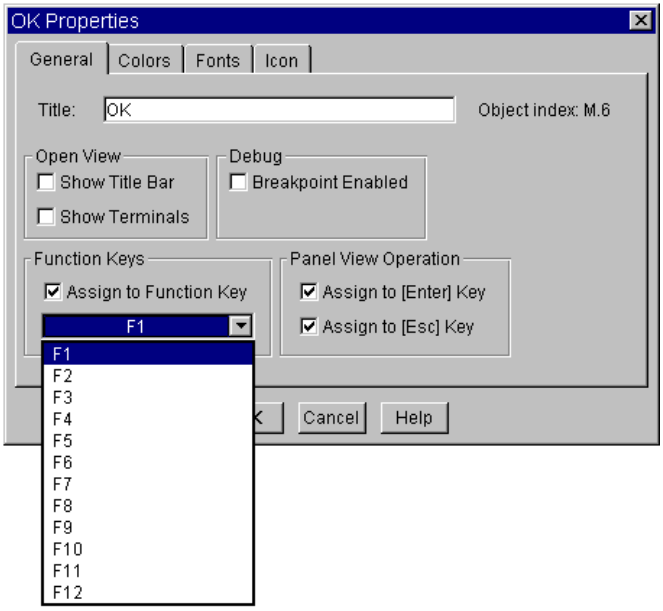


Figure 10-16. Configuring the Confirm (OK) Object as a Softkey

Furthermore, the program can be controlled with the keyboard in panel view. VEE automatically highlights a button for the panel with a dotted outline. If the operator presses **Enter**, that button will be "pressed." If the operator is editing a text input area, pressing the **Enter** key accepts the edit, and pressing the **Esc** aborts the edit. The **Tab** key moves forward through the various input object selections and shows the active object. The **Shift-Tab** keys move backward. Use the following combinations for controlling program execution:

Ctrl-G	Run or Continue (Resume)
Ctrl-P	Pause
Ctrl-T	Step

Selecting Screen Colors

To select screen colors, use the `File ⇒ Default Preferences` dialog box. Set the VEE environment as desired and save the changes. Figure 10-17 and Figure 10-18 show how to change particular screen elements to the desired color.

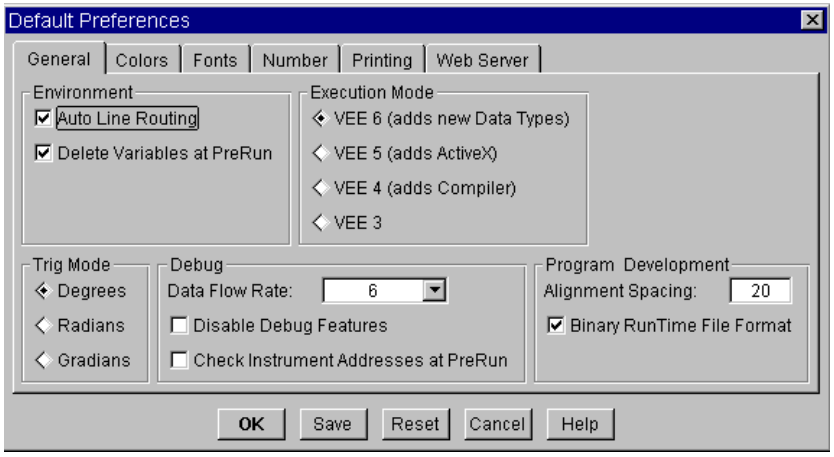


Figure 10-17. The Default Preferences Dialog Box

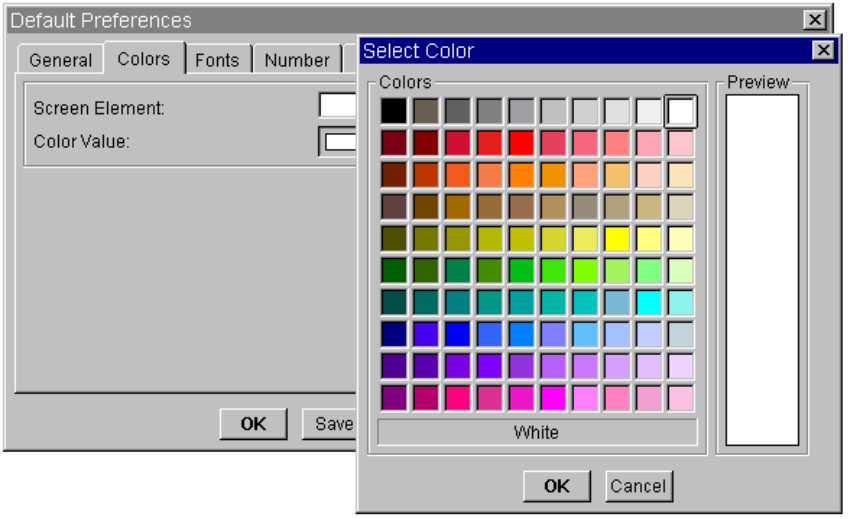


Figure 10-18. Color Selection for Screen Elements

Securing a Program (Creating a RunTime Version)

To prevent an operator from accidentally altering a program, or to prevent others from seeing how a program works (by displaying it in Detail view), you can create a run time version of a VEE program. *Save the original program and the run time version in separate files.*

Note

When you create a run time version of a VEE program, the run time version *cannot* be edited. (You cannot display the `Detail` view.) Therefore, make copies of the original program before you begin this process and follow the instructions carefully.

To create a `RunTime` version of a VEE program, follow these steps:

1. Create a panel view for the run time version of the program.

2. Save the program so that you have a copy you can edit.

3. Select `File` ⇒ `Create RunTime Version...`. VEE will automatically use a `*.vxe` extension to indicate a run time version.

Displaying a Pop-Up Panel During Execution

You can cause a panel to pop up when a `UserObject` or `UserFunction` executes in a program. To display a pop-up panel, select `Show Panel on Execute` under `Properties` in the object menu. To keep the panel on screen until the operator is ready to proceed, add a `Confirm (OK)` object. Otherwise, the panel disappears when the `UserObject` or `UserFunction` is done executing.

To keep a pop-up panel displayed during multiple calls to a `UserFunction`, use the `ShowPanel()` and `HidePanel()` functions. For example, you may want to keep the pop-up panel displayed as a status panel while the program executes. See the next section for an example.

Creating a Status Panel

In VEE, you can implement status panels to monitor the results of multiple tests or functions. This feature is implemented with the `ShowPanel()` and `HidePanel()` functions, as shown in Figure 10-19. For more information, refer to “Creating a Status Panel” on page 398.

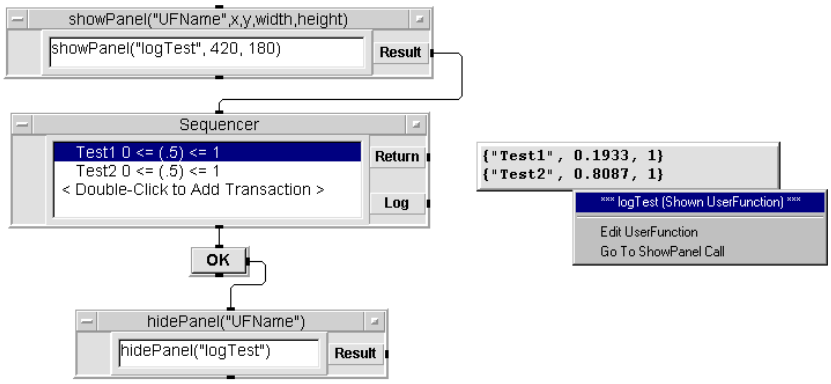


Figure 10-19. Creating a Status Panel

Common Tasks In Creating Operator Interfaces

In the following exercises, you will learn how to implement many operator interface features. Specifically, you will learn how create menus, create warnings, create a status panel, and import bitmaps to add more visual impact to programs. All of the labs will give you a chance to customize the interfaces.

Lab 10-1: Using Menus

In this exercise, you will create an operator interface that includes a menu with three choices: `die1`, `die2`, and `die3`. When the operator selects a choice, a function by the same name will be called that displays a die with one, two, or three dots on its top face. This program simulates a situation where the operator must choose a test to run from a menu. You will also learn how to import a bitmap to change the appearance of an icon. This will be called the Dice Program.

Begin by creating the three `UserFunctions`.

1. Select `Device` ⇒ `UserFunction`.

Although you could use any icon to display the imported bitmap, this example uses the `Picture` object.

2. Select `Display` ⇒ `Picture` and place it in the `UserFunction`.

3. Open the `Picture` object menu, click `Properties`, then deselect `Show Title Bar` under `Open View`. Select `die1.gif` under `Picture`, click `Scaled`, then `OK`.

Note

To access an object menu when `Show Title Bar` is turned off, click the right button over the object.

Note Although VEE defaults to the bitmaps subdirectory, you could use a bitmap from any directory.

You should now have a picture of a die with one dot on its top.

4. Select `Flow ⇒ Confirm (OK)` and place it below the die. Connect the `Picture` sequence output pin to the `OK` sequence input pin.

   Select the `Picture` and the `OK` objects (press **Ctrl** and click the objects to create a shadow). Open the pop-up `Edit` menu by placing the mouse pointer on the background and pressing the right mouse button. Select `Add to Panel`.

5. Change the `UserFunction Title` and `Panel Title` to `die1`. Arrange the objects and size them as desired.

Note To move objects in `Panel View`, right-click on the object and select `Move`.

   Select `Show Panel on Execute` from the `Properties` dialog box. Click the `Panel` folder and change the grid size to `2` for more accurate alignment. Then click `OK`.

6. Create two more `UserFunctions` by selecting `Clone` in the `die1` object menu. The new `UserFunctions` appear automatically as `die2` and `die3`. Change the picture objects to `die2.gif` and `die3.gif` respectively. Check all the settings of the new functions to make sure they match `die1` except for the names and bitmaps. The program should look like Figure 10-20. Iconize the function windows.

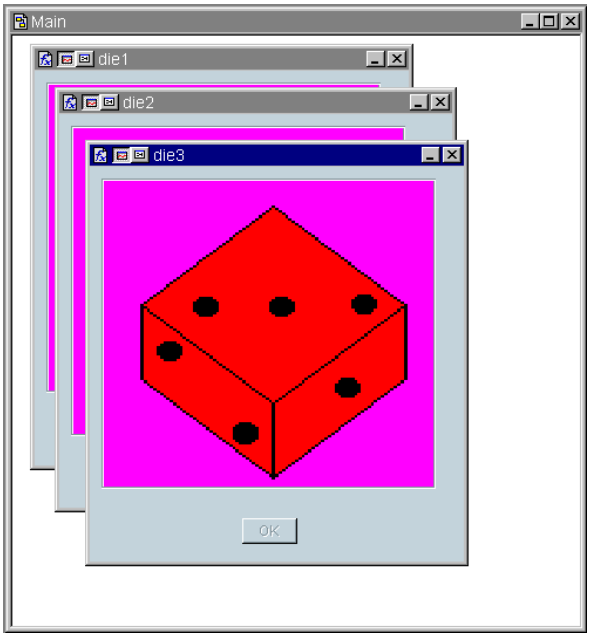


Figure 10-20. Early Stage in the Dice Program

Create a menu to select one of these three functions to call.

7. Select `Data` ⇒ `Selection Control` ⇒ `Radio Buttons`.

   `Radio Buttons` is an object that outputs an enumerated value (the `Enum` data type: a text string with an ordinal number associated to it) from a user-defined list on its upper output pin. For example, if you define the list as Monday, Tuesday, Wednesday, Thursday, and Friday, the operator could select the day from a menu, and `Radio Buttons` would then output the day.

   The first item in the list is assigned the ordinal position `0`; the *n*th item in the list is assigned ordinal position *n*`-1`. For instance, Monday in the list above has an ordinal position of `0`, and Friday has an ordinal position of `4`. The ordinal position appears on the lower output pin. Read the `Help` entry in the object menu for a more detailed explanation.

8. Open the `Radio Buttons` object menu and select `Edit Enum Values...`.

   Type in the names of the functions `die1`, `die2`, and `die3` by pressing the **Tab** key between each entry *except* the last. Click `OK`.

Note

There are six menu formats for data selection control. `Radio Buttons` displays entries as buttons. The operator's selection is output in text format as an `Enum` data type. `Cyclic Button` cycles through the enumerated values one at a time as the operator clicks the button. `List` displays all of the enumerated values in a list with the selected item highlighted. `Drop-down list`, `Pop-up list`, and `Slider list` are the other three choices.

9. Open the `Radio Buttons` object menu, click `Properties`, then select `Auto Execute`. Change the title to the prompt: `Make a Selection:`

Set up a `Call` object so that the value the operator selects on the `Radio Buttons` object will now become the function name that the `Call Function` object calls.

10. Click `Device` ⇒ `Call`. Select `Add Terminal` ⇒ `Control Input`, then select `Function Name`, and click `OK`. The `Function Name` control pin accepts an `Enum` or `Text` value as input. Connect the `Radio Buttons` data output pin to the `Function Name` input terminal on the `Call Function` object. Connect the `Radio Buttons` sequence out pin to the sequence in pin of `Call Function`. Click on `die2` in `Make a Selection:` and notice that the `Call Function Name` changes to `die2`, as shown in Figure 10-21.

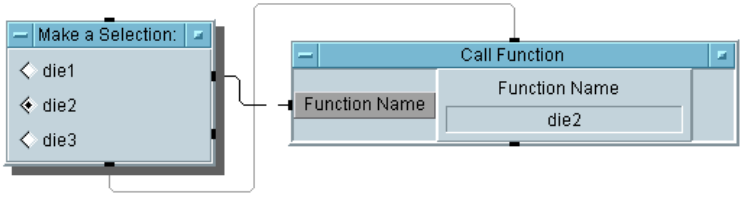


Figure 10-21. The Dice Program (Detail View)

Note The `Call` input terminal requires a `Text Scalar`, so VEE converts the `Enum Scalar` to a `Text Scalar`.

Remember the dotted line indicates a control pin. When `Auto Execute` is turned on, `Radio Buttons` executes whenever you make a change to it and sends the selection to `Call`. The control pin on `Call Function` replaces the function name as soon as the pin receives data. The `Call` object does *not* call the specified function until its sequence input pin is fired.

Note When a program uses `Auto Execute` and the sequence pins, the operator does not have to click the **Run** button to begin the program.

Add an operator interface showing only the prompt, the menu, and the pop-up panels showing the selections.

11. Select the `Radio Buttons` object by pressing **Ctrl** and clicking on the target object. Then select `Edit` ⇒ `Add To Panel`.

12. Open the object menu, select `Properties`, and adjust the colors and fonts if desired.

13. Run the program by making a selection. (Do not use the **Run** button, because it will use the selection that is already made on the menu.)

    The program should look like Figure 10-22 when executing.

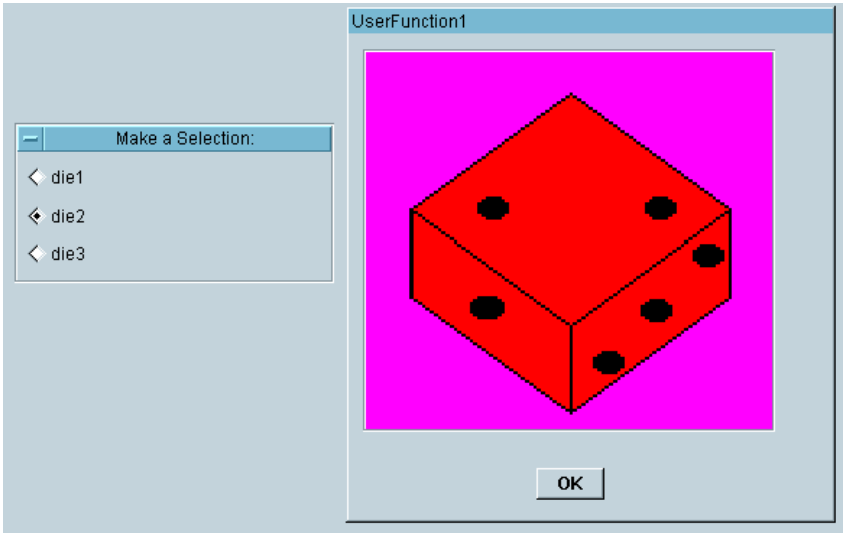


Figure 10-22. The Dice Program (Panel View)

There are a few things to note before the next lab exercise:

- You can use the same techniques in the exercise above to create menus for any program.
- `Radio Buttons` could also be used to select a compiled language program by using the `Execute Program` object with a control pin ("Command") that indicated the program to call. If you had imported a library of compiled functions, you could also use the `Call` object to run a function from the library.
- You could optimize this program by using the `File Name` data input pin on the `Picture` object inside a single `UserFunction`, and then sending the appropriate bitmap file to the object. If you are using many different bitmaps, this is a more efficient way to program.
- You will usually use **Run** instead of `AutoExecute` on more complicated programs. You can have the program pause at a data constant or selection control object by using `Wait for Input` instead of `AutoExecute`. See `Help` for more information.

Lab 10-2: Importing Bitmaps for Panel Backgrounds

Bitmaps are not essential to your programs, but they can add clarity and impact to tests. For example, you might want to import a schematic to better illustrate what is being tested. In this exercise, you will import bitmaps for panel backgrounds with standard VEE objects placed on top of them.

Bitmaps can be imported for icons, the `Picture` object, or for the panel view backgrounds in `UserObjects` or `UserFunctions`. You will create a pop-up `UserFunction` called `Bitmap` that includes a `Label` object and a `Confirm (OK)` object.

1. Select `Device` ⇒ `UserFunction`.

2. Select `Flow` ⇒ `Confirm (OK)` and `Display` ⇒ `Label`, and place them in the `UserFunction` window.

3. Change the name of the `UserFunction` to `Bitmap`.

4. Select the `OK` and the `Label` objects to highlight them with a shadow. Open the pop-up `Edit` menu by placing the pointer on the UserFunction work area and clicking on the right mouse button. Select `Add to Panel`.

5. Open the UserFunction menu, select `Properties`, then select `Show Panel on Execute`. (Remember to double-click on the title bar to get the `Properties` box.) Deselect `Show Title Bar` under `Pop-up Panel`.

   Open the `Panel` folder, change the `Grid Size` to `2`, select `default.gif` and `Scaled` under `Background Picture`, then click `OK`.

6. Open the `Properties` box for the `Label` object, and set as follows:

General/Title:	Change to `Bitmap Function`.
Label Justification	Change to `Center Justify`.
Colors/Object/ Background	Select `Light Gray` and `click OK`.
Fonts/Object/Text:	Choose a larger font with bold type, and click `OK`. Check `Automatically Resize Object on Font Change`.
Appearance/Border	Click on `Raised`. Click `OK` to make the changes and close the `Properties` dialog.

7. Position the title `Bitmap Function` and the `OK` button as desired. Iconize the `Bitmap` UserFunction.

8. Go to the Main window. Click `Device` ⇒ `Call`, then click `Select Function` in the object menu, and choose `Bitmap`. Run the program. The pop-up box should look like Figure 10-23.

Figure 10-23. The Bitmap Function

Lab 10-3: Creating a High Impact Warning

This exercise includes several `UserFunctions` that are nested. The first `UserFunction` is the alarm itself, which displays a red square and beeps. The second `UserFunction` calls the alarm repeatedly creating a blinking light effect and a pulsing sound, until the operator turns the alarm off.

Begin by programming the alarm function.

1. Select `Device` ⇒ `UserFunction`. Change the name to `alarm`.

2. Select `Display` ⇒ `Beep` and place it in the upper-left of the `UserFunction`. Adjust the settings so there is a loud beep that lasts a second. Change the `Duration (sec)` field to `1`. Change the `Volume (0-100)` field to `100`.

Note

These instructions assume your computer has the hardware to support a beep. Some Windows 95, Windows 98, Windows 2000, and Windows NT 4.0 systems have modified the `Default System configuration` for the `Default System Beep`.

Note

You do not need to connect the `Beep` object to anything. It activates when the function executes.

3. Click `Display` ⇒ `Indicator` ⇒ `Color Alarm` and place it in the `UserFunction`. Open the `Color Alarm` object menu, click `Properties`, and set as follows: under `Open View`, deselect `Show Title Bar`. Under `Layout`, click `Rectangular`. Under `Limits/High Test`, delete any text beside `High Text`. Click `OK`.

4. Click `Data` ⇒ `Constant` ⇒ `Real64`, change it to `1`, and connect it to the `Color Alarm` input pin. (This will always set the `Alarm` to its high range with the default color of red.)

   To keep the display on screen for one second to synchronize with the Beep object, use a `Delay` object set to `1` second.

5. Select `Flow` ⇒ `Delay`, set it to `1`, and connect its sequence input pin to the `Color Alarm` sequence out pin. The alarm will then last one second.

6. Select `Display` ⇒ `Note Pad` and add the message: `TURN OFF INSTRUMENTS!`. Size the `Note Pad` as needed.

7. Go to the `Main` window. Click `Device` ⇒ `Call`, choose `Select Function` from the `Call` object menu, and select `alarm`. Run the program to test it. The detail view of the `UserFunction alarm` should look like Figure 10-24.

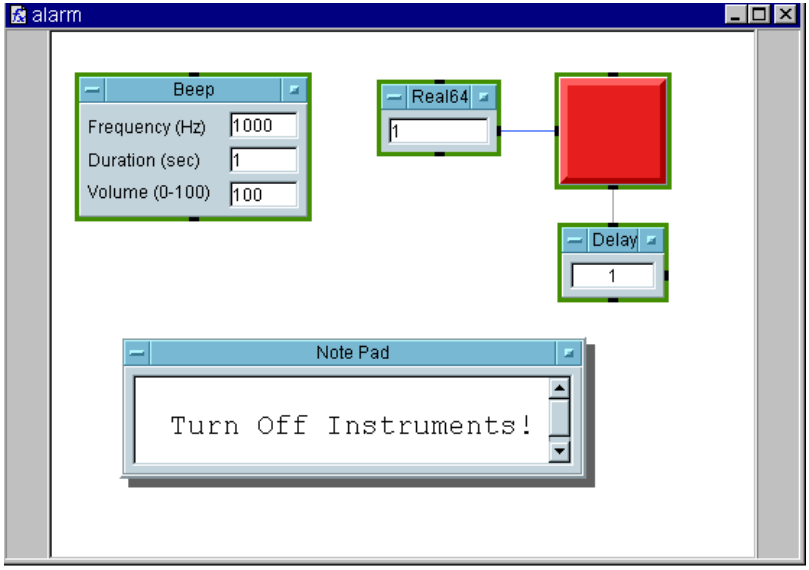


Figure 10-24. The UserFunction alarm (Detail View)

8. Return to the alarm window. Select the `Color Alarm` display and the `Note Pad`. Open the pop-up `Edit` menu and select `Add To Panel`. In panel view, size and arrange the objects. Open the `Note Pad` object menu and click `Properties`. Set as follows:

Open View/ Show Title Bar	Deselect.

Editing/Enabled	Deselect.
Fonts/Text size	Enlarge text size and `Font Style: Bold`.
Fonts	Select `Automatically Resize Object on Font Change`.
Appearance/Border	Change to a `Raised border`.

Click `OK` to close the `Properties` dialog box.

9. Change the `Color Alarm` to a `Raised Border` as well.

10. Double-click on the UserFunction title bar to get the `Properties` dialog box, and select `Show Panel on Execute`. Deselect `Show Title Bar`. Change the `Panel Title` to `alarm`. Iconize `alarm`.

11. Go to the Main window and delete the `Call` object. (VEE will still hold the function alarm in memory. If you want to edit the function again, select `Edit` ⇒ `Edit UserFunction` or double-click on its icon.)

Create the function that repeatedly calls the alarm function.

12. Select `Device` ⇒ `UserFunction` and change the name of the `UserFunction` to `warning`.

13. Select `Flow` ⇒ `Repeat` ⇒ `Until Break`.

14. Select `Device` ⇒ `Call`, change the `Function Name` to `alarm`, and connect its sequence input pin to the `Until Break` data output pin.

Add a `Check Box` object to ask the operator if he or she wants to turn off the alarm.

15. Select `Data` ⇒ `Toggle Control` ⇒ `Check Box`. Open the `Check Box Properties` box, change the name to `Turn off alarm?`, select `Scaled` under `Layout`, select `Initialize at PreRun` and make sure the value is `0`, make the font size bigger for the name, then click `OK`. Connect the `Call` sequence out pin to the `Check Box` sequence in pin.

This creates an input object that uses a `Check Box`. If the operator clicks the box, an `X` will appear and the object outputs a `1`; otherwise, the object outputs a `0`. The output can be tested with an `If/Then/Else` object to tell VEE what to do next.

16. Select `Flow` ⇒ `If/Then/Else` and place it to the right of the `Toggle`. Connect the `Toggle` data output to the data input `A` of the `If/Then/Else` object. Edit the expression in the `If/Then/Else` object to: `a == 1`. (Recall that the symbol for *"is equal to"* is `==`, not `=`.) If the terminal `A` holds a `1`, the `Then` output will fire; otherwise, the `Else` output fires.

    Connect the output of `Toggle` to the input of `If/Then/Else`.

17. Select `Flow` ⇒ `Repeat` ⇒ `Break` and connect it to the `Then` output on the `If/Then/Else` object, as shown in Figure 10-25.

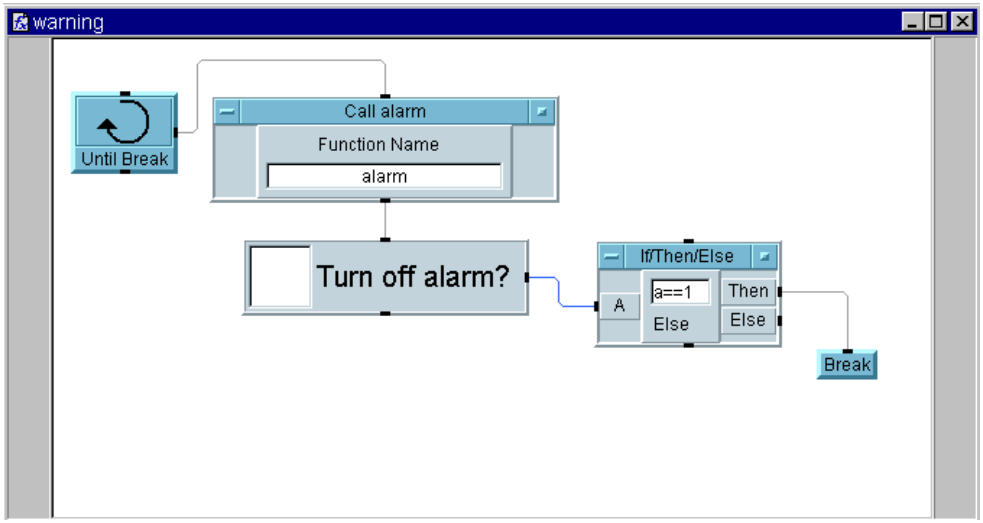


Figure 10-25. The Warning UserFunction (Detail View)

18. Select the `Check Box` object `(Turn off alarm?)` by clicking on the right side of the object. Open the pop-up `Edit` menu and select `Add To Panel`. Size the panel view to surround the `Check Box`.

19. Open the `warning UserFunction Properties` box, select `Show Panel on Execute`, deselect `Show Title` (since the title serves no purpose to the operator). Click `OK`.

20. Go to Main and click `Device` ⇒ `Call`, open its object menu, click `Select Function`, then select `warning`. Move the `Call` object to the top center of the screen. Iconize the Main window.

21. By default, VEE displays both the alarm and the warning panels in the center of the screen, so the alarm will blink on top of the check box that will stop the alarm. Since both of these screen positions are not locked, you can reposition them on the screen by clicking and dragging the pop-up panels to new locations. However, with the alarm panel blinking this is somewhat difficult. Instead, click and drag the edge of the panel. If needed, stop the program using the stop button on the tool bar. Run the program.

When you have the two panels positioned as shown in Figure 10-26, you can stop the program by clicking the box next to the `Turn off alarm?` prompt.

Figure 10-26. The Warning Program

Lab 10-4: Using an ActiveX Control

This lab shows how to use an ActiveX Control within VEE. You can incorporate ActiveX Controls from other applications into VEE programs. In this case, you will incorporate a `ProgressBar` control and use a loop to show the progress bar 0% to 100% complete. The same general principles apply to other ActiveX Controls.

1. Click `Device` ⇒ `ActiveX Control References...` and select `Microsoft Windows Common Controls 6.0`. Click `OK`.

2. Click `Device` ⇒ `ActiveX Controls` ⇒ `ProgressBar`. Size the `ProgressBar` object to be larger. Open its Object Menu and notice that the object name is `ProgressBar`. VEE has automatically created a declared variable that refers to the ActiveX control object. You can use the name `ProgressBar` in `Formula` expressions, just like any other variable or data input.

3. Click `Device` ⇒ `Formula & Object Browser`, and select `ActiveX Objects`, `Library: MSComcltLib`, `Class: ProgressBar`, `Members: Value`, and click `Create Set Formula`. Place the object at top center in the `Main` window.

4. To loop from zero to one hundred and show the percent complete, you will add a `For Range` object. Select `Flow` ⇒ `Repeat` ⇒ `For Range`, place the object below the `ProgressBar`, and set it as follows: `From: 0`, `Thru: 100`, and `Step:10`. Connect the `For Range` output to the `ProgressBar` input terminal `Value`.

5. To slow down program execution so that you can see the `ProgressBar` updating, select `Flow` ⇒ `Delay` and place the object to the right of the `For Range` object. Set it to `.2`. Connect the `ProgressBar` sequence output pin to the `Delay` object sequence input pin, as shown in Figure 10-27, and run the program.

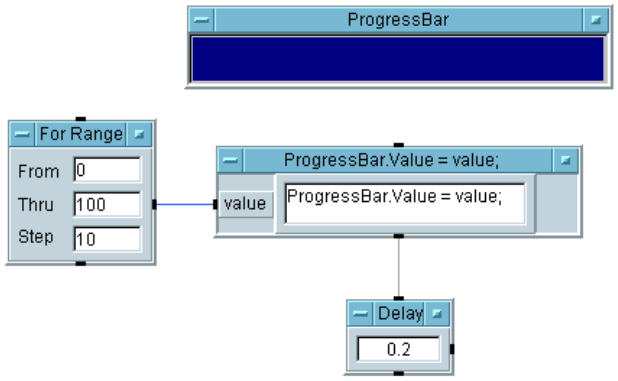


Figure 10-27. Using the ActiveX Control "ProgressBar"

Note

ActiveX Control `object menus` have both `Properties` and `Control Properties`. The `Properties` menu sets the VEE properties of the object. The `Control Properties` are supplied by the Control and can be different for each type of ActiveX Control.

