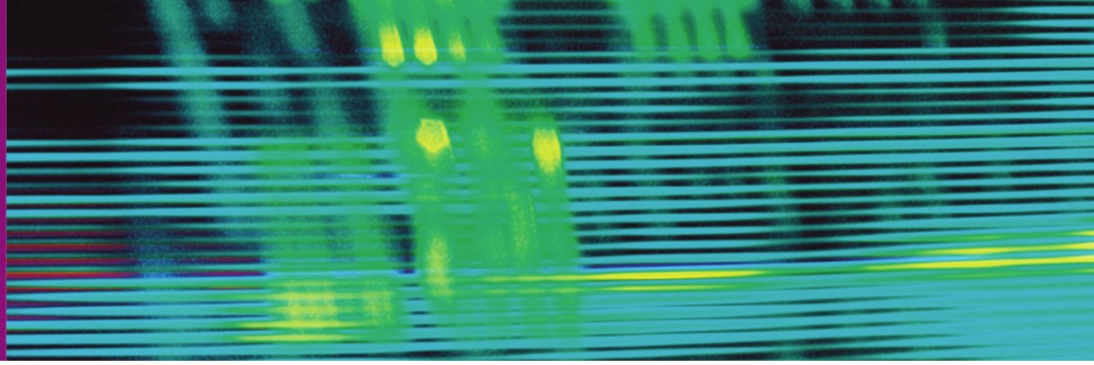


Robert B. Angus • Thomas E. Hulbert

VEE Pro

Practical Graphical Programming

Includes VEE 7.0 Features

VEE 7.0 Trial Version CD-ROM Included

Springer

VEE Pro

Robert B. Angus and Thomas E. Hulbert
Northeastern University

VEE Pro: Practical Graphical Programming

With 294 Figures

Springer

Robert B. Angus, BSEE, MSEE
Thomas E. Hulbert, BMgtE, MS eng mgt
360 Huntington Avenue, Boston, MA 02115-5000, USA

British Library Cataloguing in Publication Data
Angus, Robert B.
VEE Pro : practical graphical programming
1. VEE Pro (Computer file)
I. Title II. Hulbert, Thomas E.
006.6′86
ISBN 1852338709

Library of Congress Cataloging-in-Publication Data
Angus, Robert B. (Robert Brownell)
VEE Pro : practical graphical programming / Robert B. Angus and Thomas E. Hulbert.
p. cm.
Includes bibliographical references and index.
ISBN 1-85233-870-9 (alk. paper)
1. Visual programming languages (Computer science) 2. HP VEE (Computer program language) 3. Computer graphics. I. Hulbert, Thomas E. II. Title.
QA76.65.A54 2005
006—dc22 2004058914

ISBN 1-85233-870-9 Springer-Verlag London Berlin Heidelberg
Springer Science+Business Media
springeronline.com

Typesetting: Camera-ready by the authors
69/3830-543210 Printed on acid-free paper SPIN 10975121

We dedicate this book to our wives:

Sara M. Angus

and

Betty F. Hulbert

who have patiently endured the intensity, time, and effort required to write, test, and publish this book. Without their support, this book could not have been completed.

Preface

This book is written based upon VEE Pro Version 6.2. It contains eighteen lessons and six appendixes. The labs within the lessons introduce ActiveX support, MATLAB® functionality and display capabilities, and support for the new GPIB converters.

VEE Pro Version 6.2 is backwards compatible to at least VEE version 5.01. The labs of all eighteen lessons included in this book have been verified, opened, and run in versions 5.01, 6.01, and 6.2. Programs that work in versions 6 will work similarly in versions 5.

Previous editions of this book have been used successfully with three groups of students applying VEE to laboratory experiments, manufacturing systems, and process-control applications. VEE Pro is popular among technicians, technologists, and design engineers as well as with engineers and scientists. We have prepared this book with the former group in mind.

For those of you who are interested in learning VEE Pro in greater depth than is presented in this book or are designing complex analysis and monitoring systems, there are four excellent books:

- *VEE Pro User's Guide;* Chapter 12 (Platform Specifics and Web Monitoring)
- *VEE Pro User's Guide*; Additional Lab Exercises (Appendix A)
- *VEE Pro Advanced Programming Techniques*
- *Agilent IO Libraries Installation and Configuration Guide for Windows*

Recent improvements from Agilent can be accessed via the www.agilent.com Web site. The latest VEE Pro developments and on-line HELP are included as well.

This book introduces you to the fundamentals of VEE Pro, along with some more complicated examples in the later lessons. VEE Pro has been used to develop and operate programs as sophisticated as:

- At Woods Hole Oceanographic Institute, Agilent VEE programs help researchers to understand the earth's oceans and atmosphere. Research vessels operated by Woods Hole and the submersible Alvin use a common program for data collection, logging, display, and distribution. This program, written in VEE, has been travelling the world collecting data from the wreck of the Titanic, deep-sea hydrothermal vents near the Galapagos Island, has assisted in mapping the Indian Ocean, provided positional data for archeological projects in the Black Sea, and has helped untangle a whale off Provincetown, Mass., U.S.A.
- At the University of California, Berkeley, VEE Pro software has helped speed the design of its entry into the Defense Advanced Research Projects Agency Grand Challenge race of autonomous robotic vehicles. A team of 19 students, with no previous experience in physical test systems, entered the first two-wheel motorcycle into the race. The VEE Pro graphical interface enabled the team to develop test systems, and collect and analyze performance data. A two-wheel (versus four-wheel) vehicle design was chosen because it has a smaller structure, provides increased mobility,

and is less expensive. The challenge is to keep the vehicle upright while it is either stationary or in motion. A control-moment gyroscope was used to stabilize the vehicle; a crossbow solid-state gyro and inertial measurement unit was used to provide the data needed to ensure stabilization at high speeds. See Costlow, 2004.

- In another world, the Mars Exploration Rovers named "Spirit" and "Opportunity" contain a box known as the Small Deep Space Transponder. This box provides a direct link from the Rovers to our Earth. VEE provides the data logging, some automated testing, and command and telemetry interfaces for the testing of these communication links to and from the Earth. Also, the Mars Odyssey orbiter uses these same boxes to relay data to and from the Rovers.

In addition to VEE Pro, Microsoft Excel™ and Microsoft Word™ should be available on your computer. If a database is to be constructed, then Microsoft Access™ should also be installed.

The Agilent Visual Engineering Environment (VEE Pro) is a visual, mostly intuitive, programming language. It dramatically reduces monitoring-and-control software development time. The software programs are developed off-line with virtual equipment and devices. Thus, they do not tie up laboratory, manufacturing, and process-control equipment.

There is a special overview to acquaint supervisors and managers with the capabilities of VEE Pro, along with a related 3.5-inch PC disk of pre-developed labs. Users of this overview learn the capabilities and features without becoming programming experts. They are available at a nominal price by contacting the authors via bobangus@tiac.net.

This book provides the applicable tools for learning VEE Pro so programs can be written for

- data acquisition,
- test-data processing, and
- process control.

The detailed information is provided within the lessons.

Programs are prepared by connecting icons together within VEE Pro and possibly program segments prepared in languages such as C++ and Visual Basic. The resulting program resembles a block diagram that may be run like a program prepared in a textual language such as a "C++" or "Visual Basic". Figure 1 explains how VEE Pro monitors a data acquisition system. Any report generated can be distributed over the Internet.

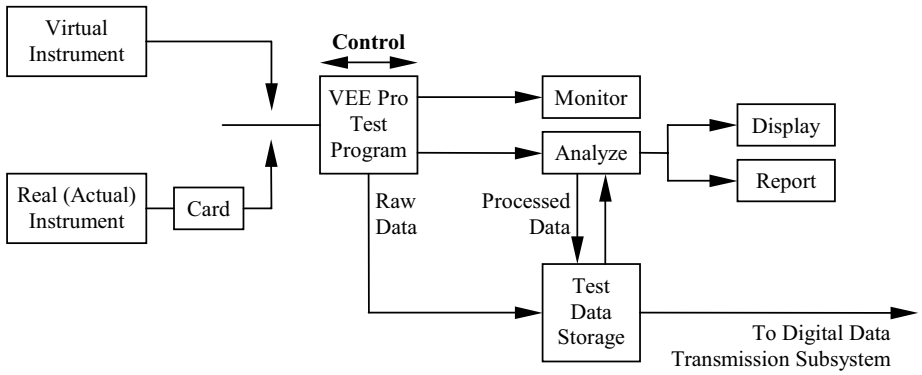


Figure 1. VEE Pro monitoring a data-acquisition system

Virtual instruments exist within the VEE Pro program. Real (actual) instruments can be attached via special plug-in cards. Often, a separate card is required for each instrument. Instruments now coming on-line contain USB links that can go directly to a computer, and some have LAN connections do not need cards.
The VEE Pro test program controls:

- which virtual or actual instrument are connected via VEE Pro,
- what parameters are to be monitored,
- how data are to be processed,
- where data are stored – as either raw data or processed data, and
- how that data are processed for spreadsheets, reports, and databases.

Microsoft Excel™ is used for spreadsheets, Word™ is used for written reports, and Access™ is used for storing large amounts of data in a database. Data can be transmitted to other locations or systems via communications media such as: USB, Fire Wire, LAN, or many of the RS series of protocols. VEE Pro is very powerful and flexible.

VEE Pro is both a language and a graphical development environment. After you have completed each pre-lab and its accompanying labs, you should be able to:

1. visualize the relationship between objects as operational instructions in VEE Pro,
2. identify the VEE Pro Main Window and use its Work Area, Program Explorer, Title Bar Tool Bar, Status Bar and control icons to size, position, monitor and set window operations,
3. use a VEE Pro menu to call a device to the screen,
4. call the Object Menu of VEE Pro objects in two ways, and use the Object Menu to control, view, and set object properties,
5. access and use VEE Pro's Help Menu, and
6. integrate VEE Pro processed data into spreadsheets, reports, and data bases.

Each VEE Pro Object added to the program symbolizes a set of instructions from the VEE Pro library. VEE Pro handles the programming chores and leaves the user free to design and test processes. As the program is being designed, VEE Pro is providing the software. This interactive technique is known as "graphical programming".

The Help menu of VEE Pro can be accessed either from the Menu Bar of any Window or from the Object Menu of any VEE Pro object. The Help Menu provides both a detailed glossary of terms and "How-Do-I" guides to performing an enormous number of tasks within VEE Pro.

All eighteen lessons have laboratories that introduce new objects, concepts, or techniques. In order to enable you to learn the capabilities of the total VEE Pro software package, these items will be presented and illustrated prior to implementing each laboratory. Also, the symbol "◊", when it precedes a laboratory-exercise step, informs you to change parameters so you may expand your understanding of the power and flexibility of VEE Pro.

At the end of the eighteen lessons are six appendixes. These appendixes are designed to guide the casual practitioner to the location of the necessary design material within this book. The following describe each appendix, its contents, and how it can be used:

- Appendix A consists of descriptions of the menu bar and tool bars (buttons or icons). This appendix contains a list of all pull-down menus and buttons for the entire VEE Pro, with shortcut-keys (i.e. Ctrl + …) shown in bold and underlined. The names of the icons on the Tool Bar are also given. This appendix includes an explanation of the mouse operation, how data flows into and out of an object, and the Object order of operation.
- Appendix B is a list of the partial program sequences developed in the lessons. A cross-index of these sequences is provided, arranged alphabetically to allow rapid reference by the program developer.
- Appendix C is a list of the virtual devices and instruments available within VEE Pro. For each of these items, a summary of each applicable Help menu is provided.
- Appendix D contains the definition of the specialized technical terms used in the book.
- Appendix E introduces the objects, icons, and features via a series of seventeen pre-labs.
- Appendix F is a brief summary of the features and functions of VEE 7.0.

Each pre-lab is designed to explain, in great detail, the material which will be first applied in the referenced lesson. If this material were included in the lesson, it would

- detract from the lesson flow and
- be difficult to reference in its entirety by a person devising a new VEE Pro program.

Thus, when going to an example lesson, the person might be confused between the generic material presented in Appendix E and the specific application being developed in the lesson.

Instruction in VEE Pro can be accomplished in at least three different formats.

The Pre-Lab Concept

The first method is to present and discuss the pre-lab in fifteen to thirty minutes, depending upon the depth and number of new concepts introduced. Students then proceed through each lab presented in the lesson by working individually at a station. Each lesson is designed to take the average student about two hours.

The Self-Study Concept

The second approach is to use this book in a self-study, self-paced mode. Students will study the applicable pre-lab material (Appendix E) and, on their own, schedule time on a computer to proceed through each lab.

The Combined Concept

The last method combines the previous two above. The pre-lab is presented to an entire class, and the labs are completed by individual students at their own pace. The only constraint is that the previous lesson must be completed before the next pre-lab is presented.

The eighteen lessons gradually present how to learn VEE Pro by focusing upon the graphical general and specialized icons available within VEE Pro. These icons can be combined and attached to virtual or real instruments via an Instrument Manager. Data analysis can be performed by a combination of built-in mathematical and statistical calculations located in the Function & Object Browser and the specialized MATLAB® script.

Demonstration of the power and flexibility of VEE Pro is presented via a virtual Vehicle Radiator. It starts with the simple monitoring of a simulated temperature and ends with monitoring and logging the Vehicle Radiator temperature variations using the VEE Pro Sequencer. Special labs include:

- Simulating a thermometer and monitoring that temperature
- Calculating, logging, and displaying temperature and pressure statistics
- Building databases and spreadsheets
- Writing reports that summarize and display that data
- Plotting temperature variations via MATLAB®
- Monitoring and logging test limits containing audio and visual warnings

When this book is used by instructors at an educational or training institution, various combinations of the above can be devised to comply with their class and laboratory scheduling requirements.

The appendixes are devised to support the reader – both during the learning of VEE Pro and when this book is used as a reference at a later time. The appendixes are:

- Appendix A – Menus, Buttons, the Mouse, and Data Flow – detailed descriptions of the menu bar, tool bar, pull-down menus and buttons, mouse operation, how data flows into and out of an object, and the Object's order of operation
- Appendix B – Partial Programming Sequences – a cross index of partial program headings
- Appendix C – Virtual Devices and Instruments – a summary of each available within VEE Pro
- Appendix D – Definition of Technical Terms – devised especially for those whose technical vocabulary includes very little related to programming
- Appendix E – Introducing Objects, Icons, and Features – seventeen pre-labs that explains them in more detail than is given within the individual labs
- Appendix F – Agilent VEE 7.0 Features – the forthcoming capabilities of the new version are described with information that will lead the reader to additional information

The Bibliography contains a list of reference manuals, technical papers, articles, and textbooks related to VEE Pro.

Exchange of information among the user community is supported via an interactive Agilent Web site (vrf). The address is www.agilent.com/find/vrf and it is available 24 hours a day, seven days a week. All users are encouraged to register and use this Web site to gain additional knowledge, insight, and information. This will help you to keep informed of any changes that the Agilent designers might propagate via this Web site.

The authors believe that the most powerful feature of this book with its appendixes is its value as a reference during the design process, the running of real manufacturing processes, and the design-and-analysis of laboratory experiments.

Robert Angus
Thomas Hulbert

English may be a difficult or second language for you.
The words in this book have been carefully chosen to ease the reading.

Acknowledgements

The material in this book was prepared from the following sources:

1. *Visual Programming with HP VEE* by Robert Helsel
2. *Introduction to VEE Pro E212E+24D Student Manual* by the Hewlett-Packard Test & Measurement Education Services
3. *VEE Pro Advanced Programming Techniques* by the Agilent Technologies staff
4. *VEE Pro User's Guide*, by the Agilent Technologies staff
5. *Installation and Configuration Guide for Windows*, by the Agilent Technologies staff

We are indebted to the authors of and contributors to these books.

All art was provided via "screen grabs" from the Agilent Technologies VEE program and literature. (Agilent VEE is a registered trademark of Agilent Technologies, Inc.)

Special thanks and appreciation are extended to the following individuals who contributed to the development, review, and publication of this book:

- Kace Dreitlein of Agilent Technologies who was the first person to train us in VEE Pro; Kace also served as our technical advisor during the entire VEE 5.01 and VEE Pro 6 development effort.
- Jack Parchesky of Agilent Technologies who provided guidance in setting up the first training program and provided advice during the development of the early versions of the manuals.
- Susan Wolber and Adam Kohler of Agilent Technologies and Michael L. Brown of Carrier Corporation who volunteered their time to provide comprehensive technical and content review of our entire book
- Scott Bayes, Ken Colasuonno, and Carrie Browen of Agilent Technologies who have guided and supported us with their personal knowledge and technical resources
- Tom Gaudette and Courtney Esposito of MathWorks for their MATLAB® input
- Anthony Doyle of Springer-Verlag London Ltd for his faith in us as we prepared the manuscript for publication
- Oliver Jackson of Springer-Verlag London Ltd for his outstanding assistance in the preparation of camera-ready copy
- Bill Miller, Principal of William D. Miller Associates, for his suggested lab examples
- Amy Andres, Chair of the Engineering Council of Texas Instruments
- Doug Strott, member of the Engineering Council of Texas Instruments

- Del Hall, John Lussier, and Peter Sampson of Texas Instruments
- John Indelicato of General Physics Corporation
- John Robidoux of Mitre Corporation
- Susan Moulton, Jennifer Alfieri, Walter Buchanan, John Cipolla, and Hameed Metghalchi of Northeastern University
- Jeffrey Doughty, Richard Weston, James Hinds, and LeBaron Briggs of the MIME Department of Northeastern University
- Bob Tortolano of Cheetah Technologies
- The nearly 100 students who provided us with constructive suggestions during the testing phase

Without their encouragement and critiques, this book could not have been written and tested.

To the Instructor

The Agilent Visual Engineering Environment (VEE Pro) is a visual, mostly intuitive, programming language. It dramatically reduces monitoring-and-control software development time. The software programs are developed off-line with virtual equipment and devices. Thus, they do not tie up laboratory, manufacturing, and process-control equipment.

VEE Pro is very powerful and flexible. It is designed to operate in a Windows environment. Microsoft Excel™ and Word™ should be available on your computer. If a database is to be constructed, then Access™ should also be available.

Instruction in VEE Pro can be accomplished in at least three different formats.

- The first method is to present and discuss the pre-lab in fifteen to thirty minutes, depending upon the depth and number of new concepts introduced. Students then proceed through each lab presented in the lesson by working individually at a station. Each lesson is designed to take the average student about two hours.
- The second approach is to use this book in a self-study, self-paced mode. Students will study the applicable pre-lab material (Appendix E) and, on their own, schedule time on a computer to proceed through each lab.
- The last method combines the previous two above. The pre-lab is presented to an entire class. Each lab is completed with no individual time constraints. The only requirement is that the previous lesson must be completed before the next pre-lab is presented.

When this book is used in a more formal way at your educational or training institution, a combination of the above can be devised to comply with your class and laboratory schedule.

Much success has been achieved by assigning projects to groups of up to three students once they reach Lesson 5. Students usually choose real problems either from their academic or work environment or from local industry. In the last class, these students present their problems along with their solutions.

The most powerful feature of this book with its appendixes is its value as a reference.

- Appendix A describes the VEE Pro program menus, buttons, the mouse, and data flow.
- Appendix B is a cross-index of partial program sequences.
- Appendix C is a list of the virtual devices and instruments available within VEE Pro.
- Appendix D contains the definition of the specialized technical terms used in the book.
- Appendix E introduces the objects, icons, and features via a series of seventeen pre-labs.
- Appendix F is a list of the features of Agilent VEE 7.0 as envisioned by the developers.

Recent improvements and upgrades from Agilent can be accessed via the www.agilent.com Web site. The latest VEE Pro developments and on-line HELP are included as well. For qualified academic institutions, a multiple-seat license with full documentation is available at a substantially discounted price. Contact Agilent via its Web site (above).

To the Student

This book has been prepared to be a study guide and reference for technical personnel responsible for designing complex test, measurement, and data-acquisition systems, and collecting and analyzing laboratory-test data. You will learn to program via a visual engineering environment (VEE). This present version is known as VEE Pro.

Portions of each program are contained in code represented by Objects and icons. Each VEE Pro Object added to the program represents a set of instructions taken from the VEE Pro internal library. Each time a new Object is added and interconnected with the others, the computer's program prepares and links additional instructions in VEE Pro. The capabilities of your program are being devised.

VEE Pro simplifies the programming chores and leaves the user free to design and test processes. As the program is being designed, VEE Pro provides the software. This interactive technique is known as "graphical programming".

Most of the eighteen lessons have laboratories that introduce new objects, concepts, or techniques. In order to enable you to learn the capabilities of the total VEE Pro software package, these items will be presented and illustrated prior to implementing each laboratory. At the end of the eighteen lessons are six appendixes. These appendixes are designed to guide the casual or experienced practitioner to the location of the necessary design material within this book. The following describes each appendix:

- Appendix A – the VEE Pro program, when opened, consists of a menu bar and a tool bar (buttons or icons). This appendix is a list of all pull-down menus and buttons for the entire VEE Pro, with shortcut-keys (i.e. Ctrl + …) shown in bold and underlined. The names of the icons on the Tool Bar are also given. The mouse operation is explained in detail. It is followed by how data flows into and out of an object and the Object order of operation.
- Appendix B is a list of the partial program sequences developed in the lessons. A cross-index of these sequences is provided, arranged alphabetically to allow rapid reference by the program developer.
- Appendix C is a list of the virtual devices and instruments available within VEE Pro. For each of these items, a summary of each applicable Help menu is provided.
- Appendix D contains the definition of the specialized technical terms used in the book.
- Appendix E introduces the objects, icons, and features via a series of seventeen pre-labs. Each pre-lab is designed to explain, in great detail, the material that will be first applied in the referenced lesson.
- Appendix F is a list of the features of Agilent VEE 7.0 as envisioned by the developers.

We wish you well in the pursuit of your educational and vocational goals and the application of VEE Pro to the solution of your technical and design problems.

Table of Contents

Lesson 1
The VEE Pro Development Environment

This lesson explores the details of the VEE Pro development environment. This lesson consists of a pre-lab summary and five labs.

Lesson 1 Pre-lab Summary

The following are described in the Lesson 1 Pre-lab. See Appendix E, page E-1:

- The Mouse
- The VEE Pro Opening Development Screen with its Title, Menu, Tool, and Status Bars,
- The VEE Pro Screen with the Program Explorer and Work Area (Main),
- The Menu Bar => Display with its Note Pad, Waveform (Time), and AlphaNumeric objects,
- The Menu Bar => Device => Virtual Source with its Function Generator and Noise Generator,
- The Menu Bar => Device => UserObject, Formula, and Function & Object Browser objects, and
- The overall reminders regarding Program Explorer, Cloning Objects, Open View vs. Iconic View, Program Development, and Run-Time Errors.

Also, Appendix B includes a cross-reference to each of these terms and to all icons, objects, and subprograms contained in the labs of this and later lessons.

Overview

This lesson will examine the fundamentals related to the VEE Pro development environment. Following the pre-lab for this Lesson, you will complete the five laboratory exercises that follow.

Lab 1.1 – Generating and Displaying a Waveform
This lab will show you how to: clear your Work Area; select, move, and size a Note Pad Object; examine the use of the Note Pad via the Help menu; select, move, and change values in a Virtual Source Object; change between Open View and Iconic View; select and move a display; connect two objects; save a program; and test and debug a program.

Lab 1.2 – Generating and Displaying a Noisy Waveform
This lab will show you how to: devise a UserObject that will contain VEE Pro interconnected Objects; select and edit a device and convert it to an Iconic View; select a Function and Object Browser Box and

create a formula within it; connect three objects within a UserObject area; display a running program with the virtual oscilloscope; change the parameters within a program; and save a program via the Menu Bar.

Lab 1.3 – Simulating a Thermometer
This lab will show you how to: create a virtual thermometer; monitor the thermometer UserObject; modify thermometer temperature-related parameters; and observe the effect of parameter changes to the thermometer program.

Lab 1.4 – Monitoring the Temperature of a Virtual Vehicle Radiator
This lab will show you how to: simulate a vehicle radiator whose temperature varies linearly; monitor the vehicle radiator UserObject; modify vehicle radiator temperature-related parameters; describe the vehicle radiator program via the Note Pad; and observe the effect of parameter changes to the vehicle radiator program.

Lab 1.5 – Using Real-World Labels in a Formula Object
This lab will show you how to: select Object program component parts; change pin names to real-world label names; and stop and correct a program.

Lab 1.1 – Generating and Displaying a Waveform

This lab will show you, via a simple program, how to start a VEE Pro program, then generate and display a waveform. You will also learn how to select and use a Note Pad (step 2 below), a virtual Function Generator (Figure 1.1), and a virtual waveform display (Figure 1.3). You will then learn how to connect these two objects to create an operable program and how to run and save that program.

Opening the VEE Pro Program and clearing your Work Area

1. Open the VEE Pro program: Click Start; go to Programs; select Agilent VEE Pro 6.
   (Always start with a clear Work Area. Select Menu Bar => File => New to clear your Work Area.)
   Note 1: Click indicates that you press, and immediately release, the mouse left button.
   Note 2: Drag indicates that you depress the mouse left button and hold it down over the object(s) you desire to move to a new location. You can also drag the Work Area, moving all its objects simultaneously.
   Note 3: Double-click indicates that you click the mouse left button twice rapidly when the mouse pointer is on the object or area of interest.

Selecting, moving, and sizing a Note Pad Object

2. Select Menu Bar => Display => Note Pad. A blank rectangle (wire frame outline) will appear on your Work Area.
   Note 1: The Note Pad allows you to document your program.
   Note 2: Objects are individual units (blocks) in a software development that perform specific tasks.
3. Move your Note Pad Object as follows:
   Move the mouse (without clicking it) and place this rectangle at the top-center of your Work Area; click to establish its location.
4. Click in the Note Pad white area to obtain a cursor.

Note: The correct technical name for the Note Pad white area is its editing area.

5. Type the following program description into your Note Pad editing area:
   Display Waveform generates a cosine waveform and sends it to a real-time display.
6. Size your Note Pad:
   Move the mouse to the lower right-hand corner of the Note Pad Object until a right-angle (sizing) icon appears. Depress the mouse left button and change the size of the Note Pad so that all typed words are displayed.
   Note: An alternate method for sizing any object is to right-click on that object and select Size from the menu that is displayed.

Examining the use of the Note Pad via the Help menu

7. *Right*-click any place on the Note Pad and a drop-down menu will appear; click on Help; examine the description of this Object and its use.

Selecting, moving, and changing values in a Virtual Source Object

8. Select Menu Bar => Device => Virtual Source => Function Generator. A wire-frame outline will appear. Move the outline until it is at the left of your Work Area and below the Note Pad. Click the mouse; the Function Generator Open View will appear in that position. See Figure 1.1.

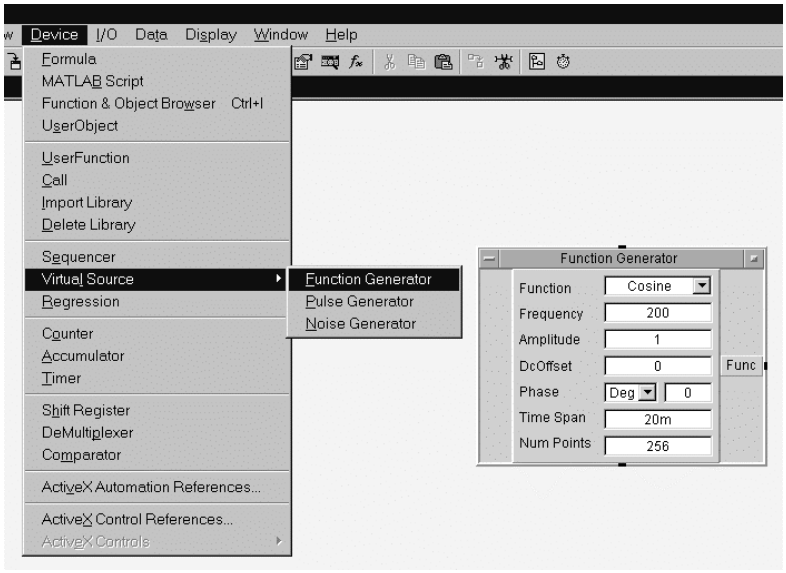


Figure 1.1. The Function Generator Object open view

Changing between Open View and Iconic View

9. Place your mouse on the title bar where the words “Function Generator” are displayed.
   Note: This view of the Function Generator is the Open View.
10. Click the dot on the right end of the Title Bar – its icon “minimize” button. The Function Generator Object converts to its Iconic View. See Figure 1.2 on the next page.

11. Double-click anywhere on the Function Generator icon. The Function Generator object will return to its Open View.
12. Drag the mouse pointer across the Frequency numbers in the white area (data field); change the displayed numbers from 200 to 100 **OR** double-click on the Frequency white area (data field) and enter: 100.
    Note: To save time and space, this instruction will be shortened to read as follows: "Edit Frequency to 100".

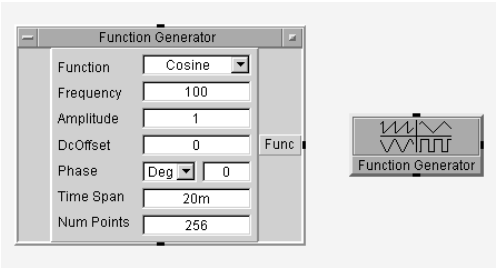


Figure 1.2. The Function Generator open view and iconic view

Selecting and moving a display

13. Select Menu Bar => Display => Waveform (Time). Move the wire-frame outline that appears to the right of the Function Generator and under the Note Pad. A virtual oscilloscope will appear labeled "Waveform (Time)".
    Note: You may select Move from the Object Menu. Touch the place where you want the Object located. Its upper-left corner will re-locate to that position.
14. Examine the Function Generator data-output pin (labeled Func); its output pin is a small black square at the right edge; it is next to the Func pin in the Open View.

Connecting two objects

15. Move the mouse pointer to the Waveform (Time) object near the data-input pin (located just above the Trace1 words). Click, then move, the mouse button. A line will appear. Click near the Function Generator "Func" pin; that line will connect the Function Generator output to the Waveform (Time) input.
16. Locate the right-pointing solid arrow on the Tool Bar – the Run arrow. (It is the fifth icon from the left.) Click on this arrow. (The program should run.) A cosine waveform will appear on the virtual oscilloscope. See Figure 1.3 below.

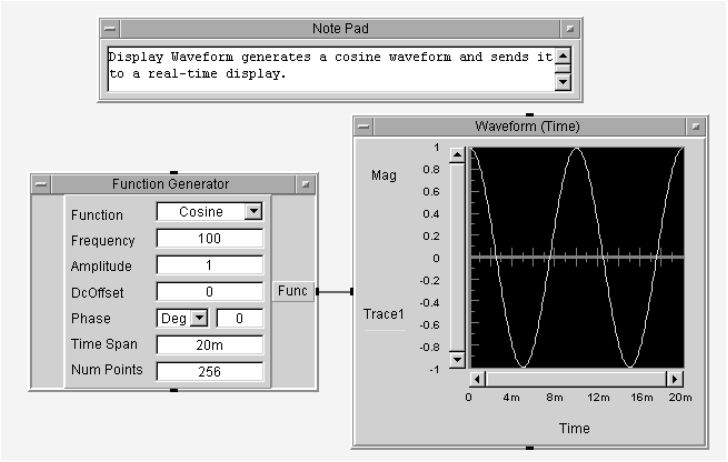


Figure 1.3. The Display Wave program

Note: If the waveform amplitude is greater than the Waveform (Time) display and "Auto Scale" is on, the display will automatically adapt to the height and width of the waveform. Auto Scale can be disabled and adjusted via the Object Properties box "Scales" tab.

Saving a program

Note: Unless otherwise noted, always save your programs to your personal floppy disk. Bring that disk to all future labs. Make a second (backup) disk for your own protection.

17. Insert your floppy disk into drive A.
18. Go to Save As…; select 3½ Floppy (A:).
19. Go to the File name field; type LAB1-1; click Save.

or

Select Menu Bar => File => Save As… .; go to the File name field; type LAB1-1; click Save.
Note: VEE Pro will automatically add the .vee extension.

Testing and debugging a program

To expand your knowledge, explore the following variations to LAB 1-1:

20. Recall the program LAB1-1.

◊ 21. Click on the Mag label in the Waveform (Time) Object; vary the scope screen amplitude**;** observe its effect on the waveform.

Note: The symbol ◊ informs you that the indicated step enables you to expand your understanding of the flexibility and functionality of VEE Pro.

◊ 22. Vary the Function Generator frequency, amplitude, and time span; run the program for each variation; record the effects that these changes have on the scope waveform, both Mag and Time.

23. Close LAB1-1 *without* saving it again.

Lab 1.2 – Generating and Displaying a Noisy Waveform

This lab will show you how to add pins, create and include UserObjects (steps 2 through 4 below), place them within other UserObjects (nest them), work with their pop-up dialog box, and devise a simple formula.

The UserObject is accessed from the Device menu. It creates a special window always within the Main Window. It can contain a group of interconnected Objects. It can be saved in a library as a single entity and reused. It can contain programs of your own design and other UserObjects.

1. Clear your Work Area.

Devising a UserObject that will contain VEE Pro interconnected Objects

2. Select Menu Bar => Device => UserObject. (This is one way to devise a program.)
3. Move the wire-frame outline (UserObject) to the left side of your Work Area (inside the Main window); click the mouse; double-click on the UserObject object; its Open View will fill the screen.

   Note 1: Minimize the UserObject. The UserObject open window will close to a miniature icon will appear in the lower-left of your screen just above the Status Bar. Whenever you minimize any window, its miniature title bar will also appear beside the previous one(s). See Figure 1.4.

   Note 2: If you lose the VEE Pro screen, then you may use "find" to re-acquire it or you may use the Program Explorer to re-acquire it.

   Note 3: Should you accidentally move a program off screen and cannot easily find it, then place the pointer over any portion of the Main white space and depress the keyboard key: Home. The entire program will be moved to the upper left of your Work Area.
4. Double-click on the UserObject Title Bar; the UserObject Properties dialog box will appear.

   or

   Right-click any place on the User Object and select Properties from the drop-down menu.

Figure 1.4. Three views of UserObject

Selecting Properties with your mouse right button.

5. Type NoisyCos in the Title box; click OK. See Figure 1.5.

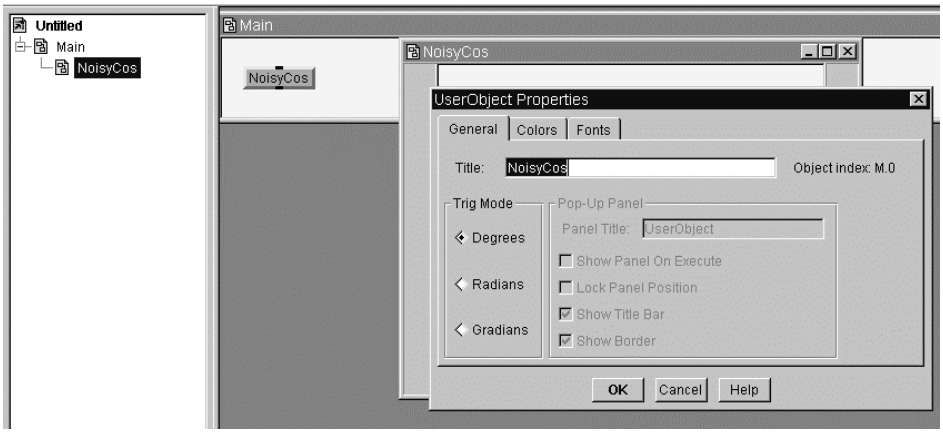


Figure 1.5. Changing UserObject title

6. Click on the NoisyCos Title Bar with the mouse *right* button.
7. Select Add terminal => Data Output. (The output pin is labeled X.)
8. Go to the upper-right corner of the NoisyCos object. Click the mouse left button on the middle NoisyCos button in the upper-right corner. (This will cause the object to reduce in size so the Main window is observable. Clicking it again will cause it to once more fill the screen; this middle button is a toggle.) Maximize the NoisyCos so it fills the screen.

Selecting and editing an object and converting it to an Iconic View

9. Select Menu Bar => Device => Virtual Source => Function Generator; move it into the left center of the NoisyCos object.
   Note: From now on, you should realize that "drag" includes "drag and release"; "move" means moving the mouse without pressing the mouse button.
10. Edit Frequency to 100.
11. Click on the Function Generator upper-right button; it will reduce the Function Generator object to Icon. Move the Function Generator object to the left of the Work Area slightly above the center.
12. Select Menu Bar => Device => Virtual Source => Noise Generator; place it below the Function Generator within the NoisyCos UserObject.
13. Click on the Noise Generator upper-right button; it will reduce the Noise Generator object to an Iconic View. Drag and place it under the Function Generator object.

Selecting a Function and Object Browser Box and creating a formula from within it

14. Select Menu Bar => Device => Function & Object Browser.
    Note: The box that appears is a "pop-up dialog box". See Figure 1.6. (The dialog box can be sized by clicking and moving its outside borders.)

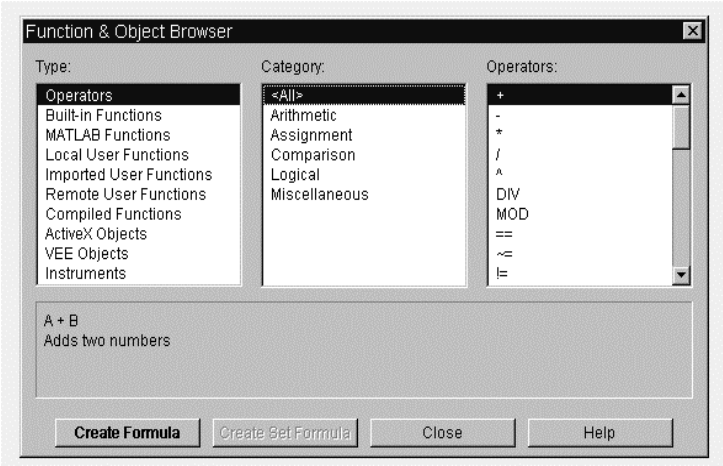


Figure 1.6. Function & Object Browser pop-up dialog box

15. Select Type: Operators; Category: <All>; Member: + .
16. Click on Create Formula in the lower-left corner of the pop-up dialog box; a new formula "A+B" will appear.
17. Place "A+B" to the right of the Function Generator and the Noise Generator.
    Note: There is another way to create the A+B (Formula) Object; it will be presented later. This object provides access to the VEE Pro math library.

Connecting three objects within a UserObject area

18. Connect the Function Generator data-output pin to the A+B data-input pin "A".
19. Connect the Noise Generator data-output pin to the A+B data-input pin "B".
20. Connect the A+B data-output pin (Result) to the X pin within the Noisy Cos Work Area (which is the output of the UserObject).
21. Go to the upper-right corner of the NoisyCos Title Bar; click on the middle button. (This will decrease the size of the Noisy Cos UserObject.) See Figure 1.7.

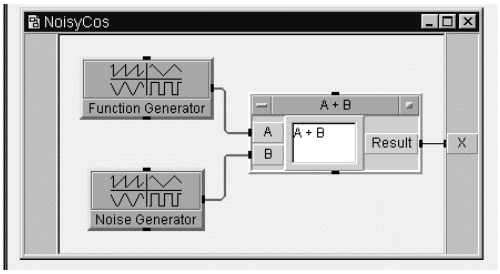


Figure 1.7. The completed NoisyCos UserObject

22. Go to the NoisyCos Title Bar; click on the right-hand "_" button. (This will reduce the NoisyCos UserObject to an Icon. The icon of the edit window will appear across the bottom of the screen under the Main window as a rectangle.)
    Note 1: Raise the bottom of the Main window to see the NoisyCos rectangle.
    Note 2: Go to pages A-8 and A-9 to learn about Data Flow and Object Order of Operation.

Displaying a running program with a virtual oscilloscope

23. Select Menu Bar => Display => Waveform (Time); place it inside Main to the right of NoisyCos.
24. Connect the NoisyCos data-output pin (X) to the Waveform (Time) data-input pin. (That pin is just above "Trace1".)
25. Go to the Tool Bar; click the (right-facing arrow) Run button. A noisy waveform will appear on the oscilloscope. See Figure 1.8.

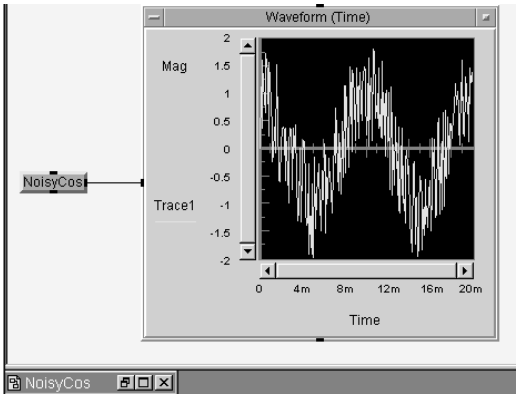


Figure 1.8. The NoisyCos(ine) program after running

Saving a program via the Menu Bar

26. Select Menu Bar => File => Save As … ;
27. Save this program to your personal disk (drive A:); type LAB1-2; click OK.

To expand your knowledge, explore the following variations to LAB1-2:

28. Open LAB1-2 if necessary; open the NoisyCos object.

Changing the parameters within a program

29. Open the Noise Generator object.
◊ 30. Change its amplitude from 1 to 0.2 and run the program; what effect does this have on the scope wave?
31. Return the amplitude to 1; run the program.
◊ 32. Open the Noise Generator object; change the time span to 40m; run the program. Describe the effect and explain why it would not run. Stop the program with the black square that is the second icon to the right of the "run" icon.
33. Close LAB1-2 without saving it.

Lab 1.3 – Simulating a Thermometer

VEE Pro provides an excellent means for simulating the behavior of devices, such as a thermometer. Modern thermometers vary in temperature quite gradually. This simulation assumes a gradual linear variation; a triangular-wave is used. This lab will show you how to apply objects that lead to a reasonable simulation of an actual thermometer with both analog and digital readouts.

1. Clear your Work Area. (See Appendix B in the rear of this book if necessary.)

Creating a virtual thermometer

2. Select Menu Bar => Device => UserObject; move it to the left-center of your (Main) screen.
3. Double-click on UserObject to obtain its Open View.
4. Double-click on its Title Bar; change the title to Thermometer; click OK.
5. Click anywhere on the Thermometer UserObject Title Bar with your mouse right button to obtain the UserObject menu.
6. Select Add terminal => Data Output; an "X" will appear on the Open View right side.
7. Select Menu Bar => Device => Virtual Source => Function Generator; move it to the left center inside the Thermometer UserObject.
8. Open the Function Generator object if necessary; change its title to Thermometer.
9. Select Tri in the Function box by clicking on the Function drop-down arrow.
10. Change Frequency to 50 and amplitude to 0.5; see Figure 1.9.

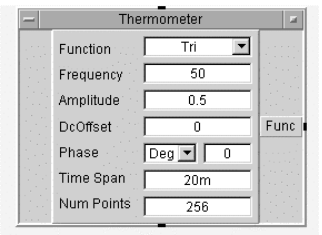


Figure 1.9. Partial program for Lab 1.3

11. Connect the Function Generator data-output pin to the Thermometer UserObject data-output pin "X".
12. Convert Thermometer to an Iconic View.
    Note 1: A second rectangle labeled: "Thermome…" will also appear at the bottom-left of your Main screen. You can access any UserObject by double-clicking on that rectangle – its icon.
    Note 2: If you close the icon via the "X" (close) button, the object will disappear. You can only access it again via the Program Explorer.

Monitoring the thermometer UserObject

13. Select Menu Bar => Display => Waveform (Time); place it in the Main screen right-center.
14. Connect the Thermometer UserObject data-output pin "X" to the Waveform (Time) input pin.
15. Run this partial program. A triangular wave should appear.

Modifying thermometer temperature-related parameters

16. Open Thermometer and its Temperature Generator.
◊ 17. Change the Temperature Generator Frequency to 100 and Amplitude to 2.
18. Close the Thermometer UserObject only.
19. Run this modified program. What effect does this change have on the scope presentation? Look closely; there is a subtle change.
20. Open Thermometer; change Temperature Frequency to 50 and Amplitude to 0.5.
Note: If you missed it, the Trace1 scale changed so the waveform would fit on the screen. Also, the number of displayed cycles changed.

Observing the effect of parameter changes to the thermometer program

21. Select Menu Bar => Display => AlphaNumeric; place it to the right of Thermometer; connect input pin to output pin (X) of Thermometer. See Figure 1.10.

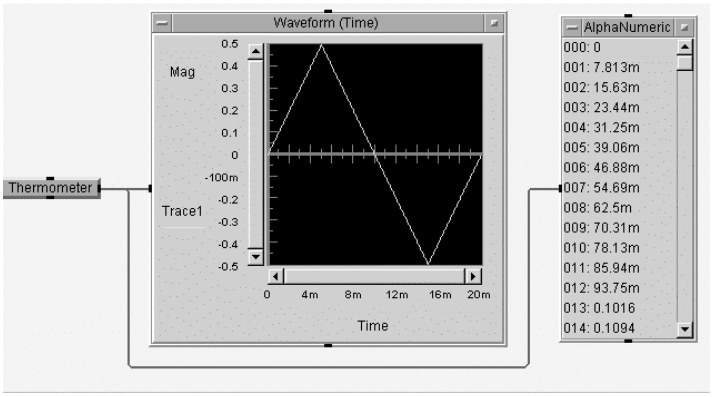


Figure 1.10. Final program for Lab 1.3

◊ 22. Open Thermometer; change the Temperature Generator Time Span to 40m.
23. Run this program; observe the effect of the Temperature changes on the scope waveform.
24. Save this program to your personal disk as LAB1-3.

Lab 1.4 – Monitoring the Temperature of a Virtual Vehicle Radiator

This lab will show you how to devise a virtual Vehicle Radiator using the Function Generator in its triangular-waveform mode. You will also observe the effect of Function Generator changes on the oscilloscope display. This is the first of several practical labs that will be expanded during later lessons.

1. Clear your Work Area. (See Appendix B.)

Simulating a Vehicle Radiator whose temperature varies linearly

2. Select Menu Bar => Device => UserObject; move it to the left-center of the Main screen.

3. Double-click on UserObject to obtain its Open View.
   Note: A UserObject is a user-defined object that can contain a number of interconnected objects. For use elsewhere in this or another program, clone the UserObject and move it to where you want to use it again. If you wish to modify the contents of the cloned UserObject, then modify it in its Open View and modify its title. (Later you will learn about UserFunctions which are a much better way to re-use code.)
4. Double-click on its Title Bar; change the title to Vehicle Radiator.
5. Click OK.
6. Click anywhere on the Vehicle Radiator UserObject Title Bar with your mouse right button to obtain the UserObject menu.
7. Select Add terminal => Data Output; an "X" will appear in the Open View on its right side. (Hint – you can add a data output by moving the mouse near the gray right-side of the UserObject and keying Ctrl-A.)
8. Select Menu Bar => Device => Virtual Source => Function Generator; move it to the left center inside the Vehicle Radiator UserObject.
9. Open the Function Generator object if necessary.
10. Select Tri in the Function box by clicking on the Function drop-down arrow.
11. Change Frequency to 100.
12. Connect the Function Generator data-output pin to the Vehicle Radiator UserObject data-output pin "X".
13. Convert Vehicle Radiator to an Iconic View.
    Note 1: A second rectangle labeled:"Vehicle Ra…" will appear at the bottom-left of your screen. You can access any UserObject by double-clicking on its icon.
    Note 2: As noted in Lab 1.3, if you close an object via the "X" (close) button, the object will disappear. You can only access it again via the Program Explorer.

◊ 14. Open your Program Explorer Work Area and experiment with retrieving your objects.

Monitoring the Vehicle Radiator UserObject

15. Select Menu Bar => Display => Waveform (Time); place it in the right-center of your main screen.
16. Connect the Vehicle Radiator data-output pin "X" to the Waveform (Time) input pin.
17. Run this partial program. A triangular wave should appear.
    Note: You are simulating a Vehicle Radiator that heats and cools linearly.

Modifying Vehicle Radiator temperature-related parameters

18. Open Vehicle Radiator and its Function Generator.

◊ 19. Change the Function Generator Frequency to 50 and Amplitude to 0.5.

20. Close Vehicle Radiator only.
21. Run this modified program. What effect does this change have on the scope presentation? Look closely; there is a subtle change.
22. Open Vehicle Radiator.
23. Return the Function Generator Frequency to 100 and Amplitude to 1.
24. Close Vehicle Radiator only.
    Note: If you missed it, the Trace1 scale changed so the waveform would fit on the screen. Also, the number of displayed cycles changed by a factor of 2 as the frequency was changed from 100 to 50.

Describing the Vehicle Radiator program via the Note Pad

25. Select Menu Bar => Display => Note Pad; move the Note Pad to the upper-left of the Main screen.
26. Type the following description:

    A triangular wave was used to simulate a Vehicle Radiator's temperature.

Observing the effect of parameter changes to the Vehicle Radiator program

◊ 27. Open Vehicle; change the Function Generator Time Span to 40m.
28. Run this program. Observe the effect of the Function Generator changes on the scope waveform.
    Note: The Time scale on the Waveform (Time) Object changed from 20m to 40m.
29. Save this program to your personal disk as LAB1-4.

Lab 1.5 – Using Real-World Labels in a Formula Object

This lab will introduce you to changing the labels of Object pins so they reflect the actual task involved. It is important to note that each label name must exactly match the corresponding entry in the Object data field (the Formula expression). The label name is not upper or lower case sensitive.

1. Clear your Work Area. (See Appendix B.)

Selecting program Object component parts

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of your Work Area.
3. Select Menu Bar => Device => Virtual Source => Noise Generator; place it in the lower-left-corner of your Work Area; set its amplitude to 0.25.
4. Select Menu Bar => Device => Formula; place it to the right of the other two objects.
5. Click on the Formula Object Menu button; select Add terminal => Data Input. A second input pin will appear labeled "B".

Changing pin names to real-world label names

6. Double-click on the Formula "A" input pin; change the pin name to Cosine.
7. Connect the Function Generator Func output pin to the Formula data-input pin "Cosine".
8. Connect the Noise Generator noiseWF output pin to the Formula data-input pin "B".
9. Double-click in the Formula Object white space; change its formula to read:

   Cosine + Noise
10. Select Menu Bar => Display => Waveform (Time); place it to the right of the other three objects.
11. Connect the Formula output pin Result to the Waveform input pin Trace1.
12. Move Waveform (Time) so its display shows fully in the Main window.
13. Run this program. What happened?

    Note 1: You should have observed a VEE Pro Run Time Error box with a red title bar. It should state: Variable was not found: Noise. Across the bottom of the screen is printed: Error occurred during program execution.

    Note 2: This one of the most frequent errors that occurs when you first design your own program; you forget to match the pin labels to the formula names in the expression.

Stopping and correcting a program

14. Click on the Stop button to stop the running program.
15. Double-click on the Formula "B" input pin; change its name to Noise.
16. Run this corrected program again; it should run correctly. See Figure 1.11.

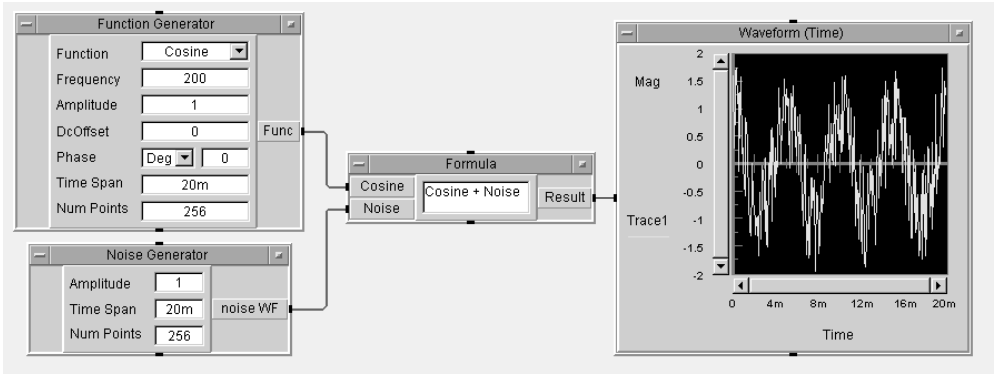


Figure 1.11. Program for Lab 1.5

17. Save this corrected program as LAB1-5.

Lesson 1 Summary

The VEE Pro development environment consists of four bars and two areas:

- the Title Bar,
- the Menu Bar,
- the Tool Bar,
- the Status Bar,
- the Program Explorer (area), and
- the Work Area.

This is the top level VEE Pro development environment. A lab-by-lab summary follows:

Lab 1.1 showed you how to start a VEE Pro program, then generate and display a waveform. You also learned how to select and use a Note Pad, a virtual Function Generator, and a virtual waveform display. You then learned how to connect these two objects to create an operable program and to run and save that program.

You also learned to: open the VEE Pro program; clear your Work Area; select, move, and size a Note Pad Object; examine the use of the Note Pad via the Help menu; select, move, and change values in a Virtual Source Object; change between Open View and Iconic View; select and move a display; create a formula; connect two objects; and test, debug, and save a program.

Lab 1.2 showed you how to add terminals, create and include UserObjects, place them within other UserObjects (nest them), work with their pop-up dialog box, and devise a simple formula.

You also learned to: devise a UserObject that will contain VEE Pro interconnected Objects; select and edit a device and convert it to an Iconic View; select a Function and Object Browser Box and create a formula within it; connect three objects within a UserObject area; display a running program with the virtual oscilloscope; change the parameters within a program; and save a program via the Menu Bar.

Lab 1.3 is where you learned to simulate a thermometer. Most modern thermometers vary in temperature quite gradually. This simulation assumes a gradual linear variation. Therefore, a triangular-wave is used. This lab shows you how to simulate an actual thermometer to provide both analog and digital readings.

You also learned to: create a virtual thermometer; monitor the thermometer UserObject; modify the thermometer temperature-related parameters; and observe the effect of parameter changes to the thermometer program.

Lab 1.4 showed you how to devise a virtual Vehicle Radiator using the Function Generator in its triangular-waveform mode. You also observed the effect of Function Generator changes on the oscilloscope display. This is the first of several practical labs that will be expanded in later lessons.

You also learned to: simulate a vehicle radiator whose temperature varies linearly; monitor the Vehicle UserObject; modify Vehicle temperature-related parameters; describe the Vehicle Radiator program via the Note Pad; and observe the effect of parameter changes to the Vehicle Radiator program.

Lab 1.5 introduced you to changing the labels of Object pins so they reflect the actual task involved. It is important to note that each label name must exactly match the corresponding entry in the Object data field (the Formula expression). The label name is not case sensitive. You also learned to: select program Object component parts; change pin names to real-world label names; and stop and correct a program.

You are now ready to prepare and test a simple program.

Details of the Menu Bar and Tool Bar objects are given in Appendix A.

A Programming Operations Guide is given in Appendix B. Starting with page B-4, it lists all of the underlined, frequently used portions of each and every lab in this manual so you can quickly locate them for later applications.

Lesson 2
Preparing and Testing a Program

This lesson will examine the fundamentals required to prepare and test a program. It consists of a pre-lab summary and four labs.

Lesson 2 Pre-lab Summary

The following are described in the Lesson 2 Pre-lab. See Appendix E, page E-16:

- The VEE Pro Opening Development Screen
- Bitmap Transferring
- Menu Bar => Device => Function & Object Browser
- Menu Bar => Data => Constant => UInt8
- Icon Title Bar
- Main Title Bar; Enter Description
- Program Documentation
- Menu Bar => Data => Continuous => Real64 Slider (or Knob)
- Menu Bar => Data => Dialog Box => Int32 Input
- Menu Bar => I/O => To => File
- Creating an Operator Interface
- Switching between the Detail View and the Panel View
- Changing colors and fonts on Panel
- Changing colors and fonts on Objects
- Menu Bar = > I/O => To => File
- Menu Bar = > I/O => From => File
- Alphanumeric Display Selection
- Print Screen
- Deleting unneeded objects and connecting lines
- Deleting unneeded terminals (pins)
- Open or Save a File

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items, and to all objects and subprograms contained in the labs of this and later lessons.

Overview

Lab 2.1 – Generating Random Number Test Data
This lab will show you how to: describe the random-number program within a Note Pad; select, clone, and modify Objects; and monitor a changing program with an alphanumeric display.

Lab 2.2 – Devising a Pulse Program
This lab will show you how to: describe your program within the Main menu description box; document your program; create a dialog box; add and restrict the parameters of a virtual device; vary the parameters of a program; discover the effect of entering an incorrect parameter value; change program parameters via the Real Slider or (Real Knob); add two device waveforms together; force input parameters so the waveform stays within the Y axis scale; store the program results in a file using the ToFile object; examine the contents of ToFile; create an operator interface; switch between the Panel View and Detail View; change colors on the Panel; and change colors and fonts on an Object.

Lab 2.3 – Monitoring Visually Vehicle Radiator Test Data
This lab will show you how to: modify a Note Pad description; examine and interpret a virtual scope waveform; and print a VEE Pro screen.

Lab 2.4 – Logging Vehicle Radiator Test-Data Values
This lab will show you how to: revise PulseProgram to allow Vehicle Radiator simulation; change object names to be more descriptive; eliminate unneeded objects and pins; change object internal parameters; create Alphanumeric displays for displaying single or multiple values; and examine the contents of FromFile.

Lab 2.1 – Generating Random Number Test Data

This lab will show you how to generate random-number test data by creating a formula object, clone an Object, set up an alphanumeric display (page C-15), change the test-data values, and observe the icon(s) in the Program Explorer window area.

Open your VEE Pro program and

1. Clear your Work Area.
   Note: If the Program Explorer work area appears, then proceed to step 2. Otherwise, go directly to step 3.
2. Select Menu Bar => View => Program Explorer. Remove its checkmark so you access the entire screen.

Describing the random-number program within a Note Pad

3. Select Menu Bar => Display => Note Pad; place it in the top center of Main work area.
4. Click on the (white) editing area; a cursor will appear.
5. Type the following, using the return or enter key to start the next line:
   This program, Random, generates
   a real number between zero and one and
   displays the result.

6. Size your Note Pad so all of the text appears.

Selecting, cloning, and modifying objects

7. Select Menu Bar => Device => Function & Object Browser.
8. Select Type: Built-in Functions; Category: Probability & Statistics; and Member: random.
9. Click on Create Formula – the Object random(low,high) will appear.
10. Place random(low,high) in the center of Main Work Area, below Note Pad.
11. Select Menu Bar => Data => Constant => Uint8; place it to the left and slightly above random(low,high).
12. Right-click anywhere on Uint8; select Clone.
    Note 1: The horizontal bar in the upper-left corner is the Object Menu button. It can be left-clicked to obtain the object menu.
    Note 2: The Object must be on the screen before it can be cloned.
13. Place the cloned Uint8 object below the original integer object and slightly below random(low,high).
14. Double-click on 0 in the clone; the 0 will be highlighted; change the 0 to a 1.
15. Connect the Uint8 0 object data-output pin to the random(low,high) low data-input pin.
16. Connect the Uint8 1 object data-output pin to the random(low,high) high data-input pin.

Monitoring a changing program with an alphanumeric display

17. Select Menu Bar => Display => AlphaNumeric; place it to the right of random(low,high).
18. Double-click on the AlphaNumeric Title Bar; Change its title to Test Data; click OK.
19. Connect the random(low,high) data-output pin to the AlphaNumeric data-input pin.
20. Click the Run button on the tool bar. A number will appear in the AlphaNumeric object. See Figure 2.1 below.

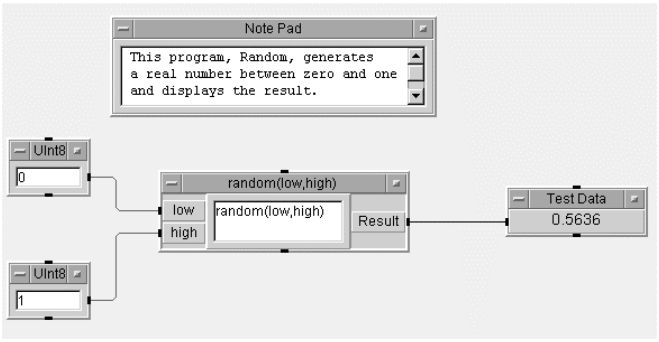


Figure 2.1. The Random program

21. Click the Run button several times; observe that the number in the AlphaNumeric object changes.
    Note: If an "m" appears with the numeric, then it is indicating that this value is to be divided by 1000. The AlphaNumeric default number format uses the standard ISO abbreviations, such as *m* for milli and μ for micro.
22. Move the objects to the left as needed for all objects to appear on the screen.
23. Save this program as LAB2-1.

24. Select Menu Bar => View => Program Explorer. The saved program title will appear in the upper-left corner of the Program Explorer area.

Lab 2.2 – Devising a Pulse Program

This lab will show you how to devise a pulse program. Other techniques will be introduced, including: how to describe your program within the Main menu description box, document your program, and create a dialog box (Figure 2.2 and Appendix D). You will also learn to add and restrict the parameters of a virtual device, vary the parameters of a program, discover the effect of entering an incorrect parameter value, and change program parameters via the Real Slider or Real Knob. See Figure 2.5. Furthermore, you will learn to add two device waveforms together and choose input parameters that force the resulting waveform to stay within the Y axis scale; store the program results in a file using the ToFile object and examine the contents of ToFile; create an operator interface (Figure 2.5), switch between the Panel View and Detail View (Figure 2.5), and change colors on a Panel, and colors and fonts on an Object.

Open your VEE Pro program and

1. Clear your Work Area.
   Note: The following is an alternative to the Note Pad when workspace is tight.

Describing your program within the Main menu description box

2. Click, with the right-hand mouse button, on the Main Title Bar; then click on Description in the resulting drop-down object menu.
3. Type the following text into the dialog box exactly as shown below using the return or enter key to start a new line:

   PulseProgram generates a pulse waveform, adds noise, stores the results,
   and displays the noisy pulse. The user will select the desired frequency
   and amount of noise each time the program runs. An operator interface will
   show only a slider bar (to allow the user to select the amount of noise input)
   and a waveform display. Frequency will be chosen via a pop-up box.

   Note: You can insert a file or a template into the Description dialog box.
4. Click OK; then enlarge the Main window workspace by clicking the middle (square) maximize button at the right side of the Main Title Bar.
   Note: You can maximize a window by clicking the middle (square) button on the right side of the Main, UserObjects, and UserFunctions title bars.

Creating a dialog box

5. Select Menu Bar => Data => Dialog Box => Int32 Input; place it in the upper-left corner of the Main window; change the title to Number Input.
6. Change Prompt/Label field to: Enter Frequency:
7. Change the Default Value to: 100.
8. Change Value Constraint and Error Message to 1 on the low end and 193 on the high end as in Figure 2.2. Click the upper-right button of Number Input to convert it to an Iconic View and drag it to the upper left corner.
   Note: The Error Message should read: You must enter an Int32 between 1 and 193.

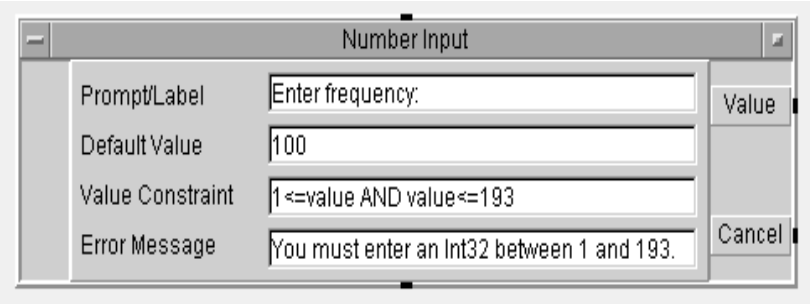


Figure 2.2. The Int32 Input Configuration box

9. Select Menu Bar => Device => Virtual Source => Pulse Generator; place it to the right of the Number-Input Dialog-Box object. Change its size, if necessary, to conserve workspace.

Adding and restricting the parameters of a virtual device

10. Open the Pulse Generator object menu; click Add terminal => Data Input.
11. Select Frequency; click OK.
12. Connect the top data-output pin (Value) of the Number Input object to the Pulse Generator data-input pin (Frequency).
    Note: You can no longer edit the Frequency input field of the Pulse Generator. The value will be set by the data input.

Varying the parameters of a program

13. Run this portion of your program; a pop-up box will appear. See Figure 2-3.

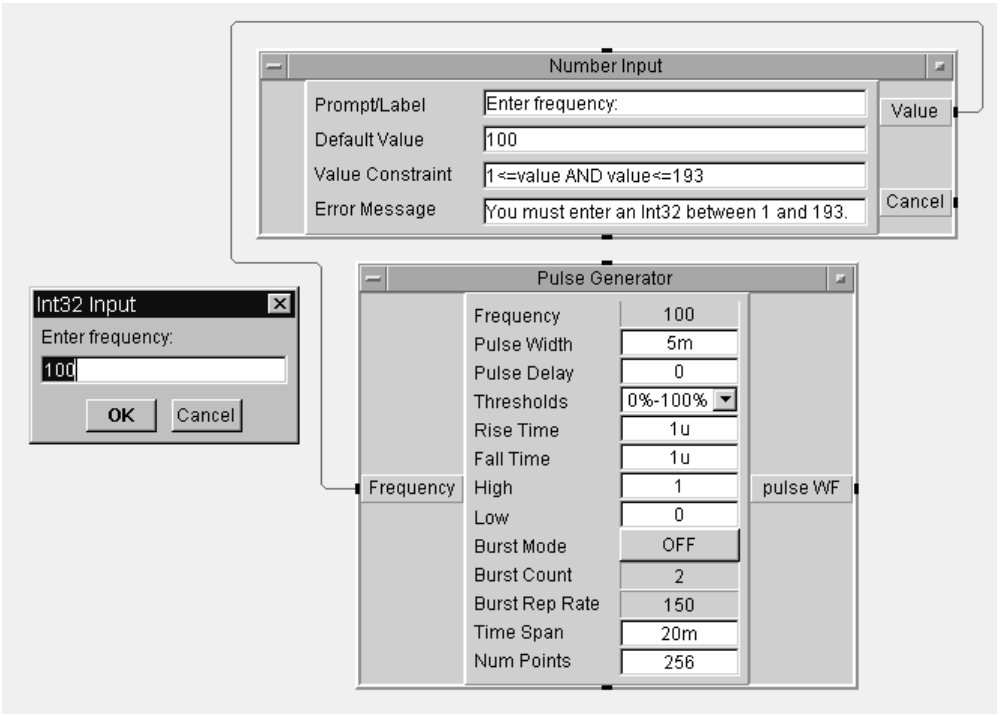


Figure 2.3. The Initial PulseProgram

14. Enter the value: 150 in the Int32 pop-up box under: Enter Frequency; click OK. (Observe the entry in the Pulse Generator Frequency field.)
    Note: You may click and drag the pop-up box to a different location.
15. Convert Number Input and Pulse Generator to Iconic Views; move them closer to each other.
16. Repeat steps 14 and 15 several times; observe the entry in the Pulse Generator Frequency field each time.

Discovering the effect of entering an incorrect parameter value

◊ 17. Run the program; then enter the value
    200 in the pop-up box; click OK;
    0.12 in the pop-up box; click OK;
    185 in the pop-up box; click OK;
    -4 in the pop-up box; click OK;
    182.6 in the pop-up box; click OK.
    For each of the above-entered values, stop the program whenever an error box appears.
18. Simplify the screen display: convert the Pulse Generator to an Iconic View and move it closer to the input box; drag both to the top-center of the workspace.

Changing program parameters via the Real Slider (or Real Knob)

19. Select Menu Bar => Device => Virtual Source => Noise Generator; place below Pulse Generator.
20. Open its object menu; select Add terminal => Data Input => Amplitude; click on OK.
21. Select Menu Bar => Data => Continuous => Real64 Slider; place it to the left of the Noise Generator.
    Note: As later labs will demonstrate, the slider object may be replaced by a knob. The tick marks on either of them can be spaced either linear or logarithmic. These tick marks can be turned on or off. The maximum range can be changed by typing a new value into the lower white box.
22. Double-click on the Real64 Slider Title Bar; change its Title to become: Enter Noise; click OK.
23. Set the Noise slider bar to a value other than zero; reduce the size of the Enter Noise object.
24. Connect the Enter Noise data-output pin (Real64) to the Noise Generator Amplitude data-input pin.
25. Convert the Noise Generator to an Iconic View; move it closer to the Enter Noise input box.

Adding two device waveforms together

26. Select Menu Bar => Device => Function & Object Browser; Type: Operators, Category: Arithmetic, Operator: +; click on Create Formula. The object A+B will appear.
    or
    Click f_x on the Tool Bar, then complete the selections noted above.
27. Place A+B to the right of, and between, Pulse Generator and Noise Generator. Convert A+B to an Iconic View; move it between the Pulse Generator and Noise Generator objects.
28. Connect the Pulse Generator, Noise Generator, and A+B as shown in Figure 2.4.

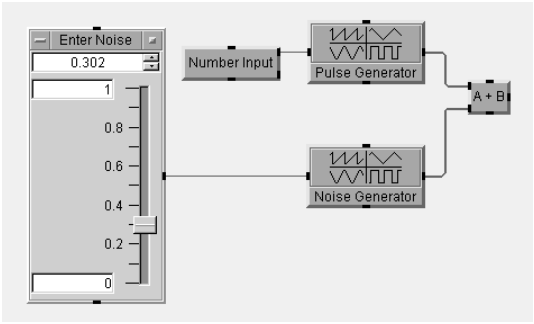


Figure 2.4. Partial PulseProgram with two waveforms added

29. Select Menu Bar => Display => Waveform (Time); place to the right of A+B.
30. Connect the A+B data-output pin (Result) to the Waveform (Time) data-input pin (Trace#1).
    Note: From now on, the Waveform (Time) display will be referred to as your virtual scope.
31. Run your program. Follow instructions given in the pop-up boxes. A pulse will appear with noise added to it. See Figure 2.5.

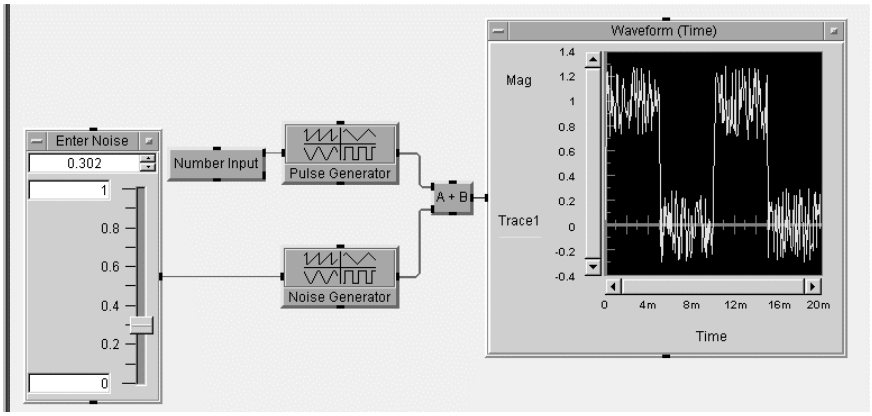


Figure 2.5. PulseProgram after running

◊ 32. Change the slider value; re-run your program and observe the changes on your virtual scope.
◊ 33. Repeat step#32, change the frequency also several times, and observe the changes on your virtual scope.

Forcing input parameters so the waveform stays within the Y axis scale

34. Turn off the Noise Generator by decreasing the Enter Noise slider to zero.
◊ 35. Open the Pulse Generator object; change its High value to 0.9 and its Low value to 0.1. Return the Pulse Generator to an Iconic View.
◊ 36. Run your program again and observe the change on the virtual scope Y axis extremes.

Storing the program results in a file using the ToFile object

Note: The ToFile object provides access to a location for storing program data. This is an introduction to the File object family; detailed coverage will be described in Lab 5.1.

37. Select Menu Bar => I/O => To => File; place it in the center of the Work Area; it will overlay the existing objects until converted to an Iconic View in a later step.
38. Click on myFile; change the name from myFile to report.txt.
39. Click in the small white square to the right of the Clear File At PreRun & Open to cause a checkmark to appear. Double-click on the words in the white area of the ToFile box that says "Double-Click to Add Transaction"; click OK; an input pin (a) will appear.
40. Convert ToFile to an icon.
41. Connect the A+B data-output pin (Result) to the ToFile data-input pin.
◊ 42. Run your program several times with different settings of the Enter Noise slider and observe the change on the virtual scope.

Examining the contents of ToFile

43. Double-click on ToFile for an open view; then double-click on its input "a" pin; examine its contents. (Data should be numbered from 0 through 255.)
Note: Small numerical values are represented as 0.xxxm, where the *m* stands for "milli".
44. Click OK.

Creating an operator interface

45. Press and hold the Ctrl key; click on both the Enter Noise and Waveform (Time) Objects; release the Ctrl key.
    Note: Each Object will display a shadow on the right side and bottom to indicate that it has been selected.
46. Open the Main Window Object Menu by placing the mouse pointer over the Main window (gray or white) background and clicking on the mouse right-hand button.
47. Click Add to Panel; the selected objects will appear in the Panel View. See Figure 2.6.

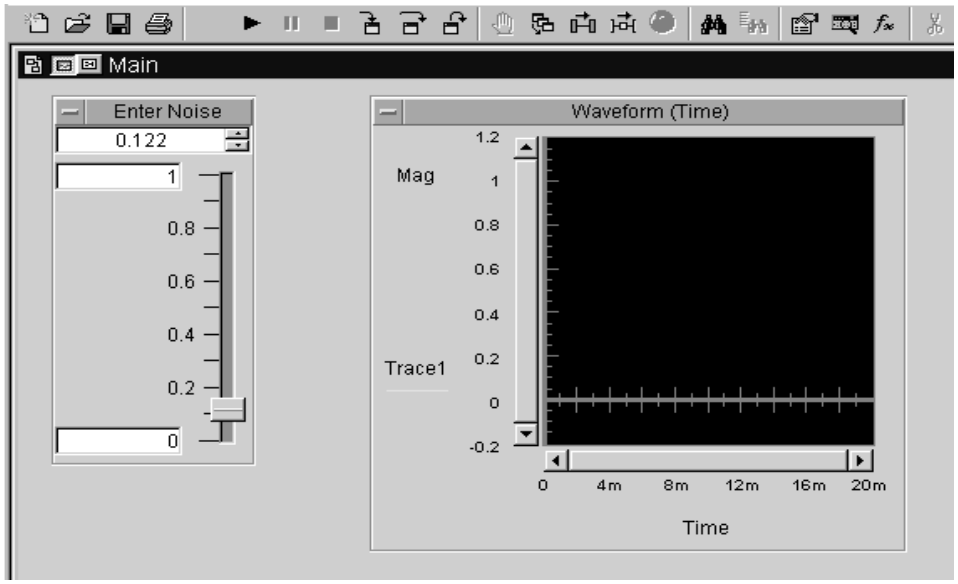


Figure 2.6. The Panel View of PulseProgram

Note: Two more icons appear beside the word Main on the Work Area Title Bar.

Switching between the Panel View and Detail View

48. Toggle between the two new icons on the Main Work Area bar (in the top right-hand corner). These icons will switch you between your program's Panel View and its Detail View.

◊ 49. Run your program several times with different settings of Enter Noise and Enter Frequency; observe the changes on your virtual scope.

50. Save your program as LAB2-2.

Changing colors on the panel

51. Return to the Panel View.
52. Select Properties from the object menu. (It is the left icon on the Panel View Title Bar.)

◊ 53. Select the Colors tab; click the Title or Object Background button; Select Color for Object (or Object Title) will appear; change both colors if desired.

54. Select your color(s); click OK.
55. Click OK on Main Properties dialog box.

Changing colors and fonts on an object

56. Double-click on the Operator Interface Main Title Bar; the Properties box will appear.
◊ 57. Click on Colors.
58. Select your preference; click OK twice to return to the Panel View.
59. Save your program again.

Documenting your program

Warning: In VEE 5.01, you must save a program before requesting Documentation because Documentation converts the program into Visual Basic. Version 6 and beyond have removed this problem.

Note: The program documentation is a description of the software composition of the program elements.

60. Select Menu Bar =>File => Save Documentation. Name this file: LAB2-2D; then click Save.
Note: You can insert a file or a template into the Documentation dialog box.

Lab 2.3 – Monitoring Visually Vehicle Radiator Test Data

This lab will show you how to modify the Note Pad description, change the Function Generator amplitude and DcOffset, learn how to adjust and read the virtual scope, print the VEE Pro screen, and run the program.

Open your VEE Pro program and

1. Clear your Work Area.
2. Select Menu Bar => File => Open => LAB1-4 in the VEE Pro folder; save as LAB2-3 immediately.

Modifying a Note Pad description

3. Modify the Note Pad description to read:
   A triangular wave is used to simulate a
   Vehicle Radiator's temperature change.
4. Size the Note Pad if necessary.
5. Double-click on the Vehicle Radiator object; change the Thermometer Amplitude to "10" and its DcOffset to "190". (Temperature will vary from 190 ± 10°F.)
   Note: For those of you who prefer to work with the metric scale, the conversion result is: 88 ± 6°C to the nearest integer.
6. Return the Vehicle Radiator Object to an Iconic View.
7. Run the program.

Examining and interpreting a virtual scope waveform

8. Examine the Waveform (Time) Trace1.
   Note: Its "Mag" starts at "190" and peaks at "200". Its minimum is "180". Thus, the virtual scope does not have a trace with a displayed zero reference value.

Printing a VEE Pro screen

9. Center the program you want to print on the Work Area.

Hint: Another option to center the entire program in the Work Area is to place the pointer over any portion of the Main white space and depress the keyboard key: Home. The entire program will be moved to the upper left of the Work Area.

10. Configure your printer; then turn on your printer.
11. Select Menu Bar => File => Print Screen. See Figure 2.7.

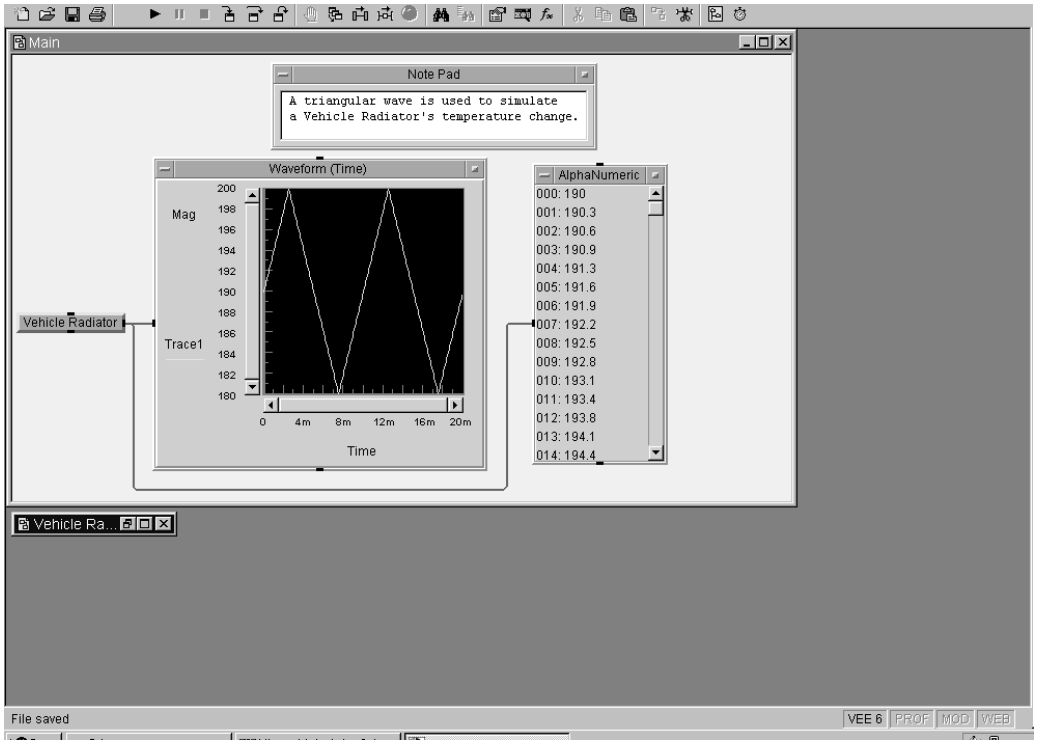


Figure 2.7. Result of Print Screen

12. Save this program again.

Lab 2.4 – Logging Vehicle Radiator Test-Data Values

This lab will show you how to revise PulseProgram to allow Vehicle Radiator simulation, change object names to be more descriptive, eliminate unneeded objects and pins, change object internal parameters, and create alphanumeric displays for displaying single or multiple values.

Open the VEE Pro program and

1. Clear your Work Area.

2. Open LAB2-2 (PulseProgram).
3. Select Menu Bar => File => Save As…; save this program as LAB2-4 immediately.

Revising PulseProgram to allow Vehicle Radiator simulation

4. Go to the Main Title Bar, click the left icon, and select Description from the Main object menu.
5. Press Clear All; then modify the PulseProgram description to read:
   LAB2-4 modifies PulseProgram so the slider bar can be used
   to vary the incoming noise (that will represent temperature) in
   an alphanumeric display.
6. Click OK.

Changing object names to be more descriptive

7. Change to the Iconic View, change the name of Enter Noise to Enter Temp, and change the name of Noise Generator to Temp Generator; convert it back to a Panel View.

Eliminating unneeded objects and pins and replacing objects

8. Click on the Int32 Input box; use the Tool Bar "cut" scissors to remove it.
9. Click on the right mouse button while on the Pulse Generator object; delete the Frequency Data Input pin using the Object Menu that appears; click OK.
10. Replace the Pulse Generator with a Function Generator using the following general method:
    a. Create and define the new object including all matching input and output pins.
    b. Click on the mouse right button while over the object to be replaced.
    c. Select Replace from the Object Menu.
    d. Click on the newly created object; position it as desired; it will be connected in the same way as was the original object.

    Note: Create and define the new (replacement) object (Function Generator) as indicated below.

Changing object internal parameters

11. Set the Function Generator fields to Function: Tri; Amplitude: 2; DcOffset: 100.
12. Change its title to: Function Generator (Tri).
13. Convert Function Generator (Tri) to an Iconic View.
14. Cut Waveform (Time) with the Tool Bar scissors.

Setting ToFile and FromFile Transactions

15. Select Menu Bar => I/O => From => File; place it below the ToFile object.
16. Click on myFile; change the name from myFile to report.txt to exactly match the ToFile name; click OK.
    Note: The name chosen within ToFile and FromFile must match an existing .vee or .dat name on the list that will appear when "MyFile" is clicked once.
17. Modify the I/O transaction into ToFile by double-clicking on "WRITE TEXT a EOL". Change the settings as shown in Figure 2.8.
    Note: The "WRITE TEXT" transaction will write "a" to the file. If "a" contains an array, then all those values are written to the file.

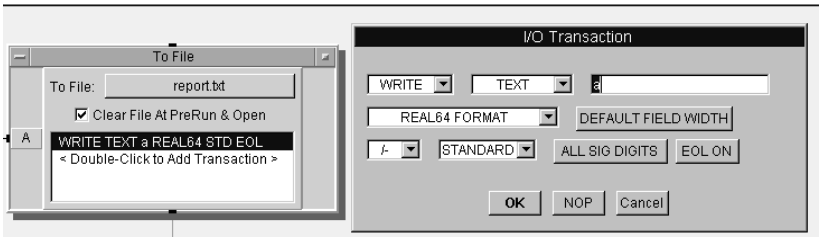


Figure 2.8. The ToFile Transaction dialog box

18. Insert an I/O transaction into FromFile by double-clicking on 'Double-Click to Add Transaction". Change the settings as shown in Figure 2.9.
    Note: The "READ TEXT" transaction is configured to read a single scalar value.

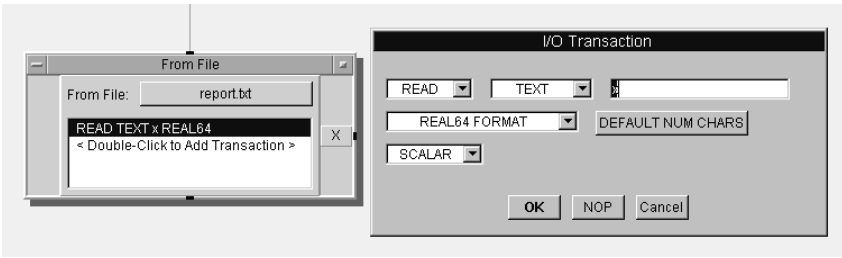


Figure 2.9. The FromFile Transaction dialog box

19. Connect the bottom pin of ToFile to the top pin of FromFile; convert ToFile and FromFile to icons.

Creating Alphanumeric displays for displaying single or multiple values

20. Select Menu Bar => Display => AlphaNumeric; place to the right of FromFile; click anywhere.
21. Connect the FromFile data-output pin to the AlphaNumeric data-input pin.
22. Double-click on the AlphaNumeric Title Bar; change its title to Vehicle Temp; click OK.

◊ 23. Run this program several times by entering different values in Enter Temp; observe the Vehicle Temp display and note that it presents but a single value.

24. If you wish to log a series of values as in Figure 2.10:

- Select Menu Bar => Display => Logging AlphaNumeric,
- Place it below Vehicle Temp,
- Open the Property box and change its name to Vehicle Temp Log,
- Remove the two checkmarks from "Initialization" within its Property box, and
- Connect its input pin to the FromFile output pin.

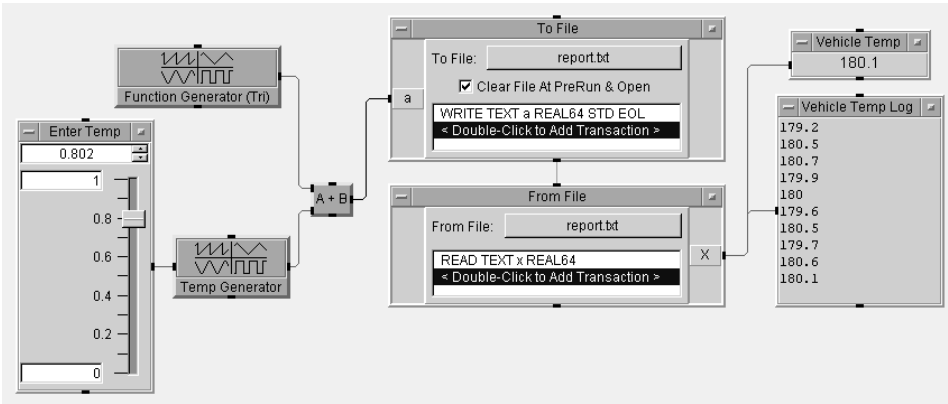


Figure 2.10. Logging vehicle temperature results

◊ 25. Run this program several times; observe the Vehicle Temp Log display.
Note: Observe the size of the Buffer Size Lines; the default is 256.
26. Click OK.