Examine all the control examples that ship with VEE to get a better understanding of how they work. Then look for other controls and displays in the marketplace you might want to add to enhance the user interface capabilities in VEE.

Figure 10-28 shows another example of VEE incorporating a control from MS Chart. After you have selected a control library in the `Device` ⇒ `ActiveX Controls References` dialog box, you can use the `Function & Object Browser` or the `Declare Variable Object` to identify a control's properties and methods.

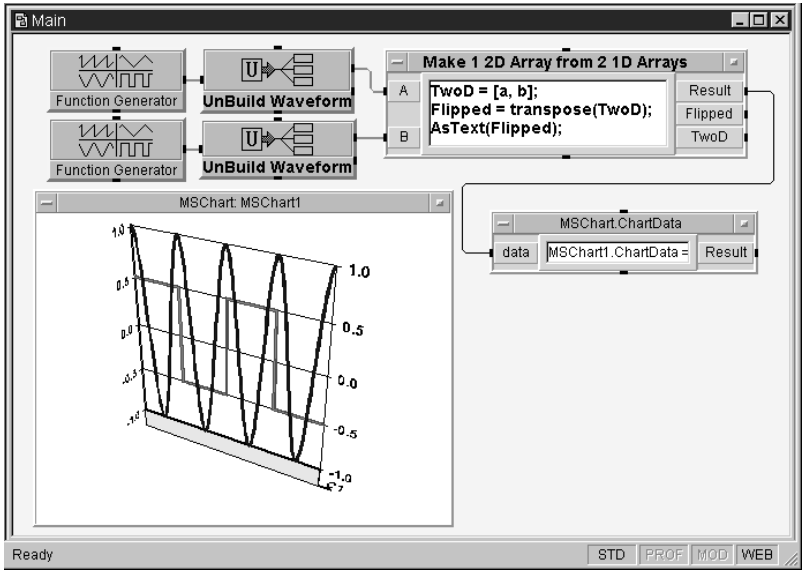


Figure 10-28. An ActiveX Control Example Using MSChart

Lab 10-5: Creating a Status Panel

In this lab, you will learn how to use the functions from the dice program to create a status panel. (The dice program exercise is in “Using Menus” on page 383.) Typically this would be used with the `Sequencer` object when there are a number of tests and you want to see the results as they are returned. You will use the function `random()`, which returns a real value between `0` and `1`, when using the default settings.

1. Click `Device` ⇒ `Sequencer`. Double-click the transaction bar, and configure your test using the default name, `test1`, and replacing the `FUNCTION:` field with `random()`. See Figure 10-29.

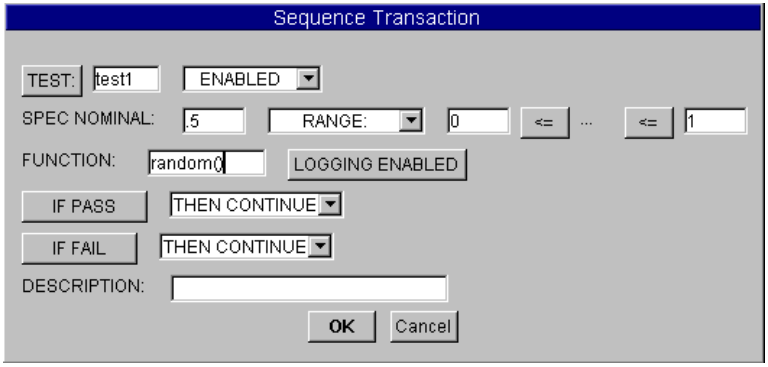


Figure 10-29. Configuring Test1

2. Configure a second test the same way named `test2`.

3. Open the `Sequencer Properties` box, choose the `Logging` tab, and under `Logging Mode` select `Log Each Transaction To: logTest(thisTest)`, then click `OK`.

   As the `Sequencer` executes each transaction, it creates a record for each test called "`thisTest`," whose fields can be configured under the same tab. You can then create a UserFunction called "`logTest`" (or another name) and the `Sequencer` will call the `LogTest() UserFunction` with the `Record thisTest` at the end of each transaction that is executed. In this way, you can update the status panel.

4. Select `Device` ⇒ `Function & Object Browser` ⇒ `Built-in Functions` ⇒ `Panel` ⇒ `showPanel` and place it above the `Sequencer`. Delete the input pins, then edit the parameters to `"logTest",420,180` leaving out the last two parameters. Connect the `Result` output pin from `showPanel` to the `Sequencer` sequence input pin. `ShowPanel` outputs a `1` if it succeeded.

   `LogTest` is the name of the `UserFunction`. The other two parameters are `X` and `Y` coordinates on your screen starting from the upper left corner. This tells VEE where to place the `UserFunction` panel when it is shown. (The panel dimensions are not included in this example.)

5. Create the `UserFunction` named `logTest`, as shown below. Add an input pin. (The logging `Record` will be the input.) Put the `Logging AlphaNumeric` on the panel and connect it to the input pin. Select `Logging AlphaNumeric` and click `Edit` ⇒ `Add to Panel`. In the panel view, adjust the sizing and placement as desired. Deselect the display and panel title bars.

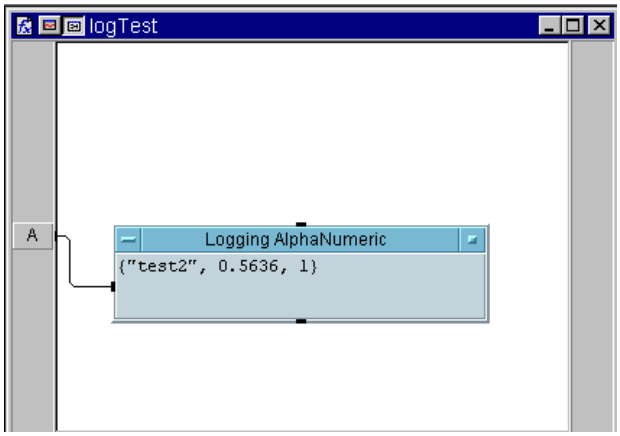


Figure 10-30. The UserFunction LogTest (Detail)

Figure 10-31 shows the panel view after running the program. It will give you an idea of how the data will be displayed.

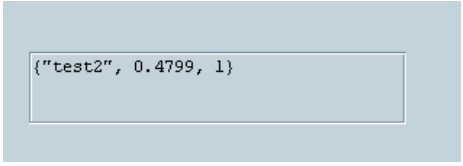


Figure 10-31. The UserFunction LogTest (Panel)

6. Select `hidePanel` from the `Function & Object Browser` box and `Flow` ⇒ `Confirm(OK)`. Change the `hidePanel()` parameter to `logTest`. Delete the input pin. Connect the objects as shown in Figure 10-32.

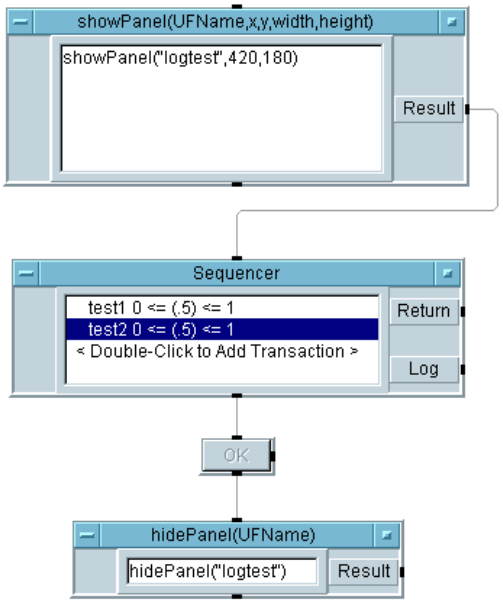


Figure 10-32. Status Panel Program (before running)

7. Run the program. It should look like Figure 10-33.

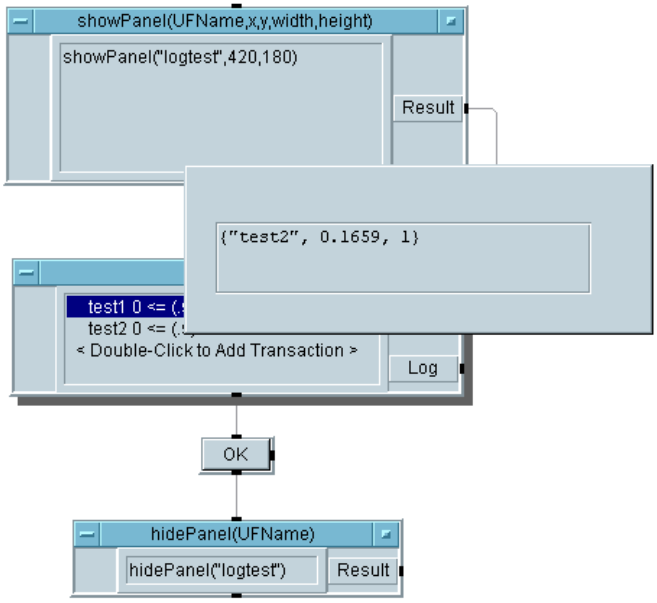


Figure 10-33. The Status Panel Program (running)

In summary, the `showPanel` object displays the `UserFunction` panel, but does not call the `UserFunction`. The `Sequencer` calls the `UserFunction` twice through its logging function, and each call updates the panel. Then, when the operator is done, he or she can press `OK` and the panel is hidden. This example uses an `OK` object to trigger the `hidePanel` object, but you could put it elsewhere in the program to time its execution. Step through the program to see the status panel appear, update with the results from `Test1` and `Test2`, and then disappear.

Chapter Checklist

You should now be able to perform the following tasks. Review, if necessary, before going on to the next chapter.

- Summarize the key points concerning operator interfaces.
- Use a menu to select tests on an operator interface.
- Import a bitmap for an operator interface.
- Create a status panel.
- List some of the operator interface features that VEE provides.
- Secure a program.
- Create a high impact warning.
- Create an ActiveX Control and find its properties and methods in the `Function & Object Browser`.

11

Optimizing Agilent VEE Programs

Optimizing Agilent VEE Programs

In this chapter you will learn about:

- Basic techniques for optimizing programs
- Using Dynamic Link Libraries (DLLs) on a PC
- Optimizing with compiled functions
- Using compiled functions in other languages on UNIX platforms
- Using the VEE Compiler
- Using the VEE Profiler

Average Time To Complete: 2 hours

Overview

In this chapter, you will learn how to improve the execution speed of VEE programs. There are three basic components in test program performance: the speed of taking the measurement, the rate at which the data is transferred to the computer, and the speed at which the program processes the data. By optimizing the VEE program, you can increase its processing speed.

In the first section, you will learn the basic principles for optimizing a VEE program. You will also learn about Dynamic Link Libraries (DLLs) on the PC. The next section describes how to optimize using compiled functions. You will also learn about how to optimize parts of programs by linking compiled functions in other languages on UNIX platforms. Then, there is an overview of the VEE compiler. Finally, there is a section on the VEE profiler.

Note The techniques in this chapter apply whether or not you use the compiler.

Basic Techniques for Optimizing Programs

To optimize VEE programs, read the information in this section. You can use the techniques described here to develop good programming habits in VEE.

Perform Math on Arrays Whenever Possible

Performing mathematical operations on arrays greatly improves program performance. For example, suppose a test must find the square root of measurements being taken. The traditional way to program this would be to take a measurement and calculate the square root in a loop. Instead, in VEE, you can store all the measurements in an array and calculate the square root of the array in one step.

In Figure 11-1, the program iterates 1024 times. Each iteration calculates a square root.

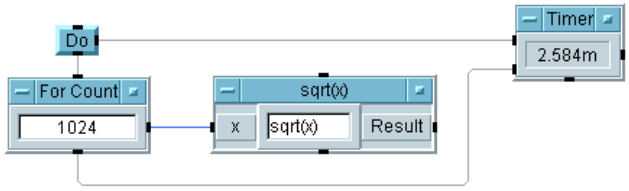


Figure 11-1. Calculating Square Roots per Measurement

In Figure 11-2, the program creates an array of 1024 elements and calculates the square root of the array (yielding an array of square roots). Although the two programs both yield the same results, the program in Figure 11-2 executes about 6 times faster than the one in Figure 11-1. (This example uses a 300 MHz HP Pavilion PC.)

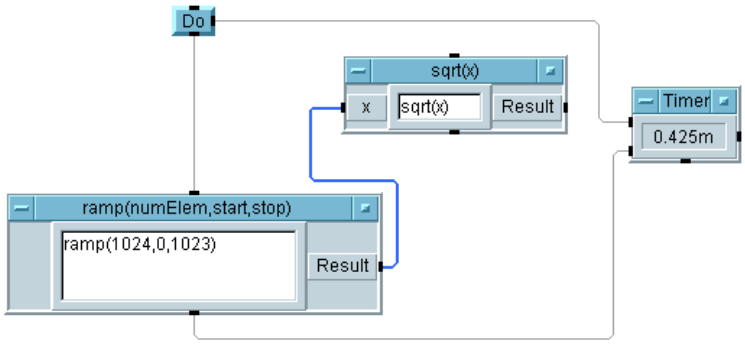


Figure 11-2. Calculating Square Roots using Math Array

The difference in the execution speeds of the two programs is due to the time required for an object to execute. There is a fixed amount of overhead when an object executes. Therefore, when you reduce the number of times an object executes by using arrays rather than scalar values, the program runs faster.

Using the `Do` object is a good idea when timing to make sure the timer triggers first in both programs. The `ramp` function generates an array with 1024 elements starting at 0 and ending at 1023.

Note

To ensure faster execution, always make sure you are using the most recent execution mode in VEE. To do this, click `File` ⇒ `Default Preferences` (or use the button on the tool bar). Select `VEE6` under `Execution Mode` and click `Save`. In the status bar at the bottom of the VEE window, you should see `VEE6` listed.

Make Objects into Icons Whenever Possible

The more information VEE has to maintain on the screen, the more time it will take the program to run. To optimize the program, use iconic views for objects that update their contents, such as the `Counter`, instead of using open views. The example in Figure 11-3 operates about 46 times faster using an iconic view for the `For Count` and `Counter` object.

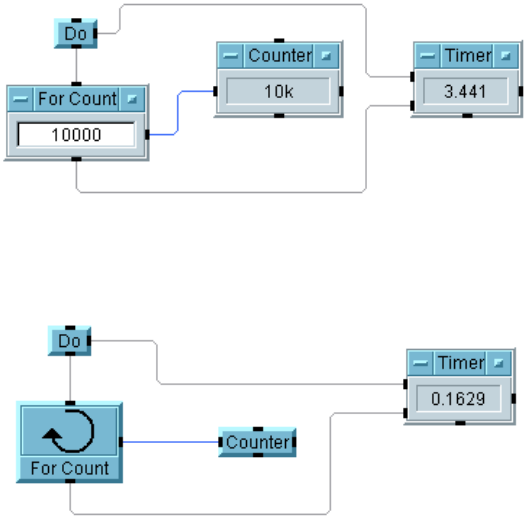


Figure 11-3. Optimizing Programs by Using Icons

Reduce the Number of Objects in Programs

As you become more experienced, you will tend to use less objects in VEE programs. There are two more techniques to reduce the number of objects and therefore optimize programs:

1. Use a single equation in a `Formula` object instead of using separate mathematical objects. For example, put the equation `((a + b) * c)/d` into a `Formula` object instead of using separate objects for addition, multiplication, and division. Also, use constants in the formula instead of constant objects connected to inputs. (Set constants with `Set Variable`.)

2. Nest function calls within other function parameter lists. For instance, in Figure 11-4, the function randomize uses the array generated by the function ramp. In Figure 11-5, the function call to `ramp` is nested in the call to `randomize`, resulting in slightly faster program execution.

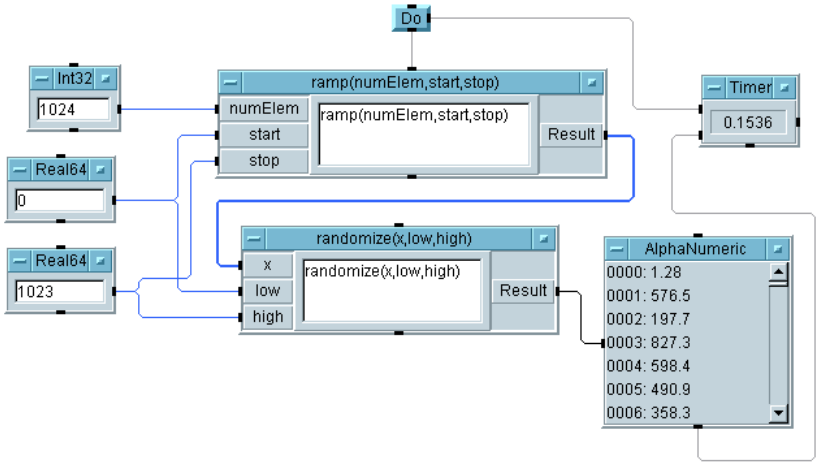


Figure 11-4. Function Calls without Optimization

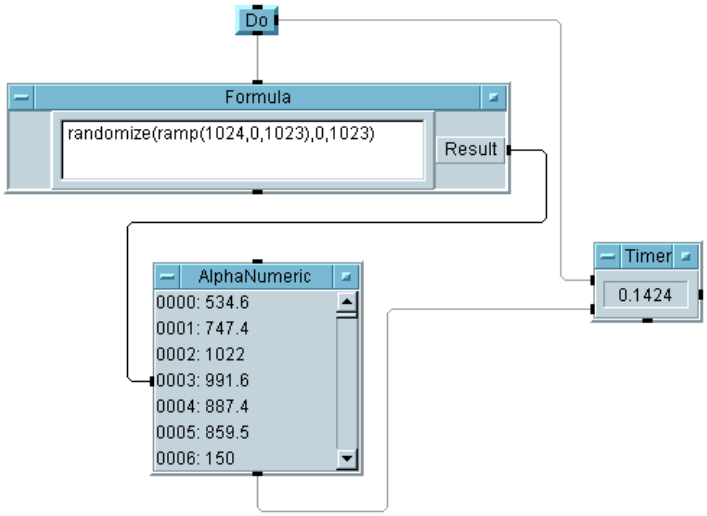


Figure 11-5. Function Calls with Optimization

Other Ways to Optimize Agilent VEE Programs

There are other optimization techniques that you can use in programs when appropriate, as follows:

- Make sure you are using the VEE compiler by running your programs in VEE 4 or higher Execution Mode. For more information, refer to "Agilent VEE Execution Modes" on page 427.

- Run the program from the panel view instead of the detailed view. VEE will have less objects to maintain on the screen.

- Use global variables rather than pass values (especially large arrays or records) into and out of `UserObjects` and `UserFunctions`. Declare all the global variables. This also allows you to use local variables. See `Data ⇒ Variable ⇒ Declare Variable`.

- Collect data for graphical displays and plot the entire array at once rather than plotting each individual scalar point. If the `X` values of a plot are regularly spaced, use an `XY Trace` display rather than an `X` vs. `Y` Plot.

- Use one `If/Then/Else` object with multiple conditions instead of multiple `If/Then/Else` objects.

- Set graphical displays to be as plain as possible. The settings that allow the fastest update times are `Grid Type ⇒ None` and nothing checked in the `Properties` dialog box. Only use `AutoScale` control pins where necessary, and turn off the `Automatic AutoScaling` if not needed (in the `Scales` folder).

- When reading data from a file, use the `ARRAY 1D TO END: (*)` transaction instead of performing `READ` transactions on one element at a time and using the `EOF` pin.

- Use the `Sequencer` to control the flow of execution of several `UserFunctions` instead of separate `Call` objects.

- When using the `Sequencer`, only enable logging for transactions where the `Log` record is required.

- When using `Strip Charts` and `Logging AlphaNumeric` displays, set the `Buffer Size` in `Properties` to the smallest number possible for your application.
- Use the triadic operator, *(condition ? expression1 : expression2)*, instead of the `If/Then/Else` object with `Gates` and a `Junction`.
- When using bitmaps, set them to `Actual` or `Centered` rather than `Scaled`, since `Scaled` will take a few moments longer.
- When using indicators such as the `Fill Bar` or `Thermometer`, turn off `Show Digital Display`.
- When using `Color Alarms`, if you are switching between colors rapidly, turn off `Show 3D Border`.

In addition to the techniques already mentioned, linking compiled functions in other languages to your VEE programs can increase execution speed. Using compiled functions on PCs (as DLLs) are described in the next section. Using compiled functions on UNIX platforms is described in the section “Compiled Function using C (UNIX)” on page 424.

Overview of Compiled Functions

You can use a compiled function in a VEE program, such as a DLL (Dynamic Link Library). To do so, you must obtain the compiled function or follow these steps to create it:

1. Write functions in C, C++, Fortran, or Pascal and compile them.
2. Write a definition file for the functions.
3. Create a shared library containing the compiled functions.

Benefits of Using Compiled Functions

Using compiled functions in a VEE program offers the following benefits:

- Faster execution speed
- Leveraging current test programs in other languages
- Developing data filters in other languages and integrating them into VEE programs
- Securing proprietary routines

Note

Adding compiled functions adds complexity to the development process. Therefore, use a compiled function *only* when the capability or performance that you need is not available with one of the following: a VEE `UserFunction`, an `Execute Program` escape to the operating system, or an `ActiveX Automation` call to another program.

Design Considerations in Using Compiled Functions

If you plan to use compiled functions in a VEE program, take the following information into consideration:

- You can use any facilities available to the operating system including math routines, instrument I/O, and so forth. However, you cannot access any VEE internals from within the program to be linked.
- You need to provide error checking within your compiled function, since VEE cannot trap errors in an external routine.
- You must de-allocate any memory you allocated in your external routine.
- When passing data to an external routine, make sure you configure the `Call` object input terminals to the type and shape of data that the routine requires.
- System I/O resources may become locked, so your external routine should be able to handle this type of event.
- If your external routine accepts arrays, it must have a valid pointer for the type of data it will examine. Also, the routine must check the size of the array. If the routine changes the size, you need to pass the new size back to the VEE program.
- The compiled function must use the `return()` statement as its last statement, not `exit()`. If the compiled function exits, then so will VEE, since a compiled function is linked to VEE.
- If you overwrite the bounds of an array, the result depends on the language you are using. In Pascal, which performs bounds checking, a run-time error will result, stopping VEE. In languages like C, where there is no bounds checking, the result will be unpredictable, but may cause intermittent data corruption or cause VEE to crash.

Guidelines in Using Compiled Functions

When you use compiled functions in a VEE program, follow these guidelines:

- Call and configure a `Compiled Function` just as you would call a `UserFunction`. You can either select the desired function using `Select Function` from the `Call` object menu, or you can type in the name. In either case, provided VEE recognizes the function, the input and output terminals of the `Call Function` object are configured automatically. The necessary information is supplied by the definition file. (VEE will recognize it if the library has already been imported.)

- Reconfigure the `Call` input and output terminals by selecting `Configure Pinout` in the object menu. For either method, VEE configures the `Call` object with the input terminals required by the function, and with a `Ret Value` output terminal for the return value of the function. In addition, there will be an output terminal corresponding to each input that is passed by reference.

- Call the `Compiled Function` by name from an expression in a `Formula` object or from other expressions evaluated at run time. For example, you could call a `Compiled Function` by including its name in an expression in a `To File` transaction.

Note

Only the `Compiled Function`'s return value (`Ret Value` in the `Call` object) can be obtained from within an expression. If you want to obtain other parameters returned from the function, you will have to use the `Call` object.

- Delete a library of `Compiled Functions` by using the `Delete Library` object in the `Device` menu. Using the `Import Library`, `Call`, and `Delete Library` objects, you can shorten the program load time and conserve memory by importing and deleting them when the program has finished calling them.

Using Dynamic Link Libraries

On PCs, you can use the compiled functions from Dynamic Link Libraries (DLLs) as a part of a VEE program. DLLs may be compiled functions that you have written yourself (contact Microsoft for documentation about writing DLLs), or DLLs that you have purchased or downloaded from the Web.

Note

VEE supports both the "`_cdecl`" and "`_stdcall`" calling conventions. Most customer-written DLLs use the `_cdecl` calling convention. Most Win32 API calls use `_stdcall`. VEE supports both naming conventions, so you can use most off-the-shelf DLLs as well as your own.

Integrating a DLL into an Agilent VEE Program

This section describes how to import a DLL into a VEE program. Write or obtain the DLL as described above, then follow these steps to use the DLL:

1. Select `Device` ⇒ `Import Library`.

   The `Library Type` includes three choices: `UserFunction`, `Compiled Function`, and `Remote Function`. Change the `Library Type` field to `Compiled Function` (the default is `UserFunction`). For a `Compiled Function`, the `Import Library` object includes a field for the `Definition File`, as shown in Figure 11-6.

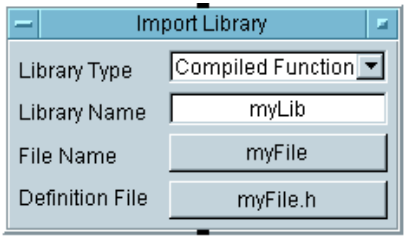


Figure 11-6. Importing a Library of Compiled Functions

The fields are described as follows:

Library Name	The name VEE uses to identify the library. Generally, this is used if you want to delete the library after it has been used in the program.
File Name	File that holds the shared library.
Definition File	The include file with the prototypes of the functions. This is usually a `*.h` file.

Note

You can also load a library manually during the development phase by selecting `Load Lib` from the object menu.

2. Select `Device` ⇒ `Call`.

When you have imported the library with `Import Library`, create a `Call` object by selecting `Device` ⇒ `Call`. You can then call the `Compiled Function` by choosing `Select Function` from the `Call` object menu, and choosing the desired function from the list box presented. For example, the `Call` object shown in Figure 11-7 calls the `Compiled Function` in `myLibrary` named `myFunction` with the parameters `arraySize` and `array`.

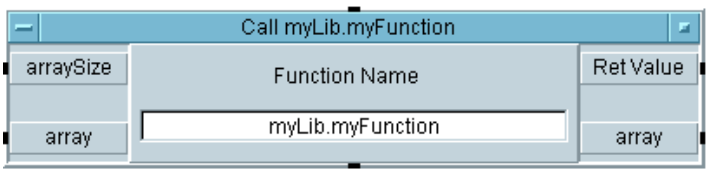


Figure 11-7. Using Call Object for Compiled Functions

VEE automatically configures the `Call` object with the function name, and the proper number of input and output pins. The second, third... output pins map to any parameters passed by reference to the function. If you have entered the function name, you can also configure the object by selecting `Configure Pinout` in the object menu.

Note

You can also call a DLL function from an expression field, provided the library has been loaded. When used in this way, you must enclose the parameters in parentheses after the function name, and the function only sends back its return value. Any parameters passed by reference can only be retrieved by using the `Call` object. For example, you might use the following expression in a `Formula` object:

```
2 * yourFunc(a,b)
```

The `a` and the `b` would refer to two input pins on the `Formula` object, and the return value of `yourFunc` would be multiplied by `2` and placed on the output pin.

3. (Optional) Click `Device` ⇒ `Delete Library`.

   While developing the program, you can also select `Delete Lib` from the object menu to delete the library programmatically. Deleting the library after it has been used in the program reduces load time and conserves memory.

An Example Using a DLL

In this exercise, you will import a DLL and call a function from the DLL. The DLL used is included with the VEE product on Windows. (The same program is designed to work on all platforms.)

Open the `manual49.vee` file. It is located under:

<*installation directory*>`\EXAMPLES\MANUAL\MANUAL49`.

Examine this example closely. It should look like Figure 11-8.

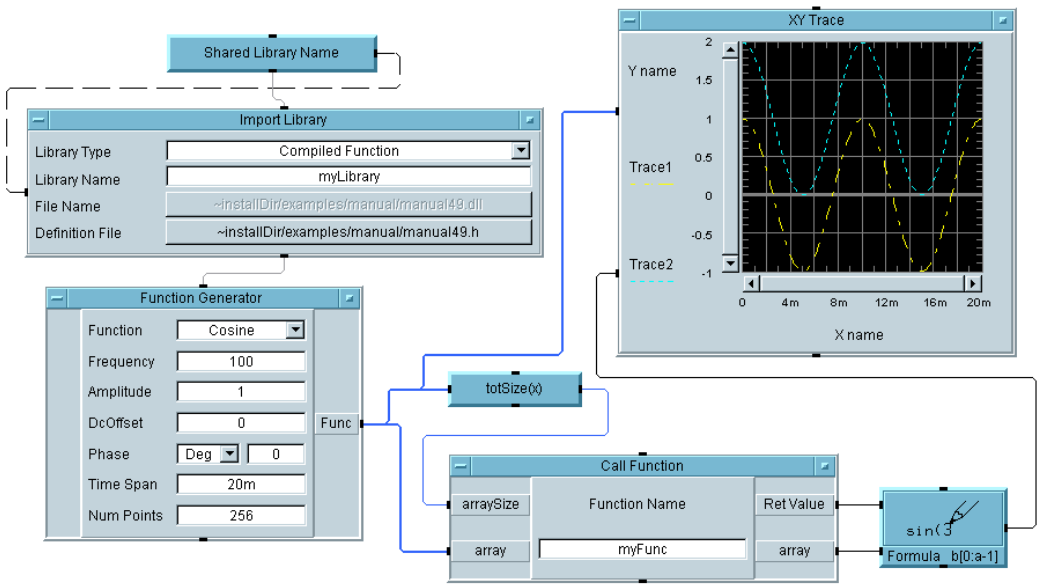


Figure 11-8. A Program Using a DLL (MANUAL49)

Import Library Before the first call to the compiled function `Call Function`, the DLL must be loaded using the `Import Library` object (in the `Device` menu).

Call Function `MANUAL49` calls a compiled function called `myFunc`. `MyFunc` requires a C datatype called `long`, which is the same as a VEE `Int32`. This number specifies the size of an array. The second input parameter is a pointer to an array of reals. The definition file is located in `MANUAL49.H`, and the source file for the C code is located in `MANUAL49.C`. `MyFunc` adds `1` to every element of the array.

Function Generator The `Function Generator` is used to create a waveform, which is output to the array pin on the `Call myFunc` object.

totSize The `totSize` object (in the `Math & Functions` box) is used to determine the size of the waveform, which is output to the `arraySize` input pin on `Call myFunc`.

XY Trace The `XY Trace` object displays both the original and the new waveforms

Formula The `Call` object output pin labeled `Ret Value` holds the size of the returned array, so that expression `B[0:A-1]` in the `Formula` object correctly specifies this array to the display object.

Run the program and notice that the second trace is one greater than the first trace at all points on the waveform.

Another key point to notice in the program is the method used for making it portable to all VEE platforms. The HP-UX platforms use shared libraries indicated by the filename extension `*.sl`. Windows 95, Windows 98, Windows 2000, and Windows NT 4.0 use a Microsoft 32-bit compiler. These DLLs are all indicated using a `*.dll` extension.

The `UserObject` called `Shared Library Name` identifies the operating system being used, and then transmits the correct library name to the `Import Library` object, as shown in Figure 11-9.

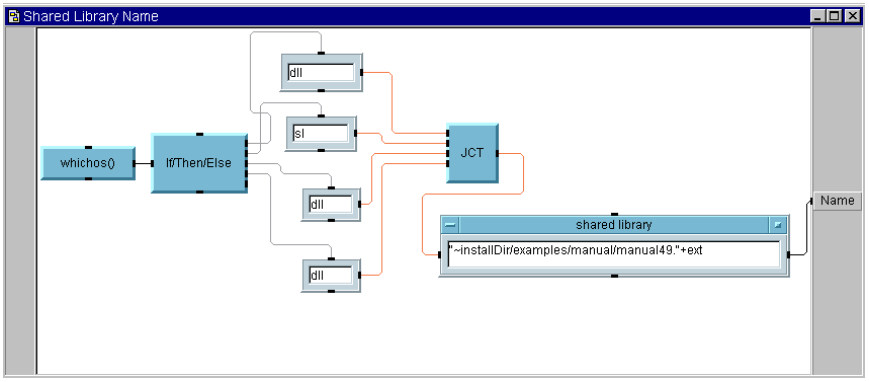


Figure 11-9. The Shared Library Name UserObject

The `whichos()` function has been used in a renamed `Formula` object to identify the operating system. An expanded `If/Then/Else` object examines the output of the `whichos()` function, then triggers the appropriate text constant. This filename extension is then added to the `MANUAL49` file using a renamed `Formula` object. (The input terminal on the `Formula` object labeled `shared library` has also been changed to `ext`.)

A control pin for a `File Name` has been added to the `Import Library` object; hence, there is a dotted line between the `UserObject` and the `Import Library`.

Note

Investigate `To/From Socket` for sharing data in mixed environments, such as sending data to a database. Also, the example games Battleship and Euchre make extensive use of sockets to communicate between multiple VEE processes.

Execute Program Object versus Compiled Functions

When you are deciding whether to use an `Execute Program` object or a compiled function to integrate your compiled language programs with VEE, consider the following information.

Execute Program Object

The `Execute Program` object has these characteristics:

- Easier to use
- Longer start-up time
- Communication through pipes (HP-UX only)
- Protected address space
- Choice of synchronous or asynchronous execution
- Service of asynchronous events
- Safer (If the program called crashes, you get an error message.)
- Better for continuous data acquisition

Compiled Functions

Compiled functions, using the `Import Library` and `Call` objects, have these characteristics:

- Short start-up time
- Communication by passing on stack and memory space shared with VEE
- Synchronous execution
- Signals and exceptions not blocked or caught (such as GPF messages)
- Compiler for textual language required
- More complicated to use. More risk in using. (An out-of-bounds array error or overwriting memory will cause VEE to crash.)

Compiled Function using C (UNIX)

The process of using compiled functions in other languages involves shared libraries on HP-UX platforms. You can dynamically link a program written in C, C++, Fortran, and Pascal with a VEE program on a UNIX workstation. Note that Pascal compiled functions are only supported on the HP 9000, Series 700 workstations.

The program shown in Figure 11-10 imports a library and calls a C function. The C function accepts a real array and adds 1 to each element in the array. The modified array is returned to VEE on the `Array` terminal of the `Call Function` object, and the size of the array is returned on the `Ret Value` terminal. This example is located in the following VEE directory:

<*installation directory*>`/examples/manual/manual49.vee`.

Note

File extensions are as follows: a "`.vee`" extension indicates a program; a "`.c`" extension indicates a file containing C source code; a "`.h`" or "`.def`" extension signifies a definition file; and a "`.sl`" extension indicates a shared library file.

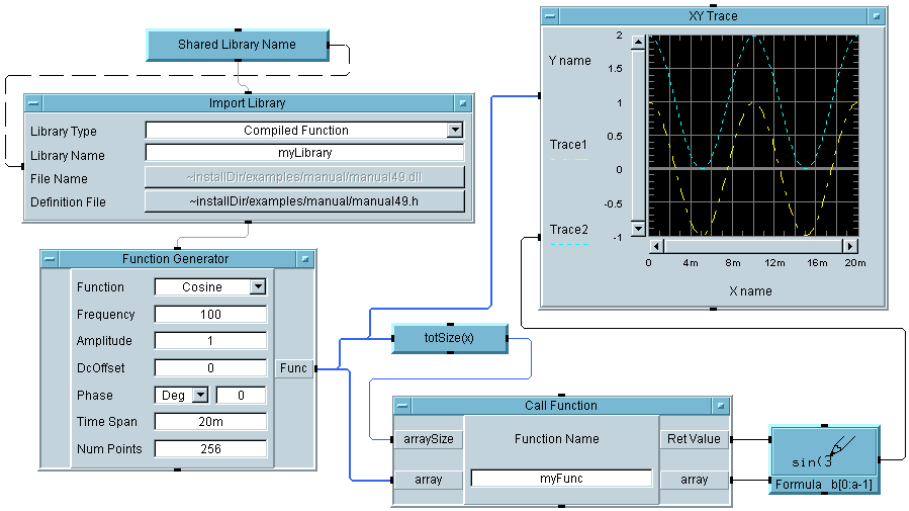


Figure 11-10. Program Calling a Compiled Function

Notice the following about the program:

size of the array

One variable in the C function (and correspondingly, one data input terminal in the `Call` object) is used to indicate the size of the array. The `arraySize` variable is used to prevent data from being written beyond the end of the array.

the Call Function object

Since array has been passed by reference, VEE automatically creates both an input and output pin on the `Call Function` object.

The `arraySize` variable has been passed by value, so VEE only creates an input terminal. However, the function's return value is used to return the size of the output array to VEE. This technique is useful when you need to return an array that has fewer elements than the input array.

the C routine	The C routine is a function, not a procedure. The `Compiled Function` requires a return value, so if you use a language that distinguishes between procedures and functions, make sure you write the routine as a function.
order of execution	The `Import Library` object executes before the `Call` object in the program. If you have any doubts about the order of execution regarding these two objects, use the sequence pins to assure the right order.
passing variables	The parameter variable array passed by reference to the function has both input and output terminals, but the variable `arraySize` passed by value has only an input terminal.
number of array elements	The `Formula` object uses the size of the array in the `Ret Value` terminal to send the correct number of array elements to the display.
XY Trace	The `XY Trace` automatically autoscales the two waveforms.

Agilent VEE Execution Modes

Agilent VEE Execution Modes allow you to run programs that were created using previous versions of VEE. The Execution Modes allow a newer version of VEE to run programs created with an older version of VEE in exactly the same way the older VEE version ran them. This makes VEE backward compatible to support your existing programs.

Note

Execution Mode was known as *compatibility mode* in previous versions of VEE.

VEE has four execution modes:

- VEE 6 (adds new data types)
- VEE 5 (adds ActiveX)
- VEE 4 (compiled)
- VEE 3.*x*

The execution mode of the program you are running is displayed in the status bar of VEE, as shown in Figure 11-11.

Existing programs that are opened in VEE will run by default in the Execution Mode for the VEE version in which they were created. For example, a VEE 5.0 program opened in VEE 6.0 will run in VEE 5 Execution Mode by default.

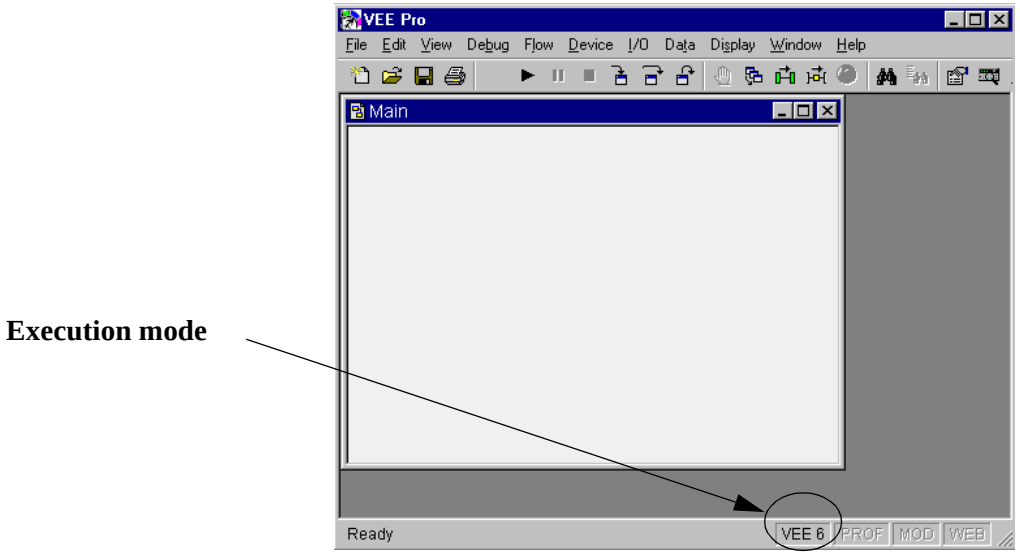


Figure 11-11. Execution Mode Display in VEE Status Bar

The Agilent VEE Compiler

The VEE compiler is automatically enabled in VEE 4 and higher Execution Modes. The compiler provides much faster program execution, as well as more predictable object propagation. For more information about the compiler and details of the differences between the Execution Modes, refer to the *VEE Pro Advanced Techniques* manual.

Changing the Execution Mode

You should create all new programs in VEE 6 mode. If you have existing programs, you will want to change the execution mode if you add any new features to an existing program. For example, if you have a program written in VEE 5.0 and you add a new feature from VEE 6.0, you should change the execution mode to VEE 6. Otherwise, the VEE 5.0 program may not run correctly.

To change the execution mode, follow these steps:

1. From the main VEE menu, click `File` ⇒ `Default Preferences`, or press the **Default Preferences** button on the tool bar as shown in Figure 11-12.

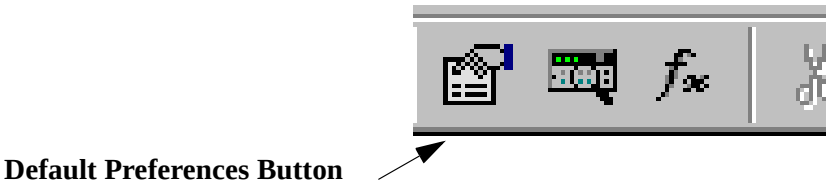


Figure 11-12. Default Preferences Button on Toolbar

2. In the `General` folder (already displayed, since it is the first folder), under `Execution Mode`, select VEE 6.0 as shown in Figure 11-13. In the same folder, make sure that `Disable Debug Features` is *not* selected. Click `OK`.

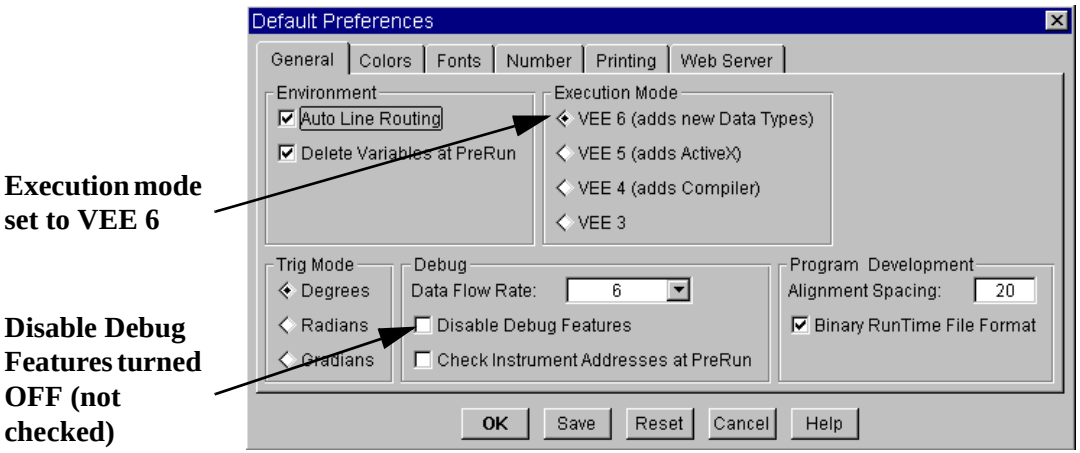


Figure 11-13. Changing the Execution Mode in Default Preferences

Effect of Changing the Execution Mode

The following example demonstrates the increase in speed when a program is updated. These example focuses on the program speed without instrument I/O.

1. Open the `chaos.vee` program in the `examples\Applications` subdirectory.

   This program illustrates explosive population growth. You can modify the program, as shown here, by using a `Timer` object to check the results. These examples were run using a 300MHz HP Pavillion PC on Windows 95 with two other large applications running concurrently.

In Figure 11-14, the program execution is timed with the displays open in VEE 3 execution mode.

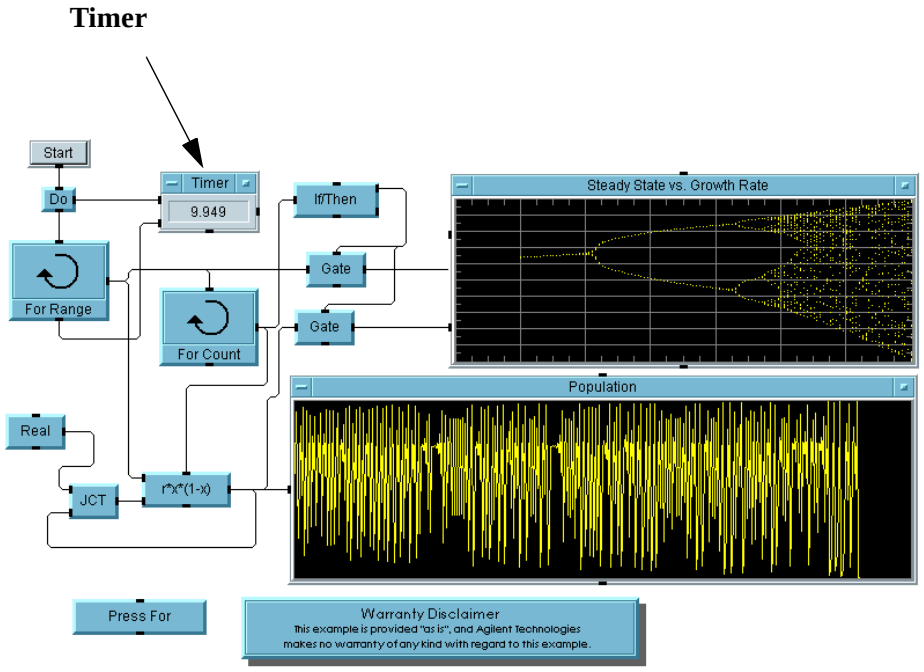


Figure 11-14. Chaos.vee in VEE 3 Mode with Open Displays

In Figure 11-15, the displays are iconized to improve speed without turning on the compiler. This cuts execution time about 1/6.

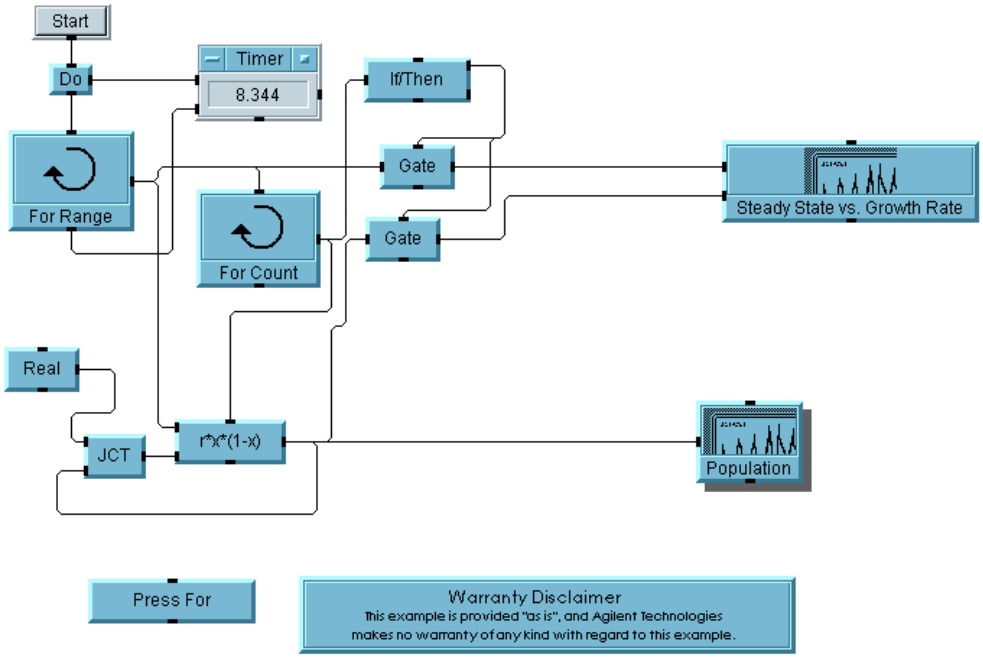


Figure 11-15. Chaos.vee in VEE 3 Mode with Closed Displays

Finally, in Figure 11-16, the compiler is turned on with the debugging features disabled. For optimum performance, check the `Disable Debug Features` box in `File` ⇒ `Default Preferences` when the program is fully debugged and ready to use.

The debugging features enable tools including the `Show Execution Flow` and `Activate Breakpoints`. When you check `Disable Debug Features`, this makes improvements in the size (in memory) and speed of the program. As you can see, the program runs about 12 times faster. These three figures show how you can get the best speed results by combining optimization techniques with the compiler.

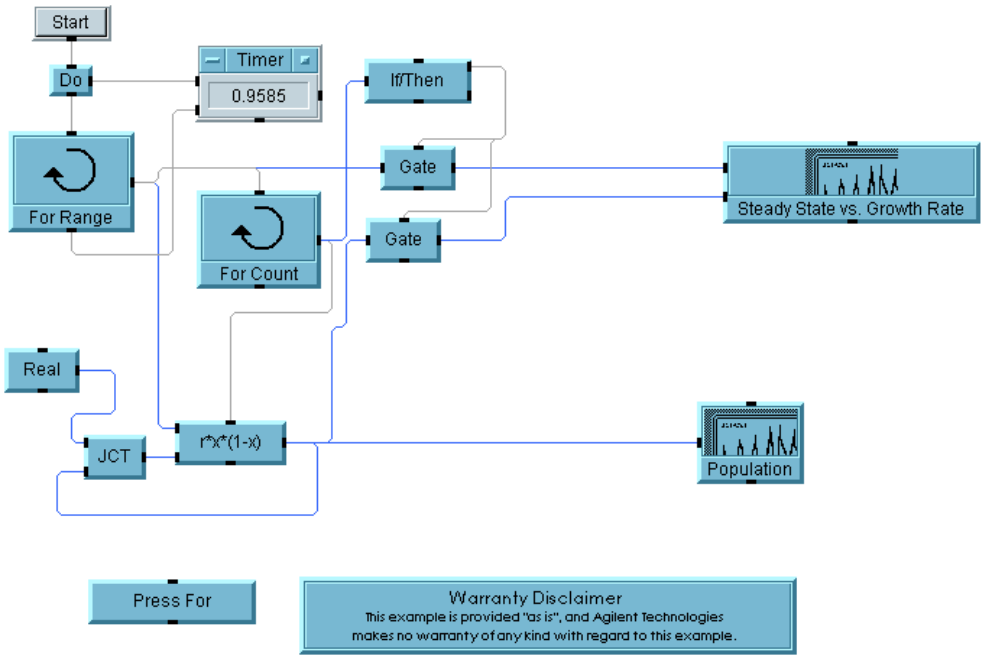


Figure 11-16. Chaos.vee in VEE 4 or Higher Mode with Debugging Disabled

In Figure 11-17 and Figure 11-18, the VEE speed improvements use the compiler on areas of programs involving iterative scalar math routines. The example calculates the square root of a scalar value. (The result is not kept.) By using the compiler, the speed is approximately 107 times faster than using the VEE 3 execution mode.

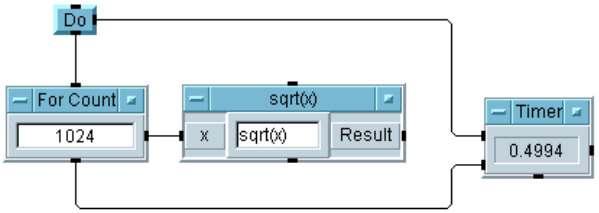


Figure 11-17. Iterative Math Example in VEE 3 Mode

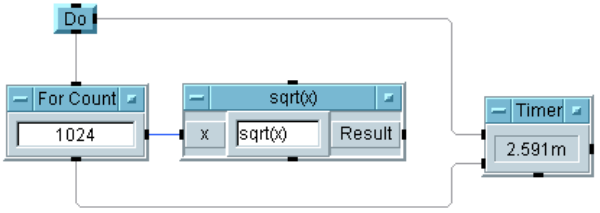


Figure 11-18. Iterative Math Example Using VEE 4 or Higher Mode

VEE includes the execution modes because there are a few programming choices allowed in older versions of VEE that are not permitted in the current version (they now produce error messages). Furthermore, with some of the advances in the ActiveX Automation and Control capabilities, some programs that ran in VEE 4 or VEE 5 modes require minor modifications to run in `VEE 6` mode. For details about the differences between the Execution Modes, refer to the *VEE Pro Advanced Techniques* manual. For all new programs you should begin in VEE 6 mode.

The Agilent VEE Profiler

The `Profiler` is a feature in the Professional Development Environment in VEE. The `Profiler` helps you optimize programs by displaying the execution speeds of `UserFunctions` or `UserObjects` in the program.

You can use the `Profiler` to identify the slow points in a program and apply the techniques described in this chapter to increase the program speed. Figure 11-19 shows the `examples\Applications\mfgtest.vee` program.

To turn on the Profiler, select `View` ⇒ `Profiler`. Then run the program. You can see the `Profiler` in the lower half of the screen. The `Profiler` lists comparative information regarding the amount of time it takes to execute each `UserObject` and each `UserFunction`.

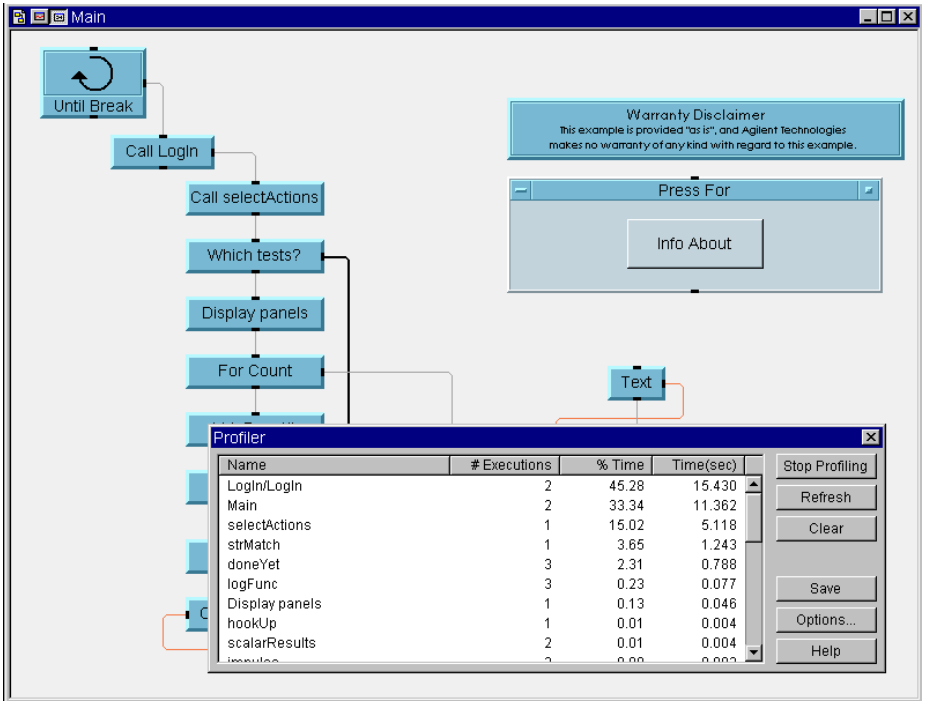


Figure 11-19. An Example of the Profiler

Chapter Checklist

You should now be able to perform the following tasks. Review topics, if necessary.

- Explain three basic techniques for optimizing VEE programs and give examples of each.
- Explain at least two more techniques in addition to the three above.
- Explain the basic concept of a DLL.
- Import a DLL, call a function within it, then delete the DLL.
- Explain how to use a compiled function in another language on an HP-UX platform.
- Step through a program using the VEE 6 execution mode or the VEE 5, VEE 4, or VEE3 execution modes, and explain the reasons you would choose one or the other.
- Use the VEE Profiler.

12

Platform Specifics and Web Monitoring

Platform Specifics and Web Monitoring

In this chapter you will learn about:

- The differences between the PC and HP-UX platforms
- Communicating with a Rocky Mountain Basic program
- Calling VEE functions from other applications using the VEE ActiveX Automation Server
- Web Monitoring

Average Time To Complete: 2 hours

Overview

In this chapter, you will learn about the key differences between operating systems, and how VEE has designed objects to handle them. Then you will learn about one of the most important techniques for incorporating VEE functions into other applications or programs using the Callable VEE ActiveX Automation Server. Finally, you will learn key concepts in web monitoring.

VEE programs transfer between the supported platforms, but there are some objects that are unique to particular operating systems. For example, on a PC, VEE uses Dynamic Link Libraries (DLLs) as compiled functions, and on HP-UX, VEE uses shared libraries. VEE uses ActiveX Automation on a PC for interprocess communication, and named pipes on HP-UX.

Differences Between PC and HP-UX Platforms

There are several differences in using VEE on a PC and using VEE on HP-UX.

Programs

All VEE features can be used in programs on all platforms. However, if the object is dependent on the operating system, it will only execute on that operating system. You can allow for these differences when porting programs by using the `System Information` objects in the `Function & Object Broswer` box: `whichOS()` or `whichPlatform()`. They output the operating system (OS) or platform so that the program can detect whether to use an "OS-dependent" object. The “OS-dependent” objects have `PC` or `UNIX` after the object name in the menu.

Named Pipes and ActiveX Capabilities

The `To/From Named Pipe (UNIX)` object and `ActiveX Automation` on PCs accomplish the same task of communicating with another program or application on their respective operating systems.

Rocky Mountain Basic

The `Initialize Rocky Mountain Basic (UNIX)` and `To/From Rocky Mountain Basic (UNIX)` are only designed to work with Rocky Mountain Basic on HP-UX.

The Execute Program Objects

The `Execute Program` object has two versions: one for HP-UX, one for the PC. Both are used for launching other programs or applications.

To/From Stdout, Stderr (UNIX)

Although these objects do work on a PC, they are implemented with files and are not recommended for general programming. You should only use them when porting a VEE program from an HP-UX platform to the PC.

Fonts and Screen Resolutions

VEE on a PC chooses a font size that looks good with the screen resolution. Use the `File ⇒ Default Preferences` dialog box to change the look and feel of programs. Use the `Properties` selection in the object menus to customize individual objects. If necessary, VEE will translate the font saved in the program to one available on the destination machine. For best results, you may want to build the program on a computer with a screen resolution and font set similarly to the destination machine. When porting programs, make sure the screen resolutions are similar to avoid problems. See `fonts, using` in the online `Help Index` for more information.

Data Files

ASCII data files constructed with the `To File` object should be readable with the `From File` object on either the HP-UX or the Windows platform. No binary files will work across platforms, since the byte ordering is reversed between HP-UX and Windows.

Communicating with Rocky Mountain Basic Programs

VEE includes objects to facilitate communication between VEE and Rocky Mountain Basic on HP-UX. For Series 700 workstations (HP-UX 10.20) you can use VEE 6.0. For Series 300 and 400 workstations (HP-UX 9.0), you must use HP VEE 3.1.

Using the Initialize Rocky Mountain Basic Object

The `Initialize Rocky Mountain Basic` object, as shown in Figure 12-1, has a single field, in which you specify the Rocky Mountain Basic (RMB) program to run.

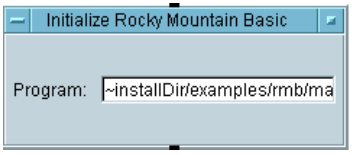


Figure 12-1. The Initialize Rocky Mountain Basic Object

Enter the entire path and any options for the program. For example, the program `~installDir/examples/rmb/man34a.bas` may have been stored or saved in RMB. The object will spawn the RMB process and run the program. You can also use relative paths from the present working directory to specify the program. This object does not provide any data path to or from RMB. Use the `To/From Rocky Mountain Basic` object to specify the program. You can use more than one `Initialize Rocky Mountain Basic` object in a VEE program.

Note

There is no direct way to terminate an RMB process from a VEE program. Instead, use a `QUIT` statement in the RMB program when it receives a certain data value from the VEE program. You could also use an `Execute Program` object to kill the RMB process using a shell command, such as `rmbkill`. When you exit VEE, any RMB processes still attached are terminated automatically.

Using the To/From Rocky Mountain Basic Object

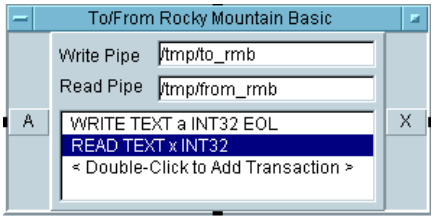


Figure 12-2. The To/From Rocky Mountain Basic Object

Figure 12-2 shows the `To/From Rocky Mountain Basic (UNIX)` object, located under `I/O` ⇒ `Rocky Mountain Basic (UNIX)`. This object facilitates data transfer to and from RMB programs. It creates and uses named pipes for interprocess communication. For simplicity, VEE implements one pipe for `READ` transactions and another pipe for `WRITE` transactions.