Examining the contents of FromFile

27. Double-click on FromFile for an open view; then double-click on its "X" output pin to examine its contents.

◊ 28. Run this program again several times and observe the data at the FromFile Output Pin relates to the last value in the Logging Alphanumeric Display.
Note 1: Observe the values on the input to the ToFile object. The FromFile object is only reading the first value written to the file since the READ transaction is configured to read a Scalar value. It can be configured to read the entire array of values.
Note 2: The program will not run unless you first close the Output Pin Information dialog box. Open it each time after you run your program; close it (click OK) before you try to run your program again.
29. Create an operator interface: Press and hold Ctrl; click on Enter Temp and Vehicle Temp Log.
30. Place the mouse pointer over the Main window background and click on the mouse right button.
31. Click Add to Panel; Enter Temp and Vehicle Temp Log will be added to the Panel View. See Figure 2.11.

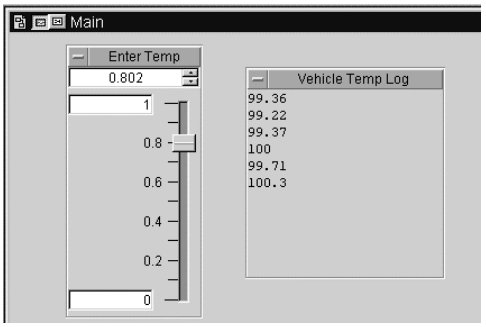


Figure 2.11. Panel view for Lab 2.4

32. Select Menu Bar => File => Save As…; name your modified program: LAB2-4LOG.
    Note: A later program will show you how to transfer these data points to a spreadsheet.

Lesson 2 Summary

This lesson illustrated how to generate a random number, modify Objects, and monitor the program with various alphanumeric displays. Also, you learned how to document a program, revise incorrect program parameters, vary parameters, and create Panel and Detail views. Then you learned to add two device waveforms together and change input parameters. Further, you learned how to add generator waveforms, use ToFile and FromFile, examine their contents, and create an Operator Interface.

Lab 2.1 showed you how to generate random-number test data by creating a formula object, clone an Object, set up an alphanumeric display, change the test-data values, and observe the object(s) in the Program Explorer window area. You also learned to: describe the random-number program within a Note Pad; select, clone, and modify Objects; and monitor a changing program with an alphanumeric display

Lab 2.2 showed you how to devise a pulse program, how to describe your program within the Main menu description box, document your program, and create a dialog box. You also learned to add and restrict the parameters of a virtual device, vary the parameters of a program, discover the effect of entering an incorrect parameter value, and change program parameters via the Real Slider or (Real Knob).

Lab 2.3 showed you how to modify the Note Pad description, change the Function Generator amplitude and DcOffset, learn how to adjust and read the virtual scope, print the VEE Pro screen, and run the program that monitors visually vehicle radiator test data. You also learned to modify a Note Pad description; examine and interpret a virtual scope waveform; and print a VEE Pro screen.

Lab 2.4 showed you how to revise PulseProgram to allow Vehicle Radiator simulation, change object names to be more descriptive, eliminate unneeded objects and pins, change object internal parameters, and create Alphanumeric displays for displaying single or multiple values. You also learned to: revise PulseProgram to allow Vehicle Radiator simulation; change object names to be more descriptive; eliminate unneeded objects and pins; change object internal parameters; create Alphanumeric displays for displaying single or multiple values; and examine the contents of FromFile.

You are now ready to select and control a variety of virtual instruments.

Lesson 3
Controlling and Communicating to Instruments

This lesson will examine the various ways that VEE can control and communicate to instruments. It consists of a pre-lab and four labs.

Lesson 3 Pre-lab Summary

The following are described in the Lesson 3 Pre-lab. See Appendix E, page E-30:

- Controlling and Configuring Instruments via Panel Drivers and Direct I/O
- Menu Bar => I/O => Instrument Manager…
- Menu Bar => Flow => Start
- Menu Bar => Flow => Do
- Menu Bar => Flow => Repeat => For Count
- Menu Bar => Data => Collector
- Menu Bar => Device => Timer
- Menu Bar => Device => UserObject

As noted in Lessons 1 and 2, Appendix B includes a cross-reference to each of these items and to all objects and subprograms contained in the labs of this and later lessons.

Overview

Lab 3.1 – Configuring a GPIB Instrument with a Panel Driver
This lab will show you how to configure an instrument from the Menu bar to use a Panel Driver, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Lab 3.2 – Configuring a GPIB Instrument for Direct I/O
This lab will show you how to configure an instrument from the Menu bar to use Direct I/O, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Lab 3.3 Using Random Number Programs for Test Development
This lab will show you how to create a program that generates 12 random numbers, displays them, and indicates the total time required to generate and to display the values.

Lab 3.4 – Devising a Virtual Vehicle Radiator
This lab will show you how to write a file description that will become part of the printed documentation, modify a partially developed virtual vehicle radiator to create varying temperatures and companion pressures, and monitor those two parameters simultaneously on a virtual scope.

Lab 3.1 – Configuring a GPIB Instrument with a Panel Driver

This lab will show you how to configure an instrument from the Menu bar to use a Panel Driver, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Open VEE and

1. Clear your Work Area, maximize Main; and toggle Program Explorer off.

Reconfiguring an existing scope Panel Driver object

2. Select Menu Bar => I/O => Instrument Manager… ; see Figure 3.1.

Note: When VEE is loaded and used for the first time, there will be no instruments displayed in the Instrument Manager icon.

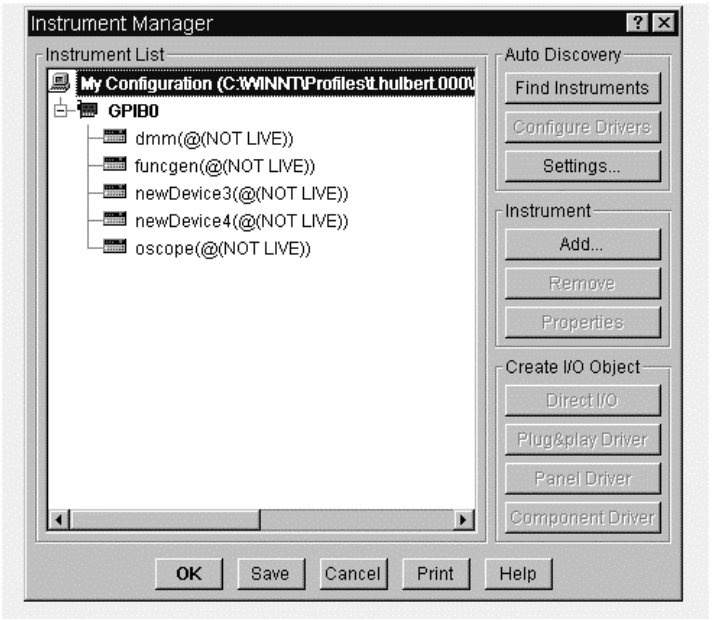


Figure 3.1. The Instrument Manager box

3. Move the dialog box, by dragging its Title Bar, if you so desire; "My Configuration" should be highlighted.
4. Click Add under Instrument… . This displays the Instrument Properties Dialog Box, which presents the following fields:

Name: Enter scope. (It would be wise for all of you working together to agree on a naming procedure. This will allow you all to understand each other's programs. Later you can save time by choosing already existing virtual instruments. We have used hp54504a.)
Interface: Select GPIB.
Address: The select code of the interface (GPIB is usually 7) plus the local bus address of the instrument (which is a number from 0 to 31).Type 0 in the Address: field because you are developing a program without an instrument present (NOT LIVE).
Gateway: Use "This host" to control instruments locally.

Note: Pressing the Tab key after typing in a field will move you to the next field; pressing Shift-Tab will move you to the previous field.

5. Leave all the other defaults as they are given.
6. Click on Advanced … or Advanced I/O Config…, whichever appears.
7. Leave Timeout (sec): at 5.
8. Toggle Live Mode to OFF if it is not dimmed (grayed out).
   Note: Live Mode should be OFF and dimmed because the address is set to 0 and no real instrument is connected.
9. Leave Byte Ordering: at MSB (as required by all IEEE488.2-compliant devices).
10. Type in Description (optional): hp54504a; this description will appear on the Title Bar.
11. Click on the Panel Driver tab.
12. Click on the blank space to the right of ID Filename: A list of instruments will appear entitled: Read from what Instrument Driver?
13. Scroll horizontally until you reach hp54504a.cid; click on it. See Figure 3.2 which contains all the dialog boxes open.

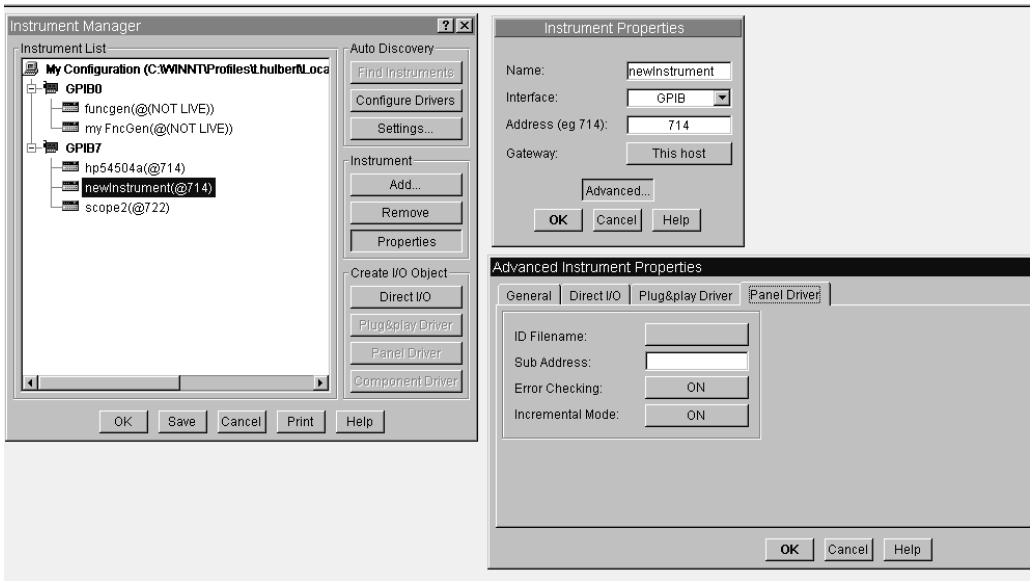


Figure 3.2. Instrument Manager programmed for a virtual scope

14. Click Open; then click OK when Advanced Instrument Properties appears.
15. Return to the Instrument Manager Dialog box; click the Save button.
    Note 1: An instrument object named Scope2 using the driver file hp54504a@(NOT LIVE) is now in your list of available instruments. It does not have a bus address specified because it is NOT LIVE at this time.
    Note 2: You can develop your program in this mode and add an address later when there is a live instrument.
16. Select Menu Bar =>I/O => Instrument Manager; highlight scope…NOT LIVE; under Create I/O object, click Panel Driver – the hp54504a @ (NOT LIVE) Object will appear on the screen. See Figure 3.3.
    Note: You are ready to connect a real instrument to your computer and select an active address.
17. Save this program as LAB3-1 to your floppy disk, drive A:.

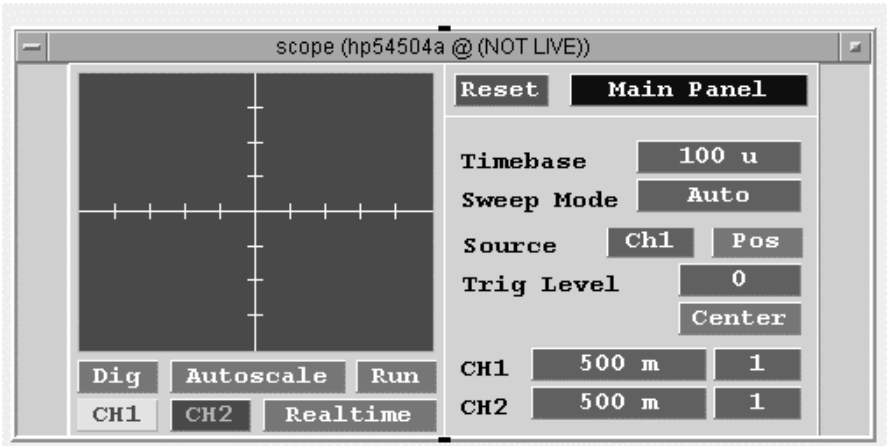


Figure 3.3. Selected hp54504a @ Not Live

Lab 3.2 – Configuring a GPIB Instrument for Direct I/O

This lab will show you how to configure an instrument from the Menu bar to use Direct I/O, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Open VEE and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Configuring a Function Generator for Direct I/O

2. Select Menu Bar => I/O => Instrument Manager… .
3. Select My configuration if it is not the default folder.
4. Click Instrument => Add … . This displays the Instrument Properties dialog box.
5. Edit the Name field to read: myFncGen
6. Press the Tab key twice. (Interface should default to GPIB.)
7. Change the address to 713; VEE will now assume that the instrument is LIVE on the bus.

Note: The VEE Pro User's Guide provides information regarding bus-address.

8. Click Advanced.
9. Toggle Live Mode to OFF. VEE will now assume that the instrument is NOT LIVE.
10. Select the Direct I/O tab.
11. Examine available options; the settings should agree with those of Figure 3.4; click OK.

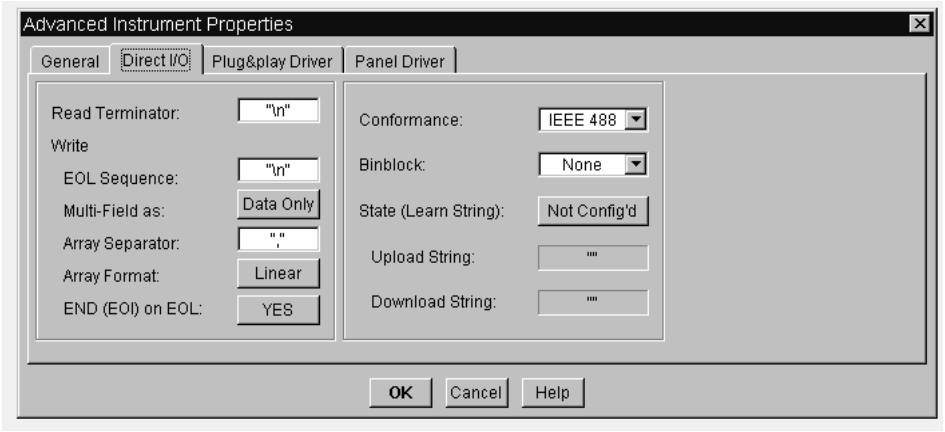


Figure 3.4. The Direct I/O Advanced Instrument Properties folder

12. Click OK on the Instrument Properties dialog box; the screen will return to Instrument Manager.
13. Click on Save; myFncGen(@NOT LIVE) will remain in the Instrument List dialog box.
    Note 1: You may have to open the Instrument Manager again.
    Note 2: This process uses the (IEEE488) GPIB interface. See VEE Pro User's Guide if you wish to configure Serial, GPIO, or VXI instrumentation.
14. Select Menu Bar => I/O => Instrument Manager; highlight myFncGen (@NOT LIVE) in the GPIB7 folder; click on Direct I/O. The object's sequence pins will be at the top and bottom of its Detail View. See Figure 3.5, left side.
15. Double-click on the myFncGen "Double-Click to Add Transaction" bar. The I/O Transaction box will appear. See Figure 3.5, right side.

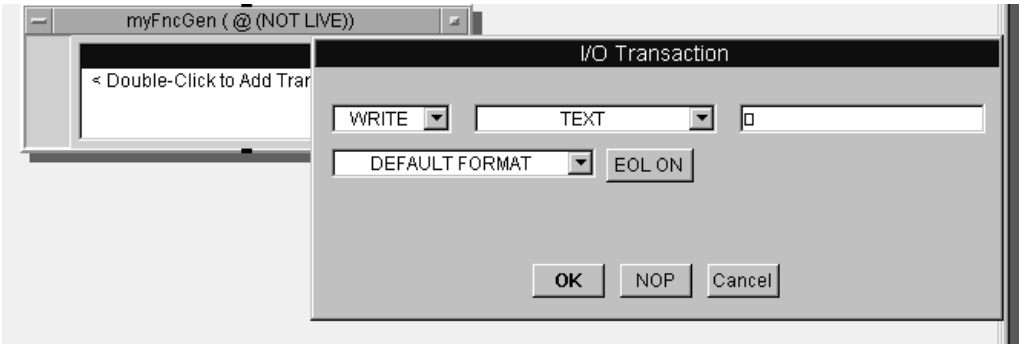


Figure 3.5. myFncGen and its I/O Transactions dialogb

16. Click OK; an input terminal "a" is added. Now the Direct I/O object can write the data on its "a" input to the instruments.
17. Save this program as LAB3-2.

Lab 3.3 – Using Random Number Programs for Test Development

This lab will show you how to create a program that generates 12 random numbers, displays them, and indicates the total time required to both generate and display the values.

Open VEE; clear your Work Area, maximize Main; toggle Program Explorer off.

Devising a random number generator

Note: Many programmers avoid using the Start button; they prefer to use the Run button only; you may prefer to begin with step 2..

1. Select Menu Bar => Flow => Start; place it in the upper-left of your screen.
   Note: Once the Start button is placed, it is moved by using the mouse right button..
2. Select Menu Bar => Flow => Do; place it under Start (or in the upper-left of your screen).
3. Connect the output of Start to the input (top pin) of Do.
4. Select Menu Bar => Flow => Repeat => For Count; place it under Do.
5. Connect the bottom pin of Do to the top pin of For Count; set the count value to 12.
6. Select Menu Bar => Device => Function & Object Browser => Type: Built-in Functions; Category: Probability and Statistics; Functions: Random.
7. Click on Create Formula; place the object to the right of For Count.
8. Go to the random Properties box and delete both input pins first low, then high.
9. Change the title of random(low,high) to random(0,10).
10. Change the random-number range in the edit (white) space from its generic low,high to the values 0,10.
11. Connect For Count output (right-hand) pin to the random(0,10) sequence input (top) pin.
12. Select Menu Bar => Display => Logging AlphaNumeric; place it to the right of random(0,10).
13. Select Menu Bar => Flow => Do; place it below For Count.

14. Select Menu Bar => Data => Collector; place it below random(0,10); connect its Data (input) pin to the random(0,10) output (Result) pin.
15. Select Menu Bar => Display => Logging AlphaNumeric; change its name to Random Numbers via its Properties box; connect its input to the Collector Output (Array).
16. Select Menu Bar =>Device => Formula; place it below the second Do object and to the left of Collector; change the formula to A+1.
17. Connect the output pin of For Count to the Formula input.
18. Select Menu Bar => Display => AlphaNumeric; place it to the right of the Formula; change its name to Final Count.
19. Connect the output of Formula to the input of Final Count adding "1" to For Count.
20. Select Menu Bar => Device => Timer; place it to the right of Random Numbers; change it name to Elapsed Time.
21. Connect the upper Do object output pin to the Elapsed Time top input pin.
22. Connect the lower Do object output pin to the Elapsed Time bottom input pin.
23. Connect the lower Do object bottom pin to the bottom-left (XEQ) Collector pin.
24. Run this program; it should look like Figure 3.6.
    Note: The Formula Object, along with its Final Count (Alphanumeric) Object confirm the number of loops performed by the For Count Object. Elapsed Time (the timer) indicates approximately how long it takes for the twelve loops to run.

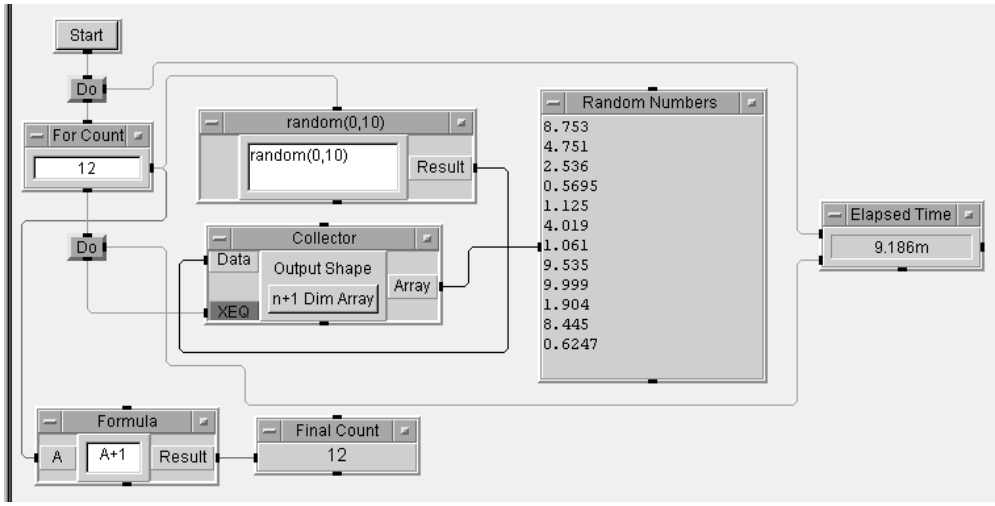


Figure 3.6. Generating and monitoring random data

25. Save this program as LAB3-3.
    Note: See Lesson 18 for an improved version that will run much faster.

◊ 26. Experiment with this program and assess the changes:

- Change the value of For Count.
- Change the range of the low,high values in random(0,10), both low and high.
- Remove the Formula object; connect For Count directly to Final Count.

- Turn on the Show Data Flow button in the Tool Bar; monitor the data flow.
- Turn on the Show Execution Flow button in the Tool Bar; run the program again.

27. Close this program *without* saving it.

Lab 3.4 – Devising a Virtual Vehicle Radiator

This lab will show you how to write a file description that will become part of the printed documentation, modify a partially developed virtual vehicle radiator to create varying temperatures and companion pressures, and monitor those two parameters simultaneously on a virtual scope.

Open VEE Pro; clear your Work Area, maximize Main; toggle Program Explorer on.

Devising a virtual Vehicle Radiator

1. Select the Main Title Bar left-hand side icon; select Description; Description of "Main" will appear.
2. Type the following into the white space exactly as shown:
   This lab will begin the simulation of a vehicle radiator. This vehicle radiator will have both temperature and pressure as variable parameters. The temperature will be measured in degrees F; the pressure will be measured in psi.
3. Click OK.
4. Select Menu Bar => Device => UserObject.
5. Move the blank UserObject object to the left side of Main Work Area; click the mouse.
6. Double-click the mouse on the UserObject to achieve an Open View.
7. Double-click on the UserObject Title Bar; the UserObject Properties dialog box will appear.
8. Type Vehicle Radiator in the Title Field; click OK.
9. Click on the Vehicle Radiator Title Bar with the mouse right button.
10. Select Add terminal => Data Output. (The output pin on the right side is labeled "X".)
11. Go to the upper-right corner of the Vehicle Radiator window; click the mouse left button on the middle Vehicle Radiator button to Maximize the window if necessary.
    Note: The following steps occur within the Vehicle Radiator UserObject workspace.
12. Select Menu Bar => Device => Virtual Source => Function Generator; move it to the left center of Vehicle Radiator; click the mouse button to place the object.
13. Change Function to Tri; edit Frequency to 100, Amplitude to 10, and DcOffset to 190. (This will represent a virtual Vehicle Radiator temperature of 190 ± 10°F.) Change the Function Generator title to Temp Generator.
14. Click on the Temp Generator upper-right button; it will reduce the Temp Generator object to an Iconic View. Move the temp-generator object to the left of the Work Area slightly above the center.
15. Place a Noise Generator in the Vehicle Radiator UserObject.
16. Select Menu Bar => Device => Virtual Source => Noise Generator and change its title to Temp Noise; change its Amplitude to 5.
17. Click on the Temp Noise upper-right button; it will reduce the Temp Noise object to an Iconic View; place it below the Temp Generator object.
18. Select Menu Bar =>Device => Function & Object Browser; a pop-up dialog box will appear.
19. Select Operators under Type, <All> under Category, and + under Member.

20. Click on Create Formula; a new A+B object will appear.
21. Place A+B to the right of Temp Generator and Temp Noise and very close to them.
22. Connect the Temp Generator data-output to the A+B data-input A.
23. Connect the Temp Noise data-output to the A+B data-input B.
24. Connect the A+B data-output pin (Result) to the X pin within the Vehicle Radiator UserObject Work Area.
25. Go to the upper-right corner of the Vehicle Radiator bar; click on the middle button. (This will decrease the size of this UserObject.)
26. Examine Figure 3.7. Arrange the Vehicle Radiator UserObject in the Main Work Area as both an Iconic View and an Open-View UserObject. Change the size and location of the Main Work Area and UserObject to display the same as shown in Figure 3.7.

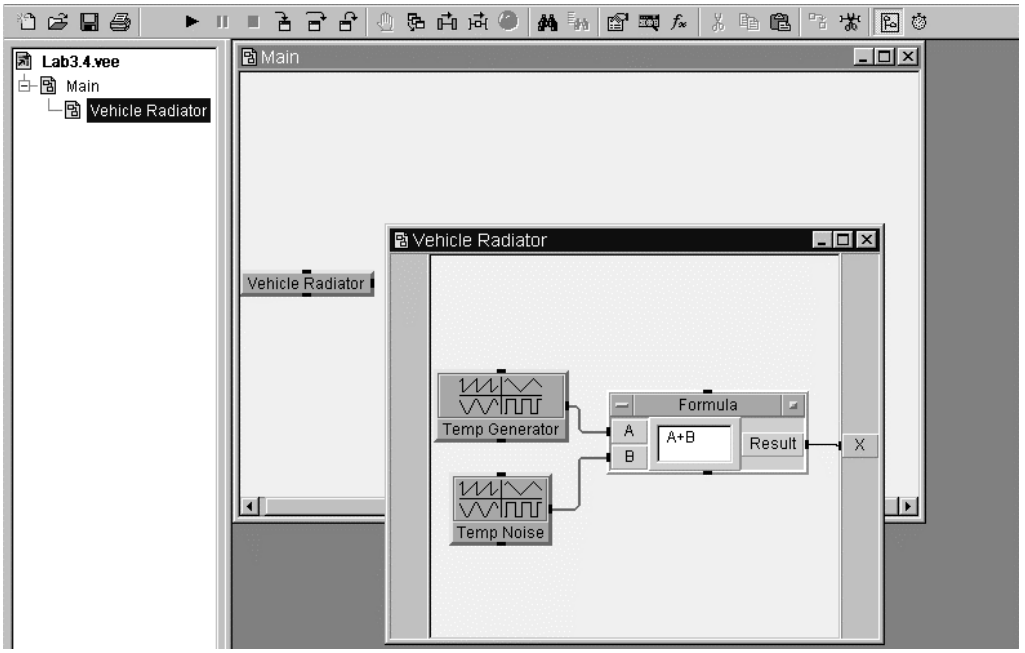


Figure 3.7. Vehicle Radiator UserObject

27. Go to the Vehicle Radiator Title Bar; click on the right-hand "-". (This will reduce the Vehicle Radiator UserObject to an Iconic View in the lower-left corner.)
    Note 1: You have completed working within the Vehicle Radiator UserObject.
    Note 2: The following steps occur in the Main Work Area.
28. Save as LAB3-4 to your computer's hard drive to avoid losing all your above work.
29. Select Menu Bar => Display => Waveform (Time); place it to the right of Vehicle Radiator.
30. Connect the Vehicle Radiator data-output pin (X) to the Waveform (Time) data-input pin. (That pin is just above "Trace1".)

31. Go to the Tool Bar; click the (right-facing arrow) run button; a noisy waveform will appear on the virtual scope. See Figure 3.8.

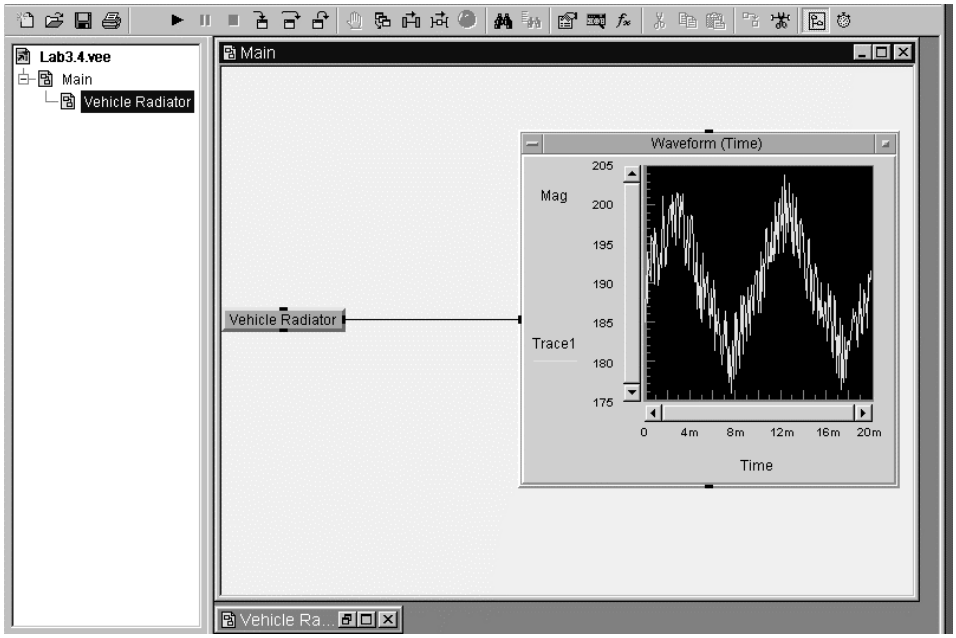


Figure 3.8. Vehicle Radiator with temperature variations

◊ 32. Open the Vehicle Radiator UserObject and its Temp Noise; re-run the program with a variety of noise amplitude levels; save your program again.

33. In the Vehicle Radiator UserObject, clone the Temp Generator; rename it Pressure Generator; change Frequency to 50; Amplitude to 2; DcOffset to 40. (This will generate a virtual radiator pressure of 40 ± 2psi.). Verify it; convert it to an Iconic View and place it below Temp Noise.
34. In the Vehicle Radiator UserObject, clone Temp Noise; rename it Pressure Noise; verify it; convert it to an Iconic View and place it below Pressure Generator.
35. Clone A+B; rename its Title Bar A+B#2; verify it; place it to the right of Pressure Generator and Pressure Noise.
36. Connect the output of Pressure Generator to input A of A+B#2; connect the output of Pressure Noise to input B of A+B#2.
37. Click on the left-most icon in the Vehicle Radiator Title Bar; select Add Terminal => Data Output; a Vehicle Radiator output pin "Y" will appear.
    Note: You may prefer to right-click anywhere on the Object to select Add Terminal.
38. Connect A+B#2 output pin (Result) to Vehicle Radiator pin "Y"; convert Vehicle Radiator to an Iconic View by clicking the left button in the upper-right corner of the Vehicle Radiator Title Bar.
39. Select the Waveform (Time) object menu; select => Add terminal => Data Input; an input labeled Trace2 will appear and have a trace of a different color.
40. Connect the Vehicle Radiator output "Y" to Trace2.

41. Change the name of Trace1 to Temp; change the name of Trace2 to Pressure.
42. Save your program to your disk, drive A: as LAB3-4.
43. Run this modified program; observe the two traces on the scope; compare the two frequencies and amounts of noise. See Figure 3.9.

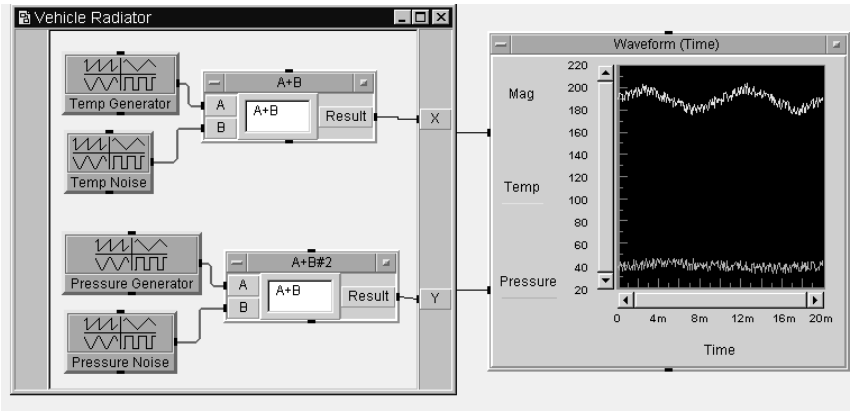


Figure 3.9. Vehicle Radiator with temp and pressure displayed

44. Open Vehicle Radiator; open all four generators so you can easily access them.
    Note: They will cover most of your screen. However, their input and output connections will continue to be the same even if you cannot see them.

◊ 45. Run LAB3-4 several times with different values of temperature amplitude variation and pressure amplitude variation; do NOT save.
    Note: You will have to continually open and close the Vehicle Radiator UserObject so you can view the oscilloscope.

◊ 46. Run LAB3-4 several times with different amounts of noise; do NOT save.

◊ 47. Run LAB3-4 several times with different values of temperature-amplitude variations, pressure-amplitude variations, and different amounts of temperature and pressure noise; do NOT save until you read the following note.
    Note: Upon completion of these several runs, do NOT save this modified program as LAB3-4. However, if you modified any titles, connections, etc that you may wish to use again later, then Save As…LAB3-4a, or b, or c to your drive A: so you will not lose your final program. LAB3-4 will be accessed and modified in several future labs.

Lesson 3 Summary

This lesson described how to control instruments, configure a panel driver for Direct I/O, collect and graph random numbers, and devise a virtual radiator for later applications.

Lab 3.1 showed you how to configure an instrument from the Menu bar to use a Panel Driver, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Lab 3.2 showed you how to configure an instrument from the Menu bar to use Direct I/O, modify that instrument's name and controlling parameters, and save the configuration for later applications.

Lab 3.3 showed you how to create a program that generates 12 random numbers, displays them, and indicates the total time required to generate and to display the values.

Lab 3.4 showed you how to write a file description that will become part of the printed documentation, modify a partially developed virtual vehicle radiator to create varying temperatures and companion pressures, and monitor those two parameters simultaneously on a virtual scope.

Next you will learn how to access and control actual (real) instruments.

Lesson 4
Controlling Real Instruments

This lesson will examine the various ways that real instruments can be investigated in a simulated (VEE) environment. These physical instruments are then connected via VEE to a program that involves their use. This lesson consists of a pre-lab and four labs.

Lesson 4 Pre-lab Summary

The following are described in the Lesson 4 Pre-lab. See Appendix E, page E-33.

- Instrument Control Object
- Instrument I/O Data Types
- Installing Hardware
- Installing IO Libraries and Configuring Interfaces
- Using Device Symbolic Names in SICL
- Additional Visual BASIC Programming Notes
- Warnings

As noted in all previous Lessons, Appendix B includes a cross-reference to each of these items and to all objects and subprograms contained in the labs of all lessons.

Overview

Lab 4.1 – Preparing to Select a Real Instrument
This lab will show you how to set the parameters within VEE Pro to configure an actual instrument driver, such as an oscilloscope.

Lab 4.2 – Communicating with GPIB Instruments
This lab will show you how to investigate a variety of GPIB instruments for communicating with VEE Pro programs.

Lab 4.3 – Monitoring Passive Devices
This lab will show you how to set up and measure temperature using a thermocouple and a digital multimeter for VEE Pro access.

Lab 4.4 – Interacting with Real Equipment
This lab will show you how to send a single text command, or an expression list, to an active instrument using Direct I/O.

Lab 4.1 – Preparing to Select a Real Instrument

This lab will show you how to set the parameters within VEE Pro to configure an actual instrument driver, such as an oscilloscope.

Open your VEE program and

1. Clear your Work Area, maximize it; toggle Program Explorer off.
2. Open Lab 3.1. It should look like Figure 4.1. Save as Lab 4.1 immediately.
   Note: Lab 3-1 showed you how to add new instruments to the Instrument Manager.

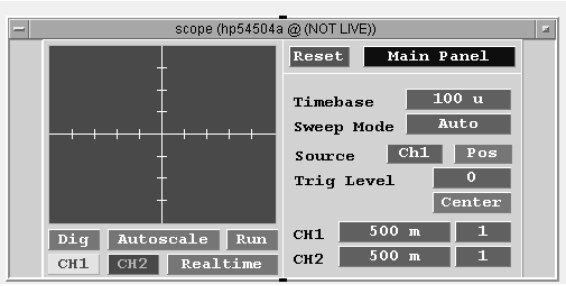


Figure 4.1. Selected hp54504a @ Not Live

Configuring a virtual instrument to a real instrument

3. Select Menu Bar => I/O => Instrument Manager.
4. Choose scope(hp54504a(@(NOT LIVE)).
5. Open Properties; double-click on the Address field; change the address to 709.
   Note: The "9" in the address signifies the default address for oscilloscopes and "7" is the GPIB logical unit number.
6. Click on Advanced under Properties; toggle the Live Mode to ON; click OK twice.
   Note 1: The Instrument List will now include a GPIB7 heading with oscope(@709) appearing under it.
   Note 2: At the top of the Instrument List, the expression: Embedded Configuration (Lab4.1.vee)* will appear.
7. Save this lab again as Lab4-1.
   Note: As this now represents a live (real) instrument, an interface to an actual oscilloscope must be connected to the PC system via a card. See Lab 4.2 below.

Lab 4.2 – Communicating with GPIB Instruments

This lab will show you how to investigate a variety of GPIB instruments for communicating with VEE Pro programs.

Warning: First turn off your computer; then install any appropriate card(s).

1. Turn off your computer and install the appropriate interface card selected either from the list given in Appendix E or from the card compatible with the real instrument that you choose to use. See page E-34.
   Note: Before you install your IO Libraries software, you should install any other new hardware in your computer. After all hardware is installed, turn the computer on and allow it to boot. If you are adding plug&play hardware, Windows may display a Wizard when you log on to guide you through the process of installing drivers for new hardware.
2. Turn on your computer.
3. Install the IO Libraries software as described starting with page E-37.
   Note: After installing any hardware and drivers needed for IO, you should then install the Agilent IO Libraries by running 'Setup' from the Agilent IO Libraries CD. Before an interface can be used with SICL or Agilent VISA, IO Libraries must configure the interface. This is normally done automatically during the installation or it can be done manually using the 'IO Config' utility located in the 'Agilent IO Libraries' program folder. For information regarding National Instruments hardware, consult their Web site.
4. Open your VEE Pro program, clear your Work Area, maximize Main; toggle Program Explorer off.

Using device symbolic names in SICL

This version of SICL supports device symbolic names. You can assign a symbolic name to a device and then use that name in place of the device's address when making an iopen() call. The advantage of doing this is that you do not need hard-code device addresses in your programs.

5. Select Menu Bar => I/O => Instrument Manager.
6. Click on Add …, this displays the Device Configuration Dialog Box. The following fields are presented:
   Name: Enter scope2. (It would be wise for those of you working together to agree on a naming procedure as illustrated in Figure 4.2. This will allow each of you to understand each other's programs. Later you can save time by choosing already existing virtual instruments. We have chosen hp54504a.)
   Interface: Select GPIB.
   Address: The select code of the interface (GPIB is usually 7) plus the local bus address of the instrument (which is a number from 0 to 31). Type 722 for now.
   Gateway: Use This host to control instruments locally.
   Note: Pressing the Tab key after typing in a field will move you to the next field; pressing Shift-Tab will move you to the previous field.
7. Leave all the other defaults as they are given.
8. Click on Advanced … .
9. Leave Timeout (sec): at 5.
10. Toggle Live Mode to ON.
    Note: Live Mode should be ON and light-gray because the address is set to a value other than zero.
11. Leave Byte Ordering: at MSB (as required by all IEEE488.2-compliant devices).

12. Type in Description (optional): hp54504a; this description will appear on the Title Bar; the Instrument Manager will now display hp54504a in the Instrument List under GPIB7. See Figure 4.2.

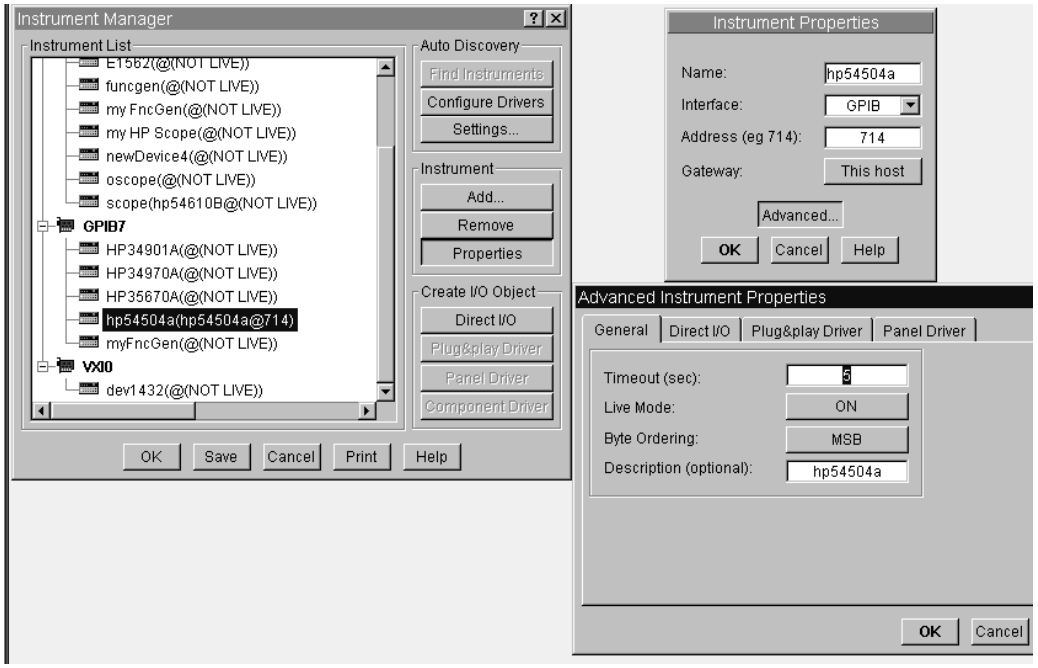


Figure 4.2. The Configured Instrument Manager

13. Click on the Panel Driver tab.
14. Click on the blank space to the right of ID Filename: A list of instruments will appear entitled: Read from what Instrument Driver? Select hp54504a.cid by clicking on Open. See Figure 4.3.

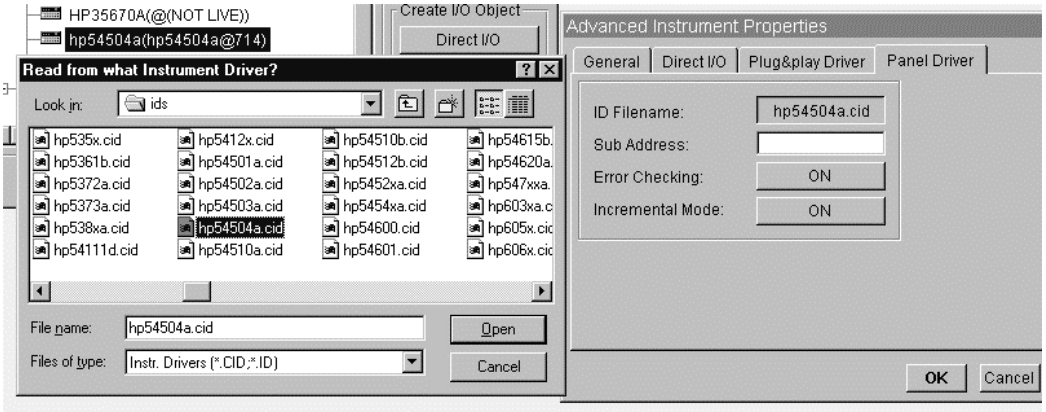


Figure 4.3. Interfacing with an Instrument Driver

15. Click OK on the Advanced Instrument Properties Dialog Box.
16. Click OK on the Instrument Properties Dialog Box.
17. Return to the Instrument Manager Dialog Box; click Save.
    Note 1: An instrument configuration named hp54504a.cid using the driver file is now in your list of available instruments.
    Note 2: You can develop a different program as described above or change an address later when there is another live instrument interfacing with VEE Pro.
    Note 3: When you decide to develop a new live-instrument program, then you must initiate a new VEE Pro program, following this total lab procedure but with the new instrument-driver names, numbers, and bus addresses.
    Note 4: When using Plug&Play drivers, open the Advanced Instrument Properties and the Plug&play Driver folder. Identify the driver name and its parameters so you can add additional drivers from the Agilent Web site. Specific instructions are given in the Plug&play Driver folder.
18. Save this program as LAB4-2.

Lab 4.3 – Monitoring Passive Devices

This lab will show you how to set up and measure temperature using a thermocouple and a digital multimeter for VEE Pro access.

Reference: Practical Temperature Measurements, Agilent Technologies Application Note 270, Publication Number 5965-7822E.

Installing the 34970A data acquisition switch unit

1. Turn off your computer.
2. Install the 34970A unit in the appropriate slot as noted in your card list. For general guidance see Appendix E, pages 32 through 42.
3. Turn on your computer.

Configuring the interface

Open your VEE program and

4. Clear your Work Area, maximize Main; toggle Program Explorer off.
5. Select Menu Bar => I/O => Instrument Manager… .
6. Select "My configuration" if it is not the default folder.
7. Select HP34970A from the "My configuration" list.
   Note 1: We have assumed that the unit for this instrument has been installed on your computer; see step 2 above.
   Note 2: The instrument label may be given as Agilent rather than HP.
8. Obtain a J-type thermocouple and install it into one of the channels on the 20-channel multiplexer (34901A). See Figure 4.4.

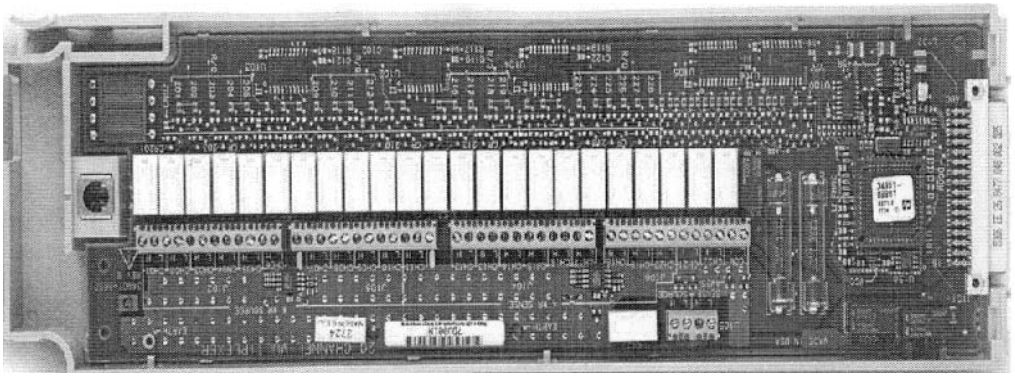


Figure 4.4. The 34901A 20-Channel Multiplexer

9. Insert the Multiplexer into the top slot in the rear of the 34970A Data Acquisition/Switch Unit. See the bottom photograph of Figure 4.5.

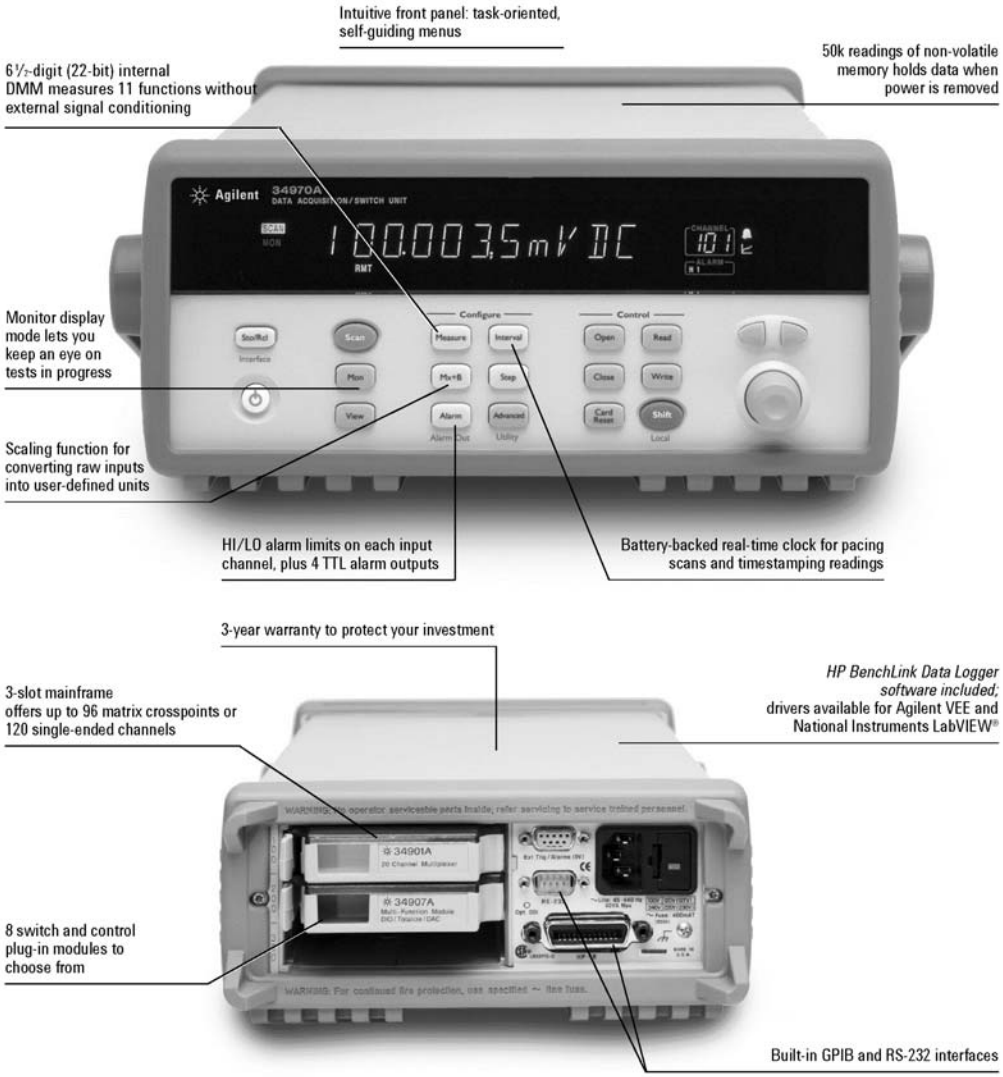


Figure 4.5. The 34970A Data Acquisition/Switch Unit

10. Place a beaker of water on a hot plate; insert the J-type thermocouple.
11. Set the controls on the Data Acquisition/Switch Unit to cause its panel to display temperature readings. (See the 34970A manual for guidance.)
12. Turn on the hot plate; note the temperature changes via the panel readout, set as advised in Application Note 290.
13. Turn off the hot plate.

Recording temperature readings on a strip chart

14. Select Menu Bar => Display => Strip Chart; place it in the upper-right corner of your work area; change the X Name to Time and the Y Name to Temperature by clicking on their existing labels.
15. Select Menu Bar => I/O => Instrument Manager… ; select HP34970A if available; it not, then following the instructions for Lab 4.2.
16. Click on Create I/O Object – Direct I/O; the I/O Transaction box will appear.
17. Place the I/O Transaction box to the left of the Strip Chart.
18. Add transactions to the Direct I/O object as shown in Figure 4.6.
19. Connect the output of the 34970A Transaction box to the Strip Chart; set the Strip Chart Time scale to 30 by clicking on the Time label; set the Temperature scale to 200 by clicking on the Temperature label. See Figure 4.6.
20. Turn on the hot plate; note the temperature data changes via the Strip Chart.
21. Run your program; save it as LAB4-3 to your preferred drive.

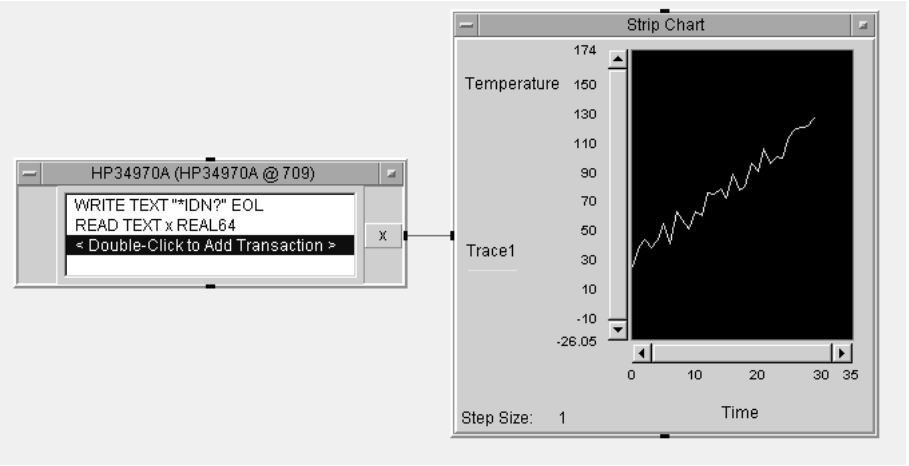


Figure 4.6. Simulation of Instrument Driver Added Manually

Note: For further information, see Agilent 34970A Data Acquisition/Switching Product Overview publication 5966-4443EN obtained via the Agilent Web site: www.agilent.com/find/assist.

Lab 4.4 – Interacting with Real Equipment

This lab will show you how to send a single text command, or an expression list, to an active instrument using Direct I/O.

Open your VEE program and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Sending a single text command to an active instrument

2. Select Menu Bar => I/O => Instrument Manager…

3. Select funcgen(@(NOT LIVE)); click on Direct I/O to create a Direct I/O object; place the object in Main.
4. Double-click on its transaction bar to obtain its Dialog Box.
5. Select WRITE, TEXT, and type "AM 5 VO" with EOL ON; click OK. See Figure 4.7.
   Note 1: The information within the quotes is the command that will be sent to the Function Generator when the program runs. The quote marks are necessary.
   Note 2: Some instruments specify characters that must be sent at the end of a command. These characters are given in the instrument documentation. They must be included in the Advanced Properties section of the I/O Dialog Box.

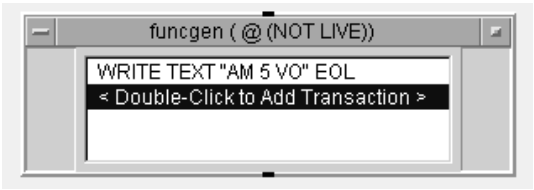


Figure 4.7. A Single-Text-Command I/O Transaction

Sending an expression list to an active instrument

6. Select Menu Bar => I/O => Instrument Manager…
7. Select funcgen(@(NOT LIVE)); click on "Direct I/O"; place it to the right of the Work Area.
8. Double-click on its transaction bar to obtain its Dialog Box.
9. Select WRITE TEXT, and type "FR",A,"HZ" with EOL ON; click OK.
   Note 1: FR represents frequency; **A** represents the frequency value at input terminal **A**, and **HZ** represents the frequency unit: "Hertz".
   Note 2: The terminal **A** was added automatically. This transaction command will write the string "FR", followed by whatever value is sent into input terminal **A**, followed by the string "HZ".
10. Select Menu Bar => Flow => Repeat => For Range; place it to the left of the Function Generator. (This will simulate the frequency value.)
11. Connect the For Range data output pin to the funcgen(@ (NOT LIVE)) data input pin.
12. Edit the fields in the For Range object as follows. See Figure 4.8.
    From: 10
    Thru: 2.1M
    Step: 50k

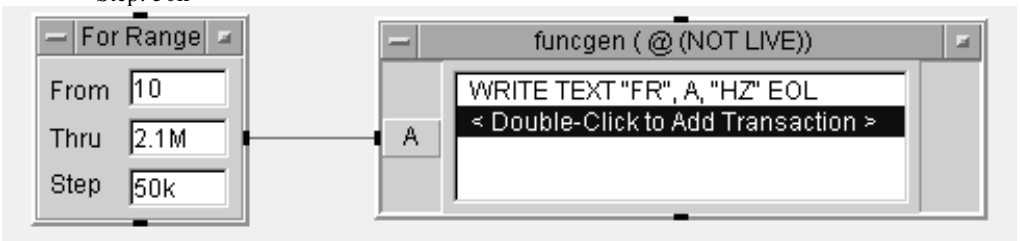


Figure 4.8. An Expression List I/O Command Transaction

Note 1: The Function Generator will start at 10 Hz, increase in steps of 50 kHz, and stop at 2.1MHz.

Note 2: This subprogram is designed to work with the Agilent3325B Function Generator. Set the Instrument Manager to call the live instrument. When the Agilent3325B is connected, change the transaction box title to "LIVE".

13. Save this program as LAB4.4b.
14. Run this program only if the Agilent3325B is attached.

Lesson 4 Summary

This lesson described how to control instruments, configure a panel driver and Direct I/O, and talk to actual instruments that you have connected.

Lab 4.1 showed you how to set the parameters within VEE Pro to configure an actual instrument driver, such as an oscilloscope.

Lab 4.2 showed you how to investigate a variety of GPIB instruments for communicating with VEE Pro programs.

Lab 4.3 showed you how to set up and measure temperature using a thermocouple and a digital multimeter for VEE Pro access.

Lab 4.4 showed you how to send a single text command, or an expression list, to an active instrument using Direct I/O.

Next you will learn how to analyze and display test data.

Lesson 5
Analyzing and Displaying Test Data

This lesson will examine the fundamentals required to analyze and display test data, customize displays, and have VEE Pro automatically calculate statistical parameters. It consists of a pre-lab and four labs.

Lesson 5 Pre-lab Summary

The following are described in the Lesson 5 Pre-lab. See Appendix E, Page E-41.

- Statistical Formulas
- Data Types
- Analysis Capabilities
- Multiline Formula Techniques
- Operators
- Built-in Functions
- Display Capabilities

As noted in all previous Lessons, Appendix B includes a cross-reference to each of these items and to all objects and subprograms contained in the labs of this and later lessons.

Overview

Lab 5.1 – Using the Formula Object
This lab will show you how to write mathematical expressions in VEE Pro by using a variety of built-in VEE Pro functions via the Formula Object.

Lab 5.2 – Modifying the Formula Object
This lab will show you how to write your own mathematical expressions in VEE Pro via the Formula Object.

Lab 5.3 – Using Multiline Formulas and Improved Displays
This lab will show you how to use a Formula Object to combine two waveforms, provide a variety of outputs, and display the results on a Waveform (Time) display.

Lab 5.4 – Customizing Displays
This lab will show you how to label, move, and size displays; change display x/y scales, modify the traces, add markers, zoom in on parts of the graphical display, interpolate between two data points, and change trace colors.

Note: Assign individual student projects as noted in The Instructor (page xvi) or assign analysis problems that apply MATLAB®.

Lab 5.1 – Using the Formula Object

This lab will show you how to write mathematical expressions in VEE Pro by using a variety of built-in VEE Pro functions via the Formula Object. The Formula Object was first introduced in Lesson 1. The variables in the expression can be used as the data-input pin names. The result of the evaluation of each expression will appear on the data-output pin. Functions created via the Function and Object Browser are formats that have their expression preset. They can be modified to combine functions and add (or delete) inputs.

Open your VEE Pro program, clear your Work Area, maximize Main; toggle Program Explorer off.

Evaluating a simple expression with the Formula Object
Example: 2*A^6-B, where A=2 and B=1.

Note 1: * is for multiplication, ^ is for exponentiation.
Note 2: The variable names are not case sensitive.

1. Select Menu Bar => Device => Formula; the Formula Object should appear. Place it in the center of Main.
   Note: For data-flow and Order of Operations information, see pages A-7 and A-8.
2. Double-click the Formula input-field window. (The Default formula is 2*A + 3.)
3. Change the default formula to read: 2*A^6-B.
4. Place the mouse pointer over the letter A on the input pin.
5. Press Ctrl-a, which will add an input pin that is labeled B.
   Note: It will be labeled B by default; you may rename it at any time.
6. Select Menu Bar => Data => Constant => Int32; place it to the left and slightly above the center of the Formula Object.
7. Go to the Int32 object menu.
8. Clone the Int32 object by selecting Clone; place the clone below the first Int32 object.
9. Connect the two Int32 objects to the Formula inputs A and B; change their titles to Integer A and Integer B respectively.
10. Enter 2 in the A Integer input box
11. Enter 1 in the B Integer input box.
12. Select Menu Bar => Display => AlphaNumeric.
13. Connect the AlphaNumeric data-input pin to the Formula (Result) data-output pin.
14. Run your program. See Figure 5.1 for the result (in the AlphaNumeric display).

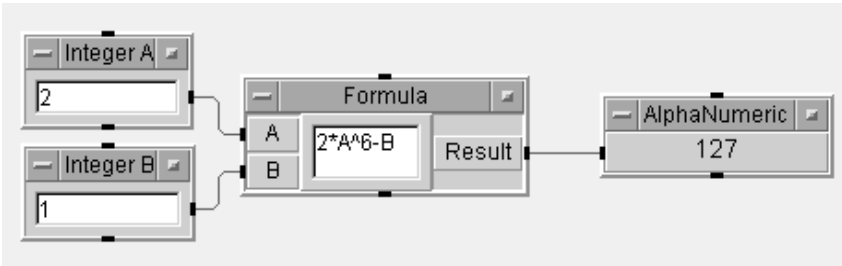


Figure 5.1. Evaluating an expression and viewing the results

15. Save your program as LAB5-1.
◊ 16. Replace the integer values with values of your choice. Run this modified program; verify that the formula processes your choices correctly.
◊ 17. Attempt to insert a non-integer number into one of the Integer objects. Note the effect on the input values when you run the program. Use values both less than and greater than 5.
◊ 18. Change the formula. Run this modified program; verify that the formula processes your choices correctly.

Your notes:

_____

_____

_____

_____

_____

19. Close LAB5-1 without saving your changes.

Lab 5.2 – Modifying the Formula Object

This lab will show you how to write your own mathematical expressions in VEE Pro via the Formula Object.

Open your VEE Pro program and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Typing a function into the Formula Object

Example: Generate a cosine wave; then calculate its standard deviation and root mean square using the Formula Object.

2. Create the Function Generator, Formula, and AlphaNumeric objects as follows, arranging the objects as in Figure 5.2 for the remainder of this lab:

- Select Menu Bar => Device => Virtual Source => Function Generator; convert Function Generator to an icon.
- Select Menu Bar => Device => Formula; clone the Formula Object.
- Select Menu Bar => Display => AlphaNumeric; clone the AlphaNumeric Object.

3. Arrange and connect the objects as shown in Figure 5.2.

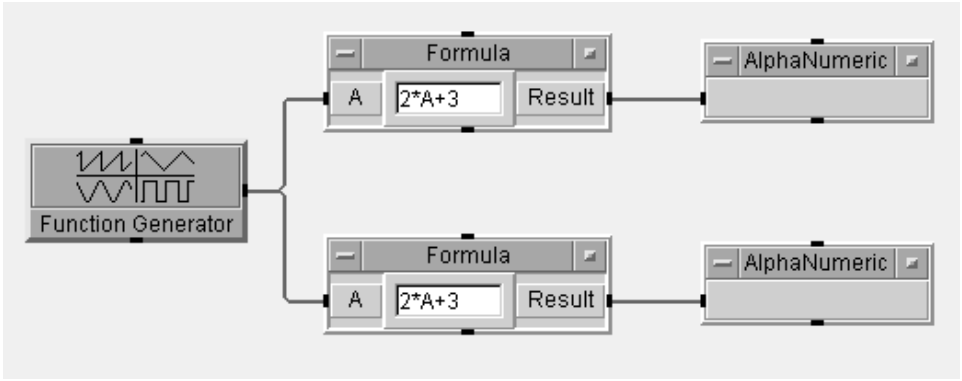


Figure 5.2. Wiring for Lab 5.2

4. Change the entries in the Formula objects as follows:

   - Double-click the input field of the upper Formula Object and enter sdev(A).
   - Double-click the input field of the lower Formula Object and enter rms(A).

5. Save your program as LAB5-2.
6. Run this program; the results should match Figure 5.3.

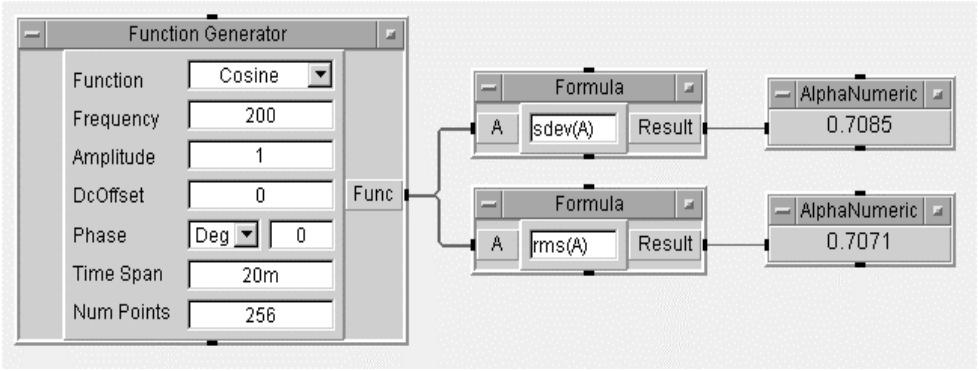


Figure 5.3. Lab 5.2 after running

Calculating two parameters using one Formula Object

7. Use Save As… to change LAB5-2 to become LAB5-2a.

8. Modify this newly saved program to calculate two formulas within one Formula Object. See Figure 5.4. Save this modified program. Run this program.
   Note: The use of one Formula Object to perform two calculations causing the program to run faster than the program shown in Figure 5.3.

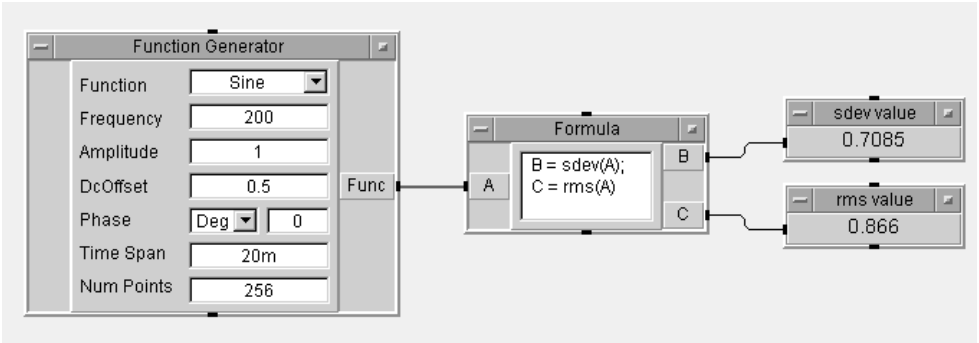


Figure 5.4. Using one Formula Object to compute two parameters

Calculating a ramp function using the Formula Object

9. Select Menu Bar => File => New; maximize the screen; remove the Program Explorer.
10. Select Menu Bar => Data => Constant => Int32; place it in the upper-left corner of your screen.
11. Clone Int32; place it below the first integer object.
12. Change the top integer value to 2048 and the bottom integer value to 1.
    Note: Recall that you should double-click on the input field.
13. Select Menu Bar => Device =>Formula; change the Title Bar to ramp(numElem,start,stop); change the formula to ramp(numElem,start,stop); add two more data-input pins using the Formula Object menu. Save this program as LAB5-2b.
    Note 1: Ramp generates a linearly ramped array. Use ramp(numElem,start,stop) to generate a one-dimensional array of length numElem, with the values linearly ramped, starting with "start" and ending with "stop". NumElem must be a scalar container with a value greater than zero; "start" and "stop" must be scalar containers.
    Note 2: The names of the ramp input terminals should match the formula expression variables. These more meaningful names are highly recommended.
14. Connect the Integer and ramp… objects as shown in Figure 5.5. Run this program.

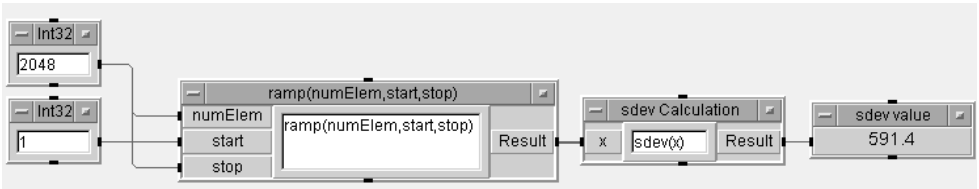


Figure 5.5. Calculating the standard deviation of a Ramp function

◊ 15. Insert a variety of numbers into the Int32 objects; run this program for each set of integer values; do not save this modified program.
Note: The standard deviation values will change. Why?

_____

_____

_____

_____

_____

Calculating standard deviations for a ramp function

16. Return to the original LAB5-2b; select Menu Bar =>Device => Formula and place this object below the Int32 objects.
17. Change the Formula title bar to read: Self-contained Ramp Formula.
18. Delete the Formula Object input pin.
19. Insert the following into the Formula Object editing space: ramp(2048,1,2048).
20. Clone sdev calculation (Formula) and sdev value (AlphaNumeric) objects.
21. Connect the sdev calculation object output pin to the sdev value input pin; connect the Self-contained Ramp Formula output pin to the sdev calculation input pin.
22. Run this program; save again as LAB5-2b. See Figure 5.6.

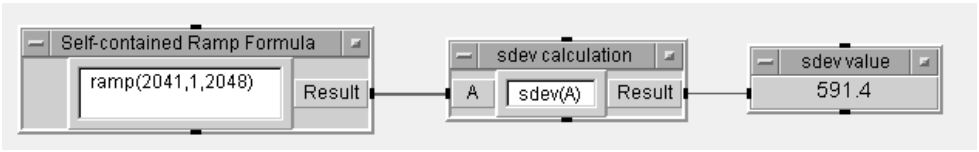


Figure 5.6. Self-contained Formula program after running

Note: The standard deviation is computed from the data at the sdev(A) input pin. The ramp and data inputs are nested into a single Formula Object. This approach eliminates two integer-objects.

23. Clone the Self-contained Ramp Formula and sdev value objects. Place them below their previous objects.
24. Change the cloned Formula Object expression to read: sdev(ramp(2048,1,2048)).
25. Change its title bar to read: Self-contained Ramp/sdev Formula.
26. Connect the formula and alphanumeric objects together.
27. Save this program again as LAB5-2b; run this program. See Figure 5.7 for a comparison of three techniques for calculating a parameter such as standard deviation. Our choice for data input was the Ramp function.
Note: In general, the fewer the VEE Pro objects, the faster the program will run.

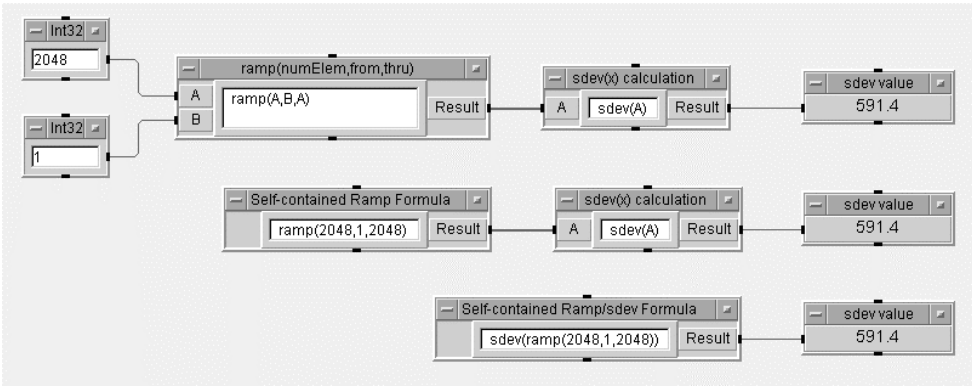


Figure 5.7. Three techniques for calculating standard deviation

◊ 28. Change ramp… values to ramp(1024,1,1024); re-run this program. What are the effects upon the sdev values?

Your notes:

_____

_____

_____

_____

_____

29. Do not save this modified program.

Lab 5.3 – Using Multiline Formulas and Improved Displays

This lab will show you how to use a Formula Object, with multiple lines of instruction, to combine two waveforms and display both their individual and combined outputs.

Open your VEE Pro program and

1. Clear your Work Area and maximize Main.

Preparing a multiline Formula Object

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of your screen; change its function to Sine.
3. Select Menu Bar => Device => Virtual Source => Noise Generator; place it in the lower-left corner of your screen; set its amplitude to 0.25.
4. Select Menu Bar => Device => Formula Object; place it to the right of and between the two generators; change its input-pin name to Sine.

5. Add a second input pin to the Formula Object via the mouse left button; change its name to Noise; expand the white (editing) area to allow viewing of the instructions given in Step 6.
6. Enter the following statements into the Formula Object white space; size the white space:

```
//Copy the Inputs to two Outputs
Orig_sine=Sine;
Orig_noise=Noise;

//Combine the inputs and place the
//results in a single output.
Combined=Sine+Noise;

//Multiply the combined output by 2.
//This demonstrates the use of
//temporary variables.
Combined=Combined*2
```

   Note: The double slashes // are used to separate explanatory statements from the program.
7. Add three more output pins to the Formula Object; change their three names to:

```
Orig_Sine
Orig_Noise
Combined
```

   Delete the original "Result" pin because its name cannot be changed.

Creating an XY trace with three inputs

8. Select Menu Bar => Display => Waveform (Time); place to the right of the Formula Object; change its name to Y vs X Trace.
9. Add two more inputs to Y vs X Trace via the mouse left button.
10. Modify the scale and trace labels to:

```
Amplitude
Noise
Sine
Combined*2
```

11. Edit the Properties for the Y vs X Trace Object as shown in the following steps:
12. Remove the Scroll Bars from the Y vs X Trace Object in the Properties dialog box.
13. Change the Amplitude Label Spacing to Every Major Tick on the Y axis.
14. Change the Time Label Spacing to Only Max & Min on the X axis.
15. Remove the checkmark from Global Format via the Properties box, Number selection; change the Time (X) axis number format to a scientific format with two fractional digits. Edit Scales to read 0 Minimum and 10 milliseconds Maximum.
    Note: This allows two cycles of the waveform to appear on the screen.

16. Change the number format used on the Amplitude (Y) axis to a fixed format with three fractional digits via the Properties box, Number selection.
17. Connect the Function Generator and the Noise Generator output pins to the Formula Object input pin. See Figure 5.8.

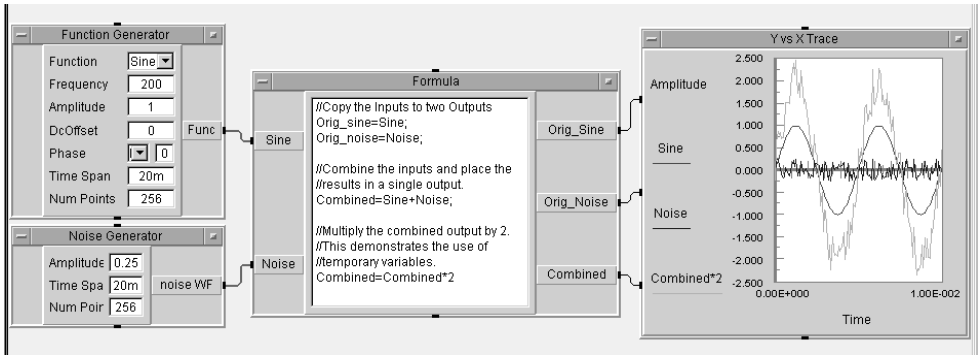


Figure 5.8. Multiline Formula display of Combined Waveforms

18. Connect the Formula Object output pins to the XY Trace input pins as shown in Figure 5.8 above.
19. Save your program as LAB5-3.
20. Run your program. Observe the three waveforms and their amplitude and time values.

◊ 21. Vary the Function Generator amplitude and Noise Generator amplitude; observe the effect on the waveform and scale readings. Record your interpretation of these variations.

Your notes:

_____

_____

_____

_____

_____

22. Close this modified program without saving it.

Lab 5.4 – Customizing Displays

This lab will show you how to label, move, and size displays; change display X/Yscales, modify the traces, add markers, and zoom in on parts of the graphical display. You will use the Noise Generator to generate a waveform and display it with the Waveform (Time) display.

Open your VEE Pro program and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Displaying a waveform

2. Select Menu Bar => Device => Virtual Source => Noise Generator; place it in the left-center of your Work Area.
3. Select Display => Waveform (Time); place it on the right-center of your Work Area.
4. Connect the data-output pin of the Noise Generator to the data input (Trace1) of Waveform (Time).
5. Run this program; it should look like Figure 5.9. Save this program as LAB5-4.

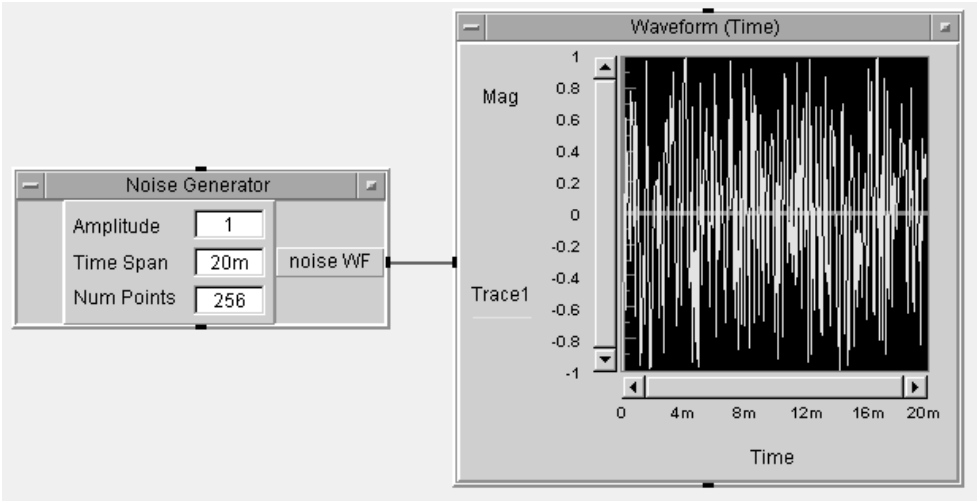


Figure 5.9. Displaying Noise Waveforms using VEE Pro functions

Changing the X and Y scales on a display scope

6. Double-click the Waveform (Time) Title Bar to get the Properties box.
   Note: The Y Properties box contains both X and Y properties.
7. Select the Scales tab.
8. Select the X axis Maximum function in the Time column; enter 1m.
   Note: This alters the time span of the X-axis display from 20 millisec to 1 millisec.
9. Double-click the Minimum input field on the Y axis in the Mag column; enter "-0.5".
10. Click OK.
11. Save your program as LAB5-4.
12. Run this program.

Zooming in on part of a waveform

13. Open the Waveform (Time) object menu via the upper-left corner; click Zoom => In.
    Note: 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.
14. Outline an area of the waveform that includes several peaks; release the mouse button.
    Note: The display zooms in to this selected area of the waveform. Notice that the X and Y scales change automatically.

15. Remove the Zoom in by returning to the object menu; select and click on Auto Scale.

Adding delta markers to a display

◊ 16. Move to the open view on the Noise Generator; change the Num Points setting to 16.
17. Run the program again.
18. Open the Waveform (Time) object menu; select Properties
or
double-click on the Waveform (Time) Title Bar.
19. Click Markers: Delta; click OK.
Note: You will see two gray arrows pointing up and down at one of the data points on the waveform. The display records the X and Y coordinates of these markers at the bottom of the display. Change their color so they stand out better via Properties; General – Marker; Pen 2 (red) seems to provide the best contrast as shown in Figure 5.10.
20. Measure the X or Y distance between two peaks by clicking-and-dragging the arrows to those peaks. You will see the markers move to those new peaks. See Figure 5.10. The new coordinates will be shown at the bottom of the display.

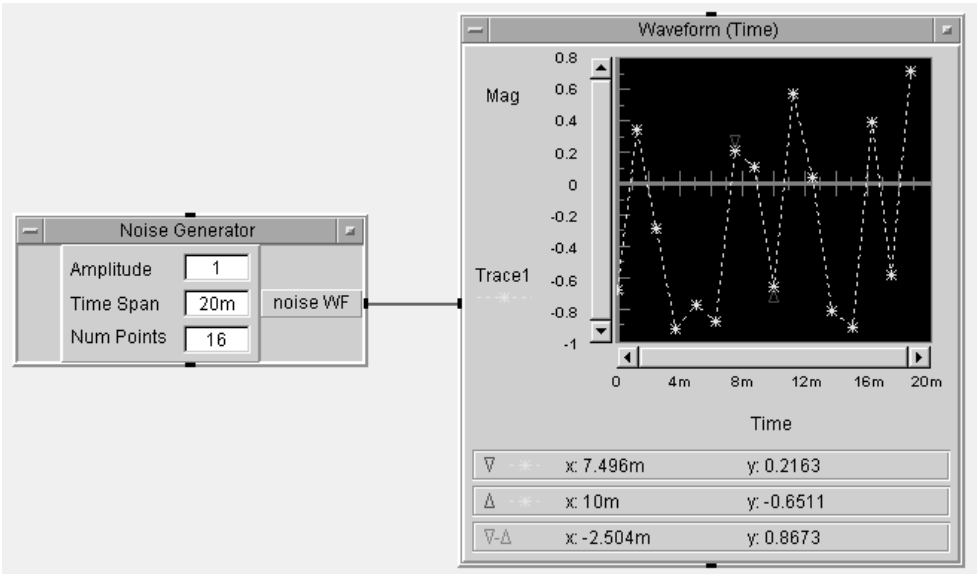


Figure 5.10. Delta markers on a Noise Waveform display

Interpolating between waveform data points

21. Open Waveform (Time) Object menu; select Properties; Y Plot Properties will appear.
22. Click Markers: Interpolate; click OK.
Note: VEE Pro automatically interpolates between any two curve points.
23. Save this modified program as LAB5-4a.

Changing the color of a trace

Note: Other display characteristics such as Panel Layout, Grid Type, Clear Control, and Add Right Scale may be customized in a similar manner.

24. Double-click the Waveform (Time) Title Bar; Y Plot Properties will appear.
25. Click the Traces folder tab.

◊ 26. Select the color, line type, and point type for the Trace selected in this folder.

27. Click OK continually until you exit the Properties box.
28. Save your modified program.

Note 1: "Plot" in the display object menu allows you to plot test results on the display without printing the remainder of the program.

Note 2: See pages C-15 through C-19 for alternative display and information-presentation techniques.

Lesson 5 Summary

This lesson examined the fundamentals required to analyze and display test data using a variety of virtual instruments that will perform two of the automated statistics calculations. Analytical and display capabilities included using VEE Pro: data types, analysis capabilities, Math objects, the Formula Object, working with virtual displays and the customizing of these displays. Details regarding other display options and capabilities are given in Appendix C.

Lab 5.1 showed you how to write mathematical expressions in VEE Pro by using a variety of built-in VEE Pro functions via the Formula Object. You also learned how to evaluate a simple expression with the Formula Object.

Lab 5.2 showed you how to write your own mathematical expressions in VEE Pro via the Formula Object. You also learned how to

- type a function into the Formula Object
- write a more complicated statistics-oriented program

Lab 5.3 showed you how to use a Formula Object with multiple lines of instruction, combine two waveforms, and display both their individual and combined outputs.

Lab 5.4 showed you how to label, move, and size displays; change display X/Y scales, modify the traces, add markers, zoom in on parts of the graphical display, interpolate between two data points, and change trace colors.

You are now ready to learn how to store data using arrays and To/From files and how to access and apply additional statistical parameters.

Lesson 6

Manipulating Arrays, To/From Files, and Statistical Parameters

This lesson will examine how to store data using arrays and To/From files. It consists of a pre-lab and four labs.

Lesson 6 Pre-lab Summary

The following are described in the Lesson 6 Pre-Lab. See Appendix E, page E-45.

- Menu Bar => Data => Collector
- Menu Bar => Data => Concatenator
- To/From Files
- Introduction to Understanding I/O Transactions
- Introduction to Arrays

As noted in all previous lessons, Appendix B includes a cross-reference to each of these items and to all objects and subprograms contained in the labs of this and later lessons.

Overview

Lab 6.1 – Creating Arrays and Manipulating Array Data
This lab will show you how to use a Collector object and math expressions to create and display an array of data and how to use a Concatenator object to combine scalars and array elements into a single array.

Lab 6.2 – Storing and Accessing Data Using To/From File Objects
This lab will show you how to move test data to and from files. You will learn to store and retrieve three common test result items: a test name, a time stamp, and a one-dimension array of real values. (The same general process will apply to all VEE Pro data types.)

Lab 6.3 – Calculating Statistical Parameters
This lab will show you how to set up a function generator, graph its waveform time-domain output, and calculate the waveform mean, median, mode, variance, standard deviation, and rms values.

Lab 6.4 – Logging Vehicle Radiator Statistical Data
This lab will show you how to simultaneously monitor Vehicle Radiator temperature and pressure data, displaying the results of calculating standard deviation and root-mean-square statistical analyses of these data points.

Lab 6.1 – Creating Arrays and Manipulating Array Data

This lab will show you how to use a Collector object and math expressions to create and display an array of data, and how to use a Concatenator object to combine scalars and array elements into a single array.

Open your VEE Pro program and
1. Clear your Work Area and maximize Main.