In the `To/From Rocky Mountain Basic (UNIX)` object shown in Figure 12-2, there are two transactions: writing an `Int32` integer, and reading an `Int32` integer. You can use the default pipes or create your own by typing in new paths and filenames for `Write Pipe` and `Read Pipe`. Transactions are configured the same way as other transaction objects in VEE.

Note

The `Write Pipe` and `Read Pipe` fields can be added to the object as control pins.

The following items describe more information about communicating with Rocky Mountain Basic:

default names for read and write pipes

All `To/From Rocky Mountain Basic` objects contain the same default names for read and write pipes. Therefore, be sure to specify the correct pipe to `READ` or `WRITE`. Make sure that pipes to different programs have unique names. In the RMB program, be sure to address `OUTPUT` and `ENTER` statements to the correct pipe.

pipes that are created automatically

If pipes do not exist before `To/From Rocky Mountain Basic` operates, the following pipes are created automatically:

```
/tmp/to_rmb
```

```
/tmp/from_rmb
```

Note, however, that if the pipes exist beforehand, the program runs more quickly.

creating additional pipes

To create additional pipes, use the operating system command `mknod`.

opening and closing pipes

Rocky Mountain Basic pipes (which are simply named pipes) are opened when the first `READ` or `WRITE` transaction to that pipe operates after `PreRun`. All named pipes are closed at `PostRun`. (For more information about `PreRun` and `PostRun`, refer to the *VEE Pro Advanced Techniques* manual.) The `EXECUTE CLOSE READ PIPE` and `EXECUTE CLOSE WRITE PIPE` transactions enable you to close pipes at any time.

structuring transactions with pipes

Because of the behavior of named pipes, it is easiest to structure transactions to transmit known or easily parsed data blocks. For example, if you are transmitting strings, determine the maximum length block you wish to transmit and pad shorter strings with blanks. This avoids the problems of trying to read more data from a pipe than is available and of leaving unwanted data in a pipe.

using DATA READY transaction

To help prevent a `READ` transaction from hanging until data is available, use a `READ IOSTATUS DATA READY` transaction in a separate `To/From Rocky Mountain Basic` object. The transaction returns a `1` if there is at least one byte to read, and a `0` if there are no bytes to read. To read all the data available on the read pipe, use a `READ ... ARRAY ID TO END:(*)` transaction.

running in environment without disk

If you are running diskless, be certain you `WRITE` and `READ` to or from uniquely named pipes. Otherwise, several workstations on the same diskless cluster may attempt to access the same named pipe, which will cause contention problems.

Figure 12-3 shows how to set up communications with an RMB program. This program is located in `manual34.vee` in the `examples/rmb` subdirectory on the HP-UX platform only.

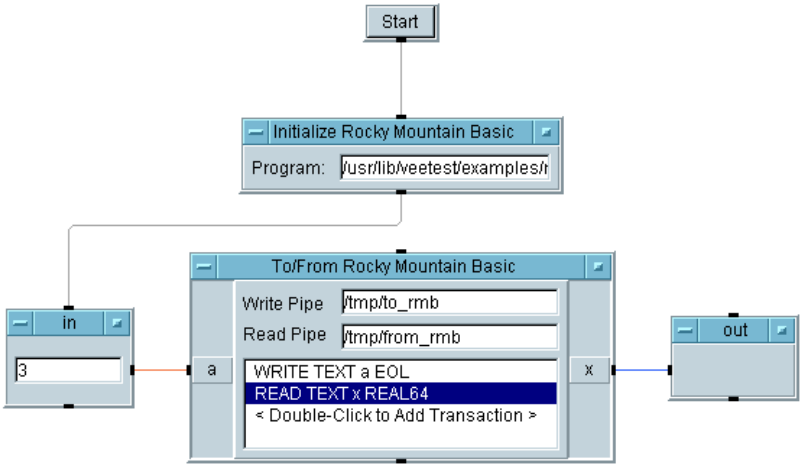


Figure 12-3. Communicating with Rocky Mountain Basic

The Callable VEE ActiveX Automation Server

In VEE on Windows 95, Windows 98, Windows NT 4.0, and Windows 2000, you can integrate VEE objects into other commercial and proprietary test systems written in standard languages, such as MS Visual Basic or C. The Callable VEE ActiveX Automation Server encapsulates VEE `UserFunctions` for integration into Automation applications, such as MS Excel and MS Visual Basic. The Callable VEE ActiveX Automation Server implements an automation interface which is language independent. It may be used by any application or language that can support calling an Automation Server.

Note

VEE can use ActiveX Automation and Controls to control other applications from VEE. The Callable VEE ActiveX Automation Server allows other applications like MS Visual Basic to control VEE.

The Callable VEE ActiveX Automation Server communicates with its clients through properties and methods. The environment from which you are calling the Callable VEE ActiveX Automation Server, such as Visual Basic or C, determines how you call it. The Callable VEE ActiveX Automation Server has extensive online `Help` available within the environment from which you are calling it. For more information, refer to the online `Help` in VEE, the *VEE Pro Advanced Techniques* manual, and the examples shipped with VEE.

Note

The Callable VEE ActiveX Automation Server replaces the Callable VEE ActiveX Control that was shipped with VEE version 5.0.

Web-enablement Technologies

You can use VEE to disseminate data you have collected in programs, to monitor test systems, or to review test results remotely. This section describes web-enablement technologies for test and measurement applications, and how VEE supports these technologies.

Overview of Web Technologies

This example describes how to create a web site internal to an organization (an intranet web site) that provides reference data as a server. It assumes that users accessing this site have browsers. The example uses a Microsoft environment (Windows 95, Windows 98, Windows NT 4.0, or Windows 2000), MS Office, and MS Internet Explorer. The communication will go from the instrument to the server, and then to the client browser, as shown in Figure 12-4.

Note This example uses PC screen dumps. VEE is also web-enabled on HP-UX.

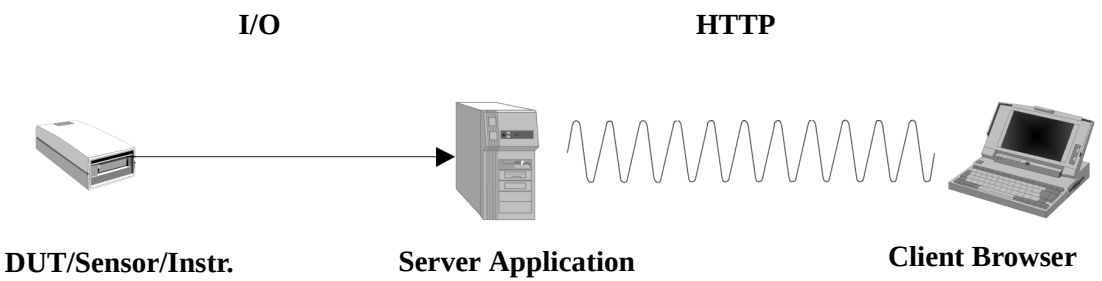


Figure 12-4. Model of Web Measurement Application

In Figure 12-4, the communication is as follows:

- The *device under test (DUT), or sensor, or instrument* sends information to the server application through the network I/O layer. The network I/O layer includes the interface and backplane (GPIB, RS232, VIX, MXI, PC Plug-in), and the I/O programming layer such as drivers, VISA, or SICL.

- The *server application* is the VEE program that generates the measurement data.

- The *HTTP (HyperText Transfer Protocol)* sends the information to the client browser.

HTTP is one protocol in the TCP/IP communications protocol used by the Web. (TCP/IP stands for Transmission Control Protocol/Internet Protocol.) Lower levels of the TCP/IP protocols include transport, network, and physical layers of communication. By definition, every TCP/IP application is a client/server application. For example, the Client Browser (such as Internet Explorer or Netscape Navigator) can request information generated by the server application.

The Universal Resource Locator (URL) below typed in Internet Explorer requests information from an Agilent server:

```
http://www.agilent.com/find/vee
```

- `http` describes the type of resource being accessed to transfer the information.

- `www.agilent.com/find/vee` is the URL for the resource. The hypertext format used by HTTP is a scripting format called HyperText Markup Language (HTML). HTML is a way to link documents together, and originally it was the only language programmers could use to create Web pages. Originally just for text, HTML now incorporates sound, video, images, interactive screens, ActiveX controls and Java applets.

Once a request is made by the browser for the server information, it will not automatically update, unless the browser is designed to do so. Also, no interaction is allowed by the browser unless it is designed into the browser page. The easiest way to do this is with a scripting language, such as VBScript, JavaScript, and JBscript.

The scripting language is an interpreted language supported by the browser. The scripting language can extend the limitations of HTML to provide a more interactive Web page. Because they are interpreted, scripting languages must be embedded into the Web page and supported by the browser. They are not independent programs. This is graphically illustrated in Figure 12-5.

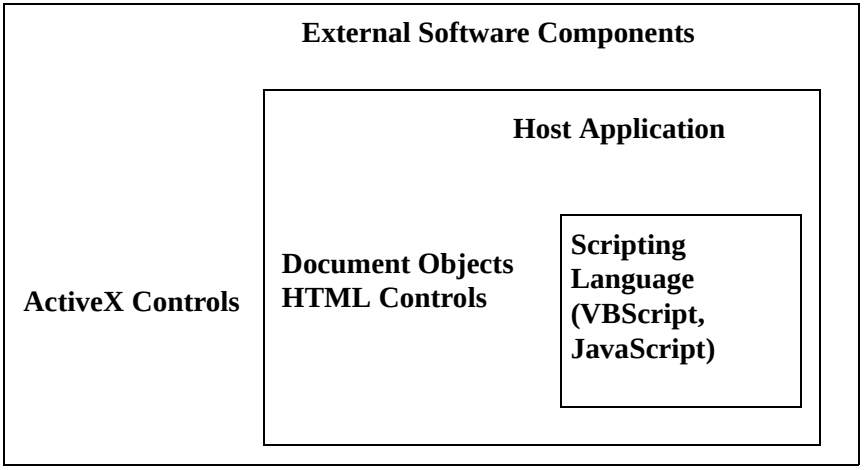


Figure 12-5. A Scripting Language Host Model

VBScript, JavaScript, and JScript are scripting languages. VBScript is based on MS Visual Basic. JavaScript is co-created by Sun (Java). JScript is based on Microsoft's version of JavaScript.

The scripting languages must reside inside a host application. The host application is typically a web browser such as Internet Explorer or Netscape Navigator. Within the browser, the overall look and feel of the web page is controlled by HTML.

Microsoft's Component Object Model (COM) defines compiled software components called ActiveX Controls that encapsulate specifically designed functions. Typically, an ActiveX Control is used to provide user interface functionality and is designed to run on the client computer. ActiveX Controls are optimized software components that used to be called OLE Controls.

Web Monitoring with Agilent VEE

VEE includes a built-in Web server that can communicate with other programs via HTTP. You can allow remote users to access VEE programs and information that reside on your computer. This section describes how you would share VEE data, and how a remote user would access the data.

General Guidelines and Tips

- The VEE program running on your system is the VEE program that the remote user will access.
- Your system must be running VEE. The remote user does not need to have VEE installed in order to access a VEE program on your system.
- When the remote user sends a request from their network browser to VEE, VEE creates a picture to display in the remote user's browser window. This picture is a "snapshot" of the VEE program. (It cannot be edited.)
- By making choices in the VEE Web Server Home Page or by specifying command line options in the browser URL, a remote user can view different parts of your VEE program, have the VEE program refresh the browser display at regular intervals (to monitor the program's progress), and display error message information.

Providing Agilent VEE Data to a Remote User

To set up the VEE Web server so that a remote user can access data on your system, follow these general steps:

1. Make sure your system is connected to a network.
2. Provide information to the remote user about the URL he or she will enter in the browser to access your system (which is described in more detail below).

3. Start VEE, and open the program that you want the remote user to access, and/or create any files you want the remote user to access.

4. Enable the Web Server by selecting the `File ⇒ Default Preferences ⇒ Web Server` dialog box settings which are described in more detail below.

5. Have the remote user run a Web browser such as Internet Explorer or Netscape.

Web Server Dialog Box

When you select `File ⇒ Default Preferences ⇒ Web Server`, the `Web Server` dialog box appears as shown in Figure 12-6.

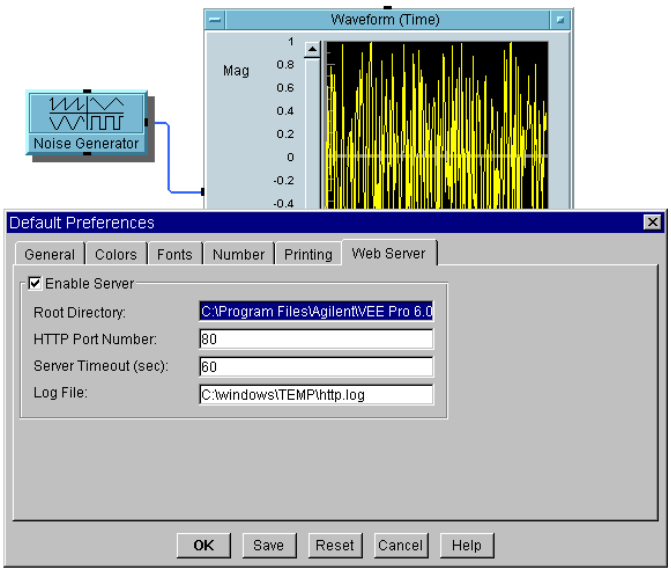


Figure 12-6. The Default Preferences Web Server Dialog Box

The fields in the `Web Server` dialog box are as follows:

Enable Server

When checked, `Enable Server` turns on VEE's built-in Web server. The Web server allows a remote user to view, monitor and troubleshoot VEE programs on your system, by using a Web browser to display your VEE program.

VEE's Web server uses the standard HTTP protocol. By default, `Enable Server` is `OFF` (not checked). Your system must be connected to a network, VEE must be running, and the remote user must have a Web browser running for the remote user to access the VEE Web server.

Root Directory

Specifies the location of files that are accessible to a remote user. The default is *~installDir*`/www`. For Windows 95, Windows 98, Windows NT 4.0, and Windows 2000, if VEE Pro 6.0 is installed with the defaults, this field will read:

`C:\Program Files\Agilent\VEE Pro 6.0\www`.

- *Do not* put private files in this directory or any of its subdirectories, since they can be viewed by any remote user with a Web browser that has access to your system.
- VEE installs the file `index.html` in the default Web server `Root Directory` when you install VEE. By default, the file will be in `C:\Program Files\Agilent\VEE Pro 6.0\www\index.html`. This is the default VEE Web server home page, which can be displayed when a remote user accesses your system. (You can edit the home page for your needs, if you like.)
- If you change the root directory for Web files, move the `index.html` file to the new directory and do not change its name.

HTTP Port Number

Specifies the port number for the VEE Web server. The default port number is 80. The only time you need to change the default port number is when another Web server, such as another instance of VEE, is running on the same system, or if you want to restrict the remote user access to your VEE Web server.

- By specifying an HTTP port between 0 and 65535, you can restrict who can access and view data on your system.
- On HP-UX, only port numbers greater than 1024 are allowed. The default port number is 8080.
- If you enter a different port number, the remote user that views the VEE program must enter the same port number in his or her browser. For example, if you set this field to port 85, the remote user would type in the URL as `http://`*hostname*`:85`. (There is more information about the definition of *hostname* later in this chapter.)

Server Timeout

Specifies the maximum time that the VEE Web server will wait for VEE to process commands. By default, this field is set to 60 seconds. If the VEE program takes longer than the specified time to process a command, VEE sends a timeout to the Web server.

Log File

Specifies the log file where all incoming HTTP requests, processed by the Web server, are recorded. The default location for this file on a PC is `C:\windows\TEMP\http.log`. If the log file does not already exist, it is created when you make changes in the `Default Preferences` ⇒ `Web Server` dialog.

Save

Saves the attributes displayed as permanent choices. This saves the settings in the `vee.rc` file on Windows or `.veerc` file on HP-UX.

The values will be the default for future VEE Web sessions until you change them again. To use the specified values *only* for the current VEE session, click `OK`.

How a Remote User Accesses Agilent VEE on Your System

To access VEE files on your system from another location, the remote user needs to follow these general steps:

1. The remote user's system must be connected to a network.
2. The remote user must be running a network browser, such as Netscape or Internet Explorer.
3. The remote user must enter a URL address for your system. (You provide the address to the remote user, which is explained in more detail below.)

Note

The remote user does not have to be running VEE or even have VEE installed in order to access VEE programs on your system.

The remote user connects to the network and runs the browser, and enters a URL. You will tell the remote user the URL to enter based on the following information. The formats you can have the remote user enter are as follows:

`http://hostname {:port}`

Displays your VEE Web Server Home Page, where the remote user can enter choices about how to view your VEE program.

`http://hostname{:port}{/command}{?parameter}`

Specifies a view or `UserFunction` for the remote user to view in the VEE program. For example, `http://`*`hostname`*`/ViewMainDetail` displays the Detail view of your main VEE program.

`http://hostname{:port}{/file}`

Specifies a file that you have saved, such as a `*.jpeg` or `*.html` file, that you want the remote user to view.

Note The fields in wavy parentheses { } are optional fields.

The fields in the URL addresses are as follows:

Hostname (Required) Identifies your system, where VEE is running, in the format <*computer_name*>`.domain.com`.

You may choose to have a remote user type in only the <*hostname*> in a URL, such as `http://`<*hostname*>. This command opens your VEE Web Server Home Page `index.html` and displays it to the remote user. The remote user can make choices from your VEE Web Server Home Page menu to display, monitor, or troubleshoot your VEE program.

Port (Optional) Identifies the Web Server port number if not using the default value 80. Specify this value only when the port number you entered in `File` ⇒ `Default Preferences` ⇒ `Web Server` is a number other than 80.

For example, if the remote user is accessing VEE on your system, and the port number in your `File` ⇒ `Default Preference` ⇒ `Web Server` is set to `85`, you would have the remote user enter `http://`<*hostname*>`:85`.

File	(Optional) Identifies a directory and/or file relative to the root directory for the browser to open. You would only specify a file when you have saved a file such as a `*.jpeg` or `*.html` for a remote user to view. In order for remote users to display a file from your system, you must specify the directory as the `Root Directory` in the `Web Server` dialog in `Default Preferences` in VEE.
Commands and Parameters	(Optional) Specifies a command or a parameter required for a command that is supported by the VEE Web Server. Commands and parameters enable a remote user to monitor and troubleshoot a remote VEE program through the browser. The following lists the commands and parameters.

The commands and parameters that can be used in URLs when a remote user accesses a VEE program on your system are as follows.

Viewing the entire VEE window	`ViewVEE` For example, `http://`*hostname*`/ViewVEE`

Panel view of main VEE program	`ViewMainPanel`
	
	`For example,`
	`http://`*hostname*`/ViewMainPanel`

Detail view of main VEE program	`ViewMainDetail`
	
	`For example,`
	`http://`*hostname*`/ViewMainDetail`

VEE execution window during runtime	`ViewExecWindow`
	
	`For example,`
	`http://`*hostname*`/ViewExecWindow`

VEE UserFunction in Panel view	`http://`*hostname*`/ViewPanel?` *<UserFunction Name>* For example, if the `UserFunction` is `AddNoise`, the remote user would enter `http://`*hostname*`/ViewPanel?AddNoise`
Detail view of a UserFunction	`http://`*hostname*`/ViewDetail?UserFunctionName` For example, if the `UserFunction` is `AddNoise`, the remote user would enter `http://`*hostname*`/ViewPanel?AddNoise`
Error window of current program	`http://`*hostname*`/ViewError`
Display list of available command URLs.	`ViewHelp`

Displaying the Agilent VEE Web Server Page

When you install VEE, it creates a default `index.html` file in the `www` directory. This file contains the VEE Web Server Home Page. You can have remote users click choices on this page for displaying your VEE program. The default VEE Web Server Home Page is shown in Figure 12-7. You can also edit the page, in MS Word, for example, to suit your needs.

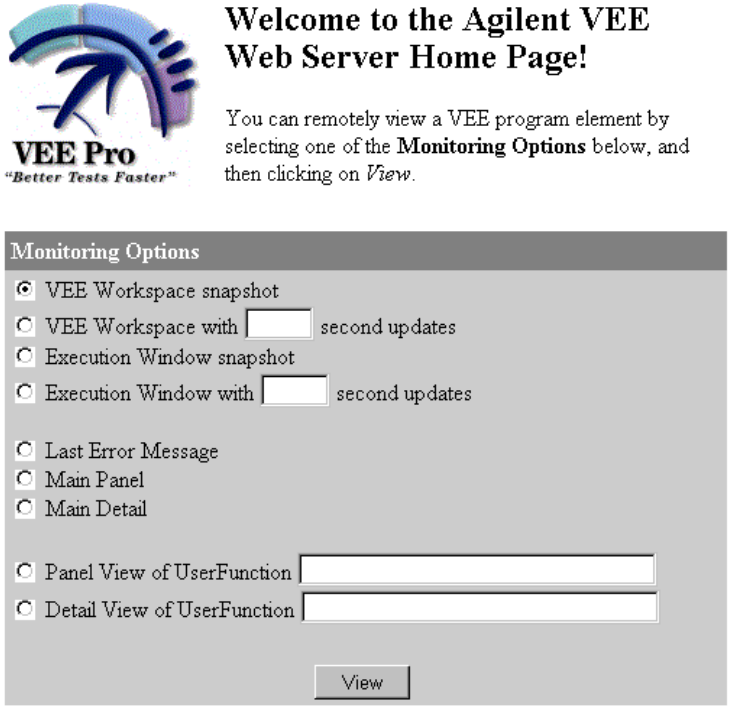


Figure 12-7. The Index.html Default Page

Note

To display this menu on your own system, you can refer to your system as `localhost`. For example, if you run VEE, run the network browser, and enter `http://localhost`, the browser will display the VEE Web Server Home Page shown in Figure 12-7. This is an easy way for you to check what the remote user will display with different commands.

When the remote user displays the menu shown in Figure 12-7, he or she can access various parts of a VEE program by making choices in the menu. For example, the user can click on `Main Detail` to view the Main VEE program in Detail view. Making the choice in the menu displays the same information as having the remote user enter the command `http://`*hostname*`/ViewMainDetail` in the network browser.

Lab 12-1: Practice Session with Agilent VEE Web Browser

This exercise simulates a Web session where you provide the `Solitaire.vee` program for a remote user to view on your system. In this case, there is an error in the program and the remote user is consulting with you on how to resolve the error.

1. Start VEE. Select `File` ⇒ `Default Preferences` ⇒ `Web Server` dialog box and click on `Enable Server`. Use the default settings. Open the `Solitaire.vee` program that you want the remote user to view. Run your network browser.

2. Contact the remote user and let him or her know to enter `http://Server5` to reach your system over the Web. (You would use your computer name instead of `http://Server5`.)

3. When the remote user enters the URL `http://Server5`, the remote user sees your VEE Web Server Home Page displayed in his or her browser. (To review what the display looks like, refer to Figure 12-7 on page 458.)

4. The remote user decides to view the entire VEE program first. In the VEE Web Server Home Page, he or she clicks `Main Detail` ⇒ `View`. The browser displays the view shown in Figure 12-8.

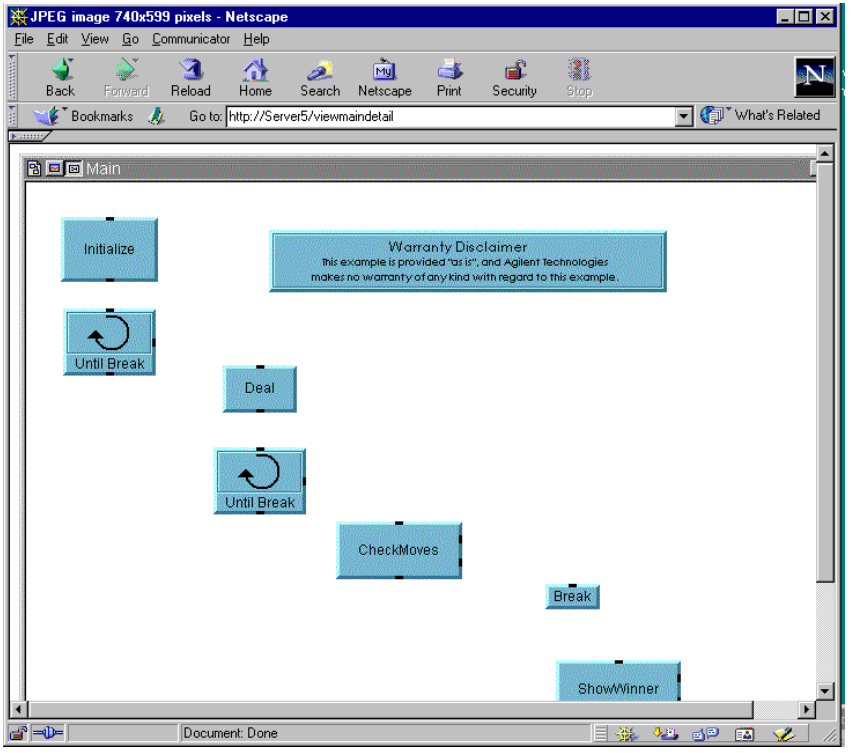


Figure 12-8. Viewing the Main Solitaire.vee Program in the Browser

Figure 12-8 displays the Main program in VEE.

Note

For this exercise, the `Solitaire.vee` program includes an error that is *not* in the VEE program example. If you would like to view the program, it is located in `Help` ⇒ `Open Example...` ⇒ `Games` ⇒ `Solitaire.vee`.

5. The remote user clicks `Back` in the browser to display the VEE Web Server Home Page and chooses `Last Error Message`. The browser displays the error message shown in Figure 12-9.

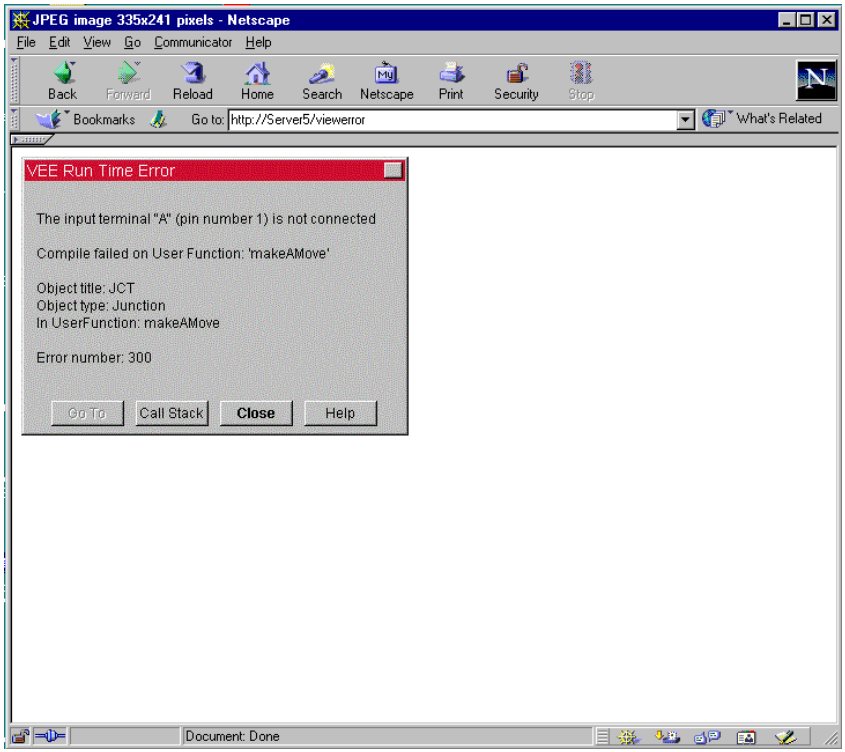


Figure 12-9. Displaying a VEE Error Message, using the Browser

Notice that the VEE error message specifies the `UserFunction makeAMove`.

6. The remote user goes back to the VEE Web Server Home Page once again and clicks on `Detail View of UserFunction`, and types in the `UserFunction` name `makeAMove`. The browser displays the `UserFunction makeAMove` as shown in Figure 12-10.

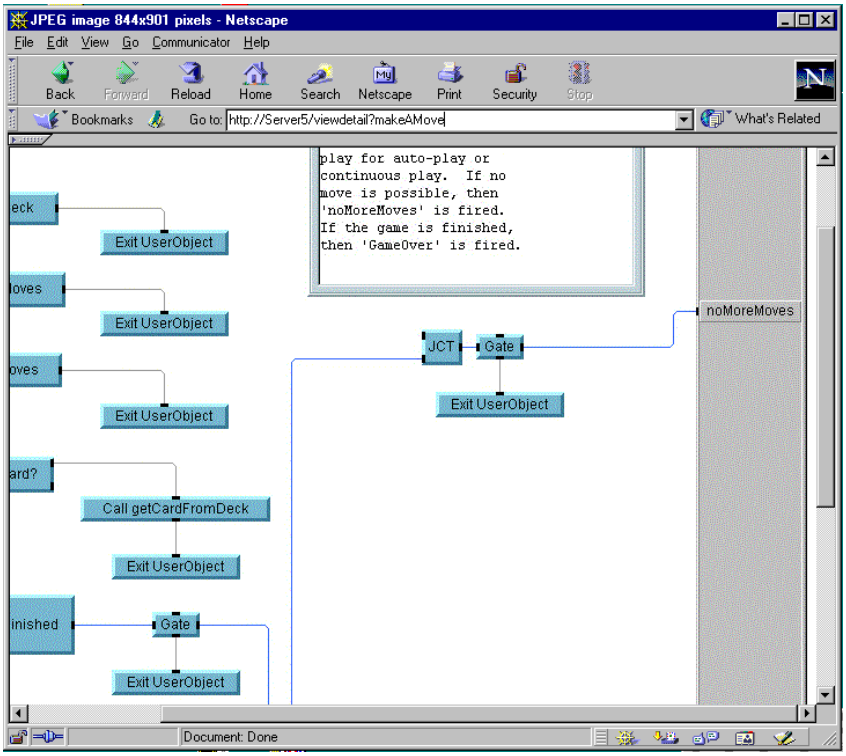


Figure 12-10. Detail View of a UserFunction Displayed in the Browser

The remote user is able to see the error in the VEE program. There is an input pin not connected on the `JCT` object shown in Figure 12-10. The remote user could now help you troubleshoot `Solitaire.vee` and resolve the error. Using a similar process of working together over the Web, you could collaborate with remote users or develop programs together.

Restricting Access to Programs Viewed over the Web

When you make a VEE program available on the Web, you may still want to restrict remote users from seeing certain parts of it. If remote users currently know the URL to your system, you want to make sure that only certain remote users are able to access particular programs or Web directory files.

To prevent remote users from viewing parts of a VEE program on the Web, you can protect the program in three different ways:

1. Change the port number in the `Default Preferences ⇒ Web Server` folder so only authorized users may view the program.

-OR-

Create a secured `RunTime` version of the VEE program. This will ensure that none of the program code can be viewed. For more information, refer to "Securing a Program (Creating a RunTime Version)" on page 380.

-OR-

Create an HTML file with the exact name of the command you want to disable, and save it in the VEE `www` directory. The browser always accesses any `*.html` file before going to VEE. In this way, you can intercept requests from remote users and display an HTML page with the appropriate warning or comments.

For example, you might want to prevent remote users from seeing the Detail view of a VEE program. You could create a file in a program such as MS Word and save it as `ViewMainDetail.html` in the `www` directory. In the file, you put the message you want the remote user to see.

When the remote user chooses `Main Detail` in the VEE Web Server Home Page or enters a URL with the option `ViewMainDetail`, the browser *does not* display the main VEE program in detail view. Instead, the browser accesses the `ViewMainDetail.html` file in the `www` directory and displays the file you created. Figure 12-11 shows an example of what you could display to a remote user.

Note

Make sure the file name is the file name of a VEE web command, and that it is located in the `Root Directory` specified in `Web Server`.

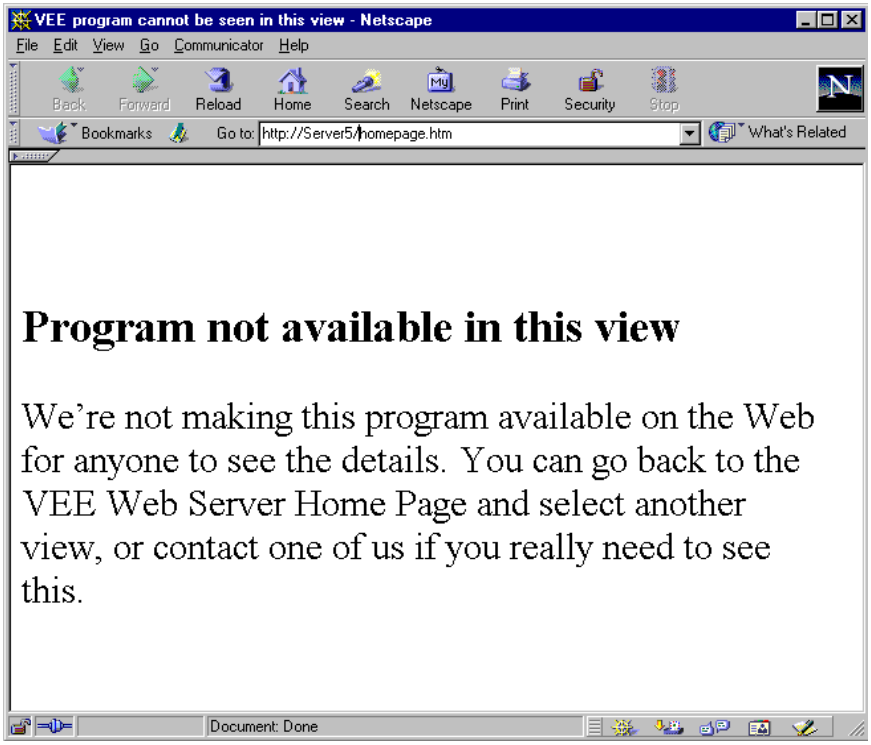


Figure 12-11. Example of Displaying HTML Message Instead of VEE Program

You could also use a `*.html` file for other purposes, such as putting password protection on a VEE program so that only users with the password can view the program. Figure 12-12 shows an example of password protection.

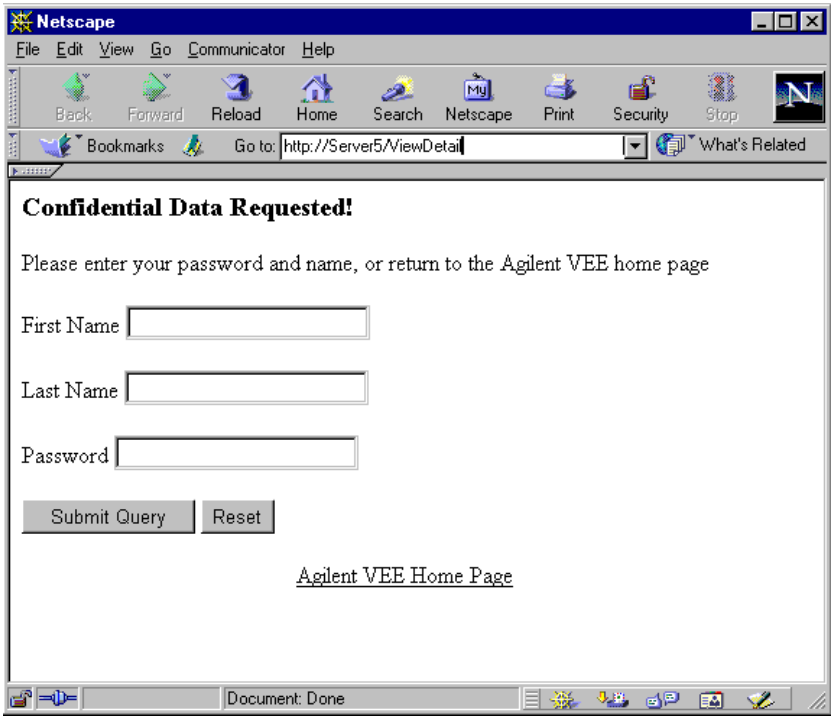


Figure 12-12. An Example of a Password Window

Chapter Checklist

You should now be able to perform the following tasks:

- Explain the key differences between PC and HP-UX platforms and the issues they raise in porting programs.
- Explain how to call and communicate with a Rocky Mountain Basic program.
- Explain how to use the Callable VEE ActiveX Automation Server, and when you would use it.
- Explain how you could integrate VEE functionality into other applications or programs.
- Explain the key concepts in using the web to monitor VEE programs.

A

Additional Lab Exercises

Additional Lab Exercises

The following exercises give you a chance to practice the VEE concepts you have learned in this book. The exercises are divided into categories.

To use this appendix, develop a solution and then compare it to the answers listed. There are many ways to program a given task, so you have a valid solution if it meets the problem specifications. However, programs that execute more quickly and are easier to use are probably better solutions. Each solution includes a short discussion of key points.

General Programming Techniques

Apple Bagger

You want to know how many apples it takes to fill a ten pound basket. Create a VEE program that counts how many apples it takes to fill the basket. Each apple weighs between 0 and 1 pound.

Suggestions

This program can be created with 10 or fewer objects. Choose from the following objects:

Start
Until Break
random() function
Accumulator
Break
Real64
Conditional (A>=B)
Stop
Counter
If/Then/Else
Alphanumeric

Note

The VEE programs for many of the lab exercises and programming examples in this manual are included in VEE, under `Help` ⇒ `Open Example...` ⇒ `Manual` ⇒ `UsersGuide`.

Solution 1—Apple Bagger

Figure A-1 shows one solution to the Apple Bagger exercise.

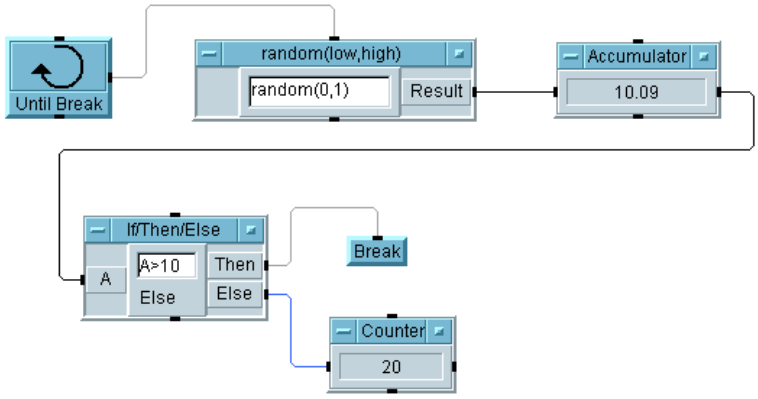


Figure A-1. Apple Bagger, Solution 1

Key Points

- **Optimal Solutions:** To optimize the performance of programs, use fewer objects, if possible. This solution uses six objects. The program could also be implemented with 10 objects, as Figure A-2 shows.

- **Until Break and Break Objects:** Use these objects for loops that require testing a condition. In this example, the loop should stop when the total weight of the apples is greater than 10 pounds.

- **Accumulator:** Use the `Accumulator` to keep a running total.

- **Counter:** Use the `Counter` to keep a running count. In this example, the `Counter` is used to track the total number of apples in the basket. Note that when the total weight is over 10, only the `Then` pin fires on the `If/Then/Else` object giving the correct answer in the `Counter`.

Solution 2—Apple Bagger

Figure A-2 gives another solution using more objects.

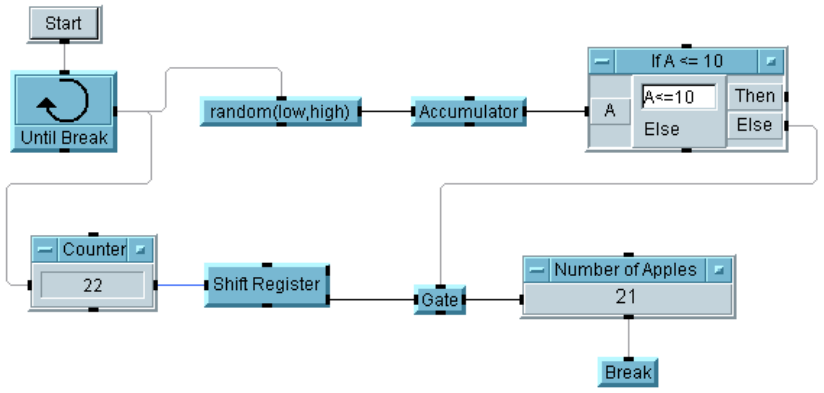


Figure A-2. Apple Bagger, Solution 2

Key Points

- **Start:** Using a `Start` object for this program is redundant, since you can use the **Run** button on the main menu bar. `Start` is best used when you have two programs on a screen, and you want to be able to run them independently. Or you have a program with a feedback loop, and you want to define where to initiate execution.

- **Shift Register:** You use a `Shift Register` to access the previous values of the output. In solution 2, the `Counter` is keeping a running count of every apple before it is weighed, so the count must be reduced by one when the total weight exceeds 10.

- **Gate:** The `Gate` is used to hold the output until another action occurs and activates its sequence pin. Here, when the condition `A<=10` is no longer true, the `Else` pin on the `If/Then/Else` object activates the gate.

Testing Numbers

Testing Numbers, Step 1

Create a program that allows a user to enter a number between 0 and 100. If the number is greater than or equal to 50, display the number. If it is less than 50, display a pop-up box with the message “Please enter a number between 50 and 100.”

Suggestions

This program can be created with 5 or fewer objects. Choose from the following objects:

Start
Int32
Slider
Real64
If/Then/Else
Formula
Gate
Text
Junction
Alphanumeric
Message Box

Solution—Testing Numbers, Step 1

Figure A-3 shows a solution to the Testing Numbers exercise using five objects.

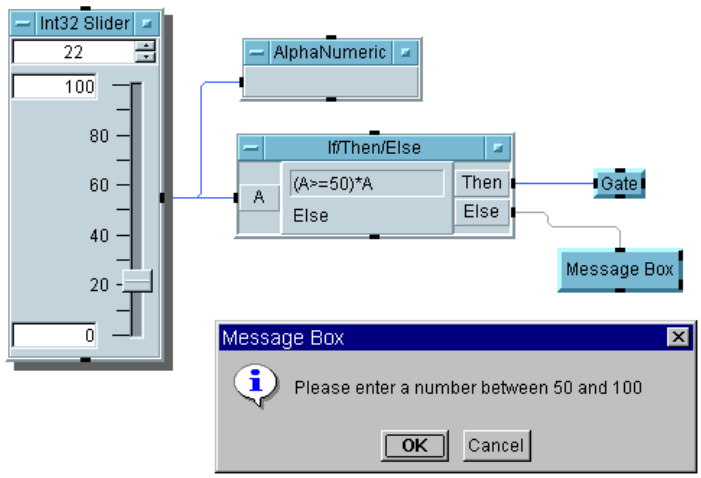


Figure A-3. Testing Numbers (pop-up shown)

Testing Numbers, Step 2

After the model is working with five objects (the Message Box produces the pop-up), try programming it with four objects without using the `Gate` object.

Solution—Testing Numbers, Step 2

Figure A-4 shows the solution to the Testing Numbers exercise with four objects.

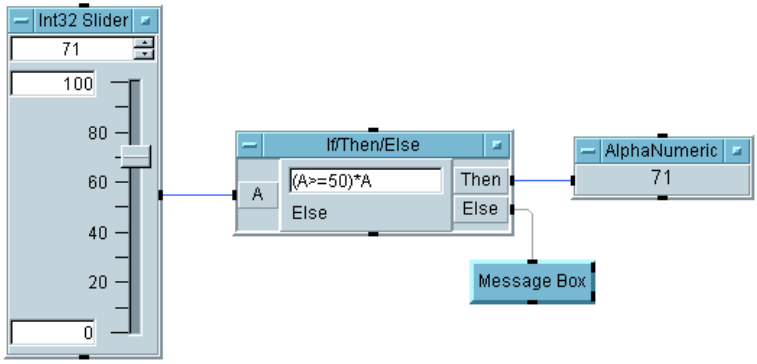


Figure A-4. Testing Numbers, Step 2

Key Points

- **Auto Execute:** All input objects such as the `Int32 Slider` have an `Auto Execute` selection in the `Properties Box`. If chosen, the object operates whenever its value is changed without needing to press `Start` or the **Run** button.

- **Eliminating Gates:** The expression `(A>=50)*A` in the `If/Then/Else` object evaluates to a `1*A`, if `A>=50` is true, or `0`, if false. So `A` is put on the `Then` pin, if the expression is true, and a `0` is put on the `Else` pin, if the expression is false. (Any expression that evaluates to a non-zero is considered true, and the value is propagated on the `Then` pin.)

Testing Numbers, Step 3

Create a solution using only three objects.

Hint: Use a triadic expression in the `Formula` object. The format is: `(`<*expression*>` ? `<*if TRUE,output value*>` : `<*if FALSE, output value*>`)`. For example, if `A < 10` evaluates to `TRUE`, you want the value of `A` on the `Result` pin; otherwise, you want the string "`FALSE`" on the `Result` pin. You would use the following triadic expression: `(A<10 ? A : "FALSE")`.

Solution—Testing Numbers, Step 3

Figure A-5 shows the solution to the Testing Numbers exercise using only three objects.

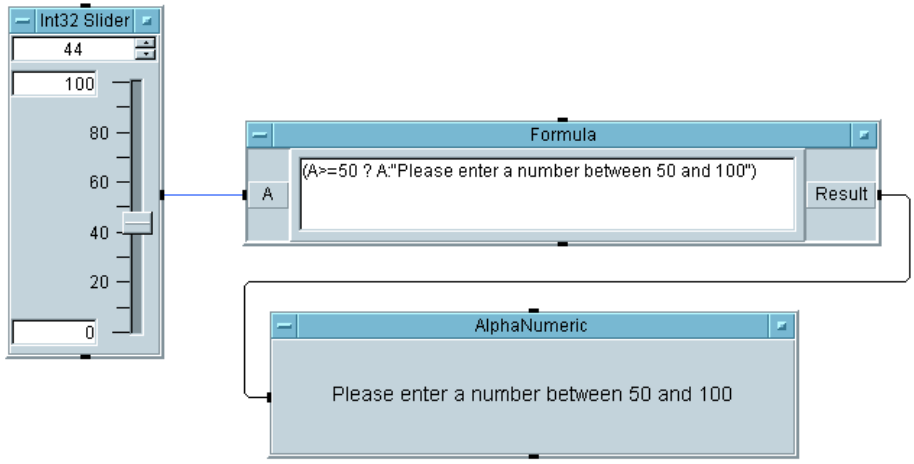


Figure A-5. Testing Numbers, Step 3

Note

This could be implemented using a `Real64 Input` dialog box with its automatic error-checking capability. However, the operator must enter a valid number before the program can complete.

Collecting Random Numbers

Create a program that generates 100 random numbers and displays them. Record the total time required to generate and display the values.

Suggestions

This program can be created with six or fewer objects. Choose from the following objects:

Start
For Range
Until Break
randomseed() function
random() function
Collector
Formula
Set Values
Alloc Int32
Logging AlphaNumeric
Strip Chart
Meter
Date/Time
Timer
Now()
Break
Do

Hint

To improve performance, send the data to the display only once by first collecting the data into an array using the `Collector` object. Note the performance differences.

Solution—Collecting Random Numbers

Figure A-6 shows a solution for the exercise Collecting Random Numbers.

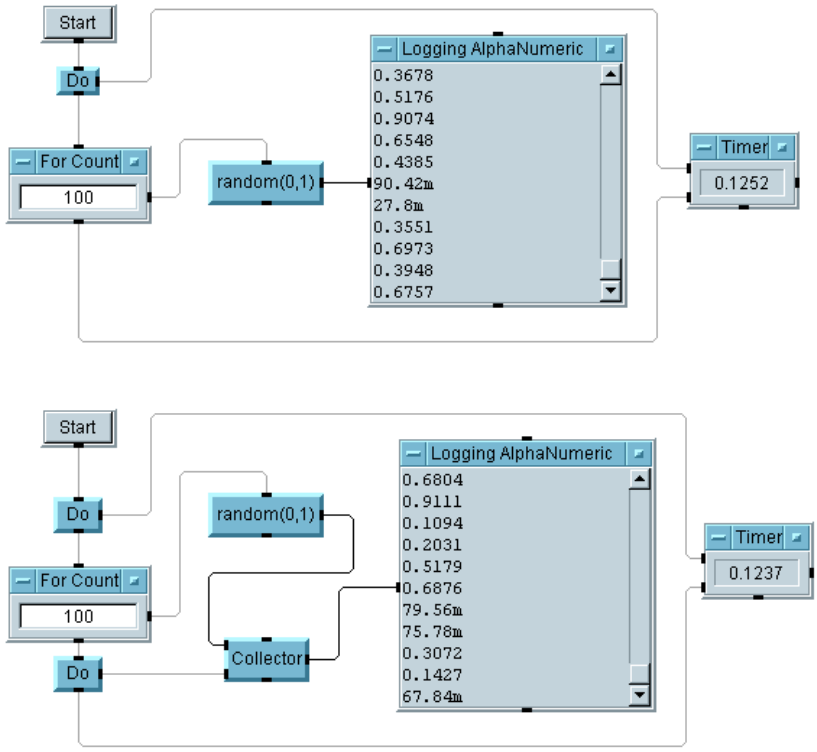


Figure A-6. Collecting Random Numbers

Key Points

- **Logging AlphaNumeric vs. AlphaNumeric:** Use `Logging AlphaNumeric` to display consecutive input (either `Scalar` or `Array 1D`) as a history of previous values. Use `AlphaNumeric` to display data from only one execution (the last) as a single value, an `Array 1D`, or an `Array 2D`. The `Logging` display is an array without index values; the `AlphaNumeric` display is the same array with optional index numbers and values.

- **Timing Pins:** The `Do` object controls which object executes first. The end of the program is timed from the sequence out pin of the `For Count` object, because that pin does not fire until all objects inside the loop have executed.

Random Number Generator

Random Number Generator, Step 1

Create a random number generator that requires external inputs. Display the numbers on a strip chart. Inputs should be allowed for:

Maximum random number
Minimum random number
Number of random numbers generated

Solution—Random Number Generator, Step 1

Figure A-7 shows a solution for the first step of the Random Number Generator exercise.

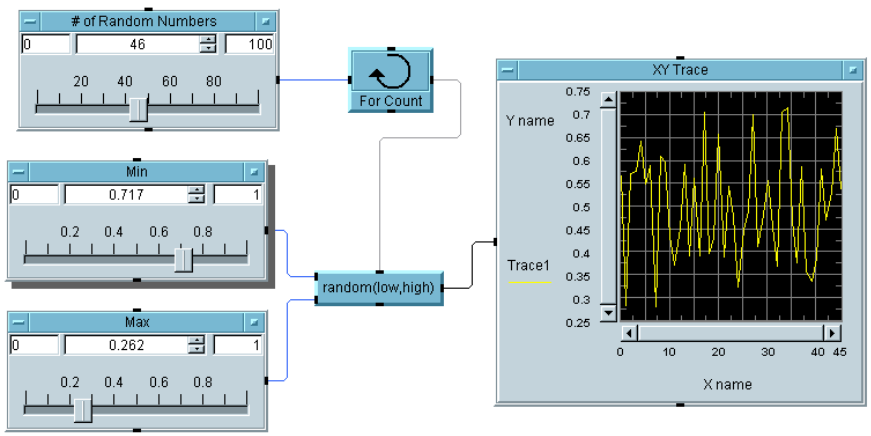


Figure A-7. Random Number Generator, Step 1

Key Points

- **Layout of Slider Objects:** You can select either a vertical or horizontal format for the screen image of the slider objects by clicking on `Horizontal` under `Layout` in the `Properties` box.
- **XY Trace:** Use an `XY Trace` to display the recent history of data that is continuously generated.

Random Number Generator, Step 2

Collect the random numbers into an array. Find the moving average and display it with the numbers.

Solution—Random Number Generator, Step 2

Figure A-8 shows a solution for Random Number Generator, step two.

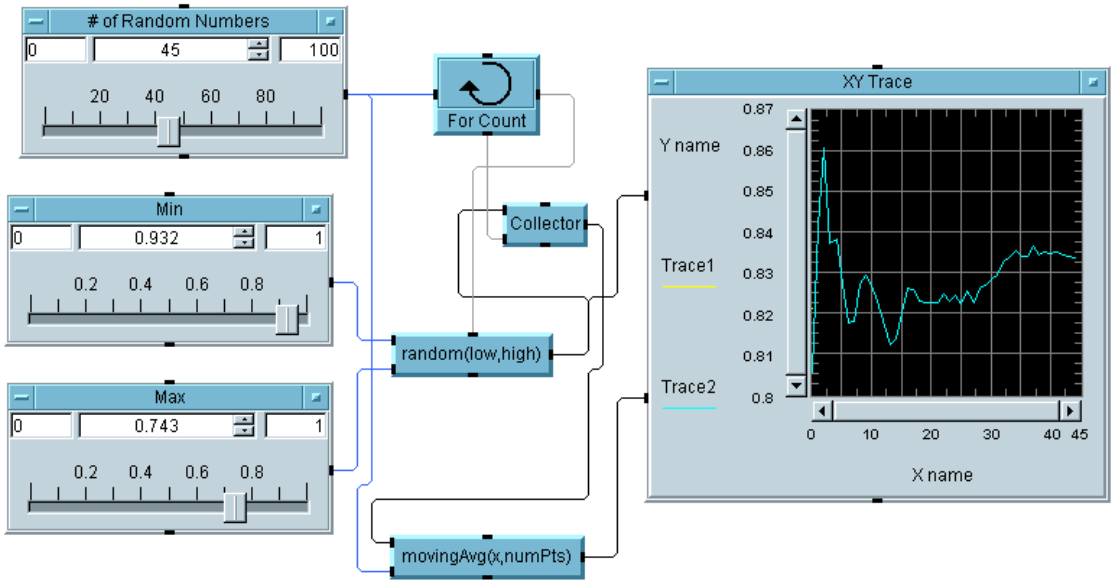


Figure A-8. Random Number Generator, Step 2

- **MovingAvg(x, numPts):** Use this object located in the `Function & Object Browser`, `Data Filtering` category to smooth the input data using the average of a specified number of data points preceding the point of interest to calculate the smoothed data point.

Using Masks

Mask Test, Step 1

Create a 50 Hz sine wave with an adjustable amount of noise. Test the `noisy sine` wave to be certain that it stays below the following limits:

(0,0.5)
(2.2m, 1.2)
(7.2m, 1.2)
(10.2m, 0.5)
(20m, 0.5)

If the sine wave exceeds the limits, mark the failing points with a red diamond.

Hints

You can change the format of the displays from lines to dots to diamonds. (In `Properties`, choose the `Traces` tab for each trace input, the line type can be solid, dashed, points only, etc. Also the `Point Type` can be just a point, a diamond, box, or other shapes.) You may find the `Comparator` object helpful.

Solution—Using Masks, Step 1

Figure A-9 shows a solution for step 1.

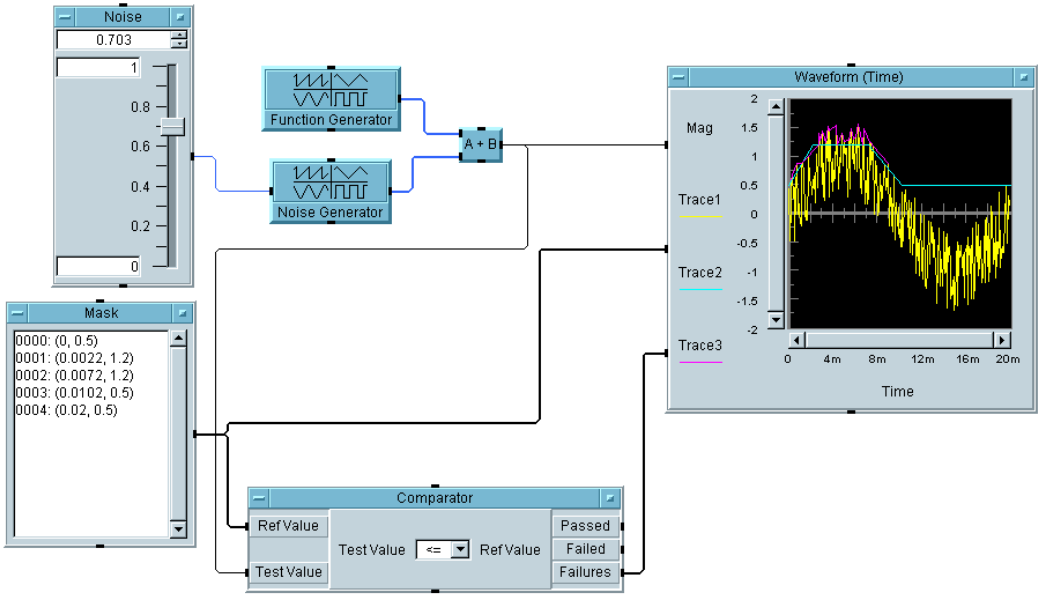


Figure A-9. The Mask Test, Step 1

Using Masks, Step 2

Add to the program to calculate and display the percentage of failures.

Solution—Using Masks, Step 2

Figure A-10 shows a solution for step 2.

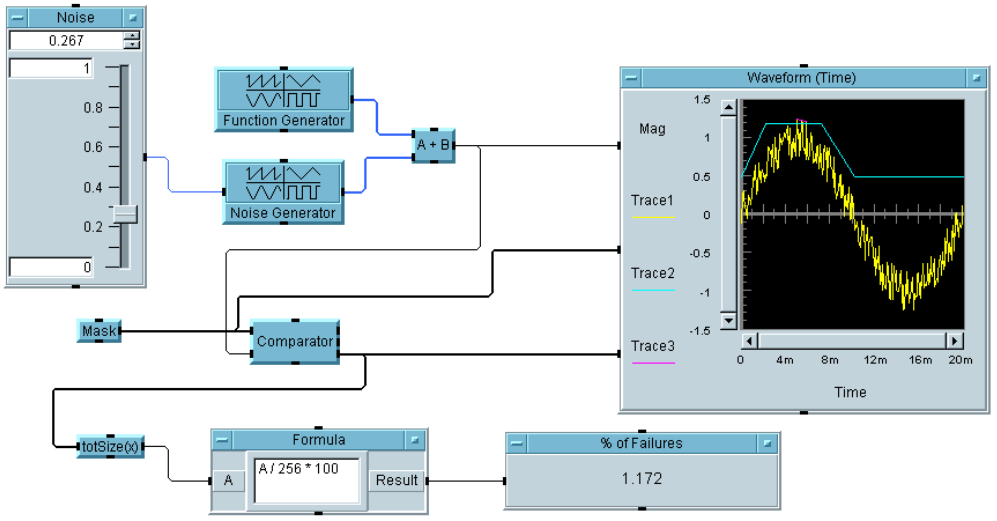


Figure A-10. Mask Test, Step 2

Key Points

- **Mask:** The mask is created using the `Data` ⇒ `Constant` ⇒ `Coord` object, then configuring it for five array elements. You input the coordinate pairs separated by commas and VEE adds the parentheses. The `x` values were chosen knowing that the time span of the waveform was 20 milliseconds. Also, note that the `Waveform (Time)` display will accept a `Coord` data type as an input. You could also use a `Data` ⇒ `Build Data` ⇒ `Arb Waveform` object, which converts a `Coord` to a `Waveform` data type by specifying the number of points in the `Waveform`.

- **Comparator:** This object compares a test value against a reference value. Once again, you can compare a waveform to an array of coordinate pairs. The `Failures` pin gives you an array of the data points that failed, which you can send to the display and highlight with a different color or type of line.

- **TotSize:** This object simply gives you the number of elements in an array. Since this array contains the number of failures, dividing this by the total number of elements in the original waveform, 256, and multiplying by 100 gives us the percentage of failures.

- **Formula:** `A/256*100` is the formula used to compute the percentage of failures, since the `Function Generator` and `Noise Generator` are set to put out 256 points.

Using Strings and Globals

Manipulating Strings and Globals

Using string objects or functions, create a program that accepts a user's name in the following format: <*space*> <*firstname*> <*space*> <*lastname*>. After the user enters a name, have the program strip off the first name and only print the last name. Store the string into a global variable. Retrieve the string using the `Formula` object.

Solution—Manipulating Strings and Globals

Figure A-11 shows a solution to the exercise Manipulating Strings and Globals.