Using the collector
2. Select Menu Bar => Flow => Repeat => For Count; place it at the left-center of your workspace; highlight the default count of 10 and replace it with: **4**.
3. Select Menu Bar => Data => Collector; place it to the right of the For Count object.
   Note: See Appendix E for the Collector description.
4. Select Menu Bar => Display => AlphaNumeric; place it to the right of the Collector object.
5. Double-click the Collector object to get its open view.
6. Read through Help in the For Count and Collector object menus to understand these objects.
   Note 1: The for-count output value is a Real64 scalar. (Real64 is a constant 8-byte real number.) It starts at 0 and counts to n-1, where **n** is the specified count value. The number of counts depends upon the number of iterations specified in the input field.
   Example: If For Count is to generate outputs of 0, 1, 2, and 3, then change the default number to 4, as in step 2 above.
   Note 2: The Collector is an object that collects data in a loop and provides an output array shape. The one-dimensional array is created from input data (generally scalar). The n+1-dimensional array is collected from input arrays. If the input arrays have "n" dimensions, the collector can either prepare an array of n+1 dimensions, or it can prepare a one-dimensional array. When all data have been collected, activating the XEQ pin will cause the resulting output array to propagate via a "ping".
7. Click n+1 Dim in the Collector to toggle the selection to 1 Dim Array.
8. Connect For Count data-output pin to the data-input pin on the Collector.
9. Connect For Count sequence output (bottom) pin to XEQ input pin on the Collector.
   Note: The XEQ pin is a control pin that exists on several different objects. It tells VEE Pro when you want that object to execute. For the above example, you want the object to fire (execute) after all of the data points from the array have been collected. In this example, XEQ is pinged (executed) when the For Count object output reaches **3**. Until then, the Collector's output will not propagate.
10. Connect the Collector data-output pin to the AlphaNumeric data-input pin.
11. Enlarge AlphaNumeric to accommodate the four-element array.
12. Run your program. See Figure 6.1.

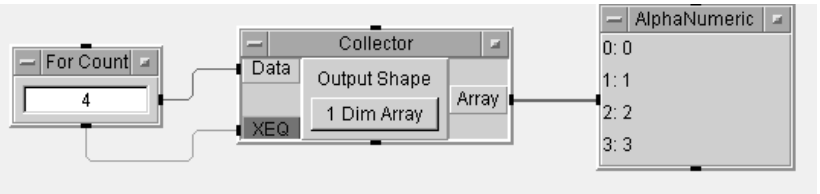


Figure 6.1. The Collector creating a displayed array

Note: The principles are the same regardless of the array size. The Collector will take any data type and create the array size, depending upon the number of elements sent.

Extracting values from an array

13. Delete the data line between the Collector and AlphaNumeric as follows:
    a. Place the mouse pointer over the line; click the mouse right button; select Delete Line; cut the line when the scissors appear. **OR:**
    b. Press Shift-Ctrl and click the mouse left button. **OR:**
    c. Use the right-hand scissors icon on the toolbar; mouse pointer changes to scissors; left-click on the line to be deleted.
14. Convert the Collector to an icon.
15. Move AlphaNumeric to the right so there is room for another UserObject.
16. Select Menu Bar => Device => Formula; clone it; place both Formula objects to the right of Collector.
17. Connect the Collector data-output pin to the data-input pins of the two Formula objects.
18. Enter A[2] in the upper Formula input field.
19. Enter A[3] in the lower Formula input field.
    Note: A[2] will extract the third element of the array as a Scalar because the indexes start at zero.
20. Clone AlphaNumeric.
21. Connect an AlphaNumeric display to each Formula object.
22. Run your program; the upper display should show a value of 2, while the lower display should show a value of 3.
23. Enter A[4] in the upper Formula input field.
24. Run this modified program. What happens? (Close the error pop-up box.)
25. Select Menu Bar => Debug => Stop.
    Note: This will release the program from the error mode.
26. Return the upper Formula input field to A[2].
27. Enter A[1:3] in the lower Formula input field.
28. Run this modified program; it should look like Figure 6.2.
29. Save this modified program as LAB6-1.

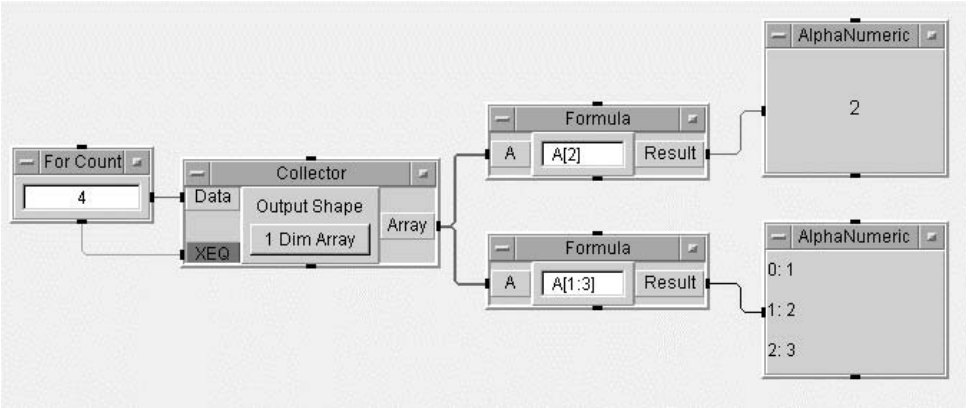


Figure 6.2. Extracting array elements with expressions

Using the Concatenator

30. Select Menu Bar => File => New.
31. Save As … LAB6-1a immediately.
32. Select Menu Bar => Data => Concatenator; place in the right-center of the screen.
33. Open the Concatenator icon; select Add Terminal; Data Input.
34. Select Menu Bar => Display => Alphanumeric; place it to the right of the Concatenator; expand it to allow for nine data lines; connect it to the Concatenator output pin.
35. Select Menu Bar => Data => Allocate Array => Real64; place it to the left of the Concatenator; connect its Array output to the Concatenator input-pin B.
36. Change Allocate Array Num Dims to 1; select Lin Ramp ─ range from 90 to 96; change its size to 7.
37. Select Menu Bar => Data => Constant => Text; place its object above Alloc Real64; connect its output pin to the Concatenator input-pin A; change its data value to Test#n.
38. Select Menu Bar => Constant => Real64; place its object below Alloc Real64; connect its output pin to Concatenator input-pin C; enter the number 82.06 into its data box.
39. Save this program again.
40. Run this program; it should look like Figure 6.3.

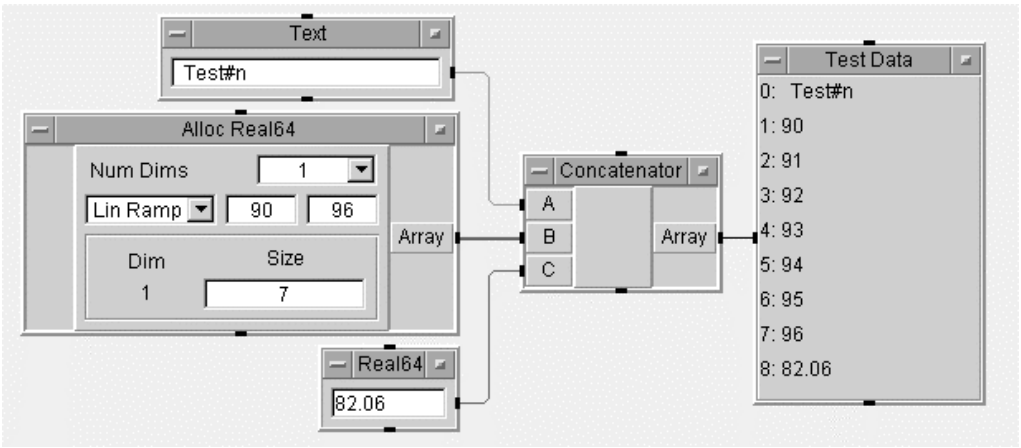


Figure 6.3. Creating arrays with the Concatenator (Lab 6-1a)

◊ 41. Experiment with several values in Alloc Real64.
Note: The Concatenator retains the data and displays it in the sequence in which it is received by its pin sequence: A, then B, then C.
42. Close this program; do not save this program again.

Lab 6.2 – Storing and Accessing Data Using To/From File Objects

This lab will show you how to move test data to and from files. You will learn to store and retrieve three common test result items: a test name, a time stamp, and a one-dimension array of real values. (The same general process will apply to all VEE Pro data types.) This lab should be performed without interruption.

Open your VEE Pro program and

1. Clear your Work Area and maximize Main.

Sending a text string to a file

2. Select Menu Bar => I/O => To => File.
Note 1: The To/File default file name is myFile; it appears in the ToFile object. This default can easily be changed by clicking on myFile. (The button to the right of the ToFile: input field label); a box with a list of files will be displayed from your home directory. See Figure 6.4.

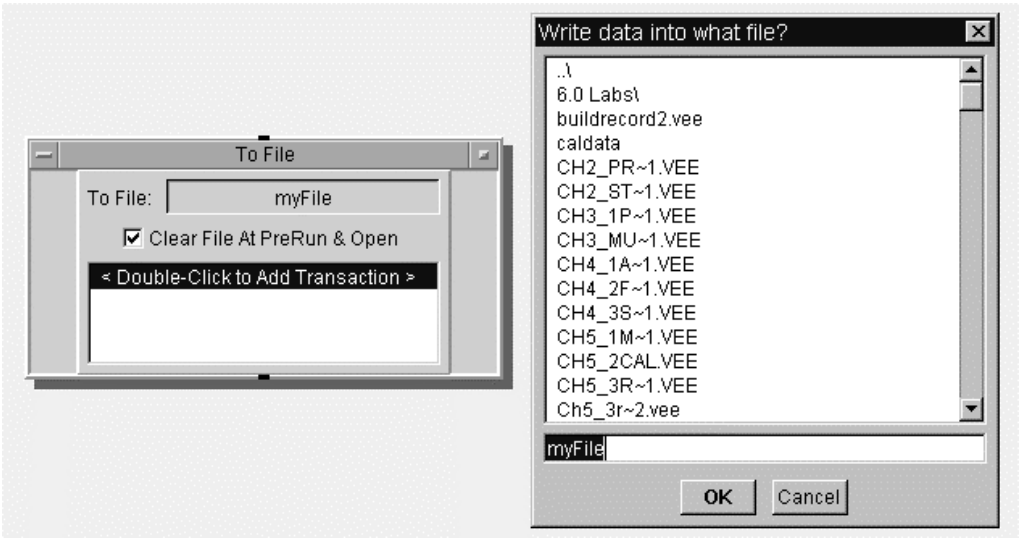


Figure 6.4 The ToFile object and its Home Directory box

Note 2: Click the box next to Clear File At PreRun & Open; by default, VEE Pro appends new data to the end of an existing file. Check the box; the file is now cleared before you write new data.
Note 3: Finally, WRITE TEXT a EOL is the default transaction. It means that you will write the data of variable a on pin A using TEXT encoding and a specified end-of-line sequence. VEE Pro is not case-sensitive with regard to pin names, so you may use lower-case or upper-case strings to for pin names. See Figure 6.5.

3. Double-click the transaction line shown in Figure 6.4 ("Double-Click here …")to open the I/O Transaction dialog box of Figure 6.5.
   Note 1: This is the default transaction. It means that you will write the data of variable **a** on pin **a** using TEXT encoding and a specified end-of-line sequence.
   Note 2: VEE Pro is not case-sensitive when labeling pin names; you may use lower-case or upper-case strings for pin names.

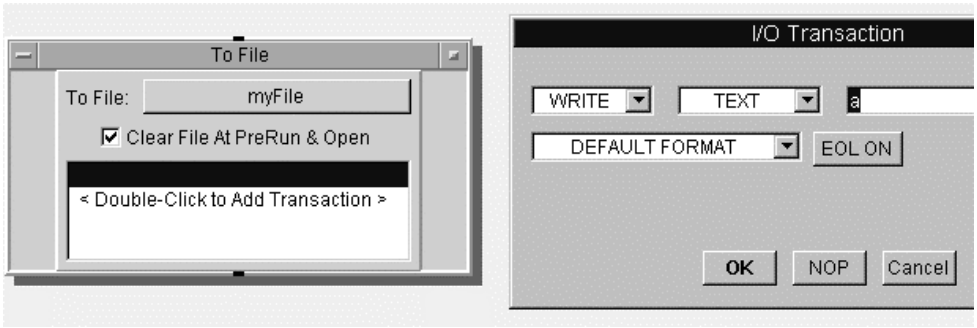


Figure 6.5. An I/O Transaction dialog box

4. In the I/O Transaction dialog box, the expression list field (the right-most field in the top row) is highlighted; type "Test1" in that field, including the quotation marks.
   Note 1: You need the quotation marks to indicate a Text string. If you typed Test1 without the quotation marks, VEE Pro would interpret the entry as a pin name or a global variable. You can leave the other defaults, since you want the action WRITE.
   Note 2: The data will be sent in the following manner:
   - The encoding TEXT will send the data using ASCII characters.
   - The DEFAULT FORMAT will choose an appropriate VEE Pro format such as STRING.
   - The default EOL sequence is the escape character for a new line, \n.
5. Click OK; the transaction line of the To File object should now display
   WRITE TEXT "Test1" EOL. This transaction will send the string Test1 to the specified file, myFile.
   Note: If you must shut down at this time, save your work as LAB6-2.

Sending a time stamp to a file

Note: This is a continuation of the prior ToFile transaction; the ToFile object we just developed must be on-screen. The function now( ) is obtained via the following menu selections: Menu Bar: Device => Function & Object Browser => Time & Date submenu provides the current time expressed as a Real Scalar. The value of the Real is the number of seconds elapsed since 00:00 hours on Jan. 1, 0001 AD. As of this date, now( ) returns a value of about 63 gigaseconds. VEE Pro provides this format because it is easier to manipulate mathematically. It also conserves storage space. However, if you want to store the time stamp in a more readable string format, use the TIME STAMP FORMAT in the ToFile object.

6. Open LAB6-2 if necessary; add an input pin to the To File object.
7. Double-click just below the first transaction line in the ToFile object; this will add a second transaction and open a new I/O Transaction box.
8. Double-click the expression list input field to highlight the a (if necessary) and type the expression: now( ).
   Note: The now() function will send the current time from the computer clock to a Real format, which we will next change to the Time Stamp Format.
9. Go to the I/O Transaction box:

10. Click the down-arrow next to the DEFAULT FORMAT field to open a menu, and select TIME STAMP FORMAT. VEE Pro now adds some additional buttons and fields to the I/O Transaction box.
11. Click the down-arrow next to Date & Time, select time.
12. Click HH:MM:SS (hour, minute, seconds format); it will toggle to HH:MM (the hour and minute format).
13. Click 24 HOUR; it will toggle to 12 HOUR (providing the am/pm format).
    Your I/O Transaction box should look like Figure 6.6. If it does, click OK.

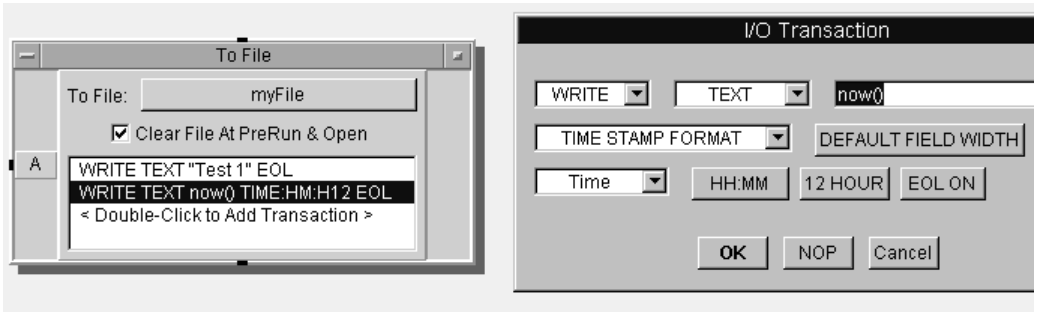


Figure 6.6. The TIME STAMP I/O Transaction box

Sending a real array to a file

We will now create a one-dimension array of four elements using the For Count and Collector objects. We will append this to myFile.

14. Maximize your workspace; drag the ToFile object to the right.
15. Select Menu Bar => Flow => Repeat => For Count; double-click the For Count input field and replace the default number with **4**; move the object to the left-center of your workspace.
16. Select Menu Bar => Data => Collector; place it between the two objects in the workspace.
17. Connect the data-output pin of the For Count object to the data-input pin of the Collector object.
18. Connect the For Count sequence output pin to the XEQ pin of the Collector.
    Note: The Collector will now create the array [0,1,2,3], which you can send to your ToFile data object containing myFile.
19. Connect the Collector data output pin (Array) to the A pin of the ToFile object.
20. Double-click below the second transaction line in the ToFile object to add another I/O Transaction box.
21. Go to the Transaction dialog box:
22. Open the DEFAULT FORMAT menu and select REAL32 FORMAT.
    Note: This offers a new choice of selections. You can leave the default choices, although you may want to investigate them for future reference.
23. Click OK to close the I/O Transaction box and complete the process. The third line of the To File object will now display WRITE TEXT a REAL32 STD EOL. Figure 6.7 shows the function, its ToFile list, and its I/O Transaction box.

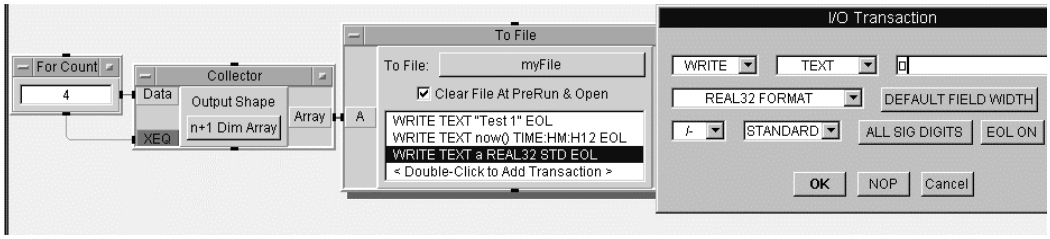


Figure 6.7. Storing data using the ToFile object

24. Save this program as LAB6-2 before proceeding.

Retrieving data using the FromFile object

Note: At this moment, you know the format and location of stored data. You will later learn to retrieve data without having to know its type. In the present example, you know that you stored the name of a test in a String Format, followed by a time stamp in Time Stamp format, then followed by an array of Real numbers. You will now create three transactions in FromFile to read that data back into VEE Pro.

25. Select Menu Bar => I/O => From => File and place it below your ToFile object.
26. Connect the sequence output of the ToFile object to the sequence input of the FromFile object.
    Note: This ensures that ToFile has finished sending data to myFile before FromFile begins to extract data (from myFile).
27. Leave the default file of the FromFile object as myFile.
28. Double-click the From File transaction line to add an I/O Transaction box, because you want tell FromFile how to read the data.
29. Click the format field down-arrow in the FromFile I/O Transaction box; change REAL FORMAT to STRING FORMAT; click OK to close the FromFile I/O Transaction box because all of the other defaults are correct.
    Note 1: The first transaction line of the FromFile object should read:
    READ TEXT x STR.
    This will read the first line of the file as format Text, into the "x" output terminal.
    Note 2: Two more transactions are required to read back the time stamp and the real array.
30. Add a data output to the FromFile object by clicking the mouse cursor over the output area and simultaneously depressing **Ctrl-a**. A dialog box will appear labeled "Select output to add" will appear.
31. Select Y and click OK. VEE Pro will add the data-output pin Y.
32. Add a third terminal by repeating the above procedure. VEE Pro will now offer a data-output pin labeled Z.
33. Select Z and click OK. You now have three outputs labeled X, Y, and Z.

Storing the time stamp

34. Double-click below the first transaction line of the FromFile object to add an I/O Transaction.
35. Go to the FromFile I/O Transaction dialog box.
36. Double-click the expression list input field to highlight **x** (if needed), and type **y**. This causes the second transaction to read data back to output pin **y**.

Note: If this pin remained **x**, the second transaction would overwrite the data placed into **x** by the first transaction.

37. Change REAL FORMAT to STRING FORMAT; then click OK.
    Note: If you want to read the time stamp back as a text string, use STRING FORMAT encoding. The TIME STAMP FORMAT converts the time stamp data back to a Real number. See Figure 6.8.

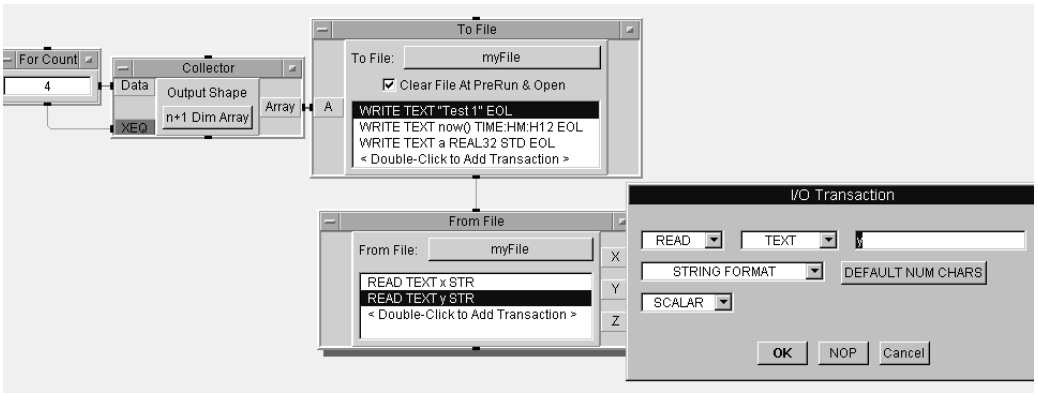


Figure 6.8. Preparing to read data using the FromFile object

Storing real numbers

38. Double-click below the third transaction line to open the I/O Transaction dialog box, go to the expression list input field; double-click to highlight x (if needed); type in z so that the Real array is read back to the Z output pin of the FromFile object.
39. Leave REAL64 FORMAT unchanged as it is correct for this case.
40. Change SCALAR to ARRAY 1D and SIZE to 4; click OK.
    Note 1: If you cannot recall the size of array, you may toggle SIZE to TO END. This will read data to the end of the file without VEE Pro knowing its exact size. You could use this feature to read the entire contents of a file as a string array to examine the file contents.
    Note 2: The FromFile object will now display READ TEXT y STR on the second transaction line, and READ TEXT z REAL64 ARRAY 4 on the third transaction line.
41. Select Menu Bar => Display => AlphaNumeric; clone it twice to get three displays.
42. Connect each output pin of the From File object to the data input pin of an AlphaNumeric display. Size the array connected to the FromFile Z output to hold four values.
    Note: Remember – you can size AlphaNumeric displays as you clone them by simply clicking and dragging the object outline when you first select it from the menu. This procedure works for any object.
43. Run the program. It should look like Figure 6.9.
44 Save again LAB6-2a.

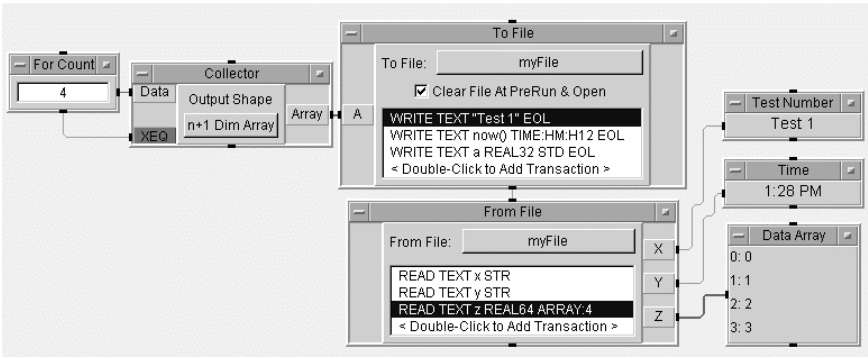


Figure 6.9. Retrieving and displaying data using the FromFile object

Lab 6.3 – Calculating Statistical Parameters

This lab will show you how to set up a function generator, graph its waveform time-domain output, and calculate the waveform mean, median, mode, variance, standard deviation, and rms values.

Open your VEE Pro program and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Calculating various statistical parameters

2. Select Menu Bar => Device => Virtual Source => Function Generator.
3. Place Function Generator in the left-center of your Work Area; change Frequency to 100 and DcOffset to 1.
4. Select Menu Bar => Device => Function & Object Browser => Type: Built-in Functions; Category: Probability & Statistics; Member: mean; click Create Formula; mean(x) will appear. Place it to the top right of the Function Generator object.

   or

   Click f_x on the Tool Bar for Function & Object Browser => Type: Built-in Functions; Category: Probability & Statistics; Member: mean; click Create Formula; mean(x) will appear.

Note: Examine Help from the mean(x) object menu for more details.

5. Repeat step 4 for the following statistical parameters:

   median, mode, variance(vari), standard deviation(sdev), and root-mean-square(rms).
6. Place each Formula object vertically under the mean(x) object.
7. Select Menu Bar => Display => AlphaNumeric; place it to the right of mean(x).
8. Clone the AlphaNumeric object five times; place them beside each Formula Object.
9. Change the names of each AlphaNumeric Object to match the name of its Formula Object.
10. Connect each Formula Object input pin to the Function Generator output (Func) pin.
11. Connect each renamed AlphaNumeric Object input pin to its corresponding Formula Object output pin (Result).
12. Select Menu Bar => Display => Waveform (Time); place it to the right of all of the AlphaNumeric objects.
13. Connect the Waveform (Time) input pin (Trace 1) to the Function Generator output pin.

14. Run your program. It should look like Figure 6.10.

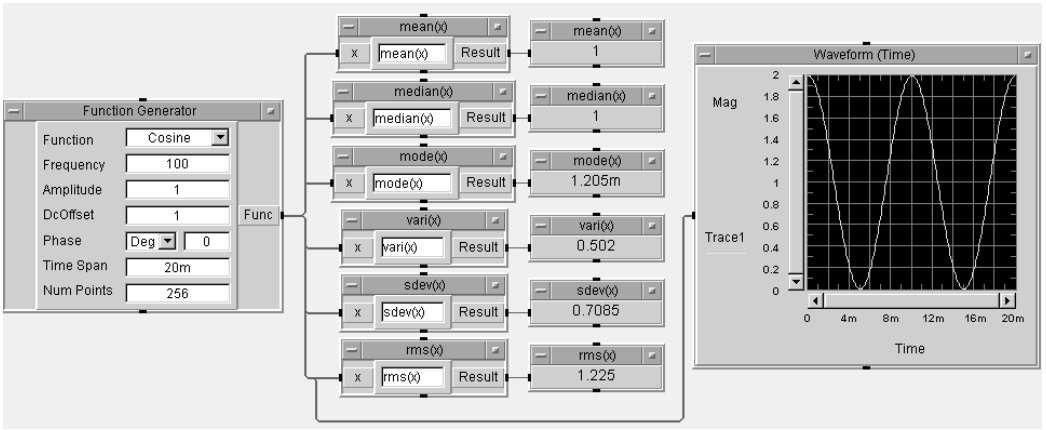


Figure 6.10. Calculating statistical parameters

15. Save your program as LAB6-3.

◊ 16. Vary the following Function Generator parameters: Frequency, DcOffset, Function, Phase, and Num Points (in exponents of 2: 2, 4, 8, 16, 32, etc).
Note: If you change the FncGen frequency, then you must also change the Time Span of FncGen and Waveform (Time) to be the reciprocal of the frequency value.

Your notes:

_____

_____

_____

_____

_____

17. Do *not* save the modifications to this program.

Lab 6.4 – Logging Vehicle Radiator Statistical Data

This lab will show you how to simultaneously monitor Vehicle Radiator temperature and pressure data, displaying the results of calculating standard deviation and root-mean-square statistical analyses of these data points.

Open your VEE Pro program and

1. Clear your Work Area, maximize Main; toggle Program Explorer off.

Displaying a waveform

2. Select Menu Bar => File => Open; open LAB3-4. It should look like Figure 6.11.

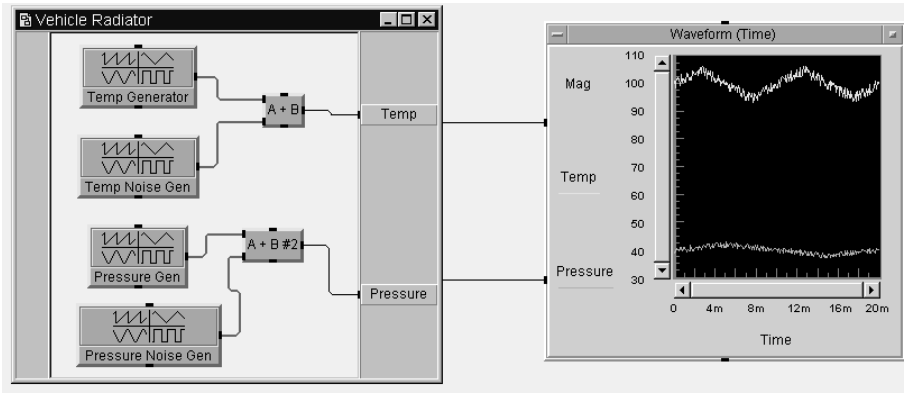


Figure 6.11. The original LAB3-4 after running

3. Reduce Vehicle Radiator to an icon; save As… this program As LAB6-4 immediately.
4. Move Waveform (Time) to the far right of your screen border.
5. Move Vehicle Radiator to the far left of your screen border.
6. Select Menu Bar => Device => Formula; place it between Vehicle Radiator and Waveform (Time), next to Vehicle Radiator.
7. Select Add Terminal, Data Output twice; use Delete Terminal, Data Output to remove "Result"; change the output-pin letters to B (top) and C (bottom).
8. Insert Formula data into the white (editing) space as B=sdev(A);C=rms(A).
9. Clone Formula; place the clone below the other Formula Object.
10. Change the upper Formula Object name to Temperature; change the lower Formula Object name to Pressure.
11. Connect the Temperature object input pin A to the Vehicle Radiator Temp output pin; connect the Pressure object input pin A to the Vehicle Radiator Pressure output pin.
12. Select Menu Bar => Display => Logging AlphaNumeric; clone it three times; place them in a square format (two over; two under) between the Formula and Waveform (Time) objects.
13. Change the name of the upper-left Logging AlphaNumeric to Temp sdev; change the name of the upper-right Logging AlphaNumeric to Temp rms.
14. Change the name of the lower-left Logging AlphaNumeric to Pressure sdev; change the name of the lower-right Logging AlphaNumeric to Pressure rms.
15. Connect the Temperature object data-output pin, pin B to the Temp sdev data-input pin (Data).
16. Connect the Temperature object data-output pin, pin C to the Temp rms data-input pin (Data).
17. Connect the Pressure object data-output pin, pin B to the Pressure sdev data-input pin (Data).
18. Connect the Pressure object data-output pin, pin C to the Pressure rms data-input pin (Data).
19. Double-click separately on each of the four logging-alphanumeric title bars; remove the checkmarks from Initialization: Clear at PreRun and Clear at Activate.
20. Save this modified program; leave its name: LAB6-4. See Figure 6.12.

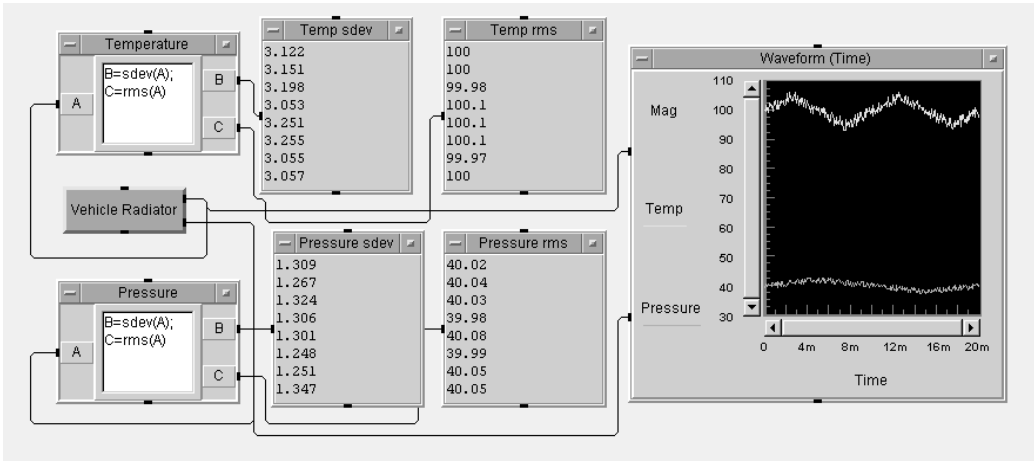


Figure 6.12. Log of vehicle radiator statistical data

◊ 21. Run this program several times as shown in Figure 6.12.
Note: You will later further modify LAB6-4 so it will send its data points to a spreadsheet.

Lesson 6 Summary

Lab 6.1 showed you how to use a Collector object and math expressions to create and display an array of data, and to use a Concatenator object to combine scalars and array elements into a single array.

Lab 6.2 showed you how to move test data to and from files. You learned to store and retrieve three common test result items: a test name, a time stamp, and a one-dimension array of real values.

Lab 6.3 showed you how to set up a function generator, graph its waveform time-domain output, and calculate the waveform mean, median, mode, variance, standard deviation, and rms values.

Lab 6.4 showed you how to simultaneously monitor Vehicle Radiator temperature and pressure data and display the results of calculating standard deviation and root-mean-square statistical analyses of these data points.

You are now ready to learn how to work with records.

Lesson 7
Working with Records

This lesson will examine how to work with records. It consists of a pre-lab and five labs.

Lesson 7 Pre-lab Summary

The following are described in the Lesson 7 Pre-Lab. See Appendix E, page E-51.

- Creating and Manipulating Records
- The Record Constant
- The Build-Record Object
- The Set Field and Get Field Objects
- The Unbuild-Record Object

As noted in all previous lessons, Appendix B includes a cross-reference to each of these items and to all objects and subprograms contained in the labs of this and later lessons.

Overview

Lab 7.1 – Building Records to Hold Various Data Types
This lab will show you how to build a record that stores three different data types. You will examine program results during development and modify the display format of the time stamp.

Lab 7.2 – Extracting and Displaying Record Fields
This lab will show you how to devise a program that can extract and display test data from a record and further examine that test data.

Lab 7.3 – Setting a Field in a Record
This lab will show you how to devise a program that can be used to alter data in specific fields so you can use the same record many times with different tests.

Lab 7.4 – Unbuilding a Record in a Single Step
This lab will show you how to extract all record fields and provide a list of the field names and their types.

Lab 7.5 – Displaying Vehicle Radiator Statistical Calculations
This lab will show you how to expand your statistical computations for the Vehicle Radiator, and how to display them.

Lab 7.1 – Building Records to Hold Various Data Types

This lab will show you how to build a record that stores three different data types: 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. You will examine program results during development and modify the display format of the time stamp.
(Later labs will show you how to change, process and store test data.)

Reminder: Note the "pins" on each object –
data-input pins are on the left,
data-output pins are on the right, and
operation-sequence pins are on the top (in) and bottom (out).
See Appendix A for more information related to pin locations.

1. Clear your Work Area; maximize Main, and toggle Program Explorer OFF.

Building a record

Reminder: This example record will contain three fields:

- the name of the 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.

2. Select Menu Bar => Data => Constant => Text; enter Test1 as the test-data name in the white-space (data field) area; double-click the Title Bar and rename the Object: Test Name; place Test Name near the upper left corner of Main.
3. Create the Time & Date field: Menu Bar => Device => Function & Object Browser;
   - Select Type: Built-in Functions;
   - Category: Time & Date;
   - Member: now;
4. Click on Create Formula – the now() Object will appear.
5. Change the name to Time in Gsec.
6. Place Time under Text Name.
7. Select Menu Bar => Data => Constant => Real32; change its name to Data.
8. Place Data below Time.
9. Double-click on the Title Bar of Data to open its Constant Properties box.
10. Select Configuration: 1D Array; check Fixed Size; enter Size: 4; click OK; 4 rows of numbers will appear in the Real data field; the left column will read: 0000, 0001, 0002, 0003.
11. Change the four zeros in the right column to the following sequence, top-down:
    2.2, 3.3, 4.4, and 5.5.
    Note: The Tab key may be used to move from one entry to the next down the right-hand column.
12. Select Menu Bar => Data => Build Data => Record; place it to the right of the three other objects.

13. Add a third input pin to Build Record; the input pins should read: A, B, C.
14. Double-click on each of the three input pins; starting from the top pin, rename them: testname, time, and data; click OK.
15. Toggle the Build Record Output Shape to 1D Array.
16. Connect the Build Record data-input pins as follows:

    - testname to Test Name (test1),
    - time to Time in Gsec, and
    - data to Data.

17. Select Menu Bar => Display => Alphanumeric; connect the Build Record data-output pin to the AlphaNumeric display data-input pin; enlarge the display enough to hold the array.
18. Run this program; it should match the results Figure 7.1.
19. Double-click on Build Record output pin (Figure 7.2); click each rectangle under the Value: column of the Data: box (First, Prev, Next, Last) and compare Figure 7.2 with Figure 7.1; click OK when done.

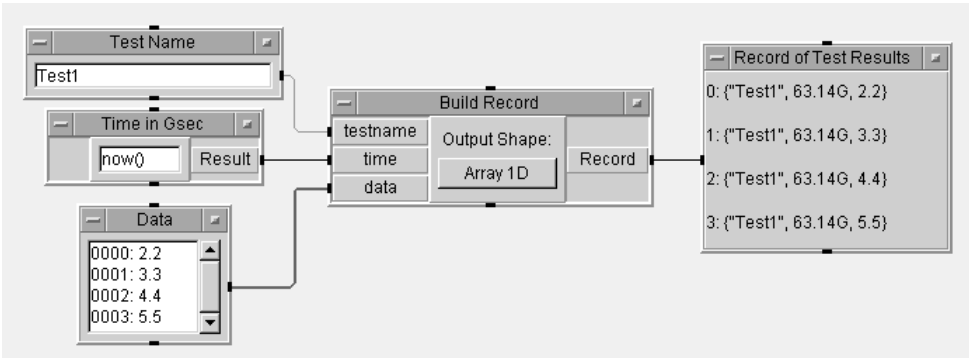


Figure 7.1. Data output of three types on a record

20. Save As LAB7-1.

◊ 21. Vary the values in the Data object; re-run the program several times, noting the difference in the Test Results – the final values in each test record.

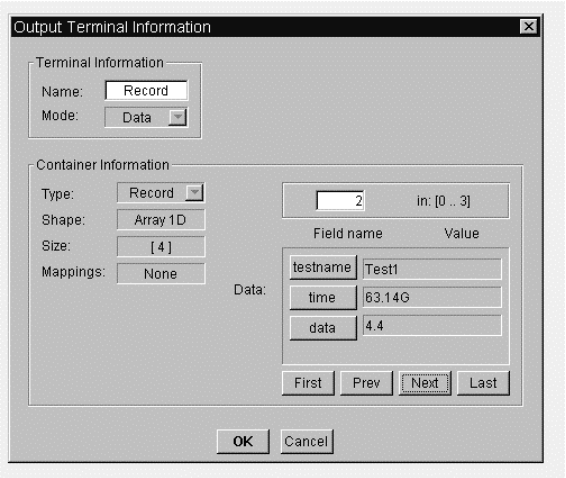


Figure 7.2. The Record Output pin information; data selection

22. Do not save this revised program.

Lab 7.2 – Extracting and Displaying Record Fields

This lab will show you how to devise a program that can extract and display test data from a record and further examine that test data. (Later labs will show you how to process and store test data.)

1. Open LAB7-1; Save as…LAB7-2 immediately; maximize Main; delete the AlphaNumeric display.

Extracting and displaying test data with the Get Field object

2. Select Menu Bar => Data => Access Record => Get Field. Place it beside and slightly above the Build Record Object. The default title on the Object that appears is rec.field.
   Note: Rec.field is the default expression in the Get Field formula. It can be edited to retrieve any field. Rec refers to the record at the data-input pin that has the same name.
3. Clone rec.field twice; place the objects under the original rec.field Object.
4. Connect the Build Record data-output to each of the rec.field object data-inputs; change the Build Record Output Shape to Scalar.
5. Edit the three rec.field object formula expressions (white edit field), from top to bottom, to separately read: rec.testname, rec.time, and rec.data.
   Note: The rec.field data-field names must exactly match the build-record input-pin names. They may be either upper-case or lower-case – they are not case sensitive.
6. Select Menu Bar => Display => AlphaNumeric; clone it twice, placing them to the right of the three rec.field objects.
7. Connect each AlphaNumeric display data-input pin to an individual rec.field data-output pin (Result).

Note: The middle (Time and Date) display will have to be stretched horizontally; the bottom (data) display will have to be stretched vertically to be approximately three times higher than the other two displays in order to accommodate the array.

8. Change the title of each AlphaNumeric display to match its rec.field input-pin name: Test1, Time, and Data; then modify the middle display to read: Time and Date.
9. From the Time and Date AlphaNumeric Object menu, open the Properties Dialog box, click the Number tab.
10. Go to the Time AlphaNumeric Object Properties Dialog box, remove the check mark to the left of Global Format, then set all values to match Figure 7.3; click OK.

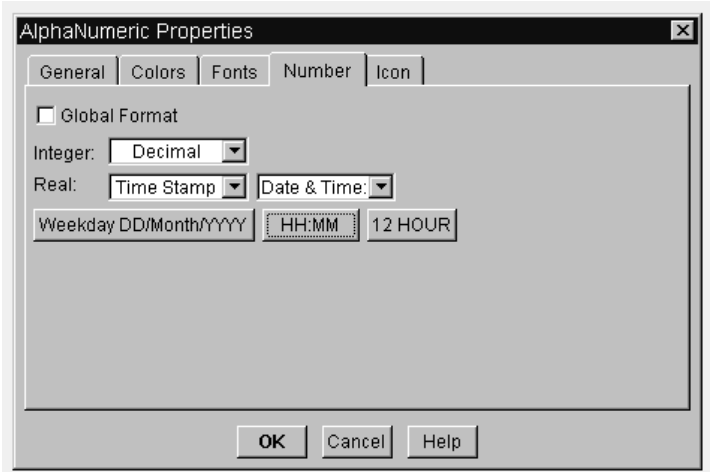


Figure 7.3. The AlphaNumeric Properties box: number

11. Run your program. It should look like Figure 7.4.

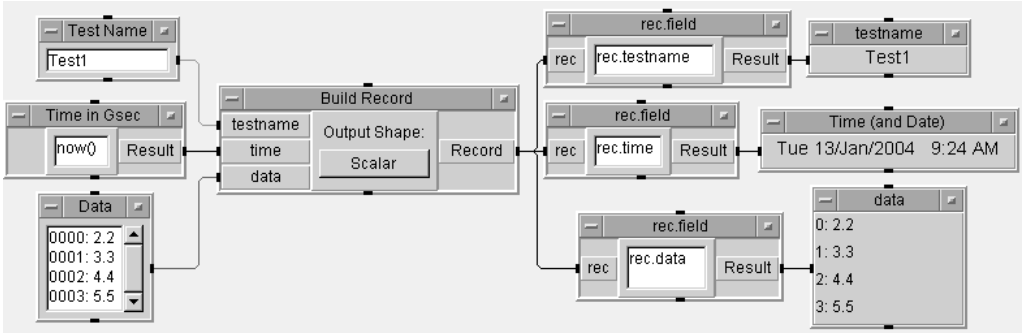


Figure 7.4. Using the Get Field object

Note 1: When the Get Field object is placed on the Work Area, it is titled: rec.field.

Note 2: You should realize that VEE Pro is not case sensitive in this situation.

12. Save your program again as LAB7-2.
◊ 13. Change the name of one or more of the rec.field data-field names; re-run your program and observe the effect.
◊ 14. Go to the data alphanumeric display; open its Properties box; select Number; remove the checkmark on Global Format; change Real to Scientific; click OK.
◊ 15. Go to the Data object white (editing) field; change at least one value to an integer and change the other three values if desired; click OK; run this program and observe the data alphanumeric values.
◊ 16. Double-click on the Build Record's "Record" output terminal to observe the output-terminal information. Note that the record shape is Scalar where the data field is actually an array. Click on the Build Record's "Scalar" button to change it back to "Array 1D"; run the program again.
◊ 17. Double-click on the Build Record's "Record" output terminal again. Note that it is an array of records even though the "data" field of each record is a scalar.
Note: The difference between an array of records and a record with a field (which is an array) is often confused.
18. Exit your program without saving it.

Lab 7.3 – Setting a Field in a Record

This lab will show you how to devise a program that can be used to alter data in specific fields so you can use the same record many times with different tests. (Later labs will show you how to store test data.)

1. Open LAB7-2; Save As… LAB7-3 immediately.

Altering data in a specific record field with the Set Field object

2. Delete all objects that follow Build Record.
Use the Tool Bar scissors.
or
Select each object and depress Ctrl-x.
3. Go to Build Record; change its Output Shape to Scalar if necessary.
4. Select Menu Bar => Data => Access Record => Set Field; place it to the right of Build Record. (The Object title will be rec.field = b.)
5. Connect the Build Record Object output pin (Record) to the data-input pin (rec) on the rec.field = b Object.
6. Edit the white-space expression in rec.field = b to read: rec.data[*]=b.
Note: rec.data[*]=b is using both record field and sub-array notation; it first accesses the "data" field from Record "rec.", then it sets all elements of the (array) "data" field to the values in "b" which must also be an array.
7. Select Menu Bar = Data => Constant => Real32; place it below and on the right side of Build Record.
8. Select its (Constant) Properties box; select Configuration: 1D Array; edit Size to 4; enable Enable Indices; click OK.
9. Enter the values 1, 2, 3, 4 into the right column in the Real Object editing (white) space starting at the top: Double-click the first row and type over the 0 with a 1. Tab down and type over the right-most zero in succeeding rows with 2, 3, and 4.

10. Connect the Real32 output to the b data-input pin of rec.field=b.
11. Select Menu Bar => Data => Access Record => Get Field; place it below rec.field=b; edit the white space to rec.data.
12. Connect the rec.field = b data-output pin (rec) to the rec.field data-input pin.
13. Select Menu Bar => Display => AlphaNumeric; place it to the right of rec.field.
14. Size AlphaNumeric to accommodate an array of four tests; rename it Four Tests.
15. Connect the rec.field data-output pin (Result) to the Four Tests data-input pin.
16. Run your program. It should look like Figure 7.5.

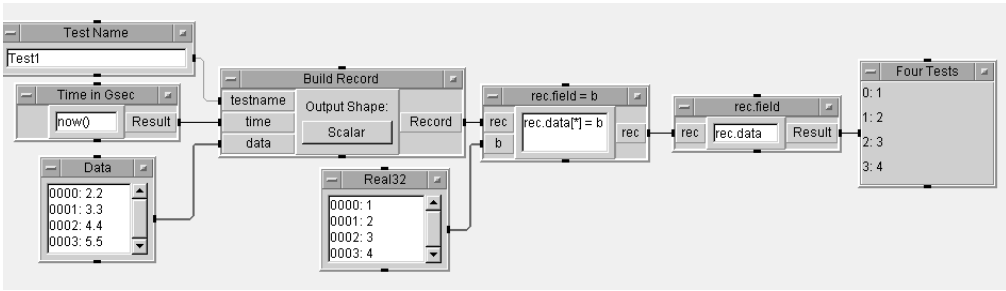


Figure 7.5. Using the Set Field object

17. Save your program as LAB7-3.

Lab 7.4 – Unbuilding a Record in a Single Step

This technique will extract all record fields, and provide a list of the field names and their types.

1. Open LAB7-3; delete all objects after Build Record; maximize Main and toggle Program Explorer OFF; save it immediately as LAB7-4.

Unbuilding a record in a single step

2. Select Menu Bar => Data => UnBuild Data => Record; place it beside Build Record.
3. Go to the Unbuild Record Open View; connect its data-input pin (Record Data) to the Build Record data-output pin (Record).
4. Add another data-output pin to the Unbuild Record Object via its object menu.
5. Double-click on, and rename the A, B, and C outputs to the field names in this order:
   testname
   time
   data
6. Select Menu Bar => Display => AlphaNumeric; clone it four times from its object menu.
7. Connect the five displays to the five output pins on Unbuild Record.
8. Change the five display title-bar names to read: Name, Type, Test, Time & Date, and Data to agree with their input signals; size and arrange them to fit on your screen.
   Note: You will have to enlarge the displays connected to Name List, Type List, and Data to accommodate arrays.

9. Open the Time & Date alphanumeric Properties box => Number.
10. Remove the check mark to the left of Global Format.
11. Open the I/O Transaction box; click on Real: Standard; select Time Stamp:
    - toggle Month DD, YYYY so it becomes Weekday, Month DD, YYYY;
    - toggle HH:MM:SS so it becomes HH:MM;
    - toggle 24 HOUR so it becomes 12 HOUR;
    - click OK.
12. Run this program. See Figure 7.6.

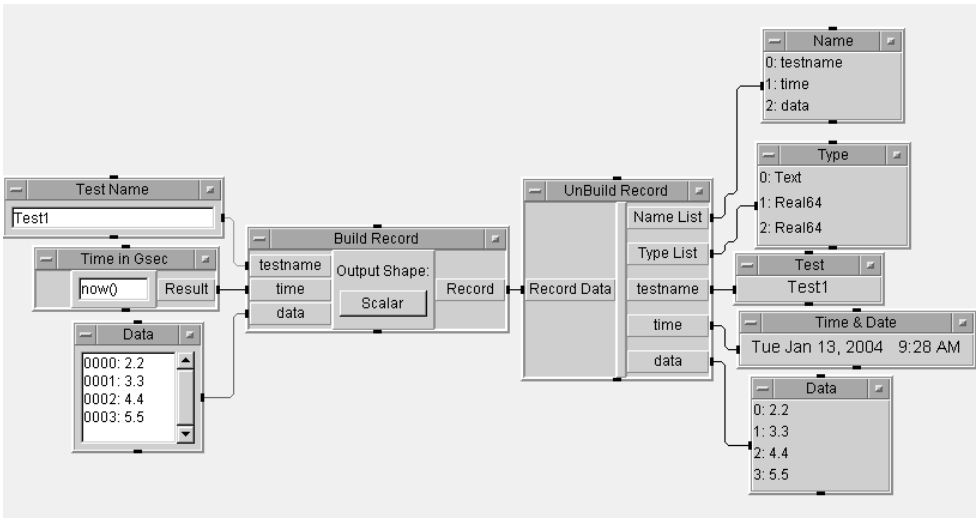


Figure 7.6. Using the Unbuild Record object

13. Save this program as LAB7-4.
    Note 1: The Name List pin provides the names: testname, time, and data of the three fields in the record. (For advanced users, this list of field names can be used to, via programming, create a Formula expression to do a GetField or a SetField operation. This is useful in situations where the Record field names are not known until run time.)
    Note 2: The Type List pin identifies testname as type Text and time and data as Real types.
    Note 3: The testname pin identifies the test data value as Test1.
    Note 4: The Time pin displays the time value in the format that you select.
    Note 5: The Data pin displays the processed data.

◊ 14. Change the Data input values; run the program again and observe the changes in the Data output object.

◊ 15. Change the Unbuild Record output pin "time" to time&date; then change the Build Record "time" record input pin to time&date also. Attempt to run this program. What do you observe?

16. Exit this program without saving it.

Lab 7.5 – Displaying Vehicle Radiator Statistical Calculations

This lab will show you how to expand your statistical computations for the Vehicle Radiator and how to display them.

1. Open LAB3-4; copy the Vehicle Radiator.
2. Close LAB3-4; open LAB6-3.
3. Paste this new Vehicle Radiator object in the lower-left of the screen.
4. Save this partial program immediately as LAB7-5.
5. Size the bottom of the Work Area so the Vehicle Radiator UserFunction Iconic View will be visible.
   Reminder: The Vehicle Radiator Iconic View will appear at the bottom of the Work Area only after you open it and then close it via its "–" button in the upper-right-hand corner.

Modifying the Vehicle Radiator program to include statistical calculations

6. Delete the Function Generator.
7. Move the Vehicle Radiator to the Function Generator location.
8. Move the Waveform (Time) object so it is under the six Formula objects.
9. Size Waveform (Time) vertically so it will fit in the Work Area.
10. Hold down the Ctrl key; use your mouse to select all six Formula and six AlphaNumeric objects; select Edit => Copy.
11. Select Edit => Paste; place the twelve objects to the extreme right of the ones you copied. See Figure 7.7.

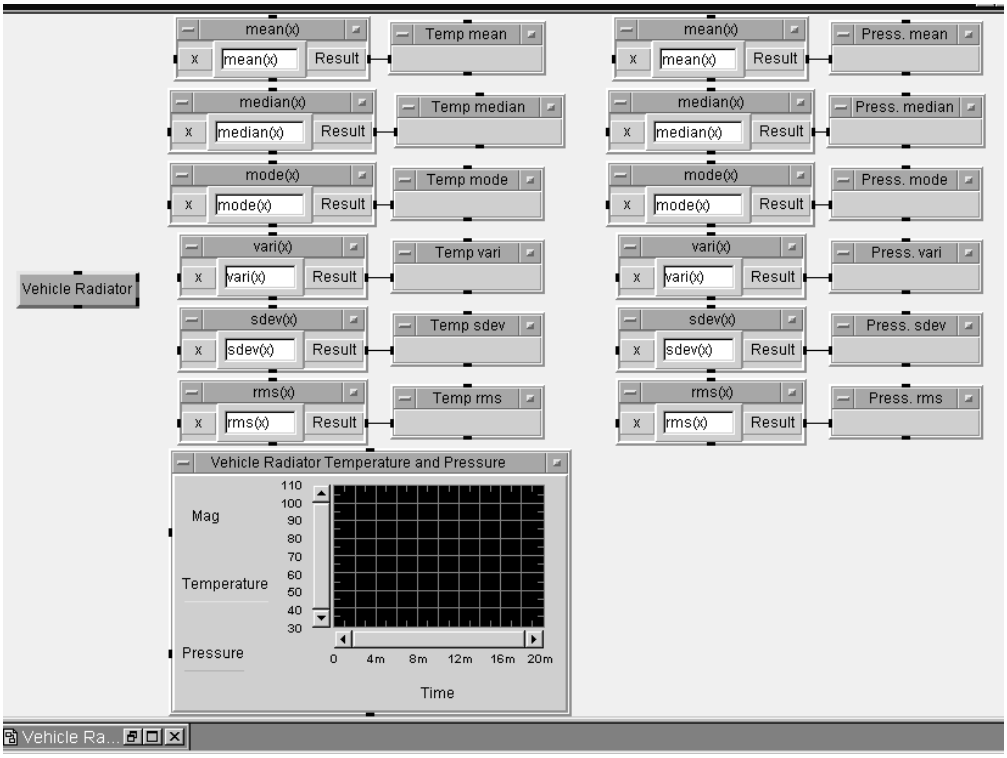


Figure 7.7. Initial positioning of all objects

12. Add to the alphanumeric title bars as follows:
    the word Temp, deleting the **(x)** on the six left-hand objects;
    the word Press., deleting the **(x)** on the six right-hand objects.
13. Connect the Vehicle Radiator upper output pin (Temp) to the six left-hand statistical formula objects.
14. Connect the Vehicle Radiator lower output pin (Pressure) to the six right-hand statistical formula objects.
15. Change the Waveform (Time) title bar to read: Vehicle Radiator Temperature and Pressure.
16. Add a second input pin to the Vehicle Radiator Temperature and Pressure display object.
17. Label the top display input pin: Temperature; label the bottom display input pin: Pressure.
18. Connect the display top input pin to the Vehicle Radiator top output pin (Temp).
19. Connect the display bottom input pin to the Vehicle Radiator bottom output pin (Pressure).
20. Save this program again.
21. Run this program; it should look like Figure 7.8.

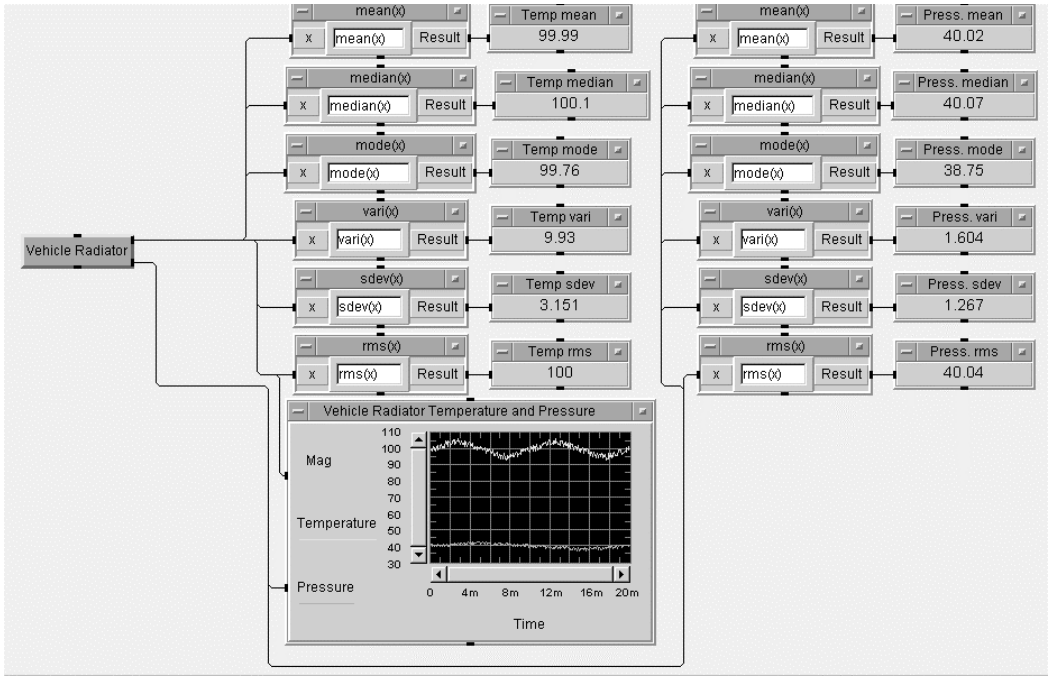


Figure 7.8. Vehicle Radiator Statistics program after running

22. Exit this program.

Lesson 7 Summary

The first four labs of this lesson examined how to work with records, including the objects:

- Build Record
- Set Field
- Get Field
- Unbuild Record

The last lesson incorporated statistical calculations into the Vehicle Radiator.

Lab 7.1 showed you how to build a record that stores three different data types. You examined the program results during development and modified the display format of the time stamp.

Lab 7.2 showed you how to devise a program that can extract and display test data from a record; you can further examine that test data.

Lab 7.3 showed you how to devise a program that can be used to alter data in specific fields so you can use the same record many times with different tests.

Lab 7.4 showed you how to extract all record fields and provide a list of the field names and their types.

Lab 7.5 showed you how to compute, display, and store statistics for a variety of vehicle radiator test runs. This lesson prepares you for learning to create spreadsheets and reports.

You are now ready to learn to work with data sets, databases, and operator interfaces.

Lesson 8
Working with Databases and Operator Interfaces

This lesson will examine how to work with data sets, databases, and operator interfaces. It consists of a pre-lab and four labs.

Lesson 8 Pre-lab Summary

The following are described in Lesson 8 Pre-lab. See Appendix E, page E-52.

- Data Sets
- Test Databases
- Operator Interfaces
- Secured Run Time Versions
- To DataSet
- From DataSet
- Record Constant

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 8.1 – Learning to Work with Data Sets
A Data Set is an array of records stored in a file. In this lab, you will create an array of ten records, each containing three fields.

Lab 8.2 – Customizing a Simple Test Database
This lab will show you how to customize a simple database that will search and sort data.

Lab 8.3 – Creating an Operator Interface for Search Operations
This lab will present operator interfaces and show you how to provide a test menu. This test menu will allow the user to select a particular test from which all related test data will be available.

Lab 8.4 – Building a Vehicle Radiator Operator Interface
This lab will show you how to compute, display, and store statistics for a variety of vehicle radiator test runs and how to control the process and displays via an operator interface. This lab will also show you how to create and apply a Secured Run Time Version of an operator interface.

Lab 8.1 – Learning to Work with Data Sets

A Data Set is an array of records stored in a file. In this lab, you will create an array of ten records, each containing three fields with

- a test name,
- a Real Scalar that could be a time stamp, and
- an array of Reals.

You will store this array of records in a Data Set, retrieve all ten records and display them.

Open your VEE program and

1. Clear your Work Area, remove the Program Explorer, and maximize Main.
   Note: If your plan is to place this lab into your library for future adaptation and use, then omit the Start button – steps 2 and 5. The program will operate from the Run button.

Storing a record from a Data Set object

2. Select Menu Bar => Flow => Start; place it in the upper-left corner of your screen.
3. Select Menu Bar => Flow => Repeat => For Count; place it below Start.
4. Select Menu Bar => Device => Formula; place it near and to the right of For Count.
5. Connect Start to the sequence-input (top) pin of For Count.
6. Connect the For Count data-output pin to the Formula data-input pin.
7. Double-click the Formula expression field (white) to highlight the default expression.
8. Type "test" + a.
   Note: When you click "Start" or "Run", For Count will place integers zero through nine sequentially on the A pin of Formula. Formula will add these integers to the word "test". The result will be Text Scalars: test0, test1, test, …, test9 as outputs. These values will fill the first fields in the ten Records. To observe this effect, first click start, then double-click Result in the Formula object.
9. Select Menu Bar => Data => Build Data => Record; place it to the right of Formula.
10. Add a third data-input pin to Build Record via the Build Record object menu.
11. Connect the data output of Formula (Result) to the A input of Build Record.
12. Select Menu Bar => Device => Function & Object Browser.
13. Choose Built-in Functions, Probability & Statistics, and "random"; click Create Formula; you will now have the random(low,high) object; place it below and between For Count and Formula.
14. Delete both random… input pins of the random(low,high) object.
15. Change the input parameters in the parentheses of random(low,high) from low to 0, and from high to 1.
16. Rename the Object: random(low,high) to Random Number.
17. Connect the Random Number data-output pin (Result) to the B pin of Build Record.

18. Connect the Formula sequence-output (bottom) pin to the sequence-input (top) pin of Random Number.
    Note: This connection will assure you that, on each of the ten iterations of this program, a new random number will be placed into the B field of that particular record.
19. Select Menu Bar => Data => Constant => Real32; place it temporarily below and to the left of Build Record; click Properties; change its name to Real Array.
20. Go to the Real Array Constant Properties Box; click 1DArray under Configuration; change its Size to 3; click OK.
21. Place the Real Array object under Build Record and to the right of Random Number.
22. Connect the data-output pin of Real Array to the C input pin on Build Record. See Figure 8.1.

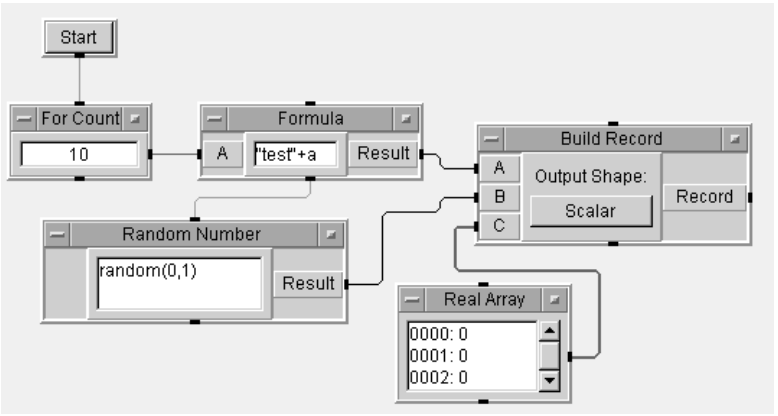


Figure 8.1. Partial layout of Lab 8.1

23. Double-click the first array entry to highlight it; type the numbers 1, 2, and 3; use the Tab key to select each consecutive number.
24. Select Menu Bar => I/O => To => DataSet; place it below Build Record.
25. Connect the data-output pin of Build Record to the To Data Set data-input pin.
26. Examine the To Data Set object; leave the default filename (myFile); place a checkmark in Clear File At PreRun.
    Note: For later reference, it is better to change pin names to names that are more descriptive; programmers changing programs at a later date will find these more specific references more helpful in following the old program.
27. Save this program as LAB8-1.

Retrieving and displaying a record from a Data Set

Note: You will be using the From Data Set and Record Constant objects.

28. Convert Random Number and Real Array to icons.
29. Select Menu Bar => I/O => From => DataSet; place it below and to the left of Real Array.
30. Leave the default-filename of From Data Set: myFile.
31. Click on Get Records; toggle from One to All.
32. Leave the default of 1 in the expression (white) field at the bottom.

Note: VEE will now look at the Data Set in myFile and find "All" the records that meet the criterion in the expression field. The 1 signifies a TRUE condition. That is, all of the records fit the criterion. Thus, the entire array of records in that file will be placed on the output pin labeled Rec.

33. Connect the For Count sequence-output (bottom) pin to the From Data Set sequence-input (top) pin.
34. Select Menu Bar => Data => Constant => Record; place it below To Data Set; change its name to Record Constant via the Properties box.
35. Open Record Constant Object menu; select Add Field, select Field C; click OK; the icon will expand vertically to display the three Field Names: A, B, C.
36. Select the Object Menu => Add Terminal => Control Input; select Default Value from the list box presented; click OK.
37. Connect From Data Set output pin (Rec) to input pin (Default Value) on Record Constant; save this program as LAB8-1.

    Note 1: The record received becomes the default value – Record Constant receives an array of records from the From Data Set. Then it formats itself to display that array.

    Note 2: A dashed line will appear between the two objects. This dashed line indicates a control line. See Figure 8.2.

    Note 3: The output pin of the From Data Set could be connected to the sequence-in (top) pin of the Record Constant rather than to the Default (control) pin. This connection is considered more reliable because the control inputs will fire either without being connected or without receiving data.

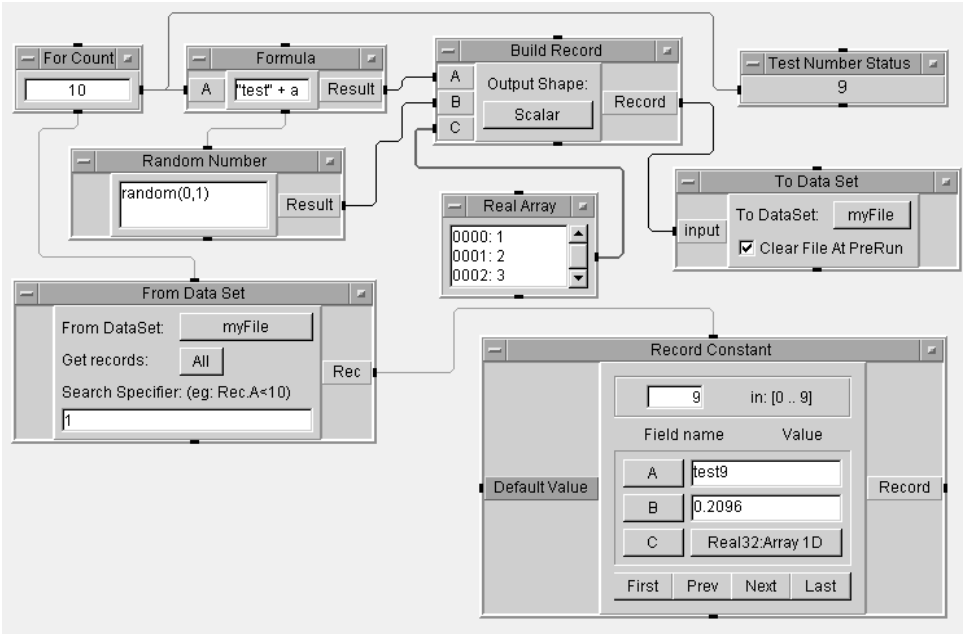


Figure 8.2. Storing and retrieving data using Data Sets

38. Add a temporary AlphaNumeric connected to For Count; change its name to: Test Number Status.
39. Turn on the "Show Data Flow" icon in the Tool Bar.
    Note: The part that sends data to myFile is finished executing before trying to read data from the file. Double-click on the From Data Set Rec output pin to observe the ten records (0 through 9) via the First, Prev, Next, and Last buttons.
40. Run this program; try to follow the "ping" square as VEE progresses through the program execution; watch the Test Number Status display. (If you have not done this earlier, then expand the Record Constant object to show its three input pins: A, B, and C, its Field name, and each Value as shown in Figure 8.2.)
41. Select Record Constant; toggle among First, Prev, Next, and Last to view the data.
42. Turn on both "Show Execution Flow" and "Show Data Flow" and run this program to follow the flow.
43. Disconnect the sequence line from the Formula to the Random object; run this program again. What do you observe?
44. Do not save this program.

Lab 8.2 – Customizing a Simple Test Database

This lab will show you how to customize a simple database that will search and sort data. This program will be modified in the next lab so you can learn to create an operator interface.

Open your VEE program and

1. Clear your Work Area, remove the Program Explorer, and maximize Main.

Performing a search and sort operation with Data Sets

2. Open LAB8-1; Save As… LAB8-2 immediately.
3. Double-click on the From Data Set expression (white) field; enter Rec.B>=0.5.
   Note: Field B is the random number in our code. The From Data Set object will now provide all records that match the search criteria (field B value >=0.5) at its Rec pin.
4. Add a data-output pin to the From Data Set object; it will appear as EOF.
   Note: This pin will "fire" if no records match the expression-field criteria.
5. Save this program again as LAB8-2.
6. Run this program; it should look like Figure 8.3.

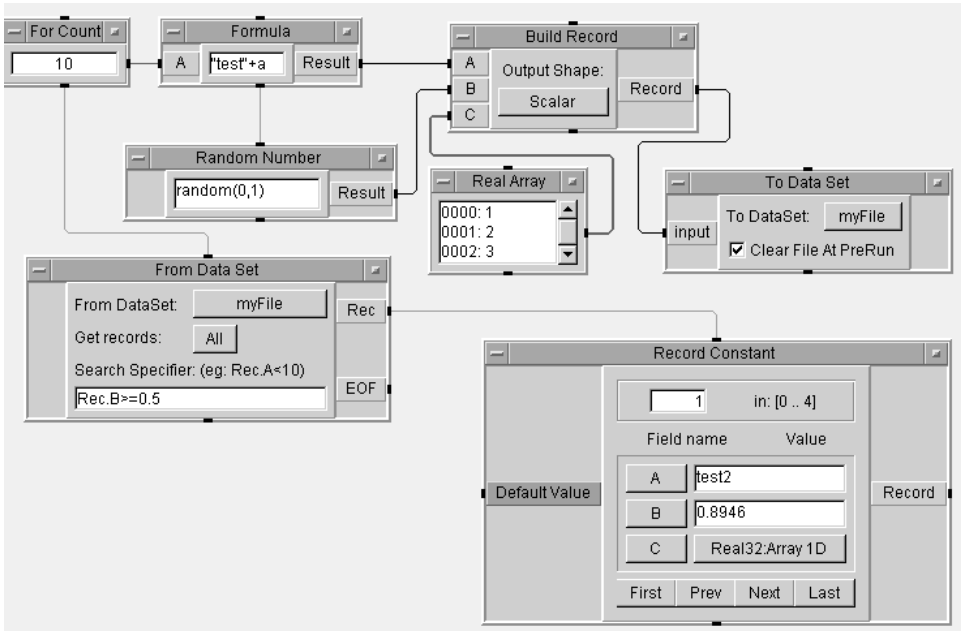


Figure 8.3. A Search operation with Data Sets

◊ 7. Change the value in the From Data Set expression field, starting with 0.7. Run this program again; cycle through the buttons at the bottom of Record Constant; note those tests that are missing.
Note: The value in the Record Constant upper-left corner will tell you how many tests passed the criterion of the From Data Set Search Specifier.

◊ 8. Choose other values for the Search Specifier; note the quantities of tests that pass this criterion.
Note: If VEE does not find a record that meets your criterion, you will receive an error message which can be received by adding an error output terminal to the From Data Set object. Debugging techniques will learned in future labs.

Lab 8.3 – Creating an Operator Interface for Search Operations

This lab will present operator interfaces and show you how to provide a test menu. This test menu will allow the user to select a particular test (from test0 through test9 in this lab) from which all related test data will be available. It will also allow the user to display the specified test results with the fields and values labeled. The user can interact with the display to gain more detailed information and provide clear instructions for operating the program. These are later considered to be program specifications.

Note: A sample of Data Controls and Boxes is given below in Figure 8.4:

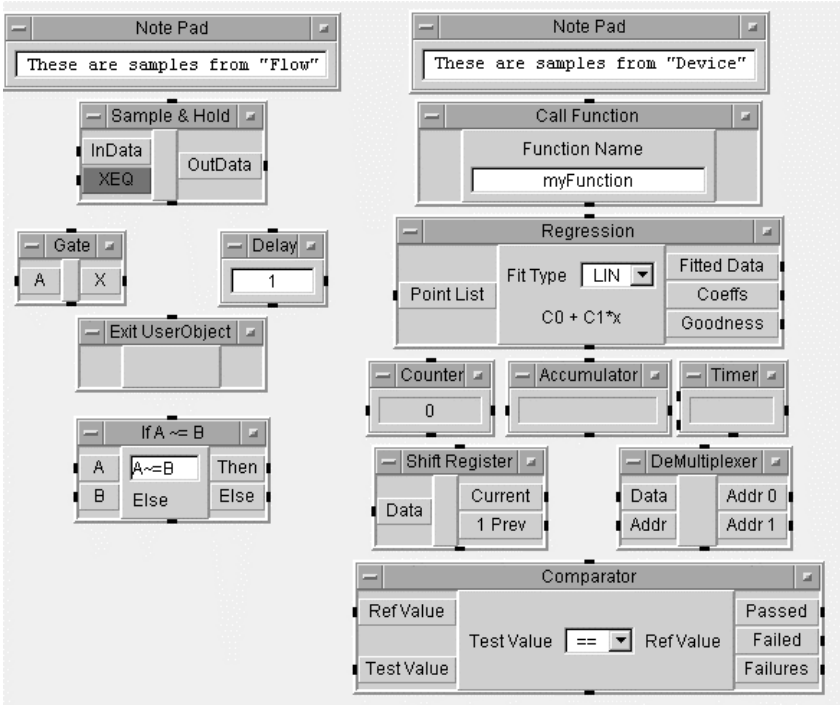


Figure 8.4. A sample of Data Controls and Boxes

Open your VEE program and

1. Clear your Work Area, remove the Program Explorer, and maximize Main.

Preparing for a search operation

2. Open LAB8-2; Save As… LAB8-3 immediately; move the entire program to the right approximately 5 cm (4 inches).
3. Open the From Data Set Object menu.
4. Select Add Terminal… => Control Input… .
5. Select Formula from the menu presented, click OK. A Formula input pin will appear.
   Note: This will allow you to choose the expression via the From Data Set object.
6. Click the From Data Set Get Records field; toggle it from All to One if necessary.
   Note 1: You will work with one test record at a time using this approach.
   Note 2: You want the user to select a particular test name. These names are located in field A of all records. When the program is run, the following expression will appear in the From Data Set Search Specifier window:
   Rec.A==<test name in quotation marks>
   This statement means that the From Data Set object should provide the record where field A matches the test name that the user has selected.
   Example: Rec.A==“test6” would extract the test record for that test which you could then display.

7. Select Menu Bar => Data => Selection Control => Radio Buttons; place it in the open space to the left of the other objects.
8. Open the Radio Button Object menu and select Edit Enum Values… . (Double-click; item 1 will be highlighted.)
9. Type the values test0, test1, test2, …, test9 using the Tab key after each entry for other than that entry of test9.
10. Click OK. (You will see all ten entries.)
11. Use the Properties selection in the Radio Buttons object menu to change the object name to Test Menu.
12. Go to the same Properties Box; under Execution, select Auto Execute; click OK.
    Note: Your program can now execute whenever the operator makes a menu selection. Therefore, you can now delete the Start object.
13. Press the mouse right button over the Start Object and select Cut.
    Note: You will only want the program to execute after you make a menu selection, thus:
14. Connect the Test Menu upper data-output pin (Enum) to the For Count (top) sequence-input pin.
    Note: The output of the Test Menu goes into a Formula object, sending the correct formula to the From Data Set object. Next, create a formula for this program.
15. Delete the sequence line between For Count and From Data Set.
16. Select Menu Bar => Device => Formula; enter the following expression in the white space:

    “Rec.A==” + “\”” + A + “\””

    Note 1: This is a text string where data are connected to text.
    Note 2: The slashes are backwards slashes!
17. Connect the data-input pin on the Formula object to the Test Menu Enum data-output pin.
18. Connect the Formula output-control (bottom) pin to the sequence input (top) pin on the From Data Set object.
19. Connect the Formula data-output pin (Result) to the From Data Set input pin (labeled Formula).
20. Connect the For Count sequence-output (bottom) pin to the Formula:

    “Rec.A= =”+“\””+A+“\”” sequence-input (top) pin.

    Note: This assures that the From Data Set object (and the Formula, in this case) do not execute until the For Count loop finishes sending data to the data set.
21. Select Menu Bar => Display => Note Pad, change the title to Test Results Database Instructions.
    Note: This will create a box displaying instructions for the user.
22. Click on the Note Pad white space to get a cursor. Type:

    Select the test results you
    want from the Test Menu.

    Note: Use the Size command in the Note Pad Object menu to reduce its size.
23. Change the name of Record Constant to Test Results; change the From Data Set Rec output pin to the Test Result Default Value input pin.
24. Save, then run, this program by selecting a Test value from the Test Menu object. See Figure 8.5.
    Note: The Test Menu object has Auto Execute turned on. Thus, you need only to make a menu selection to run this program. The Auto Execute menu acts like a start button.

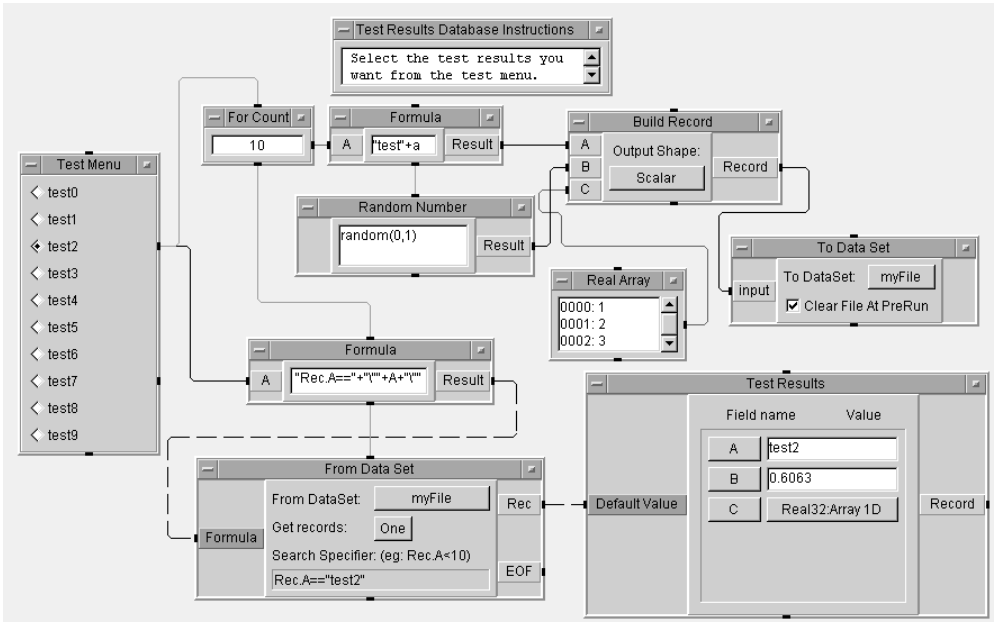


Figure 8.5. Adding a Test Menu to the Search Operations with Data Sets

◊ 25. Change the Test Menu test-number selection several times; observe the information changes in: From Data Set and Test Results.

Your notes:

_____

_____

_____

_____

_____

Creating an operator interface

26. Return to the LAB8-3 screen.
27. Depress and hold down Ctrl; click on
    Test Menu,
    Test Results Database Instructions, and
    Test Results.
    Note 1: The selected objects will show a shadow. Verify that no other objects are selected.
    Note 2: Be certain that the Note Pad is not an icon.
28. Select Menu Bar => Edit => Add to Panel; the selected objects will appear in a Panel View.

Note 1: Your operator interface appears as a panel view. You can then move and size the objects to suit yourself. For an example, see Figure 8.6.
Note 2: If the Add to Panel selection is "grayed out", it means that you do not have any objects selected in the Work Area.
Note 3: The Record Field data can be displayed by clicking on the C name: Real32:Array1D.

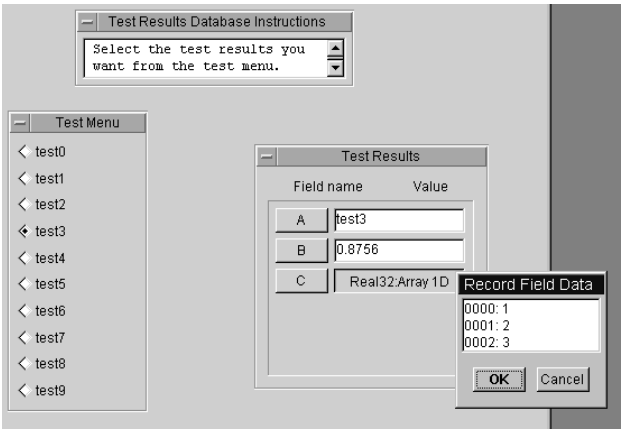


Figure 8.6. The Operator Interface for your database

29. Save As… LAB8-3a; run your program.
    Note 1: You can get more detailed information on any given record by clicking the field names or the values in the Record Constant object, now named Test Results.
    Note 2: To prevent someone else from changing the program and/or the operator interfaces, click File => Save Secured Run Time Version… . Name the program; VEE will automatically add a *.vxe extension to separate it from unsecured versions. Be certain to always retain the original program as there is no way to "unsecure" it.

Lab 8.4 – Building a Vehicle Radiator Operator Interface

This lab will show you how to compute, display, and store statistics for a variety of vehicle radiator test runs, and how to control the process and displays via an Operator Interface.

Open your VEE program and

1. Clear your Work Area, remove the Program Explorer, and maximize Main.

Preparing a Vehicle Radiator test database

2. Select File => Open; click on LAB7-5.
3. Save As… LAB8-4 immediately.
4. Select Menu Bar =>Display => Note Pad; place it in the lower-right corner of your workspace.
5. Type in its descriptive (white) area:

   Control Vehicle Radiator

temperature and pressure
via an Operator Interface.

6. Size your Note Pad to be as small as possible.
7. Copy the waveform display to your clipboard.
8. Open the Vehicle Radiator UserObject; expand the UserObject to receive the display.
9. Paste the Vehicle Radiator display and center it between the Temp and Pressure output pins as shown in Figure 8.7.

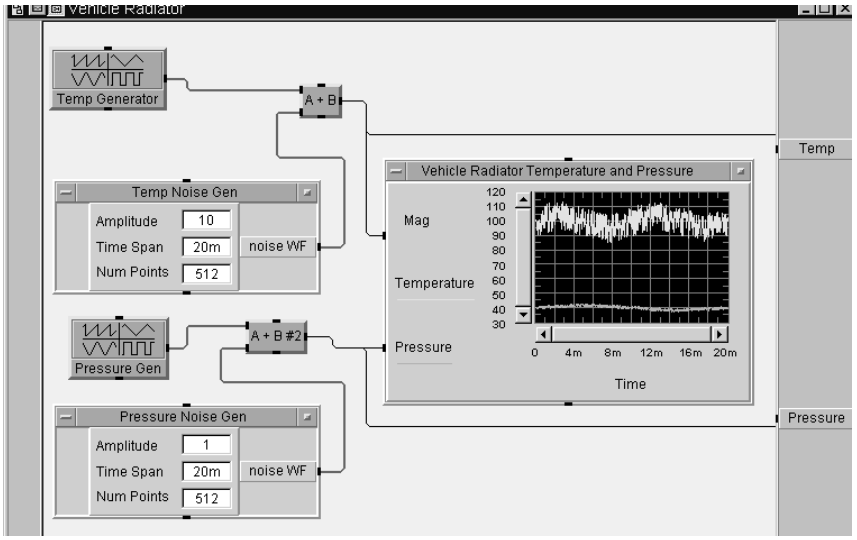


Figure 8.7. The modified Vehicle Radiator UserObject

10. Connect the Vehicle Radiator display input pins as shown in Figure 8.7 above.
11. Save this program as LAB8-4 again.

◊ 12. Run this program several times with different amplitude values to be certain all portions are displaying properly.

13. Close the Vehicle Radiator User Object via the upper-right "-"; run this program. See Figure 8.8.

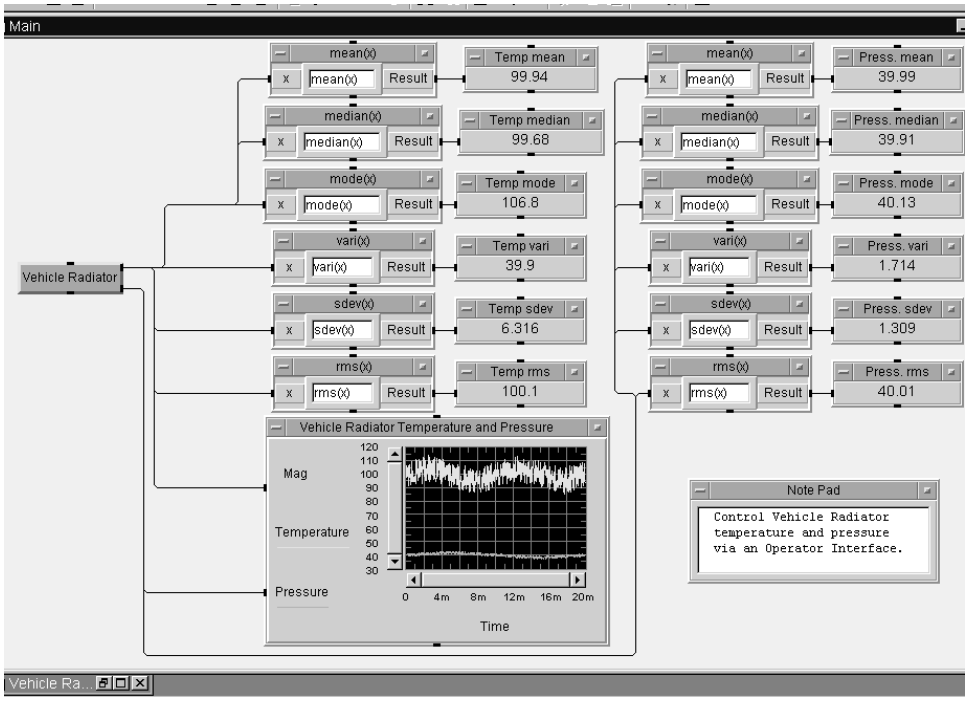


Figure 8.8. Vehicle Radiator Temperature and Pressure parameters displayed

Creating a Vehicle Radiator operator interface

14. Open the Vehicle Radiator UserObject; depress and hold down Ctrl; click on
    Temp Noise Generator
    Pressure Noise Generator
    Vehicle Radiator Temperature and Pressure (the display)
15. Select Menu Bar => Edit => Add to Panel; the selected objects will appear in a Panel View; move these icons and size the interface box to appear as shown in Figure 8.9.

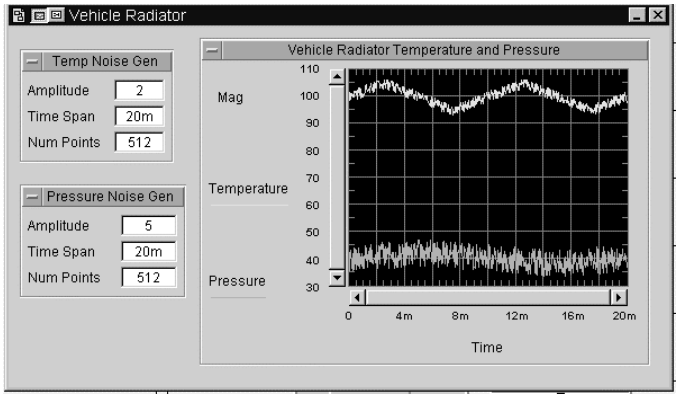


Figure 8.9. Vehicle Radiator operator interface

16. Save Lab 8-4 again; run this program to verify the displays in Figure 8.9.
17. Close the Vehicle Radiator UserObject via the upper-right "dash".
18. Right-click anywhere on the Vehicle Radiator UserObject; select Restore.
19. Hold down the Ctrl key; then highlight, from Main, the following icons:

Temp mean	Press. mean
Temp median	Press. median
Temp mode	Press. mode
Temp vari	Press. vari
Temp sdev	Press. sdev
Temp rms	Press. rms
Note Pad	

20. Click on the "Vehicle Radiator" title bar; remove the check from "Show Title Bar"; then click OK.
21. Select Menu Bar => Edit => Add to Panel; arrange the icons as shown in Figure 8.10.

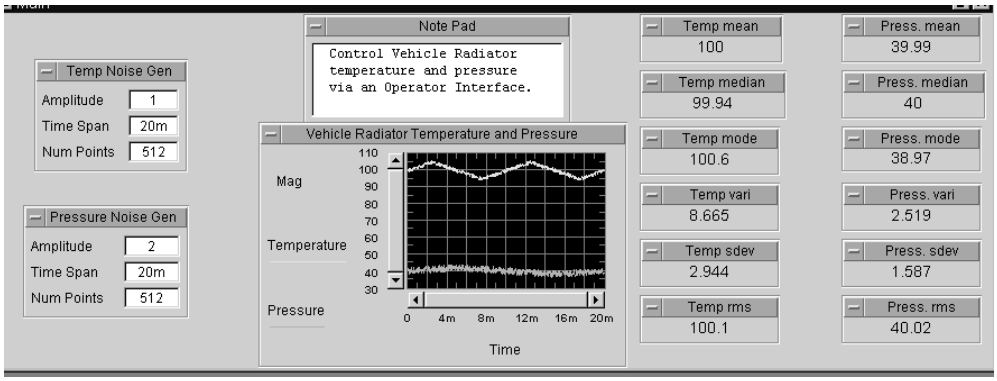


Figure 8.10. The Vehicle Radiator operator interface displaying statistics

22. Save this program again as Lab 8-4.

Securing an operator interface

23. Select Menu Bar => File => Save Secured Run Time Version …
24. Select Menu Bar => File => Open; return to the Secured Run Time Version by first selecting LAB8-4.vee and changing the .vee extension to .vxe.
    Note: The Main title bar now will read: "Main (Secured).
◊ 25. Change the noise amplitudes by factors of 5, 10, or more; run the program several more times. What changes in the statistics have you observed? How do these changes compare with the waveform noise distortions?

Your notes:

_____

_____

_____

_____

◊ 26. For a given set of noise amplitudes, change the Num Points by factors of two, starting with values of 8 for both noise generators.
◊ 27. Increase the Num Point values by factors of two; observe the effect on the statistical values.

Your notes:

_____

_____

_____

_____

28. Exit LAB8-4.vxe.

Lesson 8 Summary

This lesson examined how to work with data sets, databases, and operator interfaces.

Lab 8.1 showed you how to work with data sets, which is an array of records stored in a file.

Lab 8.2 showed you how to customize a simple database that searched and sorted data.

Lab 8.3 showed you how to create an operator interface for search operations; you also provided a test menu that allowed you to select a particular test from which all related test data became available.

Lab 8.4 showed you how to build a Vehicle Radiator operator interface that allowed you to compute, display, and store statistics for a variety of vehicle radiator test runs. This lab also showed you how to create and apply a Secured Run Time Version of an operator interface.

You are now ready to learn to create spreadsheets and reports.

Lesson 9
Creating Spreadsheets and Reports

This lesson will examine how to create spreadsheets and reports. It consists of a pre-lab and four labs.

Lesson 9 Pre-lab Summary

The following are described in Lesson 9 Pre-lab. See Appendix E, page E-59.

- VEE Pro 6: ActiveX References
- Microsoft Excel™ Column and Row Notation
- Microsoft Excel™ Cell Access
- The ActiveX Object Variable
- ActiveX Object Discovery
- Declare Variable

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 9.1 – Sending VEE Pro Data to an Excel™ Spreadsheet via Globals
This lab will show you how to transfer VEE Pro data to Microsoft Excel™. Declared global variables and function calls are also introduced so you can learn to generate spreadsheets.

Lab 9.2 – Creating a VEE Pro to Excel™ Template
This lab will enable you to create a generic template. It will also allow you to store test data as arrays and to modify the template to fill cells with data in whatever format you may specify.

Lab 9.3 – Using Microsoft Word™ to Prepare VEE Pro Reports
This lab will show you how to send text, a screen-dump of a VEE Pro pop-up panel with an XY display, and a time stamp to a Microsoft Word™ document.

Lab 9.4 – Using VEE Pro to Prepare and Directly Print Reports in Microsoft Word™
This lab will show you how to load data into Microsoft Word™ and automatically print the resulting document.

This lesson will focus upon using ActiveX to access Microsoft Excel™ and Word™ in the preparation of spreadsheets and reports. VEE Pro allows you to document your test data and save it both electronically and as a hard copy. Excel™ and Word™, accessed via ActiveX, are so flexible that you are only limited in its applications to your everyday test documentation by the amount of time that you can devote to developing new and meaningful presentations. It is imperative that you consider the potential reader of your reports and spreadsheets so that you can prepare them in a standard format from your tests.

Lab 9.1 – Sending VEE Pro Data to an Excel™ Spreadsheet via Globals

This lab will show you how to send (transfer) VEE Pro data to Microsoft Excel™. Virtual test data will be generated. Declared global variables and function calls are also introduced so you can learn to generate spreadsheets.

Open your VEE Pro program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Sending VEE Pro data to an Excel™ spreadsheet via ActiveX

2. Select Menu Bar => Device => ActiveX Automation References…. Figure 9.1 will appear.

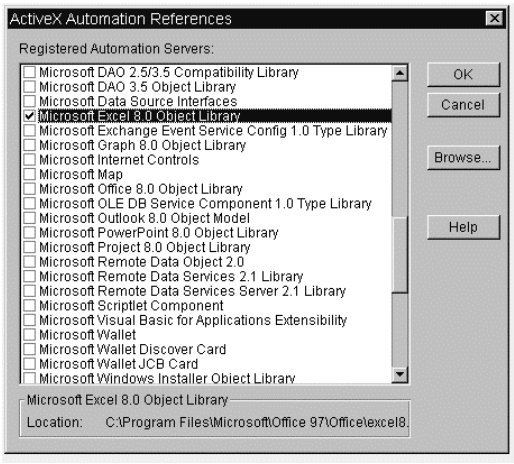


Figure 9.1. The ActiveX Automation References box

3. Select Microsoft Excel™ Object Library; click OK.
   Note 1: This approach may not work with older Windows programs; consult their manual.
   Note 2: This assumes that you have a version of Microsoft Excel™ installed.
4. Select Start on your computer; (lower left-hand corner); run Microsoft Excel™.
   Note 1: At the bottom of your screen, in the Windows Tool Bar, you should now have a long, rectangular icon labeled Microsoft Excel™; return to VEE Pro by selecting the icon: VEE Pro.

Note 2: Microsoft Excel™ will automatically start via ActiveX Automation when the Vehicle Radiator program is run.

5. Select Menu Bar => Device => UserFunction; rename it globals; click OK.
6. Select Menu Bar => Data => Variable => Declare Variable; place it in the top-left corner inside the globals UserFunction.
7. Change the white-space Name to sheet; change Type to Object.
   Note 1: When you click outside this, or any other Declare Variable object, its title-bar name is automatically converted; the word "Variable" becomes the name that you typed into the Name window.
   Note 2: The "Specify Object Type" area can be used select the exact type of Object (such as Excel sheet versus an Excel cell) for error checking. It can also be used to create event callback function to "catch" an event from Excel.
8. Clone this object three times; change the variable names to: app, range, and window.
   Note: Declare Variable allows you to specify the Object Type and Class. This action allows for more specific type checking.
9. Convert all four of them to icons; place them as shown in Figure 9.2; size the globals UserFunction.

Figure 9.2. The globals UserFunction

10. Size the Main window so you can place the globals UserFunction below the Main window; you can now see both on the screen.
11. Save As… this partial program as LAB9-1.
    Note: At the bottom of your screen you should now have two long, rectangular icons labeled VEE Pro – LAB9-1; the other labeled Microsoft Excel™.
12. Select Menu Bar => Device => Call; an object entitled "Call Function" will appear in the Main window; place it in the top-left corner of Main.
    Note: Call globals is technically not required; the Declare Variable objects will activate globals at pre-run.
13. Change its "myFunction" name (in the white space) to globals; click outside the object; its Title Bar will automatically change to Call globals.
    Note: Local User Functions will appear in the Program Explorer Work Area.
14. Convert Call globals to an icon.
    Note: Events allow you to catch, via your VEE Pro UserFunction, events that could occur in an application, such as "right-button-down" in an Excel™ worksheet.
15. Click Device => Formula; place it in the upper center of the Main window.
16. Rename its icon: Set up Excel Worksheet.
17. Connect the Call globals sequence-out (bottom) pin to the "Set up Excel Worksheet" sequence-in (top) pin.
18. Enter, inside "Set up Excel Worksheet", the formula lines as follows; see Figure 9.3.

```
set sheet =CreateObject("Excel.Sheet").worksheets(1);
```

```
set app =sheet.application;
app.visible =1;
set window =app.windows(1);
window.caption =“Test System Results”
```

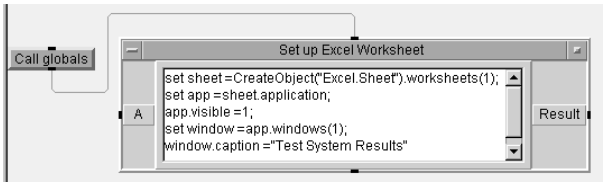


Figure 9.3. Setting up the Microsoft Excel™ worksheet

Note 1: Semicolons are used for expression separators. One expression per line makes the program easier to understand.

Note 2: Properties usually refer to some type of data you either set or get, such as

- get properties via object.property
- set properties via object.property=MaxSize
- call a method via Object.method(parameters)
- get the application property of the sheet object via sheet.application

A method is just a property with parameters. In many cases, a property refers to the attribute of the object (a noun); a method performs an action an the object (a verb). Another difference is: a property (get or set) is very fast to execute; a method requires some time to execute. See Appendix E, page E-xx. Choosing descriptive names for your variables is important.

Note 3: Set is Microsoft syntax used to assign (or set) whatever is on the right-hand side of the assignment operator = to the variable on the left-hand side of the expression.

Example: You have previously declared the variable app as an Object type. Thus you can “set” app to the value on the right-hand side of the expression.

Note 4: CreateObject(“Excel.Sheet”) is located in the Function & Object Browser as a Member of the Built-in Functions Type and the ActiveX Automation Category.

Note 5: GetObject( ) is used to get some data that already exists in a running Excel™ or to load a file into a running Excel™. (A useful reference is the Microsoft™ Office Visual Basic Programmer’s Guide.)

19. Delete the Set up Excel™ Worksheet input pin and output pin.
20. Convert Set up Excel™ Worksheet to an icon.
21. Select Menu Bar => Device => Formula twice.
22. Select Menu Bar => Flow => Repeat => For Range once. Change the For Range object white space as follows:

    From: 1 Thru: 20 Step: 1
23. Rename the Formula objects, connect them, and configure them as shown in Figure 9.4; be certain to delete the “Fill in Title” input and output pins.

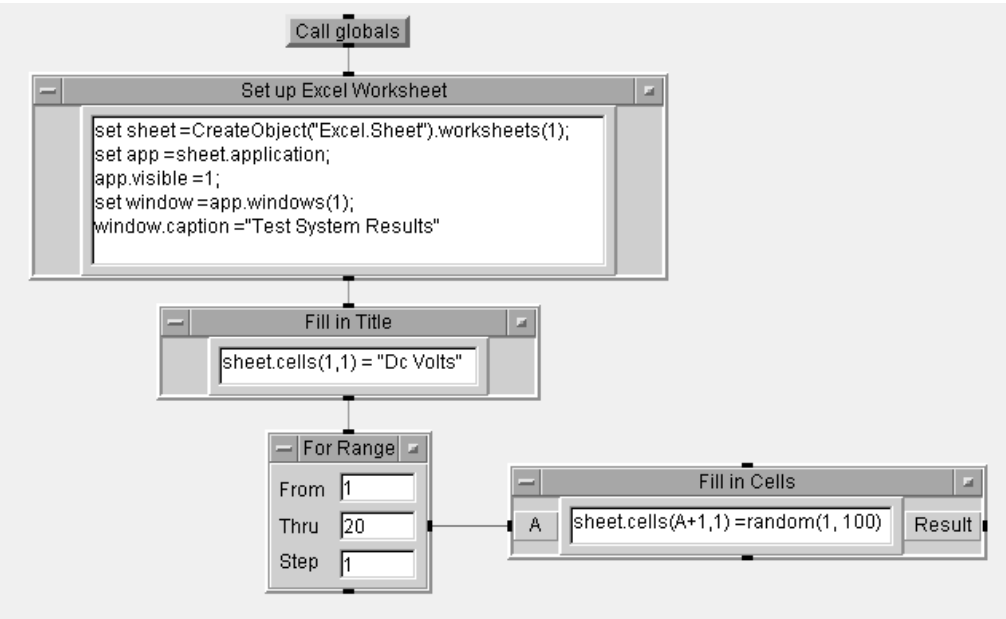


Figure 9.4. Worksheet title and data entries

Note 1: Sheet.cells(1,1) refers to the first row and first column in the Excel™ worksheet. The text "Dc Volts" will be placed there.

Note 2: For Fill in Cells, the variable sheet.cells(A+1,1) gets the row number by adding 1 to the input pin "A" value; it stays in column 1.

Note 3: The value between 1 and 100 returned by "random" is assigned to the specified cell in the worksheet.

Note 4: The For Range object generates the integers from 1 through 20; the "Fill in Cells" places the random number in the specified cell.

24. Get a Formula Object; delete its input terminal; change its formula entries (3) as indicated in Figure 9.5.
25. Get an AlphaNumeric object from the Menu Bar: Display menu; rename it Results Average.
26. Configure and connect all objects exactly as shown in Figure 9.5.

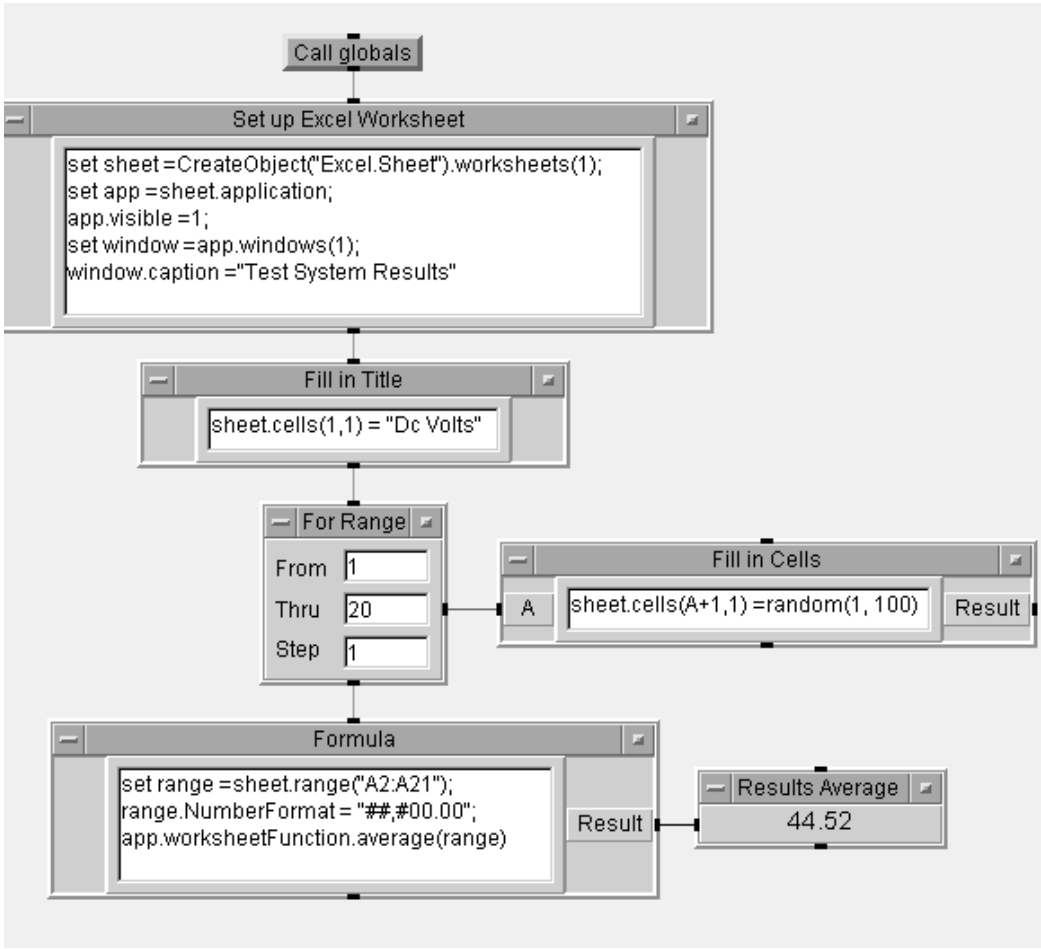


Figure 9.5. The complete Lab 9.1 program

Note 1: The statement set range =sheet.range("A2:A21") sets the VEE Pro variable range to reference the range A2 to A21 on the Excel™ worksheet; "A" refers to the first column in a worksheet.

Note 2: range.NumberFormat = "##,#00.00" assigns the format to each of those cells with the # signs allowing for larger numbers.

Note 3: app.worksheetFunction.average(range) calls an Excel™ function average( ) that returns the average value of the designated range of values; it is displayed in Results Average.

27. Save your program again as LAB9-1.
28. Open the Excel™ program.

    or

    Open the Excel™ program, if necessary, within Microsoft Windows.

Note: Excel™ may open automatically; run this program to determine if it will open automatically before going to the "Start" screen and finding Excel™.

29. Run your program; the Excel™ worksheet appear as noted in Figure 9.6.
    Note: Sheet1 will appear at the bottom of your new worksheet.

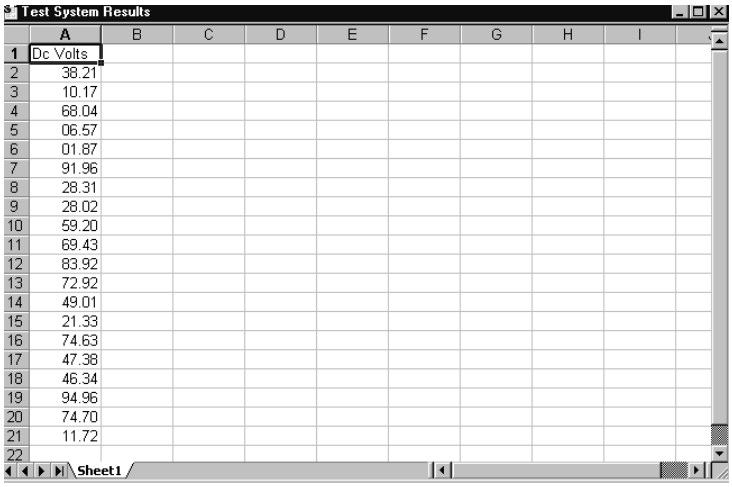


Figure 9.6. The generated Microsoft Excel™ worksheet

◊ 30. Re-run this program several times; switch between VEE Pro and Excel™; the Results Average displayed value will also change.

◊ 31. Change "For Range" to values less than 20; run this program change several times; observe the effect upon the amount of data taken and printed on the spreadsheet.

◊ 32. Change the Formula Object "range.NumberFormat" by increasing the number of zeros to the right of the decimal marker; run this program.

◊ 33 Change the Formula Object "range.NumberFormat" by increasing the number of zeros to at least eight; run this program; note the ability of the spreadsheet column to adapt to these additional decimal values.

34. Close your program without saving it.

Lab 9.2 – Creating a VEE Pro to Excel™ Template

This lab will enable you to create a more generic template keyed to your own work. It will also allow you to store your test data as arrays and to easily modify the template to fill cells with data in whatever format you may specify.

Open your VEE Pro program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Creating an VEE Pro to Excel™ template

2. Open LAB9-1; change the For Range object to loop ten times; Save As… LAB9-2 immediately.
3. Add an input B to Fill in Cells; alter the statement inside to read
   sheet.cells(A+1,1) = B[A-1]
4. Select => Menu Bar => Click Device => Formula; rename it Array of Test Data.
5. Enter the expression: randomize(ramp(10),4.5,5.5) in Array of Test Data.
   Note: This will create a random array of ten elements with values from 4.5 to 5.5.
6. Delete the input pin of Array of Test Data; change For Range to an icon.
7. Connect the data-output pin of Array of Test Data to the B input of Fill in Cells.
8. Change the range in the Formula box on the bottom of your screen from A21 to A11 which should result in the modified statement reading
   set range=sheet.range("A2:A11");
9. Save your program again as LAB9-2. Your program should look like Figure 9.7.

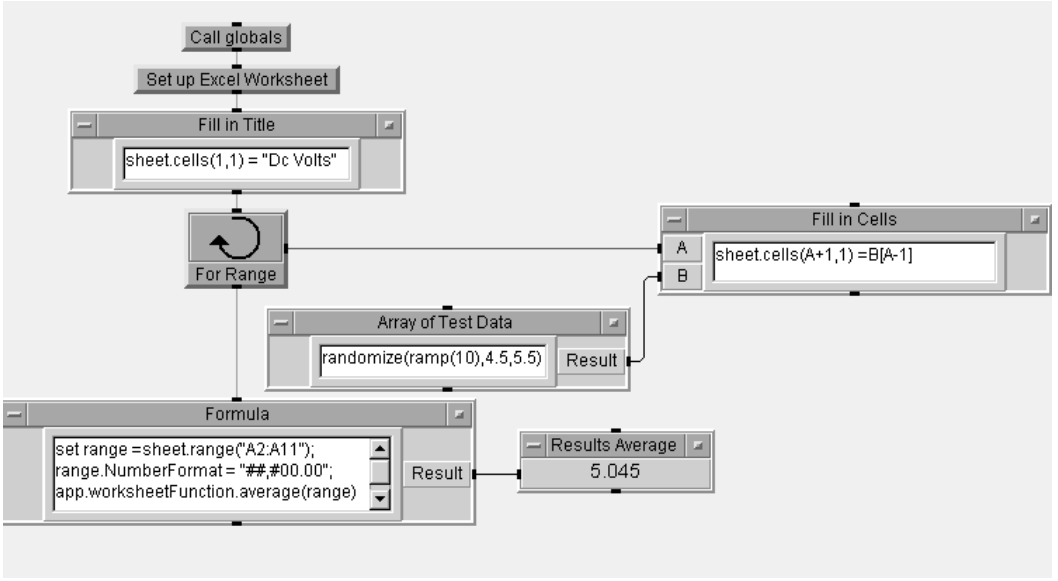


Figure 9.7. The complete Lab 9.2 program

10. Run this program.
    Note 1: Your Excel™ worksheet should look like Figure 9.8 but will display different values.

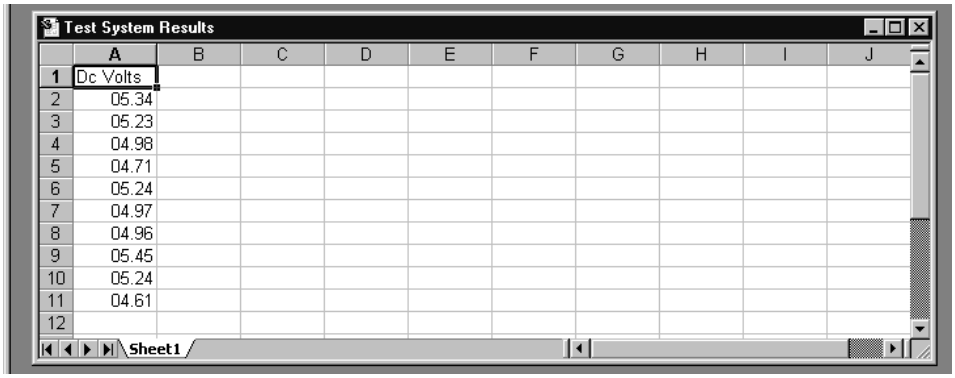


Figure 9.8. Microsoft Excel™ worksheet generated from Lab 9.2

Note 2: For additional methods and properties available in the Excel™ library, go to Menu Bar => Device => Function & Object Browser; Type: ActiveX Objects; Library: Excel. Choose the "Class" you desire to activate. Consult Microsoft documentation for more complete information on these libraries – their classes and their members.

11. Close this program without saving it again.

Lab 9.3 – Using Microsoft Word™ to Prepare VEE Pro Reports

This lab will show you how to send text, a screen-dump of a VEE Pro pop-up panel with an XY display, and a time stamp to a Microsoft Word™ document.

Open your VEE Pro program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main; Save As… LAB9-3 immediately.

Transferring VEE Pro data into a Microsoft Word™ document

2. Select Menu Bar => Click Device => ActiveX Automation References…; select Microsoft Word™ Object Library, Microsoft Office Object Library, and Microsoft Internet Controls; click OK. (The revision numbers will depend upon the version of Microsoft Office™ you have installed.) See Figure 9.9.

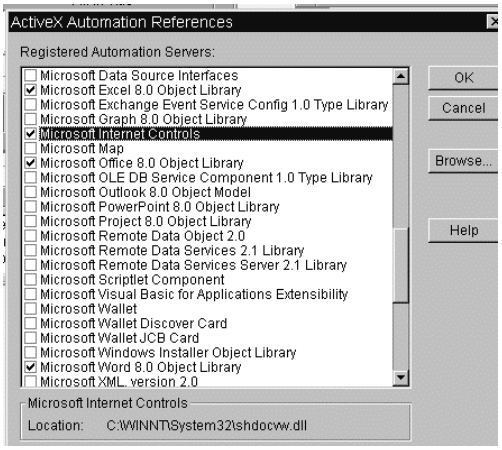


Figure 9.9. The ActiveX Automation References box

3. Select Menu Bar => Click Data => Variable => Declare Variable; change the Type field to Object.
4. Clone Declare Variable four times; name the five Declare Object variables in their "Name" edit field:
   App
   Doc
   Wnd
   Sel
   Bmp

   Note: The Title Bar label will change automatically.
5. Click each Specify Object Type, then click its Edit… button; select the appropriate library and class as follows:
   for App – Library: Word; Class: Application; check the "Enable Events" box
   for Doc – Library: Word; Class: Document; check the "Enable Events" box
   for Wnd – Library: Word; Class: Window
   for Sel – Library: Word; Class: Selection
   for Bmp – Library: Word; Class: Shape

   Note: You cannot access the Enable Events box for the last three objects.
6. See Figure 9.10 to verify your selections.

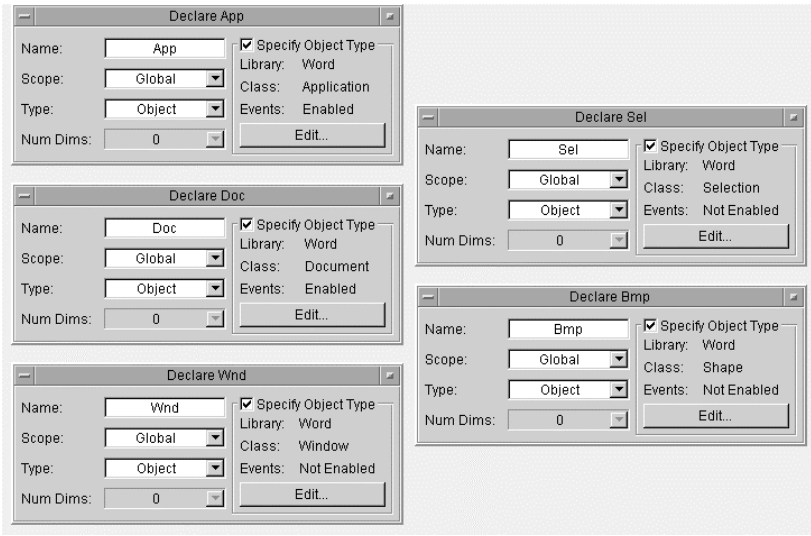


Figure 9.10. The five Declare Variable objects

7. Convert these five Declare Variable objects to icons; place them in a column in the lower-right corner of Main.
8. Select Menu Bar => Device => UserFunction; name it Graph.
9. Select Menu Bar => Device => Virtual Source => Function Generator; place it on the left-center side of Graph.
10. Select Menu Bar => Device => Waveform (Time); place it to the right of Function Generator.
11. Connect the Function Generator to Waveform (Time) Trace1.
12. Highlight Waveform (Time); place the mouse pointer over the window background of Graph; click on the mouse right-hand button.
13. Click Add to Panel; Waveform (Time) will appear in a Panel View.
14. Select Menu Bar => View => Program Explorer; double-click on Main; the Main window will appear.
    Note: An alternative would be to click on the upper-right "_" symbol of Graph so it will appear in the lower-left corner of the screen under Main as an icon.
15. Select Menu Bar => Device => Call; place Call Function in the upper-right corner of Main, click OK.
16. In the Call Graph Function "Name" field, change myFunction to Graph; convert it to an icon. (Its Title Bar will automatically change to Graph.)
17. Select Menu Bar => Device => Formula; place it in the upper-left corner of Main; change its name to **ImageFileName**; delete its input terminal.
18. Change its Formula expression field to: installDir()+"panel.bmp" with no spaces within the quotes.
19. Select Menu Bar => Device => Formula; place it to the right of ImageFileName and under Call Graph.
20. Change the name of Formula to **savePanelImage**; change its formula to SavePanelImage("Graph",fileName,256)

and change its input A pin to read **fileName** with no spaces within the quotes to match the variable name within the expression.

Note 1: Your saved picture will have a depth of 256 colors per pixel.

Note 2: The savePanelImage() function saves the specified UserFunction panel image in either a Windows Bitmap or JPEG format.

21. Select Menu Bar => Device =>Formula; place it below **savePanelImage**; change its formula to:
    Set App = CreateObject("Word.Application")
    and delete its input terminal.
22. Connect these four objects as shown in Figure 9.11; save this program as LAB9-3.

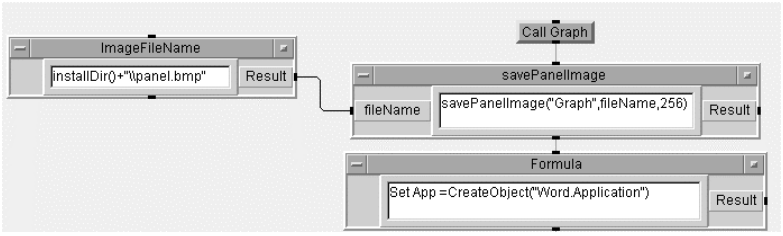


Figure 9.11. Connections for first portion of Lab 9.3

23. Select Menu Bar => Display => Note Pad; enter the following description
    This program is designed to
    stop when Word with the Graph
    appears so you can add your
    text to the Word memo.
    Size the Note Pad around the description.
24. Select Menu Bar => Device =>Formula; change its input-pin name to fileName; delete its output terminal.
25. Enter the formula statements shown in the box of Figure 9.12 below; connect the output and sequence pins as shown.
26. Save your program again

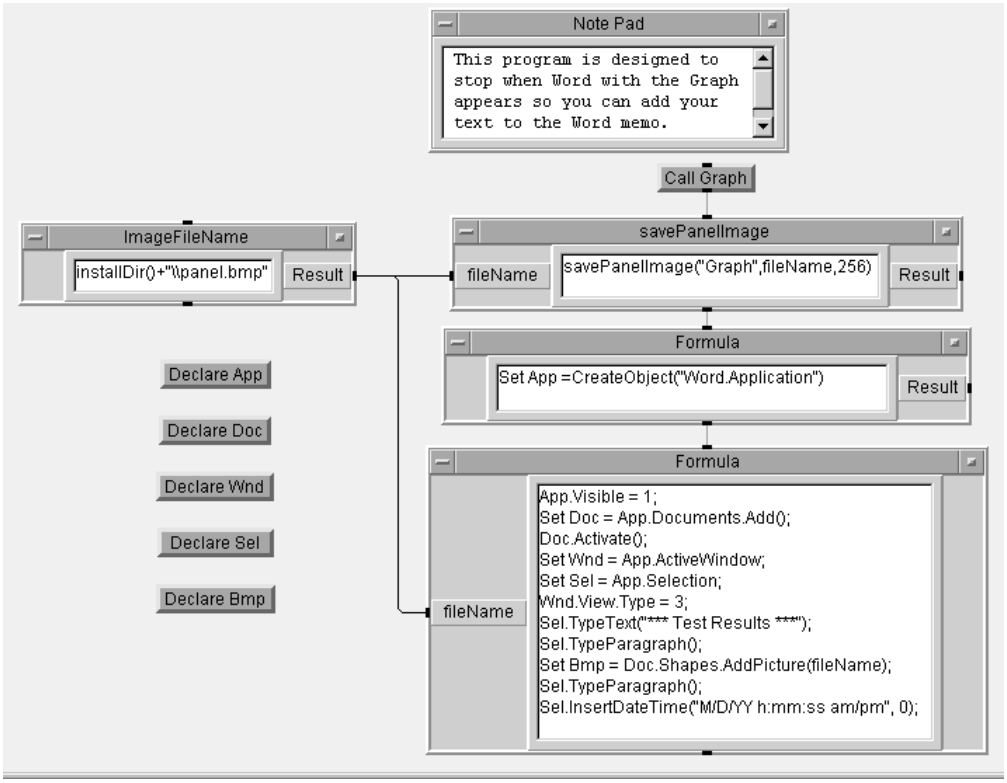


Figure 9.12. The complete Lab 9.3 program

27. Run this program; move the graph downward with the "down-arrow" key so the "Test Results" and date appear above the graph; add whatever statements you prefer. An example is shown in Figure 9.13.

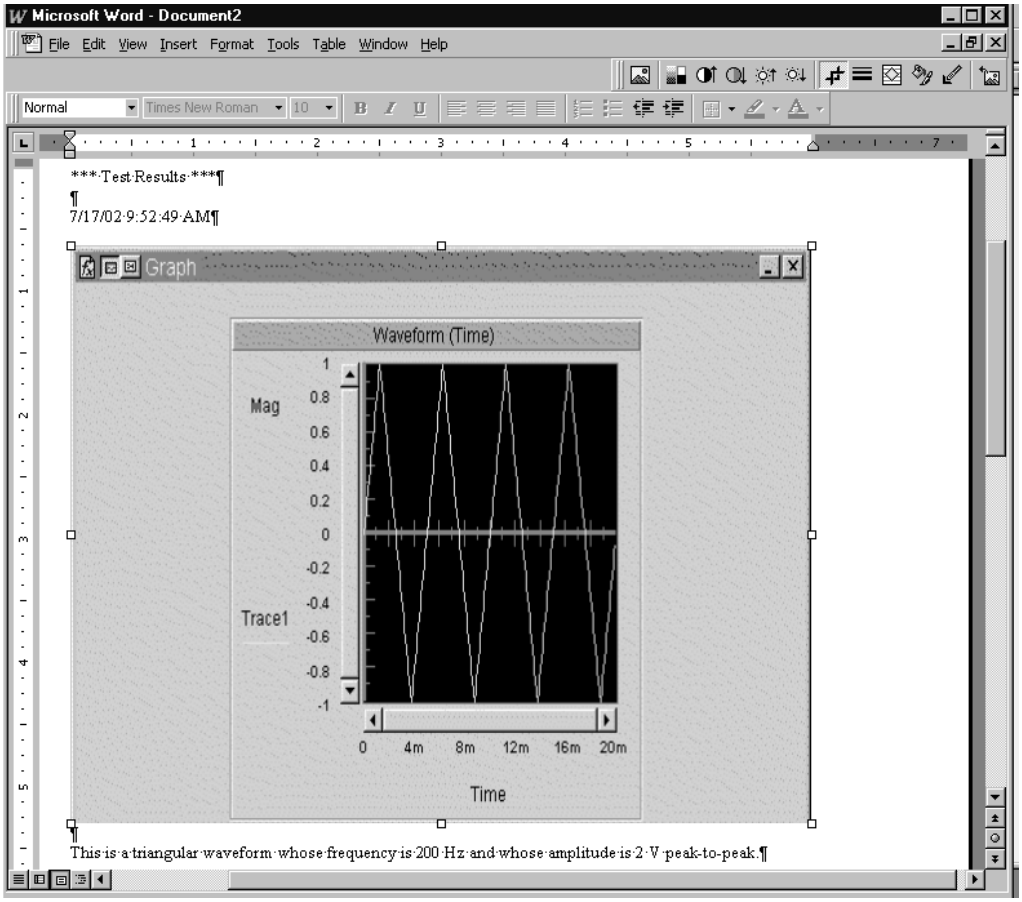


Figure 9.13. A Microsoft Word™ document containing graph and an explanatory statement

◊ 28. Crop your picture so only the Waveform (Time) oscilloscope is displayed by performing the following internal steps:
a. Click on the object to select it.
Select Menu Bar => View => Tool Bars => Customize => Drawing; near the bottom is "Crop"; click on it; close the Customize window.
Use the "Crop" icon to grab the middle object handles; remove ("trim") that portion of the picture that is not the oscilloscope.

◊ 29. Size the remaining picture by grabbing the corner object handles; a double-ended arrow will appear; increase or decrease its size by moving in the direction of the arrow.

◊ 30. Move the location of the picture object using the keyboard "arrow" keys to a location of your preference; note the effect on the location of the text lines.

31. Print your final document if you so desire.

32. Close both the Word™ and the VEE Pro documents without saving them.

Lab 9.4 – Using VEE Pro to Prepare and Directly Print Reports in Microsoft Word™

This lab will show you how to load data into Microsoft Word™ and automatically print the resulting document.

Printing reports in Microsoft Word™

1. Open LAB9-3; immediately Save As … LAB9-4.
2. Change the Note Pad to read:
   This program is designed to immediately print the
   Test Results and its Graph in Word.
3. Move the Note Pad so it is beside and to the right of the largest Formula Object.
4. Select Menu Bar => Device => Formula; place it below and to the left of the largest Formula Object.
5. Select Menu Bar => Flow => If/Then/Else; place it to the right of the Formula Object in step 4; change its Properties box to read: Is Printer Configured?; change its input pin label from A to **str**.
6. Select Menu Bar => Device => Formula; place it to the left of Formula in step 4.
7. Select Menu Bar => Device => Formula; place it to the right of If/Then/Else.
8. Connect the four objects as shown in Figure 9.14; change the formula expressions as shown in Figure 9.14; you must delete terminals on some of the Formula objects.

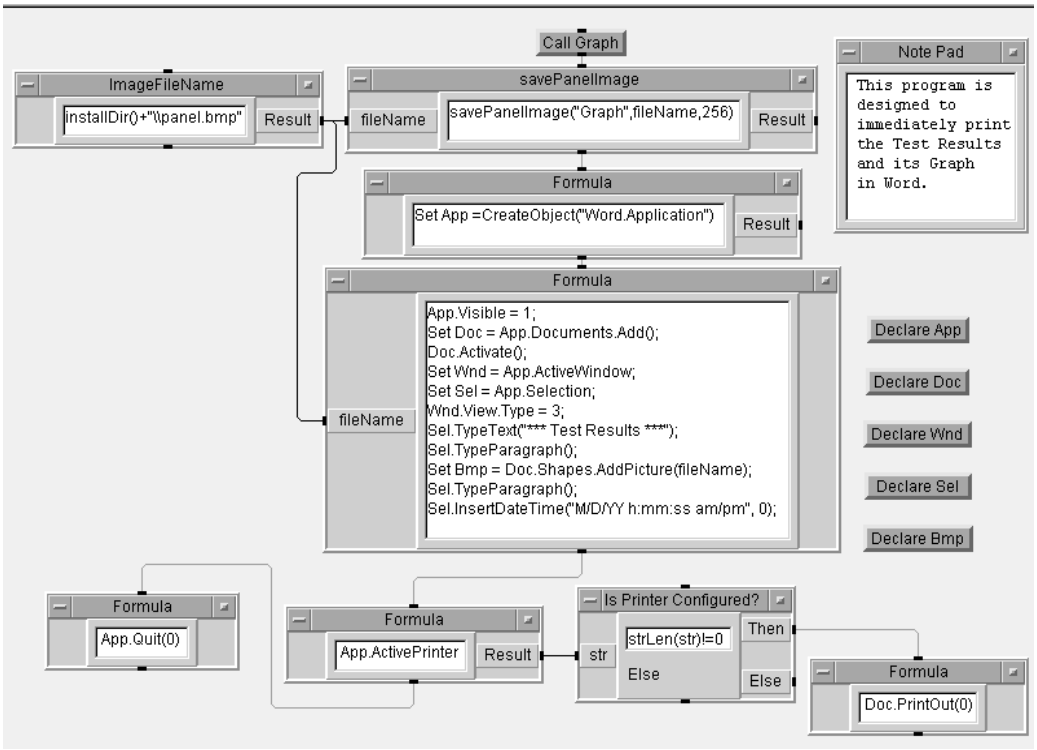


Figure 9.14. The complete Lab 9.4 program

9. Save this program again.
10. Run this program; the results should automatically print and the Word™ document will not remain on the screen.
    Note: It may be necessary to key your computer to select the desired printer and its configuration.
11. Cut the left-most Formula Object; re-run this program; the Word™ document will remain on the screen for you to modify the document as you so desire.
12. Save this program as LAB9-4a.
    Note: When closing LAB9-4a, you will be prompted: Do you want to save changes… because of the addition of the Word™ document during running. This provides you with the option of saving the Word™ document with a name of your preference.

Lesson 9 Summary

This lesson addressed using ActiveX to access Microsoft Excel™ in the preparation of spreadsheets. (VEE Pro allows you to document your test data and save it both electronically and as a hard copy.)
Excel™, accessed via ActiveX, is so flexible that you are limited in its applications to your everyday test documentation only by the amount of time that you can devote to developing new and meaningful presentations. It is imperative that you consider the potential reader of your reports and spreadsheets so that you can prepare them in a standard format.
Your time is precious; so is the time of your co-workers and managers.

Lab 9.1 showed you how to transfer VEE Pro data to Microsoft Excel™. Declared global variables and function calls are also introduced so you can learn to generate spreadsheets.

Lab 9.2 showed you how to create a generic template. It also allowed you to store test data as arrays and to modify the template to fill cells with data in any specified format.

Lab 9.3 showed you how to send text, a screen-dump of a VEE Pro pop-up panel with an XY display, and a time stamp to a Microsoft Word™ document.

Lab 9.4 showed you how to load data into Microsoft Word™ and automatically print the resulting document.

You are now ready to use VEE Pro to communicate more sophisticated data to Excel™. Also, see – under VEE Pro Help; Contents and Index – the Guide to Agilent VEE Example Programs. The examples are a subset under Microsoft Excel™. There are a variety of preprogrammed Excel™ spreadsheets.

Lesson 10
Practicing Monitoring via the Vehicle Radiator

This lesson will use the Vehicle Radiator to expand a test database, automatically record desired data onto a spreadsheet and move the results to Word™. It consists of a pre-lab and four labs.

Lesson 10 Pre-lab Summary

The following are described in the Lesson 10 Pre-lab. See Appendix E, page E-61.

- The Dynamic I/O Automation Server
- ActiveX Automation References
- Dialog Information

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items, and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 10.1 Expanding a Vehicle Radiator Test Database
This lab will show you how to monitor temperature and pressure values while simultaneously computing mean, median, mode, variance, standard deviation, and rms values of both Vehicle Radiator parameters.

Lab 10.2 – Preparing a Vehicle Radiator Three-Column Spreadsheet
This lab will show you how to send (transfer) VEE data from your Vehicle Radiator program to Microsoft Excel™; record parameter identification in one column; and record temperature and pressure readings in two other columns.

Lab 10.3 – Using Excel™ to Document Six Sequential Vehicle Radiator Tests
This lab will show you how to send six sets of sequential tests from your Vehicle Radiator program to a single Excel™ spreadsheet.

Lab 10.4 – Moving Vehicle Radiator Information from Excel™ to Word™
This lab will show you how to copy and paste a VEE Pro/ Excel™ spreadsheet with a date & time stamp, send a graph of the temperature and pressure waveform, and add text to a Microsoft Word™ document.

Note: Review draft of individual student projects as noted in To the Instructor (page xv).

Lab 10.1 – Expanding a Vehicle Radiator Test Database

This lab will show you how to monitor temperature and pressure values while simultaneously computing mean, median, mode, variance, standard deviation, and rms values of both Vehicle Radiator parameters.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Determining maximum and minimum test values automatically

2. Open LAB6-4; Save As… LAB 10-1 immediately.
3. Change the Temperature object title to Temp Formulas.
4. Add extra output terminals D, E, F, and G to Temp Formulas.
5. Change the Temp Formulas expression to read:

   ```
   B=mean(A);
   C=median(A);
   D=mode(A);
   E=vari(A);
   F=sdev(A);
   G=rms(A)
   ```

   Note: Remember to enter semicolons (;) at the end of each line.
   The semicolon is optional for the last line.
6. Clone Temp sdev five times; change all Properties descriptions to agree with Figure 10.1.
   Note: You may need to move the entire program to the left so you can place the cloned objects of step 5 to the right. This movement is performed by clicking on the Work Area and moving the entire Work Area to the left.

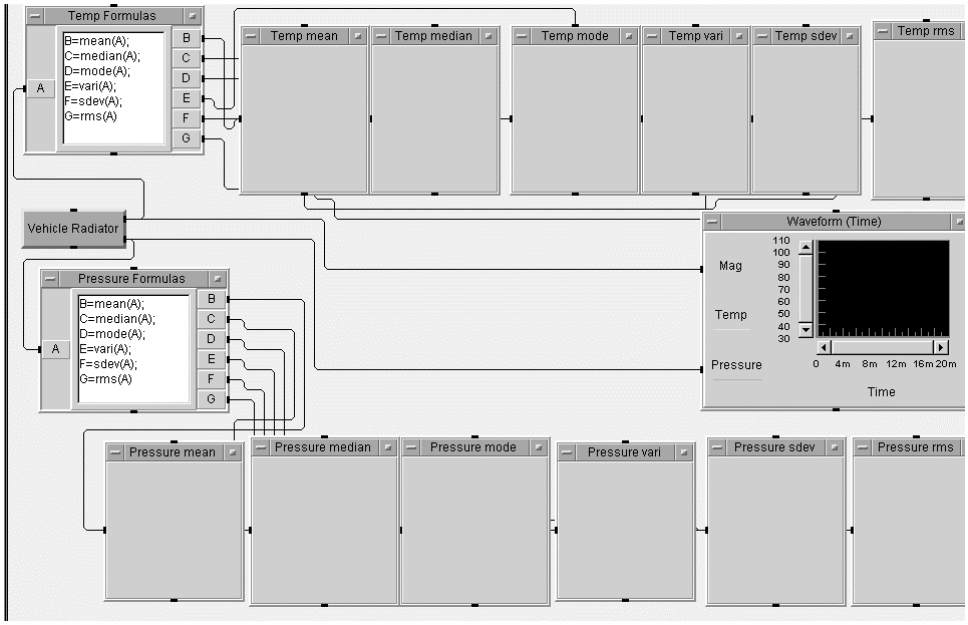


Figure 10.1. The Lab 10.1 program layout

7. Connect the six AlphaNumeric displays to their appropriate Temp Formulas output pins.
8. Delete the Pressure object.
9. Clone Temp Formulas; place it below Vehicle Radiator; change its name to Pressure Formulas.
10. Connect Pressure Formulas to the lower output pin of Vehicle Radiator.
11. Delete one Pressure AlphaNumeric display; clone the remaining object five times.
12. Label these six displays as shown in Figure 10.1; connect them to their appropriate Pressure Formulas output pins.
13. Shrink Waveform (Time) to fit between the displays as shown in Figure 10.1.
14. Save your program again.

◊ 15. Run your program several times; it should look like Figure 10.2.

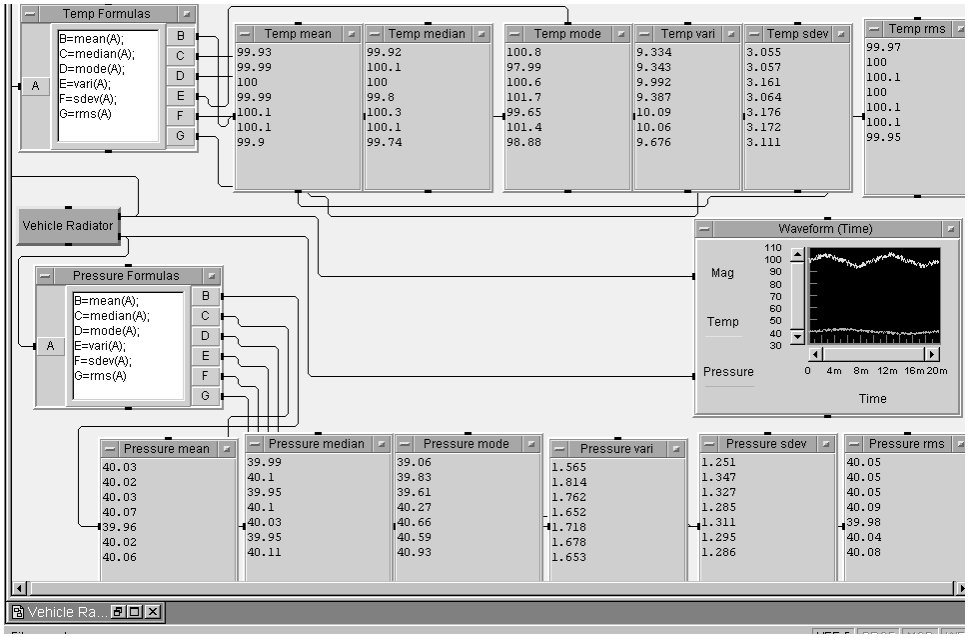


Figure 10.2. The Lab 10.1 program after several runs

16. Open the Vehicle Radiator UserObject.

◊ 17. Run your program several times with different Temp Noise Gen amplitude values and Pressure Noise Gen amplitude values by a factor of at least five; return the Vehicle Radiator UserObject to an icon; examine the change in values of the statistical parameters.

18. Close your program without saving it; open LAB10-1 again.

    Note: This will clear all previous runs and Vehicle Radiator noise settings and test runs. This is a valuable fact that will be useful to you as you change parameters that you do not need to compare with previous settings. On the other hand, changing amplitude and other values and then running the program again without closing allows you to compare values that you recorded with previous settings.

Lab 10.2 – Preparing a Vehicle Radiator Three-Column Spreadsheet

This lab will show you how to send (transfer) VEE data from your Vehicle Radiator program to Microsoft Excel™ via globals, to record parameter identification in one column, and temperature and pressure readings in two other columns that will match the parameter-identification column.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Preparing and transferring data to a multi-column Excel™ spreadsheet

2. Open LAB10-1; Save As… LAB10-2 immediately.
3. Cut all six temperature alphanumeric displays; cut all six pressure alphanumeric displays.
4. Select Menu Bar = > Device => UserFunction; change its name to globals.
5. Open Lab9-2 with Lab10-2 remaining open; copy the content of globals, which is the four Declare Variables icons (Declare sheet, Declare app, Declare range, Declare window); paste these four icons into the Lab10-2 globals icon; close Lab9-2.
6. Size the UserFunction globals; click on the "_" button to place it under the Main window; it will appear next to Vehicle Ra… which was minimized in LAB10-1.
7. Select Menu Bar => Device => Formula; change its name to Set up Excel Worksheet; delete its input and output pins.
8. Type the following formula into the Set up Excel Worksheet expression:

```
set sheet =CreateObject("Excel.Sheet").worksheets(1);
set app = sheet.application;
app.visible =1;
set window =app.windows(1);
window.caption = "Vehicle Radiator Test Results"
```

9. Select Menu Bar => Device => Formula; change its name to Fill in Parameters and delete is input and output pins.
10. Type the following formula into the Fill in Parameters expression:

```
sheet.cells(1,1) ="Statistic";
sheet.cells(2,1) ="mean";
sheet.cells(3,1) ="median";
sheet.cells(4,1) ="mode";
sheet.cells(5,1) ="vari";
sheet.cells(6,1) ="sdev";
sheet.cells(7,1)="rms";
sheet.cells(8,1)="date";
sheet.cells(9,1)="time"
```

11. Select Menu Bar => Device => Call; change the white-space name to Globals; connect the bottom (sequence) pin of "Call globals" to the top (sequence) pin of "Set up Excel Worksheet".
12. Connect the sequence output (bottom) pin of "Set up Excel Worksheet" to the sequence input (top) pin of "Fill in Parameters".
13. Select Menu Bar => Device => Formula; change its name to Fill in Cells; Temp.
14. Add eight Data Input terminals to Fill in Cells; Temp; change the input terminal names as follows: mean, median, mode, vari, sdev, rms, date, and time.
15. Insert the formula white space to read:

```
sheet.cells(1,2)="Temp'ture";
sheet.cells(2,2)=mean;
sheet.cells(3,2)=median;
sheet.cells(4,2)=mode;
sheet.cells(5,2)=vari;
sheet.cells(6,2)=sdev;
sheet.cells(7,2)=rms;
sheet.cells(8,2)=date;
sheet.cells(9,2)=time
```

16. Clone “Fill in Cells; Temp”; change its icon name to “Fill in Cells; Pressure”; delete the two input terminals “date” and “time”; change the white-space formula to read as follows:
    sheet.cells(1,3)="Pressure";
    sheet.cells(2,3)=mean;
    sheet.cells(3,3)=median;
    sheet.cells(4,3)=mode;
    sheet.cells(5,3)=vari;
    sheet.cells(6,3)=sdev;
    sheet.cells(7,3)=rms
    Note: The only changes, besides removing “date” and “time”, are to convert all the cell address zeros to ones.
17. Connect the six Temp Formulas output pins to the six appropriate Fill in Cells; Temp input pins.
18. Connect the six Pressure Formulas output pins to the six appropriate Fill in Cells; Pressure input pins.
19. Select Menu Bar => I/O => To => String; place this icon in the lower-left corner of Main.
20. Clone To String; place it to the right of the previous icon.
21. In the left To String transaction edit field (white space), double-click as instructed; change “a” to “now()”.
22. Change Default Format to Time Stamp Format; change the Default Field Width to Field Width: enter 24.
23. Change Date & Time to Date; DD/Month/YYYY will appear; click OK.
24. In the right To String transaction edit field; double-click; change “a” to “now()”.
25. Change Default Format to Time Stamp Format; change Date & Time to Time; HH:MM:SS will appear; leave EOL on.
26. Connect the Date To String “result” to Fill in Cells; Temp “date” input.
27. Connect the Time To String “result” to Fill in Cells; Temp “time” input.
    Note: It is not necessary to open the Fill in Cells; Temp icon as the names “date” and “time’ will appear when the pin connection is about to be made.
28. Select Menu Bar => Display => Note Pad; change its title to Document Vehicle Radiator Statistics and insert the following in its white space:
    Monitor and document vehicle radiator temperature and pressure on a three-column Excel™ spreadsheet.
29. Save your program again; your completed program should look like Figure 10.3.

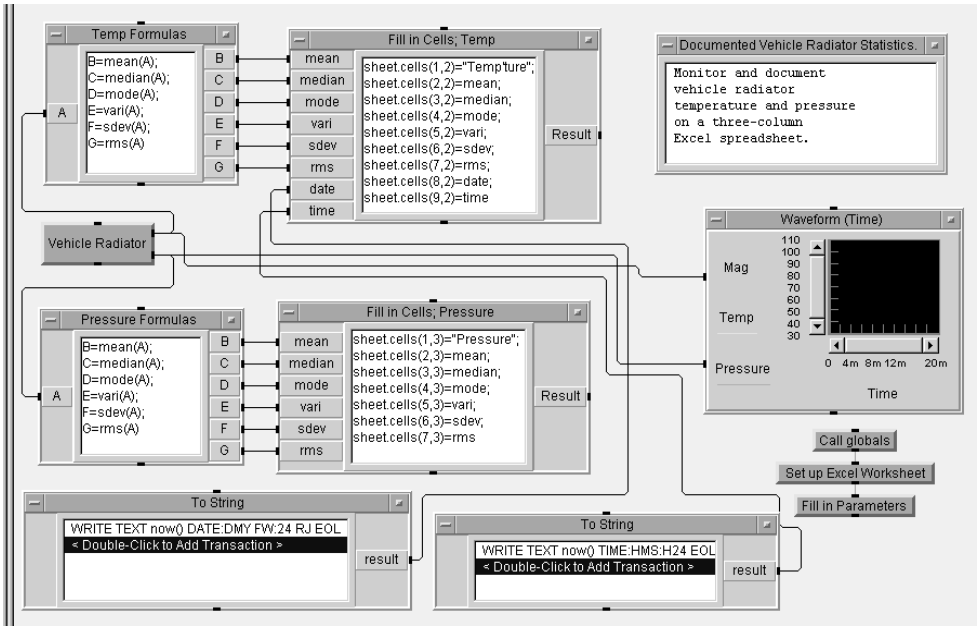
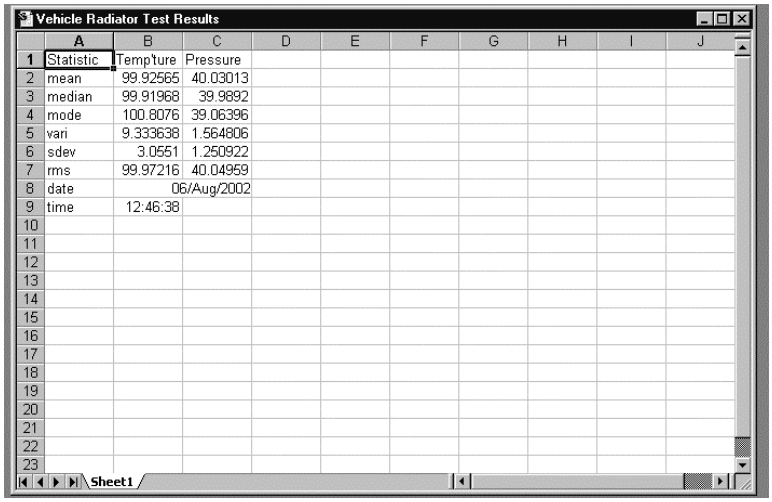


Figure 10.3. The completed Lab 10-2 program

30. Run this program.
31. Examine the Excel™ three-column spreadsheet; it should look like the one in Figure 10.4 but with a different set of values.



	A	B	C
1	Statistic	Temp'ture	Pressure
2	mean	99.92565	40.03013
3	median	99.91968	39.9892
4	mode	100.8076	39.06396
5	vari	9.333638	1.564806
6	sdev	3.0551	1.250922
7	rms	99.97216	40.04959
8	date		06/Aug/2002
9	time	12:46:38	

Figure 10.4. The three-column spreadsheet

32. The VEE Pro program should look like Figure 10.5.

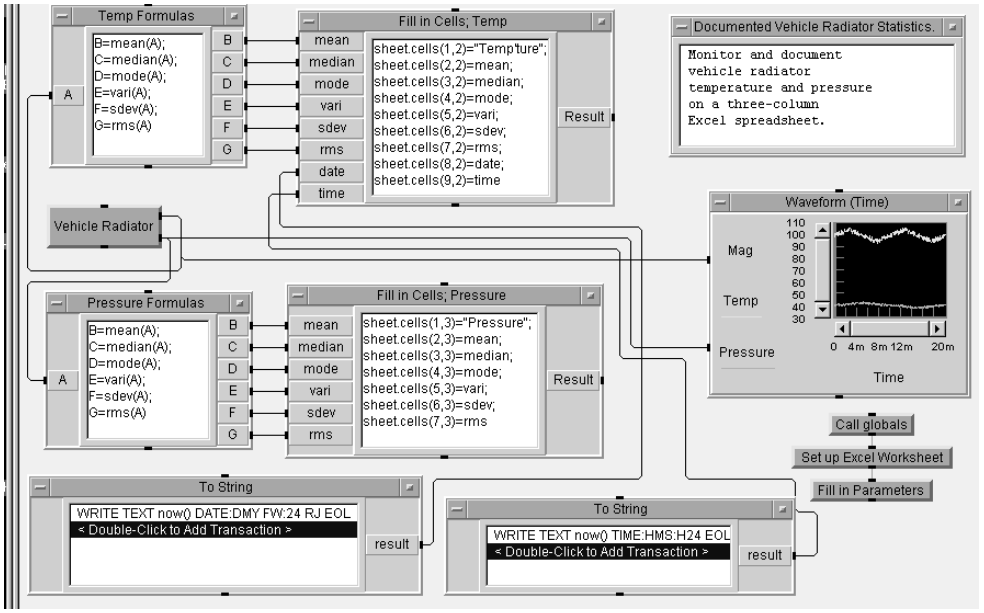


Figure 10.5. The VEE Pro program for Lab 10-2 after running

33. Close this program without saving it again.

Lab 10.3 – Using Excel™ to Document Six Sequential Vehicle Radiator Tests

This lab will show you how to send six sets of sequential tests from your Vehicle Radiator program to a single Excel™ spreadsheet.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Using Excel™ to document several test runs

2. Open LAB10-2; Save As… LAB10-3 immediately.
3. Close the following icons: Temp Formulas, Fill in Cells; Temp, Pressure Formulas, and Fill in Cells; Pressure.
4. Tighten the spacing between the two pairs of icons.
5. Change the wording in the Note Pad to read:

   Monitor, display, record, and document six tests of vehicle radiator temperature and pressure parameters on an Excel spreadsheet.
6. Select Menu Bar => Flow => Repeat => For Range; place under Fill in Parameters.

7. Change the For Range title to For Six Tests; change From to 1; Thru to 6; and Step to 1.
8. Connect the bottom (sequence) output of Fill in Parameters to the top (sequence) input of For Six Tests.
9. Add an input "A" to each of the two To String icons.
10. Connect both To String inputs to the data output (right side) of For Six Tests.
11. Add another input "A" to Fill in Cells; Temp.
12. Add another input "A" to Fill in Cells; Pressure.
13. Connect each of these new "A" inputs to the data output of For Six Tests.
14. Open the Fill in Cells; Temp icon; revise the white space to read:

```
sheet.cells(1,0 +2*A)="Temp'ture";
sheet.cells(2,0 +2*A)=mean;
sheet.cells(3,0 +2*A)=median;
sheet.cells(4,0 +2*A)=mode;
sheet.cells(5,0 +2*A)=vari;
sheet.cells(6,0 +2*A)=sdev;
sheet.cells(7,0 +2*A)=rms;
sheet.cells(8,0 +2*A)=date;
sheet.cells(9,0 +2*A)=time
```

15. Close the Fill in Cells; Temp to become an icon.
16. Open the Fill in Cells; Pressure icon; revise the white space to read:

```
sheet.cells(1,1+2*A)="Pressure";
sheet.cells(2,1+2*A)=mean;
sheet.cells(3,1+2*A)=median;
sheet.cells(4,1+2*A)=mode;
sheet.cells(5,1+2*A)=vari;
sheet.cells(6,1+2*A)=sdev;
sheet.cells(7,1+2*A)=rms
```

17. Close the Fill in Cells; Pressure to become an icon.
    Note: The "A" inputs causes each "sheet.cells" statement to step two cells to the right for the six tests.
18. Save this program; it should look like the one shown in Figure 10.6.

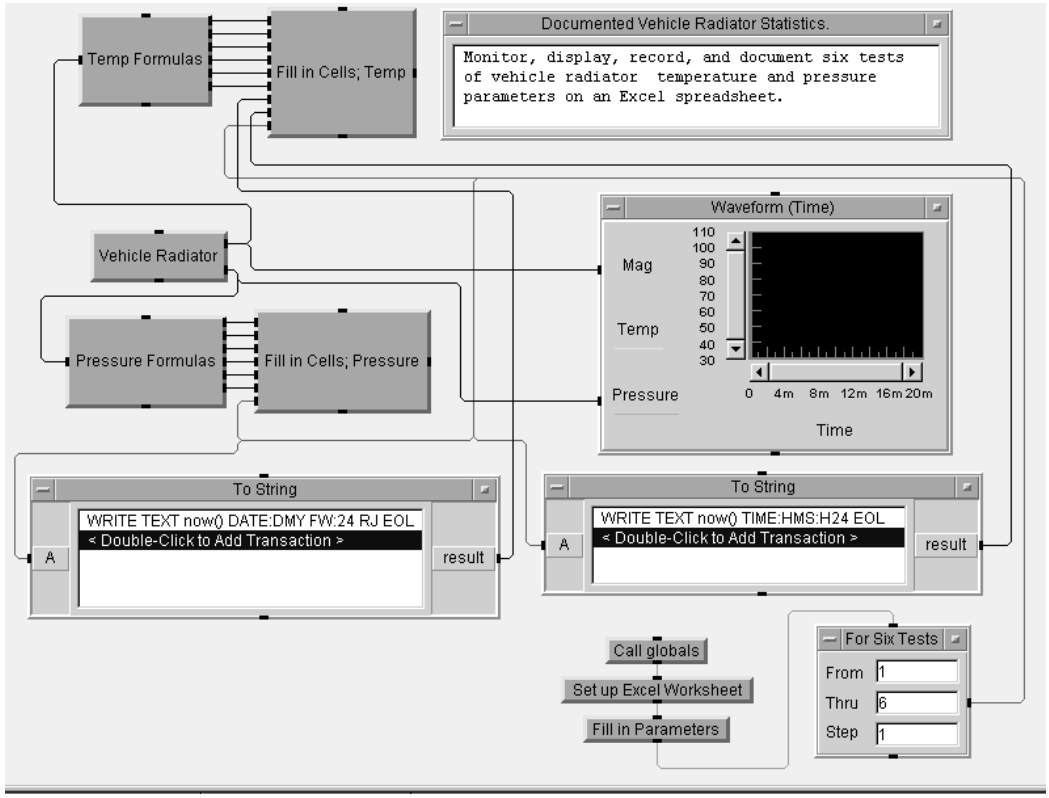
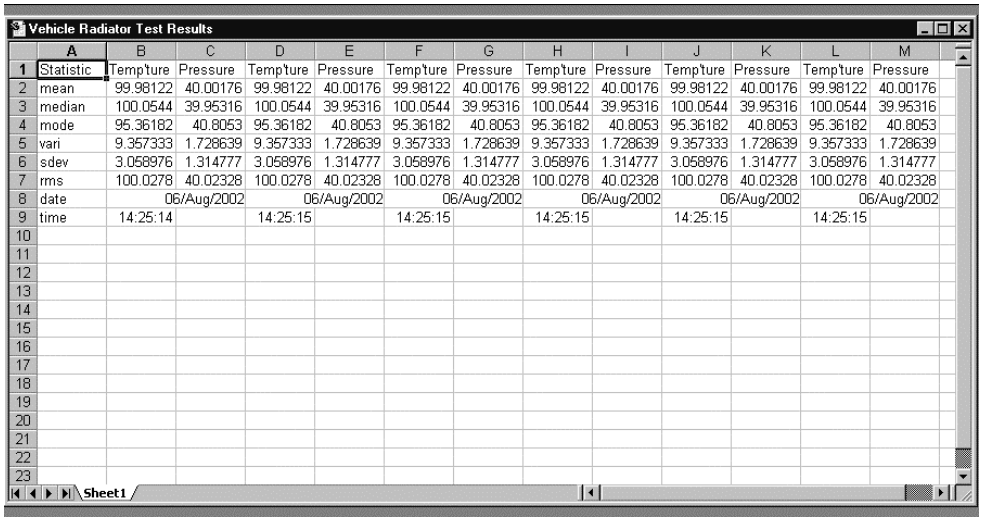


Figure 10.6. Lab 10-3 program before running

19. Run this program. The Excel™ spreadsheet should look like the one shown in Figure 10.7 but with different data values.



Vehicle Radiator Test Results

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Statistic	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure
2	mean	99.98122	40.00176	99.98122	40.00176	99.98122	40.00176	99.98122	40.00176	99.98122	40.00176	99.98122	40.00176
3	median	100.0544	39.95316	100.0544	39.95316	100.0544	39.95316	100.0544	39.95316	100.0544	39.95316	100.0544	39.95316
4	mode	95.36182	40.8053	95.36182	40.8053	95.36182	40.8053	95.36182	40.8053	95.36182	40.8053	95.36182	40.8053
5	vari	9.357333	1.728639	9.357333	1.728639	9.357333	1.728639	9.357333	1.728639	9.357333	1.728639	9.357333	1.728639
6	sdev	3.058976	1.314777	3.058976	1.314777	3.058976	1.314777	3.058976	1.314777	3.058976	1.314777	3.058976	1.314777
7	rms	100.0278	40.02328	100.0278	40.02328	100.0278	40.02328	100.0278	40.02328	100.0278	40.02328	100.0278	40.02328
8	date	06/Aug/2002		06/Aug/2002		06/Aug/2002		06/Aug/2002		06/Aug/2002		06/Aug/2002	
9	time	14:25:14		14:25:15		14:25:15		14:25:15		14:25:15		14:25:15	
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													
21													
22													
23													

Sheet1

Figure 10.7. The six-test Excel™ spreadsheet

◊ 20. Try additions to these sets of data; examples are: the inclusion of max and min values.
Note: This may be awkward for you at first. You must switch back and forth between LAB10-3 and Microsoft Excel™. You must also open and close the appropriate display icons whose values should agree with those in the spreadsheet.

21. Save this revised program for your library if so desired.

Lab 10.4 – Moving Vehicle Radiator Information from Excel™ to Word™

This lab will show you how to copy and paste a VEE Pro/ Excel™ spreadsheet with a date & time stamp, send a graph of the temperature and pressure waveform, and add text to a Microsoft Word™ document.

Open your VEE Pro program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main; open LAB10-3; immediately Save As… LAB10-4.

Transferring spreadsheet(s) and graph(s) to Microsoft Word™ reports

2. Select Menu Bar => Device => ActiveX Automation References….
3. Check the Microsoft Excel and Microsoft Word boxes; click OK.
4. Open separately a new Microsoft Word™ document; title the file: VR Test Report.doc and save it. Then add a title within the document, such as: Vehicle Radiator Test Report.
5. Type an introductory description of what the reader should expect to see in your report.

The following report is a summary of the first tests of the Vehicle Radiator monitoring program. Included in this report is a copy of the program (Figure 1) that was used to perform the six tests.

Note 1: If the entire program shows on your screen, then copy using Print Screen if it has been configured for your one printer.

Note 2: If the entire program does not show on your screen, then Select All (Ctrl A), then Edit => Copy (Ctrl C) and Edit => Paste (Ctrl V) the bitmap of the VEE program into your Word document. Crop your figure to show only the VEE program.

Note 3: If you want to eliminate the right-hand and bottom desktop information, then press Alt and Print Scrn prior to pasting it into a document.

6. Choose a title for Figure 1 in the Word document.
7. Type a description of what will be on your spreadsheet that you are placing in your document. Our suggestion is:

   The following report is a copy of the spreadsheet (Figure 2) that was completed during six of the tests.
8. Choose a title for Figure 2, such as: “Results of the Six Tests”.
9. Copy and paste your spreadsheet, cropping it, if necessary, as explained in Lab 9-3.
10. Type a description of what will be on your graph that you are placing in your document. Our suggestion is:

    Also included is a graphic (Figure 3) that indicates how much the temperature and pressure deviate from a given value that is 100°F for the temperature and 40 psi for the pressure during the last test. This presentation changes somewhat for each of the six tests.
11. Choose a title for Figure 3.
12. Insert your Waveform (Time) graph, cropping it if necessary as explained in Lab 9.3.

    Note 1: You cannot access the Enable Events box for the other three objects.

    Note 2: Our report example is shown on the next two pages.
13. Insert a managerial request. Our suggestion is:

    Additional tests will be performed upon receipt of a work order with instructions.
14. Add your signature line(s).
15. Save your report if you have not done so already.
16. Close your program without saving it.
17. Close your report; print a hard copy if so desired.

    Note: If required, physically sign the report when printed as a hard copy.

Vehicle Radiator Test Report

The following report is a summary of the first tests of the Vehicle Radiator monitoring program. Included in this report is a copy of the program (Figure 1) that was used to perform the six tests.

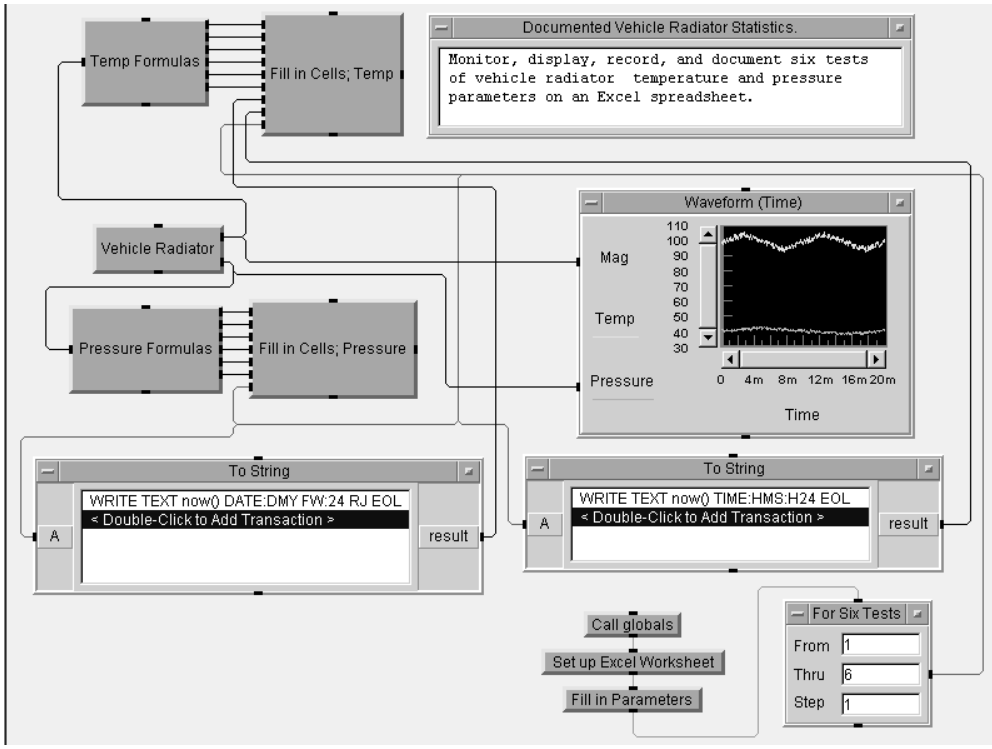


Figure 1. The six-test Vehicle Radiator program

The following report is a copy of the spreadsheet (Figure 2) that was completed during six of the tests.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Statistic	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure	Temp'ture	Pressure
2	mean	99.97737	39.99837	99.97737	39.99837	99.97737	39.99837	99.97737	39.99837	99.97737	39.99837	99.97737	39.99837
3	median	100.0255	39.93268	100.0255	39.93268	100.0255	39.93268	100.0255	39.93268	100.0255	39.93268	100.0255	39.93268
4	mode	101.218	39.0025	101.218	39.0025	101.218	39.0025	101.218	39.0025	101.218	39.0025	101.218	39.0025
5	vari	9.744733	1.713123	9.744733	1.713123	9.744733	1.713123	9.744733	1.713123	9.744733	1.713123	9.744733	1.713123
6	sdev	3.121656	1.308863	3.121656	1.308863	3.121656	1.308863	3.121656	1.308863	3.121656	1.308863	3.121656	1.308863
7	rms	100.0259	40.01969	100.0259	40.01969	100.0259	40.01969	100.0259	40.01969	100.0259	40.01969	100.0259	40.01969
8	date	28/Aug/2002		28/Aug/2002		28/Aug/2002		28/Aug/2002		28/Aug/2002		28/Aug/2002	
9	time	9:24:44		9:24:45		9:24:45		9:24:45		9:24:45		9:24:45	
10													

Figure 2. Results of the six tests

Also included is a graphic (Figure 3) that indicates how much the temperature and pressure deviate from a given value that is 100°F for the temperature and 40 psi for the pressure during the last test. This presentation changes somewhat for each of the six tests.

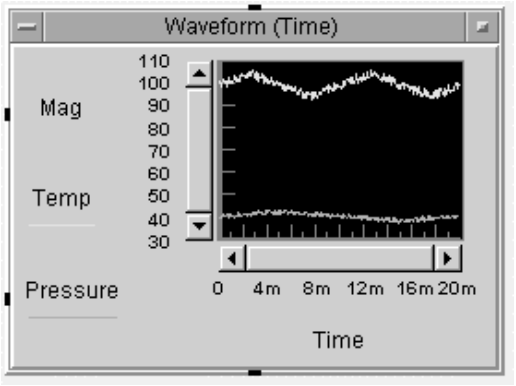


Figure 3. Graphs of the Vehicle Radiator last test

Additional tests will be performed upon receipt of a work order with instructions.

Thomas E. Hulbert, P.E.
Robert B. Angus, P.E.

Lesson 10 Summary

This lesson described and illustrated how to create spreadsheets using ActiveX and globals to link between VEE Pro data and a Microsoft Excel™ spreadsheet. You also learned how to compress your data, via statistics, so others can examine your test results either in their entirety or in a statistical format. Others can decide if they want to examine your recorded data in more depth prior to its compression or if they want you to perform and analyze additional tests. Next you learned how to send a spreadsheet, a graph, and then add text to a Microsoft Word™ document.

Excel™ and Word™ are accessed via ActiveX are very flexible. You are limited in their applications to your everyday test documentation only by the amount of time that you can devote to developing new and meaningful presentations. It is imperative that you consider the potential reader of your reports and spreadsheets so that you can prepare them in a standard format.

Your time is precious; so is the time of your co-workers and managers.

Lab 10.1 showed you how to monitor temperature and pressure values while simultaneously computing mean, median, mode, variance, standard deviation, and rms values of both Vehicle Radiator parameters.

Lab 10.2 showed you how to send (transfer) VEE data from your Vehicle Radiator program to Microsoft Excel™; record parameter identification in one column; and record temperature and pressure readings in two other columns.

Lab 10.3 showed you how to send six sets of sequential tests from your Vehicle Radiator program to a single Excel™ spreadsheet.

Lab 10.4 showed you how to copy and paste a VEE Pro/Excel™ spreadsheet with a date & time stamp, send a graph of the temperature and pressure waveform, and add text to a Microsoft Word™ document. (Note: Agilent VEE Pro Help Contents and Index; select the Guide to Agilent VEE Example Programs; select Microsoft Word (Windows only), and select “Send Data and Bitmap to Word97”.)

You are now ready to learn to more about using UserFunctions.

Lesson 11
Using VEE Pro to Create UserFunctions

This lesson will examine the Merge… command and how it allows you to merge your new test programs with existing VEE Pro programs. This lesson will also examine how to use VEE Pro to create user functions. It consists of a pre-lab and four labs.

Lesson 11 Pre-lab Summary

The following are described in the Lesson 11 pre-lab. See Appendix E, page E-62.

- Merging Objects
- User Objects
- User Functions

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 11.1 – Merging a Bar Chart Display Program
This lab will show you how to merge existing VEE Pro programs with other programs.

Lab 11.2 – Working with UserFunction Operations
This lab will show you how to create a UserFunction and send data to its output pins.

Lab 11.3 – Learning to Call and Edit a UserFunction
This lab will show you how to call and edit a UserFunction and how to adjust the number of pins to match your recent edit.

Lab 11.4 – Monitoring the Vehicle Radiator with UserFunctions
This lab will show you how to display the statistics gathered from a monitored device on one record per parameter.

Lab 11.1 – Merging a Bar Chart Display Program

This lab will show you how to merge a bar chart display program with a program that contains the ramp() function.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Displaying VEE Pro data on a bar chart

2. Select Menu Bar => Device => Formula; place it to the left-center of your Work Area.
3. Delete the data-input terminal.
4. Change the formula expression to ramp(5,1,5).
   Note 1: The first parameter is the number of elements desired in the ramp array.
   Note 2: The second parameter is the starting number.
   Note 3: The third parameter is the ending number.
   Note 4: Select Help in the Formula Properties from the object menu if you desire more information; it will now default to showing "help" on the "ramp" function.
   or
   Try Menu Bar => Help => Contents & Index and use the Search feature in the Index folder.
5. Select Menu Bar=> File => Merge… to get the Merge File list box, then click on VEE Pro => Library => BarChart; place it to the right of the Formula object.

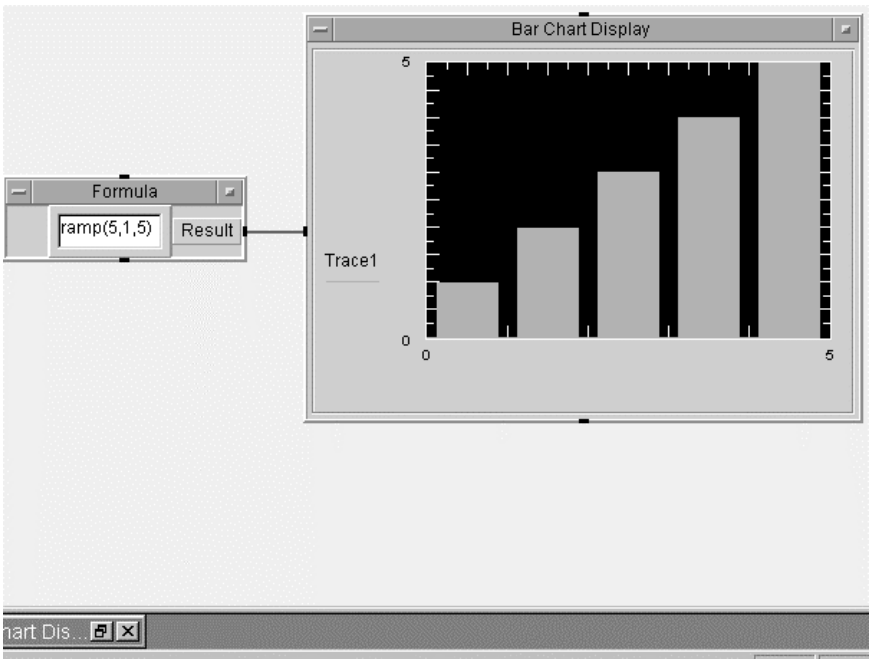


Figure 11.1. Displaying data on a BarChart

6. Double-click on Bar Chart in Program Explorer to edit it; examine the detail of the UserObject by clicking on the upper-left third icon from the left.
7. Click on its "_" button in the upper right corner to place it at the bottom-left of your screen.
8. Connect the two objects: Formula output (Result) to BarChart Display input (Trace1).
   Note: The Bar Chart Display takes a one-dimensional array and displays the value as vertical bars. It uses the number of bars necessary to display the values in the array.
9. Run this program. It should look like Figure 11.1.
◊ 10. Change the values in the "ramp" formula; run several combinations of values and observe the bar chart.
11. Save this program as LAB11-1 with its original (5,1,5) ramp settings.
12. Close this saved program.

Lab 11.2 – Working with UserFunction Operations

This lab will show you how to create a UserFunction (named ArrayStats). This UserFunction will accept an array, calculate its maximum, minimum, mean, and standard-deviation values, and send data to its UserFunction output pins.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Creating a UserFunction

2. Select Menu Bar => Device => Formula; place it in the upper-left corner of your Work Area; delete its default input pin.
3. Change its default expression to ramp(1024,1,1024).
   Note: This will create a 1024-element array with values from 1 through 1024.
4. Select Menu Bar => Device => UserFunction; rename it ArrayStatistics with no space.
5. Add one data-input terminal (A) for the array.
6. Add four data-output terminals (X, Y, Z, W) for the results in the order they appear.
7. Rename the output pins in any order:
   max min mean sdev
8. Select Menu Bar => Device => Function & Object Browser; select the Probability & Statistics category.
9. Select max, min, mean, and sdev from the Member menu – four formula objects. Place each of them in the ArrayStatistics UserFunction.
10. Verify that each of the four formulas (max, min, mean, and sdev) appears in ArrayStatistics; <u>not</u> in Main.
11. Connect the data input of each object to the ArrayStatistics A pin.
12. Connect the data outputs of each object to the appropriate output pin.
13. Make the ArrayStatistics window smaller so you can see both windows. See Figure 11.2.

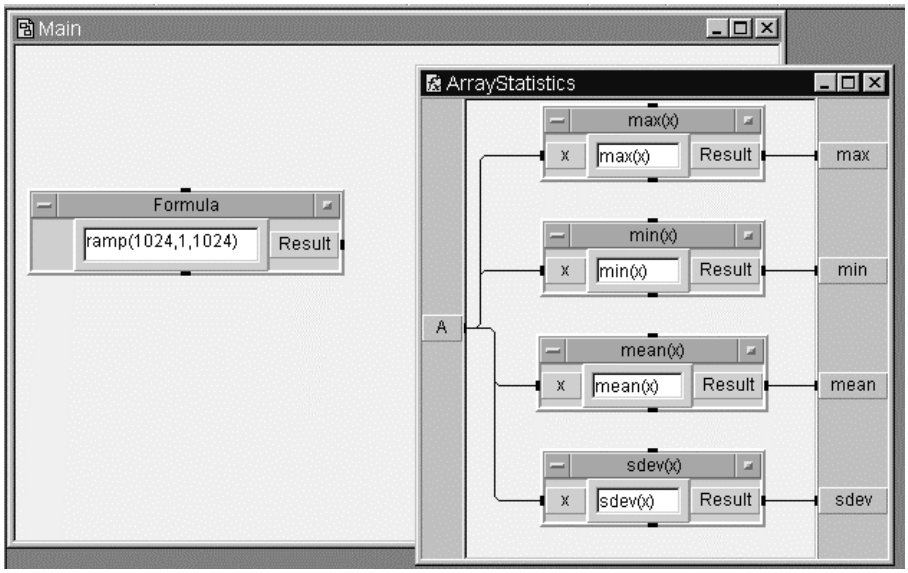


Figure 11.2. The Main and ArrayStatistics windows

14. Convert ArrayStatistics to an icon (minimize) at the bottom of the workspace.
15. Select Menu Bar => Click Device => Call; place it to the right of Formula.
16. Open the Call Function object menu; click on Select Function. See Figure 11.3 (the default values are correct); click OK.