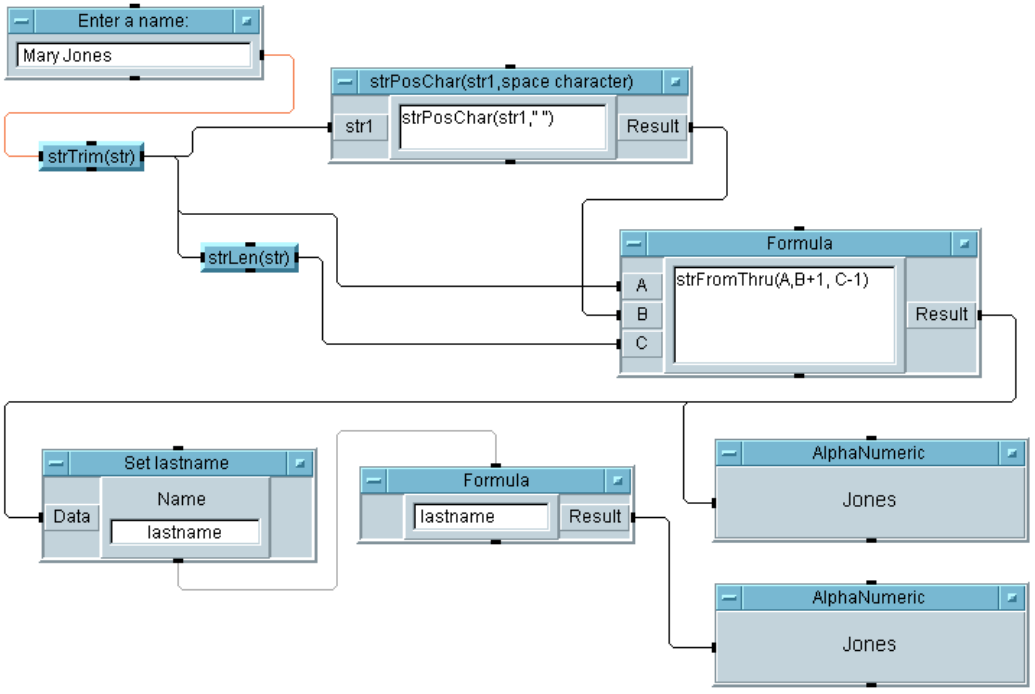


Figure A-11. Manipulating Strings and Global Variables

Key Points

- **String Objects and Functions:** `StrTrim(str)` first strips off any spaces or tabs from the front and back of the name. `StrPosChar(str1, " ")` yields the index of the space character between the `firstname` and `lastname`. `StrLen(str)`, of course, gives the length of the string. All of these were performed using the string objects, but they could also be done using string functions within a `Formula` object.

- **Formula Object:** `StrFromThru(A,B+1,C-1)` takes the string from input `A`, adds `1` to the index of the space from input `B`, and subtracts `1` from the string length at input `C`. (Recall that all indexing is zero-based.)

- **Set Variable:** Notice how easily you can set a global variable called `lastname`, which can then be referenced in any expression field, such as the `Formula` object in this example.

- **Optimizing:** The three formulas could be combined into one formula. It is recommended to leave `strTrim()` on its own since its output is used multiple times, but the others could be combined into one to optimize speed. (This could reduce readability, however.)

Optimizing Techniques

For this lab, you will build a VEE program two different ways and note the difference in execution speed.

Optimizing Techniques, Step 1

Create a program that sends the range `0 to 710 step 10` through both a sine function and cosine function. Put the results of the functions on an `X vs. Y` display. Use the `Timer` object to clock how long the program takes. (Set your default preferences for `Trig Mode` to `Radians`.)

Solution—Optimizing Techniques, Step 1

Figure A-12 shows a solution to step 1.

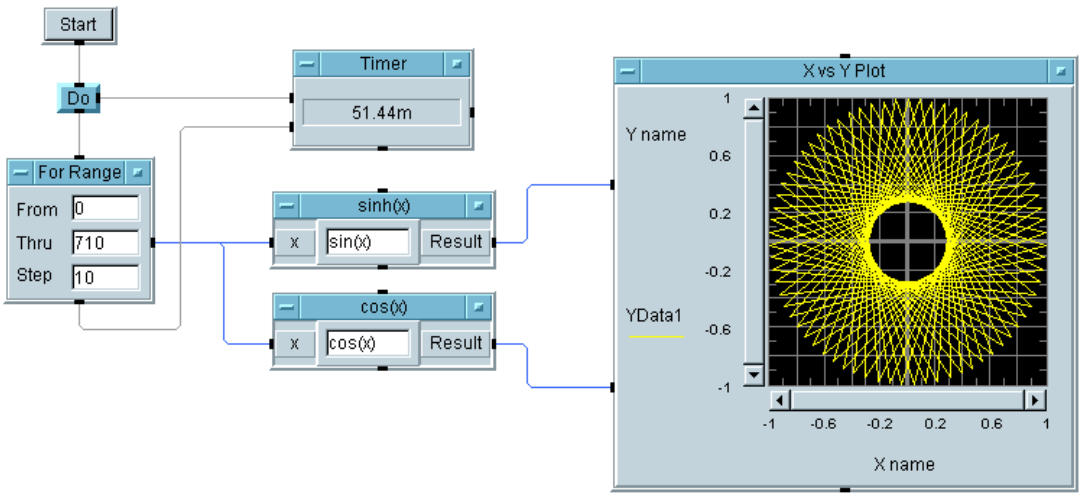


Figure A-12. Optimizing VEE Programs, Step 1

Optimizing Techniques, Step 2

Clone all of the objects from the first program. Modify the new set to collect the range into an array. Now, the `sine` and `cosine` functions are run against an array of points, and only plotted one time. Note the time savings.

Solution—Optimizing Techniques, Step 2

Figure A-13 shows a solution to step 2.

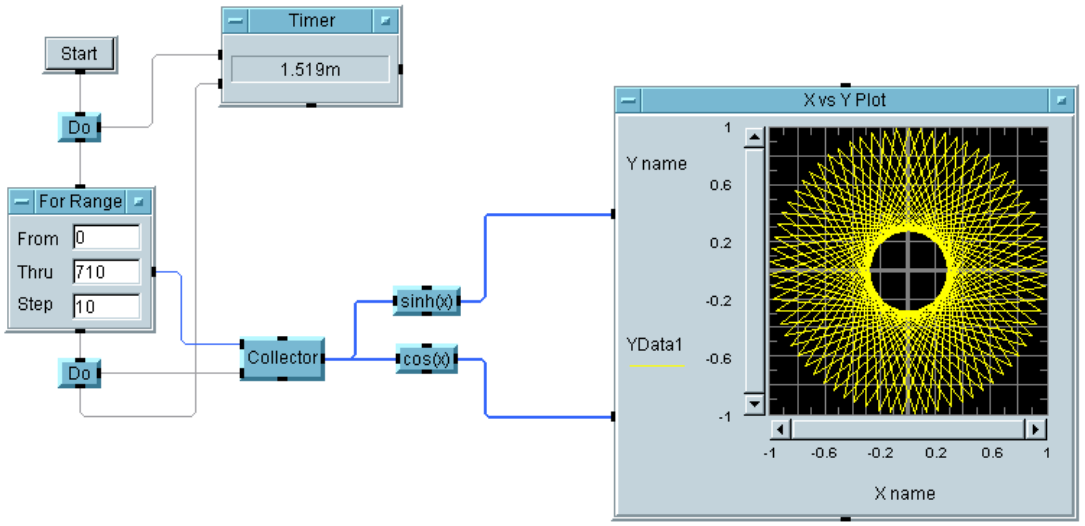


Figure A-13. Optimizing VEE Programs, Step 2

Key Points

- **Optimizing with Arrays:** Note the increase in performance between step 1 and step 2 that comes from using arrays. Whenever possible, perform analysis or display results using `arrays` rather than `scalar values`.

- **X vs. Y Display:** This example uses this display instead of the `Waveform` or `XY` displays, because there is separate data for the `X` and `Y` data.

UserObjects

Random Noise UserObject

Random Noise UserObject, Step 1

Create a `UserObject` that generates a random noise waveform. Display the noisy waveform and the noise spectrum outside the `UserObject`. Provide control outside the `UserObject` for the following: `amplitude`, `number of points`, `interval (time span)`, `DC offset`.

Note

Do not use a virtual source inside the `UserObject`. Use objects such as `Build Waveform` and `Random` to create the `UserObject`.

Solution—Random Noise UserObject

Figure A-14 shows a solution for the `Random Noise UserObject`.

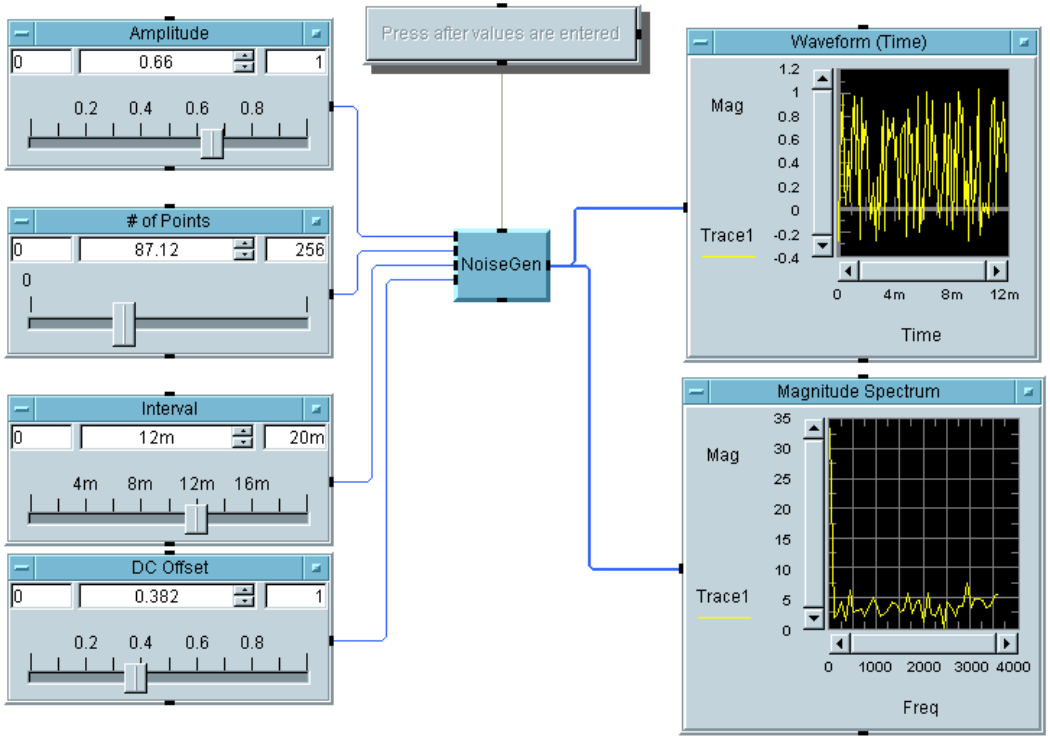


Figure A-14. A Random Noise UserObject

Solution—NoiseGen Object in Random Noise

Figure A-15 shows a solution for the `NoiseGen UserObject`.

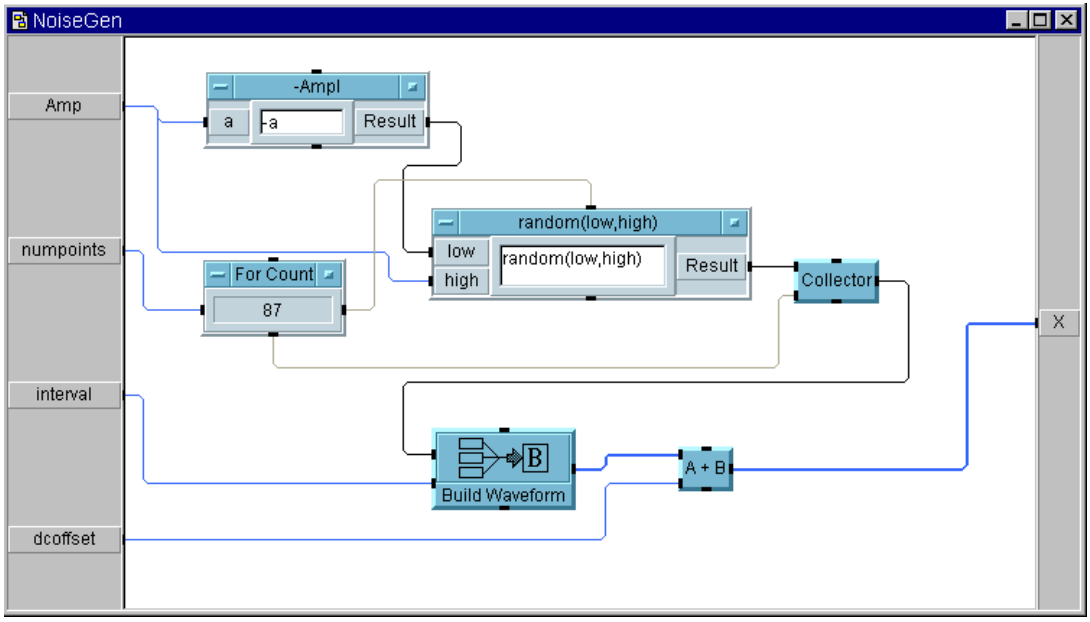


Figure A-15. The NoiseGen UserObject

Key Points

- **UserObject:** Notice that the `UserObjects` you build are essentially customized objects that you add to VEE.

- **Build Waveform:** This object creates a `Waveform` data type from a `Real` array of amplitude values and a `time span` (the length of time in seconds over which the `y` data was sampled).

Agilent VEE UserFunctions

Using UserFunctions

UserFunctions, Step 1

Create a function called `NoiseGen` that accepts an amplitude value (0-1) from a slider and returns a noisy waveform.

Do Not Use
Virtual Source
For Count
For Range

Do Use
Formula
Ramp
Build Waveform

Hint

Use `randomize(array, -a,a)` where the array must be 256 points, and `a` is the amplitude. Build a simple main program that calls this function to be certain the function works correctly.

Solution—UserFunctions, Step 1

Figure A-16 shows a solution for step 1.

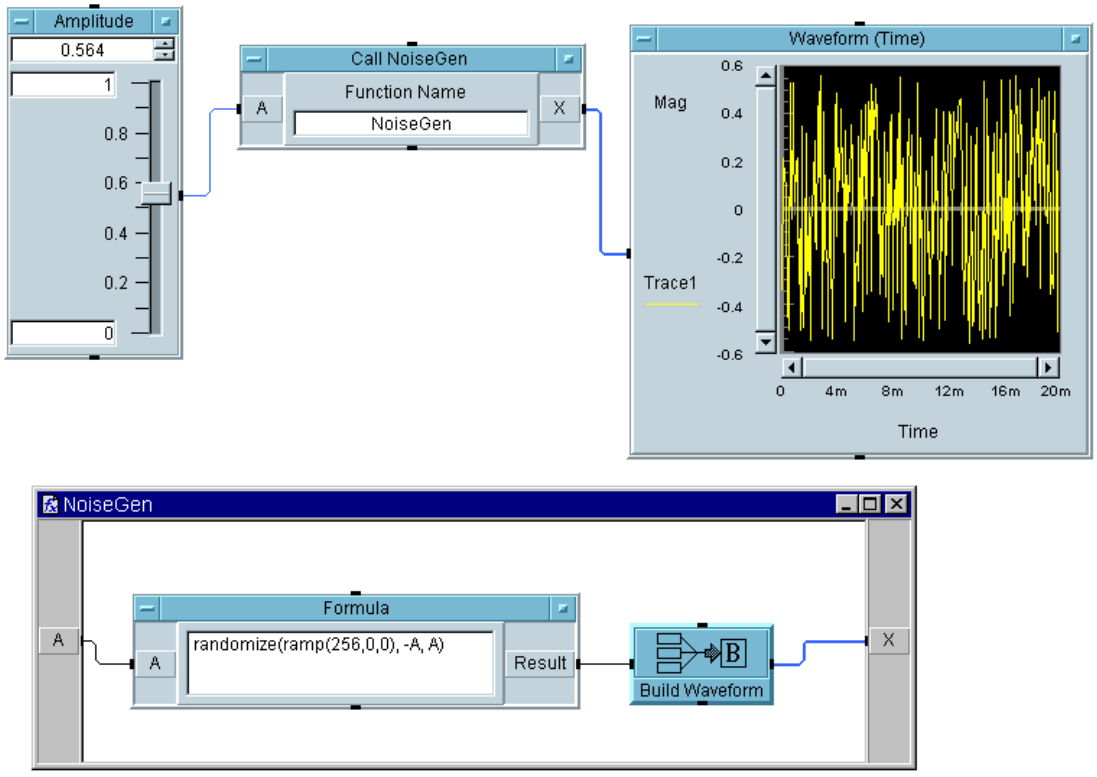


Figure A-16. User Functions, Step 1

Key Points

- **Ramp():** Notice that the `ramp()` function is used to generate an array of 256 points within the parameter list for `randomize()`.
- **Build Waveform:** Notice that the default time span is 20 milliseconds, so that you only need to send an array to this object to build a waveform.

UserFunctions, Step 2

In the same program, create another function called `AddNoise` that calls the first function `NoiseGen`. `AddNoise` should add the noisy waveform from the `NoiseGen` function to a sine wave. `AddNoise` should have two inputs, one for the NoiseGen amplitude and one for the sine wave. It should have one output for the result.

Build a simple main program with a slider for the noise amplitude, and the `Virtual Source` ⇒ `Function Generator` (`sine wave`, `Freq = 100 Hz`) for the good waveform to add to the noise. Display the resultant waveform.

Solution—UserFunctions, Step 2

Figure A-17 shows a solution for step 2.

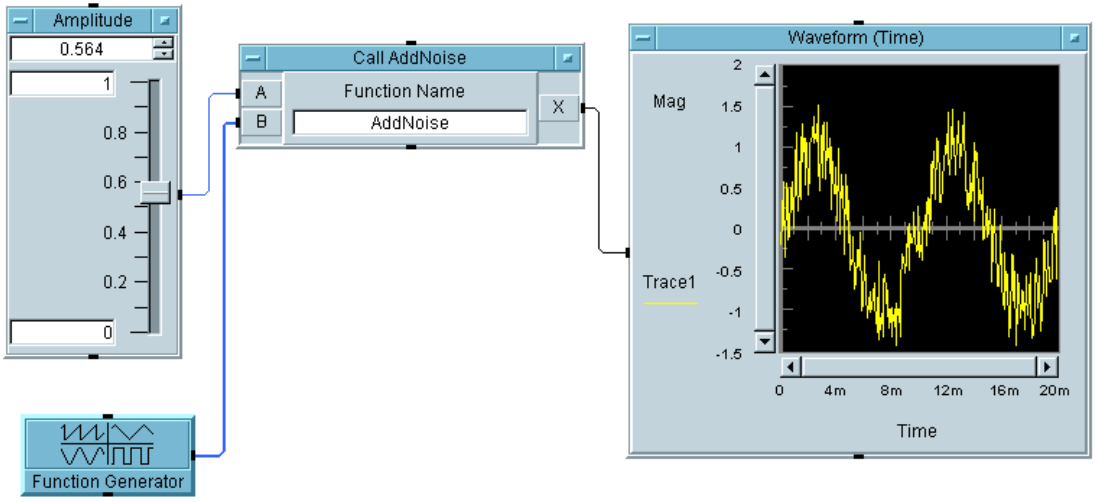


Figure A-17. User Functions, Step 2

UserFunctions, Step 3

In the same program, call the `AddNoise` function again, this time from a `Formula` object, taking the absolute value of the result. Display the absolute value waveform on the same display. Next prepare to edit the `AddNoise` function. Turn on `Debug` ⇒ `Show Data Flow`. Leave the `AddNoise` window open and run the program. Notice how useful this capability is for debugging.

Solution—UserFunctions, Step 3

Figure A-18 shows a solution for step 3.

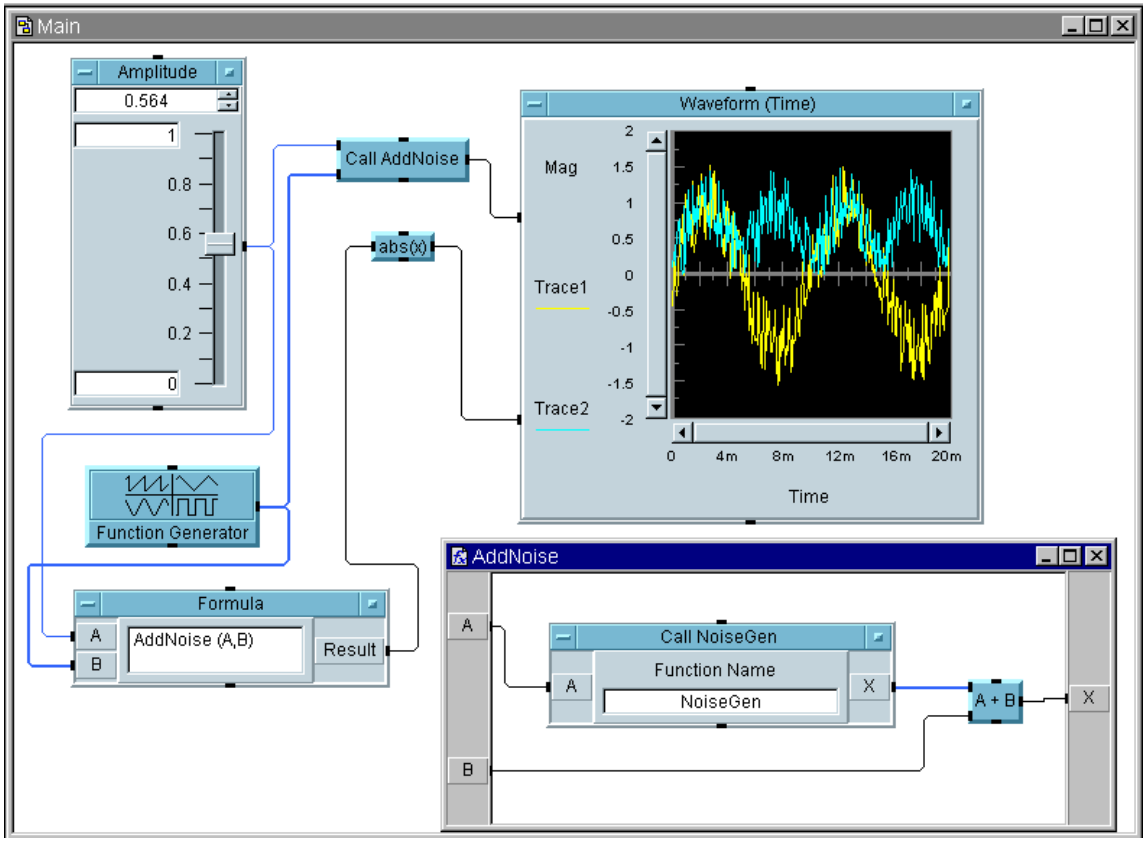


Figure A-18. User Functions, Step 3

UserFunctions, Step 4

Now change the program so that the slider sets a global variable called `Amplitude`. Have the `NoiseGen` function use that global (so NoiseGen will no longer require an input pin). Make the program run correctly. Save this file as `uflab.vee`.

Solution—Using UserFunctions, Step 4

Figure A-19 shows a solution for step 4.

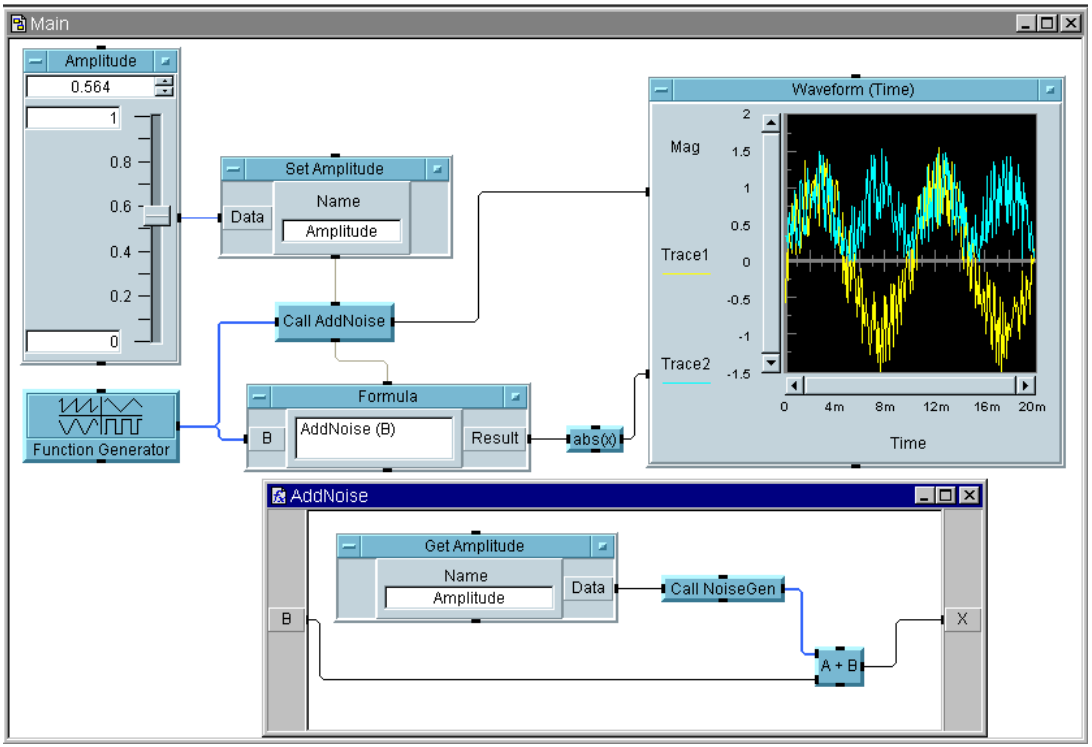


Figure A-19. User Functions, Step 4

Hint: Notice the `Call AddNoise` and `Formula` objects use the global `Amplitude`, so both of the objects need to run after the `Set Amplitude` object executes. Connecting the Sequence pins from `Set Amplitude` to `Call AddNoise`, and `Call AddNoise` to `Formula` ensure the objects execute in the required order.

Importing and Deleting Libraries of UserFunctions

Build a simple program to import the `uflab.vee` functions from the previous exercise. Call the function that adds the noise, and then delete the library programmatically. Use the `Select Function` choice in the object menu of the `Call` object.

Hint: Click on `Load Lib` in the `Import Library` object menu to manually load the library you specified, so that you can use the `Select Function` feature in `Call`.

Solution—Importing and Deleting Libraries Programmatically

Figure A-20 shows a solution for deleting the library programmatically.

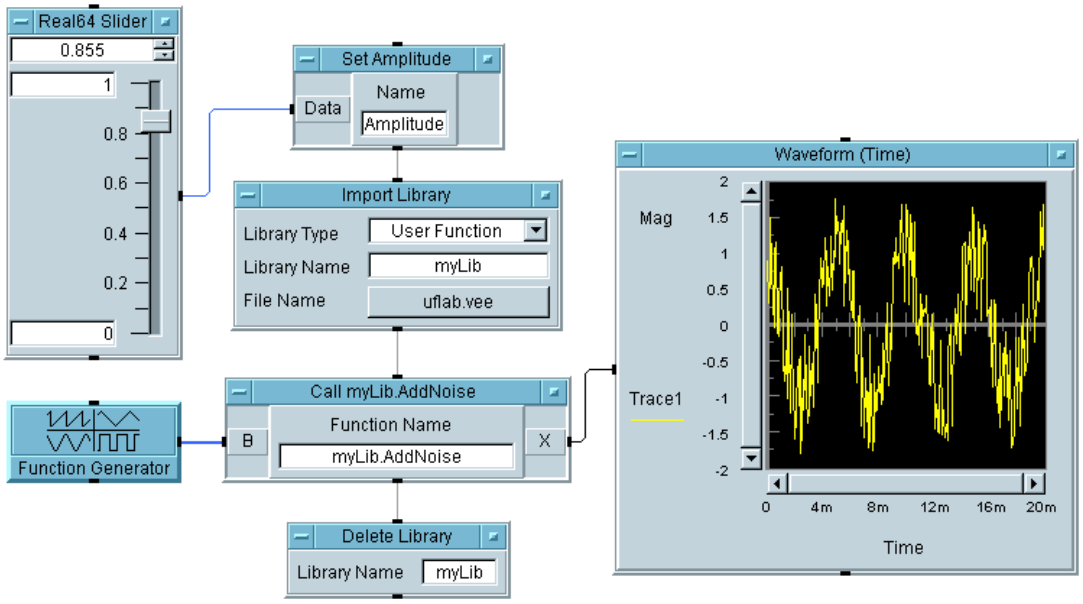


Figure A-20. Importing and Deleting Libraries

Key Points

- **Select Function:** Notice that this selection will configure the proper input and output pins for the function you select.
- **Editing UserFunctions:** If you import a library of `UserFunctions` programmatically, you will not be able to edit them. You can view them and set breakpoints to debug. If you want to edit the `UserFunctions` you import, use the `Merge Library` command.
- **Set Variable Caution:** Notice that when you use a global variable in a function, you have to remember to create that global when using the function in other programs. One of the advantages of explicitly creating inputs and outputs is that they are easier to track.

Creating Operator Panels and Pop-ups

Creating Operator Panels and Pop-ups, Step 1

Create a panel to ask an operator to enter numbers. Create a `UserObject` to interact with an operator. Ask the operator for 2 inputs, `A` and `B`. Send both inputs to a display. Use a `UserObject` with `Show On Execute` checked to display the panel.

Solution—Creating Operator Panels and Pop-ups, Step 1

Figure A-21 shows a solution in detail view. Figure A-22 shows the panel that appears when the program runs.