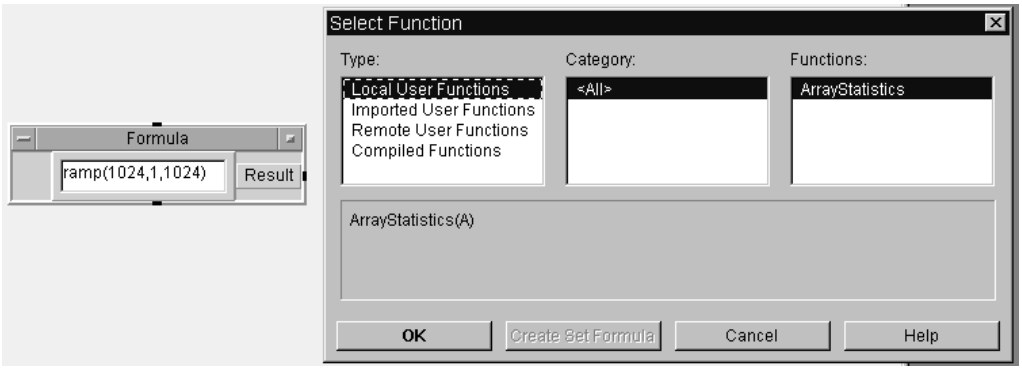


Figure 11.3. Configuring Call myFunction

Note: VEE Pro automatically renames the object and adds the correct pins.

17. Connect the Formula output (Result) to the Call ArrayStatistics input (A).
18. Select Display AlphaNumeric and clone it three times.
19. Connect these displays to the Call ArrayStatistics output pins.

20. Rename the displays to match the Call ArrayStatistics output-pin names.
21. Save this program as LAB11-2.
22. Run this program. It should look like Figure 11.4. Note ArrayStati… in the lower-left corner.

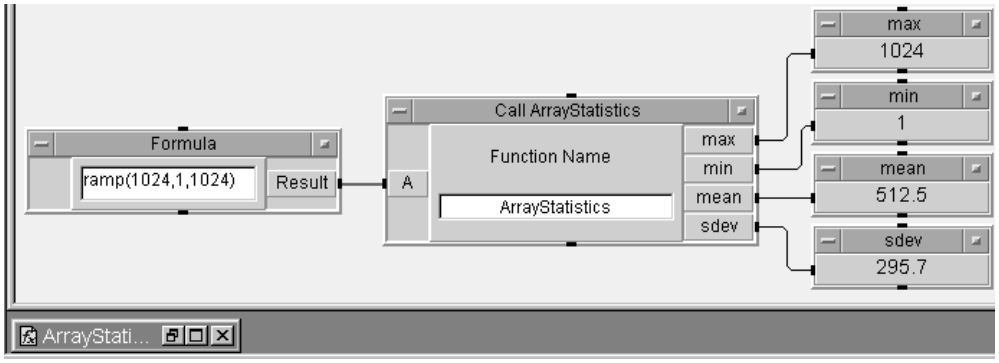


Figure 11.4. ArrayStatistics program after running

Note: If you wish to use ArrayStatistics in another program, you will later learn to use the Import Library function to accomplish your desire.

◊ 23. Change the values within the ramp formula box; run the program with several different values; return the ramp-formula values back to the original ones given.
24. Save and close the final program as LAB11-2.

Lab 11.3 – Learning to Call and Edit a UserFunction

This lab will show you how to call and edit a UserFunction, and how to adjust the number of pins to match your recent edit.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Editing a UserFunction

Note: This exercise will show you how to edit ArrayStatistics to deliver a record with four fields that provide the array statistics.

2. Open LAB11-2; Save As… LAB11-3. In Main, delete (cut) the four AlphaNumeric displays.
3. Select Menu Bar => Edit => Edit UserFunction… .
4. Select ArrayStatistics in the Edit UserFunction list box; click OK.
   Note 1: All of the User Functions in your existing programs will be displayed here.
   Note 2: You may double-click on a UserFunction from the Program Explorer to edit it.
5. Click-and-drag the lower-right-hand corner to enlarge the editing window of ArrayStatistics to approximately twice its present width.
6. Click on the ArrayStatistics Work Area with the mouse right button; click on “Clean Up Lines”.
7. Use the scissors to cut the four lines going to the output pins.
   or

Hold down the Shift and Ctrl keys and move the mouse over the line to be cut; the pointer will change to scissors; left-click to cut the line.

8. Select Menu Bar => Data => Build Data => Record; place it to the right side in the ArrayStatistics window.
   Note: You may need to further increase the size of ArrayStatistics so Build Record will fit.
9. Add two more data-input terminals to Build Record.
10. Rename these four pins on Build Record after the statistical functions:
    max min mean sdev
11. Connect the four Build Record input pins to the appropriate Formula objects.
12. Double-click on the ArrayStatistics output pin: Max, rename it **X**; click OK.
13. Delete the three other ArrayStatistics data-output terminals by placing the mouse over the terminal and depressing Ctrl + d (new shortcut).
14. Connect the Build Record output to the X output pin on the User Function editing window. See Figure 11.5.

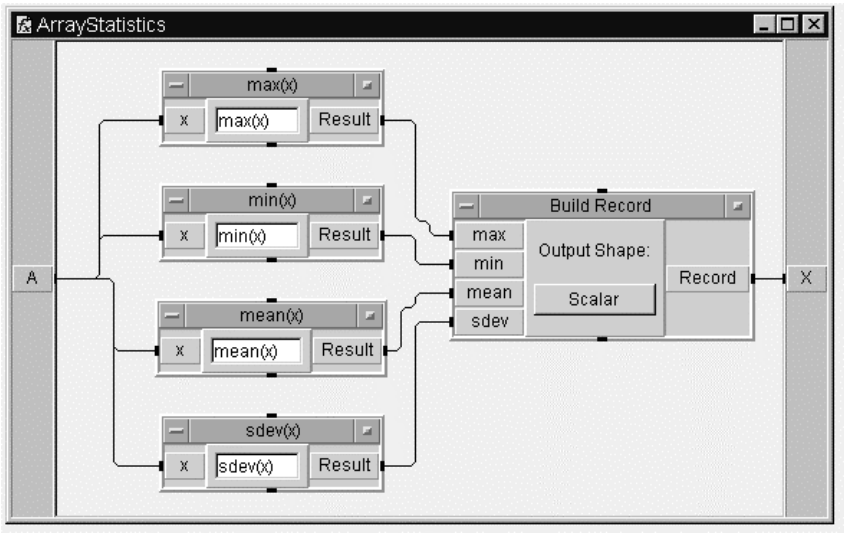


Figure 11.5. The ArrayStatistics UserFunction after editing

15. Minimize ArrayStatistics to appear in the lower-left corner.
16. Open the Call ArrayStatistics object menu in Main; click Configure Pinout.
    Note: This action will adjust the number of pins to match your recent edits. (You must do this every time you change the UserFunction number of inputs or outputs.)
    Select Menu Bar => Data => Constant => Record; place it to the right of Call ArrayStatistics.
    Note: You may use the Record Constant object to display a record.
18. Open the Record Object menu; click Add Terminal => Control Input… .
19. Select Default Value from the list box presented; click OK; lengthen the Record box so that it can display all four field names.

Note: Use the Default Value control input to accept a record from ArrayStatistics. VEE Pro will automatically configure the Record Constant to hold the incoming record.

20. Connect the Call ArrayStatistics data-output pin to the control-input (left-hand, Default Value) pin on the Record Object.
    Note 1: Control lines are indicated by dashed lines to distinguish them from data lines.
    Note 2: When you run the program, the field names will automatically assume the UserFunction output names.
21. Save your LAB11-3 program.
22. Run this program. It should look like Figure 11.6.

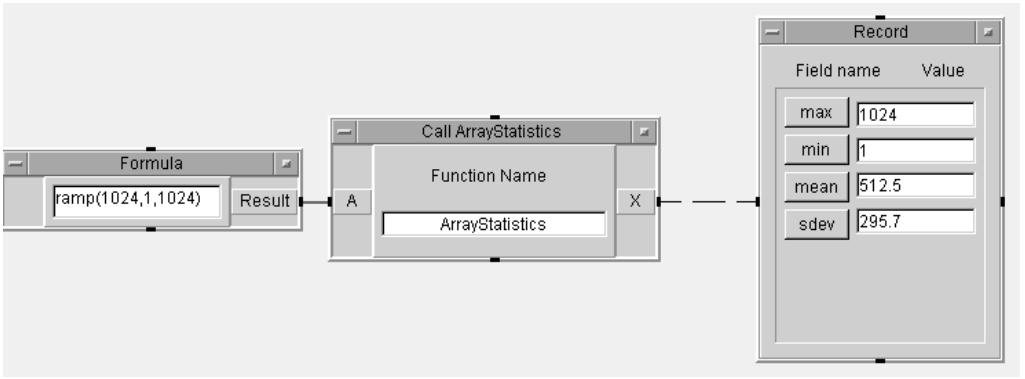


Figure 11.6. ArrayStatistics output edited to display a record

Calling a UserFunction from an expression

Note: You will call ArrayStatistics from an expression in the Formula Object.

23. Open LAB11-3; Save As… LAB11-3a immediately.
24. Select Menu Bar => Device => Formula.
25. Replace the default formula expression with ArrayStatistics(A).
26. Go to the Call ArrayStatistics object menu; click on Replace.
    Note: The cursor will become a cross, click on the object that will replace the older object.
27. Click on the Formula Object you just created to call the ArrayStatistics function; then click in the workspace where the old Call ArrayStatistics was located.
    Note 1: VEE Pro will automatically retain the wiring of the data lines.
    Note 2: The Formula Object will take the input at pin A and feed it to the UserFunction ArrayStatistics of LAB11-3. This action will deliver the record of statistics to pin X on the UserFunction.
    Note 3: The first output value from the UserFunction (X) of LAB11-3 will be returned to the Formula Object and delivered to its Result output and then delivered to the Record Constant: "Record".
28. Save your LAB11-3a program again.
29. Run this program. It should look like Figure 11.7.

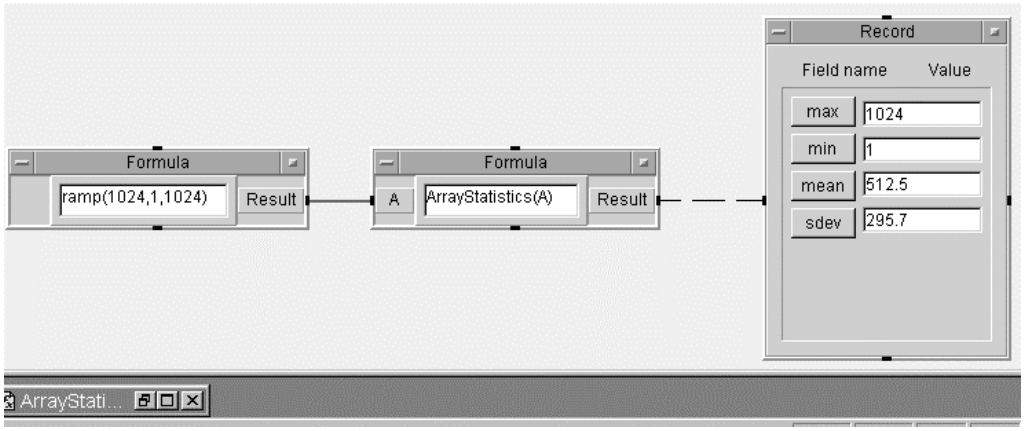


Figure 11.7. Calling the ArrayStatistics UserFunction

Note 1: The functionality of ArrayStatistics in the Formula Object is exactly the same as it was in the Call ArrayStatistics object. A Formula Object was used in this example. You could call ArrayStatistics from any input field that accepts expressions, such as ToFile or IfThen.

Note 2: When you are calling a UserFunction from an expression, the UserFunction will deliver only a single output – the uppermost data-output pin to the expression. If you need all of the outputs or they cannot be placed into a Record, then you should use the Call Function Object.

Note 3: If no data are to be passed to your function, then you must still include empty parentheses after the function name in an expression.

Note 4: The Call object does not require the parentheses because VEE Pro knows you are referring to a function.

Note 5: As your programs become more complex, the Program Explorer can assist you in navigating through programs. See Figure 11.8. You can easily see the hierarchy of your program via the View menu; click on the "+" sign and ArrayStatistics will appear as a UserFunction (f_x).

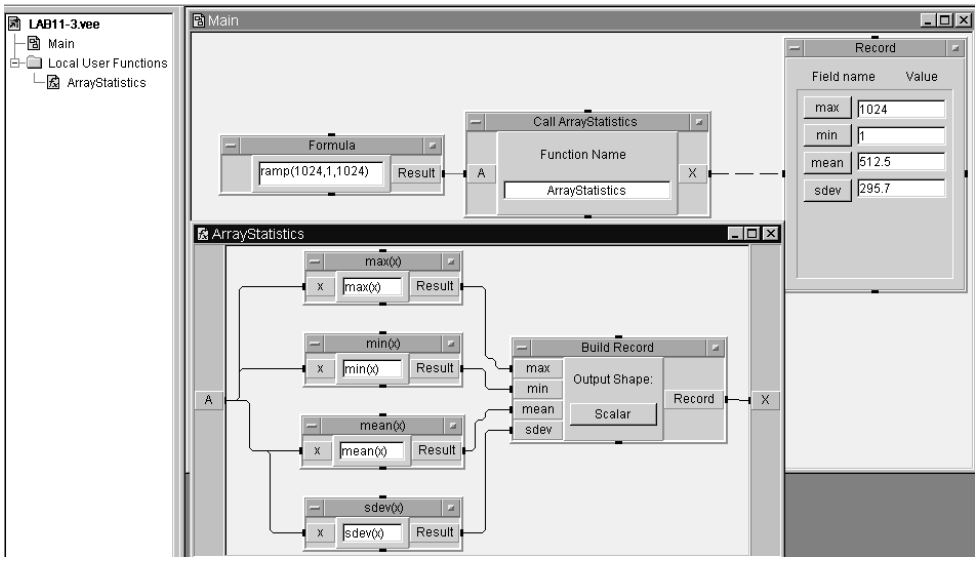


Figure 11.8. The Program Explorer hierarchy for Lab 11-3

Lab 11.4 – Monitoring the Vehicle Radiator with UserFunctions

This lab will show you how to display the statistics gathered from a monitored device on one record per parameter by editing an existing UserFunction.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Editing a UserFunction

2. Open LAB7-5; Save As… LAB11-4.
3. Delete (cut) all twelve alphanumeric displays.
   Note: A rapid method for cutting the displays is to hold down the Ctrl key while clicking on each icon. Then use the object-cutting pair of scissors to cut all twelve at once or use Ctrl + X to cut all selected objects.
4. Repeat step 3 for the twelve statistical-formula objects. Save the program again.
   Note: Only the Vehicle Radiator UserFunction and the Display will remain.
5. Open LAB11-3a; open the ArrayStatistics UserFunction; select Copy from its f_x object menu.
6. Open LAB11-4; (do not re-save Lab 11-3a if it asks you to do this); paste ArrayStatistics onto the Main screen; re-name it VehicleRadiatorData without any spaces.
7. Increase the VehicleRadiatorData UserFunction size vertically to accept two more icons.
8. Clone two of the statistical-formula icons; place them beneath the other four icons; connect these two cloned icons to the X input pin of VehicleRadiatorData.
9. Change the titles and formula expressions to mean, median, mode, vari, sdev, and rms; the formula expressions will require the"(x)" following each statistical function name.

Note: The shortcut to re-name an object is: double-click on its Title Bar. The title string will appear highlighted so you may immediately re-type the entire title.

10. Go to Build Record; add two more input terminals; label all six pins to agree with the statistical-formula titles. See Figure 11-9.

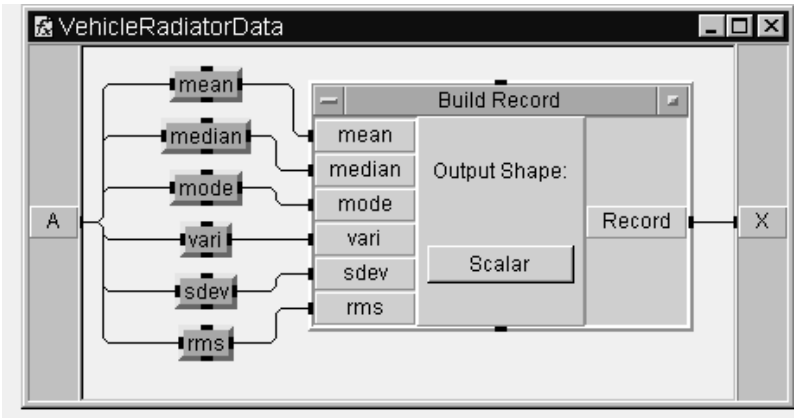


Figure 11.9. The VehicleRadiatorData UserFunction

11. Minimize the VehicleRadiatorData UserFunction to appear at the bottom of the screen.
12. Save this program again.
13. Return to LAB11-3a; copy Record; close LAB11-3a without saving the program.
14. Paste Record into LAB11-4; move it to the right side of your screen; change its title to VRTemperature Record.
15. Select Menu Bar => Device => Call; place it to the left of VRTemperature Record; change its Function Name to VehicleRadiatorData.
16. Connect its input pin to the Vehicle Radiator "Temp" output pin; connect its output pin to the VRTemperature Record input pin.
17. Run this partial program to determine if de-bugging is necessary.
18. Clone Call VRTemperature; change its Function Name to VehicleRadiatorData.
    Note: Whenever two Call functions are identical, one of them may be eliminated. The single Call function should be identified with a common name; in this example: VehicleRadiatorData. In the first case, it is used to process VRTemperature values. In the second case, this same Call function is used to process VRPressure values. This becomes one icon that contains reusable code.
19. Connect and place the two UserFunction icons as shown in Figure 11.10.

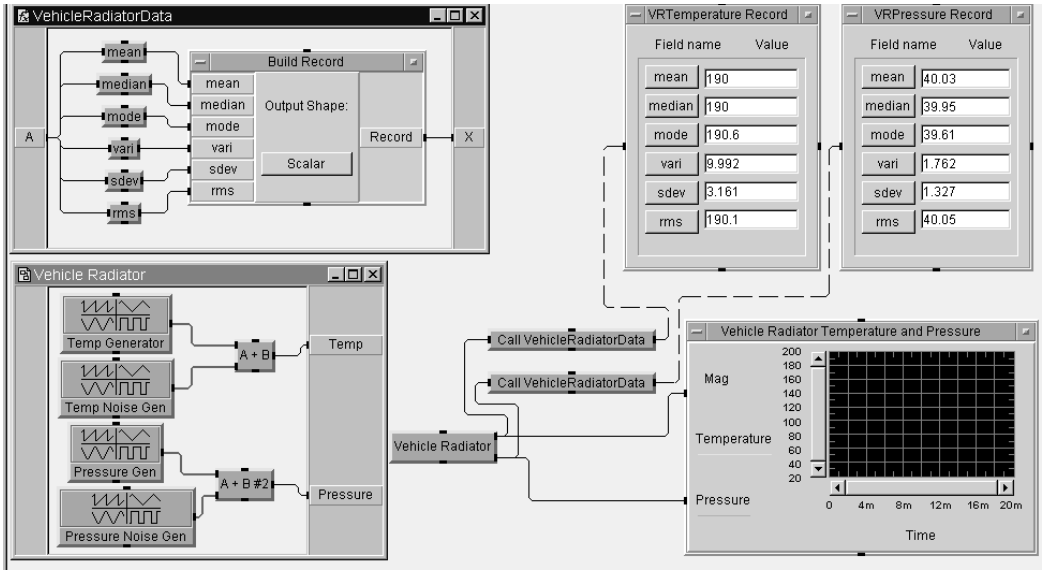


Figure 11.10. The completed Lab 11-4 program

20. Run this program; it should look like Figure 11.11.

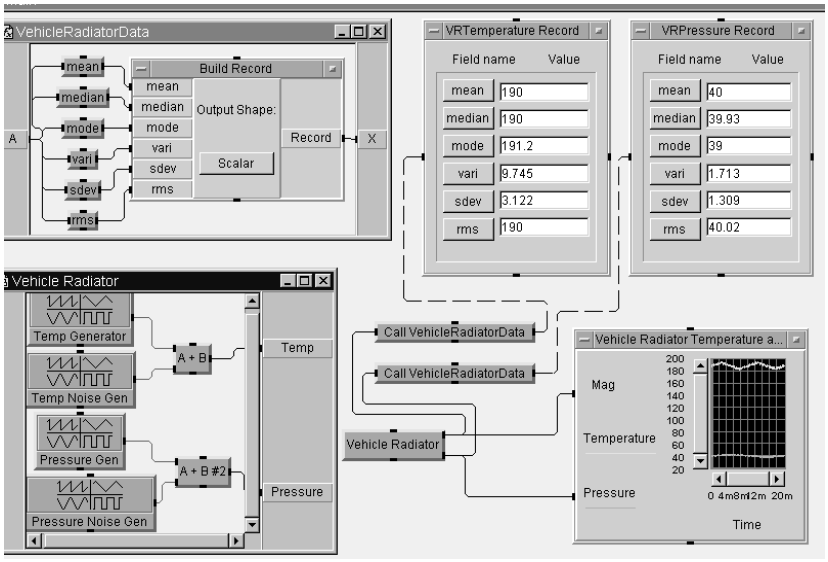


Figure 11.11. The Lab 11-4 program after running

21. Save and close this program.

Lesson 11 Summary

There are three types of user-defined functions in VEE Pro.

- UserFunctions
- Compiled Functions
- Remote Functions

Other functions may exist in user-supplied libraries.

Your new programs can be easily merged with existing VEE Pro programs using the Merge… command. This command is located in the File menu. (There is also the Merge Library command that is used to merge libraries of UserFunctions. Therefore, importing a library of UserFunctions is recommended for re-using and maintaining codes [subprograms]).

Lab 11.1 showed you how to merge existing VEE Pro programs with other programs.

Lab 11.2 showed you how to create a UserFunction and send data to its output pins.

Lab 11.3 showed you how to call and edit a UserFunction and how to adjust the number of pins to match your recent edit.

Lab 11.4 showed you how to display the statistics gathered from a monitored device on one record per parameter by editing a UserFunction. A simplified version (VehicleRadiatorData) was used because the Call functions were identical.

You are now ready to learn how to simulate a variety of devices via VEE Pro.

Lesson 12
Using VEE Pro for Application Simulations

This lesson will examine how to use VEE Pro for application simulations. It consists of a pre-lab and five labs.

Lesson 12 Pre-lab Summary

The following are described in the Lesson 12 pre-lab. See Appendix E, page E-64.

- The Instrumentation Amplifier
- Understanding a Strain Gauge
- The VEE Pro-provided application programs
- MATLAB® Script
- MATLAB® Function & Object Browser

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 12.1 – Simulating an Instrumentation Amplifier
This lab will show you how to generate a differential signal with an ideal instrumentation amplifier, examine the ability of an IA to measure a small signal buried in noise, and change the differential gain of the IA by a factor of ten.

Lab 12.2 – Simulating a Strain Gauge
This lab will show you how to construct a four-element simulated strain gauge using simulated strain-gauge resistors and a Wheatstone bridge.

Lab 12.3 – Exploring Four Mechanical Simulations
This lab will show you how to access and explore four application programs embedded within VEE Pro. These applications are the cantilever beam deflection, angular deflection of a round torsion shaft, the natural frequencies of a coil spring, and heat transfer coefficients for several body or surface types.

Lab 12.4 – Exploring a Simulated Manufacturing Test System
This lab will show you how to combine features of VEE Pro to build a manufacturing test system.

Lab 12.5 – Plotting Simulated Vehicle Radiator Temperatures via MATLAB®
This lab will show you how to use MATLAB® to display changes in the Vehicle Radiator temperature distribution.

Lab 12.1 – Simulating an Instrumentation Amplifier

This lab will show you how to generate a differential signal with an ideal instrumentation amplifier, examine the ability of that amplifier to measure a small signal buried in noise, and change the differential gain of the amplifier.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Generating a differential signal with an ideal instrumentation amplifier

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper left corner of Main.
3. Clone the Function Generator object and place at the lower left corner of Main; reduce the size of both internal Function Generator objects as much as possible.
4. Select Menu Bar => Device =>Virtual Source => Noise Generator; reduce its size to as small as possible and place it between the two Function Generator objects.
5. Select Menu Bar => Device => Formula; place it to the right of the upper Function Generator; add an input terminal by placing the cursor over the input terminal area and pressing Ctrl-a; type A+B in the formula expression.
6. Clone the Formula object and place it to the right of the lower Function Generator object.
7. Rename the Title Bars of the on-screen objects from their Property Menus as follows:
   The Upper Function Generator: $V_{in}(+)$
   The Noise Generator: Common Mode Noise Voltage (CMV)
   The Lower Function Generator: $V_{in}(-)$
   The Upper Formula Object: $V_{in}(+)$ + CMV
   The Lower Formula Object: $V_{in}(-)$ + CMV
8. Interconnect the on-screen data pins as follows:
   $V_{in}(+)$ Func data-output pin to $V_{in}(+)$ + CMV data input pin A.
   CMV Func data-output pin to $V_{in}(+)$ + CMV data-input pin B.
   CMV Func data-output pin to $V_{in}(-)$ + CMV data-input pin A.
   $V_{in}(-)$ Func data-output pin to $V_{in}(-)$ + CMV data-input pin B.
   Convert the $V_{in}(+)$ + CMV and $V_{in}(-)$ + CMV objects to icons.
   (See Figure 12.2, page 12.4 for labeling and connection verification.)
9. Select Menu Bar => Device => UserObject and place it in Main; double-click the resulting icon; change the name of the UserObject in the Title Bar to Instrumentation Amplifier.
   Place the cursor in the left margin of the Instrumentation Amplifier UserObject and press Ctrl-A; double-click the terminal and rename it Plus Input (A). Add another terminal using the same procedure and name it Minus Input (B). Add an output terminal to the Instrumentation Amplifier using the same method, and name the terminal:
   Out = K1(A) - K2(B).
11. Select Menu Bar => Device => Formula; place it to the right of, and in line with, Plus Input (A); rename it +10X Gain (K1).
12. Go to its formula expression to: type 10*A.

13. Clone +10X Gain (K1); place it to the right of, and in line with, Minus Input (B); rename it -10X Gain (K2); change its input terminal from A to B.
14. Go to its formula expression to: type -10*B.
15. Connect the input pin of each Formula Object to its associated UserObject input pin.
16. Clone the upper Formula Object; place it to the left of the UserObject output terminal; change its name to Adder K3*(A + B); add another input terminal.
17. Go to its expression; change it to: 1*(A+B).
18. Connect the Adder K3*(A + B) input pins to the respective output pins (Result) of the other two formula objects; connect its output pin to the UserObject output pin. See Figure 12.1.

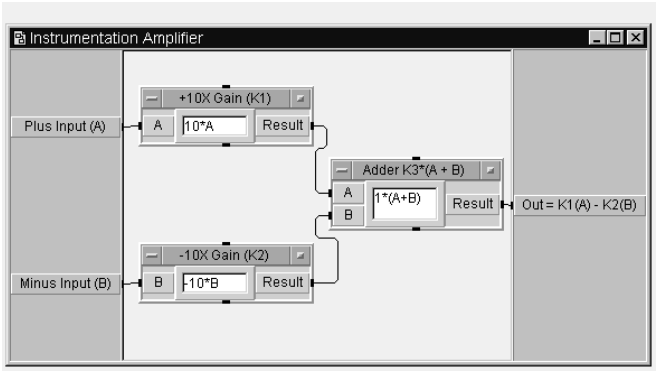


Figure 12.1. The Instrumentation Amplifier UserObject

19. Reduce the Instrumentation Amplifier UserObject to an icon – click on its (_) icon.
20. Connect the $V_{in}(+)$ + CMV output pin to the Instrumentation Ampl upper input pin Plus Input (A).
21. Connect the $V_{in}(-)$+ CMV output pin to the Instrumentation Ampl lower input pin Minus Input (B).
22. Select Menu Bar => Display => Waveform (Time); place it to the right of Instrumentation Amplifier UserObject; reduce the size of the display so it fits on the screen.
23. Connect the Instrumentation Amplifier output pin to the Waveform (Time) input pin (Trace1).
24. Save your program as LAB12-1.
25. Run this program; your output should be a zero line on the scope (Waveform (Time)).

Examining the ability of an IA to measure a small signal buried in noise

26. Go to the $V_{in}(+)$ amplitude; change it to 10.
27. Go to the $V_{in}(-)$ amplitude; change it to 9.99.
28. Save your program again.
29. Run your program; note the scope amplitude change. See Figure 12.2.
    Note: You are observing the amplified differential-input amplitude on the scope. You are measuring a 10mV differential-input signal in the presence of a common-mode error voltage of 1V peak.

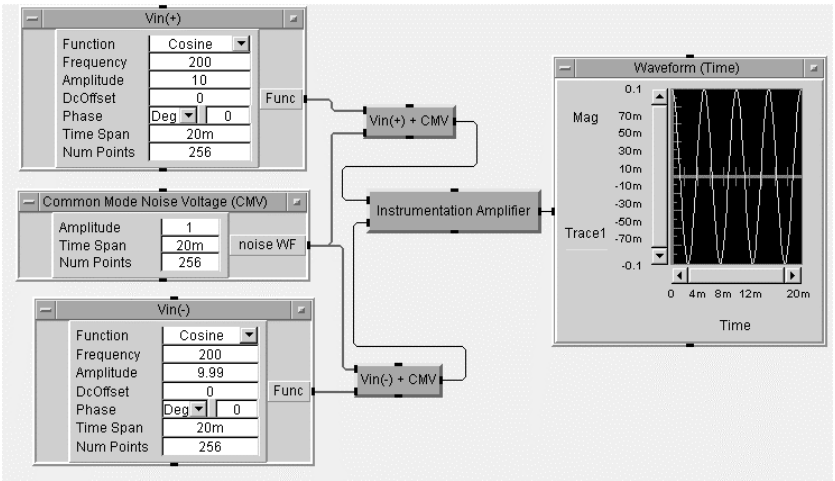


Figure 12.2. Display of a common mode voltage

◊ 30. Re-run this program several times entering differential-input voltages; observe the Waveform (Time) display result; note the absence of common-mode noise.

31. Close your program without saving it; re-open it as LAB12-1.

Changing the differential gain of the instrumentation amplifier by a factor of ten

32. Open the Instrumentation Amplifier UserObject; go to the Adder…; change its formula white-space field from 1*(A+B) to 10*(A+B); close this UserObject.
    Note: This gain change will increase the instrumentation-amplifier sensitivity by a factor of ten; it demonstrates that this does not degrade the common-mode rejection ratio so long as the absolute gain value of the preceding two amplifiers exactly match.
33. Save this program as LAB12.1a; run this program.
34. Change the Instrumentation Amplifier Adder… back to 1*(A+B); change the -10X Gain (K2) Formula Object to -(9.99*B); run this program.

◊ 35. Run this program several times with a variety of Common Mode Noise Voltage (CMV) amplitudes. Note the effect of this noise on the scope display.
    Note 1: In the real world, operational-amplifier gain-value exact matching is impossible. This portion of the lab demonstrates the effect. Also, external circuit imbalances can cause common-mode error because they will modify the differential inputs unequally.
    Note 2: Other (identical) gain values may be applied in steps 12 and 14.

36. Close this program without saving unless you desire to keep the changes.

Lab 12.2 – Simulating a Strain Gauge

This lab will simulate a four-element strain gauge. You will then study its sensitivity and the range of output strain and accompanying voltages by varying five different parameters.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Constructing a four-element simulated strain gauge

2. Select Menu Bar => Display => Note Pad. Place it in the upper-right corner of your Main window. Type the following information into its editing area:
   This is a simulation of a four-element strain gauge.
   It is monitoring a mechanical structure.
   This implementation also compensates for the structure's temperature change.
3. Size the Note Pad.
4. Select Menu Bar => Data => Continuous => Real64 Knob. Place it in the upper-left corner of your main window. Go to Properties, change its title to Gauge Excitation Voltage (V_{exc}), its Label Spacing to Every Major Tic, and its maximum value to 10.
5. Clone Gauge Excitation Voltage. Place the new Object to the right of Gauge Excitation Voltage. Change its title to Gauge Full Load (N/sq m), its Label spacing to Every Other Tic, its maximum value to 200, and its minimum value to "2".
6. Select Menu Bar => Data => Continuous => Real64 Knob. Place it below Gauge Excitation Voltage. Change its name to Gauge Sensitivity (mV/V_{exc}), its Label Spacing to Every Other Tic, and its maximum value to 10m.
7. Clone Gauge Sensitivity. Place the new Object to the right of Gauge Sensitivity. Change its title to Gauge Input Load (N/sq m) and its maximum value to 10.
8. Select Menu Bar => Device => Formula, place it between the real knobs.
   Note: This may require that you "size" the other four objects so the Formula Object will fit between them.
9. Add a Data Input Terminal. Change the A data-input terminal name to: Vexc, change the B data-input terminal name to: Sensitivity. Change its formula to read: Vexc*Sensitivity. Change its title bar to Strain Gauge (mV). See Figure 12.3.

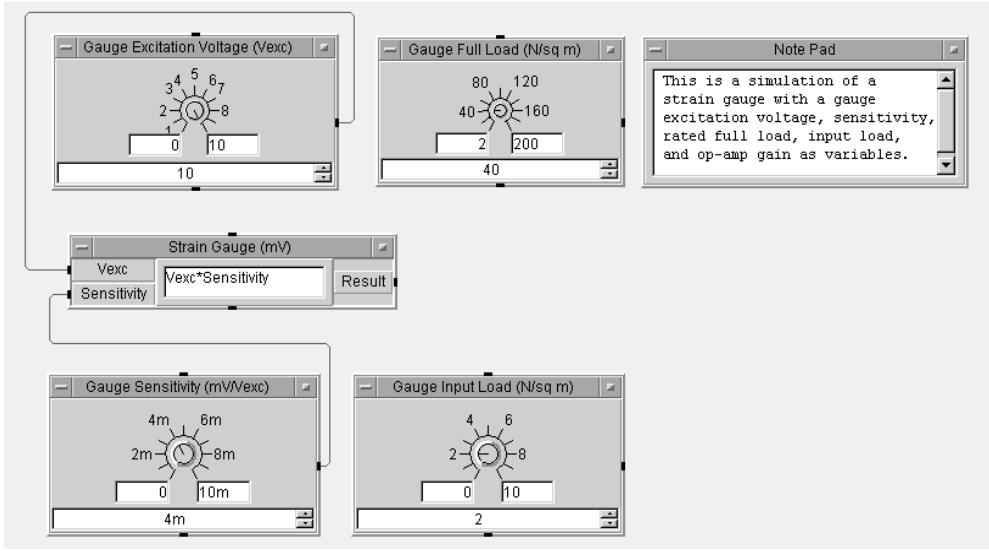


Figure 12.3. Strain Gauge simulation; initial construction

10. Connect the output pin of Gauge Excitation Voltage to the Vexc input pin of Strain Gauge Output; connect the output pin of Gauge Sensitivity to the Sensitivity input pin of Gauge Output. See Figure 12.3 to verify your connections.
11. Select Menu Bar => Device => Formula, place it the right of Strain Gauge Output.
12. Add two Data Input Terminals. Change the A data-input terminal name to: Full_Load, change the B data-input terminal name to Gauge_Output. Change its C data-input terminal name to Input_Load.
13. Change its formula to Gauge_Output*(Input_Load/Full_Load). Change its title bar to Strain Gauge Output (mV).
14. Connect the Gauge Full Load Object output pin to the input pin "Full Load" of the Strain Gauge Output Object.
15. Connect the output pin (Result) of Strain Gauge Output to the input pin "Gauge_Output" of the Strain Gauge Output Formula Object.
16. Connect the output pin of Gauge Input Load to the input pin "Input_Load" of Strain Gauge Output.
17. Select Menu Bar => Device => Formula, place it below Strain Gauge Output. Change its title bar to Op Amp with Gain K.
18. Change its formula to 1*A. Connect the Strain Gauge Output to the Op Amp input.
19. Select Menu Bar => Display => AlphaNumeric, place it below Op Amp. Change its name to Amplified Strain Gauge Output (V). Connect its input to the Op Amp output. Verify your Object positioning and connections using Figure 12.4.

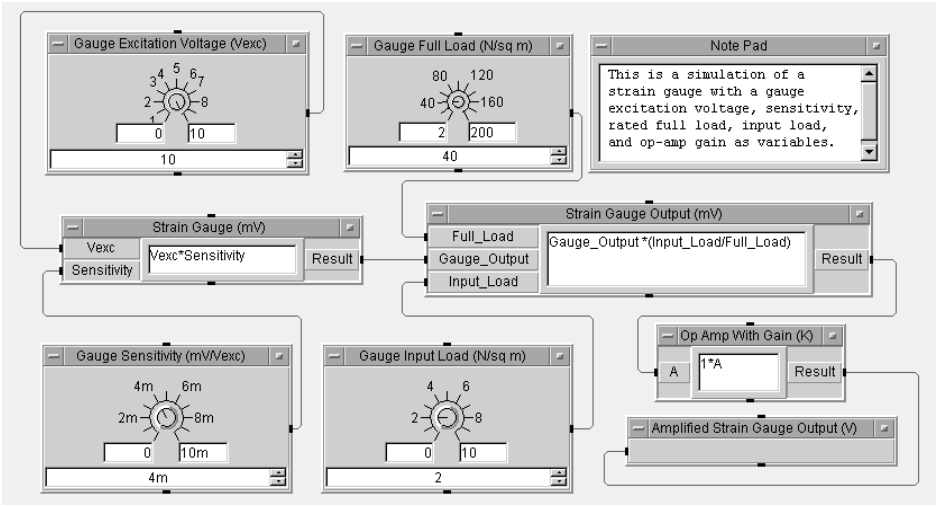


Figure 12.4. Strain Gauge simulation; total construction

20. Set the Gauge Excitation Voltage to its maximum value (10), the Gauge Full Load to 40, the Gauge Sensitivity to 4m and the Gauge Input Load to 2.
    Note: Your VEE Pro program may not allow you to set your values at exactly the values given; "close enough" is satisfactory.

21. Save your program as LAB 12-2; run this program; observe the Amplified Strain Gauge Output value, can you justify the small value of the result? Complete the following tables.

Gauge Excitation Voltage	Gauge Sensitivity	Strain Gauge (Result)
Ours: 10V	$4mV/V_{exc}$	40mV = 0.04V
Yours:		

Gauge Full Load	Gauge Input Load	Strain Gauge Output (Result)
Ours: 40 N/sq m	2N/sq m	0.002V
Yours:		

Strain Gauge Output (Result)	Op Amp Gain	Amplified Strain Gauge Output
Ours: 0.002 V	1	0.002 V = 2mV
Yours:		

Note: Are your observed values more understandable?

◊ 22. Vary a number of parameters; record their values in the tables below, including those that are not allowed by the program.

Gauge Excitation Voltage	Gauge Sensitivity	Strain Gauge (Result)

Gauge Full Load	Gauge Input Load	Strain Gauge Output (Result)

Strain Gauge Output (Result)	Op Amp Gain	Amplified Strain Gauge Output

23. What have you learned from these assumptions? Is your op amp output value large enough so you could connect it directly to a multiplexer? What values would you have to choose so it would provide an overall range from 0 to 5V full scale into the multiplexer? Record these values in the following table.

Gauge Excitation Voltage	Gauge Sensitivity	Strain Gauge (Result)
Gauge Full Load	**Gauge Input Load**	**Strain Gauge Output (Result)**
Strain Gauge Output (Result)	**Op Amp Gain**	**Amplified Strain Gauge Output**

24. Save and close this program as LAB12-2.

Lab 12.3 – Exploring Four Mechanical Simulations

This lab will show you how to access and explore four application programs embedded within VEE Pro.

1. Go to the hard-drive location where the program files are stored; open the Program Files folder. (You may use Ctrl F as an option to find the Program Files folder.)
2. Open, in the following sequence, the Agilent folder, the VEE Pro 6.2 folder, the examples folder, and the Applications folder.

Examining a cantilever beam deflection problem

3. Open Beam.vee; change to Detail View; click on the "Info About" icon and study its contents.
   Note: If you desire more information regarding the program content, then click on the third icon on the left side of the title bar. It will display: "To Detail" if you hold your mouse over that icon for a few moments.
4. Go to the Formula object labeled "length(numElem,from,thru)"; note the object Description.
5. Run this program; it should look like Figure 12.5.

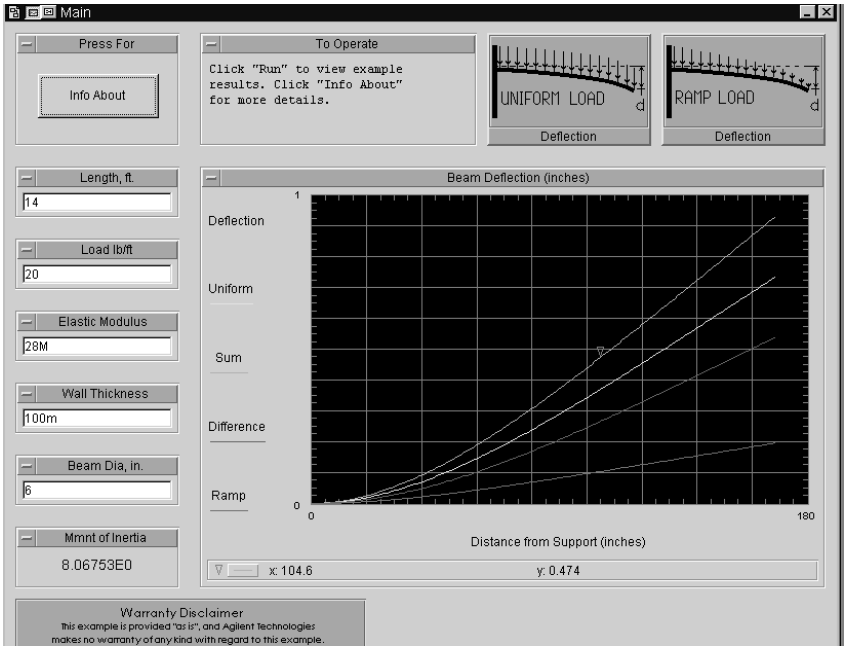


Figure 12.5. Beam.vee program after running

◊ 6. Change the parameter values such as beam length, elastic modulus, load, beam diameter, and wall thickness. Record the results in an Excel™ spreadsheet embedded in a Word™ report (if desired).
7. Close this program; rename and save it if you want your program modifications retained.

Examining the angular deflection (torsion) of a round shaft

8. Open torsion2.vee; change to Detail View; click on the "Info" icon and study its contents.
9. Go to the object labeled "Modulus of Elasticity/Rigidity"; note the description of this UserObject.
10. Repeat step 9 for "Young's Mod(ulus), Shear Mod(ulus), and whatever other icons are of interest.
11. Run this program; it should look like Figure 12.6.

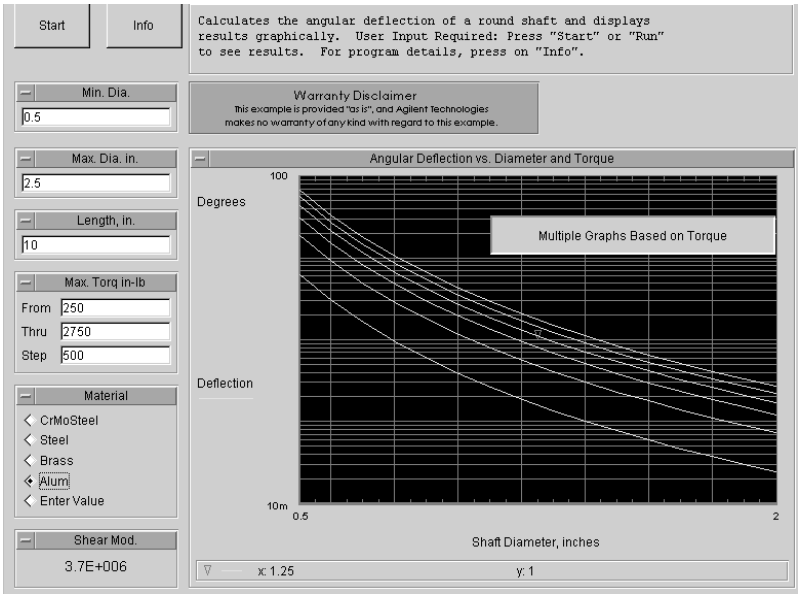


Figure 12.6. Deflection in degrees versus shaft diameter

◊ 12. Change the values of such parameters as pipe diameter (max and min), length, Max torque, and material type. Record the results in a Word™ report if desired.

13. Close this program; rename and save it if you want your program modifications retained.

Examining the natural frequency of a coil spring

14. Open coilspr.vee; change to Detail View; click on the "Info About" icon and study its contents. **Note:** There are no descriptions of interest within any of the icons.
15. Run this program; it should look like Figure 12.7.

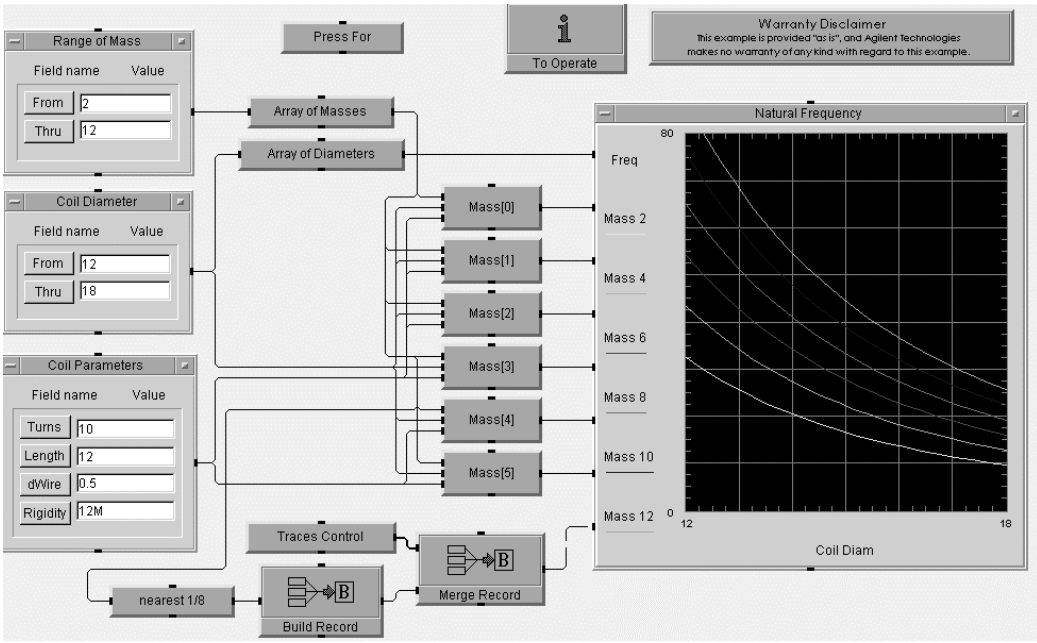


Figure 12.7. Display of coil spring natural frequencies

◊ 16. Change the values of such parameters as Range of Mass, Coil Diameter, and Coil Parameters (Turns, Length, dWire, and Rigidity). Record the results in a Word™ report if desired.

17. Close this program; rename and save it if you want your program modifications retained.

Examining the natural convection of heat

18. Open Convcoef.vee – The Natural Convection of Heat application; change to Detail View.
19. Click on the "Info About" icon and study its contents.
    Note: If you desire more information regarding the program content, then click on the third icon on the left side of the title bar. It will display: "To Detail" if you hold your mouse over that icon for a few moments.
20. Run this program. It will look like Figure 12.8.
    Note: The Flow Range display indicates that the "Flow is in Laminar Range". It will change to "Flow is in Turbulent Range" for other setting combinations.

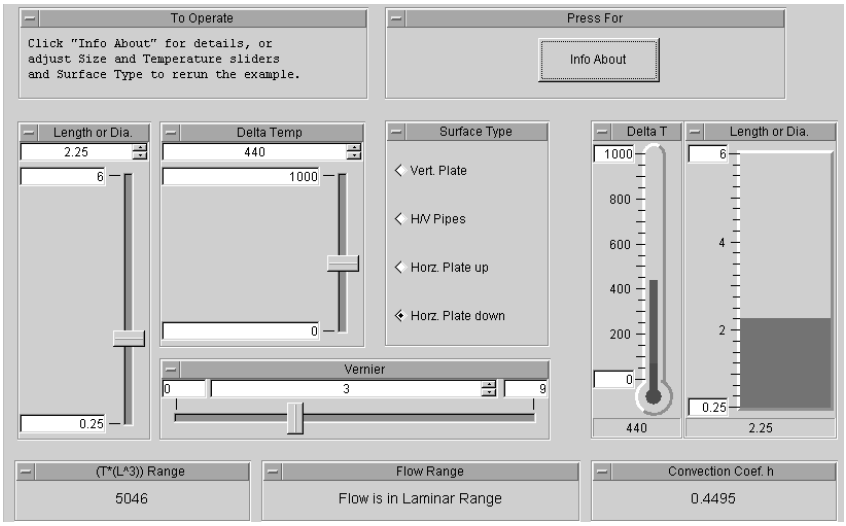


Figure 12.8. Convection Coefficient program after running

◊ 21. Change the values of such parameters as Length or diameter, Delta Temp, Surface Type, and Vernier. Transfer the "run" program to a Word™ report if desired via "Print Screen".
22. Close this program; rename and save it if you want your program modifications retained.

Lab 12.4 – Exploring a Simulated Manufacturing Test System

This lab will show you how to combine features of VEE Pro to build a manufacturing test system.

Examining a manufacturing test system program

1. Clear your Work Area, deselect the Program Explorer, and maximize the Work Area.
2. Go to the hard-drive location where the program files are stored; open the Program Files folder.
3. Open, in the following sequence, the Agilent folder, the VEE Pro 6.2 folder, the examples folder, and the Applications folder.
4. Open mfgtest.vee; change to Detail View; click on the "Info About" icon and study its contents. The program should look like Figure 12.9.

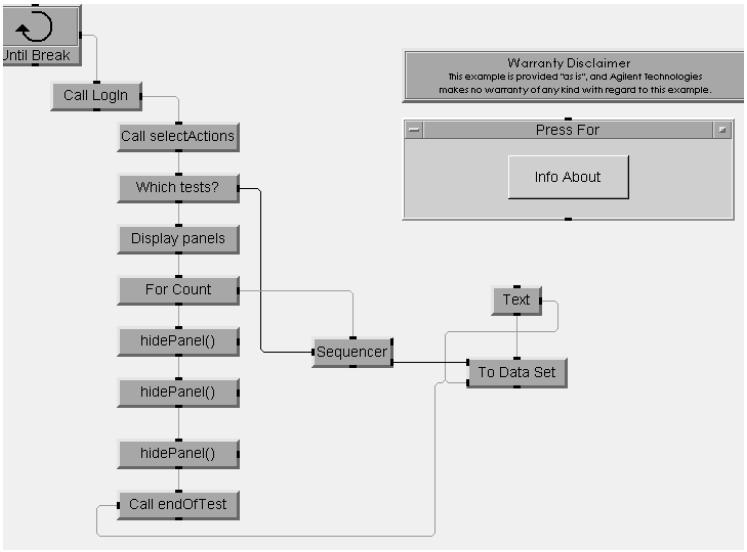


Figure 12.9. Program description for Mfgtest.vee

5. Run this program; select your login name from the list of 4; select the default password.
6. Examine the “run” program first step; it should look like Figure 12.10.

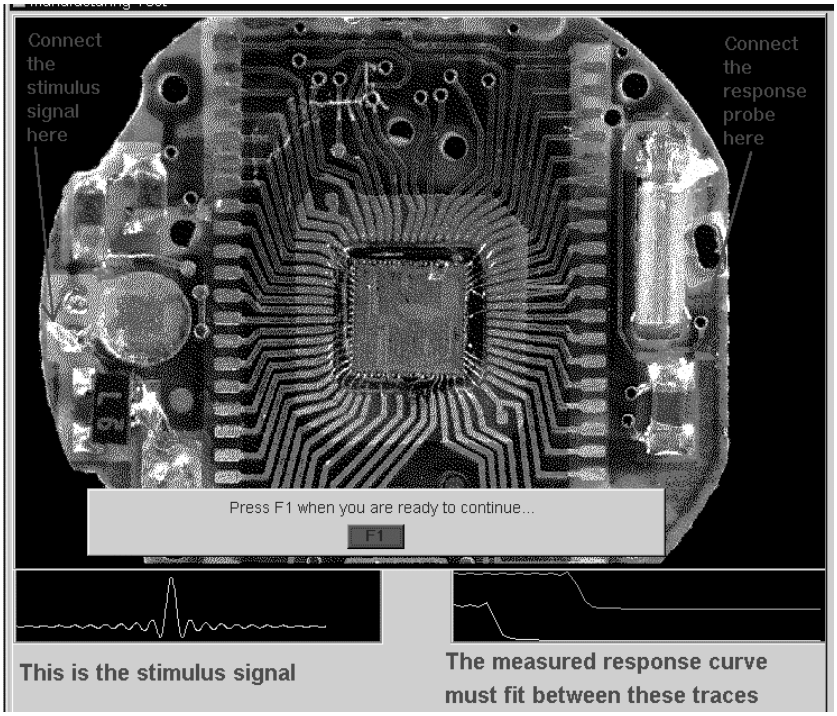


Figure 12.10. Mfgtest.vee illustrating the circuit under test

7. Observe the manufacturing tests cycling; turn off the program.
   Note: These tests are
   - Impulse Test
   - Power Consumption
   - Input Impedance
   - Output Impedance
   - Crossover Power
   - Crossunder Power
   - Gain
8. Examine the content and subprograms within the two dialog boxes: LogIn and Which Tests?
9. Close the mfgtest.vee program.

Lab 12.5 – Plotting Simulated Vehicle Radiator Temperatures via MATLAB®

In this lab you are to use MATLAB® to display changes in the Vehicle Radiator temperature distribution.

1. Go to your computer hard drive; type **Ctrl f** (the "find" command); enter Surf3Dplot.vee which will locate the MATLAB® graphing program.
2. Double-click on the icon displayed.
3. Save this program in your lab library as LAB12-5.

Applying MATLAB® example displays to an existing Vehicle Radiator application

4. Change the title of the Function Generator to read: Vehicle Radiator Temperature.
5. Change the following parameters in the Vehicle Radiator Temperature Object:
   Function: Tri
   Frequency: 50
   DcOffset: 190
6. Cut Warranty Disclaimer and Note Pad.
7. Open Alloc Real64; change Dim 1 from 41 to 61.
8. Change the upper For Range parameter Thru from 1 to 15; Step from 50m to 0.5.
9. Change the lower For Range parameter From 0.95 to 14.5; Step from –50m to –0.5
10. Change MATLAB® Script.PLOT as follows:
    axis ([0, 50, -1, 1]) to ([0, 50, 180, 200]) with shown spaces within the brackets.
11. Change MATLAB® Script.SURF as follows:
    axis ([0, 50, 0, 60, -1, 1]) to ([0, 50, 0, 60, 180, 200])
12. Save this program again; arrange the icons as shown in Figure 12.11.

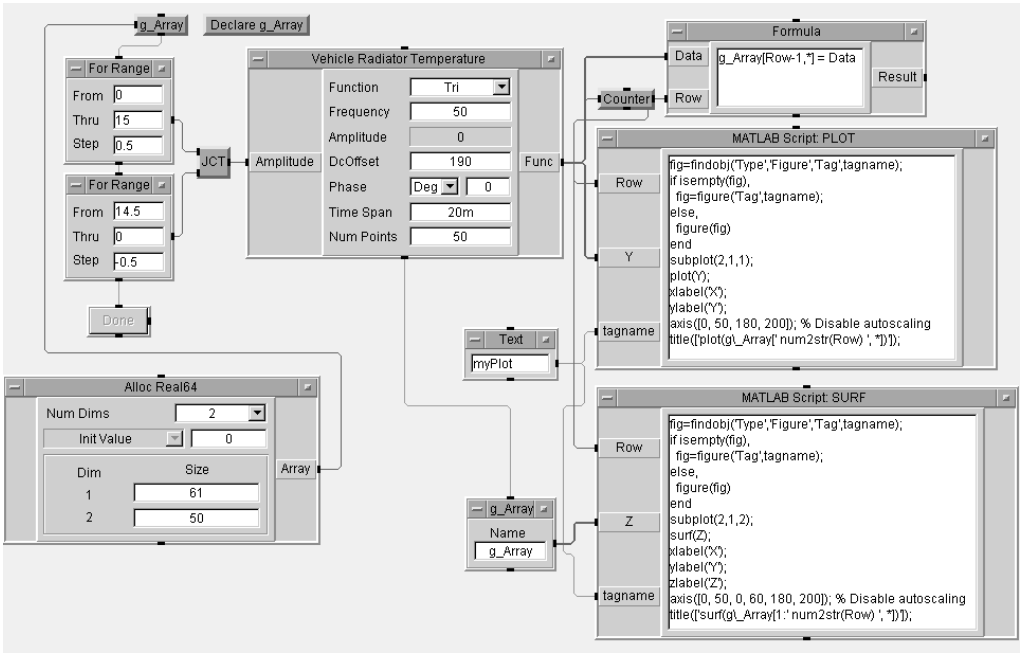


Figure 12.11. Lab 12.5 before running

13. Run this program; the result should look like Figure 12.12.
    Note: First a straight horizontal line will appear at the top. This line gradually becomes a triangular wave that generates a second plot that becomes the figure shown below.

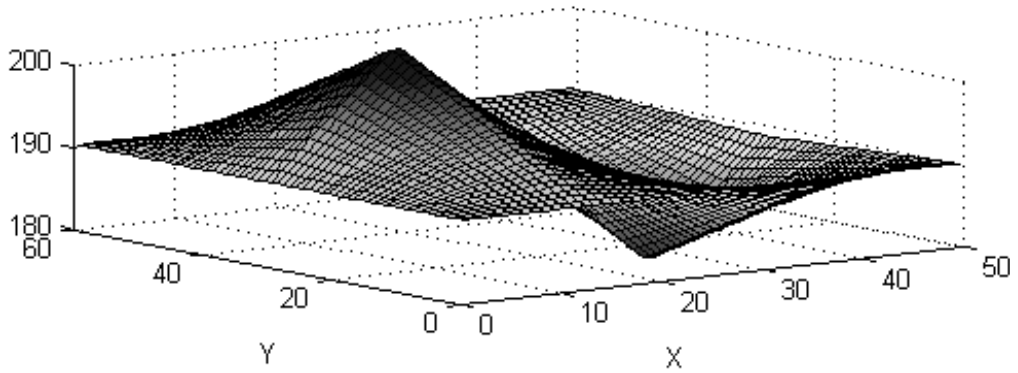


Figure 12.12. A three-dimensional plot of Vehicle Radiator data

14. Go to the upper-right corner of the Figure #1 Tool Bar; click on: Rotate 3D.
15. Place the cursor on the right corner of the 3D plot; turn it in small steps until the peaks are beyond the 180 and 200 Z scale. (This indicates that the scale of the plot is greater than the scale of the display.)
    Note: Changing parameters within this program requires knowledge of both MATLAB® and VEE.
16. Turn the plot until it becomes "two dimensional". Two white spaces will appear; they indicate that the data exceeds the scale of the plot. See Figure 12.13.

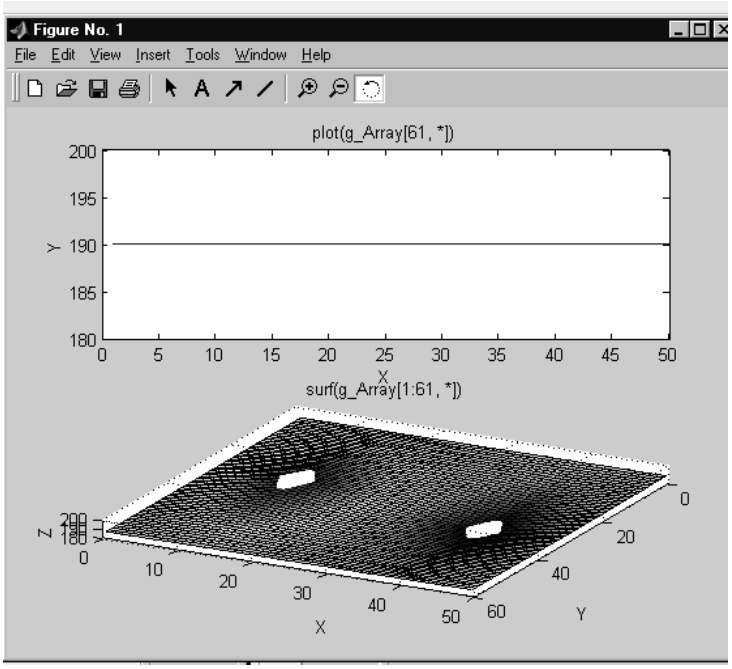


Figure 12.13. Vehicle Radiator temperature rotated 3D graph

17. Change MATLAB® Script.PLOT as follows:
    axis ([0, 50, -1, 1]) to ([0, 50, 170, 210]) with shown spaces within the brackets.
18. Change MATLAB® Script.SURF as follows:
    axis ([0, 50, 0, 60, -1, 1]) to ([0, 50, 0, 60, 170, 210])
19. Rotate the plot to verify that all points are within the graph.
20. Save this program as LAB12-5a; close this program.

Lesson 12 Summary

You have used VEE Pro to simulate a variety of applications. The first two of these labs focused upon unique, derived simulations. The second two of these labs focused upon VEE Pro-provided applications.

Lab 12.1 showed you how to generate a differential signal with an ideal instrumentation amplifier, examine the ability of an IA to measure a small signal buried in noise, and change the differential gain of the IA by a factor of ten.

Lab 12.2 showed you how to construct a four-element simulated strain gauge using simulated strain-gauge resistors and a Wheatstone bridge.

Lab 12.3 showed you how to access and explore four application programs embedded within VEE Pro. These applications are the cantilever beam deflection, angular deflection of a round torsion shaft, the natural frequencies of a coil spring, and heat transfer coefficients for several body or surface types.

Lab 12.4 showed you how to combine features of VEE Pro to build a manufacturing test system.

Lab 12.5 showed you how to use MATLAB® to display changes in the Vehicle Radiator temperature distribution on a three-dimensional plot.

You are now ready to simulate virtual functions and relations, and explore a VEE-prepared program for a filter.

Lesson 13
Functions, Relations, and Filtering

This lesson will examine how to devise and test functions and relations. It consists of a pre-lab and five labs.

Lesson 13 Pre-lab Summary

The following are described in the Lesson 13 pre-lab. See Appendix E, page E-69.

- Menu Bar => Display Spectrum (Freq)
- The Time and Frequency Domains
- Periodic Waveforms and Their Fourier Series

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 13.1 – Comparing Time-Domain and Frequency-Domain Waveforms
This lab will show you how to configure a waveform generator and how to compare the output of that generator to both a virtual time-domain and a virtual frequency-domain display.

Lab 13.2 – Creating a Square Wave
This lab will show you how to configure several waveform generators and how to connect these generators to simulate a square wave.

Lab 13.3 – Creating a Triangular Wave
This lab will show you how to configure several waveform generators and connect these generators to simulate a triangular wave.

Lab 13.4 – Creating a Trapezoidal Wave
This lab will show you how to configure several waveform generators and connect these generators to simulate a trapezoidal wave.

Lab 13.5 – Examining the Square Wave Power Spectrum with MATLAB®
This lab will show you how to apply MATLAB® to simulate the frequency and power spectrum for a square wave.

To the Student: When you learn to write, develop, apply, and explore programs, you will be of greater value to both yourself and your company.

Lab 13.1 – Comparing Time-Domain and Frequency-Domain Waveforms

This lab will show you how to configure a waveform generator and connect that generator to both a virtual time-domain oscilloscope and a virtual spectrum analyzer. You will then compare the output of several time-domain waveform generator combinations with both a time-domain and frequency-domain display.

Open the VEE program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Devising time-domain and frequency-domain monitors for a time-domain generator

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of Main; change its Num Points to 4096; clone the Function Generator and place it below the first Function Generator.
   Note: This increase in the number of sample points will cause the spectrum analyzer to more accurately present the monitored spectrum.
3. Select Menu Bar => Display => Waveform (Time); place it in the upper-right corner of Main; shrink it vertically so it consumes approximately one-half the vertical area of the screen; lengthen it horizontally so it reaches the right-hand edge of the screen.
4. Select Menu Bar => Display => Spectrum (Freq) = > Magnitude Spectrum; place it in the bottom-right corner of Main; shrink it vertically so it consumes approximately one-half the vertical area of the screen; lengthen it horizontally so it reaches the right-hand edge of the screen.
5. Select Menu Bar => Device => Formula; place it to the right of the function generators, add an input terminal to Formula and change its expression to A*B; connect the output of each Function Generator (Func) to the inputs (Trace1) of the two displays.
6. Change the top Function Generator Function to Sine; change the bottom Function Generator to Cosine, its frequency to 900, and its amplitude to 0.4.
7. Save your program as LAB13-1.
8. Run this program; move the horizontal axis on both displays to the right so you can more accurately determine the period of the time-domain waveform and the frequency of the frequency-domain waveform; turn the frequency-domain horizontal-axis Automatic Scaling Off; change its Maximum scale setting to 1200. Record the results and remarks in the table on page 13-6 for these *and the waveforms below*. See Figure 13.1.

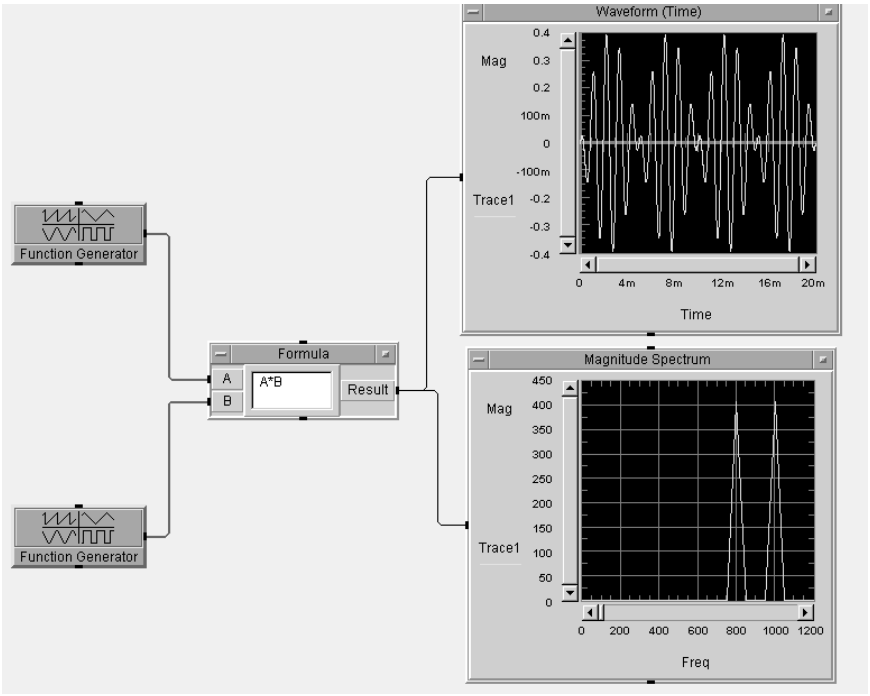


Figure 13.1. A frequency modulation wave display

◊ 9. Change the top Function Generator Function to Cosine; run your program again. What differences do you observe? Record the results and remarks in the table on page 13-6 for these *and the waveforms below*.

◊ 10. Select Square from the Function Generator Function list; change the Spectrum Analyzer setting to 2000; change the Formula equation to A+B; change the bottom Function Generator amplitude to zero; run your program again. What differences do you observe? Can you explain these differences? See Figure 13.2. Record the results and remarks in the table on page 13-6 for these *and the waveforms below*.

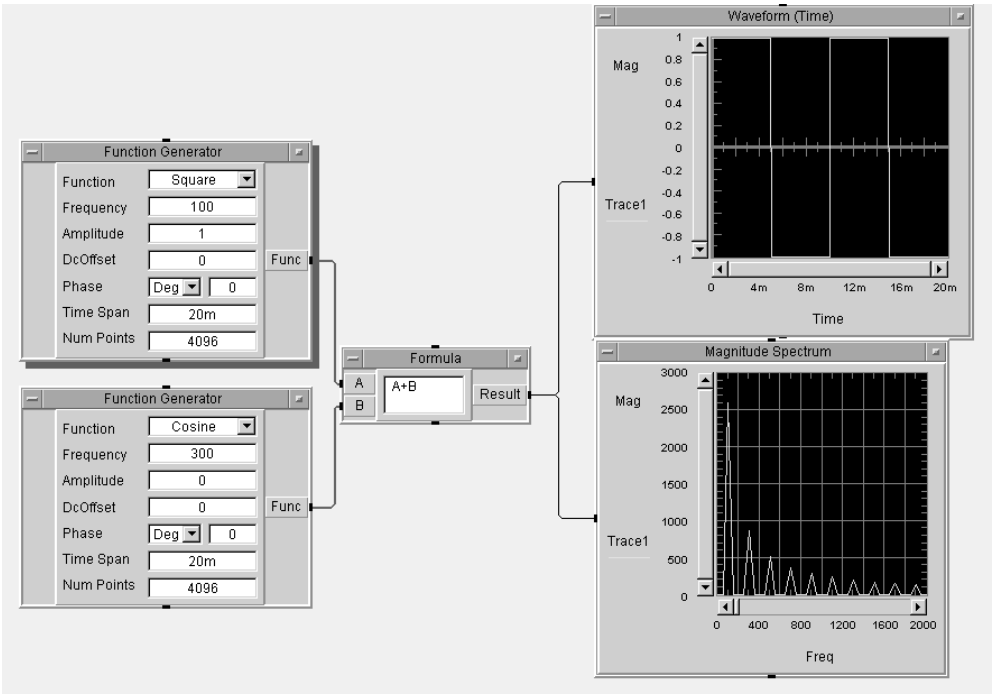


Figure 13.2. A square wave and its frequency spectrum

11. Save this modified program as LAB13-1a.
◊ 12. Select Tri from the Function Generator Function list; run your program again. What differences do you observe?
◊ 13. Select +Ramp from the Function Generator Function list; run your program again. What differences do you observe?
◊ 14. Select -Ramp from the Function Generator Function list; run your program again. What differences do you observe?
15. Save this modified program as LAB13-1b.
16. Replace the top Function Generator with a Pulse Generator; change the Num Points to 4096.
17. Save As…LAB13-1c immediately.
18. Run your modified program again; what differences do you observe? See Figure 13.3. Record the results and remarks in the table on page 13-6 for these *and the waveforms below*.

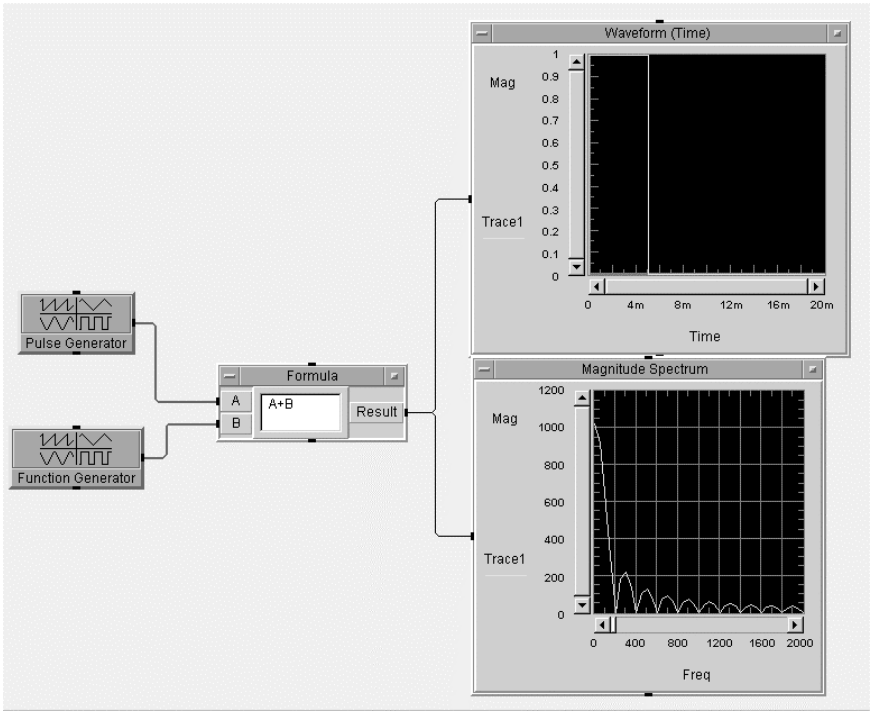


Figure 13.3. Effect of pulse width on the frequency spectrum

◊ 19. Change the Pulse Generator Pulse Width from 5m to 1m; change the Spectrum Analyzer Freq maximum to 10 000; run this program; what differences do you observe? Very carefully compare the two frequency-domain changes in frequency distribution.

20. Replace the Pulse Generator with a Noise Generator; change the Num Points to 4096; cut the remaining Function Generator and the Formula icon; connect the Noise Generator directly to the two displays.

21. Save As…LAB13-1d immediately.

◊ 22. Change the Noise Generator Num Points to 32; run this program; what differences do you observe?

◊ 23. Change the Noise Generator Num Points to 1024; run this program; what differences do you observe?

◊ 24. Change the Noise Generator Num Points to 4096; save and run this program; what differences do you observe? Very carefully compare the two frequency-domain changes in frequency distribution.
Note: For a perfect frequency distribution of noise, the spectrum would have a constant height.

25. Complete this matrix of Time Domain waveform type versus Frequency Domain waveform type in Table 13.1; include notes in the Remarks column where appropriate.

Table 13.1. Comparison of time-domain and frequency-domain waveforms

Time Domain	**Frequency Domain** **Max Heights & Frequencies**	**Remarks** **Peaks Damped Exponentially?**
Sine		
Cosine		
Square		
Triangular		
+Ramp		
-Ramp		
Pulse Width:_____ _____		
Pulse Width:_____ _____		
Noise; Num Points:_____ _____		
Noise; Num Points:_____ _____		

26. Repeat the above Lab steps until you are satisfied with your waveform comparison effort.
27. Return to LAB13-1.
28. Save As…LAB13-1e immediately.
29. Change the Formula expression to A+B.
30. Change the bottom Frequency Generator to Sine.
31. Set the Frequency of the bottom Function Generator to 300.
32. Set the Spectrum Analyzer maximum Freq to 500.

◊ 33. Run this program several times with different amplitude settings on each Function Generator. See Figure 13.4.

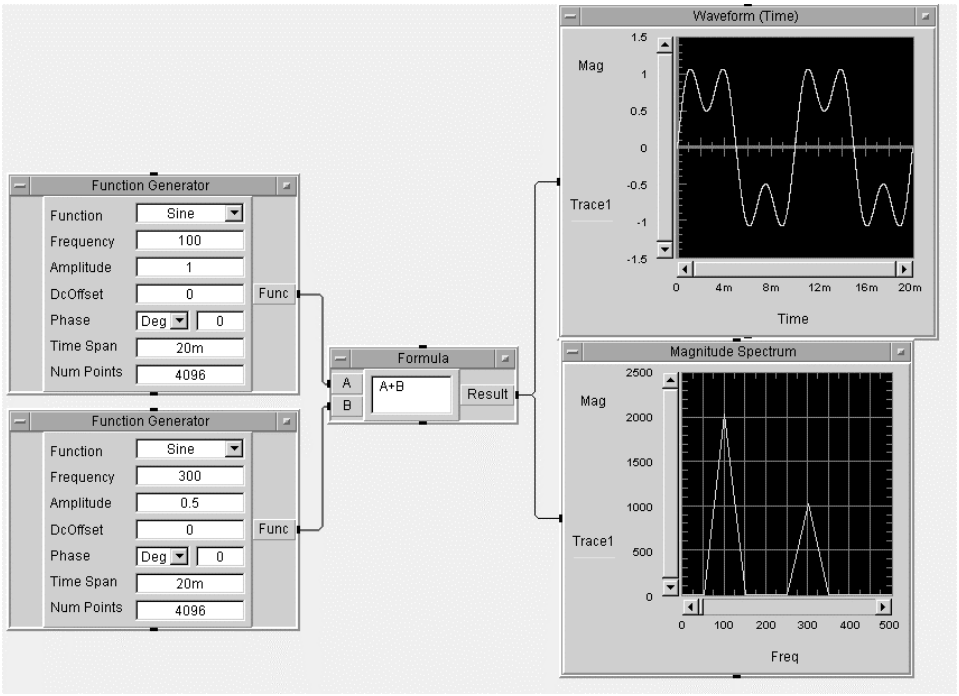


Figure 13.4. One pair of Function Generator settings

34. Change the Formula expression from A+B to A*B again. See Figure 13.5.
    Note: You are now changing from waveform addition (A.M.) to modulation (F.M.).

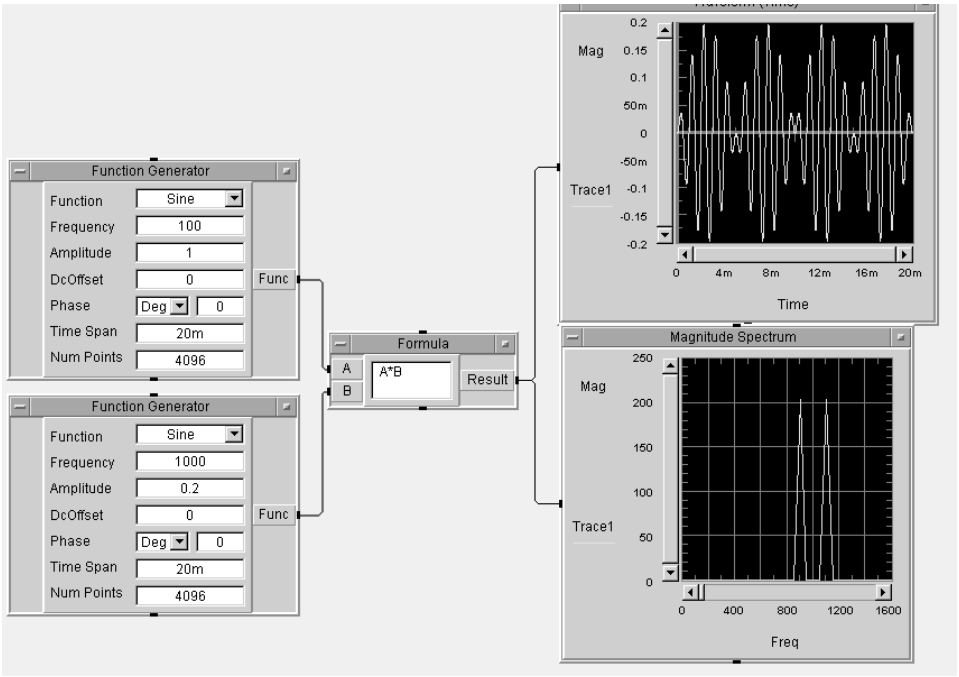


Figure 13.5. Effect of modulation on time and frequency domains

◊ 35. Save your program again; run this program with several different amplitude settings; note the change in both the time-domain display and the frequency-domain display.

◊ 36. Change the frequency setting of the bottom Function Generator to 900 and the Spectrum Analyzer Freq Maximum to 2000; run this program.
Note: The time-domain waveform should look more like a modulated wave.

◊ 37. Change the frequency setting of the bottom Function Generator to 2700 and the Spectrum Analyzer Freq Maximum to 6000; run this program.
Note: You are now observing the effects of modulation on especially the time-domain display. Also, the frequency-domain magnitude display should not change.

38. Shrink the vertical size of each Function Generator so you can add a third one under the other two; add the third Function Generator by cloning one of the other two.
39. Add a third input terminal to the Formula Object; change its equation to A+B+C.
40. Connect the third Function Generator to the Formula C terminal input.
41. Change the three Function Generator frequencies 100, 300, and 500 and their respective amplitudes to 10, 5, and 2; change the FREQ scale of the Spectrum Analyzer to 1000.
42. Save As…LAB13-1f immediately.
43. Run this program. See Figure 13.6.

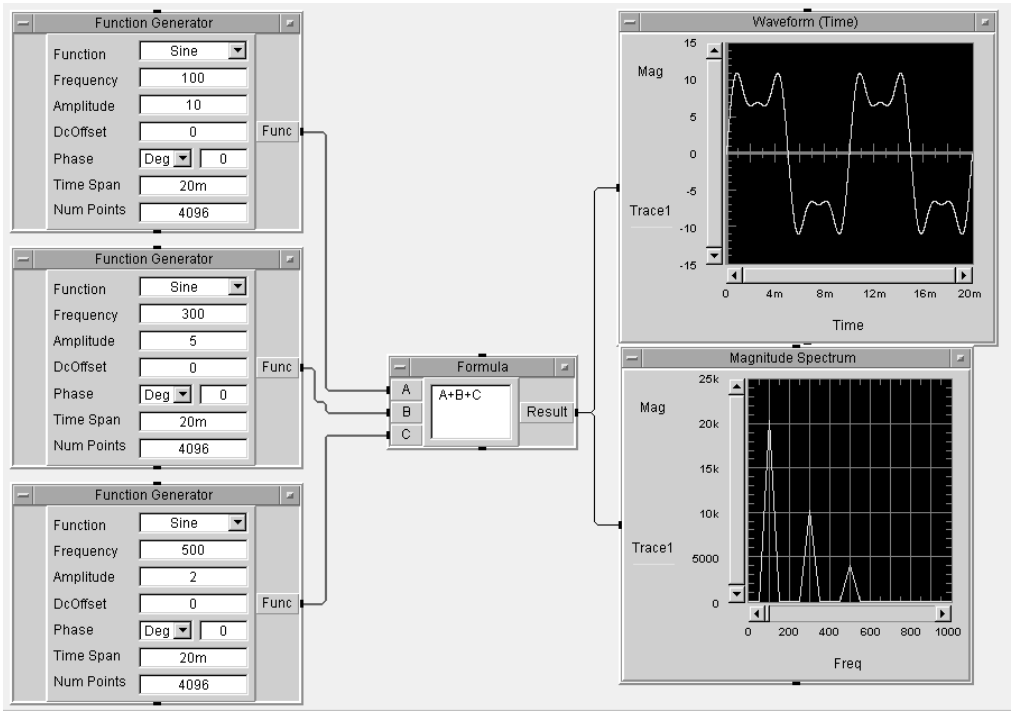


Figure 13.6. Spectra for three added sine-shaped waveforms

◊ 44. Vary the Phase of each of the three function generators; observe the lack of change in the frequency-domain display and observe the significant change in the time-domain display.
Note: You are simulating the effects of sine-wave distortion that may occur in both the electrical power being supplied to a power-supply and the distortion of output signals being "fed" to an RF modulator (such as a modem or transmitter) for information-transmission purposes.

45. Save this modified program again.

Lab 13.2 – Creating a Square Wave

This lab will show you how to configure several waveform generators and connect those generators to a virtual time-domain oscilloscope via a summing formula object. You will then program those generator outputs to simulate a square wave.

Open the VEE program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Devising a virtual square wave

2. Select Menu Bar => Device = > Virtual Source => Function Generator, set Function to Sine, Frequency to 100, and Amplitude to 1; place it in the upper-left corner of Main.

3. Change the name of the Function Generator to Fundamental.
4. Select Menu Bar => Device => Formula, edit the expression in the white space to read: A+B+C+D+E+F+G+H+I+J+K+L; place it in the center of the Work Area; add eleven data input terminals labeled B, C, D, E, F, G, H, I, J, K, and L.
5. Select Menu Bar => Display => Waveform (Time); place it in the upper-right corner of the Work Area.
6. Select Menu Bar => Display => Spectrum (Freq) => Magnitude Spectrum; place it in the lower-right corner of the Work Area, directly under Waveform (Time); click on Freq; turn its Automatic Scaling Off; change its Maximum to 2500, its Minimum to zero, and its Label Spacing to Every Other Tic.
7. Connect the Formula output terminal to the Waveform (Time) and Spectrum (Freq) input terminals, each labeled Trace 1.
8. Connect the Fundamental output terminal to Formula Input terminal A.
9. Clone Fundamental eleven times; change the settings and connect to Formula according to Table 13.2; also change the generator names (via each Properties box) to agree with each harmonic's numerical value. See Lesson 13 Pre-lab, page E-71.

Table 13.2. Virtual square-wave settings

Harmonic	Frequency	Amplitude	Formula Connection
Third	300	1/3	B
Fifth	500	1/5	C
Seventh	700	1/7	D
Ninth	900	1/9	E
Eleventh	1100	1/11	F
Thirteenth	1300	1/13	G
Fifteenth	1500	1/15	H
Seventeenth	1700	1/17	I
Nineteenth	1900	1/19	J
Twenty-first	2100	1/21	K
Twenty-third	2300	1/23	L

10. Save this program as LAB 13-2.
11. Run this program; it should look like Figure 13.7.

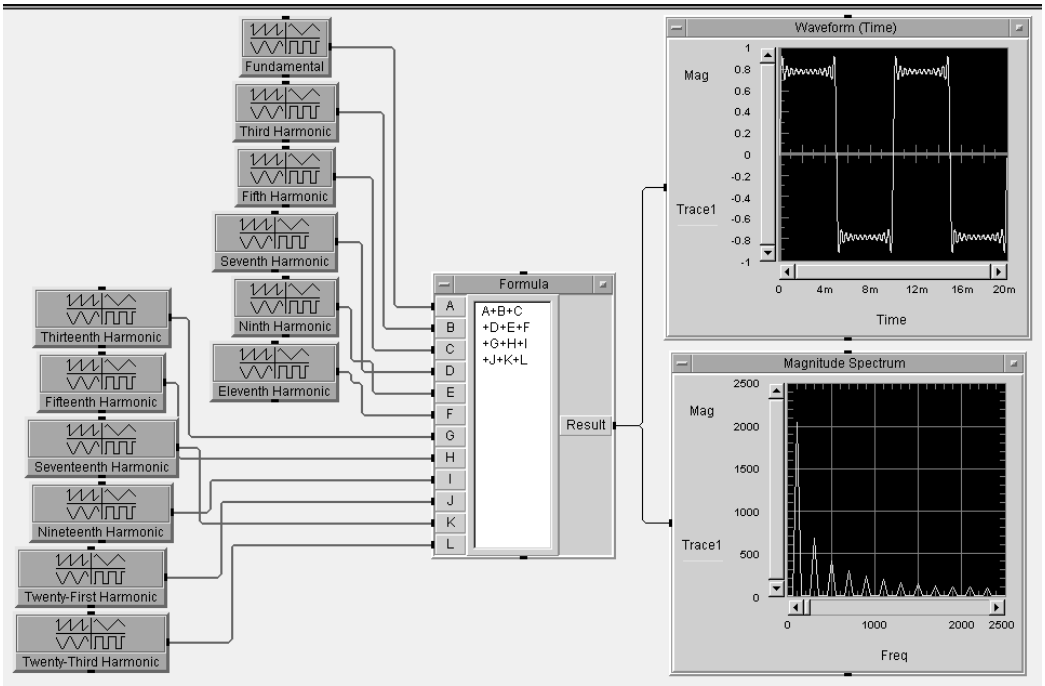


Figure 13.7. A square wave generated by adding twelve sine waves

Changing the number of added sine waves for a wave shape display

12. Reduce some of the larger harmonic amplitudes to zero; run your program again. What changes do you notice in the shape of the "square" wave?

Your notes:

_____

_____

_____

_____

_____

◊ 13. Vary the number of zero amplitudes; observe the changes in square-wave quality.

14. Exit this program; do not save the changes.

Lab 13.3 – Creating a Triangular Wave

This lab will show you how to configure several waveform generators and connect those generators to a virtual time-domain oscilloscope via a summing formula object. You will then program those generator outputs to simulate a triangular wave.

Open your VEE program and

1. Clear Main, deselect the Program Explorer.

Devising a virtual triangular wave

2. Open LAB13-2 immediately; Save As… re-naming it: LAB13-3.
3. Open each Function Generator; change all functions to Cosine.
4. Save this program again.
5. Run this program; it should look like a distorted triangular wave because the amplitudes have not been corrected.
6. Revise each harmonic generator according to Table 13.3; the Frequency values are given so you can verify that you have opened the correct harmonic generator. See Lesson 13 Pre-lab, page E-73.

Table 13.3. Amplitude values for a simulated triangular wave

Harmonic	Frequency	Amplitude
Third	300	1/9
Fifth	500	1/25
Seventh	700	1/49
Ninth	900	1/81
Eleventh	1100	1/121
Thirteenth	1300	1/169
Fifteenth	1500	1/225
Seventeenth	1700	1/289
Nineteenth	1900	1/361
Twenty-first	2100	1/441
Twenty-third	2300	1/529

7. Run this program; it should look like Figure 13.8.
   Note: This waveform looks very much like a periodic triangular waveform because it is a function with only one vertical value for each horizontal value; it is *not* a relation.

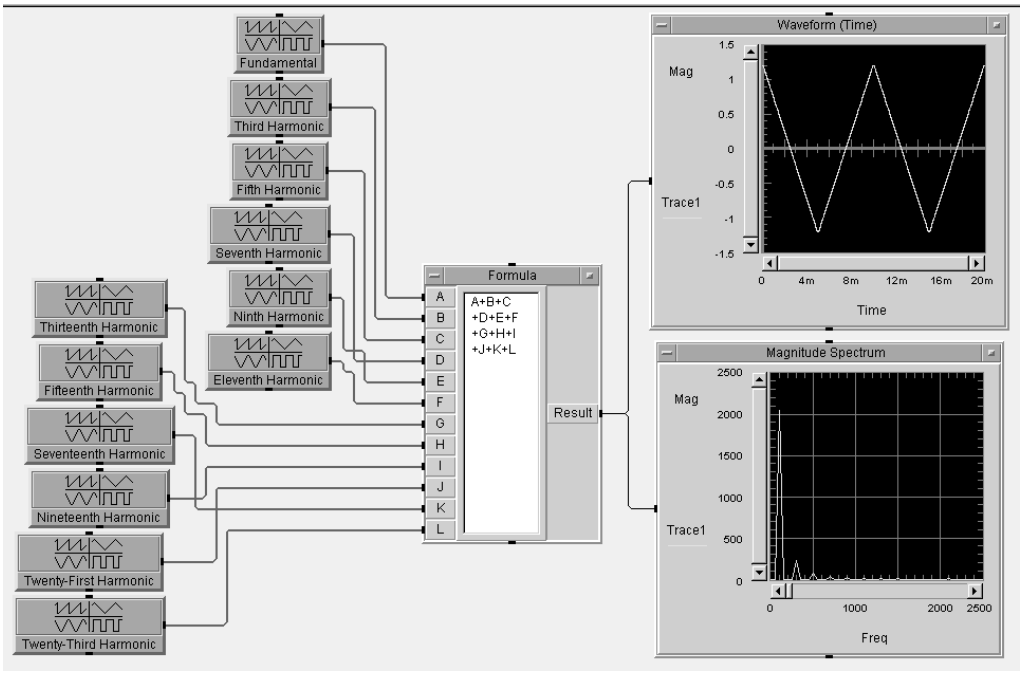


Figure 13.8. Simulation of a triangular wave

8. Save this program again.

Changing the number of added cosine waves for a wave shape display

9. Reduce some of the larger harmonic amplitudes to zero; run your program again. What changes do you notice in the shape of the "triangular" wave?

Your notes:

_____

_____

_____

_____

_____

◊ 10. Vary the number of zero amplitudes; observe the changes in triangular-wave quality.

11. Exit this program; do not save the changes.

Lab 13.4 – Creating a Trapezoidal Wave

This lab will show you how to configure several waveform generators and connect those generators to a virtual time-domain oscilloscope via a summing Formula Object. You will then program those generator outputs to simulate a trapezoidal wave.

Open your VEE program and

1. Clear Main, deselect the Program Explorer.

Devising a virtual trapezoidal wave

2. Open LAB 13-3 immediately Save As…, re-naming it: LAB 13-4.
3. Save this program again.
4. Run this program; it should look like a triangular wave because the signs of the amplitudes have not been corrected.
5. Verify that each harmonic generator's Amplitude agrees with Table 13.4; the Frequency values are given so you can verify that you have opened the correct harmonic generator. See Lesson 13 Pre-lab, page E-73.

Table 13.4. Amplitude values for a simulated trapezoidal wave

Harmonic	Frequency	Amplitude
Third	300	1/9
Fifth	500	1/25
Seventh	700	1/49
Ninth	900	1/81
Eleventh	1100	1/121
Thirteenth	1300	1/169
Fifteenth	1500	1/225
Seventeenth	1700	1/289
Nineteenth	1900	1/361
Twenty-first	2100	1/441
Twenty-third	2300	1/529

6. Change the signs of the Formula Object so they agree with the following:
   A-B-C+D+E-F-G+H+I-J-K+L to simulate a trapezoidal wave.
   See Lesson 13 Pre-lab, page E-73.
7. Run this program; it should look like Figure 13.9.
   Note: This waveform looks very much like a periodic trapezoidal waveform because it is a function with only one vertical value for each horizontal value; it is *not* a relation.

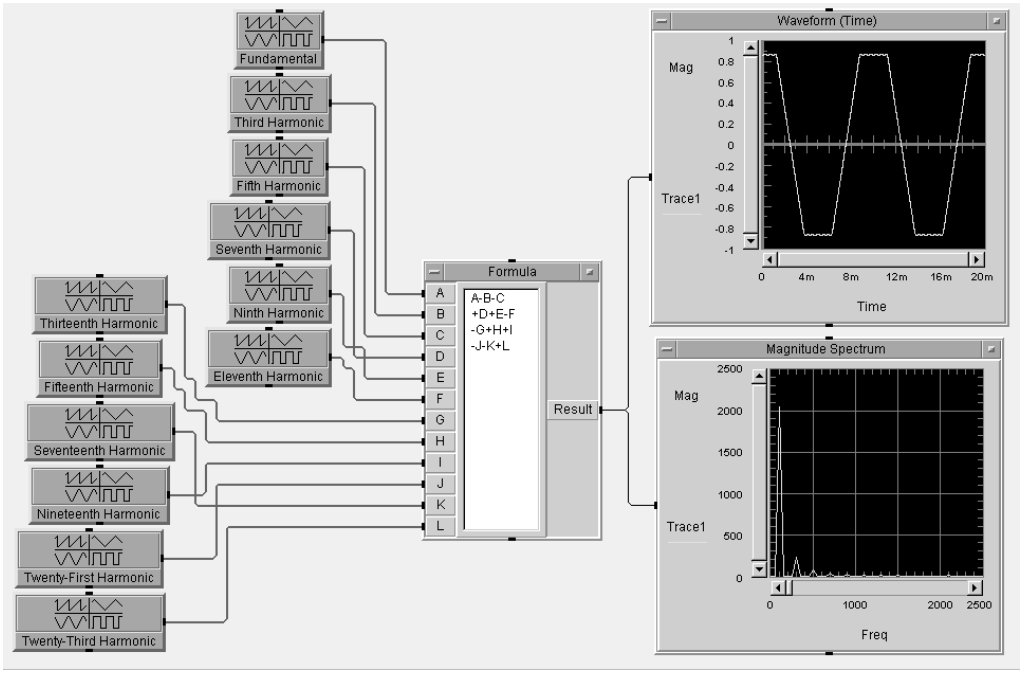


Figure 13.9. Simulation of a trapezoidal wave

8. Save this program again.

Changing the number of combined cosine waves for a wave shape display

◊ 9. Reduce some of the larger harmonic amplitudes to zero; run your program again. What changes do you notice in the shape of the "trapezoidal" wave?

Your notes:

_____

_____

_____

_____

_____

◊ 10. Vary the number of zero amplitudes; observe the changes in trapezoidal-wave quality.

11. Exit this program; do not save the changes.

Lab 13.5 – Examining the Square Wave Power Spectrum with MATLAB®

This lab will show you how to apply MATLAB® to simulate the frequency and power spectrum for a square wave.

Open the VEE program and

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Devising a virtual square wave via MATLAB®

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it upper left corner of Main; change the properties as follows:
   Function: Square
   Frequency: 100
   Num Points: 100
3. Select Menu Bar => Data => Unbuild Data => Waveform; place it to the right of Function Generator.
4. Connect the output of Function Generator to the input (WF Data) of Unbuild Waveform.
5. Select Menu Bar => Display => Waveform (Time); place it below Function Generator; connect its input (Trace) to the output of Function Generator.
6. Open Time; turn off its Automatic Scaling; change its maximum value to 20m; close Edit Time.
7. Open Mag; turn off its Automatic Scaling; change its maximum value to 1.2 and its minimum value to -1.2; close Edit Mag.
8. Select Menu Bar => Device => MATLAB Script; place it to the right of Unbuild Waveform.
9. Change the top input-terminal name (A) of MATLAB Script to “data” and its bottom input-terminal name (B) to “time”; delete its output terminal (X).
10. Enlarge MATLAB Script and enter the following program into its formula expression space:

```
%Establish length of data = N
N=length(data);
%Transform data from time domain to frequency domain
Y=abs(fft(data)/(N));
%Calculate sample frequency: fs
fs=N/time(1);
%Calculate the harmonic number (freq) of the transformed data
freq=[0:fs/N:fs-1/N];
%Plot amplitude versus frequency
half = floor(N/2);
plot(freq(1:half),Y(1:half));
xlabel(‘Frequency in Hz’);
ylabel('Amplitude');
title('Power Spectrum');
```

    Note: The “half” avoids displaying the Nyquist reflections of the presentations.
11. Connect the Unbuild Waveform terminal “Real 64Ary” to the MATLAB script input terminal “data” and the Unbuild Waveform terminal Time Span to “time”.
12. Select Menu Bar => Flow =-> Confirm (OK); place it below MATLAB Script; change its name to “Done”. See Figure 13.10.

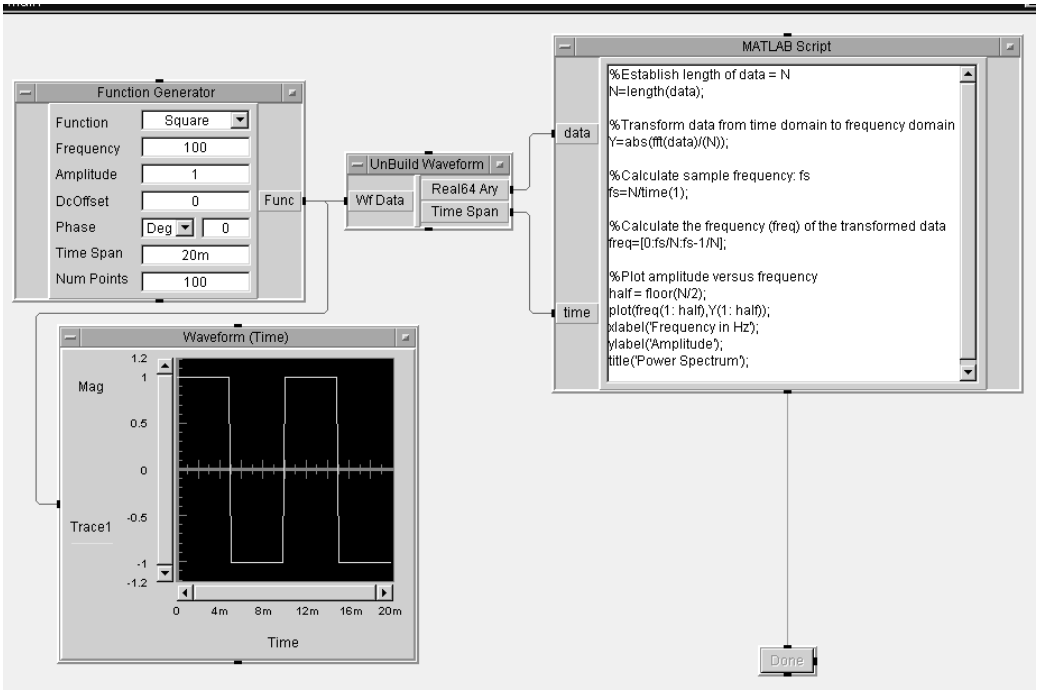


Figure 13.10. Lab 13.5 before running

13. Run this program; it should look like Figure 13.11.

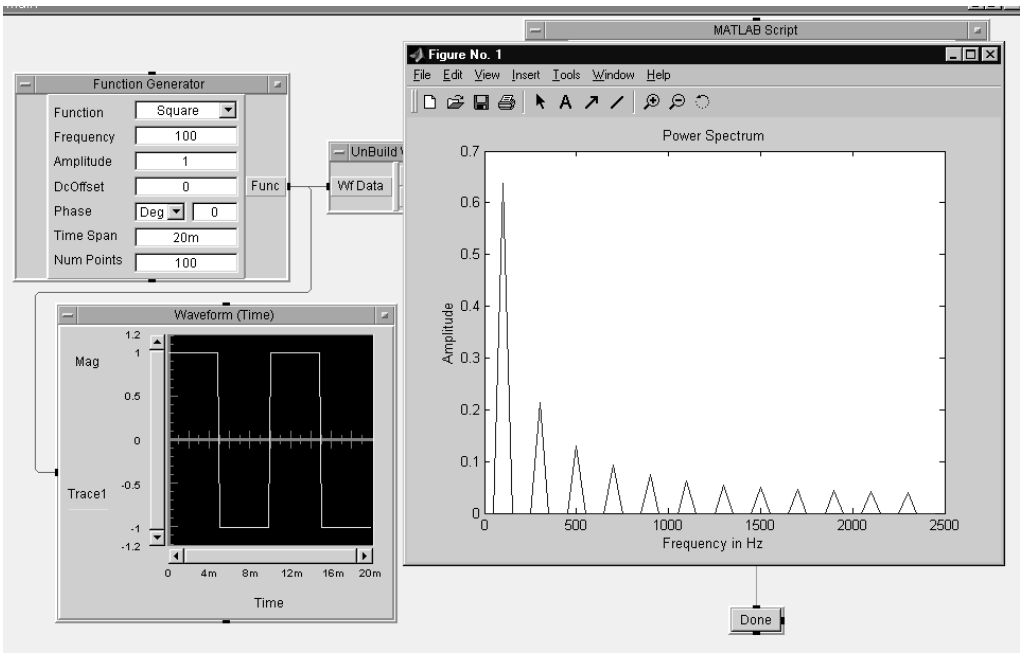


Figure 13.11. Lab 13.5 after running with MATLAB® plot

14. Stop this program by either clicking on the Done button or by clicking on the Stop button.
15. Save your program as LAB13-5; close this program.

Lesson 13 Summary

This lesson showed you how to devise and test functions and relations. The Appendix E pre-lab also reviewed filter theory for those of you who design analog and digital filters.

Lab 13.1 showed you how to configure a waveform generator and how to compare the output of that generator to both a virtual time-domain and a virtual frequency-domain display.

Lab 13.2 showed you how to configure several waveform generators and how to connect these generators to simulate a square wave.

Lab 13.3 showed you how to configure several waveform generators and connect these generators to simulate a triangular wave.

Lab 13.4 showed you how to configure several waveform generators and connect these generators to simulate a trapezoidal wave.

Lab 13.5 showed you how to apply MATLAB® to simulate the frequency and power spectrum for a square wave.

You are now ready to learn to use and apply VEE Library Functions.

Lesson 14
Using and Applying VEE Pro Library Functions

This lesson will examine how to use the VEE Pro library of UserFunctions and how to apply them to your newly designed programs. It will also show you how to develop an additional application and to record test data on spreadsheets. It consists of a pre-lab and four labs.

Lesson 14 Pre-lab Summary

The following are described in the Lesson 14 pre-lab. See Appendix E, page E-76.

- Using Library Functions
- Generating Lissajous Patterns
- The Junction (JCT) Object
- The Call Object
- The If/Then/Else Object
- The Break Object
- The Until Break Object

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 14.1 – Creating and Merging a Library of UserFunctions
This lab will show you how to use libraries of VEE Pro UserFunctions. You will devise and save a library of UserFunctions.

Lab 14.2 – Importing and Deleting Libraries
This lab will show you how to import and delete the specific libraries that you select.

Lab 14.3 – Studying Lissajous Patterns
This lab will show you how to devise a VEE Pro program that will permit the study of Lissajous patterns, which include the presentation of frequency ratios and phase shift between the two incoming frequencies.

Lab 14.4 – Monitoring Vehicle Radiator Test Limits
This lab will show you how to devise a program that will allow you to monitor several tests and note where any given test exceeds previously established limits.

Lab 14.1 – Creating and Merging a Library of UserFunctions

This lab will show you how to use libraries with the VEE Pro UserFunctions. You will devise and save a library of UserFunctions. You will devise the top-level program without programming the details of the UserFunctions. You will then devise a report-generation program by building a VEE Pro program within that program. Last, you will devise a new program that saves your newly created library of UserFunctions into it.

1. Clear your Work Area, do not deselect the Program Explorer, maximize Main.

Creating and saving a library of UserFunctions

2. Create four user functions via Menu Bar => Device => UserFunction:
   BuildRecAry with one output pin
   ReportHeader
   ReportBody with one input pin
   ReportDisplay
3. Minimize the four UserFunctions; they will appear across the bottom of your Work Area.
4. Configure four Menu Bar => Device => Call objects; label and connect these objects as shown in Figure 14.1; size to fit your Main window.

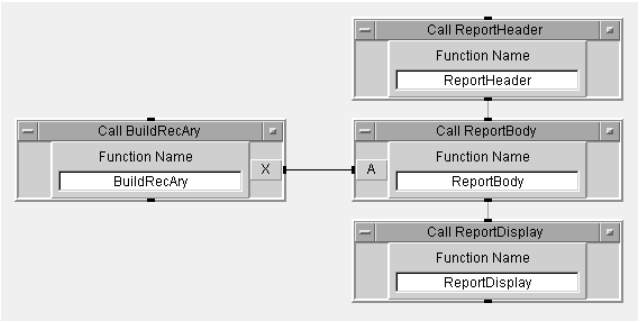


Figure 14.1. Report.vee from the "Top Level"

5. Save As… LAB14-1.
   Note: If you were calling the function from a Formula Object, you would need to include parentheses; if you are using the Call Object, you do not need parentheses.
6. Select Menu Bar => Edit => Edit UserFunction… to see the four new functions in the library; cancel this list box.
7. Double-click on the BuildRecArray UserFunction to open it.
8. Add the six objects; program and connect them as shown in Figure 14.2; minimize the BuildRecAry UserFunction.
   Note 1: This is an excellent opportunity to learn to use Appendix A. It describes additional functions on the Menu Bar and the other pull-downs.

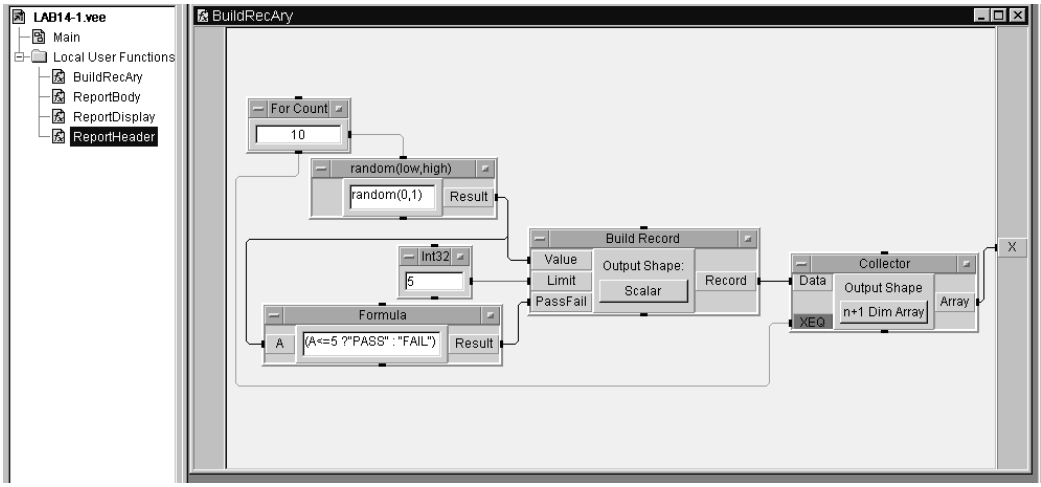


Figure 14.2. The BuildRecAry UserFunction

Note 2: The formula: (A<=5 ? "PASS" : "FAIL") means:
If A<=5, **then** the output is "PASS";
or **else**
If A>5, **then** the output is "FAIL".

9. Double-click on the ReportHeader UserFunction to open it; select a ToFile; program it as shown in Figure 14.3; delete the data-input terminals; minimize the ReportHeader UserFunction.
Note: The path required to locate files via ToFile and FromFile are defined based upon the default directory of Windows. See Help for both of these icons for additional information regarding file paths.

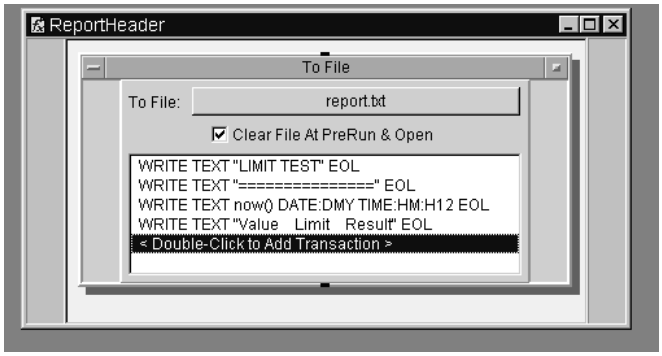


Figure 14.3. The ReportHeader UserFunction

Applying I/O Transaction guidelines

Note 1: See the pre-lab for this lesson for I/O Transactions Guidance setup for the ReportHeader and ReportBody setup. Also, study the following notes.

Note 2: In the ReportHeader To File Object – for the four transaction lines:

- Within WRITE TEXT "LIMIT TEST", select WRITE then TEXT; type "LIMIT TEST" then select DEFAULT FORMAT and EOL ON.
- Within WRITE TEXT"==========", select WRITE then TEXT; type "==========" (15 equal signs); and select DEFAULT FORMAT and EOL ON.
- Within WRITE TEXT now() DATE…, select WRITE then TEXT; type: now(); select TIME STAMP FORMAT with DEFAULT FIELD WIDTH; and select Date & Time, then DD/Month/YYYY, HH:MM, 12 HOUR, & EOL ON.
- Within WRITE TEXT "Limit Value Result", select WRITE then TEXT; type "Value Limit Result" and select DEFAULT FORMAT and EOL ON.

Note 3: The number of "equal" signs above must be chosen to be one per character to be displayed.

Note 4: The spacebar can be used to center the headings over the data. In this example, there must be four spaces either side of the word "Limit" to center these data.

10. Open the ReportBody UserFunction; select a ToFile and ForCount; program it as shown in Figure 14.4.

Note: In the ReportBody To File Object – for the two transaction lines:

- Within WRITE TEXT A[B].Value…, select WRITE then TEXT, then type: A[B].Value, " | ",A[B].Limit, " | " then select REAL64 FORMAT, DEFAULT WIDTH; select /- and FIXED, NUM FRACT DIGITS: 4 with EOL OFF.
- Within WRITE TEXT A[B].PassFail…, select WRITE, then TEXT, type: A[B].PassFail; select STRING FORMAT, then FIELD WIDTH: 5; and RIGHT JUSTIFY; and select ALL CHARS and EOL ON.

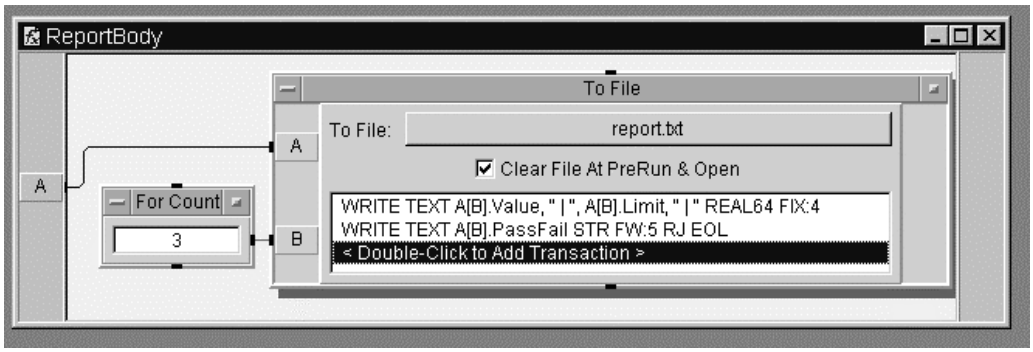


Figure 14.4. The ReportBody UserFunction

Note 1: The "A" input to the report body is assumed to be an Array of Records A[B]. Value means that, as B changes from 0 to 1 to 2, you will access first that element in the Record Array. You are

using the "A[B]" sub-array syntax; followed by that particular field in the Record (that is Value, Limit, and PassFail), using the <record><field> notation.
Note 2: The first transaction has EOL off.
Note 3: "|" represents a constant string character for a vertical bar.
Note 4: FW:5 stands for a string field width (FW) of 5.
Note 5: RJ stands for right justified.

11. Rename ToFile to: report.txt.
12. Minimize ReportBody.
13. Double-click on ReportDisplay UserFunction; select a FromFile and Logging AlphaNumeric; rename it Limit Test. See Figure 14.5.
    Note 1: In the ReportDisplay From File Object – for the one transaction line:
    Within READ TEXT STRING ARRAY:* select READ then TEXT, select STRING FORMAT, then ARRAY 1D; and TO END (which will provide the asterisk).
    Note 2: You are reading a string array 1D TO END of file, specified by the * sign after the STR ARRAY format.
14. Select Menu Bar => Flow => Confirm (OK); rename it **Done** via its Properties box; connect the icons as shown in Figure 14.5.

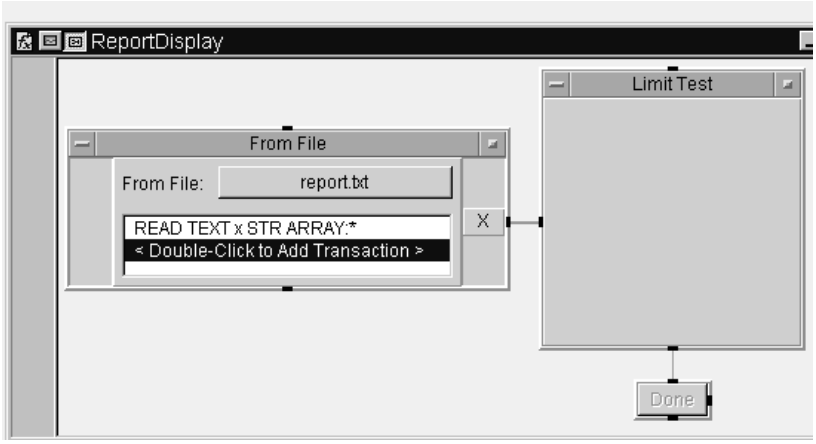


Figure 14.5. The ReportDisplay UserFunction detail view

15. Create an Operator Interface for ReportDisplay: add the Limit Test display and the Done button to a panel; go to the ReportDisplay Properties box; select Show Panel on Execute. See Figure 14.6.
    Note: If you have difficulty setting up the Operator Interface, then go to Appendix B and examine the "operator interface" options. (Hint: you can also use: Menu Bar => Edit => Add to Panel.)

Figure 14.6. The ReportDisplay operator interface

16. Save your LAB14-1 program again. Run this program; it should look like Figure 14.7.

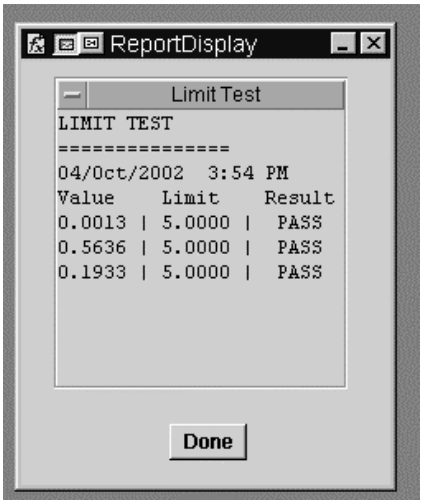


Figure 14.7. The operator interface after running

17. Click on the Done button to complete your program execution.

◊ 18. Change the Limit of **5** to cause some of the Result information to fail. See step 8 above.

19. Close this program after running.

Alternate to Lab 14.1

Adam Kohler of Agilent has suggested an alternate approach to Lab 14.1 that will use the File Name Selection object. This approach forces the use of the same file name.

Using File Name Selection

20. Add to your existing Lab 14.1 the File Name Selection object found as indicated in Figure 14.8. Follow the instructions given in the three note pads.
21. Modify the three Call objects to include FileName inputs. Connect these as shown.

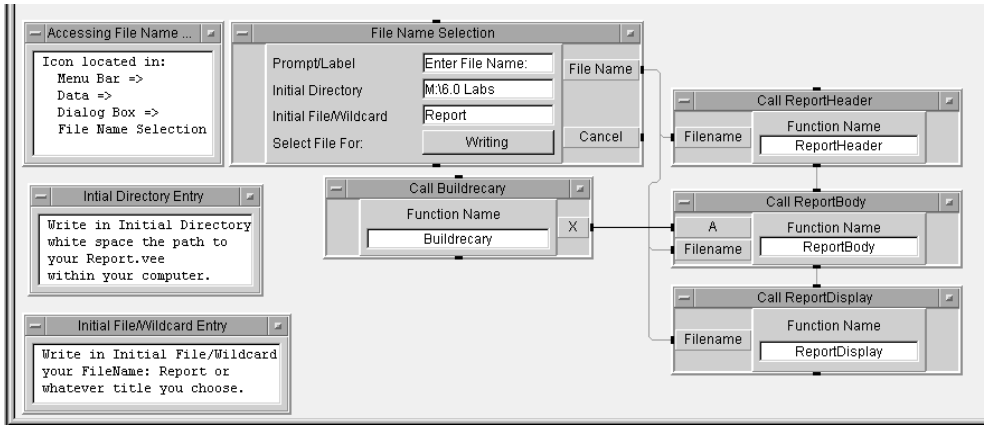


Figure 14.8. Report.vee using File Name Selection

22. Modify your ReportBody Function as shown in Figure 14.9 – both the instructions in ToFile and the additional terminal (FileName) and its connection.

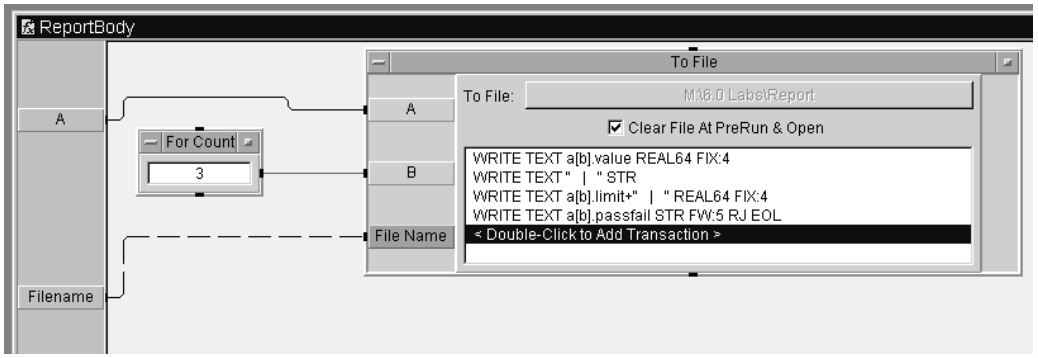


Figure 14.9. ReportBody revisions

23. Modify your ReportDisplay Function as shown in Figure 14.10 – both the instructions in FromFile and the additional terminal (FileName) and its connection.

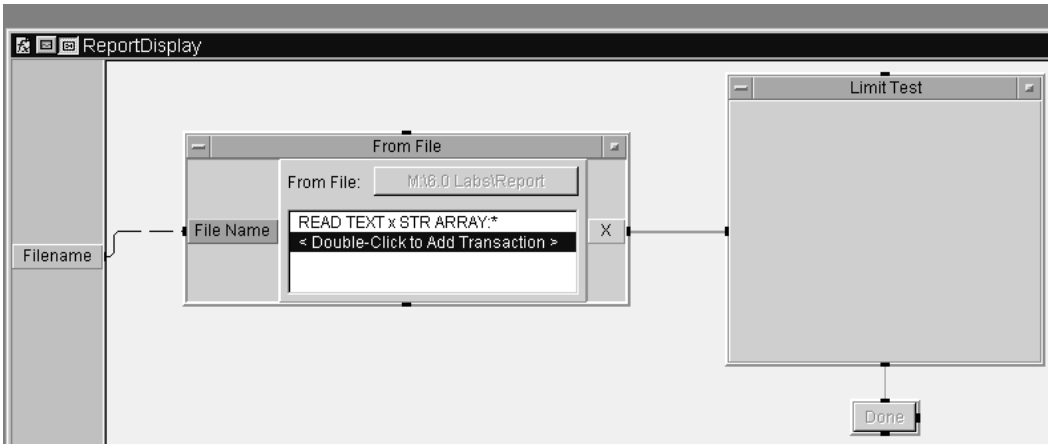


Figure 14.10. ReportDisplay revisions

24. Modify your ReportHeader Function as shown in Figure 14.11 – both the instructions in ToFile and the additional terminal (FileName) and its connection.

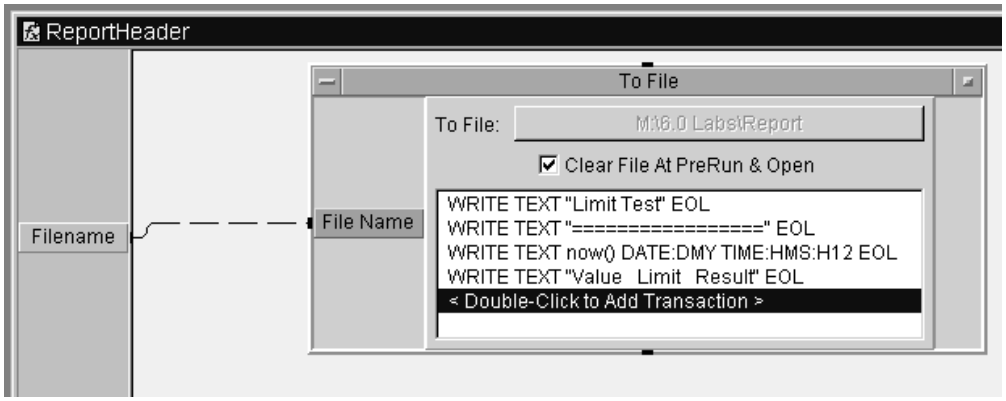


Figure 14.11. ReportHeader revisions

25. Save this alternate program as LAB14.1-ak; run this program if desired.
    Note: The program may search for your report. Be certain the Initial Directory shown in Figure 14.8 is modified to agree with the Report location in your computer.
26. Close this alternate program.

Lab 14.2 – Importing and Deleting Libraries

This lab will show you how to import and delete the specific libraries that you select.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Importing a library

2. Select Menu Bar => Device => Import Library; place it in the upper-left corner of Main; leave its default name.
3. Click on myFile; a box will appear that asks: Import library from what file?
4. Select LAB14-1.vee; click OK.
5. Open the Import Library object menu; select Load Lib.
   Note: Object Menu; Load Lib: Immediately loads the specified library into VEE.
6. Select Menu Bar =>View => Program Explorer; click on the lower "+" sign; the four user functions will appear from LAB14-1.vee.
   Note: You will not be able to edit these UserFunctions; you may only call them. Editing must be performed via the Merge… command. Using Merge… creates a separate copy of the UserFunction that must be maintained separately. Avoid using Merge… if possible to avoid duplication of files.
7. Select Menu Bar => Device => Call; place it below Import Library; connect its top pin to the bottom pin of Import Library.
8. Open the Call Function object menu; click on Select Function.
9. Choose Type: Imported User Functions; Category: <All>; Member: myLib.BuildRecAry; click OK.
   Note: The Call Function is automatically renamed; its output terminal is also added.
10. Select Menu Bar => Display => AlphaNumeric; place it to the right of Call… ; enlarge it vertically so it can accept up to ten rows of data.
11. Connect Call … X output pin to the AlphaNumeric input pin.

Deleting a library

12. Select Menu Bar => Device => Delete Library; place it below Call… .
13. Connect the bottom pin of Call… to the top pin of Delete Library.
    Note: This step will cause the library of UserFunctions to be deleted after it the Call Object executes. This saves memory requirements in large programs.
14. Save your program as LAB14-2.
15. Run this program. It should look like the one in Figure 14.12.

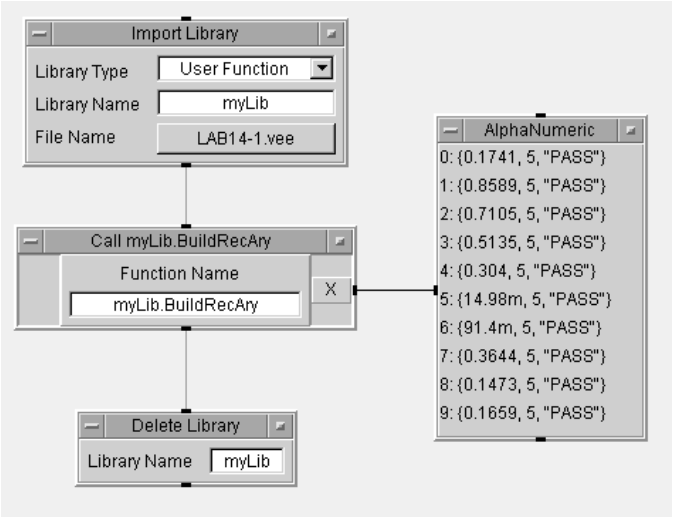


Figure 14.12. Calling a UserFunction from a library

Finding functions in large programs

16. Select Menu Bar => Edit => Find to locate a UserFunction or UserObject in a large program. The Dialog Box is shown in Figure 14.13.

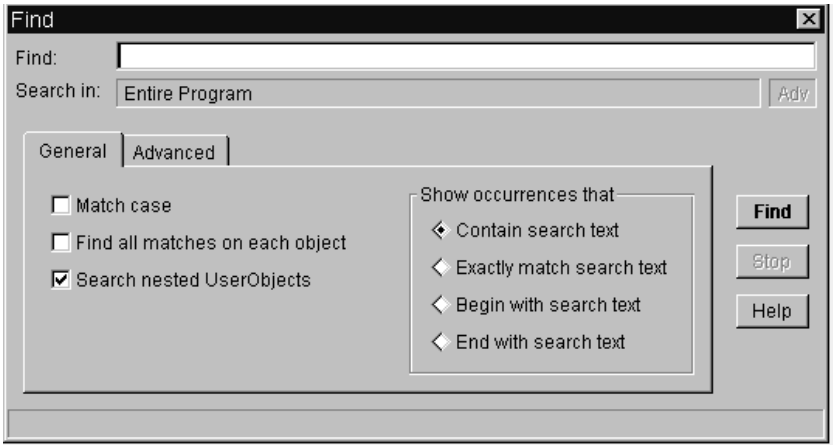


Figure 14.13. The Find dialog box

17. Click Find; it is on the right side of the Dialog Box. The results of the search are the objects listed in Figure 14.14. (If you find it necessary, examine its HELP first.)

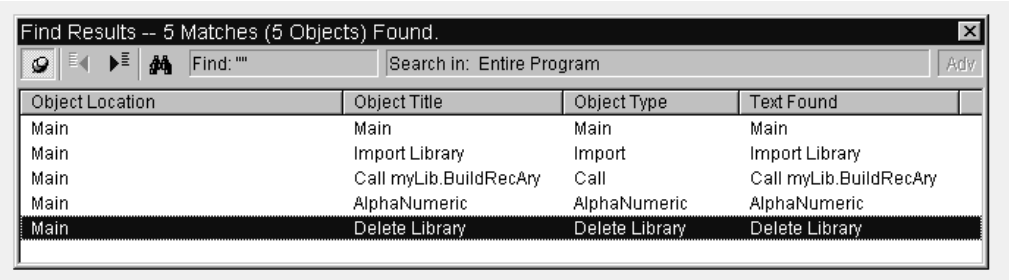


Figure 14.14. The Find Results dialog box

Note: Find can be used to locate variables, titles, settings on objects, etc. Double-click on the desired line that is displayed; it will take you to the location of the object.

18. Close Lab 14-2 without saving it again.

Lab 14.3 – Studying Lissajous Patterns

This lab will show you how to devise a VEE Pro program that will permit the study of Lissajous patterns, which include the presentation of frequency ratios and phase shift between the two incoming frequencies.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Preparing a Lissajous pattern-generation display

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of your work area.
3. Clone the Function Generator; place it under the first Function Generator.
4. Change the name of the upper Function Generator to X Frequency.
5. Change the name of the lower Function Generator to Y Frequency.
6. Select Menu Bar => Display => X vs Y Plot; place it to the right of, and between, the two Function Generators.
7. Change the name of the X vs Y Plot to Lissajous Patterns.
8. Remove the check marks from the entire vertical list of six selections labeled “Layout” in the Object Properties box.
9. Select “None” under Grid Type.
10. Select “Off” under Markers; click OK.
11. Change the title of the upper input terminal to X Data; connect this terminal to the upper “X Frequency” Function Generator.
12. Change the title of the lower input terminal to Y Data; connect this terminal to the lower “Y Frequency” Function Generator.
13. Change the frequency of the upper generator to 1000; change the Function to Sine.
14. Change the frequency of the lower generator to 4000; change the Function to Sine.
15. Change the Num Points to 2048 for both generators.
16. Run this program; it should look like Figure 14.15, save this program as LAB14-3.

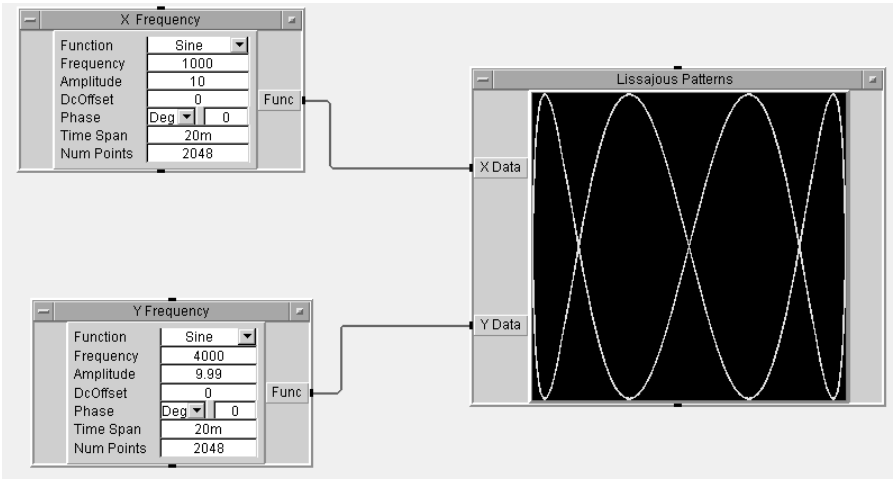


Figure 14.15. Four-to-one Lissajous pattern; no phase shift

◊ 17. Change the Frequency and Phase values of the lower (Y Frequency) generator to a variety of values. Run this program again. Record the changes below.
Note: Select some frequency values that are not precise ratios.

Your notes on waveform shapes:

Note: You may wish to insert additional pages of record information.

18. Do not save this modified program.

Displaying the frequency and phase ratios for a Lissajous pattern

19. Move your icons to the right-hand side of the screen, go to Save As… and save this program as LAB14-3a.
20. Go to the X Frequency generator; select Add Terminal, Data Input; choose Frequency; click OK.
21. Go to the Y Frequency generator; select Add Terminal, Data Input; choose Frequency; click OK.
22. Go to the Y Frequency generator; select Add Terminal, Data Input; choose Phase; click OK.
23. Convert the two generators to icons.
24. Select Menu Bar = > Flow => For Range, change its settings to: From 1000, Thru 4000, Step 500.
25. Select Menu Bar => Device => Formula, change its Title Bar and expression to Y/X; change the name of the input terminal to Y; add a second input pin and name it X.
26. Select Menu Bar => Display => AlphaNumeric, place it above Lissajous Pattern, change its Title Bar to Frequency Ratio.
27. Place the Formula Y/X object to the left of Frequency Ratio, connect the Y/X output terminal to the Frequency Ratio input terminal.
28. Select Menu Bar => Data => Constant => Real64, place it above and to the left of the X Frequency generator, change its name to Real and its value to 1000.
29. Connect the Real output pin to both the X Frequency input pin and the Y/X input pin labeled X.
30. Select Menu Bar => Display => AlphaNumeric; place it under Y Frequency, change its name to Phase Shift.
31. Select Menu Bar = > Flow => For Range, place it to the left of Phase Shift change its name to Phase Shift Input, change its settings to: From 0, Thru 180, Step 10.
32. Connect the output pin of Phase Shift Input to both the input pin of Phase Shift and the bottom (Phase) input pin of Y Frequency.
33. Connect the upper For Range output pin to both the Y Frequency top (Frequency) input pin and the Y/X input pin labeled Y.
34. Connect the upper For Range output pin to the Phase Shift Input control (top) pin.
35. Select Menu Bar => Flow => Start, change its title to Press to Start, increase its size so the entire title can be seen.
36. Connect Press to Start output (bottom) pin to the For Range control (top) pin.
37. Save this program again.
38. Run this program; it should look like Figure 14.16.

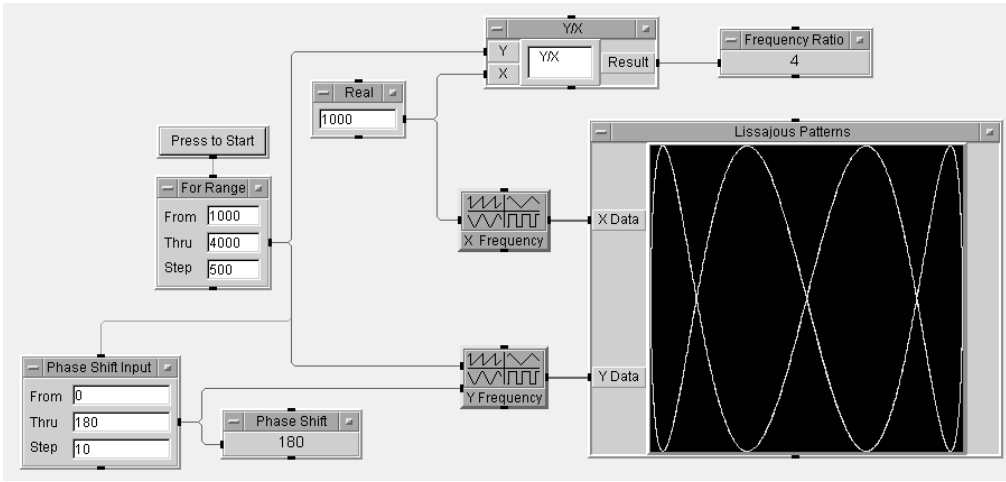


Figure 14.16. Frequency and phase ratio Lissajous displays

◊ 39. Run this program for a variety of settings of Y Frequency values of frequency and phase shift; use the Step into icon to observe gradual changes in frequency and phase shift.

40. Do not save this program unless you end at a setting that you prefer.
Note: Also select some frequency values that are not precise ratios.

Your notes on waveform shapes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Note: You may wish to insert additional pages of record information.

Lab 14.4 – Monitoring Vehicle Radiator Test Limits

This lab will show you how to devise a program that will allow you to monitor several tests and note where any given test exceeds previously established limits. If you are a programmer, you may want to experiment with other approaches to monitoring, labeling, and recording these tests.

1. Clear your Work Area, deselect the Program Explorer.

Recording several Vehicle Radiator tests

2. Open LAB11-4; Save As… LAB14-4 immediately.
3. Minimize Vehicle Radiator, VR Temperature and VR Pressure if they are not at the bottom of your screen.
4. Cut the following objects: VRTemperatureRecord, VRPressureRecord, and Vehicle Radiator Temperature and Pressure.
5. Select Menu Bar => Device => Formula; place it near the upper left of your screen; clone it three times and place them in a column.
6. Change the titles of these formula objects to read:
   temp min(x)
   temp max(x)
   pressure max(x)
   pressure min(x)
7. Change their formulas to read as follows:
   min(x)
   max(x)
   max(x)
   min(x)
8. Change all four formula object input terminals from **A** to **x**.
9. Connect the upper-two formula objects to the Vehicle Radiator upper terminal (Temp).
10. Connect the lower-two formula objects to the Vehicle Radiator lower terminal (Pressure).
11. Place four AlphaNumeric objects to the right of the four formula objects.
12. Change their titles to
    min temp
    max temp
    max pressure
    min pressure
13. Connect the four AlphaNumeric objects to their respective formula objects.
14. Select Menu Bar => Flow => If/Then/Else; place it above your “min temp” AlphaNumeric object.
15. Clone If/Then/Else; place it between your “max temp” and “max pressure” AlphaNumeric objects.
16. Change the upper If/Then/Else title to Max Allowed Temp.
17. Connect its input (A) terminal to the output terminal of “temp max(x)”.
18. Change its formula to: 196.5<=A.
    Note: The numeric value will be changed to a value that you desire your Vehicle Radiator temperature not to exceed. Test runs will require a different value, based upon how often the maximum-high warning object (high) turns on.
19. Change the lower If/Then/Else title to Max Allowed Pressure.
20. Connect its input (A) terminal to the output terminal of “pressure max(x)”.
21. Change its formula to: 42.75<=A.

Note: The numeric value will be changed to a value that you desire your Vehicle Radiator pressure not to exceed. Test runs will require a different value, based upon how often the maximum-high warning object (high) turns on.

22. Select Menu Bar => Display => Indicator => Color Alarm; place it to the right of "Max Allowed Temp".
23. Clone Color Alarm three times; place these objects in a column below the first one.
24. Change the Color Alarm titles, from top to bottom, as follows:
    Max Temp Exceeded
    Within Temp Safety Range
    Max Press. Exceeded
    Within Press. Safety Range
25. Select Menu Bar => Display => Indicator => Beep; place it below the "pressure min(x)" formula object.
26. Change the title bar of the "Beep" to read: Max-Pressure Extra Beep.
27. Connect the "Max Allowed Temp" **Then** output terminal to "Max Temp Exceeded" input (data) terminal.
28. Connect the "Max Allowed Temp" **Else** output terminal to "Within Temp Safety Range" input (data) terminal.
29. Connect the "Max Allowed Pressure" **Then** output terminal to "Max Press. Exceeded" input (data) terminal.
30. Connect the "Max Allowed Pressure" **Else** output terminal to "Within Press. Safety Range" input (data) terminal.
31. Connect the "Max Allowed Pressure" **Then** output terminal to "Max Pressure Extra Beep" input (top) terminal.
32. Select Menu Bar => Flow => Start; place it to the left of "Vehicle Radiator"; connect its output terminal to the top terminal of Vehicle Radiator.
33. Verify your object locations, titles, and connections via Figure 14.17.

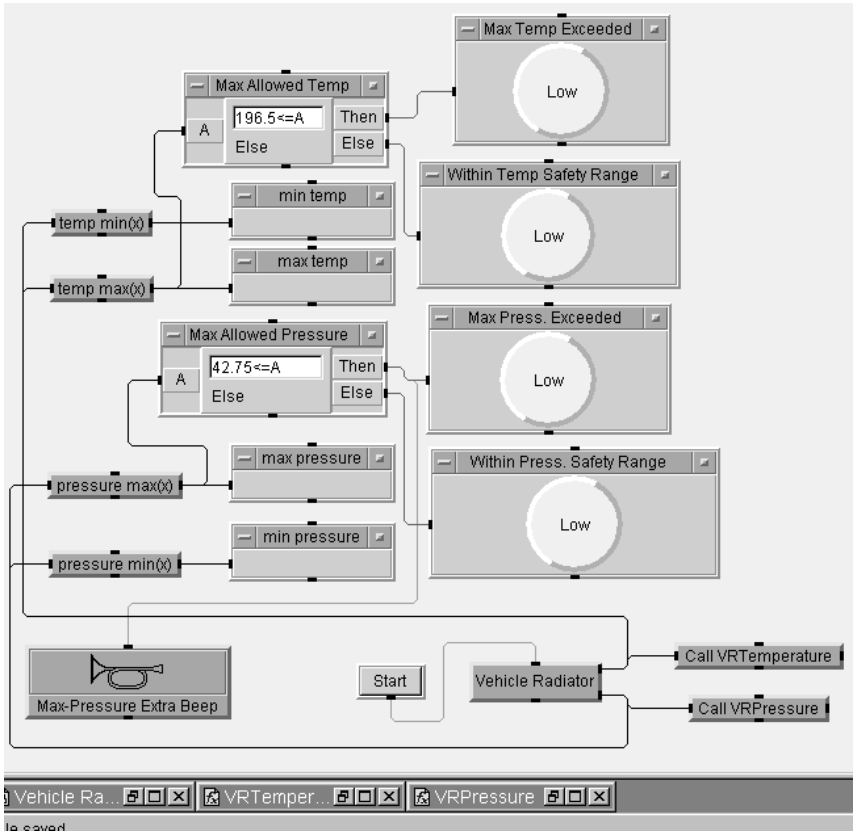


Figure 14.17. Lab 14-4 partial layout and connections

34. Save your program again.
    Note: Right-click your mouse on the open space of the screen; select and click on "Clean Up Lines" from the Edit menu that appears.
35. Select Menu Bar => Device => UserFunction; change its name to **alarm**.
36. Select Menu Bar => Display => Indicator => Beep; place it in the upper-left corner of the **alarm** UserFunction.
37. Select Menu Bar => Data => Constant => Real64; place it to the right of **Beep**.
38. Select Menu Bar => Display => Indicator => Color Alarm; place it to the right of **Real64**.
39. Open the Color Alarm Properties box; remove the checkmark from Open View – Show Title Bar; under Layout, change the shape selection to Rectangular; and under Limits; delete the names for High Text, Mid Text, and Low Text.
40. Select Menu Bar => Flow => Delay; place it below the rectangular Color Alarm.
41. Connect the output terminal of Real64 to the Color Alarm left-hand (data) terminal.
42. Connect the (bottom) Color Alarm output terminal to the Delay (top) terminal.
43. Minimize the **alarm** UserFunction so it appears at the bottom of your screen.

44. Select Menu Bar => Flow => Junction; place it to the right of and between “Max Temp Exceeded” and “Within Temp Safety Range”.
45. Select Menu Bar => Flow => Until Break; place it below and slightly to the right of JCT; double-click to open it.
46. Connect the lower terminal of “Max Temp Exceeded” to the A-terminal input of JCT.
47. Connect the lower terminal of “Max Press. Exceeded” to the B terminal input of JCT.
48. Connect the data output of JCT to the top terminal of Until Break.
49. Select Menu Bar => Device => Call; place it below and to the right of Until Break.
50. Change the Function Name field of Call myFunction to **alarm**; its Title Bar will be changed automatically.
51. Connect the output terminal (continuous) of Until Break to the top terminal of Call alarm.
52. Select Menu Bar => Data => Toggle Control => Check Box; place it below Call alarm; connect the sequence output (bottom) pin of Call alarm to the sequence input (top) pin of Check Box.
53. Open its Properties box; go to Fonts; select Arial Black, size 14 Italics for both the Title and Object boxes; place checkmarks in Layout Scaled and Initialization: Initialize at PreRun with an initial value of zero (0); change its title to: **TURN OFF VEHICLE RADIATOR!**
54. Select Menu Bar => Flow => If/Then/Else; place it below **TURN OFF VEHICLE RADIATOR!**; change its formula to A==1 (two “equal” signs).
55. Select Menu Bar => Flow => Repeat => Break; place it below and to the right of If/Then/Else.
56. Connect the output terminal of **TURN OFF VEHICLE RADIATOR!** to the input **A** terminal of If/Then/Else.
57. Connect the If/Then/else **Then** terminal to the top terminal of Break.
58. Verify your object locations, titles, and connections via Figure 14.18.

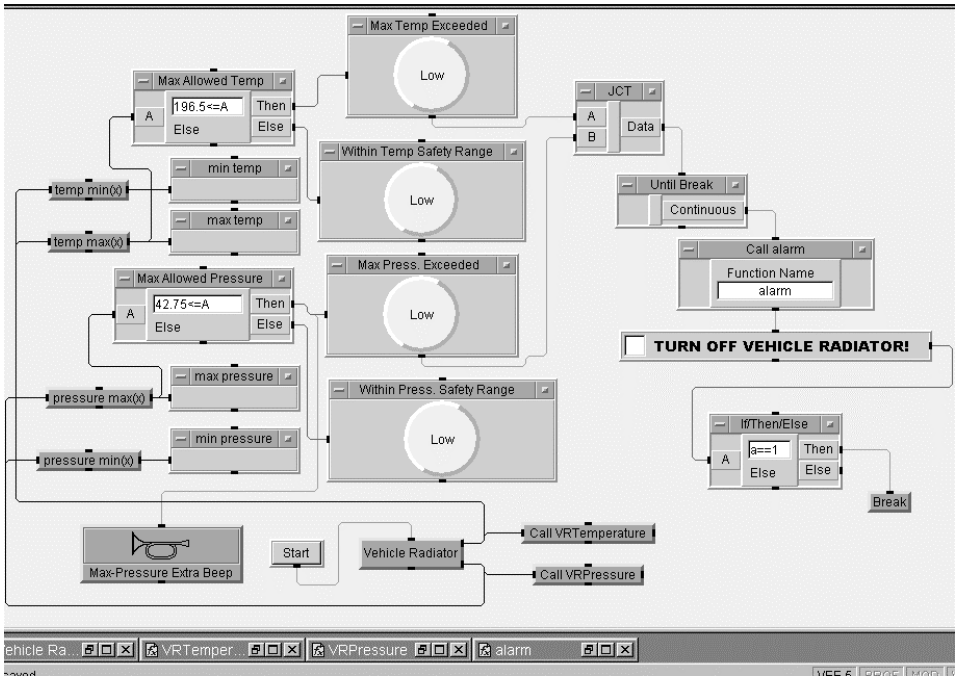


Figure 14.18. Lab14-4 before running

Building the Vehicle Radiator operator interface

59. Hold down the Ctrl key; click the mouse on the following objects:
    Start Max Temp Exceeded
    Within Temp Safety Range Max Press. Exceeded
    Within Press. Safety Range max temp
    max pressure TURN OFF VEHICLE RADIATOR!
60. Click the mouse on the Main menu background; select Add To Panel.
61. Arrange the objects as shown in Figure 14.19.

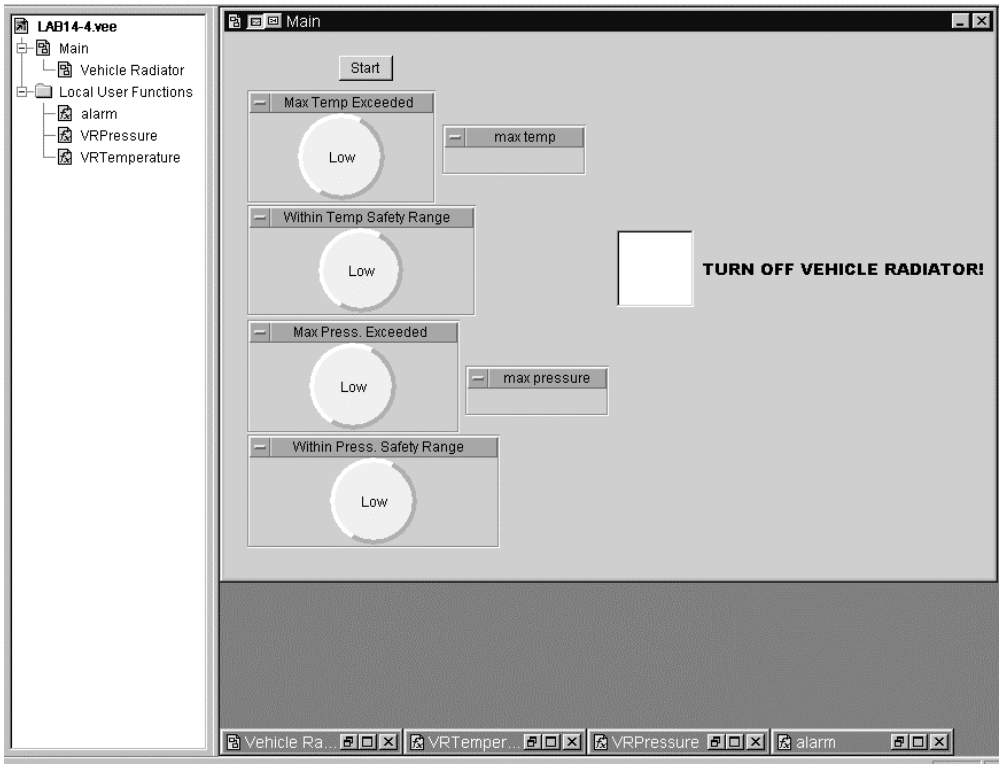


Figure 14.19. Lab14-4 operator interface

◊ 62. Run this program several times to observe the various combinations of "Low" and "High" and note the Beep sound (if you have active speakers); see Figure 14.20.

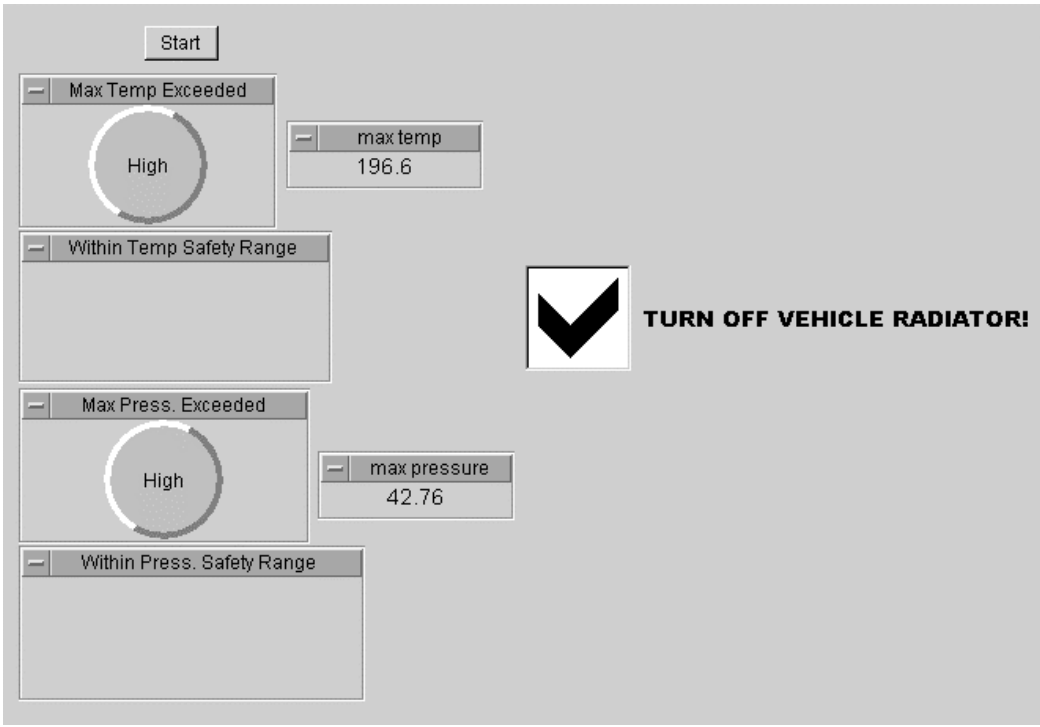


Figure 14.20. Lab14-4 after running

63. Double-click on the Vehicle Radiator UserFunction.
64. Open "Temp Noise Gen" and change its amplitude settings; open "Pressure Noise Gen" and change its amplitude settings.
    Note: Changing the settings of the Temperature and Pressure noise generators will require you to select a different set of "Max Allowed Temp" and "Max Allowed Pressure" values in the Main Detail program.
◊ 65. Run this program several times while changing the "Max Allowed Temp" and "Max Allowed Pressure" values to satisfy your preferred alarm limits.
66. Save this program to another LAB number if you desire to access it again later; close this program.

Securing the Vehicle Radiator operator interface

67. Select Menu Bar => File => Create Run Time Version… .
68. Select Menu Bar => File => Open; return to the Secured Program (Run Time) Version by first selecting Lab 14-4 and changing the .vee extension to .vxe.
    Note 1: No detailed view will be displayed. The operator can run this Secured Program. The operator cannot modify the Secured Program.
    Note 2: It is now a Secured Program and its icon will be darkened in color as displayed in the Windows directory.

Lesson 14 Summary

This lesson explored how to use libraries of VEE Pro UserFunctions. You learned to work with UserFunction operations, edit and call UserFunctions, create and save a library of UserFunctions, and import and delete libraries.

These VEE Pro capabilities will be applied in forthcoming lessons. You are now in a position to study more advanced material regarding VEE Pro programming. Thus, you are only a few lessons away from modifying the programs that you have already devised, adapting them to your own needs, and writing new and more sophisticated programs on your own.

VEE Pro provides a FIND Feature that will help you locate a UserFunction or UserObject in a large program via the Dialog Box. FIND can be used to locate such objects as variables, titles, settings on objects, etc.

Lab 14.1 showed you how to use libraries of VEE Pro UserFunctions. You learned to devise and save a library of UserFunctions.

Lab 14.2 showed you how to import and delete the specific libraries that you selected.

Lab 14.3 showed you how to devise a VEE Pro program that will permit the study of Lissajous patterns, which include the presentation of frequency ratios and phase shift between the two incoming frequencies.

Lab 14.4 showed you how to devise a program that will allow you to monitor several tests and note where any given test exceeds previously established limits. It also reviewed how to prepare a Secured Run Time Version.

We realize that you cannot remember all of the VEE Pro controls and shortcuts as you probably have other work to do as a part of your daily job assignment. Therefore, you should begin seriously documenting, organizing, and cataloging your new knowledge while it is still fresh in your mind. You may also use Appendix B to locate programming sequences that you have used in previous labs and will find useful when generating your own VEE Pro programs.

You are now ready to learn to use the Sequencer to create, pass, and compare data.

Lesson 15

Using the Sequencer to Create, Pass, and Compare Data

This lesson will examine how to prepare a Test Execution Order, test and select data with the Sequencer, pass data via a Global Variable, and compare a waveform output with a Mask. It consists of a pre-lab and four labs. For a description of the Sequencer functions, see Appendix C, page C-11.

Lesson 15 Pre-lab Summary

The following are described in the Lesson 15 pre-lab. See Appendix E, page E-78.

- Accessing Logged Test Data
- The Sequence Transaction Dialog Box
- Declaring Globals

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 15.1 – Creating a Test Execution Order
This lab will show you how to configure a test, add or insert a test, delete a test, and access logged test data.

Lab 15.2 – Passing Data via the Sequencer
This lab will show you how to pass data using an input terminal and set up (three) tests in the Sequencer to call Rand.

Lab 15.3 – Passing Data Using a Global Variable
This lab will show you how to pass data using a global variable.

Lab 15.4 – Comparing a Waveform Output with a Mask
This lab will show you how to compare a waveform output with a mask.

Lab 15.1 – Creating a Test Execution Order

This lab will show you how to simulate test results using the random( ) function with a specified range of expected test results, establish a test execution order, modify that order, and retrieve specific data from the logged results.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Configuring and specifying a pass/fail test

2. Select Menu Bar => Device => Sequencer; place it in the upper-left corner of Main.
3. Select Menu Bar => Display => AlphaNumeric; place it below the Sequencer.
4. Increase its width by at least a factor of two; connect its data-input pin to the Sequencer object Log output terminal.
5. Double-click on the Sequencer transaction bar (blue space) to get the Sequence Transaction dialog box.
   Note 1: The TEST field has the default name test1. It is not the test function.
   Note 2: The FUNCTION expression field holding the string testFunc(a) is the default; it holds the actual function that will perform the test.
6. Double-click on the FUNCTION expression field; type random( ).
   Note 1: Random( ) will return a real value between zero and 1; the default parameters.
   Note 2: SPEC NOMINAL represents the expected test value; its default is 0.5.
7. Change the SPEC NOMINAL to 0.25.
8. Alter the upper RANGE field (located on the far right) from 0 to 0.5.
   Note: This configuration will lead to a PASS result in approximately one-half of your runs. Your dialog box should look like Figure 15.1.

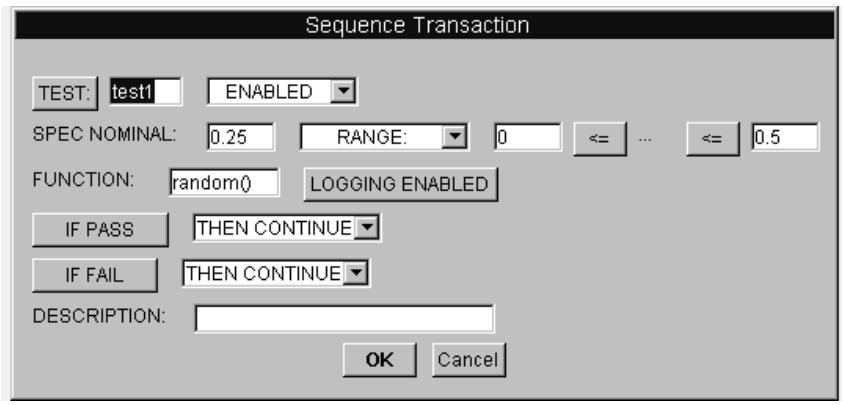


Figure 15.1. The Sequence Transaction dialog box

9. Click OK to close the dialog box.
   Note: The first transaction bar should now display test1 0 <= (0.25) <= 0.5.
10. Save this program as LAB15-1.

◊ 11. Run your program several times.

Note 1: In the display, you should see the name of the test, the test result, and the pass-fail indicator (1 for PASS, 0 for FAIL).

Note 2: This setup causes test1 to pass all results between and including 0 and 0.5; the expected result is 0.25.

12. Double-click below your first expression to add another Transaction; this will open the Sequencer Transaction dialog box. The test will be test2.
13. Review the content options in the Sequence Transaction field, starting on page E-95.

Setting specifications within a sequence transaction dialog box

TEST ENABLED or DISABLED
SPEC NOMINAL: RANGE, LIMIT, TOLERANCE, and %TOLERANCE
LOGGING ENABLED or LOGGING DISABLED: IF PASS or IF FAIL,
THEN CONTINUE, THEN RETURN, THEN GOTO,
THEN REPEAT, THEN ERROR, or THEN EVALUATE.
DESCRIPTION

Adding or inserting a configured test

Note: In this example, you will compare the result to a limit value.

14. Change SPEC NOMINAL from 0.5 to 0.25.
15. Click RANGE: to get the Select Spec Type dialog box; select LIMIT.
16. Choose < for your operator located to the right of LIMIT.
17. Change your right-hand limit from 1 to 0.5.
18. Change the FUNCTION field from testFunc(a) to random( ).
19. Click OK to return to the Sequencer.
    The transaction bar should read: test2 (0.25) < 0.5. See Figure 15.2.
    Note: You could add a transaction after the highlighted one by selecting Add Trans… from the Sequencer object menu.
20. Insert a transaction between these two tests by highlighting the second transaction bar.
21. Open the Sequencer object menu and select Insert Trans… .
22. Change the TEST field to Insert.
23. Change the FUNCTION field to random( ).
24. Click OK.
    Note: You will now see Insert 0 <= (.5) <= 1 as the second transaction bar.
25. Save this program again.

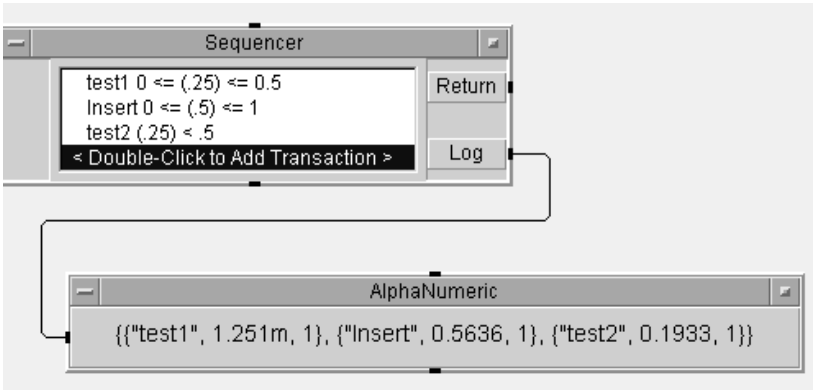


Figure 15.2. Inserting a test

◊ 26. Run your program several times; observe and verify the pass/fail changes.
Note: You should see the three records from your three tests on one line. Enlarging your alphanumeric display may be required.

Deleting a configured test

27. Highlight the Insert transaction bar; hold the mouse here.
28. Select (right-click) Cut Trans from the object menu.
Note 1: You can also paste a transaction that has been cut by choosing Paste Trans from the object menu.
Note 2: You can copy a transaction with the Copy Trans selection from the object menu.
29. Save your program as LAB15-1a.

◊ 30. Run this program several times; note the two records of test data.
Note 1: The braces indicate a Record data type. Since the AlphaNumeric shows nested braces, then the Sequencer output is a Record of Records. Double-click on the Sequencer "Log " output pin to observe the data after running.
Note 2: Examine the AlphaNumeric display. You could place the Sequencer in a loop and run the same sequence of tests several times. This would yield an array of Records of Records.

Accessing logged "record of records" test data

Note: The Sequencer output is considered a Record of Records – not an array of Records. (An array of Records requires that all elements have the same field structure.)
In the Sequencer record, each test uses the test name as its field name. The fields within each test are named according to your logging configuration. If you use the default configuration with the fields whose titles are Name, Result, and Pass, you could access the Result in test2 with the notation Log[*].Test2.Result. This is the *entire* middle-right column (Runs 1 through 4) of Figure 15.3.

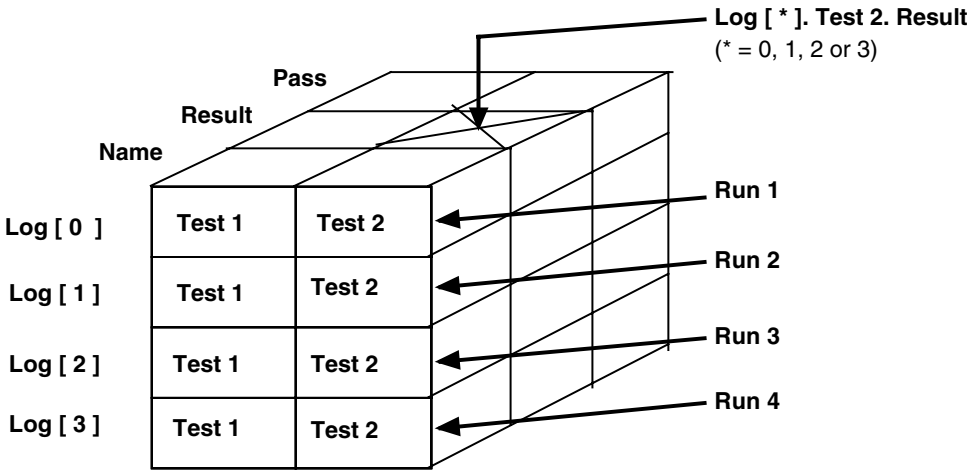


Figure 15.3. A logged array of Record of Records

31. Select Menu Bar => Device => Formula; place it below your AlphaNumeric display in program LAB15-1a.
32. Change the input terminal name from A to Log; connect it to the Sequencer Log terminal.
33. Change the default formula of the Formula object to: Log.Test2.Result.
    Note: You could leave the default name A. Your formula would then read: SA.Test1.Result
34. Select an AlphaNumeric display; connect its data-input object to the Formula output (Result).
35. Run your program.
    Note 1: You have accessed the Result field in Test2. See Figure 15.4.

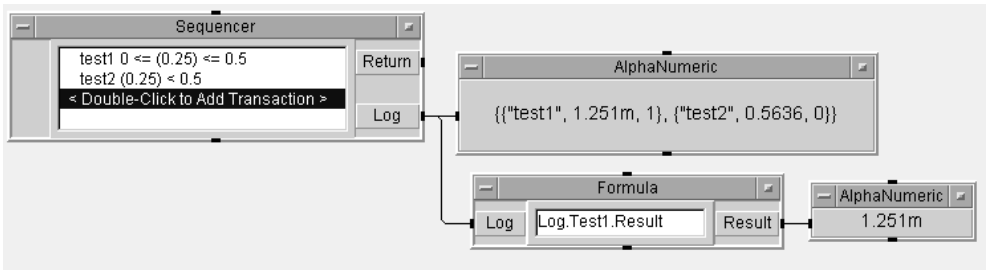


Figure 15.4. Accessing Logged Data

Note 2: This record could be used in subsequent tests.
Note 3: If you need to access test data in an expression field inside the Sequencer while the program is still running, then test data is stored in the temporary record "thistest".

36. Save this program As… LAB15-1b and stay with this program. (Leave it on the screen.)
37. Select Menu Bar => Data => Constant => Record; place in the right-center of the screen.

38. Add a Default Value (Control Input) terminal to the record.
39. Connect the Default Value terminal to the Sequencer Log terminal.
40. Examine the stored data in the Record object by clicking the button to the right of the test of interest (test1 or test2). See Figure 15.5. Save your modified program As…LAB15-1c.

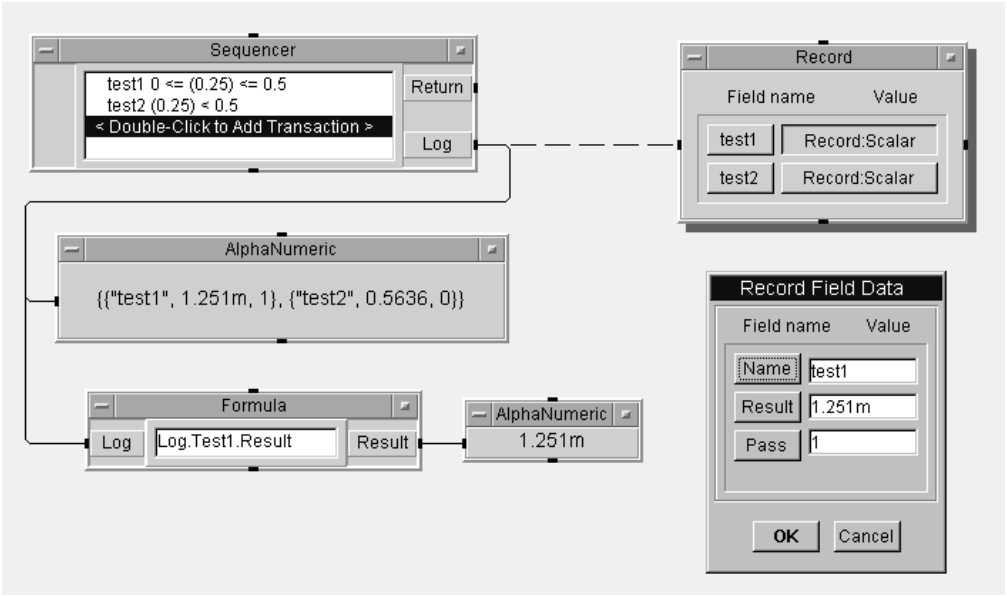


Figure 15.5. Stored data in the Record object

◊ 41. Run this program several times.
Note: This is another method for storing data. It is more compact than the data stored and displayed in the AlphaNumeric display.
42. Close this program without saving it.

Lab 15.2 – Passing Data via the Sequencer

This lab will show you how to create a UserFunction and call it from three different tests. You will also learn to call a function in the EXEC mode rather than in the TEST mode.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Passing data via a UserFunction

Note: You will first create the UserFunction Rand that will simulate a measurement procedure. Rand( ) will add an input parameter to the output of the random(low,high) object. The result will be placed on the output pin.

2. Select Menu Bar => Device => UserFunction.

3. Change the name from UserFunction1 to Rand.
4. Select Menu Bar => Device => Function & Object Browser.
5. Select Type: Built-in Functions; Category: All; and Member: random.
6. Click on Create Formula – the Formula Object random(low,high) will appear.
7. Place random(low,high) in the upper-left corner of Rand.
8. Delete the input terminals and the parameters: low,high in the Formula field; leave the parenthesis pair; change the object title to random().
9. Select Menu Bar => Device => Function & Object Browser.
10. Select Type: Operators, Category: <All>, and Member: +.
11. Click on Create Formula; a new Formula Object "A+B" will appear.
12. Place "A+B" to the right of, and slightly below, random().
13. Connect the Result output of random() to the A input of the A+B object.
14. Add a data-input terminal to the UserFunction Rand.
15. Connect the Rand input terminal A to the B terminal of the A+B object.
16. Add a data-output terminal to Rand.
17. Connect the output of the A+B object to the Rand output terminal. Your result should look like Figure 15.6.

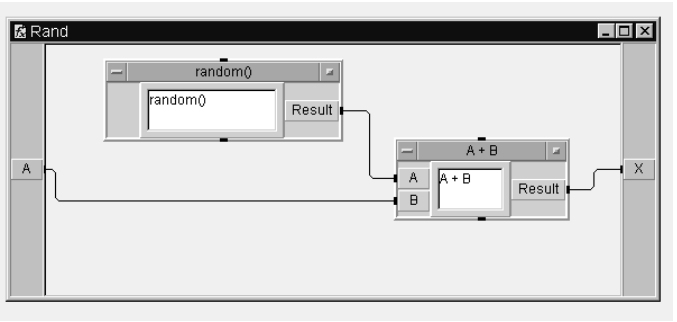


Figure 15.6. The Rand UserFunction

18. Save this program as LAB15-2.
19. Minimize the Rand window by using the "_" button on its top-right corner; it will convert to an icon at the bottom of the workspace under Main.

Setting up (three) tests in the Sequencer to call Rand via an input terminal

20. Select Menu Bar => Device => Sequencer; place it in the top-center of Main; size your Main window so you can see the UserFunction Rand.
21. Add a data-input terminal to the Sequencer.
22. Double-click the transaction (blue) bar to get the Sequence Transaction dialog box.
23. Change the FUNCTION field from testFunc(a) to rand(a).
24. Click OK to return to the Sequencer open view.
    Note: You may use a Sequencer input terminal name, such as "A", to pass data to any expression within the Sequence Transaction box.
25. Place the cursor on the transaction (blue) bar; press Ctrl-k to cut the test.

26. Press Ctrl-y (three) times to paste the test back into the Sequencer or do the same with object menu selections.
    Note: The default test names are test1x2, test1x1, and test1.
27. Open the (three) Sequence Transaction dialog boxes – one at a time.
28. Change their names to test1, test2, and test3 (for clarity) as each is opened; click OK.
29. Select Data => Continuous => Real64Slider; place it to the left of the Sequencer.
30. Change the Real Slider name to: Select Num; size the object to be shorter.
31. Connect the Select Num output pin to the Sequencer input terminal A.
32. Select an AlphaNumeric display; place it below the Sequencer and slider (Select Num).
33. Enlarge the display to be as wide as the slider and Sequencer combined.
34. Connect the display input to the Sequencer Log output terminal.
35. Save this program as LAB15-2a. It should look like Figure 15.7.