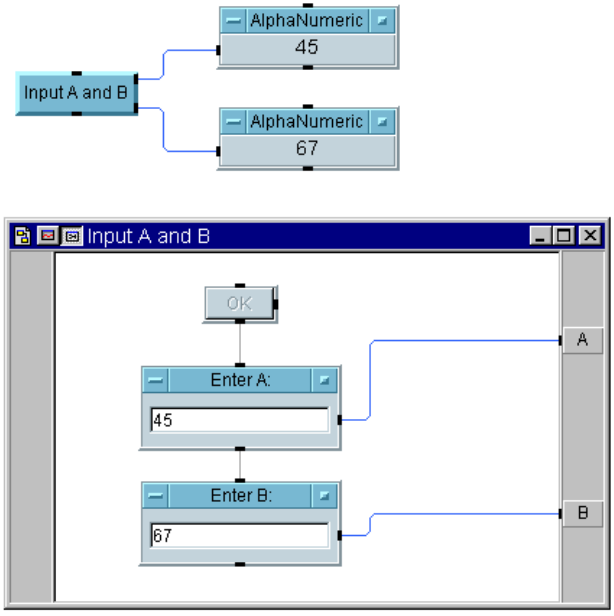


Figure A-21. UserObject to Ask Operator to Input A and B

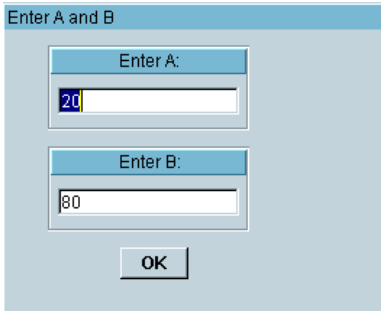


Figure A-22. Panel for Operator to Enter A and B

Key Points

- **UserObject Properties:** In the `UserObject Properties` dialog box, select `Pop-Up Panel` and click to turn on `Show Panel On Execute`. Change the `Pop-Up Panel` ⇒ `Panel Title` name to "`Enter A or B`."

Creating Operator Panels and Pop-ups, Step 2

Instead of displaying both `A` and `B`, ask the operator whether to display `A` or `B` if the two numbers are different. After asking for the two values, if the values `A` and `B` are equal, display the value. If the two values `A` and `B` are different, ask the operator to pick a value to display. Display `A` or `B` depending on the operator's choice.

HINT: Add another `UserObject` with a pop-up panel that is set to `Show Panel on Execute`, and ask the operator for the value there.

Solution—Creating Operator Panels and Pop-ups, Step 2

Figure A-23 shows the `UserObject` that asks the operator to make a choice when `A` and `B` are different numbers. Figure A-24 shows the second pop-up panel that appears to ask the operator whether to display `A` or `B`.

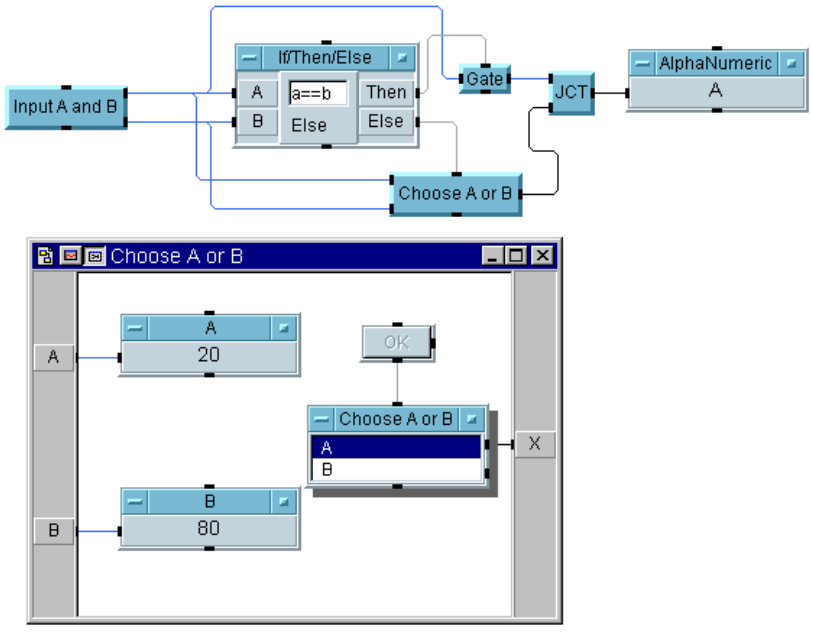


Figure A-23. UserObject to Ask Operator Whether to Display A or B

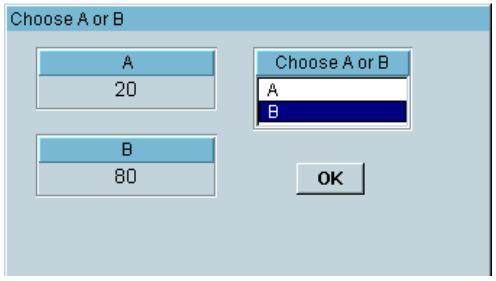


Figure A-24. Panel for Operator to Choose Whether to Display A or B

Key Points

- **Gate:** The `Gate` object only sends a value if the two numbers are equal.
- **Junction:** The `JCT` object allows multiple inputs to the object `Alphanumeric`. The `JCT` object is a “wired OR” object.
- **List Object as a Menu:** Note the use of the `Data ⇒ Selection Controls ⇒ List` object edited for two choices and formatted for a list. This configuration will output a text `A` or `B`. If you need the ordinal value (`0` or `1`), then use the `List` object's ordinal data output instead.

Creating Operator Panels and Pop-ups, Step 3

If the operator does not enter numbers, generate an error message. On the second `UserObject`, which asks the operator to choose whether `A` or `B` is displayed when the two numbers are different, add an error. If the operator does not choose `A` or `B` within 10 seconds, generate the error.

Solution—Creating Operator Panels and Pop-ups, Step 3

Figure A-25 shows the `UserObject` modified to generate an error if the operator does not choose `A` or `B` in 10 seconds.

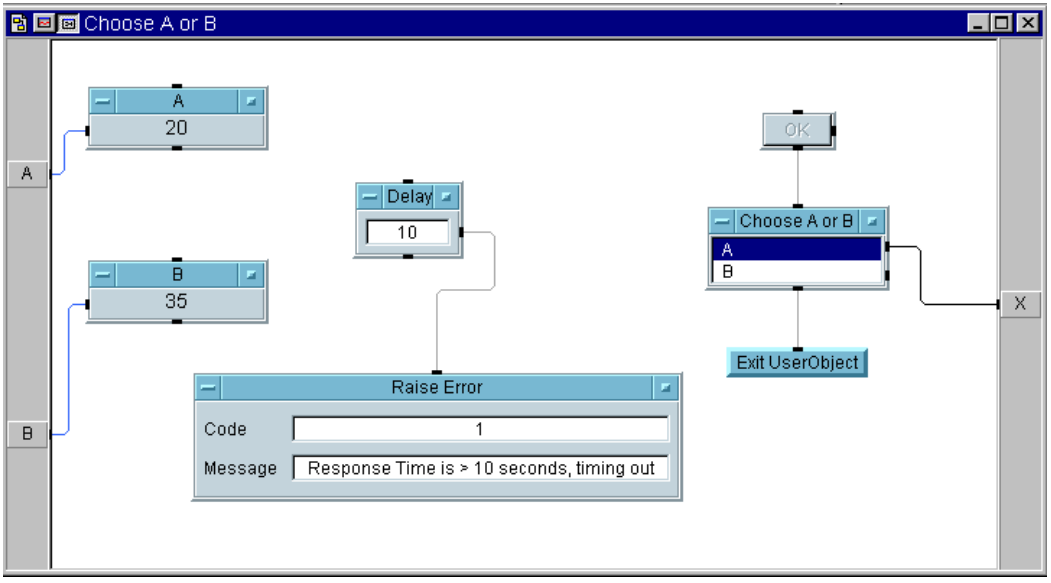


Figure A-25. Generate an Error if Operator Does Not Enter a Choice

Key Points

- **Exit UserObject:** If the user responds in under 10 seconds, this object will exit the `UserObject`, even though the `Delay` object may not have finished executing.

- **Delay and Raise Error:** After 10 seconds the `Delay` object pings the `Raise Error` object, which will pause execution of the program and display the `Error Message` you have typed in. A red outline will also appear around the object that caused the error, which goes away when you click on the **Stop** or **Run** buttons on the main menu bar.

- **OK and Delay:** Notice the two threads in `AnotB` are separate, so that the `OK` and `Delay` are both running concurrently.

Working with Files

Moving Data To and From Files

Create a VEE program to write the time of day to a file. Generate 100 random points and write them to the file. Calculate the mean and standard deviation of the numbers and append them to the file in the following format:

Mean: xxxxxx
Std Dev: yyyyyy

Next, read only the mean and standard deviation from the file. Figure A-26 shows moving data to and from files.

Solution—Moving Data To and From Files

Figure A-26 shows a solution for moving data to and from files.

.

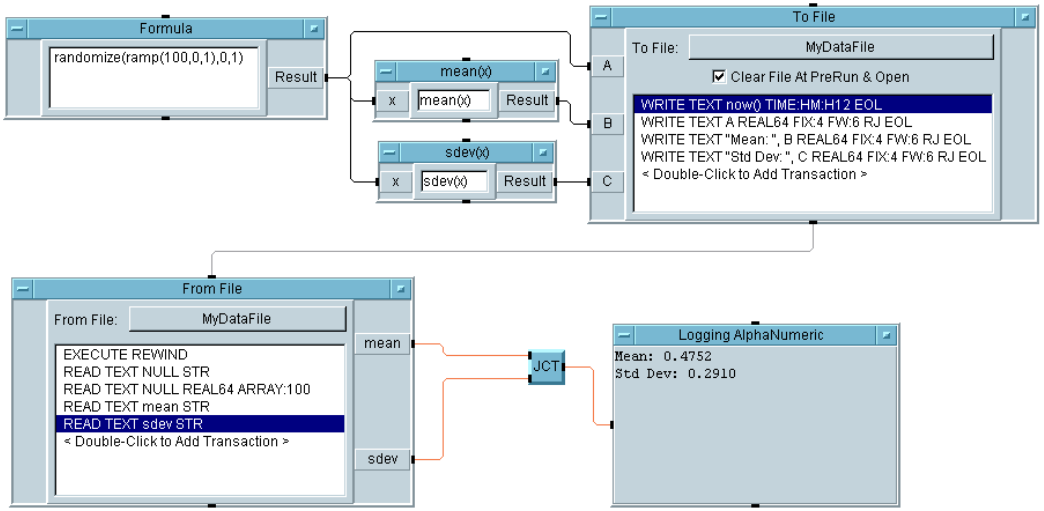


Figure A-26. Moving Data To and From Files

Key Points

- **Generating an Array:** Use `randomize(ramp(100,0,1), 0, 1)` in the `Formula` object to create an array of 100 random numbers. The `ramp()` function generates an ordered array and delivers it to the `randomize()` function, which then generates random values between `0` and `1`.

- **Time Stamp:** The `now()` function is used in the expression field of the `I/O Transaction` dialog box for transaction one in the `To File` object. When you change the format to `TIME STAMP FORMAT`, the dialog box displays additional buttons to specify how the time will be stored.

- **Storing Two Values in a Line:** In both the third and fourth transactions in the `To File` object, a constant `Text` string is stored, followed by a `Real` value. For example, in the third transaction you type `"Mean: ",B` in the expression field of the `I/O Transaction` box (assuming the mean value will be on the `B` input pin).

- **Extracting a Value From a File:** To get to the mean and standard deviation, first send an `EXECUTE REWIND` to position the read pointer at the beginning. Then use `NULL` with the proper format to `READ` past the time stamp and real array. (This will throw away the values read instead of putting them in an output terminal.) Finally, read the last two lines in the file as strings.

- **Junction:** Use the `Flow ⇒ Junction` object to connect more than one output to a single input, such as connecting the `mean` and `sdev` outputs to the `Logging AlphaNumeric` display.

Records

Manipulating Records

Manipulating Records, Step 1

Build a record with three fields holding an integer, the time right now as a string, and a four element array of reals. The fields should be named `int`, `daytime`, and `rarry`, respectively. Merge this record with another that holds a random number between 0 and 1, and a waveform. Name these fields `rand` and `wave`.

Solution—Manipulating Records, Step 1

The resulting record should have five fields, as shown in Figure A-27.

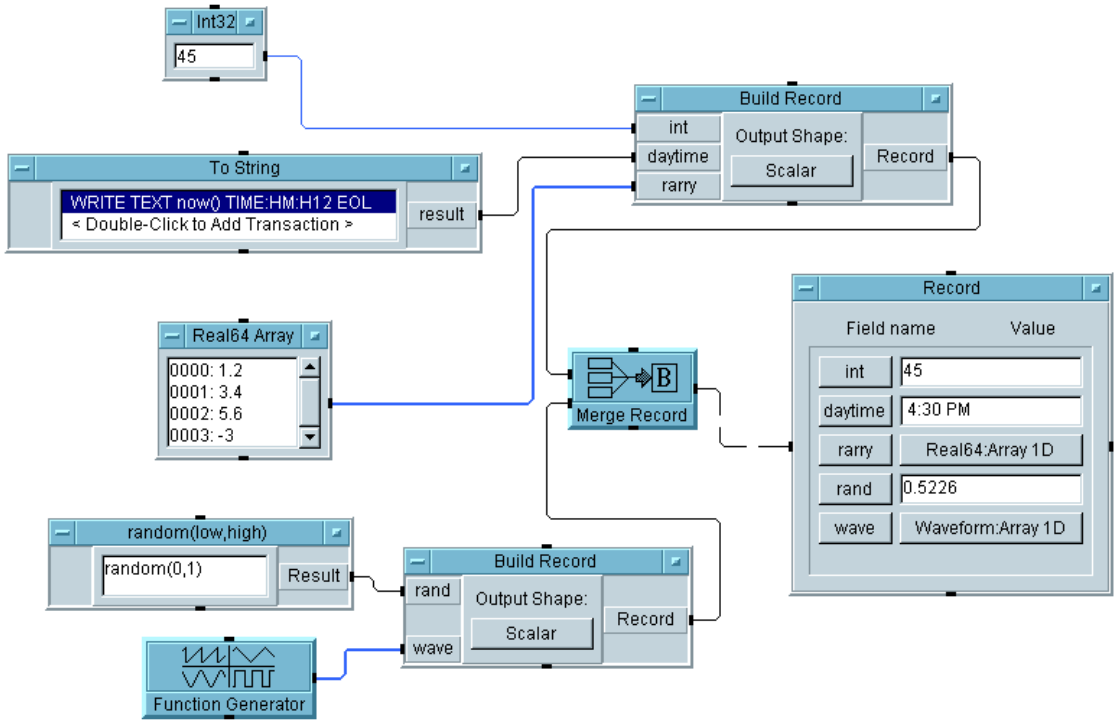


Figure A-27. Manipulating Records, Step 1

Key Points

- **Time Stamp:** Use the `now()` function within the `To String` object to create your time stamp for this program. Then you can specify the format.

- **Configuring a Data Constant as an Array:** Any data type in the `Data ⇒ Constant` menu can become an array by selecting `Properties`, then under `Configuration` choose `1D Array`. The size may be entered here, or as you are typing in the values an **Enter** will keep appending values.

- **Naming Fields:** By renaming the input terminals on the `Build Record` object, you can give your record specific field names such as `int`, `rand`, and `wave`.

- **The Default Value Control Input:** A `Record Constant` makes an excellent interactive display object by adding a `Default Value Control` pin. The `Record Constant` will automatically configure itself for the record it receives.

Manipulating Records, Step 2

Use a conditional expression in a `Formula` object to test the random value in the record, and display either the value or a text string. If the value is less than 0.5, display that random value; otherwise, output a text string "`More than 0.5`." Next, extract only the time and the waveform.

Hint

Do not use a `Formula` object to extract the time and waveform. Display this record with an `AlphaNumeric` object.

Solution—Manipulating Records, Step 2

Figure A-28 shows manipulating records, step 2.

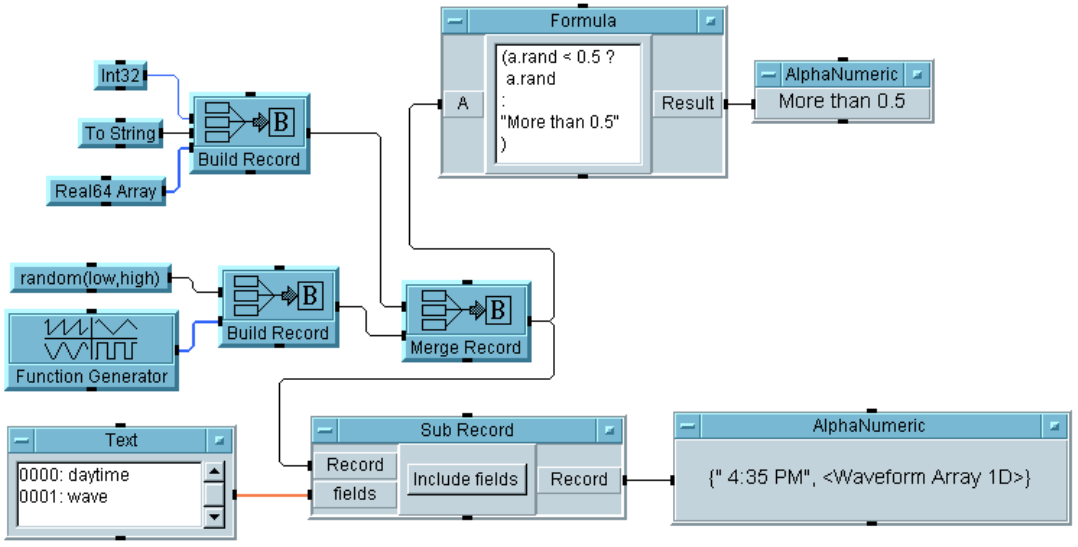


Figure A-28. Manipulating Records, Step 2

Key Points

- **Using a Conditional Expression:** VEE supports a conditional expression, which provides an efficient way to implement an if-then-else action. The conditional expression in this `Formula` object is known as a triadic. It is `(a.rand < 0.5? a.rand: "More than 0.5")`. Notice that it is all one expression, and you can write it with line breaks in the `Formula` object as shown. If there were more than one expression in the `Formula` object, the expressions would be separated with semi-colons (`;`).

- **The Sub Record Object:** Notice the `Text` array of the fields on the `Sub Record` input pin labeled fields. When you configure the `Sub Record` object to include fields, it will output a record that only contains the fields specified.

Manipulating Records, Step 3

Replace the integer input for the first field with a `For Count` object and step through 10 iterations. Be certain to "ping" the random number generator and the time function on each iteration. Send the complete record into a `To DataSet` object. In a separate thread, retrieve all records from the dataset where the random value is greater than 0.5. Put the resultant records into a record constant

Hint

You'll need a control pin for a `Default Value` on the `Record Constant` object.

Solution—Manipulating Records, Step 3

Figure A-29 shows a solution for manipulating records, step 3.

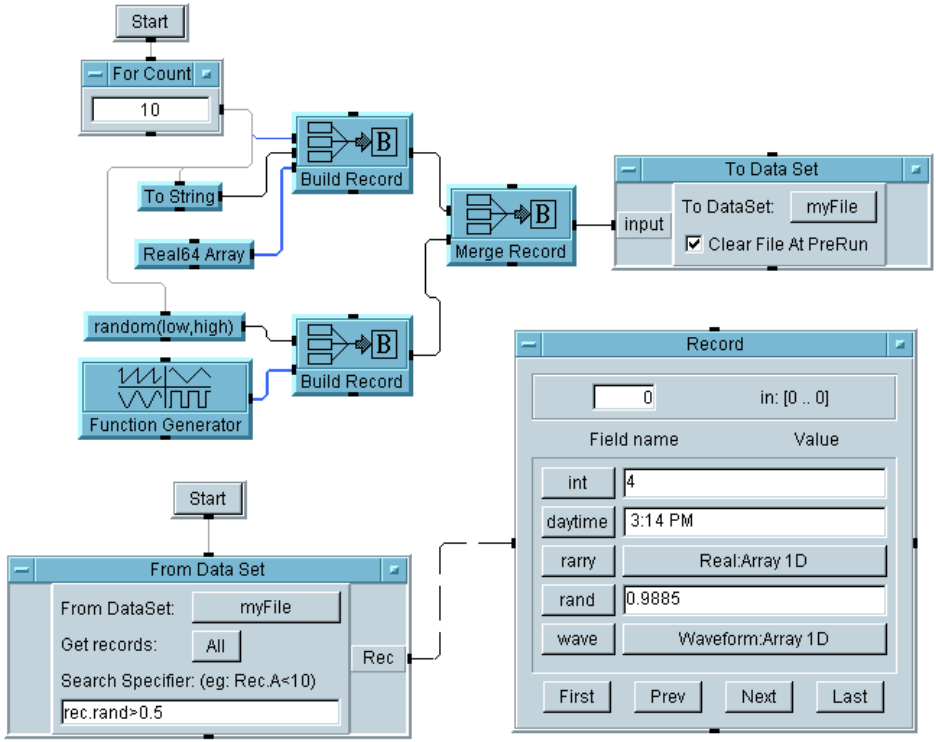


Figure A-29. Manipulating Records, Step 3

Key Points

- **The To DataSet Object:** The `Clear File at PreRun` option only clears the file before data is sent the first time. Notice that the program sends 10 different records to the same file sequentially, and they are appended to the file.

- **The From DataSet Object:** This object is configured to retrieve all records where the `rand` field is greater than 0.5. In this case, five out of ten records meet that criterion and the first record is shown with an index number of 0.

Test Sequencing

Using the Test Sequencer, Step 1

Create a simple `UserFunction` called `UpperLimit` that is a pop-up panel with a `Real64 Slider` and a `Confirm (OK)` object. Send the output of the slider to a global variable called `UpLimit` and also to an output terminal. Create a `Sequencer` object, and configure `test1` in the `Sequencer` as an `EXEC` transaction that calls `UpperLimit`.

Create another function called `AddRand` that simulates the test you might call. This function should add an input value to a random value (`0` to `1`). It will have one input pin and one output pin.

From the `Sequencer`, create `test2` to call `AddRand` and send in a zero. Test the return value to do a limit comparison less than the global `UpLimit` value. If it passes, then return `"PASS" + test2.result`. If it fails, return `"FAILED" + test2.result`. Put an `Alphanumeric` display on the `Return` pin of the `Sequencer`.

After the `Sequencer` object, ping a `Get Variable` object (`UpLimit`) and another `Alphanumeric` display. Run the program several times.

Solution—Using the Test Sequencer, Step 1

Figure A-30 shows a solution for the first step of using the `Sequencer`.

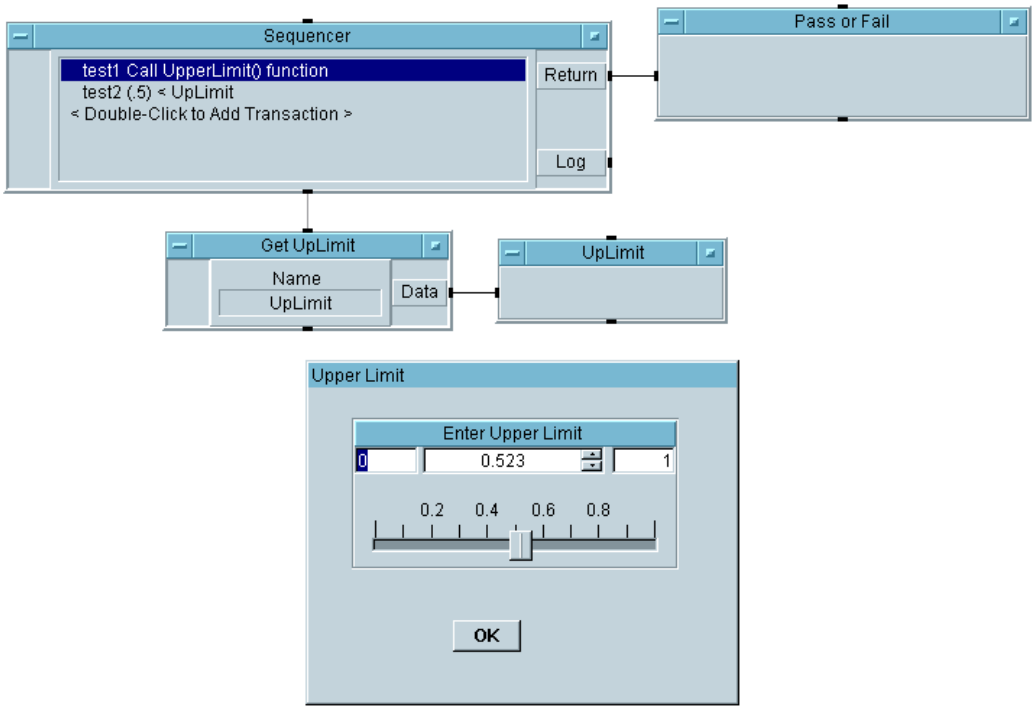


Figure A-30. Using the Sequencer, Step 1

Key Points

- **Setting Global Variables with a UserFunction:** A typical use of the first `Sequencer` transaction is to call a `UserFunction` that sets the `Global Variables`, as it does in this case. Then you can utilize these variables in any test that follows, as is shown here.

- **The Sequencer Return Pin:** The `Return` pin in this example delivers a PASS or FAIL message plus the test value. You could use this pin to deliver any message or value from a particular test.

Using the Test Sequencer, Step 2

Disable the first test. Assuming that you do not need the global anywhere else, you can call the `UpperLimit` function directly. Change `test2` so that it compares the return value of `AddRand(0)` against the result of the `UpperLimit` function.

Hint: For disabling the first test, use the `Sequencer Transaction` box as shown in Figure A-31.

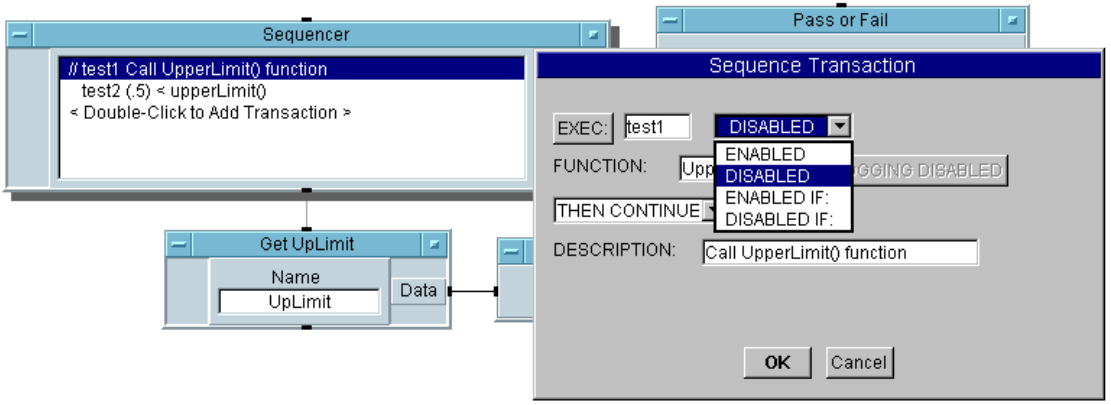


Figure A-31. Disable the First Test in the Sequence

Note that in Figure A-31, the first test in the `Sequencer` is “commented out” with two slashes to show that it is disabled.

Solution—Using the Test Sequencer, Step 2

Figure A-32 shows a solution to using the test `Sequencer`, step 2.

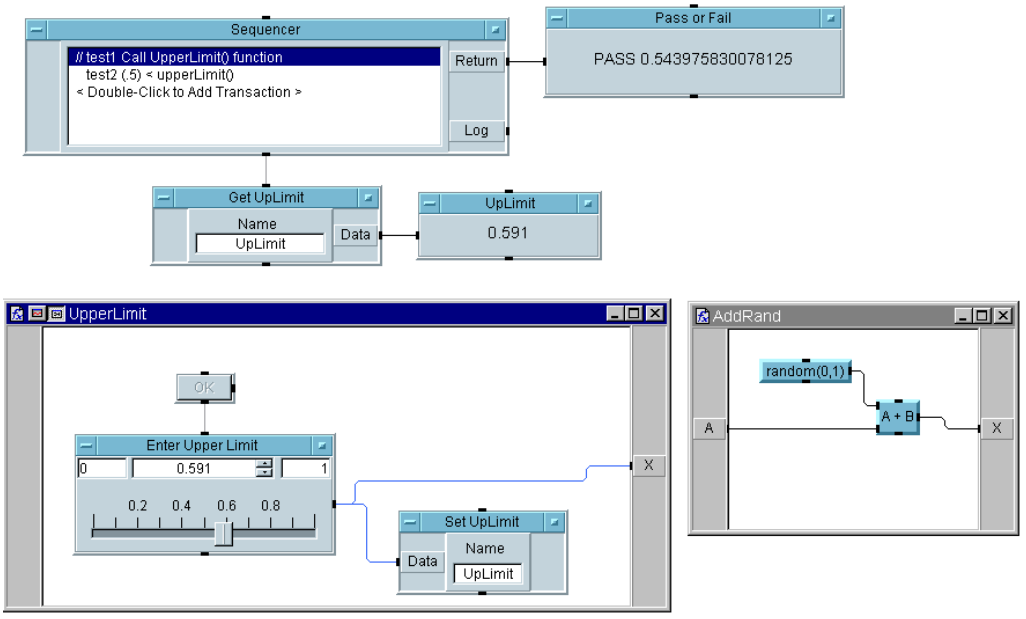


Figure A-32. Using the Sequencer, Step 2

Key Points

- **The UserFunction in an Expression Field:** In this example, instead of comparing a test result to the `UpLimit` variable, you can type the function name `UpperLimit()` in the expression field where the variable would go.

Using the Test Sequencer, Step 3

Edit the `test2 Sequencer` transaction that calls the VEE function `random (0,1)`. Compare the result against a limit less than `0.5`. Cut and paste the `test1` transaction until you have a total of four tests.

Build a program to run the `Sequencer` five times. Record the data in a data set of records and collect the data in an array. Using the array, find the minimum, maximum, mean, and standard deviation of the results of the second test.

Solution—Using the Test Sequencer, Step 3

Figure A-33 shows a solution to Step 3.