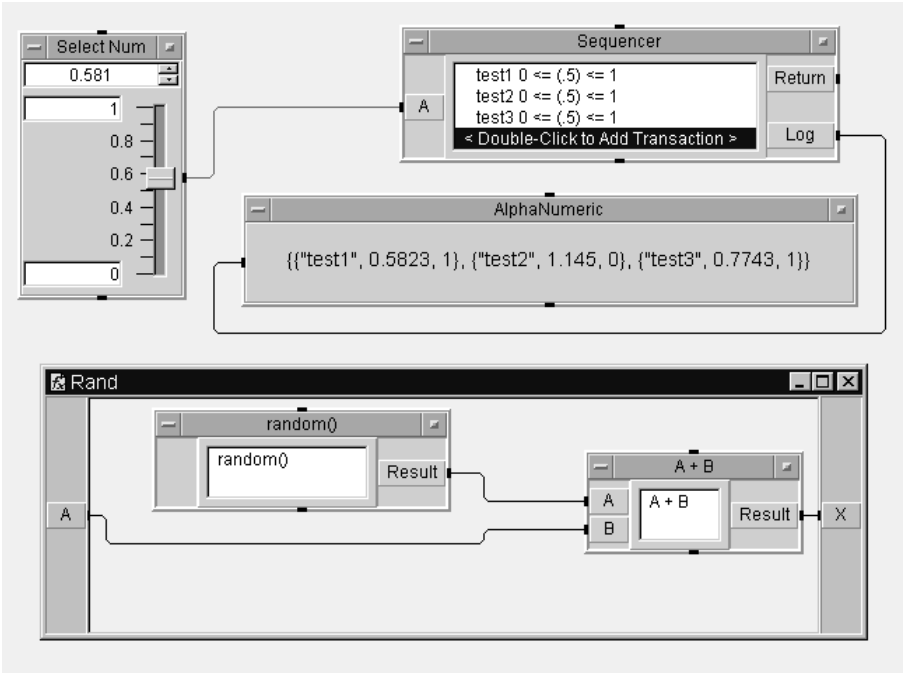


Figure 15.7. Passing three tests using an input terminal

Note 1: You can reduce the number of input pins if you pass records to input terminals. You can then use individual fields within records for separate tests.
Note 2: You can instead use a separate UserFunction to set up global variables as explained below.

◊ 36. Run your saved program several times with different Select Num values; observe the AlphaNumeric display; verify that each test agrees with the test constraints for pass/fail.
Close this program without saving it.

Lab 15.3 – Passing Data Using a Global Variable

This lab will show you how to pass data using a global variable; it is described in Appendix E, page E-79. It will start with the program just completed in Lab 15.2.

Note: A global variable is a UserFunction with internal, user-controlled values. The UserFunction name is changed so it can be accessed for a number of programs. A specific global-variable Object contains one or more subprograms whose Panel View can be automatically called and its internally variable number or other variable can be changed by the user.

1. Clear your Work Area.

Passing data using a global variable

2. Open program LAB15-2a; Save As… LAB15-3 immediately.
3. Delete (cut) the Real Slider object labeled Select Num.
4. Delete the A input terminal on the Sequencer.
5. Highlight the test1 transaction bar and open the object menu.
6. Click Edit Trans…; the Sequence Transaction box will appear.
7. Click Test to toggle the selection to EXEC; change the test name to Setup.
   Note: The EXEC mode will evaluate the expression and not test the result for "Pass/Fail".
8. Change the FUNCTION field to setGlobal( ); click OK to close the dialog box.
   Note: This allows you to call the UserFunction setGlobal( )
9. Open the Transaction box for test2; change its name to test1; open the transaction box for test3, change its name to test2.
10. Highlight "Double-click to Add Transaction", change its name from test4 to test3.
11. Change the three "test" transactions that currently have "Rand(a)" to "Rand(g_data)".
12. Select Menu Bar => Device => UserFunction; change the name UserFunction1 to setGlobal; click OK.
13. Select Menu Bar => Data => Variable => Declare Variable; place it in the upper-right corner of the UserFunction: setGlobal. Change the name to "g_data".
    Warning: Declaring and initializing variables are often neglected. Be certain that all information is provided.
14. Select Menu Bar => Data => Continuous => Real64Slider; place it in lower-left corner of the UserFunction: setGlobal; change the name of Real64Slider to: Select Num; click OK; size it to be smaller vertically.
15 Select Menu Bar => Data => Variable => Set Variable; place it to the right of Select Num.
16. Change the global variable name in the white field from globalA to g_data.
17. Connect the Select Num output pin to the Set g_data object input pin.
18. Select Menu Bar => Flow => Confirm(OK); place it above Select Num.
19. Connect the OK sequence-output (bottom) pin to Select Num sequence-input (top) pin.
    Note 1: The Confirm (OK) button causes the panel to remain on the screen until a user has made a selection.
    Note 2: You can turn on Show Data Flow to watch the execution order.
20. Select Menu Bar => Display => Note Pad; place it to the right of the OK button.
21. Enter the following user prompt in the Note Pad:
    Please select a number for this run of tests 2 and 3.
22. Size the Note Pad. See Figure 15.8.

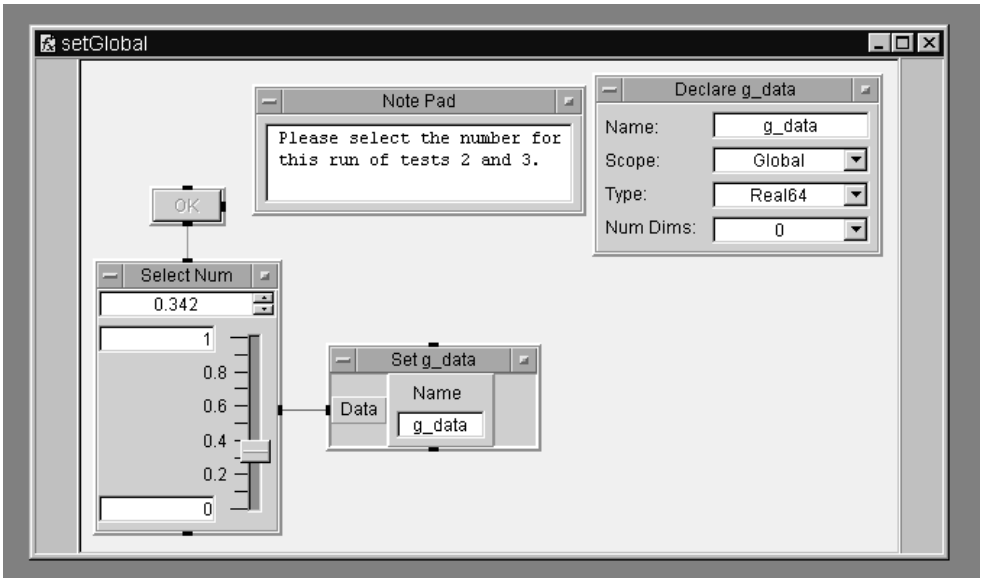


Figure 15.8. The setGlobal UserFunction detail view

23. Depress Ctrl and click on Note Pad, Select Num, and OK to highlight them; release Ctrl.
24. Select Menu Bar => Edit => Add To Panel.
    Note: Right-clicking on the setGlobal white space would have also brought up "Add to Panel" on the pull-down menu.
25. Size the Panel View to be smaller: position the Note Pad on top, the Select Num in the middle, and the OK button on the bottom. See Figure 15.9.

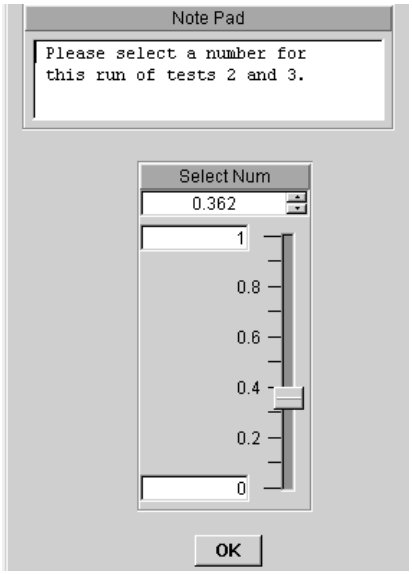


Figure 15.9. The setGlobal UserFunction panel view

26. Open the setGlobal Properties window.
27. Click on "Show Panel on Execute"; click OK.
28. Save program LAB15-3 again.

◊ 29. Run your saved program several times and observe the AlphaNumeric display; verify that each test agrees with the test constraints for pass/fail. See Figure 15.10; there will be three tests showing on your AlphaNumeric display.

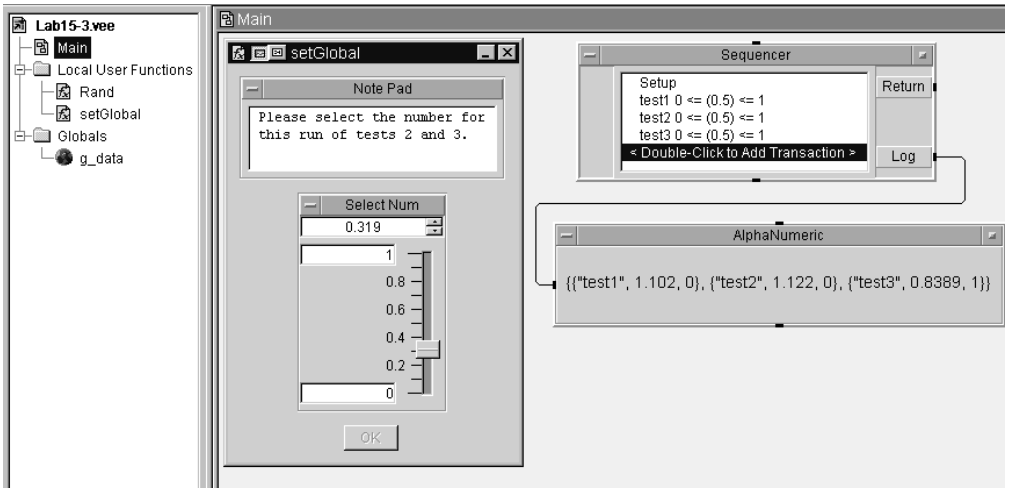


Figure 15.10. Passing data using a global variable

Note 1: Move the pop-up panel to anywhere on the screen by dragging its Title Bar.
Run this program once more using the Step Into (Ctrl + T) button to observe the data flow. Note when the OK button is activated. Observe that, once the OK button has been depressed, the Rand UserFunction generates all three test results.
Note: The OK button is depressed prior to Rand selecting the three data points for the tests.
Close this program without saving it.

Lab 15.4 – Comparing a Waveform Output with a Mask

You will learn to create a noisy waveform: "noisyWv", and call it from a Sequencer transaction. (Your user will be able to vary that waveform amplitude from 0 to 1.) You will then learn to arrange the Sequencer to test that noisy waveform.

1. Clear your Work Area, deselect the Program Explorer, and maximize the Work Area.

Comparing a waveform output with a mask

2. Create the UserFunction called noisyWv shown in Figure 15.11.

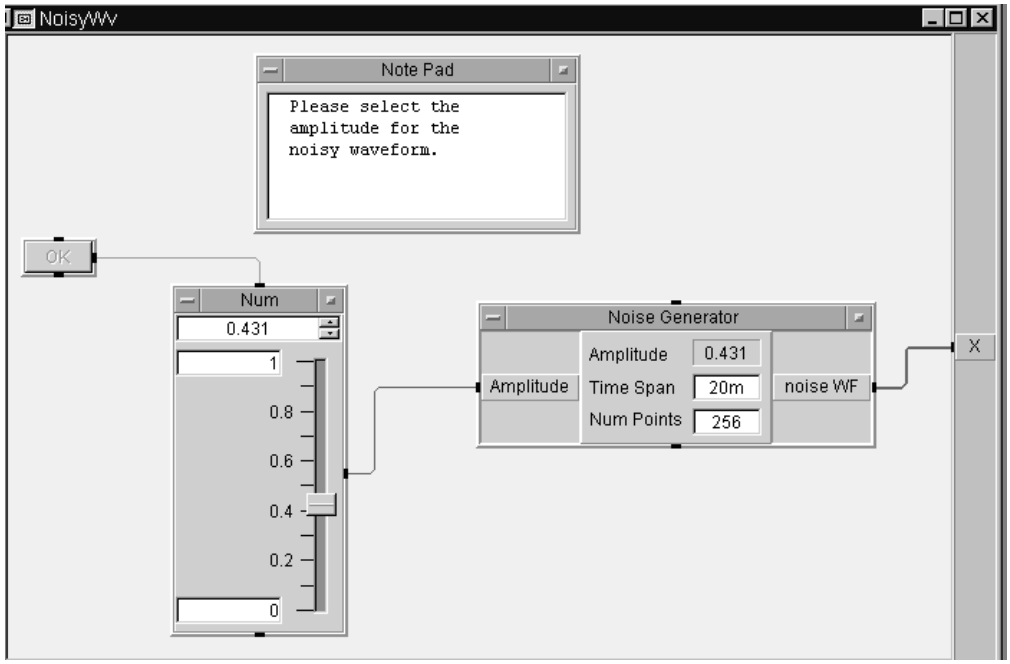


Figure 15.11. The NoisyWv Waveform UserFunction detail view

3. Change the Real64Slider Title Bar to Num; press Ctrl and click on the OK button; select Num, and Note Pad to highlight them.
4. Select Menu Bar => Edit => Add To Panel; see Figure 15.12 to arrange these objects; size the noisyWv window.

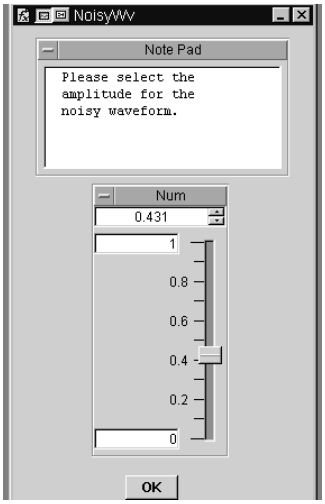


Figure 15.12. The NoisyWv Waveform UserObject pop-up panel

5. Open the object menu of the noisyWv object; click Properties.
6. Go to the Pop-up Panel, General tab; click Show Panel on Execute; click OK; minimize noisyWv.
7. Select Menu Bar => Device => Sequencer and place it in the center of Main.
8. Add a data-input terminal; name it mask.
9. Double-click the (dark blue) transaction bar to get the Sequence Transaction dialog box.
10. Type noisyWv( ) in the FUNCTION field.
11. Click RANGE and select LIMIT from the pop-up menu.
12. Type the terminal name mask in the LIMIT expression field (which is to the far right of the dialog box; it should have a default 1).
13. Click OK.
    Note: Test1 will get a result from noisyWv( ). It will test it against the limit value at the input terminal named mask. All points will pass that are less than or equal to the mask value.
14. Select Menu Bar => Data => Constant => Coord; place it above and to the left of Sequencer.
15. Connect the Coord output to the Sequencer input terminal (mask).
16. Open the Coord object menu and click Properties.
17. Click 1D Array under Configuration; enter "2" in the Size field; click OK.
    Note 1: You choose "2" because only two pairs of coordinates are needed to specify a straight line.
    Note 2: Two pairs of indices will appear.
18. Double-click on the first index which is 0000; a cursor will appear.
19. Type **0, 0.6**; press the Tab key; type **20m, 0.6**; click on the work area outside the object.
    Note 1: Parentheses are not needed; VEE Pro adds them automatically.
    Note 2: The 0.6 values will provide you with a Y-axis straight-line mask; the 0 to 20m values will provide you with an X-axis from zero to 20 millisecond.
    Note 3: Clicking on the work area outside the object tells VEE Pro that you are done with your entries.

20. Select an AlphaNumeric display; place it under the Sequencer; increase its width to be greater than that of the Sequencer; and connect its input to the Sequencer Log output.
21. Select Menu Bar => Device => Formula; place it below the Logging AlphaNumeric.
22. Right-click anywhere on Formula; select Add Terminal; Data Input; rename it Log.
23. Change the Formula expression space to: Log.test1.Result.
24. Connect its Log input terminal to the Sequencer Log output terminal.
25. Select Menu Bar => Display => Waveform (Time); place it under Formula.
26. Add a second Data Input terminal.
27. Click on Mag; turn Automatic Scaling off; change Maximum to 0.8; Minimum to -0.6.
28. Connect the Formula Result pin to Trace1 and the Coord output pin to Trace2.
29. Select Menu Bar => Display => Note Pad; enter the following in its white space.

    Reminder:

    Pass = 1

    Fail = 0

    on the Logging AlphaNumeric
30. Size the Note Pad to as small as possible without obscuring its words.
31. Go to the Properties box, select Colors; change the Display Background to White.
32. Click on Trace1; select Color; Pen 5 (blue); click on Trace2; select Color; Pen 0 (black).
33. Save this program As…LAB15-4.
34. Run this program; see Figure 15.13.

◊ 35. Run it several times, following the data flow, observing the Pass/Fail result.

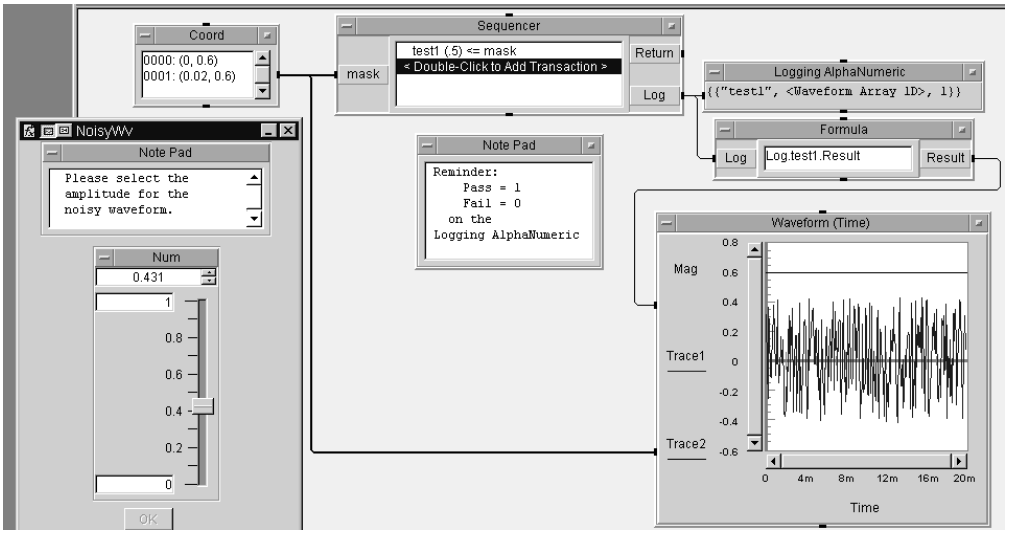


Figure 15.13. Comparing a waveform to a mask

36. Close this program without saving it.

Lesson 15 Summary

This lesson examined the VEE Pro Sequencer object, also known as the Test Sequencer. The Sequencer is a transaction-based object that has program-development benefits regarding tests, their sequencing, and automatic logging. It includes a Sequence Transaction dialog box that is extensive and needs to be explored further as you become more comfortable with VEE Pro.

Using the Test Sequencer object, tests can be configured, added, inserted, and deleted as your job assignment requires. The Sequencer stores data as a record of records. Stored data can be analyzed immediately or stored and retrieved at a later time for further, more detailed, analysis.

Previously logged (stored) test data can be accessed via the result and pass fields. A global variable can also be used to pass data. The Log Data output can be sent to a DataSet for later retrieval and searching. Which approach you choose will depend upon your expectations for later exploration and application.

Thus the VEE Pro Sequencer allows you to document tests, compare them against previously developed specifications, and compare test runs with each other.

Lab 15.1 showed you how to simulate test results using the random( ) function with a specified range of expected test results, establish a test execution order, modify that order, and retrieve specific data from the logged results.

Lab 15.2 showed you how to create a UserFunction and call it from three different tests. You also learned to call a function in the EXEC mode rather than in the TEST mode.

Lab 15.3 showed you how to import pass data using a global variable.

Lab 15.4 showed you how to create a noisy waveform: “noisyWv”, and call it from a Sequencer single-transaction bar. You then learned to arrange the Sequencer to test that noisy waveform.

You are now ready to learn to log, store, select, retrieve, and analyze data via custom menus.

Lesson 16
Logging, Storing, Selecting, and Analyzing Data

This lesson will examine further the VEE Sequencer Object to obtain analyzed data via custom menus and show you how to simulate a situation, using a menu, where the operator must choose which test to run next. The operator can then monitor the data extremes and decide what action should happen next. It consists of a pre-lab and four labs.

Lesson 16 Pre-lab Summary

The following is described in the Lesson 16 pre-lab. See Appendix E, page E-82.

- Data Selection Control
- The six Data Selection Control objects
- Comparisons of the To/From Files with other To/From Objects

As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 16.1 – Extracting Data from Records
This lab will show you how to analyze several runs of data from the Sequencer and extract part of the data to be analyzed.

Lab 16.2 – Storing and Retrieving Logged Data
This lab will show you how to use the To/From File Objects with logged data and use the To/From DataSet Objects with logged data.

Lab 16.3 – Logging Vehicle Radiator Temperature Extremes
This lab will show you how to monitor and log the Vehicle Radiator minimum and maximum temperatures via the Sequencer.

Lab 16.4 –Selecting Data via Custom Menus
This lab will show you how to use menus to guide an operator.

Thus, the VEE Sequencer allows you to document tests, compare them against previously developed specifications, and compare test runs with each other.

Records Held in the Sequencer

The Sequencer will hold records that represent each test run. The concept is shown in the three-dimensional diagram of Figure 16.1. They are the "layers" of Figure 15.3 for runs.

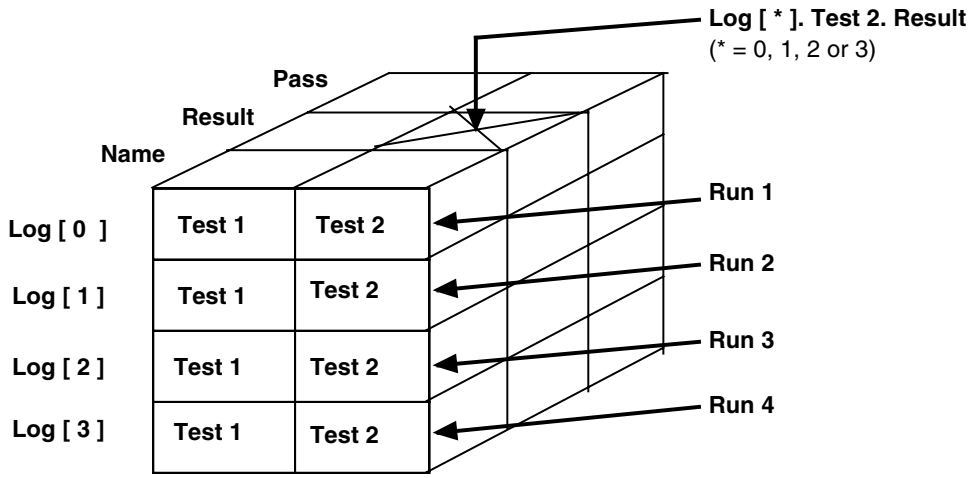


Figure 16.1. A logged array of Records of Records

An example of the display of records of records would be:

0: {{"test1", Result10, Pass (0 or 1)}, {"test2", Result20, Pass (0 or 1)}}
1: {{"test1", Result11, Pass (0 or 1)}, {"test2", Result21, Pass (0 or 1)}}
2: {{"test1", Result12, Pass (0 or 1)}, {"test2", Result22, Pass (0 or 1)}}
3: {{"test1", Result13, Pass (0 or 1)}, {"test2", Result23, Pass (0 or 1)}}

Note 1: The array of records is named Log because that name is associated with the Sequencer output pin. Accessing a run requires that you use array indexing with bracket [ ] notation.
Log[0] is the first run through the Sequencer,
Log[1] is the second run, etc.

Note 2: Within a record, such as Test1, there are three fields:
Name, Result, and Pass.
Other fields can be configured in the Sequencer properties.
Thus, Log[*].Test1.Result provides an array of four values, each one represents the Result of the one transaction "Test1" in each of the four runs shown above.
Log[0].Test1.Result provides an output scalar value that is the Result of Test1 in the first run. The entire first run (all the tests and all their values) is represented by Log[0].

Note 3: Because of the information shown in Figure 16.1 above, you may access and analyze data in a variety of ways.

Lab 16.1 – Extracting Data from Records

This lab will show you how Sequencer data are stored as a record of records. Several runs will generate an array of records of records. Each record of records represents one run through the Sequencer. Each record represents one test transaction. The concept is shown in the three-dimensional diagram of Figure 16.1 above.

1. Clear your Work Area.

Analyzing several runs of data from the Sequencer

2. Open LAB15-2a; Save As… LAB16-1 immediately.
3. Select Menu Bar => Flow => Repeat => For Count.
4. Place it in Main above and to the left of the Sequencer object.
5. Change the number of iterations to "3".
6. Connect the For Count data-output pin to the Sequencer sequence-input (top) pin.
7. Delete the data line between the Sequencer Log pin and the display.
8. Select Menu Bar => Data => Collector; place it to the right of the Sequencer.
9. Connect the Collector upper-left (Data) input pin to the Sequencer Log pin; connect its lower-left XEQ pin to the For Count sequence-output (bottom) pin.
10. Connect the Collector data-output pin (Array) to the AlphaNumeric display input pin.
11. Enlarge the display vertically to accommodate an array with three elements.
12. Select Menu Bar => Display => Note Pad; place it to the right of the AlphaNumeric object.
13. Type the following into the Note Pad white space:

    REMEMBER: Pass = 1
    Fail = 0

14. Save your program again; run this program. See Figure 16.2.

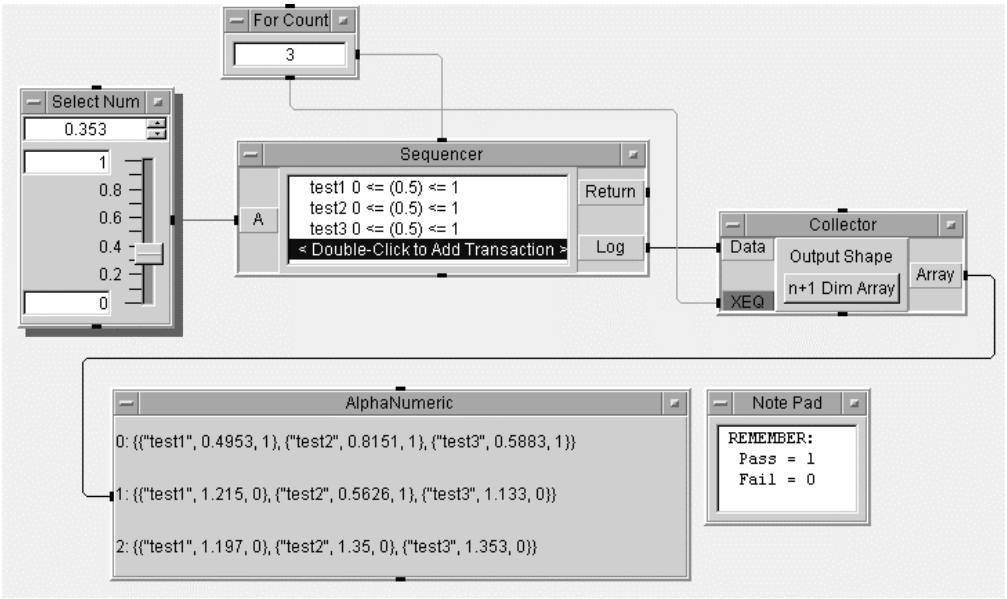


Figure 16.2. Analysis of three runs of Sequencer Data

Extracting a portion of the data to be analyzed

15. Select Menu Bar => Device => Formula; place it under Select Num and to the left AlphaNumeric.
16. Connect the Formula input pin to the output (Array) of the Collector.
17. Change the Formula expression field to read a[*].test1.result.
18. Add another Formula Object; change its title bar to mean(x); connect its input to the Formula Object data output (Result) pin.
19. Change the mean(x) input-pin name to "x".
20. Change the mean(x) expression formula to mean(x).
    Note: Steps 18 through 20 can be accomplished with the Menu Bar => Device => Function & Object Browser => mean(x) => Create Formula.
21. Connect a new AlphaNumeric display to mean(x); change it title bar to **mean**; connect its input (x) pin to the Formula output (Result) pin; see Figure 16.3.
22. Save your program as LAB16-1a.

◊ 23. Run this program several times, following the data flow.

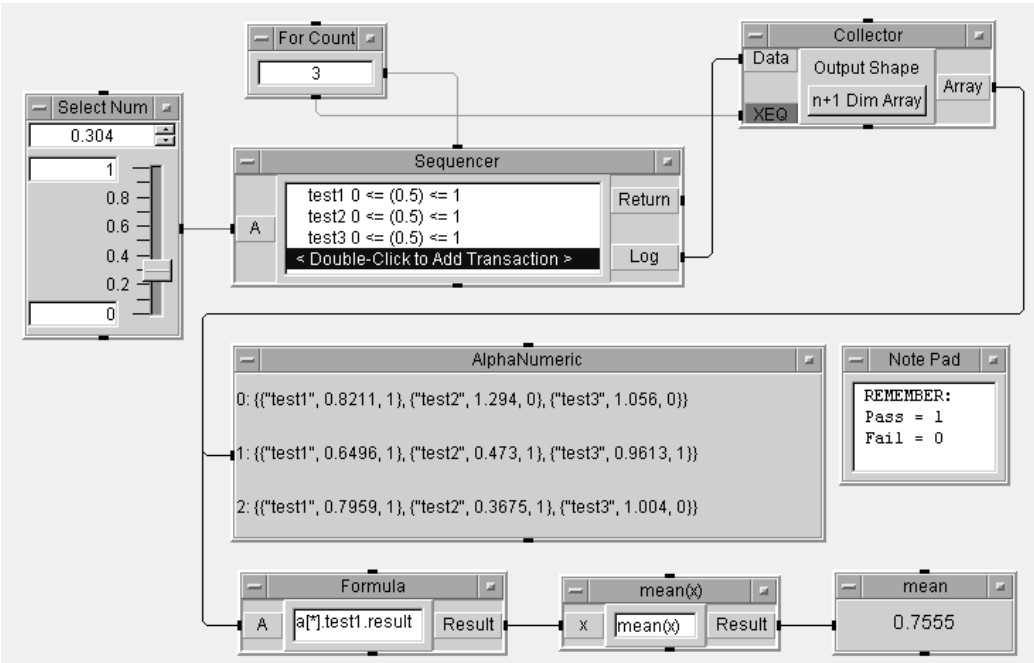


Figure 16.3. Layout, Sequencer Data, and Calculation of Mean

Note: You can easily change which fields are saved to the Sequencer Log output by opening the Logging folder in the Sequence Properties dialog box.

Lab 16.2 – Storing and Retrieving Logged Data

This lab will show you how to adapt an existing program by using the To/From File objects for data storage. Then you will learn how to use the To/From DataSet objects.

1. Clear your Work Area.

Using the To/From file objects with logged data

2. Open LAB15-3; Save As… LAB16-2 immediately; open Program Explorer and double-click on global; click on the upper-left f_x and select Cut; delete the data line to the AlphaNumeric display.
3. Select Menu Bar => Flow => Repeat => For Count; place it to the left and above the Sequencer; change the For Count number to "3"; connect the For Count data-output (right) pin to the Sequencer sequence-input (top) pin.
4. Select Menu Bar => Data => Continuous => Real64Slider; place it to the left of Sequencer; change its title bar to Select Num; connect it to the Sequencer input **A**.
5. Place the Sequencer object near the top of the Main; place the AlphaNumeric display near the bottom.

6. Select Menu Bar => Data => Collector; place it to the right of Select Num and to the left of and below the Sequencer.
7. Connect the Sequencer Log pin to the Collector data-input pin.
8. Connect the For Count sequence-output (bottom) pin to the Collector XEQ pin.
   Note: The Collector will create an array of "records of records" from the Sequencer. You can write any VEE data container to a file by using the WRITE CONTAINER transaction in the To File object.
9. Select Menu Bar => I/O => To => File; place it to the right of the Collector; shrink it vertically; place a checkmark in Clear File At PreRun & Open.
10. Double-click on the To File blue transaction bar; I/O Transaction will appear.
11. Select WRITE in the left-most bar; CONTAINER in the second bar from the left; type **a** if it is not there in the right-most white area; click OK.
12. Select Menu Bar => I/O => From => File; place it below the To File object and shrink it vertically.
13. Double-click on the From File blue transaction bar; I/O Transaction will appear.
14. Select READ in the left-most bar; CONTAINER in the second bar from the left; type **x** if it is not there in the right-most white area; click OK.
15. Connect the Collector output (Array) to the To File input.
16. Connect the To File sequence output pin (bottom) to the From File sequence-input (top) pin.
17. Connect the From File data output (X) to the AlphaNumeric display input pin (Data).
    Note: You can use the default data file name for storage.
18. Convert To File and From File to icons; move them upwards towards the Sequencer; then move AlphaNumeric up and lengthen it vertically so its results can be viewed.
19. Save this program again.

◊ 20. Run this program several times. Click Run, then click the OK button on the panel three times. It should look like Figure 16.4.

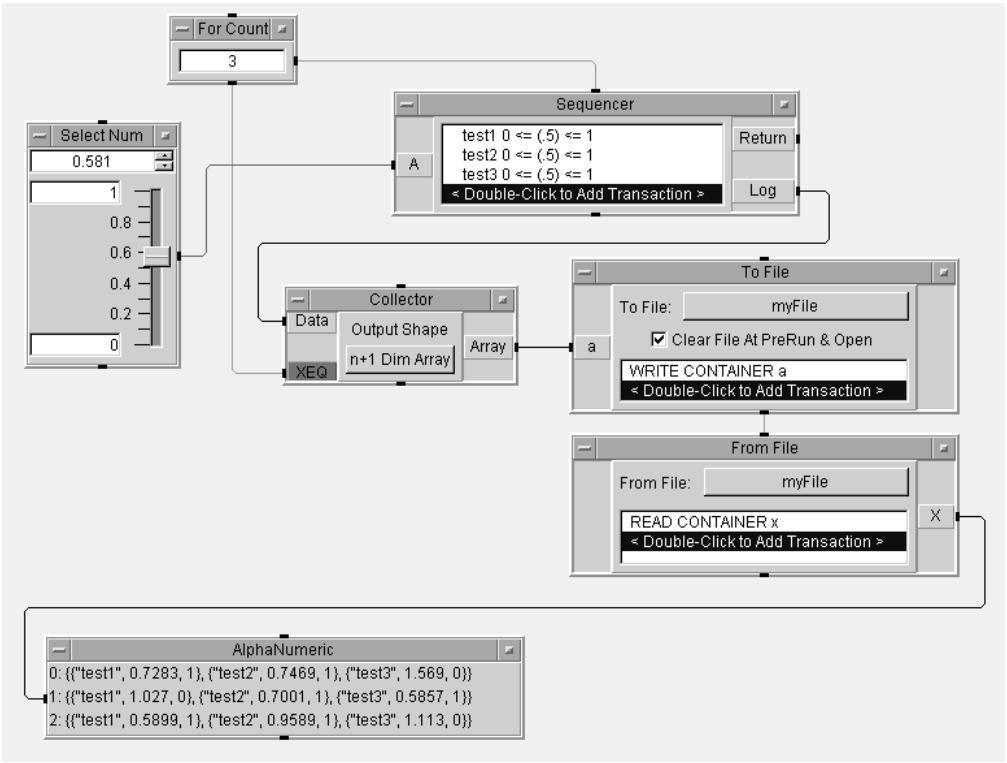


Figure 16.4. Storing Logged Data with To/From File

Using the To/From DataSet objects with logged data

Note: You do not necessarily need a Collector because you can append each run of the Sequencer to the end of the DataSet. It continues to be shown in Figure 16.5.

21. Select Menu Bar => File => Open => LAB16-2; Save As… LAB16-2a immediately.
22. Modify this program to look like Figure 16.5; replace To File and From File with To Data Set and From Data Set; for the From Data Set object, change Get Records to All.

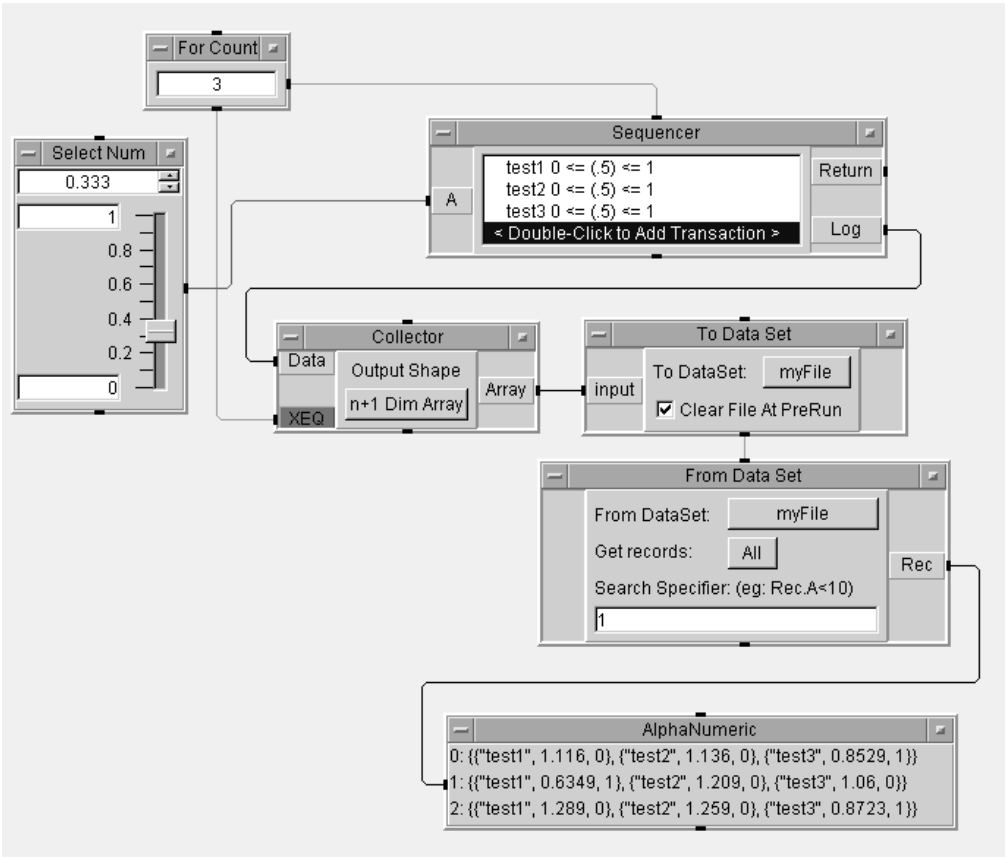


Figure 16.5. Storing Logged Data with To/From DataSet

Note: The Search Specifier feature in the From DataSet object can turn your data into useful information.

23. Save this program again.

◊ 24. Run this program several times; observe the data flow.

Lab 16.3 – Logging Vehicle Radiator Temperature Extremes

You will devise a program that will monitor and log the Vehicle Radiator minimum and maximum temperatures via the Sequencer.

Monitoring and recording Vehicle Radiator temperature extremes

1. Clear your Work Area.

2. Open LAB 16-2; save it as LAB 16-3 immediately.
3. Move the screen objects to the right.
4. Cut the Select Num object; save LAB 16-3 again; close LAB 16-3.
5. Open LAB 14-4; then open its Vehicle Radiator User Function.
6. Hold down the Ctrl key; click on the following three objects: Temp Generator, A+B, and Temp Noise Generator; copy these objects.
7. Close LAB 14-4 without saving it; open LAB 16-3 again; paste the copied objects on the left side of LAB 16-3.
8. Connect the A+B object output (Result) to the Sequencer input terminal **A**.
9. Change the value of For Count to **4**. The layout is shown in Figure 16.6.

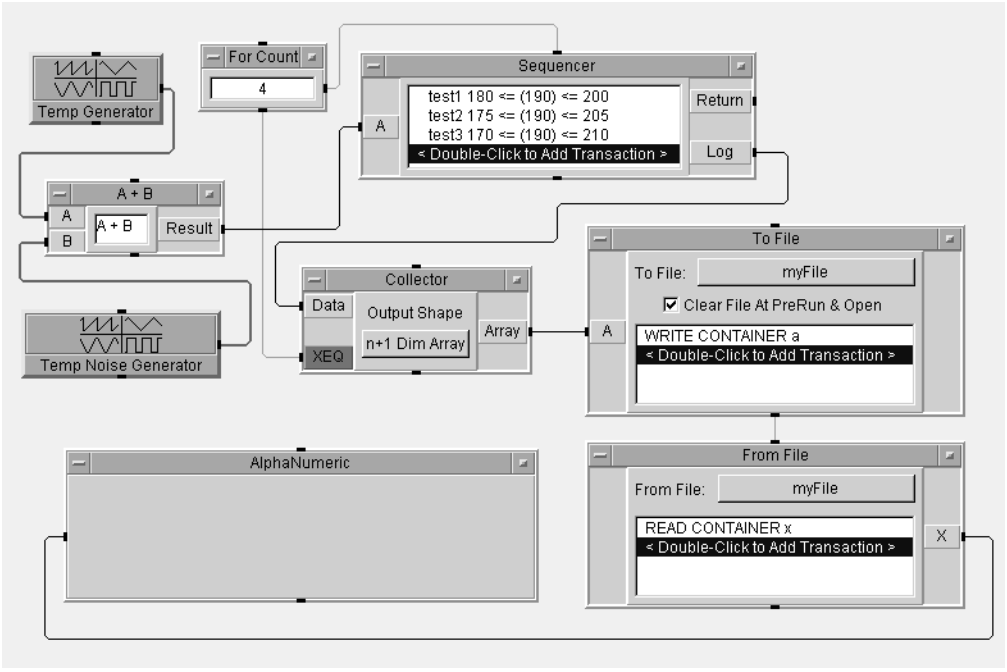


Figure 16.6. Partial layout of LAB 16.3

10. Save LAB 16-3 again.

Modifying Vehicle Radiator test limits

11. Open each Sequence Transaction dialog box; modify the three tests as follows:

Test Number	Spec Nominal	Range Min	Range Max
test1	190	180	200
test2	190	175	205
test3	190	170	210

12. Cut the following three objects: To File, From File, and AlphaNumeric.

Building a Vehicle Radiator test record

13. Select Menu Bar => Data => Build Data => Record; place it in the upper-right corner of Main.
14. Add an input terminal to Build Record.
15. Change the three input-terminal names to testname, time, and data.
16. Select Menu Bar => Data => Constant => Text; place it to the left of Build Record.
17. Change its Title Bar and string value to Test Family.
18. Connect the Test Family output to the testname input of Build Record.
19. Select Menu Bar => I/O => To => String; place it below the Sequencer.
20. Open the To String and its I/O Transaction dialog box; double-click to add a transaction.
21. Change **a** to **now()**; select Time Stamp Format;
    change Date & Time to DD/Month/YYYY, HH:MM, 12 HOUR; click OK.
22. Connect the To String output terminal to Build Record **time**.
23. Cut the Sequencer **Log** connection; connect the **Log** to Build Record **data** input.
24. Connect the Build Record output terminal to the Collector **Data** input terminal.

Logging and monitoring Vehicle Radiator test data

25. Select Menu Bar => Display => Logging AlphaNumeric; place it below To String.
26. Enlarge the Logging AlphaNumeric to cover most of the lower part of the screen; change its title to Record of Vehicle Radiator Tests; toggle Show Terminals; connect its **Data** input terminal to the Collector **Array** output terminal.
27. Select Menu Bar => Device => Formula & Object Browser; select
    Type: Built-in Functions; Category: All; Functions: max.
28. Click OK; place it below and to the left of the "Record of the Vehicle Radiator Tests" object; connect its input terminal to the A+B object output terminal.
29. Select Menu Bar => Display => AlphaNumeric; place it to the right of max(x).
30. Change its name to Maximum Temp; connect its input to the output of max(x).
31. Select Menu Bar => Device => Formula & Object Browser; select
    Type: Built-in Functions; Category: All; Functions: min.
32. Click OK; place it below and to the right of the "Maximum Temp" object; connect its input terminal to the A+B object output terminal.
33. Select Menu Bar => Display => AlphaNumeric; place it to the right of min(x).
34. Change its name to Minimum Temp; connect its input to the output of min(x).
35. Insert a Note Pad to the Right of To String; write the note:
    Remember:
    Pass = 1
    Fail = 0
36. Arrange the objects similar to those in Figure 16.7.
    Note: If your screen looks cluttered, then right-click your mouse on the screen white space and select Clean Up Lines.

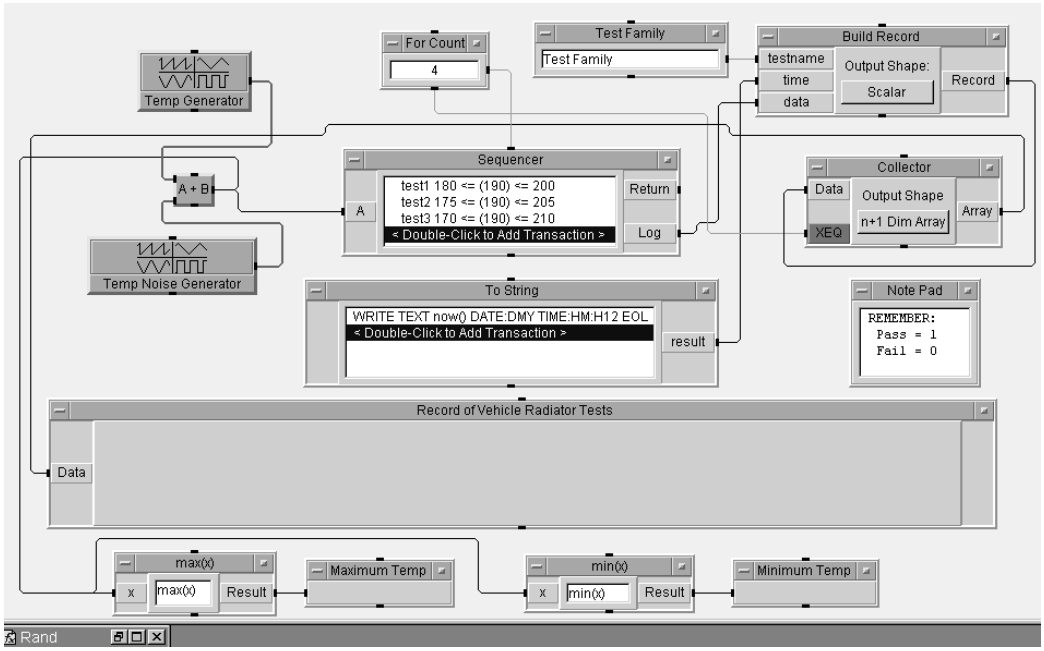


Figure 16.7. The completed LAB 16.3 before running

37. Close the A+B object; open the Temp Noise Generator object.
38. Run this program; it should like Figure 16.8.
    Note: The reason "Waveform Array 1D" is displayed under the "Record of Vehicle Radiator Tests" is to describe the output of the combined Temperature Generator and Temperature Noise Generator. To observe the reason for these titles, double-click on the output of Temperature Noise Generator (Noise WF) or Temperature Generator (Func) and observe the Container Information under Type and Shape. Since the waveform data contains 256 values, it would not be feasible for the Logging AlphaNumeric to show all these values. Thus, the string "Waveform Array 1D" is shown to indicate that there are many values available that are not displayed.

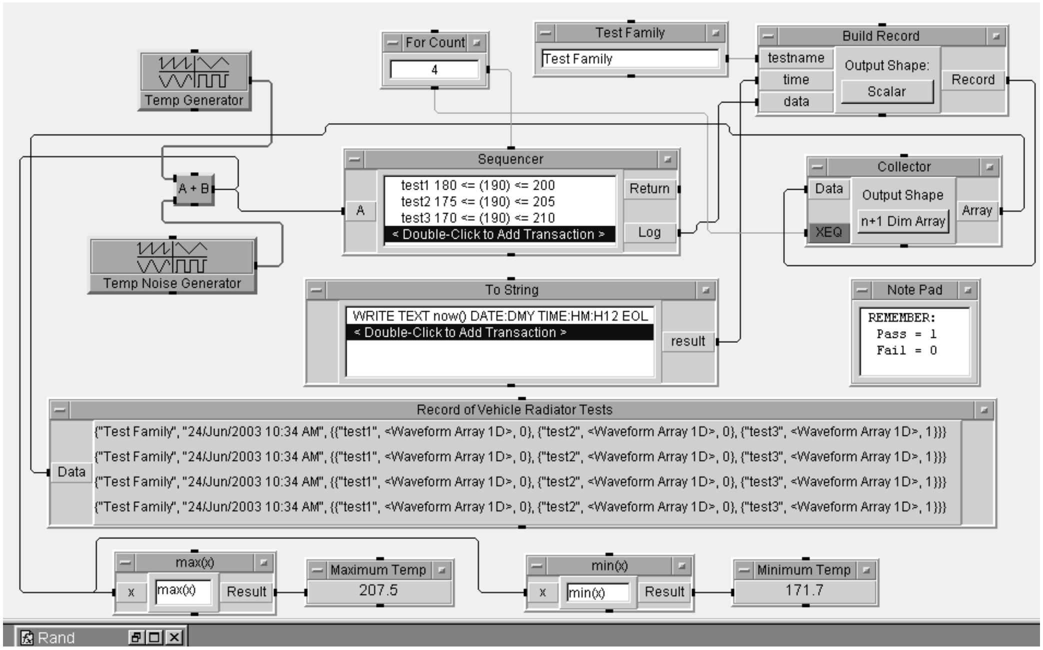


Figure 16.8. LAB 16.3 after running

◊ 39. Vary the setting of the Temp Noise Generator amplitude by at least a factor of two; run this program again. What is the effect on the Pass/Fail” test results, and why?

◊ 40. Revise the three test extremes until you are satisfied that their settings meet with your preferred limits once the tests are run.

Your notes:

____

____

____

____

____

41. Return the settings of this program to those you prefer to be stored.
42. Save and close this program.

Lab 16.4 – Selecting Data via Custom Menus

This lab will show you how to simulate a situation, using a menu, where the operator must choose which test to run next.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Using operator-interface menus to guide an operator

2. Select Menu Bar => Device => UserFunction.
3. Select Menu Bar => Display => Picture; place it in the upper-right corner of the UserFunction.
4. Open the Picture object menu, click Properties; deselect Show Title Bar under Open View.
5. Select die1.gif under Picture; click Scaled; then OK.
   Note: You should now have a colorful picture of a die with one dot on its top.
6. Select Menu Bar => Flow => Confirm (OK); place it below the die.
7. Connect the Picture sequence-output (bottom) pin to the OK sequence-input (top) pin.
8. Hold down Ctrl; select both the Picture and the OK button; a shadow will occur around both of these objects.
9. Place the mouse pointer on the background and press the right-hand mouse button.
   Note: This will open the pop-up Edit menu.
10. Select Add to Panel from the pop-up Edit menu.
11. Change the Pop-Up Panel "Panel Title" and the UserFunction Properties name to die1.
12. Move the OK button so it is centered under the die.
13. Go to the die1 Properties dialog box; select Show Panel on Execute.
14. Click the Properties box Panel folder tab; change the grid size to 2 for more precise alignment; click OK.
15. Select Clone twice in the Die 1 object menu to create two more user functions.
    Note: The new user functions will automatically be named die2 and die3.
16. Rename the picture objects die2 and die3; go to each UserFunction Properties box and select Panel; select die2.gif and die3.gif respectively; click OK each time.
17. Return to Main window.
18. Select Menu Bar => Data => Selection Control => Radio Buttons; place it in the upper-left corner of your Main window.
    Note 1: You now have a menu that will allow selection of one of your three user functions to be called later.
    Note 2: The Radio Buttons will create an enumerated value with an associated ordinal number.
    Example:
    Ordinal number "0" Monday
    Ordinal number "1" Tuesday
    etc. (Ordinal number "6" will be "Sunday.)
19. Open the Radio Buttons object menu; select Edit Enum Values… .
20. Access the radio buttons individually via the Tab key:
    Double-click Item 1 and overtype with die1.
    Double-click Item 2 and overtype with die2.
    Double-click Item 3 and overtype with die3; click OK.
21. Open the Radio Buttons object menu; click Properties.
22. Select Auto Execute under Execution.
23. Change the Title Bar to the prompt – Make a Selection: and click OK.

24. Select Menu Bar => Device => Call; place it to the right of Make a Selection. (You now have the Call object with its Function Name control pin added. This will accept an Enum or Text value as an input.)
25. Right-click on Call Function => Add Terminal => Control Input; click OK.
26. Connect the Make a Selection data-output (upper right) pin to the Call Function object Function Name input terminal.
27. Connect the Radio Buttons sequence-out (bottom) pin to the sequence-in (top) pin of Call Function. See the top two icons of Figure 16.9.

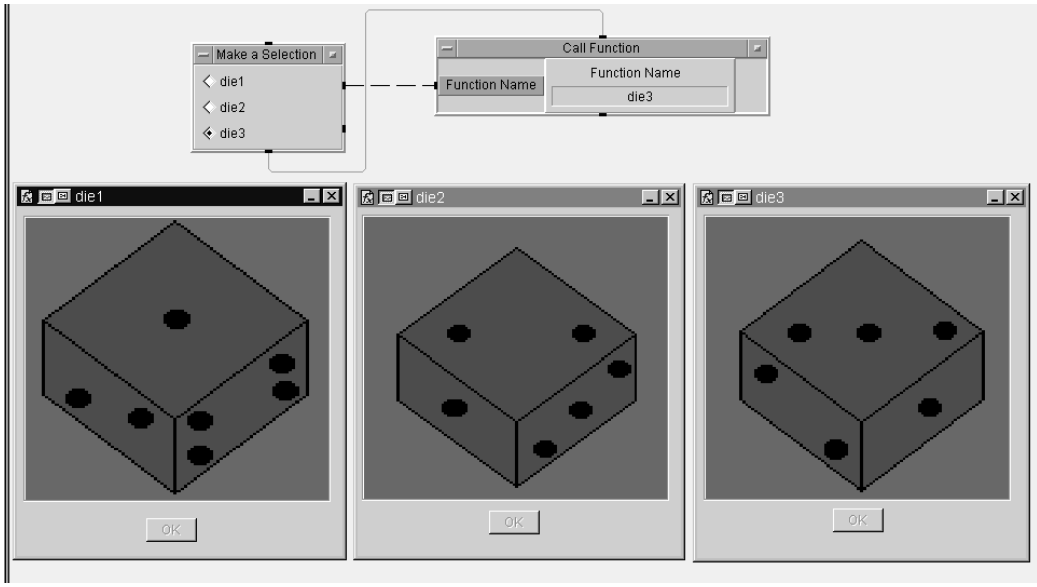


Figure 16.9. Detail view of the dice program

Note 1: VEE will automatically convert an Enum Scalar to a Text Scalar.
Note 2: The control pin on Call Function will replace 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. Otherwise, the Call Function box would execute with no die selected.

28. Select Menu Bar => Display => Picture; open its Properties; select die1.gif; click OK.
29. Select the UserFunction Properties box; select Menu Bar => Edit => Add to Panel for each of the three UserFunctions; right-click each picture and resize each die; click OK to store each set of adjustments. See Figure 16.9 above.
30. Return to Main – Panel View; select the appropriate radio button to choose a die; click OK once it appears.
31. Save your program as LAB16-4.

◊ 32. Run your program several times by selecting a Radio button. See Figure 16.10 where **die2** was selected.

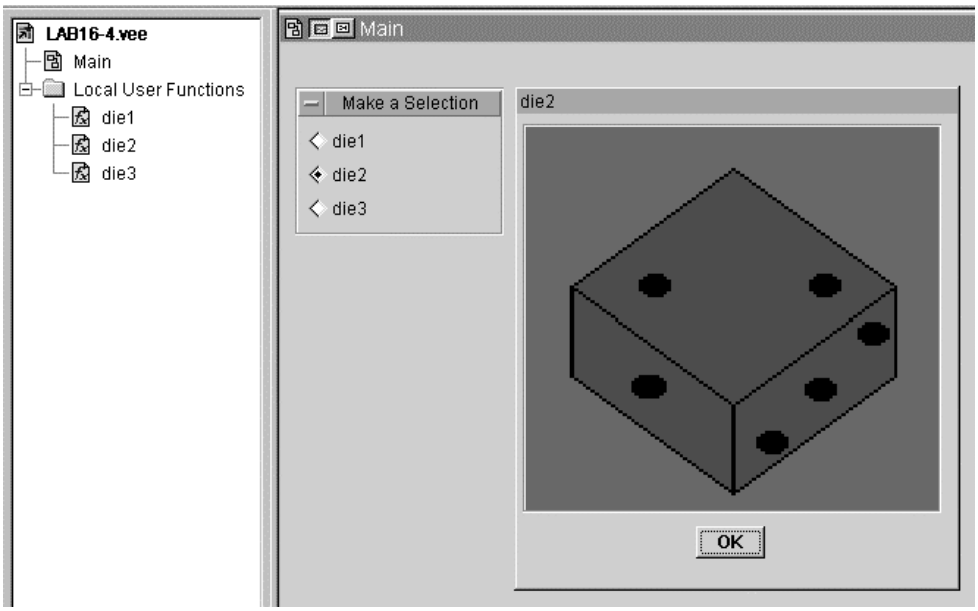


Figure 16.10. Panel view of the dice program

Note 1: Do not use the run button because it will only use the selection that is already selected on the menu.
Note 2: Radio Buttons can also be used to select a compiled-language program by using the Execute Program object with a control pin ("Command") that indicates which program to call.
Note 3: You could improve the program Run Time by using the File Name data-input pin on the Picture object inside a single UserFunction. Then send the appropriate bitmap file to the object. See LAB 18-2. You could use this alternate approach for any labs that use Picture objects.

Lesson 16 Summary

This lesson further explored the Sequencer Object, also known as the Test Sequencer. Using the Test Sequencer Object, tests can be configured, added, inserted, and deleted as your job assignment requires. The Sequencer stores data as a record of records. Stored data can be analyzed immediately or stored and retrieved at a later time for further, more detailed, analysis.
Previously logged (stored) test data can be accessed via the result and pass fields. A global variable can also be used to pass data. The DataSet object can be used to pass data. Which approach you choose will depend upon your expectations for later exploration and applications.

Lab 16.1 showed you how the Sequencer data are stored as a record of records. Several runs generated an array of records. Each record represented one run through the Sequencer. The Sequencer holds other records that represent each test run.

Lab 16.2 showed you how to adapt an existing program by using the To/From File objects for data storage. You then learned how to use the To/From DataSet objects.

Lab 16.3 showed you how to monitor and log the Vehicle Radiator minimum and maximum temperatures with the Sequencer object.

Lab 16.4 showed you how to simulate a situation, using a menu, where the operator must choose which test to run next.

You are now ready to learn to apply graphical operator interfaces and use MATLAB® to graphically display test data.

Lesson 17

Applying Graphical Operator Interfaces and Filtering

This lesson will apply operator interfaces to a variety of tasks such as using menus, creating status panels, importing bitmaps for panel backgrounds, creating a high-impact warning, and plotting via MATLAB®. This lesson will restrict itself to applying a few operator-interface fundamentals to customizing user interfaces. It consists of a pre-lab and five labs.

Lesson 17 Pre-lab Summary

The following are described in the Lesson 17 Pre-lab. See Appendix E, page E-85.

- Status Panel
- Bitmap
- Nested UserFunctions
- an overview of filter theory

Details of accessing and applying the operator interfaces are provided in Appendix C.

The filter-theory material in Appendix E is included for those of you who design analog and digital filters. Included with VEE Pro is a special, pre-devised program for Digfilt – a Third Order Elliptical filter. As noted in Lesson 1, Appendix B includes a cross-reference to each of these items and to all objects and subprograms in the labs of this and later lessons.

Overview

Lab 17.1 – Creating a Status Panel
This lab will show you how to create a status panel that provides test-in-progress information.

Lab 17.2 – Importing Bitmaps for Panel Backgrounds
This lab will show you how bitmaps can add impact to your test displays.

Lab 17.3 – Creating a High Impact Warning
This lab will show you how to nest UserFunctions.

Lab 17.4 – Exploring a Pre-Designed Digital Filter Program
This lab will show you how to explore the capability of a VEE Pro supplied digital filter program.

Lab 17.5 – Using MATLAB® to Display the Pre-Designed Digital Filter
This lab will show you how to use MATLAB® to graph the Agilent digital filter output waveform versus time for a variety of excitations.

This is the last lesson that will directly assist you in the development of programs. The next, and final, lesson will assist you in learning how to improve your programs and devised tests, better analyze those tests, and present the results of those tests to others so they can decide with you what to do next.

Lab 17.1 – Creating a Status Panel

This lab will show you how to create a status panel and use this panel to provide you with status information regarding your in-progress test. This lab could be devised as a modular program (UserFunction) that could be called upon later.

1. Clear your Work Area, deselect the Program Explorer, and maximize the Work Area.

Creating a status panel for in-progress test

2. Select Menu Bar => Device => Sequencer; place it on the left-center of your Work Area; double-click on its transaction bar.
3. Leave the default name test1.
4. Replace the FUNCTION: field with random( ); click OK. See Figure 17.1.

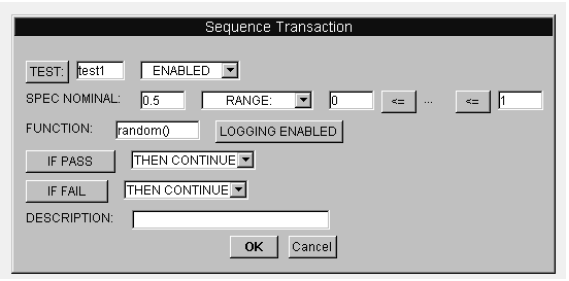


Figure 17.1. Configuring test1

5. Double-click in the Sequencer object on the white space below test1; configure a second and identical test named test2. Replace FUNCTION field with random(); click OK.
   Note: See Figure 17.1 again.
6. Open the Sequencer Properties box; choose the Logging tab.
7. Go to Logging Mode; set Log Each Transaction To:logTest(thisTest).
8. Click OK.
   Note: As the Sequencer executes each transaction, it will create a record for each test – "thisTest" – whose fields can be configured under the same tab. Then you can create a UserFunction such as

"logTest". The Sequencer will call your LogTest( ) UserFunction with the Record "this Test" at the end of each executed transaction. Your status panel will update itself automatically.

9. Select Menu Bar <= Device <= Function & Object Browser; select Type: Built-in Functions; Category: Panel; Member: showPanel.
10. Click on Create Formula; place it above the Sequencer.
11. Delete the five input pins; then edit the showPanel parameters within the parentheses to read: showPanel("logTest",420,180); reduce the Sequencer's vertical size.
12. Connect the Result (output) terminal from showPanel to the Sequencer input (top) terminal.
13. Select Menu Bar => Device => UserFunction; name it logTest.
14. Add a logTest input pin. (The logging Record "thisTest" will be the input.)
15. Select Menu Bar => Display => Logging AlphaNumeric; place it in the logTest UserFunction; deselect Clear at Activate; size it; connect it to the input pin as shown in Figure 17.2.

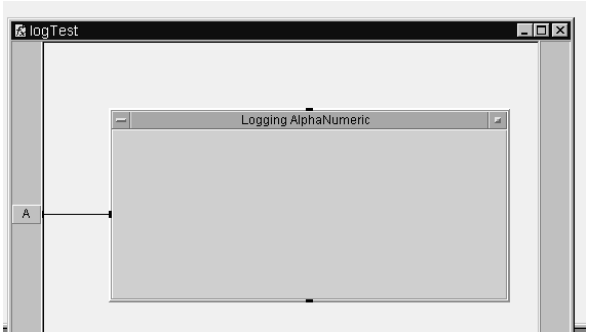


Figure 17.2. Detail view of the UserFunction logTest before running

16. Select the Logging AlphaNumeric Object and make it a panel: click Edit => Add to Panel.
17. In the Panel View, adjust the sizing and placement to satisfy yourself.
18. Deselect the display and Panel View Title Bars in the Logging Alphanumeric Properties box to simplify the later presentation. Run this program. See Figure 17.3.

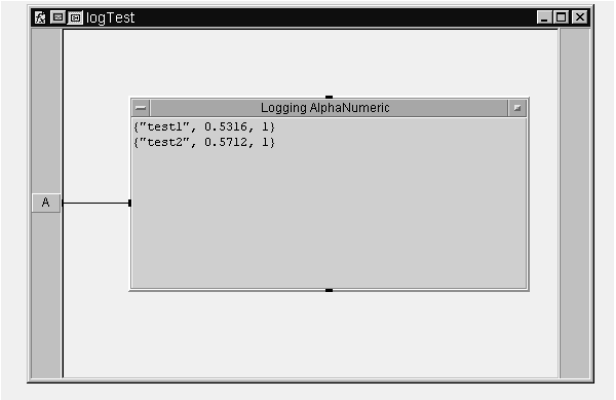


Figure 17.3. Detail view of the UserFunction logTest after running

19. Return to the Main window.
20. Select Menu Bar => Flow => Confirm(OK); place it below Sequencer.
21. Select Menu Bar => Device => Function & Object Browser; choose Type: Built-in Functions; Category: Panel; Member: hidePanel; click Create Formula; click OK.

    or

    Go to the Explorer window; right-click on the logTest function; Generate; HidePanel; an automatically labeled object will appear.
22. Place the Function & Object Browser below the "OK" button.
23 Change the hidePanel parameter (within the parentheses) to "logTest"; delete its data-input terminal.
24. Connect the objects as shown in Figure 17.4.

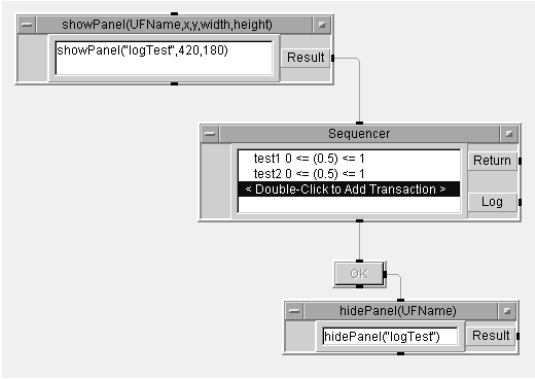


Figure 17.4. Main program before running

25. Save As… LAB17-1.
26. Run your program. It should look like Figure 17.5. Step through this program to see the status panel appear, update with the results from Test1, update with the results from Test2, then disappear.

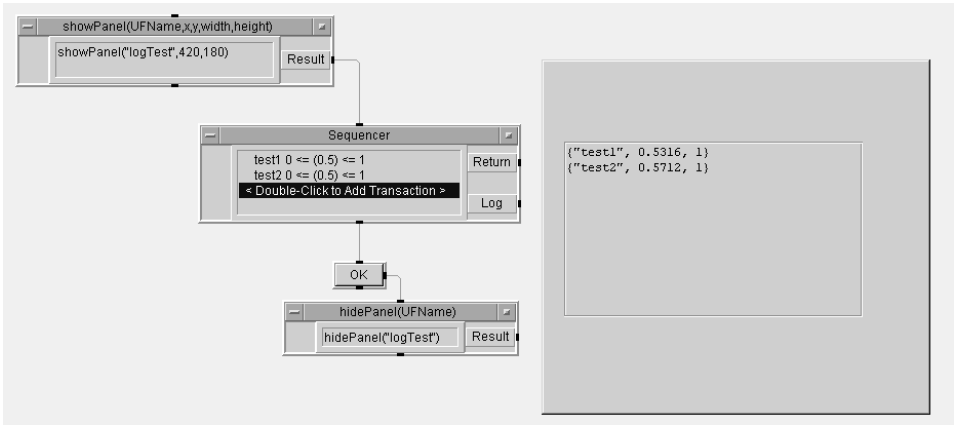


Figure 17.5. Main program after running with status panel

Note 1: What does the "1" in each logTest line represent? Change the limit to "2" and observe the result of a new run to see if it has any effect on the "1". Experiment with other variations to determine the meaning of the "1".

Note 2: You have used a Confirm OK object to trigger the hidePanel object; you could place it elsewhere in your program to time its execution.

◊ 27. Remove checkmarks from "Clear at Activate" and "Clear at PreRun"; run your program several times; observe the output display.

◊ 28. Add a checkmark to "Clear at PreRun"; run your program several times; observe the output display.

◊ 29. Remove checkmark from "Clear at PreRun"; add checkmark to "Clear at Activate"; run your program several times; observe the output display.

30. Save your final program again if you are satisfied with the ending checkmark status. Or else, close your program without saving to return to the original settings.

Lab 17.2 – Importing Bitmaps for Panel Backgrounds

This lab will show you how bitmaps can add impact to your test displays. Bitmaps can be imported for icons, for the Picture Object, or for the panel view backgrounds in a UserObject or a UserFunction.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Importing a bitmap for a panel background

2. Select Menu Bar => Device => UserFunction.
3. Select Menu Bar => Flow => Confirm (OK); select Menu Bar => Display => Label; place them in the UserFunction window.
4. Change the name of the UserFunction to Bitmap.
5. Select the OK and the Label objects; highlight them with a shadow.
6. Open the pop-up Edit menu by placing the pointer on the UserFunction Work Area and clicking on the right-hand mouse button.
7. Select Add to Panel.
8. Open the Bitmap object menu; select Properties.
9. Select Show Panel on Execute.
10. Deselect Show Title Bar under Pop-up Panel.
11. Open the Panel folder tab; change the Grid Size to 2.
12. Select default.gif and Scaled under Background Picture.
13. Click OK.
14. Open the Properties box for the Label object by clicking it with the mouse right-hand button; change the title to Bitmap Function.
15. Click Center Justify.
16. Open the Colors folder; select Light Gray for the Background of the label.
17. Click OK.
18. Re-open the Label Properties dialog box; open the Fonts folder; choose a larger font with bold type.
19. Select Automatically Resize Object on Font Change.
20. Open the Appearance folder tab; go to Border; choose Raised; click OK.
21. Convert (minimize) Bitmap to an object.
22. Go to the Main window.
23 Select Menu Bar =>Device => Call; go to its object menu and click on Select Function.
24. Choose Type: Local User Functions; Category: <All>; Member: Bitmap.

25. Click OK.
26. Save As… LAB17-2.
27. Run this program. The pop-up box should look like Figure 17.6.

Figure 17.6. The Bitmap Function

Note 1: To secure this program from alteration:

a. Create a panel view for your run-time version of the program.
b. Save the program so you have a copy you can alter.
c. Select Menu Bar => File => Create Run Time Version… .

Note 2: VEE will automatically use a *.vxe extension to indicate a secured version.

Lab 17.3 – Creating a High Impact Warning

This lab will show you how to nest UserFunctions. As an example:

- The first function will be the alarm, which will display a big red square and then beep.
- The second function will call the alarm to beep repeatedly. It will cause a blinking-light effect and a pulsing sound until the user turns off the alarm.

For a specific application of a warning, see previously presented Lab 14.4.

1. Clear your Work Area, do not deselect the Program Explorer, maximize Main.

Creating a high impact warning

2. Select Menu Bar => Device => UserFunction; change the name to alarm; click OK.
3. Select Menu Bar => Display => Beep; place it in the upper-left corner of the "Alarm" UserFunction.

4. Adjust the settings so you have a loud beep:
   Frequency (Hz): 1000
   Duration (sec): 1
   Volume (0-100): 100
5. Select Menu Bar => Display => Indicator => Color Alarm; place it in the upper-right corner of "alarm".
6. Open the Color Alarm object menu; click Properties.
7. Deselect Show Title Bar; click Layout: Rectangular.
8. Go to Limits; delete Mid Text and Low Text; click OK.
9. Select Menu Bar => Data => Constant => Real32; place it to the left of the green Alarm button; change its value to 1.
10. Connect the Real32 output (right) pin to the Color Alarm input (left) pin.
    Note: The Alarm will receive the value of **1** on its output which, at runtime, will change its color to red. You may customize the display to have different high and mid-limit values. You may even change those values at runtime via a control input.
11. Select Menu Bar => Flow => Delay; place it below Color Alarm; set it to 1; connect its sequence-input (top) pin to the Color Alarm sequence-out (bottom) pin.
    Note 1: This will synchronize the Beep object with the display on the screen.
    Note 2: On some pc-Windows machines, the Beep object is no longer supported. Thus, the duration and frequency in the Beep Objects will not work.
12. Select Menu Bar => Display => Note Pad; add the message:

    TURN OFF INSTRUMENTS!

    size the Note Pad to emphasize the instructions. See Figure 17.7.

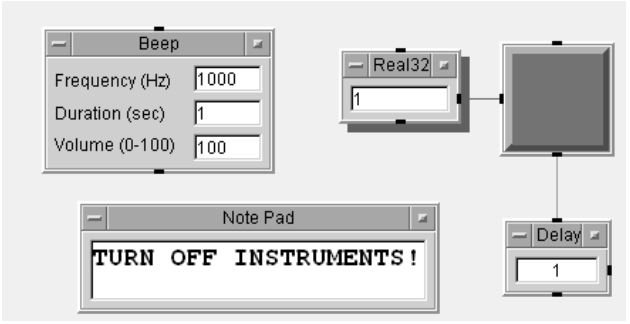


Figure 17.7. Detail view of the UserFunction: alarm

13. Go to the Main window.
14. Select Menu Bar => Device => Call.
15. Go to the Call Function object menu; choose Select Function.
16. Select Type: Local User Functions; Category: <All>; Member: alarm.
17. Click OK.
18. Save As… LAB17-3.

19. Test your program by running it. In the "alarm" UserFunction window, the Color Alarm button should now be red. Stay in the alarm window.
20. Select the Color Alarm display and the Note Pad.
21. Use the mouse right-hand mouse button; open the pop-up Edit menu; select Add To Panel; size and arrange the objects so the Note Pad is under the High (Color Alarm) object.
22. Open the Note Pad object menu; click Properties.
23. Go to Open View and deselect Show Title Bar; go to Editing and deselect Enabled; open the Fonts folder tab; go to Object, Text: deselect Default; select Arial Black, Size: 14; click OK.
    Note: Arial Black avoids the need to select the Font Style: Bold.
24. Select Automatically Resize Object on Font Change; click OK.
    Note: The Properties dialog box will automatically close.
25. Return to the Panel View; go to the Color Alarm Properties dialog box; select the Appearance folder; change the Border to: Raised
26. Repeat Step 25 for the Note Pad.
27. Double-click on the alarm Title Bar to get its Properties dialog box.
28. Select Show Panel on Execute; deselect Show Title Bar.
29. Click OK.
30. Convert (minimize) alarm to an icon.
31. Go to the Main window; delete the Call object.
    Note 1: VEE will still hold the function alarm in memory.
    Note 2: The following steps will cause the alarm function to be repeatedly called.
32. Select Menu Bar => Device => UserFunction; change the name of UserFunction to warning; click OK.
33. Select Menu Bar => Flow => Repeat => Until Break; place it in the upper-left corner of warning.
34. Select Menu Bar => Device => Call; place it below Until Break; change the Function Name to alarm.
35. Connect the Until Break data-output (right) pin to the sequence-input (top) pin of Call alarm.
    Note: The following steps will ask users if they want to turn off the alarm.
36. Select Menu Bar => Data => Toggle Control => Check Box; place it below Call alarm.
37. Right-click on Check Box; go to the Properties box in the resulting object; change its name to "Turn off alarm?"; go to Layout and select Scaled.
38. Go to Initialization; check Initialize at PreRun; set initial value to 0.
39. Go to Fonts; select Arial Black, size 14 for the Object Text; click OK twice.
40 Connect the Call alarm sequence-out ((bottom) pin to the Turn off alarm? sequence-in (top) pin.
    Note: When the user clicks the box, a checkmark will appear and the object will be set up to display: a 1. Otherwise, the object will be set up to display a 0. (The output can be tested with an If/Then/Else object to tell VEE what to do next.)
41. Select Menu Bar => Flow => If/Then/Else; place it the right of the Turn off alarm?.
42. Connect the Turn off alarm? data-output (right: Toggle) pin to the If/Then/Else data-input (A) pin.
43. Edit the expression in the If/Then/Else object (white space) to a==1.
    Note: If terminal A holds a 1, the Then output will fire. Otherwise, the Else output will fire.
44. Select Menu Bar => Flow => Repeat => Break; place it below and to the right of If/Then/Else; connect its top pin to the Then output pin of If/Then/Else. See Figure 17.8.

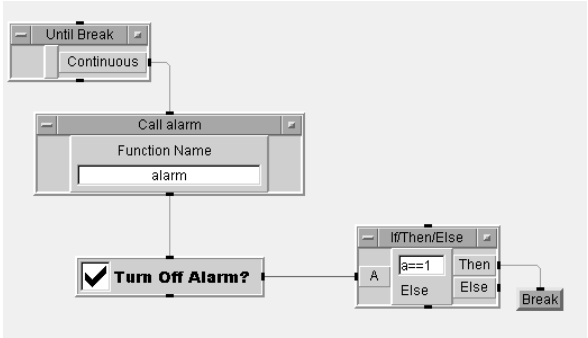


Figure 17.8. Detail view of the UserFunction: warning

45. Select the Turn off alarm? object by clicking on its right side.
46. Open the pop-up edit menu (by clicking the right-hand mouse button on the white area of warning); select Add To Panel.
47 Size the panel view to surround Turn off alarm?
48. Open the warning Properties box, select Show Panel on Execute, and deselect Show Title Bar; click OK. (The Title Bar does not need to be seen by the operator.)
    Note: You should now have three icons at the bottom of your screen: alarm, Main, and warning. Align them along the bottom of your screen if necessary.
49. Go to Main; Select Menu Bar => Device => Call; place it in the top-center of Main; open its object menu, click Select Function, select Type: Local User Functions; Category: <All>; Member: warning; click OK.
50. Minimize the Main window.
51. Run this program; reposition the alarm and warning panels if necessary. See Figure 17.9.

Figure 17.9. The Warning program

52. Save As… LAB17-3.

Lab 17.4 – Exploring a Pre-Designed Digital Filter Program

This lab will show you how to explore the capability of a VEE Pro supplied digital filter program.

Close your VEE program if necessary.

1. Go to the hard-drive location where the program files are stored; open the Program Files folder. (You may use Ctrl F as a means for locating these files.)

   or:

2. Go to the Help menu in VEE; select Open Example… => Applications to reach the Agilent digital filter program.

Exploring the Agilent digital filter program

3. Double-click on the digfilt.vee icon. The program is shown in Figure 17.10.

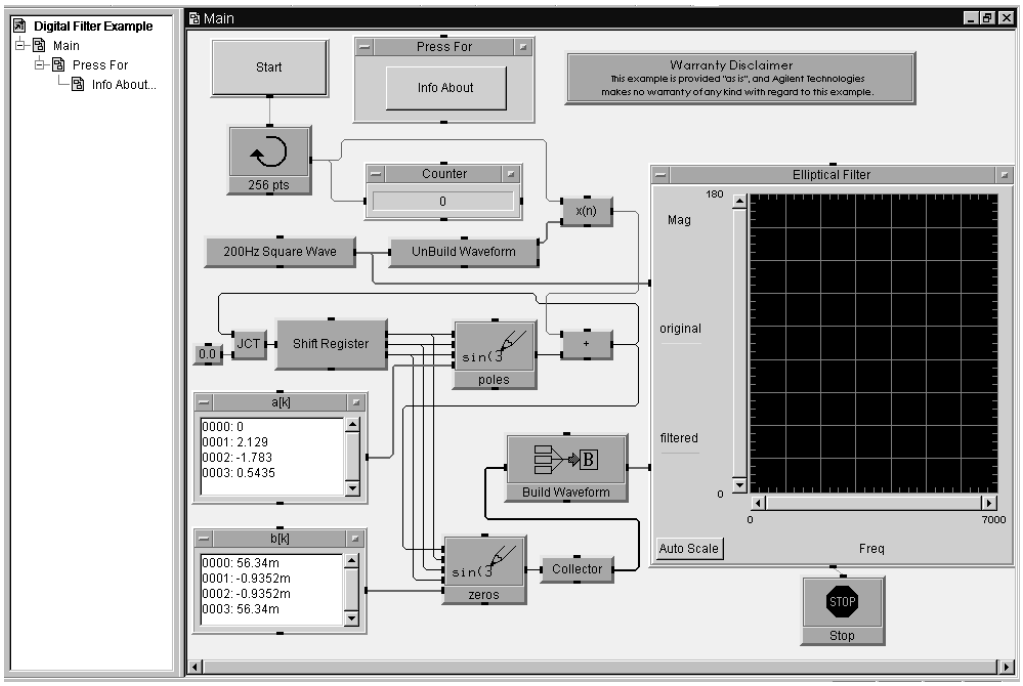


Figure 17.10. The digfilt.vee program

4. Run this program by clicking on its Start button. The display will show the frequency-magnitude distribution of a 200 Hz square wave in green and its filtered output superimposed in yellow.
5. Open the "200Hz Square Wave" object.

6. Run the program with each of the following waveform function settings:
   Sine or cosine
   Tri
   and either
   +Ramp or –Ramp.
7. Observe, for each of the above selected waveforms, the change in frequency and magnitude.
8. Return to the square-wave waveform selection.
9. Apply your knowledge of elliptical filter poles and zeros (see note 2 below), change the a[k] (pole) and b[k] (zero) values; start the program to observe the results on the frequency-domain display.
   Note 1: It may be necessary to select the display Auto Scale.
   Note 2: This step is designed for college students exploring filter design. Therefore, it may be skipped if you have no interest in this material.
10. Close this program without saving it unless you plan to use the results of your computations.
11. Rename the program to describe your chosen filter values if you so desire. Place it in whatever folder you prefer.
    Note 1: If you intend to design several filters, then create a new folder for these new programs.
    Note 2: You may prefer to adapt the fundamental elliptical-filter program to provide other types of filters. See Appendix E, page E-76.

Lab 17.5 – Using MATLAB® to Display the Pre-Designed Digital Filter

This lab will show you how to use MATLAB® to graph the Agilent digital filter output to show directly the harmonic numbers and their companion heights.

Open your VEE program.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Displaying the Agilent digital filter output using MATLAB®

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of Main.
3. Change the following Function Generator settings:
   Function: Square
   Frequency: 250
   Amplitude: 1.5
   Dc Offset: 1
   Num Points: 250
4. Select Menu Bar => Data => Unbuild Data => Waveform; place it to the right of the Function Generator.
5. Select Menu Bar => Display => Waveform (Time); place it below the Function Generator; change the following:
   Title Bar to Unfiltered Waveform
   Mag values to Maximum: 3, Minimum: -1, turn Automatic Scaling Off.
6. Connect the Function Generator output (Func) to the Unbuild Waveform input and to the Unfiltered Waveform (Trace1).

7. Select Menu Bar => Device => MATLAB Script; place it to the right of Unbuild Waveform.
8. Change the title of the MATLAB Script input terminals to “data” (top) and “time” (bottom). Connect these two terminals to the output terminals of Unbuild Waveform.
9. Expand the MATLAB Script object to accept the following code in its expression space:

```
%Filter Waveform
a=[2.129,-1.783,0.5435];
b=[56.34,-0.9352,-0.9352,56.34]*1e-3;
X=filter(b,a,data);
t=[0:length(data)-1]*time/length(data);

%Plot
plot(t,X);
xlabel('time');
ylabel('amplitude');
title('Filtered Waveform');
```

   Adjust the white space to display all of this code.
10. Delete the MATLAB Script output terminal.
11. Select Menu Bar => Flow => Confirm (OK); place it below MATLAB Script; change its title bar to: Done.
12. Connect the Done button to the bottom terminal of MATLAB Script. See Figure 17.11.

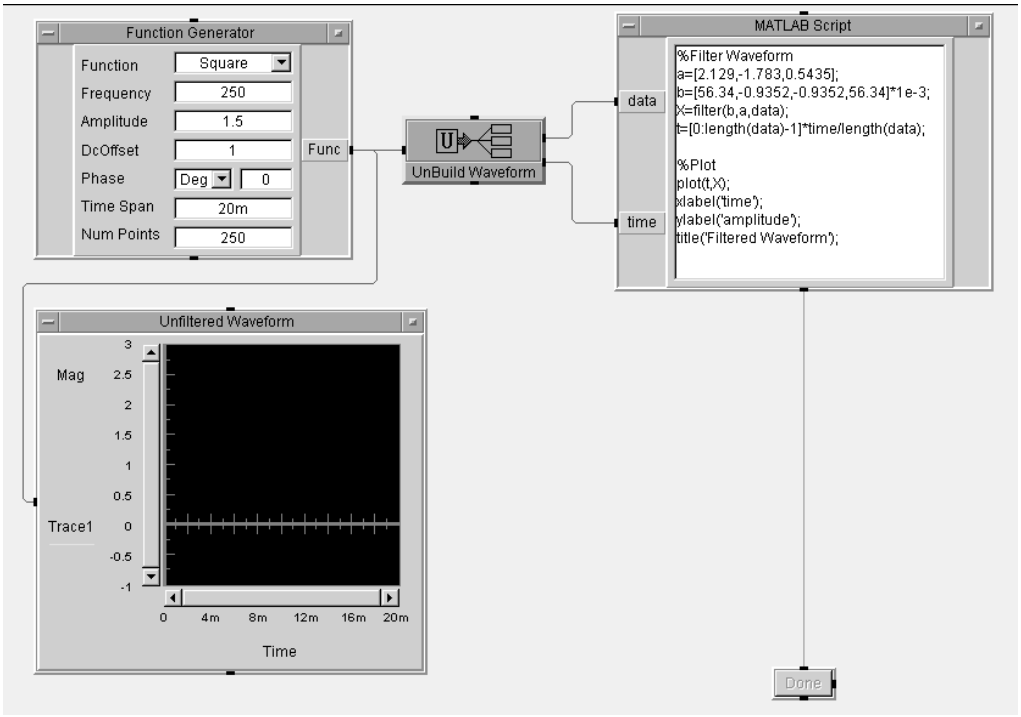


Figure 17.11. LAB 17.5 before running

13. Save this program as LAB 17-5.
14. Run this program. See Figure 17.12 where the MATLAB Figure No. 1 has been sized and placed next to the unfiltered waveform so these two waveforms may be easily compared.
    Note 1: Compare the two waveforms. The filter causes a change in slope of the waveform, a shift in the waveform's vertical location (offset), and a change in its magnitude. In a real-life situation, an operational amplifier would be added by the designer to correct these vertical-value changes.
    Note 2: If the MATLAB® program is installed on your computer, then it is possible to open the MATLAB Command icon at the bottom of your screen. In the space provided, additional programming may be entered. For further information, visit: www.mathworks.com.