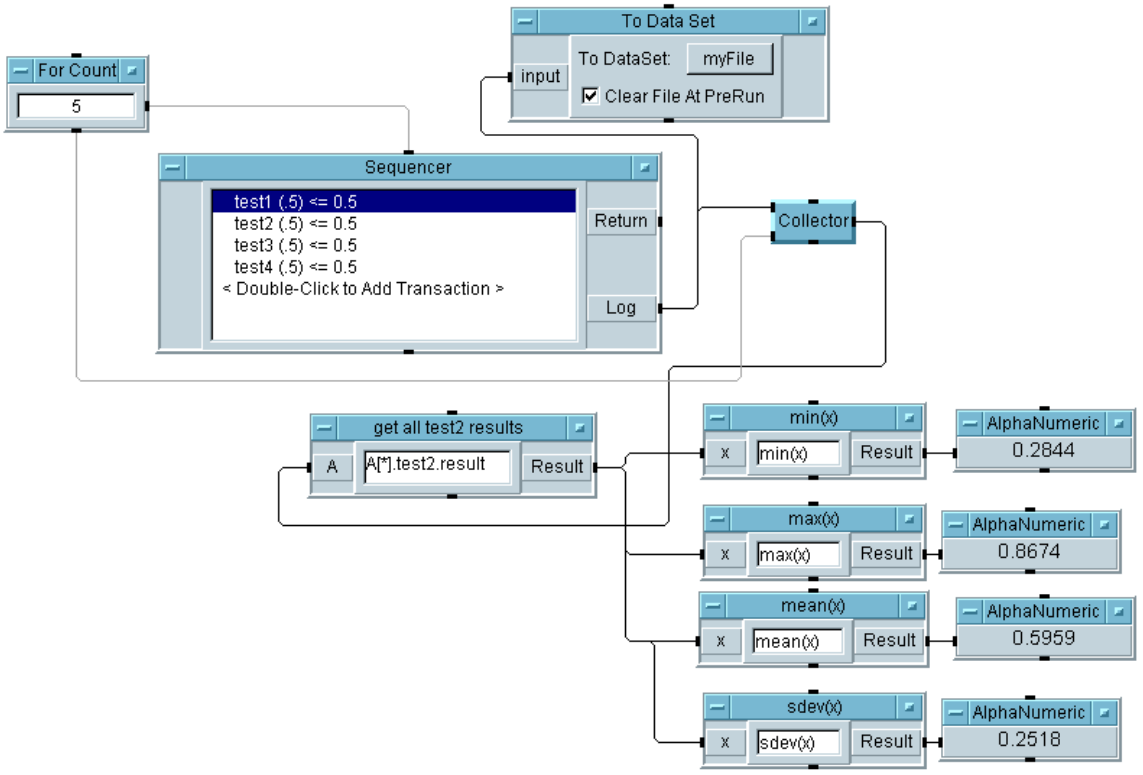


Figure A-33. Using the Test Sequencer, Step 3

Key Points

- **The Data Format for Several Runs of the Sequencer (First Thread):** When the `Sequencer` executes once, it outputs a `Record of Records`. The first record has field names that match the test names, then each field holds a record containing the different pieces of data for that particular test. When the `Sequencer` runs several times, each `Record of Records` can be added to an array, which can then be investigated. If you use the *<record>[*].<record>.<field>* format in the

`Formula` object, you will get an array of data. In this case, you get an array of real values giving the test results for five runs of `test2`. You can then calculate the minimum, maximum, mean, and standard deviation from this array. You could specify a single run of `test2` by indicating a particular element in the array of records of records. For example, to get the first run result of `test2` you would use the expression: `A[0].test2.result`.

Using the Test Sequencer, Step 4

Add a timestamp field to the logging record. Add a delay so that each step runs one second apart. In a separate thread, get all the results of `test2` and send them to a record constant.

Hints

- **The Delay Object (First Thread):** This object holds execution flow for the specified number of seconds. Here it is used to ensure the time stamp values vary between each run of the `Sequencer`.

- **Adding a Time Stamp:** To add a time stamp, open the `Sequencer` object menu and select `Properties` ⇒ `Logging` tab to check `Record Fields to Log` ⇒ `Time Stamp`. Figure A-34 shows the `Properties` ⇒ `Logging` tab dialog.

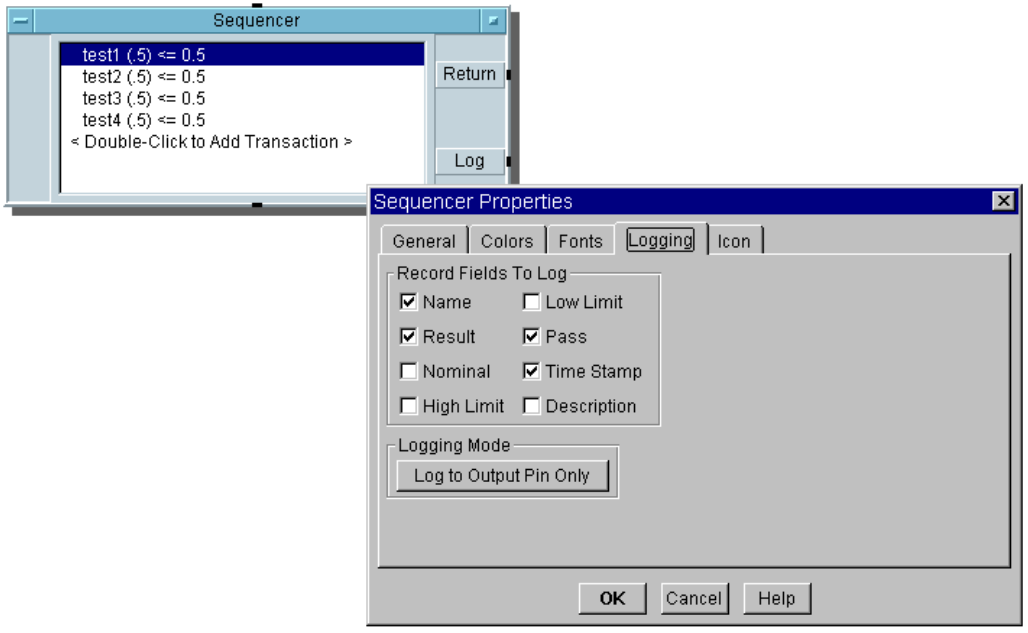


Figure A-34. Add a Time Stamp to the Logging Record

Solution—Using the Test Sequencer, Step 4

Figure A-35 shows a solution to step 4 of using the test sequencer.

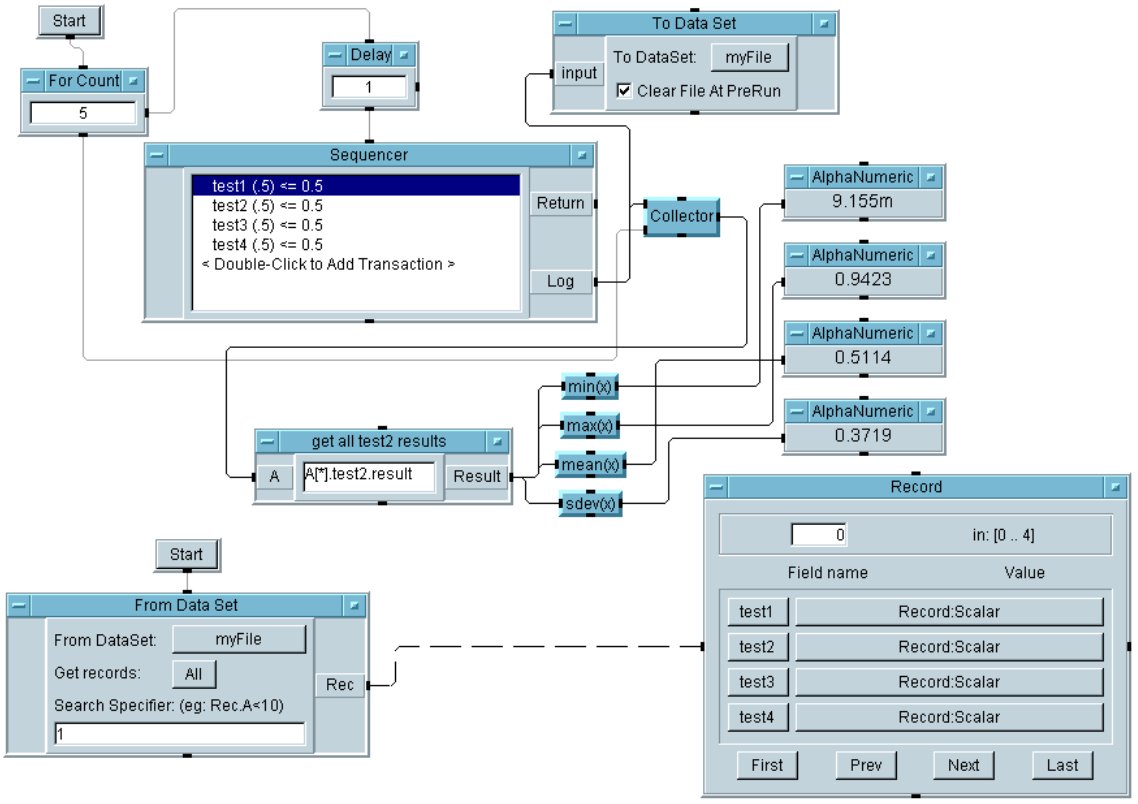


Figure A-35. Using the Test Sequencer, Step 4

Hint:

To display a record, click on the `Record` ⇒ `Record: Scaler` field for one of the tests and the `Record Field Data` dialog box appears. Figure A-36 shows the `Record Field Data` dialog box.

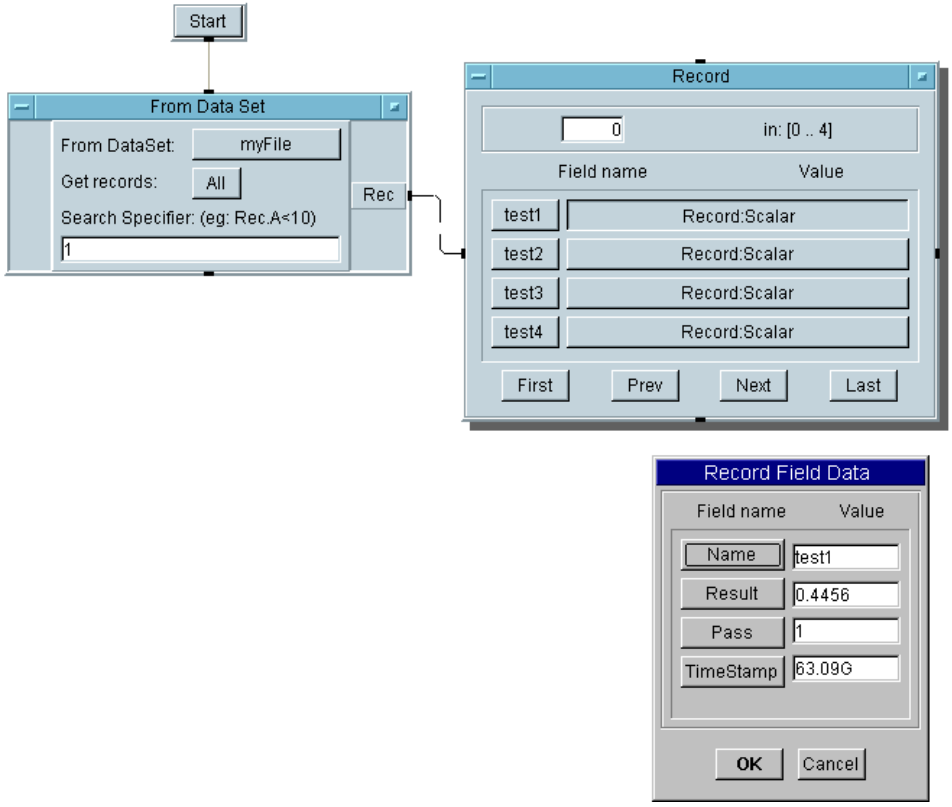


Figure A-36. Checking a Record

Using the Test Sequencer, Step 5

Print the time stamp fields from the records on a `Logging Alphanumeric` display.

Hint: Use four `Formula` objects (one for each test). To show all four `Formula` results in one `Logging Alphanumeric` display, add a `Junction` object. Use a `To String` to format the 63G time stamp value into a more readable string.

Solution—Using the Test Sequencer, Step 5

Figure A-37 shows the program thread to print the time stamps to a display, step 5 of using the test sequencer.

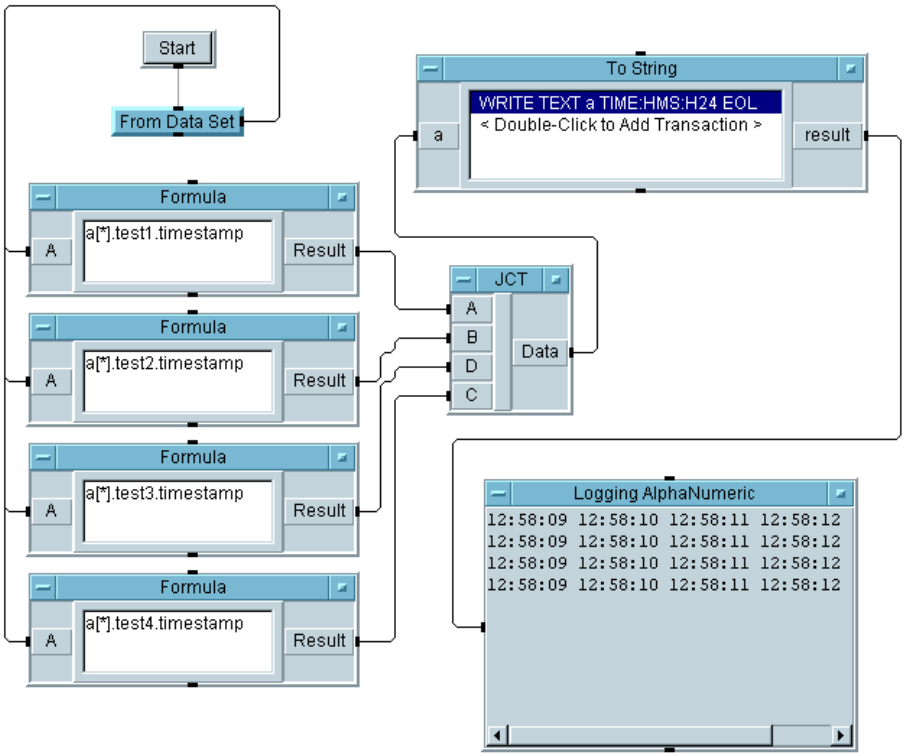


Figure A-37. Using the Test Sequencer, Step 5

Key Points

- **Converting Time Stamp Formats:** The `To String` object before `Logging AlphaNumeric` converts the time stamps from a `Real` format to a `Time Stamp` format for more clarity.

Using the Test Sequencer, Step 6

If the `Sequencer` includes many tests, it can become cumbersome to use many individual `Formula` objects connected to a `Junction`. Instead, you can use a `Formula` that contains an expression, generate the expression at run time, and loop through the possible expressions.

First the example will generate the expression strings.

In a separate thread, use a Loop and a `Formula` to generate a test expression string. Output the information as a string in a `Logging Alphanumeric`. The string generated in the `Formula` should be `"a[*].test<x>.timestamp"` where `<x>` goes from 1 to 4.

Solution—Using the Test Sequencer, Step 6

Figure A-38 shows a solution for step 6.

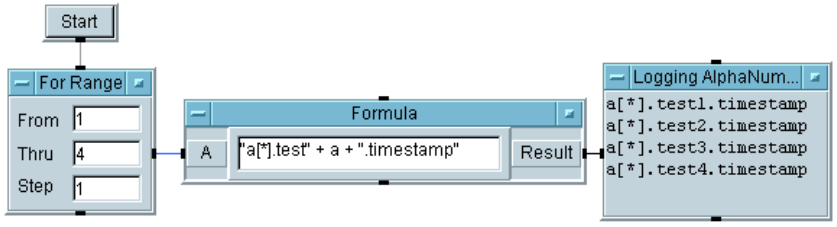


Figure A-38. Using the Test Sequencer, Step 6

Using the Test Sequencer, Step 7

Now take the Loop and the `Formula` you built in Step 6, and replace the four `Formulas` and `Junction` in the previous step with the Loop and `Formula`. Plus, you now want to evaluate the string you built. Send the string generated (the expression `"a[*].test<x>.timestamp"`) into a `Formula` to be evaluated at runtime.

Hints

- **Formula Control Pin on the Formula Object:** The `Formula` you want to evaluate is generated by the `Formula` inside the Loop. You can create a second `Formula` box with a control input for its `Formula` expression. The expression the second `Formula` evaluates is generated at runtime.

Solution—Using the Test Sequencer, Step 7

Figure A-39 shows a solution to step 7.

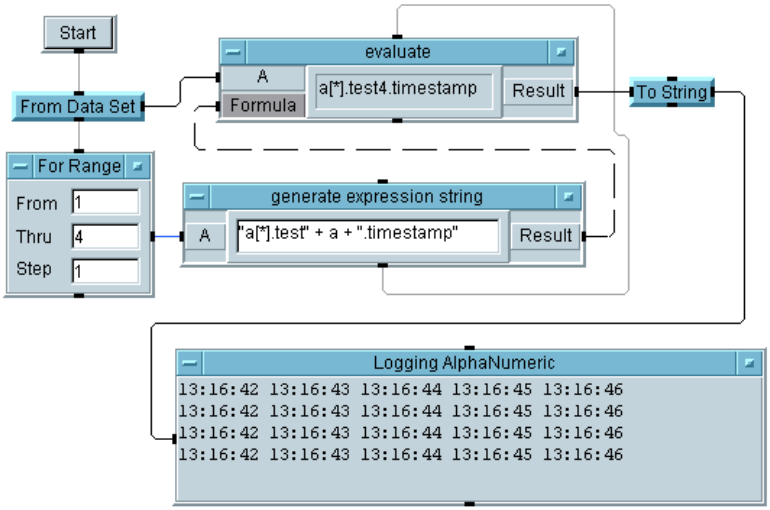


Figure A-39. Using the Test Sequencer, Step 7

Key Points

- The `To String` object is still being used to format the `Real64` into a time stamp format.
- Notice the sequence line between the first "generate" `Formula` and the second "evaluate" `Formula`. This ensures the second `Formula` will not execute until it gets the new string to evaluate.

Using the Test Sequencer, Step 8

Display only the records in which test 1 passed and test 2 failed.

Solution—Using the Test Sequencer, Step 8

Figure A-40 shows a solution to the final step in using the test sequencer.

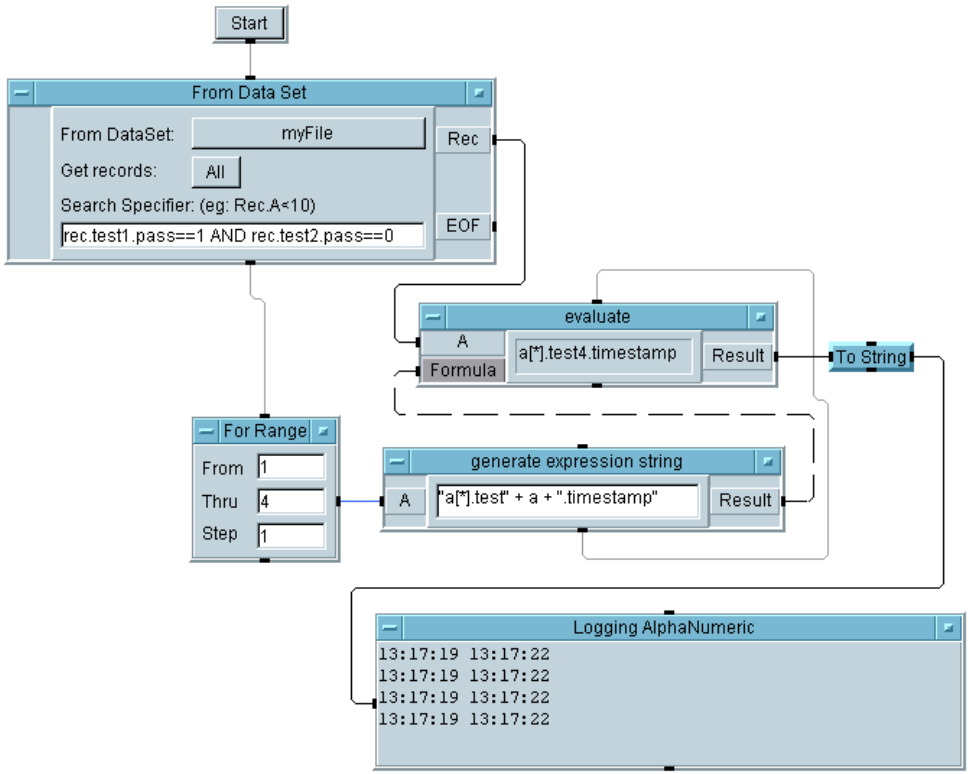


Figure A-40. Using the Test Sequencer, Step 8

Key Points

- **The EOF Pin on the From Data Set Object (Second Thread):** The `EOF` pin is added in case there are no records that fit the criteria. If this happens then the `EOF` pin will fire, instead of VEE halting the program with an error message.

- **The Conditional Expression in the From Data Set Object (Second Thread):** The expression is *(Rec.test1.pass==1) OR (Rec.test2.pass==*

0), with the same <*record*>.<*record*>.<*field*> format. `Rec` is the name of each record in the dataset as it is read and tested. `Test1` and `test2` specify which tests VEE should examine, and the field name pass is the default name for the pass-fail indicator (`1` or `0`) assigned by VEE. (You enable or disable different fields for all tests by selecting `Logging` tab in the `Sequencer Properties` box.)

Glossary

Glossary

This Glossary defines terms used in this manual. For a complete glossary of VEE terms, select `Help` ⇒ `Contents and Index`. Next, select `Reference`. Then, select `Glossary`. In the glossary, clicking a term displays a definition. When you have finished reading the definition, click anywhere to clear the text.

Button
A graphical object in VEE that simulates a momentary switch or selection button, and which appears to pop out from the screen. When you "press" a button in VEE, by clicking on it with the mouse, an action occurs. (May also refer to the left or right mouse button.)

Cascading Menu
A sub-menu on a pull-down or pop-up menu that provides additional selections.

Checkbox
A recessed square box on VEE menus and dialog boxes that allows you to select a setting. To select a setting, click the box and a check mark appears in the box to indicate a selection has been made. To cancel the setting, simply click the box again.

Click
To press and release a mouse button. Clicking usually selects a menu feature or object in the VEE window. See also **Double-Click** and **Drag**.

Clone
A menu item on the VEE object menus that duplicates objects and their interconnections, placing a copy of them in the `Paste` buffer. `Clone` copies all the attributes of the cloned objects including pins, parameters, and size.

Component

A single instrument function or measurement value in a VEE instrument panel or component driver. For example, a voltmeter driver contains components that record the range, trigger source, and latest reading.

Component Driver

An instrument control object that reads and writes values to components you specifically select. Use component drivers to control an instrument using a driver by setting the values of only a few components at a time. (Component drivers do not support coupling.)

Container

See **Data Container**.

Context

A level of the work area that can contain other levels of work areas (such as nested `UserObjects`), but is independent of them.

Cursor

A pointer (caret) in an entry field that shows where alphanumeric data will appear when you type information from the keyboard.

Cut Buffer

The buffer that holds objects that you cut or copy. You can then paste the object back into the work area with the `Paste` toolbar button (`Edit` ⇒ `Paste`).

Data Container

The data package that is transmitted over lines and is processed by objects. Each data container contains data and the data type, data shape, and mappings (if any).

Data Flow

The flow of data through and between VEE objects. Data flows from left to right through objects, but an object does not execute until it has data on all of its data input pins. Data is propagated from the data output pin of one object to the data input pin of the next object. Data flow is the chief factor that determines the execution of a VEE program.

Data Input Pin

A connection point on the left side of an object that permits data to flow into the object.

Data Output Pin

A connection point on the right side of an object that propagates data flow to the next object and passes the results of the first object's operation on to the next object.

Data Shape

Each data container has both a shape and type. The data shape can be either a scalar or an array of one or more dimensions. In VEE, a one-dimension array is called Array 1D, a two-dimension array is Array 2D, and so forth.

Data Type

Each data container has both a type and shape. VEE supports many data types including Text, Real64, Real32, and Int32.

Detail View

The view of a VEE program that shows all the objects and the lines that connect them.

Direct I/O Object

An instrument control object that allows VEE to directly control an instrument without using an instrument driver.

Double-Click

To press and release a mouse button twice in rapid succession. Double-clicking is usually a short-cut to selecting and performing an action. For example, double-clicking on a file name from `File` ⇒ `Open` will select the file and open it.

Drag

To press, *and continue to hold down*, a mouse button while moving the mouse. Dragging moves something (for example, an object or scroll bar).

Drop-Down List

A list of selections obtained by clicking on the arrow to the right of a selection field.

Entry Field

A field that is typically part of a dialog box or an editable object, which is used for data entry. An entry field is editable when its background is white.

Expression

An equation in an entry field that may contain input terminal names, global variable names, math functions, and user-defined functions. An expression is evaluated at run time. Expressions are allowed in `Formula`, `If/Then/Else`, `Get Values`, `Get Field`, `Set Field`, `Sequencer`, and `Dialog Box` objects, and in I/O transaction objects.

Font

VEE allows you to change the font size and style of type used to display text for various VEE objects, titles, and so forth.

Grayed Feature

A menu feature that is displayed in gray rather than black, indicating that the feature is not active or not available. Dialog box items such as buttons, checkboxes, or radio buttons may also be grayed.

Group Window

A group window in Microsoft Windows is a window that contains icons for a group of applications. Each icon starts an application in the group.

HP-UX

The derivative of the UNIX operating system that has been developed by Hewlett-Packard Company.

Hypertext

A system of linking topics so that you can jump to a related topic when you want more information. In online help systems, typically hypertext links are designated with underlined text. When you click such text, related information is presented.

Icon

1. A small, graphical representation of a VEE object, such as the representation of an instrument, a control, or a display.

2. A small, graphical representation of an application, file, or folder in the Microsoft Windows and HP-UX (with VUE) operating system.

Main Window

A window that contains the primary work area in which you develop a VEE program. The work area for this window resides in the work space for the VEE window.

Maximize Button

A button on a `UserObject`, `UserFunction`, or the `Main` window, that makes the `UserObject`, `UserFunction`, or `Main` window, occupy all of the available work space.

Menu Bar

The bar at the top of the VEE window that displays the titles of the pull-down menus from which you select commands and objects.

Minimize Button

A button on an object, or the VEE window, that iconifies the object, or the VEE window.

Object

i. A graphical representation of an element in a program, such as an instrument, control, display, or mathematical operator. An object is placed in the work area and connected to other objects to create a program.

ii. A data type used for ActiveX Automation and Controls.

Object Menu

The menu associated with an object that contains features that operate on the object (for example, moving, sizing, copying, and deleting the object). To obtain the object menu, click the object menu button at the

upper-left corner of the object, or click the right mouse button with the pointer over the object.

Object Menu Button
The button at the upper-left corner of an open view object, which displays the object menu when you click it.

Open View
The representation of a VEE object that shows more detail than the minimized view (icon). Most object open views have fields that allow you to modify the operation of the object.

Panel Driver
An instrument control object that forces all the function settings in the corresponding physical instrument to match the settings in the control panel displayed in the open view of the object.

Panel View
The view of a VEE program, or of a `UserObject` or `UserFunction`, that shows only those objects needed for the user to run the program and view the resulting data. You can use panel views to create an operator interface for your program.

Pin (or Pins)
An external connection point on an object to which you can attach a line.

Pointer
The graphical image that maps to the movement of the mouse. The pointer allows you to make selections and provides you feedback on a particular process underway. VEE has pointers of different shapes that correspond to process modes, such as an arrow, crosshairs, and hourglass.

Pop-Up Menu
A menu that is raised by clicking the right mouse button. For example, you can raise the `Edit` menu by clicking the right mouse button in an empty area within the work area. Or you can raise the object menu by clicking the right mouse button on an inactive area of an object.

Preferences

Preferences are attributes of the VEE environment that you can change using the `Default Preferences` button on the toolbar, or the menu `File` ⇒ `Default Preferences`. For example, you can change the default colors, fonts, and number format.

Program

In VEE, a graphical program that consists of a set of objects connected with lines. The program typically represents a solution to an engineering problem.

Program Explorer

A facility in the VEE window that permits exploration of a program, especially the parts of a program that might not be visible on the physical screen.

Propagation

The rules that objects and programs follow when they operate or run. See also **Data Flow**.

Properties

Object properties are attributes of VEE objects that you can change using `Properties` on the object menu, such as colors, fonts, and titles.

Pull-Down Menu

A menu that is pulled down from the menu bar when you position the pointer over a menu title and click the left mouse button.

Scroll Arrow

An arrow that, when clicked on, scrolls through a list of data files or other choices in a dialog box, or moves the work area.

Scroll Bar

A rectangular bar that, when dragged, scrolls through a list of data files or other choices in a dialog box, or moves the work area.

Select

To choose an object, an action to be performed, or a menu item. Usually you make a selection by clicking the mouse.

Selection Field
A field in an object or dialog box that allows you to select choices from a drop-down list.

Sequence Input Pin
The *top* pin of an object. When connected, execution of the object is held off until the pin receives a container (is “pinged”).

Sequence Output Pin
The *bottom* pin of an object. When connected, this output pin is activated when the object and all data propagation from that object finishes executing.

Status bar
A line at the bottom of the VEE window in which information about the current status of and information about VEE is displayed.

Status field
A field displaying information that cannot be edited. A status field looks like an entry field, but has a gray background.

Terminal
The internal representation of a pin that displays information about the pin and the data container held by the pin. Double-click a terminal to view the container information.

Title Bar
The rectangular bar at the top of the open view of an object or window, which shows the title of the object or window. You can turn off an object title bar using `Properties` in the object menu.

Toolbar
The rectangular bar at the top of the VEE window which provides buttons for quick access to frequently used commands. The buttons run commands from menus such as `File`, `Edit`, `View`, `Device`, and `Debug`.

Transaction

The specifications for input and output (I/O) used by certain objects in VEE. Examples include the `To File`, `From File`, `Sequencer`, and `Direct I/O` objects. Transactions appear as phrases listed in the open view of these objects.

UserObject

An object that can encapsulate a group of objects to perform a particular purpose within a program. A UserObject allows you to use top-down design techniques when building a program, and to build user-defined objects that can be saved in a library and reused.

Views

VEE presents a program in one of two views: panel view which provides a user interface for a VEE program, or detail view which provides a window for developing a VEE program.

Windows 95, Windows 98, Windows NT 4.0, Windows 2000

Operating systems, developed by Microsoft Corporation, in which VEE runs.

Work Area

An region within the `Main` window (also the `UserObject` and `UserFunction` windows) in which you place VEE objects and connect them together to create a VEE program.

Work Space

A region in the VEE window that contains the programming or editing windows such as `Main`, `UserObject`, and `UserFunction`. These windows contain work areas in which you place VEE objects and connect them together.

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z