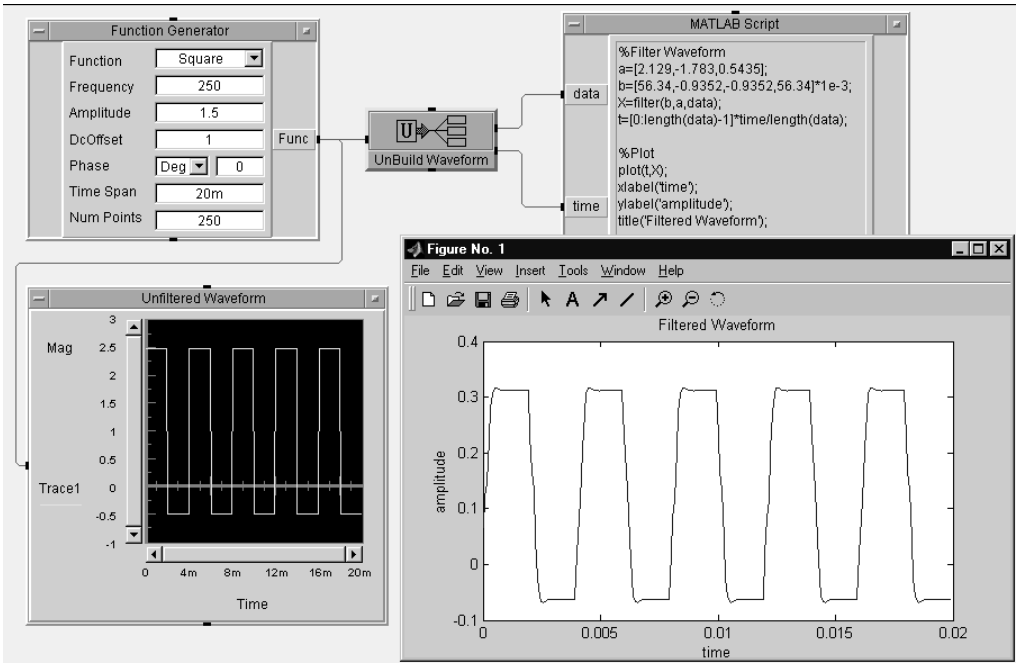


Figure 17.12. LAB 17.5 after running

15. Change the Function Generator setting to DcOnly; run this program again.
16. Compare the filtered and unfiltered waveforms; see Figure 17.13.
    Note: Time is required for the Dc value to reach its final value. In the process, the number of poles and zeros within the filter design will cause an overshoot from the filter's output final value. Again, an operational amplifier could be added to correct only the location of the final value. See Appendix E for additional detail on filter design.

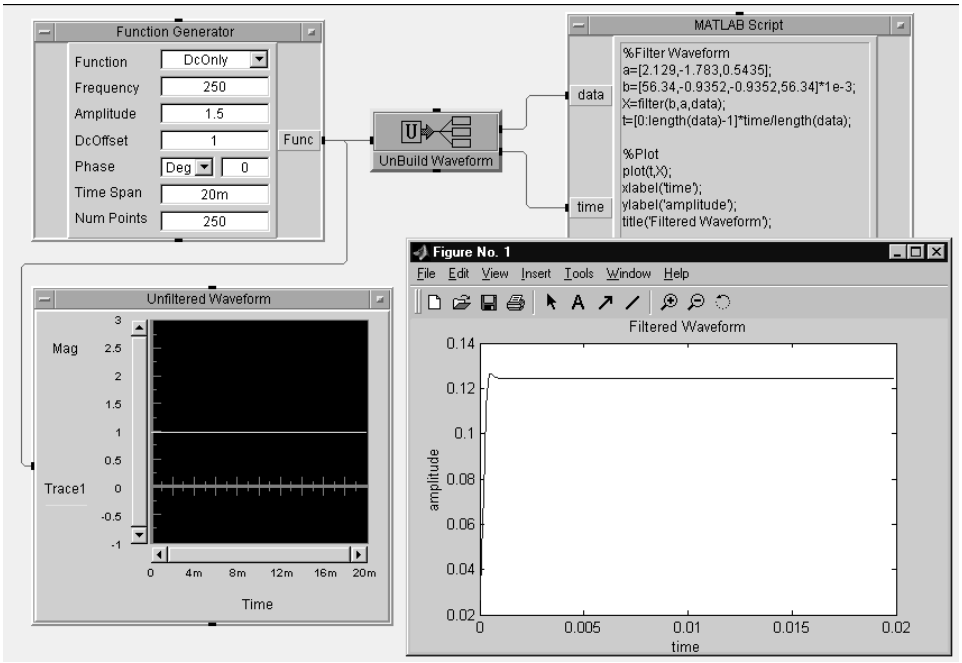


Figure 17.13. A filtered Dc offset

◊ 17. Examine the effects on the filter design for the following Function Generator functions: sine, cosine, triangular (tri), +Ramp, and -Ramp.

18. Compare these waveforms to the square-wave; write your notes in the following table:

Waveform	Change in Amplitude	Change in Offset	Other
Square			
Sine			
Cosine			
Tri(angular)			
-Ramp			
DcOnly			

19. Close this program without saving it.

Lesson 17 Summary

VEE Pro is a very versatile programming approach. New vendor-developed ActiveX controls are becoming available on an almost monthly basis.

In this lesson you learned to apply operator interfaces to a variety of tasks such as using menus, creating status panels, importing bitmaps for panel backgrounds, creating a high-impact warning, and using an ActiveX control. This is the last lesson that will directly assist you in the development of programs.

Lab 17.1 showed you how to create a status panel and use this panel to provide you with status information regarding your in-progress test. This lab can also be devised as a modular program (UserFunction) that could be called upon later.

Lab 17.2 showed you how bitmaps can add impact to your test displays. You learned that bitmaps can be imported for icons, for the Picture Object, or for the panel view backgrounds in a UserObject or a UserFunction.

Lab 17.3 showed you how to create a high impact warning.

Lab 17.4 showed you how to use the Agilent pre-designed filter program.

Lab 17.5 showed you how to use MATLAB® to graph the Agilent digital filter output waveform versus time for a variety of excitations.

If you are using a DAQ card, then there would be five steps to follow when developing a program:

1. Identify and define your objectives.
   Examine the characteristics of the sensors that you will use to monitor your equipment.
2. Expand your objectives to include specific parameters that you plan to measure.
   Select your sensors; identify the VEE Pro icons that will provide their simulation.
3. Identify the VEE Pro Objects that will process your specific parameters.
   Write the program that applies these objects and the hardware simulators.
4. Examine the hardware that you plan to monitor and the related plug-in DAQ cards.
   Select those DAQ cards; assure yourself that there is a slot for them in your computer.
5. Validate your program after you have attached the hardware-access cards.
   Include in your program the ability to prepare Excel™ and Word™ hard copy.

There will be variations to the above steps; these variations will depend upon the program that you want to develop, test, apply, and evaluate.

You are now ready to learn how to improve various aspects of VEE Pro programs.

Lesson 18
Improving VEE Pro Program Productivity

This lesson will guide you in the improvement of the VEE Pro programs that you have already devised. You may, if you wish, devise new programs or bring to class programs that you have prepared on your own. Completion of this lesson should improve the

- rate of program performance, such as test measurement speed, data transfer rate, and program processing speed,
- display of information and controls to users, and
- presentation of spreadsheets and reports to executives.

Acquiring these capabilities will improve your data-acquisition, programming, and presentation skills.

There is no pre-lab involved with this lesson.

Overview

This lesson will focus on the execution speed of your existing program(s). Your program-implementation speed will be improved as you apply the "best" programming techniques.

For the latest information for developing faster programs, go to www.agilent.com on the Internet. The topics listed in Advanced Programming Techniques, Appendix D are:

- Use the Profiler
- Look at line colors
- Add Terminal Constraints
- Use Declared Variables
- Eliminate the Autoscale control input
- Send a complete set of data
- Execute the display only once
- Turn debugging features off

Lab 18.1 – Improving One or Two VEE Pro Programs
This lab will assist you in learning how to improve an existing VEE Pro program and provide insight into improving program design.

Lab 18.2 – Using a Single UserFunction to Select Many Picture Objects
This lab will show you how to access many picture files using a single UserFunction.

Lab 18.3 – General Purpose Harmonics Generation Program
This lab will show you how to devise a general-purpose Harmonics Generation program.

This is an open-ended lesson. Ideas should be exchanged and discussed among your peers.

Improving VEE Pro Programs

Routing data through object operation

The order of object operation shown in Figure 18.1 is as follows:

1a. If the object's Sequence input is connected, then the object will wait for both the data input and sequence input to receive data (or a "ping"). It will then go to step 2 below. (Because the object waits for data input to be received, the Sequence input is used as a "holding" input; it is not a trigger input.)

1b. The object accepts data on its data input pin(s). It will then wait until there are data on its all data input pins. It will then go to step 2.

2. The object performs its designated function. Any error that occurs can be caught by an optional error output pin.

3. The object sends data out on its data output pin(s).

4. The object waits for a "receipt acknowledged" signal. This signal is sent when every "down-thread" from this object completes its operation. (There may be instances when a "down-thread" object cannot complete its operation. For instance, when an object has a data input that is not yet satisfied. In these cases, the "receipt acknowledge" is sent when that thread has done all it can do.)

5. The object sends a signal out its Sequence Out pin.

Then the object deactivates.

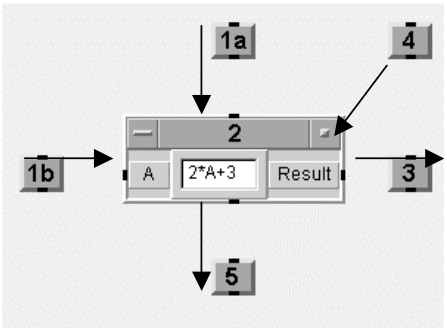


Figure 18.1. Object operation

Refining VEE Pro Programs with math arrays, icons, and function calls

The following topics will assist you in the refinement of your programs. Most of these topics will apply, depending upon the complexity of your programs:

- Perform math on arrays whenever possible. See Figure 18.2.

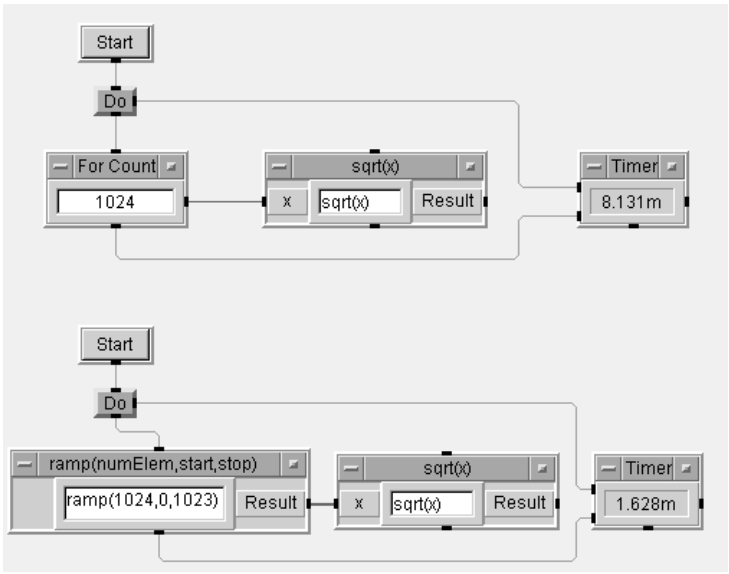


Figure 18.2. Math array examples

- Make objects into icons whenever possible. See Figure 18.3.

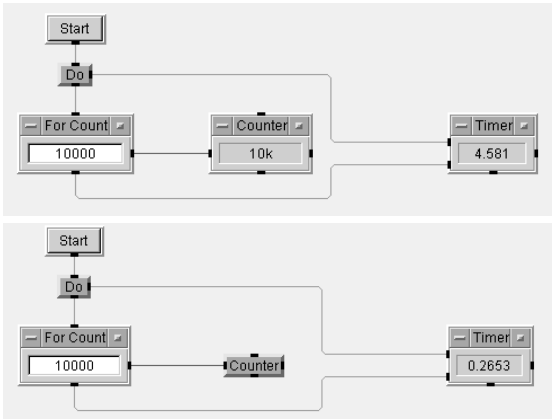


Figure 18.3. Improving by using icons

- Reduce the number of objects in your programs. See Figures 18.4 and 18.5. (Note: this reduction can be carried to the extreme; readability and maintainability should also be considered as well as speed of execution, particularly for instrument-control programs.)

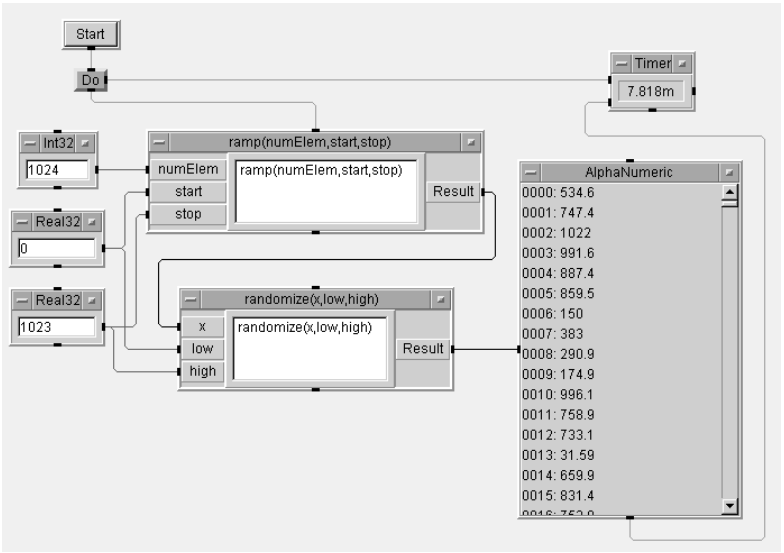


Figure 18.4. Function calls with many objects

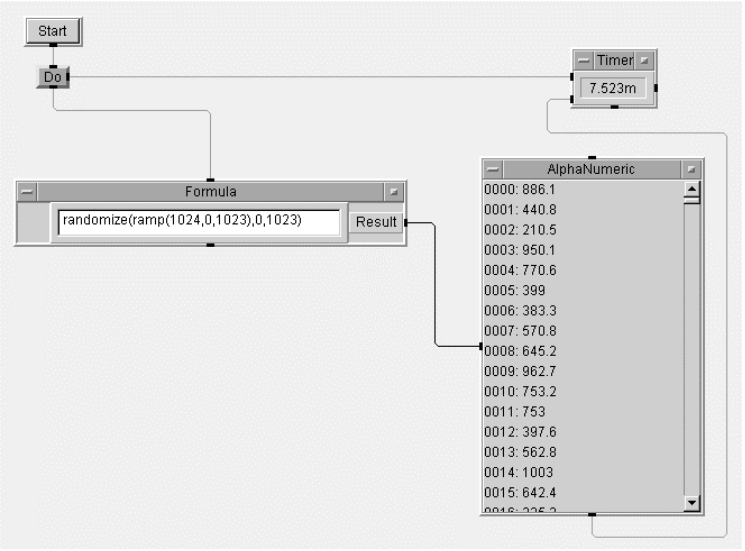


Figure 18.5. Function calls with fewer objects

Applying techniques and guidelines that will assist in improving programs

- Link compiled functions in other languages to your VEE Pro programs. Faster execution speed will occur; you may develop data filters in other languages and integrate them into your programs and you can secure proprietary routines. Only use a compiled function when the capability or performance you need is not available by using a VEE Pro UserFunction, or an Execute program escape to the operating system, or ActiveX Automation calls to another program. See Appendix D.
- Run your program from the Panel View instead of from the Detail View.
- Use global variables rather than pass values.
- Declare all your global variables. (This allows you to use local variables.)
- Collect data for graphical displays; plot the entire array at once; if your X values are evenly spaced, then use an XY trace display instead of an X versus Y display.
- Use one If/Then/Else object with multiple conditions rather than multiple If/Then/Else objects.
- Set graphical displays to be as plain as possible. The fastest update times are:
  Grid Type => None with nothing checked in the Properties dialog box.
  Use only AutoScale control pins where necessary.
  Turn off the Automatic AutoScaling (in the Scales folder) if not needed.
- Read data from a file using the ARRAY ID TO END: (*) transaction.
  Do not perform READ transactions on one element at a time.
  Do not use the EOF pin.
- Depending upon the program that you have developed, avoid using separate Call objects; instead use the Sequencer to control the flow of execution of several UserFunctions.
- When using the Sequencer, only enable logging for transactions where the Log record is required.
- On Strip Charts and Logging AlphaNumeric displays, set the Buffer Size in Properties to the smallest number possible for your application.
  Note: To monitor the time required to run a program, run your program using Step Into to determine at what object the program starts and at what object the program ends. Select Menu Bar => Device => Timer and connect your Timer to these two object outputs. You may want to change the Timer Title Bar to include the word: (seconds).
- Use the Triadic Operator (condition ? expression1 : expression2) instead of the If/Then/Else objects with Gates and a Junction. See Figure 18.6.
  The Triadic Operator is a special function that can be typed into a Formula box. It is based on a "C" program expression. It combines the conditional expression and the result of an "If/Then" expression.

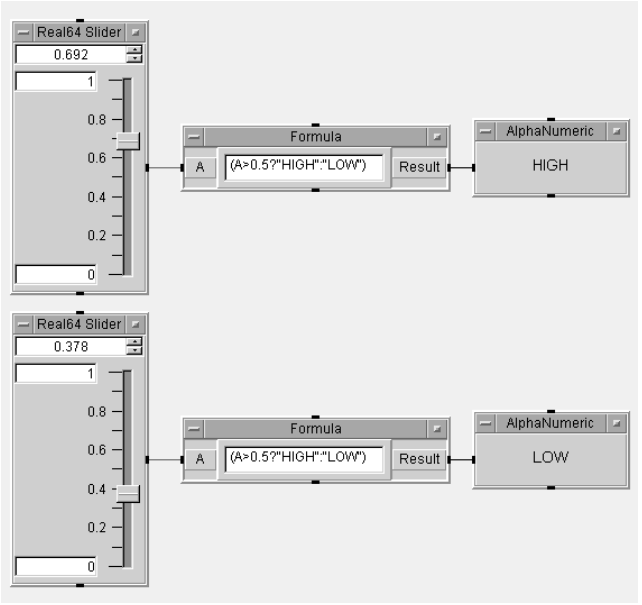


Figure 18.6. Formula object – the Triadic Operator

- When using bitmaps, set them to Actual or Centered rather than Scaled. (Scaled will take a few moments longer.)
- On indicators such as the Fill Bar or Thermometer, turn off Show Digital Display.
- On Color Alarms, if you are switching between colors rapidly, turn on Show 3D Border.

For additional, more detailed, information regarding program-performance improvements, see Keys to Faster Programs (Index: Programs, Speeding Up) within your Advanced Programming Techniques book.

Applying propagation rules to improve "auto execute" and "wait for input"
"Auto Execute" and "Wait for Input" are properties of objects than can affect the propagation of the VEE Pro thread to which these objects are connected. When an object is set for "Auto Execute", the object activates after a user input occurs as follows:

- the user clicks and drags the slider to an integer value and then releases the mouse button,
- that integer value is sent to the slider's data output, and
- execution begins for that thread without having to click on a "Start" or "Run" button.
  See Figure 18.7.

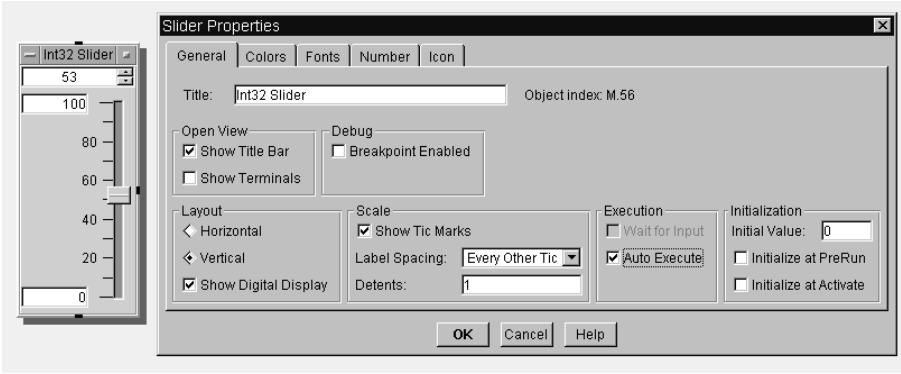


Figure 18.7. Propagation: Auto execute and wait for input

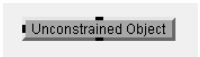
Propagation Rules; the order in which objects occur:

a. If there are Start objects in a program, they operate first.
b. Unconstrained objects operate after the Start objects. (Unconstrained objects are any objects that do not have data inputs; their sequence input is not connected to anything.)
c. Propagation occurs from left to right.
d. All of an object's data inputs must receive a container of data before that object can operate.
e. If an object's sequence input is connected, then that object will not operate until its input is "pinged" even if the object has already accepted data on all of its data inputs.

See Figure 18.8.

"Start" objects operate first.

Unconstrained objects operate after "Start" objects. (For example, a Formula Object containing the expression "random()" will execute only once at the start.)



Propagation occurs from left to right.

All Data Inputs must accept a data container before that object can operate.

If the Sequence Input is connected, then that Input must be pinged before that object can operate.

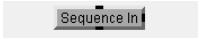


Figure 18.8. Propagation rules

Using stopping loops to improve programs
See Figure 18.9.

a. VEE Pro provides two objects that terminate or interrupt a repeater object.
b. The Next object ends the current iteration, allowing the repeating object to proceed to the next iteration.
c. The Break object ends the current and all future iterations for that iterator object. The iterator activates its Sequence Out pin and stops operating.

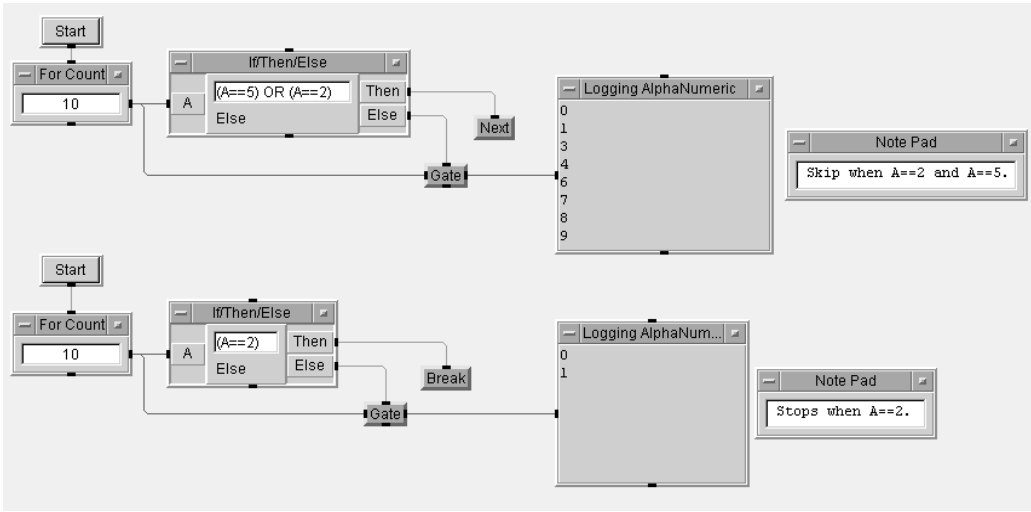


Figure 18.9. "Loop-Stopping" objects

Positioning Read/Write (sequential) Pointers to improve programs

In the example shown in Figure 18.10 below, the read and write pointers have been previously positioned by programming. Next, a ten-element array of numbers 1 through 10 is passed on to a ToFile object. Because Clear File at PreRun & Open is checked, the data in myFile are cleared and both the write and read pointers are positioned at the beginning of the file. The ToFile object writes the ten-element array container to myFile. After the WRITE transaction, the write pointer points to the next available storage space in the file while the read pointer stays at the beginning of the file. The FromFile object contains a transaction that reads a container's worth of data from the place where the read pointer is located. After the READ transaction, the read pointer points to the same place as the write pointer.

The write pointer can only be moved to the beginning of the file by checking Clear File at PreRun & Open or with an EXECUTE CLEAR transaction. In both cases, all data are cleared from the file. The read pointer can be moved to the beginning of the file without affecting the data by an EXECUTE REWIND transaction in a FromFile object. To read a specific piece of data from a file, you must READ all the data, in sequence, prior to the desired data. You then READ the desired data. See Figure 18.10.

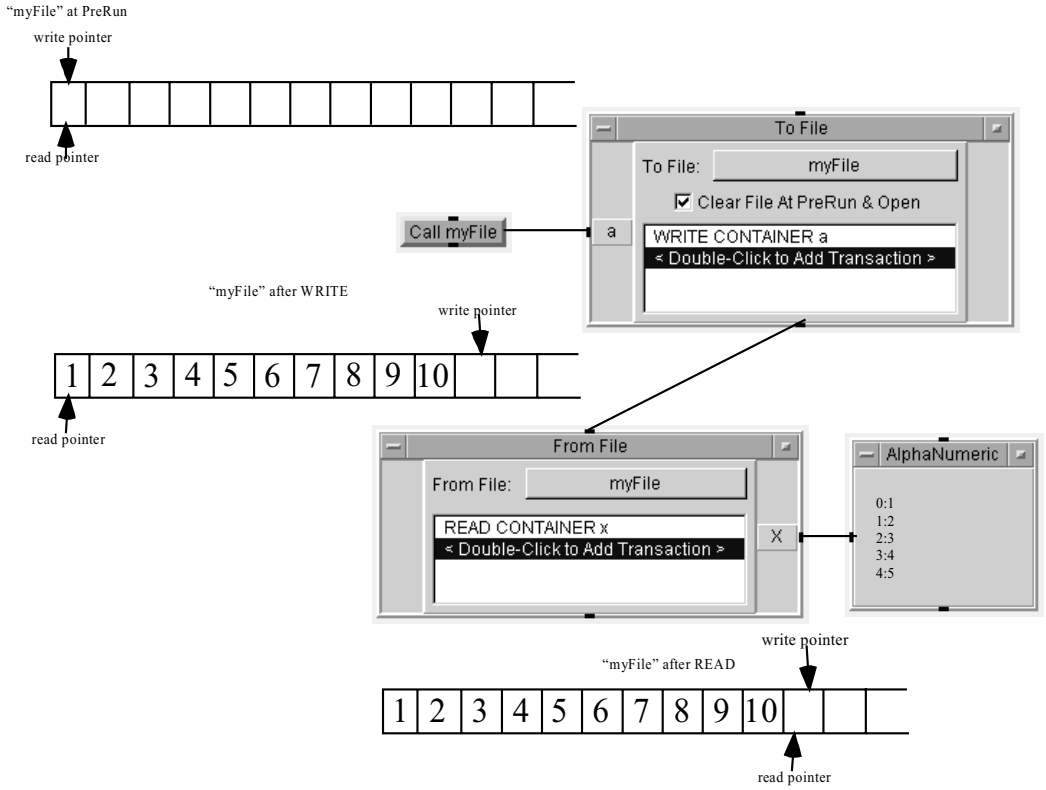


Figure 18.10. Read/Write pointers example

Note: During the "To File" execution, the READ pointer pointing to 1 did not move; during the "From File execution, only the READ pointer moved.

Applying miscellaneous techniques and procedures to improve programs

Note: To determine the Order of Test Specifications, see Figure 18.11.

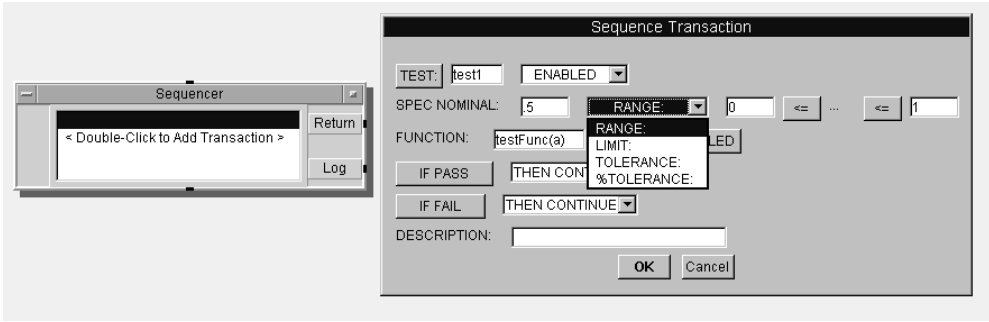


Figure 18.11. Test specifications

Each sequencer transaction can be tested in one of four ways to determine whether the test passed or failed:

Range:	compares the result to a lower and upper limit
Limit:	compares the result to one number
Tolerance:	compares the result to the nominal value ± a tolerance
% Tolerance:	compares the result to the nominal value ± a percent of that nominal value

The TEST: test1 must be a unique name.
The SPEC NOMINAL is used for logging purposes.

The Note Pad

Note Pad objects are used to write development notes to yourself in the Work Area. For most applications, they are preferable to the "description" box because the Note Pad always appears when the program is run.

Alternate to Note Pad Use

The space for the Note Pad on a program page can be avoided. As an alternate, the information may be placed in the Description box. The comments can be entered in that box; the icon to which it refers can have its Title and/or Background color changed to indicate to the user that there is important information in the Description box.

SCPI: the Standardized Programming Language

Hewlett-Packard's Test and Measurement Systems Language (TMSL) was accepted by industry and renamed the Standard Commands for Programmable Instrumentation (SCPI). SCPI is now managed by a consortium of instrument manufacturers. At the present time, there are more than one thousand instrument products that use SCPI.

The SCPI commands describe the signal that you are trying to measure, rather than the instrument that you are using to measure it. This feature, known as "horizontal compatibility", means that the same SCPI commands apply to many different types of instruments. See Figure 18.12.

OUTPUT @ Dmm; ”*RST”	! Reset all the
OUTPUT @ Scope; “*RST”	! test system
OUTPUT @ Counter; “*RST”	! instruments.
OUTPUT @ Scope; “MEASURE:VOLTAGE:RISETIME?”:	! Use oscilloscope to
ENTER @ Scope; Risetime	! measure waveform risetime.
OUTPUT @ Dmm; “MEASURE:VOLTAGE:DC?”	! Use DMM to accurately
ENTER @ Dmm; Dc_level	! measure final signal level.
OUTPUT @ Counter; “MEASURE:FREQUENCY?”	! Use counter to measure
ENTER @ Counter; Frequency	! frequency of another signal.

Figure 18.12. The horizontal compatibilities of SCPI

SCPI is designed to grow as instrument capabilities grow. The “vertical compatibility” of SCPI allows you to buy, for example, a multimeter that has more features than your current multimeter. The functions of your existing multimeter will be programmed exactly like the functions of the newly purchased multimeter.

VISA I/O Library

A common I/O library allows you to inter-operate system components. The VXI*plug&play* Alliance software allows you to communicate over a physical connection. It is the foundation on which other standards are built. The library is independent of instruments, interfaces, operating systems, languages, and networking mechanisms.

The development of Virtual Instrument Software Architecture (VISA) provides a single foundation for multi-vendor instrumentation software. VISA provides access to new capabilities as technology changes; VISA maintains a migration path for existing systems. VISA offers all of these as a single, easy-to-use set of I/O control functions very similar to existing I/O libraries, such as the Agilent SICL.

VISA provides a set of core functions, such as Location and Life Cycle Control (open, close), events (enable, disable), and Message-Based Control (write, read, print). Each of these functions applies to all instruments. Thus, you can use VISA’s open (viOpen) function call for any device in your test system.

VXI technology improvements

The VXIbus and VXI*plug&play* standards provide a well-defined hardware and software environment. Modules from many different manufacturers can work together. VEE Pro can access and load any instrument driver written to the VXI*plug&play* standard. The instrument driver then provides a procedural interface to the instrument for programmatic control. VEE Pro uses a graphical user interface to make it easy for you to develop the code necessary for controlling your instruments. These drivers are available for GPIB and other non-VXI instruments.

VXI provides scalable, configurable solutions that adapt to your unique measurement situation, allowing you to tradeoff cost and performance. VXI is easily integrated into existing or new rack-and-stack systems. Complete solutions can reduce your cost of test and documentation.

VXI references:

- SCPI Beginners' Guide
- VXIbus System Specification
- VMEbus Specifications
- VXI*plug&play* Specifications
- Agilent's VXI*plug&play* Instrument Driver Availability on the World Wide Web
- Agilent's Test System and VXI Components Catalog on the World Wide Web

Directory of VXIbus and related test system products

- Agilent-IB Interfaces
- VXI Accessories
- VXI Amplifiers
- VXI Analog Sources
- VXI Application-Focused Products
- VXI Command Modules
- VXI Counters/Time Interval Analyzer
- VXI Data Storage
- VXI Development Tools
- VXI Digital I/O Modules
- VXI Digital Multimeters/Power Meters
- VXI Digitizers/Signal Processor
- VXI Direct Access Interfaces
- VXI Embedded Controllers
- VXI General Purpose Switches

Directory of VXIbus and related test system products (*continued*)

- VXI Instrument Control Software
- VXI Interconnect & Wiring
- VXI Mainframes
- VXI Matrix Switches
- VXI Microwave Switches
- VXI Multiplexer Switches
- VXI Oscilloscopes
- VXI RF Multiplexer Switches
- VXI Scanning A/D Converters/Closed Loop Control
- VXI SCSI System Disks
- VXI Special Purpose Modules
- (All of these products are available on the Agilent Web site)

VXI Interface Configurations

There are four typical VXI interface connections. The diagrams that describe these connections are shown in Figure 18.13.

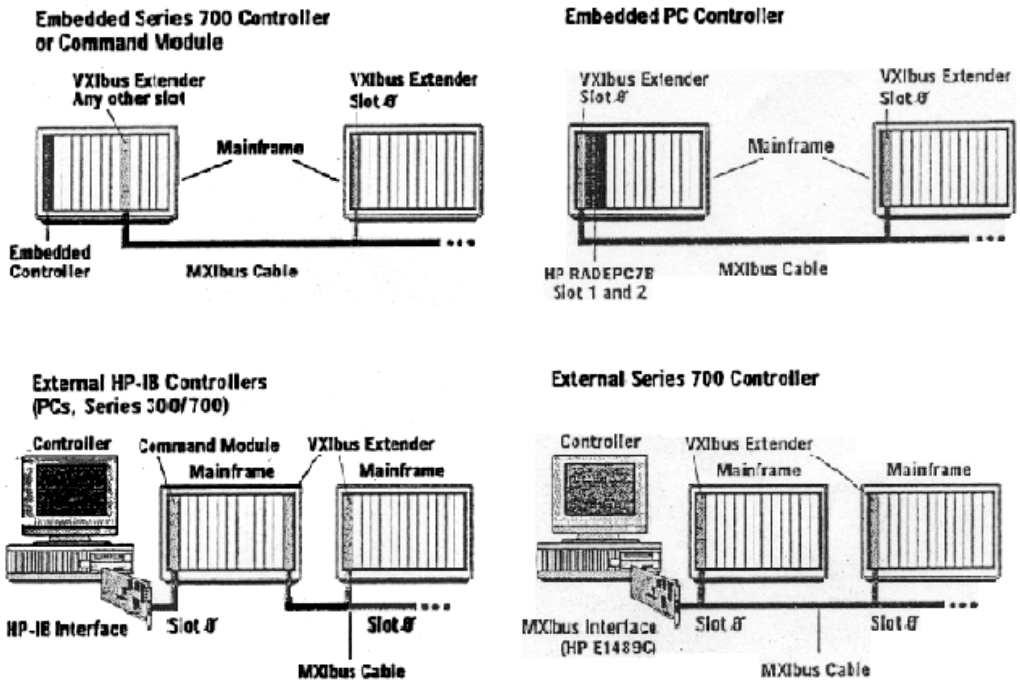


Figure 18.13. Four typical VXI interface connections

Lab 18.1 – Improving One or Two VEE Pro Programs

This lab will assist you in learning how to improve an existing VEE Pro program. It should also provide clues as to how to improve your program designs as you progress in their development. Experience remains the best "teacher" for all of us.

1. Choose one or two programs from either previous labs of from a program that you have written on your own.
2. Determine the
    a. rate of your program performance,
    b. display of information and controls to users other than yourself, and
    c. presentation of its spreadsheets and reports, if any, to executives.

3. Obtain the approval of your instructor regarding which programs that you plan to improve and, approximately, how you plan to accomplish it.
4. Revise those programs that were selected and approved.
5. Determine the new

    a. rate of program performance,
    b. display of information and controls to users other than yourself, and
    c. presentation of its spreadsheets and reports, if any, to executives.

6. Test your program performance; record the

    a. speed of obtaining test measurements,
    b. time required for your program to process the test data, and
    c. rate at which data are transferred to a computer or server.

7. Demonstrate your newly improved program.

Lab 18.2 – Using a Single UserFunction to Select Many Picture Objects

This lab will show you how to access many picture files using a single UserFunction. See Lab 16.4 where three UserFunctions were used to access three pictures. This will allow you to create more general programs.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Using one UserFunction to access many picture files

2. Open LAB 16-4 and immediately Save As… LAB 18-2.
3. Cut UserFunctions die2 and die3.
4. Rename die1 to showPicture; add an input terminal to this renamed UserFunction.
5. Add an input terminal to the Picture object; connect the two added terminals.
6. Expand the radio button “Make a Selection” to include four more choices via Edit Enum:
    die4
    die5
    die6
    arrow
7. Delete Call Function object; select Menu Bar => Device => Formula; change its expression to A+”.gif” and connect the top terminal of Make A Selection (Enum) to the Formula input.
8. Select Menu Bar => Device => Call; change its expression to showPicture.
9. Connect the Formula output terminal “Result” to the Call Function data input “A”.
10. Save this program and run this program. See Figure 18.14.

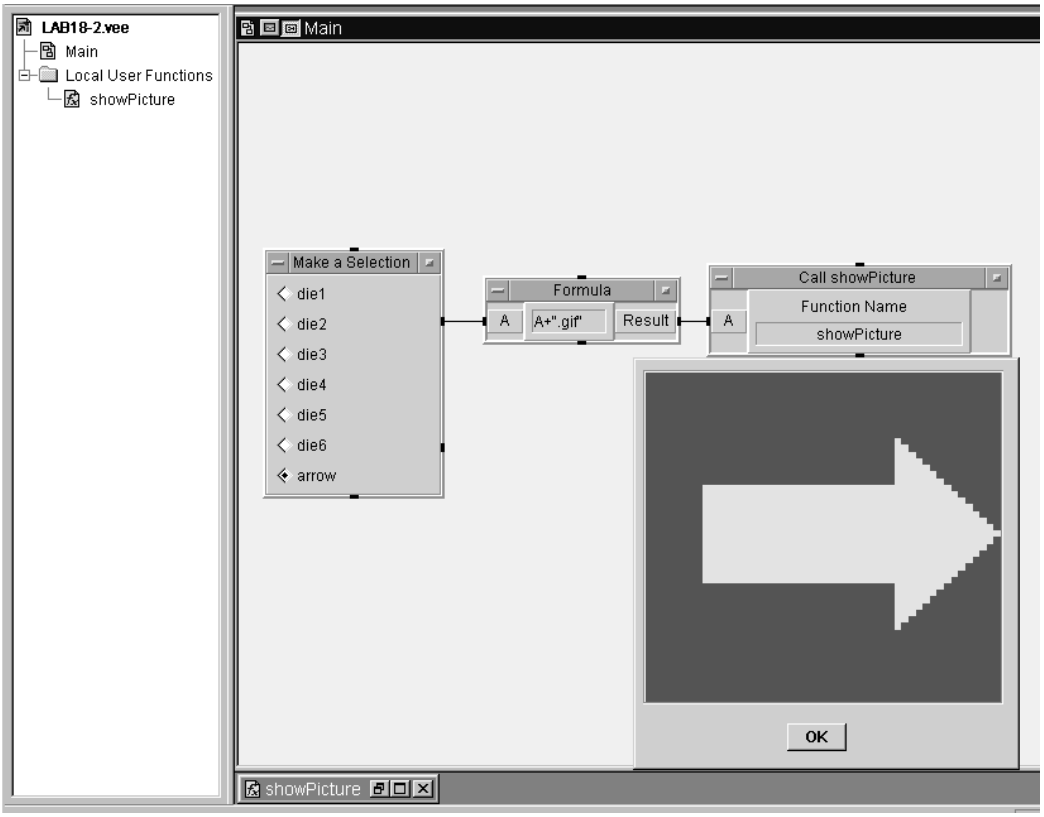


Figure 18.14. LAB 18.2 after running

Note: Copying and cloning UserFunctions is not efficient programming practice. However, using one UserFunction usually simplifies the design and allows re-use of one object.

11. Run this program several times.
12. Close this program; you may have to save it again.

Lab 18.3 – General Purpose Harmonics Generation Program

This lab will show you how to devise a general-purpose re-usable Harmonics Generation program. Lab 13.2 showed you how to write a program, one generator at a time, to develop a square wave containing thirteen function generators. This lab, although more complicated and slower in operation, is more flexible and allows for the study of simulated square waves containing a variety of harmonics, the number chosen by the user. This alternative was devised by Susan Wolber of Agilent for this book.

1. Clear your Work Area, deselect the Program Explorer, and maximize Main.

Devising a flexible square-wave harmonic-generation program

2. Select Menu Bar => Device => Virtual Source => Function Generator; place it in the upper-left corner of your Work Area change its Title Bar to Fundamental.
3. Change the "Fundamental" Generator settings to Function: Sine; Frequency: 100.
4. Select Menu Bar => Data => Variable => Set Variable; place it to the right of Function Generator; change its expression to g_sum.
5. Connect the Function Generator output (Func) to the Set g_sum input terminal.
6. Select Menu Bar => Data => Dialog Box => Int32 Input; place it below the Function Generator.
7. Change the title of Int32 Input to: Enter Harmonic.
8. Change its Prompt/Label expression to: Enter Highest Odd Harmonic Desired.
9. Change its Default Value to: 13.
10. Change its Value Constraint to: (value MOD 2==1) AND value>=3.
11. Change its Error Message to: You must enter an odd number greater than 1.
12. Connect the bottom (sequence out) terminal of Fundamental to the top (sequence in) terminal of Enter Harmonic.
13. Select Menu Bar => Flow => For Range; place it below and to the left of Enter Harmonics;.
14. Enter the following:
    From: 3
    Thru: 63
    Step: 2
    Add an input terminal labeled: Thru.
15. Select Menu Bar => Data => Variable => Get Variable; place it below For Range; change its expression to g_sum.
16. Select Menu Bar => Device => Formula; place it to the right of For Range; change its name to Frequency; enter in the expression space: A*100.
17. Select Menu Bar => Device => Formula; place it below Frequency; change its name to Amplitude; change its expression:
    1.0 / A
    // 1.0 to ensure real
    // division not integer
18. Connect Enter Harmonic output terminal to the For Range input terminal.
19. Connect For Range output terminal to both Frequency and Amplitude input (A) terminal.
20. Connect For Range bottom terminal to Get g_sum top terminal.
21. Select Menu Bar => Device => Virtual Source => Function Generator; change its name to Function Generator – Harmonics; change Function to Sine; Frequency to 4300; and Amplitude to 23.26m; add two input terminals: Frequency and Amplitude.
22. Connect the Frequency output terminal (Result) to the Function Generator – Harmonics Frequency terminal.
23. Connect the Amplitude output terminal (Result) to the Function Generator – Harmonics Amplitude terminal.
24. Select Menu Bar => Data => Variable => Declare Variable; change it name to g_sum, its Scope to Global, and Type to Waveform.
25. Select Menu Bar => Device => Formula; place it below Declare g_sum; change its expression to g_sum = g_sum + A; connect the output (Func) of Function Generator – Harmonics to the input (A) of Formula.

26. Select Menu Bar => Display => Waveform (Time); place it below Formula; connect its input (Trace1) to the Get g_sum output (Data); change its Time axis value to 10m and turn Automatic Scaling off; shrink Waveform (Time) to approximately one-half its vertical size.
27. Select Menu Bar => Display => Spectrum (Freq) => Magnitude Spectrum; place it below Waveform (Time); connect its input (Trace1) to the Get g_sum output (Data); shrink Magnitude Spectrum to approximately one-half its vertical size.
28. Select Menu Bar => Display => Note Pad; place it between Enter Harmonic and Waveform (Time); enter in its white space:

    Your odd number
    must be
    less than 65.

    See Figure 18.15.

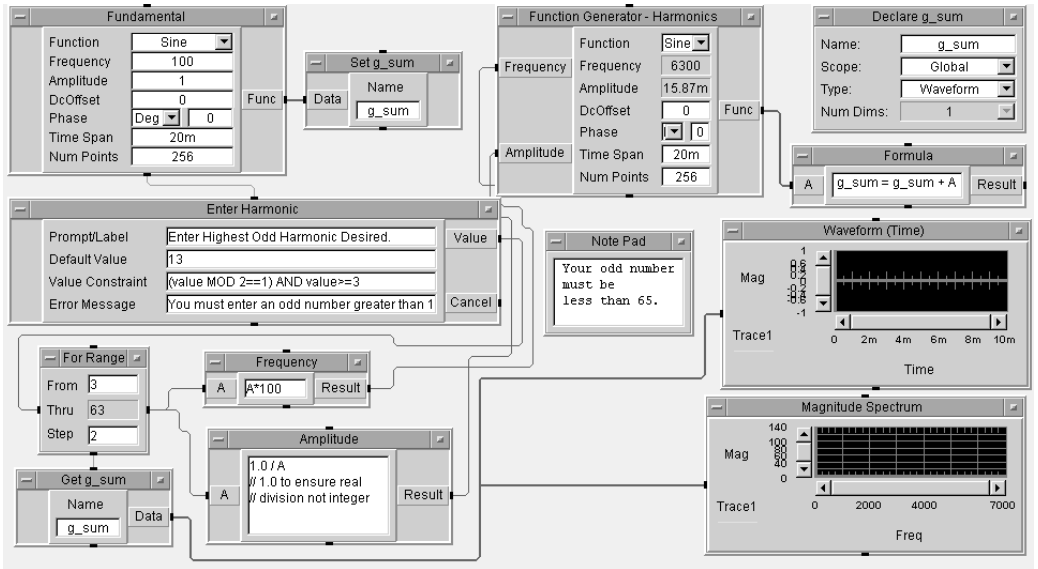


Figure 18.15. LAB 18.3 before running

29. Hold down your Ctrl key and click on Note Pad, Waveform (Time), and Magnitude Spectrum to highlight them; *right*-click anywhere on the Main white space; select Add to Panel. Expand the displays and move the Note Pad and place it under Waveform (Time). See Figure 18.16.

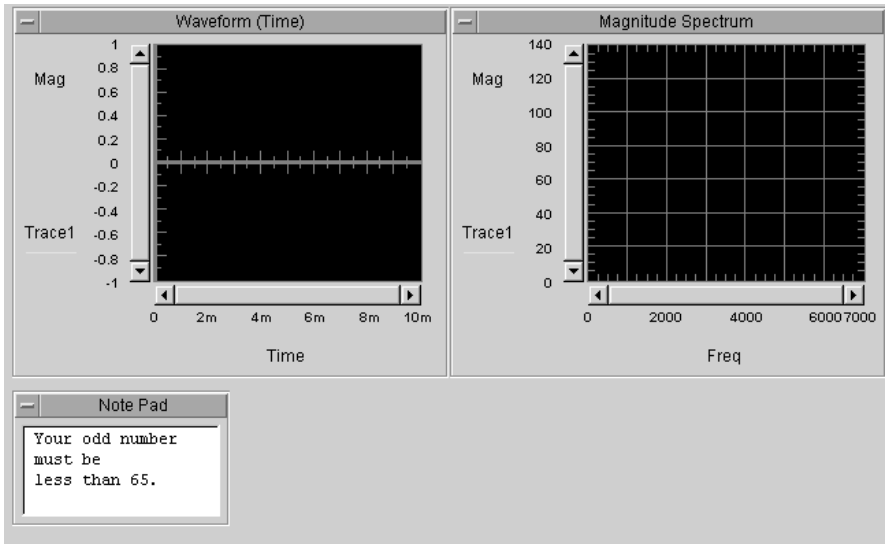


Figure 18.16. LAB 18.3 panel before running

30. Run your program; move your Int32 object so it is beside the Note Pad; select a value for the harmonic range. The Int32 Input Dialog Box will appear. See Figure 18.17.
    Note: We have selected the value: 63.

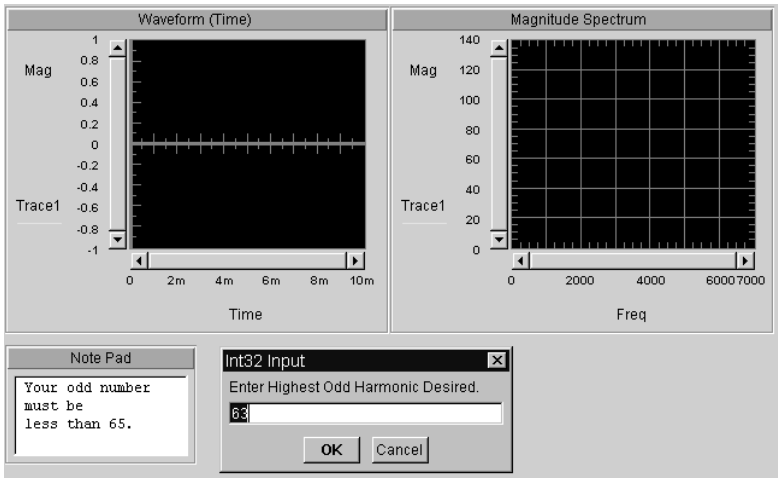


Figure 18.17 – LAB 18.3 panel dialog box before running

31. Click OK; see Figure 18.18 for the display of your results.

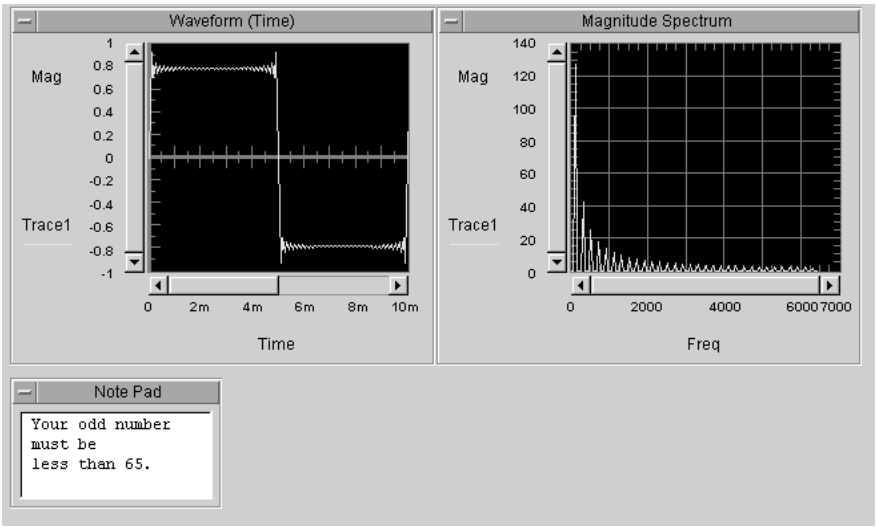


Figure 18.18. LAB 18.3 panel display after running

32. Save and close your program.

◊ 33. Verify the length of time required to run this flexible program. Compare it with the time required to run LAB 18.3. Use the technique described on page 18-5, bottom.

Lesson 18 and Text Summary

VEE Pro programs can range from the very simple to very complicated. As they increase in complexity, at least three factors may become significant:

- the rate of program performance,
- the display of information and controls to users, and
- the presentation of spreadsheets and reports to executives.

This lesson introduced you to ways of improving upon each of these factors.

There are three basic components to test-program performance:

- the speed of obtaining test measurements,
- the rate at which data are transferred to a computer, and
- the speed at which your program processes that data.

This lesson concentrated on the execution speed of existing programs. Also, eleven techniques and guidelines that you can explore on your own were presented on page 18-5.

One user's experience related the following:

Question: Why is VEE Pro more useful than straight C or C++ programming?

Answer: If you are after a measure of your individual productivity, then you need to compare time to produce the "code" to implement a specific function. Try a time comparison: make a VEE Pro function containing from ten to twenty objects. Then, translate it to C or C++. The experience of others in the field is: An hour of VEE programming is about the same as four hours of C for just the coding!

Organizing Large Programs

Below are some tips for organizing large VEE Pro programs:

- Devise UserFunctions rather than UserObjects unless you are using the UserObject only once.
- Design UserFunctions that require no more than 70% of your available screen. Thus, UserFunctions should contain twenty or less objects.
- Design programs whose wiring flows simply and contains few intersecting lines.
- Devise and import libraries with common functionalities.
- Use descriptive names for Object terminals, objects, UserObjects, and Functions. Otherwise, use note pads where an icon or object needs explanation.
- Use colors to assist in distinguishing program segments and information flow.

In summary, keep your programs, and their flow, simple; the user may not be you!

Working with Very Large Amounts of Data

Below are some tips for collecting and processing large amounts of data:

- Expand your PC RAM as much as possible.
- Avoid passing copies of data between UserFunctions and UserObjects; use declared variables instead.
- Revise and re-organize data at the earliest possible point of entry into your program.
- De-bug your program using small amounts of equivalent data.
- Expand the availability of memory by using Delete Variable or Delete Library objects.
- Examine sample amounts of data (such as every 100^{th} point) by writing a custom data viewer and working with these data samples – one at a time.
- Create run-time versions for final executions of programs.

In summary, keep your data collection and flow both simple and modular.

Productivity improvement may not always be your primary design goal. As you plan the design of a program, you may prefer to add flexibility so you can use parts of it another time. Also, you may want to design your program in a modular fashion so you can offer pieces of it to other designers working with you. Such a design approach may not always result in an optimum program. However, the savings in design time by you and by others may be sufficient for you to tolerate a less than optimum design.

We wish you well as you continue to learn and experiment with VEE Pro. It is a programming technique worth the effort required of you. Working initially with virtual instruments and controls provides you with the opportunity to try new concepts that could simplify and enhance data taking, analysis, and presentation. The improvements that you discover will be of great value, both to you and the organization for which you work.

We, the authors, extend our best wishes to you. We hope that this new knowledge and set of skills will lead you to rewarding progress in the achievement of your life-time-learning goals.

Appendix A
Menus, Buttons, the Mouse, and Data Flow

Introduction

This appendix explains instructions on how to use the mouse. The explanation is followed by a description of how data flows into, within, and out of an Object. This appendix concludes with the Object Order of Operation sequence.

Menus and Buttons

The menus for VEE Pro are the top line of words; this line is known as the Menu Bar. Pull down a Menu Bar that includes an arrow to the right of the listing. It indicates a submenu.

Three dots … after a menu word indicate that one or more dialog boxes will follow. (File => Open … is an example.)

The second line of icons (also known as buttons) is known as the Tool Bar. Their contents are given below:

Menu Bar: Menus and Submenus

File	
New	Ctrl + N
Open…	Ctrl + O
Merge…	
Merge Library…	
Save	Ctrl + S
Save As…	Ctrl + W
Save Objects …	
Create Secured RunTime Version…	
Save Documentation …	
Default Preferences	
Send Program by Email	
Print Screen …	

File (*continued*)	
Print Program…	
Print Objects…	
Print Setup…	
Programs (previously opened):	
1	
2	
3	
etc	
Exit	Ctrl + E
Edit	
Cut	Ctrl + X
Copy	Ctrl + C

File (*continued*)
- Paste Ctrl + V
- Select All Ctrl + A
- Delete Lines Shift + Ctrl + LB
- Clean Up Lines
- Find… Ctrl + F
- Find Results
- Add To Panel
- Align
  - Middles
  - Tops
  - Bottoms
  - Centers
  - Lefts
  - Rights
  - Space Equally
  - Undo Last Arrangement
- Create UserObject…
- Create UserFunction…
- Edit UserFunction

View
- Variables…
- Last Error
- Main Ctrl + M
- Execution Window
- Program Explorer (toggle)
- Call Stack
- Profiler
- Toolbar (toggle)
- Status Bar (toggle)

Debug
- Run/Resume Ctrl + G
- Pause Ctrl + P
- Stop
- Step Into Ctrl + T
- Step Over
- Step Out
- Toggle Breakpoint

Debug (*continued*)
- Clear All Breakpoints
- Activate Breakpoints (toggle)
- Show Data Flow
- Show Execution Flow
- Line Probe Shift + LB
- Object Probe Shift + LB

Flow
- Start
- If/Then/Else
- Conditional
  - If A == B
  - If A ~= B
  - If A != B
  - If A < B
  - If A > B
  - If A <= B
  - If A >= B
- Repeat
  - For Count
  - For Range
  - For Log Range
  - Until Break
  - On Cycle
  - Next
  - Break
- Junction
- Do
- Gate
- Sample & Hold
- Confirm (OK)
- Delay
- Exit Thread
- Exit UserObject
- Stop
- Raise Error

- Device
  - Formula
  - MATLAB Script
  - Function & Object Browser Ctrl + l
  - UserObject
  - UserFunction
  - Call
  - Import Library
  - Delete Library
  - Sequencer
  - Virtual Source
    - Function Generator
    - Pulse Generator
    - Noise Generator
  - Regression
  - Counter
  - Accumulator
  - Timer
  - Shift Register
  - DeMultiplexer
  - Comparator
  - ActiveX Automation References…
  - ActiveX Control References…
  - ActiveX Controls (See Menu extensions)

- I/O
  - Instrument Manager…
    - Advanced I/O
      - Interface Operations
      - Instrument Event
      - Interface Event
      - MultiInstrument Direct I/O
  - Bus I/O Monitor
    - To
      - File
      - DataSet
      - Printer
      - String
      - StdOut (UNIX)
      - StdErr (UNIX)
    - From
      - File
      - DataSet
      - String
      - StdIn (UNIX)
    - To/From Named Pipe (UNIX)
    - To/From Socket
    - To/From DDE (PC)
  - Rocky Mountain Basic (UNIX)
    - Initialize Rocky M'tn Basic (UNIX)
    - To/From Rocky M'tn Basic (UNIX)
  - Execute Program (UNIX)
  - Execute Program (PC)
  - Print Screen

- Data
  - Selection Control
    - Radio Buttons
    - Cyclic Button
    - List
    - Drop-Down List
    - Pop-Up List
    - Slider List
  - Toggle Control
    - Button
    - Check Box
    - Vertical Paddle
    - Horizontal Paddle
    - Vertical Rocker
    - Horizontal Rocker
    - Vertical Slide
    - Horizontal Slide
  - Dialog Box
    - Text Input
    - Int32 Input
    - Real64 Input
    - Message Box
    - List Box
    - File Name Selection

Data (*continued*)
- Continuous
  - Int32 Slider
  - ueal64 Slider
  - Int32 Knob
  - Real64 Knob
- Constant
  - Text
  - UInt8
  - Int16
  - Int32
  - Real32
  - Real64
  - Coord
  - Complex
  - PComplex
  - Date/Time
  - Record
- Variable
  - Set Variable
  - Get Variable
  - Delete Variable
  - Declare Variable
- Build Data
  - Coord
  - Complex
  - PComplex
  - Waveform
  - Arb Waveform
  - Spectrum
  - Record
- Unbuild Data
  - Coord
  - Complex
  - PComplex
  - Waveform
  - Spectrum
  - Record

Data (*continued*)
- Allocate Array
  - Text
  - UInt8
  - Int16
  - Int32
  - Real32
  - Real64
  - Coord
  - Complex
  - PComplex
- Access Array
  - uet Values
  - Get Values
  - Set Mappings
  - Get Mappings
- Access Record
  - Merge Record
  - SubRecord
  - Set Field
  - Get Field
- Concatenator
- Sliding Collector
- Collector

Display
- AlphaNumeric
- Logging AlphaNumeric
- Indicator
  - Meter
  - Thermometer
  - Fill Bar
  - Tank
  - Color Alarm
- XY Trace
- Strip Chart
- Complex Plane
- X vs Y Plot
- Polar Plot
- Waveform (Time)

Display (*continued*)
- Spectrum (Freq)
  - Magnitude Spectrum
  - Phase Spectrum
  - Magnitude vs Phase (Polar)
  - Magnitude vs Phase (Smith)
- Picture
- Label
- Beep
- Note Pad

Window
- Cascade
- Tile Horizontally
- Tile Vertically
- Arrange Icons
  - by Position
  - by Name
- Minimize All
- Close All
- 1 Main

Help
- Contents and Index
- Welcome
- MATLAB Script
  - Help Desk
  - MATLAB on the Web
    - (See Menu Extensions)
- Agilent VEE on the Web
  - Product Registration
  - Support
  - Instrument Drivers
  - Education/Training
  - Agilent VEE Home Page

Help (*continued*)
  - Agilent Technologies Home Page
- Open Example…
- About VEE Pro

Tool Bar Buttons

New (Ctrl + N)
Open (Ctrl + O)
Save (Ctrl + S)
Print Screen
Tool Bar Buttons (*continued*)
Run (Ctrl + G)
Pause (Ctrl + P)
Stop (and reset from the latest run)
Step Into (Ctrl + T)
Step Over
Step Out
Toggle Breakpoint (Ctrl + B)
Show/Hide Call Stack
Show Execution Flow
Show Data Flow
View Variables
Find… (Ctrl + F)
Previous Find Results (Ctrl + R)
Default Preferences
Instrument Manager
Function & Object Browser (Ctrl + 1)
Cut (Ctrl + X)
Copy (Ctrl + C)
Paste (Ctrl + V)
Add to Panel
Delete Line (Ctrl + Shift + LB)
Show/Hide Program Explorer
Show/Hide Profiler

Mouse Instructions

These instructions are for the left-hand mouse button:

This statement:	*means you should:*
Click on an item	Place the mouse pointer over the desired item and press the left mouse button.
Double-click on an item	Place the mouse pointer over the desired item and press the left mouse button twice quickly.
Drag an item	Place the mouse pointer over the desired item, hold the left mouse button down, and move the item to the appropriate location. Then, release the mouse button.

The *right*-hand mouse button is to be depressed only when specifically noted in the text.

Programming

When solution steps for solving a decision portion of a program become complex, then it can first be prepared as a flow chart or as pseudo code. Flow charts are either structured (single entry at the beginning with a single exit at the end) or unstructured which consists of boxes connected by lines with multiple entry and exit points. Pseudo code is programming-like notation, ranging from simple English words to programming steps. A Pascal pseudo-code example for calculating interest is:

Pseudo Code	*Pascal*
BEGIN	PROGRAM SIMPLEINTEREST (OUTPUT);
Set the PRINCIPAL	
amount invested at $620	VAR
Set the interest RATE at 0.075 (7.5%)	PRINCIPAL, RATE, INTEREST, REAL;
Calculate the INTEREST as RATE *	
PRINCIPAL	BEGIN
WRITE the PRINCIPAL and	
the interest RATE	PRINCIPAL := 620;
WRITE the amount of	
INTEREST earned	RATE := 0.075;

Pseudo Code (continued)

END

Pascal (continued)

```
    INTEREST := RATE * PRINCIPAL;
    WRITELN (PRINCIPAL, RATE);
    WRITELN (INTEREST)
END.
```

VEE Pro Data Flow

Data flows from left to right through an object. On all the objects with data pins, the left data pins are inputs and the right data pins are outputs.

All of an object's data-input pins must be connected. Otherwise, an error will occur when the program is run. If you do not wish to use them, then delete those pins.

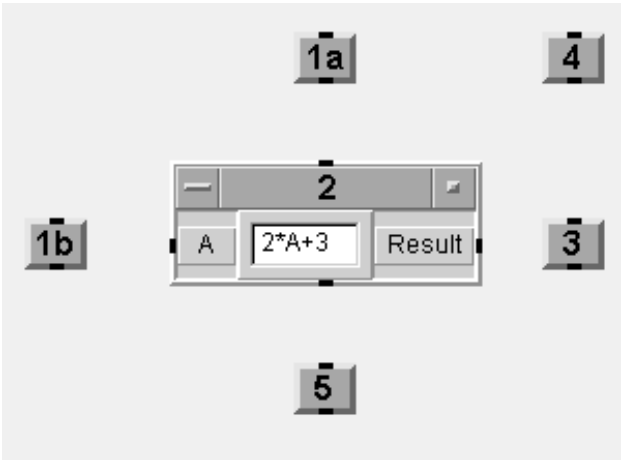
An object will not execute until all of its data-input pins have received new data.

An object finishes executing only after all connected and appropriate data-output pins have been activated.

You can change the order of execution by using sequence input and output pins. You do not normally need to use these pins except to ensure the order of execution (when controlling external devices such as instruments). Let data flow control the execution of your program.

Object Order of Operation

The order in which things happen *within* a VEE Pro object is:



1a. If the object's Sequence Input pin is connected, then the object will wait for both the Data Input and Sequence Input to receive data (or a ping) before it goes to step 2 below.

1b. The object accepts data on its Data Input Pin(s). It will wait until there is data on all Data Input Pins.
2. The object performs its designated function.
3. The object sends data out its Data Output Pin(s).
4. The object waits for a “receipt acknowledged” signal. The “receipt acknowledged” is then sent when every object “down-thread” from this object has completed its operation.
5. The object sends its signal out its Sequence-Out pin.

The object then deactivates.

Appendix B
Partial Programming Sequences

Introduction

This appendix identifies those portions of programs that will help you during the preparation of your programs. They are known as Partial Program Sequences. This is a different type of index; reference to these portions of programs is listed alphabetically. Thus, you may quickly reference partial programs that you use infrequently.

VEE Pro Subprograms

Appendix C
Virtual Devices and Instruments

Introduction

This appendix presents the characteristics of the VEE Pro virtual devices and instruments available within VEE Pro. For each of these items, a summary of their "help" menus is given.

Table of Contents Page Number

Devices

The Formula Object performs an arithmetic multiplication on two operands by using a*b to multiply the values of two input containers (terminals or pins) labeled: a and b. These containers may be of any type or shape. If one of the containers is an array, then the other must be either a scalar or an array of the same size and shape. The result is a container of the highest type, with the same shape as the operands. If both operands are of type Coord, then they must have their independent variable(s) match exactly. Otherwise, an error is returned. The multiplication is only performed on the dependent (last) variable. If one of the containers is Text, then the other must be an Int32. Text multiplication consists of repeating the string the number of times given by the value of Int32. Enums are converted to Text for multiplication. This multiplication operation performs a parallel multiplication on all elements of the arrays, including matrices. For a matrix multiply, see the function matMultiply(A,B). For detailed information, see the Formula Help dialog box.

The Menu Bar => Device => Function & Object Browser contains an enormous number of selections, such as:

- Type
  - Operators
  - Built-in Functions
  - MATLAB® Functions
  - Local User Functions

Imported User Functions
Remote User Functions
Compiled Functions
ActiveX Objects
VEE Objects
Instruments

Category
<All>
ActiveX Automation
Array
Bessel
Bitwise
Calculus
Complex Parts
Data Filtering
Generate
Matrix
Panel
Power
Probability & Statistics
Real Parts
Signal Processing
String
System Information
Time & Date
Trig
Type Conversion

For Built-in Functions; Category – <All>: The available Functions are:

abs	acos	acosh	acot	acoth
Aj	asComplex	asCoord	asin	asinh
asInt16	asInt32	asPComplex	asReal32	asReal64
asText	asUInt8	asVariant	asVariantBool	asVariantCurrency
asVariantDate	asVariantEmpty	asVariantError	asVariantNull	atan
atan2	atanh	bartlet	baseName	beta
Bi	binomial	bit	bitAnd	bitCmpl
bitOr	bits	bitShift	bitXor	blackman
ceil	charToint	clearBit	clipLower	clipUpper
cofactor	comb	commandLine	concat	conj
convolve	cos	cosh	cot	coth
CreateObject	cubert	defIntegral	deriv	derivAt

det	dirName	dmyToDate	erf	erfc
errorinfo	exp	exp10	factorial	fft
floor	fracPart	gamma	getEnv	getHostname
GetObject	hamming	hanning	help	hidePanel
hmsToHour	hmsToSec	i0	i1	identity
ifft	im	inDaylightSavings	init	installDir
integral	intPart	intToChar	inverse	
isVariant	isVariantBool	isVariantCurrency	isVariantDate	isVariantEmpty
isVariantError	isVariantNull	j	j0	j1
jn	k0	k1	lockPosition	log
log10	logMagDist	logRamp	mag	magDist
matDivide	matMultiply	max	maxindex	maxX
mday	mean	meanSmooth	median	min
minIndex	minor	minX	mode	Month
movingAvg	now	ordinal	panelPosition	panelSize
perm	phase	poly	polySmooth	product
programName	ramp	random	randomize	randomSeed
re	recip	rect	rms	rotate
round	savePanelImage	sdev	setBit	showPanel
signof	sin	sinh	sort	sq
sqrt	strDown	strFromLen	strFromThru	strLen
strPosChar	strPosStr	strRev	strTrim	strUp
sum	tan	tanh	totSize	transpose
typeName	unlockPosition	vari	wday	whichOS
whichPlatform	whichVersion	xcorrelate	xlogRamp	xramp
y0	y1	year	yn	

See: “The Function & Object Browser Help” dialog box for each of the above selections.

The UserObject is an object that may contain other objects. It is used to logically and physically group objects together. It constitutes a separate “context” from the root context of a program. Multiple UserObjects may be nested within a program. Create a UserObject by moving other objects into the work area of the Open View of that UserObject (or select objects and use Menu Bar => Edit => Create UserObject). The Make UserFunction item in the Object menu allows you to convert the UserObject into a UserFunction. That UserFunction can then be called using the Call Function Object or from certain expressions. No objects inside of a UserObject will operate until all data inputs to the UserObject are

activated. All operations inside of the UserObject must be completed before the data outputs are activated. When objects inside the UserObject are connected to objects outside the UserObject, data-input pins and data-output pins are automatically created whenever Create UserObject is selected. Add data-input pins and data-output pins to existing UserObjects using Add Terminal from the Object menu. The right mouse button will provide a pop-up menu when the pointer is over an object within the UserObject. Additional information is given in the UserObject Help dialog box.

The UserFunction is created empty from Menu Bar => Device => UserFunction or converted from a UserObject by executing Make UserFunction. You can also create a UserFunction by selecting a group of objects and using Menu Bar => Edit => Create UserFunction. Edit UserFunction will allow you to edit the UserFunction once it has been created. UserFunctions exist in the "background" and can be called with Call Function or from certain expressions. The UserFunction will have the same functionality as the UserObject(s). When you convert a UserObject to a UserFunction, the UserObject will disappear from the screen and will be replaced with a Call Function Object containing a call to the new UserFunction and the new UserFunction window. The UserFunction will be added to the list of available UserFunctions that exist in the "background" within the program. You can call the UserFunction using the Call Function Object, or from certain expressions. You can edit the UserFunction by using Menu Bar => Edit => Edit UserFunction or by double-clicking on it from the Program Explorer.

The Sequencer is a transaction-based object. It has program-development benefits that include:

- easy development of a test plan,
- a wide array of branching capabilities between tests,
- major components for building a customized test executive,
- the ability to call tests in VEE Pro and other languages, and
- automatic logging of test results.

More information is available from the (Test) Sequencer Help menu. The Sequencer is a very powerful object that can also do simple operations well.

The Sequencer Object executes tests in a specified order based upon previous or programmed results. Each test may be

- a VEE Pro User Function,
- a Compiled Function,
- a Remote Function that returns a single result, or
- any expression or a combination of the above functions calls in an expression.

Several variable names are also available, such as "thisTest" to correspond to the currently executing text. Also, the name of each test that has already been executed is available as a record, such as "test1.Pass".

That result is compared to a test specification to determine whether or not it has passed pre-established criteria. This result can also be an expression or a function call. The Sequencer then uses a pass-or-fail

indicator to determine the next test it should perform. More information is available from the (Test) Sequencer Help menu.

The options for branching to the next test are:

- executing the next test,
- repeating the same test, or
- jumping back to an earlier test.

The Sequencer can ask for user input to decide the appropriate course of action. As a test-sequence developer, you can specify the order of tests. Then you can instruct your program to branch to a different test based upon an analysis of your previously completed tests.

After the specified tests have been executed, the Sequencer automatically logs the test data to an output terminal. The specific field log can be customized in the Object properties. The data can then be analyzed and displayed. It can also be stored in a file, such as a server file, for later analysis.

Virtual Sources

The Virtual Source icon contains a Function Generator, Pulse Generator, and a Noise Generator. (See Menu Bar => Device => Virtual Source.)

The Function Generator Object is used to simulate repetitive waveforms. It will provide sine, cosine, square, triangle, +ramp, and -ramp waveforms as well as a simple Dc (horizontal) line. It will allow you to vary the waveform frequency, amplitude, Dc Offset, phase, time span, and number of points in the display.

The Pulse Generator Object is used to simulate pulses. It will allow you to vary the pulse frequency, width, delay, thresholds, rise time, fall time, pulse high value, and pulse low value. It also contains a Burst mode whose burst count, repetition rate, time span and number of points in the display can be varied.

The Noise Generator Object is used to simulate noise that can be added to a waveform. Its amplitude, time span, and number of points in the display can be varied.

Other Devices

The Regression Object is used to fit a linear regression line to (x,y) data. It uses the linear regression to fit a straight line to the data using the equation

$$y = c0 + c1*x$$

where x is the x coordinate, and c0 and c1 are calculated coefficients. This type of regression should be thought of as fitting the best straight line through the data. The (linear) Regression Object expects an array of Coord type of input with one independent variable – an (x,y) pair

- If the input array is not a Coord type, then an attempt is made to convert it to Coord.
- If the input data are mapped (Waveform, Spectrum, or a mapped array), then the conversion to Coord uses the mapping information to create the x part of the (x,y) pairs.

- If the input data are not mapped (example: an array), then the x part of the (x,y) pair is implicitly generated from its position in the array.

The Fit Type field on the open view is used to change the regression type to linear, logarithmic, exponential, power curve, or polynomial regression. Click on the field to bring up a list of the different regression types.

The Counter can also be cleared by activating its Clear Control input pin. It can be cleared at Prerun or cleared at Activate Prerun at Program Start. The object displays the number of times its data-input pin has been activated by a previous object's output. It provides this number, a Real scalar, on its data-output pin. The data input for Counter does not require any particular type of data. It even counts an input with a nil value. The Counter can also be cleared by activating its input Clear control pin.

The Accumulator Object displays a running sum (total) of its input values. It provides that value on its data-output pin. Its Clear pin clears the contents of the Accumulator. It can be cleared at PreRun or cleared at Activate. The type of the accumulated output data is the same as the highest order of its input value. Thus, if the Accumulator data input is activated with a Real and an Int32, its output is a Real. If it accumulates a Complex, the Accumulator converts the previously received Real data to Complex. If one of the inputs is text, then all of the data will be converted to text. For further warnings, see "Use caution" in the Accumulator Help dialog box.

The Timer Object provides an output in seconds that is the difference between the activation times of the top and bottom data-input pins. It uses the high-resolution performance counter if it is available in your system. The Timer is used to measure how much time passes between two events. This occurs when the first and second inputs are activated. If the second input is activated before the first, the container on the second input is ignored and the Timer Object does not execute. The Timer can be cleared at PreRun or cleared at Activate. Timing a VEE Pro program can be tricky whenever multiple input terminals are connected to the same output pin. (There is no control over which output line runs first.)

The Shift Register Object provides an output of the previous values of its inputs. It is used to access the previous values of its input. It can be cleared at PreRun or cleared at Activate. It starts with all its outputs set to nil. Each time the Shift Register is activated, the data from its input is copied to the "current" output terminal. Data that were in the current output terminal are moved to the 1 Prev output terminal. Data from previous executions are shifted down the output terminals to the last output terminal. Additional outputs may be added so the user may address the data from the "n" previous executions of a thread or program. Turn Clear at PreRun and Clear at Activate off (located in the Properties dialog box) to retain data over successive program executions.

The DeMultiplexer Object directs its input value to a selected output pin. That output is activated depending upon the value of the address input. The DeMultiplexer has two inputs: one for data output and one for the address value. (It is the address value that determines the output to be propagated.) Only one output is propagated each time the object operates. If the value of the address input is not within the range of the number of outputs [0=>(N-1)], then an error is generated. Additional outputs can be added to the DeMultiplexer. Outputs can be deleted, but then the following outputs (if there are any) are re-numbered in order.

The Comparator Object compares two data-input values. It then places the coordinates (where the comparison failed) on its data-output pin. It is used to compare a numeric test value, such as a Waveform or Coordinate, with a reference value. If one or more of the values in Test Value fails the comparison, then a Coord Array ad is placed on the Failures data-output pin and the Failed data-output pin activates. If all values pass the comparison, then an empty coordinate is placed on the Failures data-output pin and the Passed data-output pin activates. The independent field (x) in the Failures output Coord Array ad contains the index or mapping of each failure point. The dependent field (y) contains the Test Value that failed. You may change the comparison operator buy clicking on it and selecting the function from the list that is displayed. Default is Test Value == Ref Value.

ActiveX Automation References is a list of all Registered Automation Servers. This list is contained in its object dialog box. It lets you access the available ActiveX automation libraries. Only those libraries appear if their associated applications have been installed.

Select the libraries you want and click OK. This loads the selected libraries into memory for use with VEE Pro. You may now search for their object classes, dispatch interfaces, and exported events. The selected libraries will appear in the Function & Object Browser.

Use the built-in functions CreateObject() or GetObject() to connect libraries with an automation object. Select libraries, open the VEE Pro Function & Object Browser (Menu Bar => Device => Function & Object Browser); and select ActiveX Objects in the Type: area to generate expressions in Formula Objects that manipulate the objects. Use additional Formula expressions to “get” and “set” their properties and “call” methods.

If a library exists on your system that does not appear in the list, click Browse to search for it in the Add ActiveX Automation Reference dialog box. Once you find and open the library, VEE Pro adds it to the list and attempts to register it.

VEE Pro must be set to its VEE5 or higher execution mode for ActiveX support to occur. Execution mode (compatibility) is set in Menu Bar => File => Default => Preferences. Adding Automation References lets you use VEE Pro as an Automation Controller for applications capable of acting as Automation Servers. You may now interact with Microsoft Word, Excel, and Access for activities such as sending data from VEE Pro for report generation. ActiveX Automation Servers each run their own process. For example, it VEE Pro is controlling Word to perform Report Generation, then VEE and Word are running as separate processes.

ActiveX Control References provides a list of Registered Controls. In Windows, it displays the ActiveX Control References dialog box. This box lets you choose ActiveX controls available for use in VEE Pro.

Use the ActiveX Control References dialog box to select the ActiveX controls that you want to use in your VEE Pro program. Controls appear in this dialog box if they or their associated applications have been installed so the Windows Registry recognizes them. When you select the controls that you want, click OK to load them into memory for use in VEE Pro, search for their object classes, dispatch interfaces, or export events.

Menu Bar => Device => ActiveX Controls allows you to pick a control and place it in your program’s detail view. The resulting control “object” appears with a variable name in its title bar. Since controls have no pins like other objects, you must manipulate it using the control’s variable name in Formula expressions. Function & Object Browser helps you generate these expressions.

If a control exists in your system that does not appear in the list, click Browse to search for it in the Add ActiveX Control Reference dialog box. Once you find it, open that control. VEE Pro will automatically add it to the list and will attempt to register it. Checkmark its box; click OK.

ActiveX controls are available as individual products. They may also be installed as part of a larger application. ActiveX controls are loaded into VEE Pro process space directly. If an ActiveX control corrupts memory, then the VEE Pro program may crash. This is a major difference between ActiveX Automation Servers and ActiveX controls.

Displays

AlphaNumeric is an Object that displays alphanumeric data. It is used to display any of the data types as a single value, an Array 1D, or an Array 2D. An array is viewed by scrolling the display with the scroll bars. Enable Indices will add an index to each line in the display area. The AlphaNumeric display may be cleared at PreRun or cleared at Activate. Global format, Integer, Real, and Significant Digits are described in more detail in the AlphaNumeric Help dialog box.

Logging AlphaNumeric is an object that displays a Scalar or Array 1D of alphanumeric data without overwriting the display. It is used to display consecutive input values, thus providing a history of all received values. Clear will clear all the data that were displayed on Logging AlphaNumeric; Clear may be added as a control input. Logging AlphaNumeric may be cleared at PreRun or cleared at Activate. Its Buffer Size value may be changed to whatever size you prefer. However, if your data points exceed the buffer size, then previously received values will be overwritten. Much more information is contained in the Logging AlphaNumeric Help dialog box.

Indicators include:

Meter is an object that graphically displays a Scalar numeric value on an analog meter face whose default is zero center. The meter minimum and maximum values are set on the Open View of Meter by clicking on the preset value and typing a new value. If Show Digital Display is selected, then a digital display of the input value is displayed at the bottom of the object's Open View. Additional information is given in its Help dialog box.

Thermometer is an object that displays a Scalar numeric value on an analog scale using a color bar inside a graphical thermometer. Thermometer minimum and maximum values are set on the Open View of Thermometer by clicking on the preset value and typing a new value. (Its output may be attached to a display that will provide the digital numeric value displayed on the analog thermometer.) These values may also be added as control inputs. Thermometer may be cleared at PreRun or cleared at Activate. Thermometer may be turned to a horizontal display. If Show Digital Display is selected, then a digital display of the input value is displayed at the bottom of the object's Open View. High, low, and warning colors can be added to Thermometer. All input data must be Scalar and be convertible to Real. Additional information is given in its Help dialog box.

Fill Bar is an object that displays a Scalar numeric value on an analog scale using a color bar. There is an optional digital numeric field. Fill Bar minimum and maximum values are set on its Open View by clicking on the preset value and typing a new value. These values may also be added as control inputs. Fill Bar may be cleared at PreRun or cleared at Activate. Fill Bar may be turned to a horizontal display. If Show Digital Display is selected, then a digital display of the input value is displayed at the bottom of the object's Open View. High, low, and warning colors can be added to Fill Bar. All input data must be Scalar and be convertible to Real. Additional information is given in its Help dialog box.

Tank is an object that displays a Scalar numeric value on an analog scale using a color bar. There is an optional digital numeric field. Tank minimum and maximum values are set on its Open View by clicking on the preset value and typing a new value. These values may also be added as control inputs. Tank may be cleared at PreRun or cleared at Activate. Tank may be turned to a horizontal display. If

Show Digital Display is selected, then a digital display of the input value is displayed at the bottom of the object's Open View. High, low, and warning colors can be added to Tank. All input data must be Scalar and be convertible to Real. Additional information is given in its Help dialog box.

Color Alarm is an object that displays a different color and text string that depends upon the value of its Scalar input. It can be used as an LED that displays a color and a text string, based upon an input value and user-defined ranges. The input value may also be displayed in an optional digital numeric field. Color Alarm may be cleared at PreRun or cleared at Activate. The alarm shape may be changed from circular to rectangular. A border may be added to give the Color Alarm a three-dimensional appearance. If Show Digital Display is selected, then a digital display of the input value is displayed at the bottom of the object's Open View. Additional information is given in its Help dialog box.

XY Trace is two-dimensional, Cartesian-plot display. It displays mapped arrays or a set of values when y data is generated with evenly-spaced x values. XY Trace is useful for a quick look at data when you neither need nor have scaled x data values. The automatically generated x value depends upon the data type of the trace data. If the trace is a Real value, then the x values are 0, 1, 2, …. If the trace is a waveform, then the x values are time values. Open View parameters are controlled by selections on the object menu and in the Properties dialog box. Click on Auto Scale to automatically rescale the both axes of the display after data points are collected. Zoom is selected by dragging on the graph area. A "rubber band" box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Strip Chart is an object that displays continuously-generated data, thus providing a recent history of that data. The Step size automatically increments the input value; it will cause scrolling when new data "runs off" the right side of the display. Strip Chart saves all data sent to it until you clear it unless its buffer is set to a finite value. (Default value is 10 000.) Array data will be appended to the end of the Strip Chart trace without clearing the trace. Much additional information is given in its Help dialog box.

Complex Plane is an object that displays continuously-generated complex data as a Cartesian plot. It will display Complex, PComplex, or Coord data values on its Real versus Imaginary axes. Open View parameters are controlled via its Properties dialog box. Auto Scale will automatically rescale the display to show the entire trace. Zoom is selected by dragging on the graph area. A "rubber band" box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

X vs Y Plot is an object that displays a Cartesian plot using the same x-axis values. Open View parameters are controlled via its Properties dialog box. Auto Scale will automatically rescale the display to show the entire trace. Zoom is selected by dragging on the graph area. A "rubber band" box will appear. It will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Polar Plot is an object that displays a graphical plot in polar coordinates when separate polar information is available for radius and angle data. When more than one trace is to be plotted, each execution of the Polar Plot Object uses the single-angle input data with each trace's Radius-input data. Therefore, all traces share

the single-angle input. In is selected by dragging on the graph area. A “rubber band” box will appear. In will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Waveform (Time) is an object that displays time-domain waveforms on a two-dimensional graphical display. Spectrums are automatically converted to waveforms via an Inverse Fast Fourier Transform (ifft). The X axis is in the sampling units of the input waveform, typically “seconds”. The Open View parameters are: Mag, Trace1, and Time. Auto Scale automatically rescales the display to show the entire trace. Zoom is selected by dragging on the graph area. A “rubber band” box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Spectrum (Freq) includes the following:

Magnitude Spectrum is an object that graphically displays a spectrum in the frequency domain. Waveforms are automatically converted to spectrums via a Fourier Transform. The X axis is in the sampling units of the input Spectrum, typically frequency. The Open View parameters are: Mag, Trace1, and Time. Auto Scale automatically rescales the display to show the entire trace. Zoom is selected by dragging on the graph area. A “rubber band” box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Phase Spectrum is an object that graphically displays the magnitude of a spectrum as a function of its phase angle versus frequency. Waveforms are automatically converted to spectrums via a Fourier Transform. The X axis is in the sampling units of the input spectrum. The Y axis units are the chosen Trig Mode setting. The Open View parameters are: Phase, Trace1, and Freq. Auto Scale automatically rescales the display to show the entire trace. Zoom is selected by dragging on the graph area. A “rubber band” box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Magnitude vs Phase (Polar) is an object that displays a polar plot of Magnitude versus Phase of a complex spectrum. Waveforms are automatically converted to spectrums via a Fourier Transform. The Radius of each data point is the spectrum’s magnitude; the angle is the spectrum’s phase presented in the chosen trig units. Marker values are displayed in polar format (r:radius, a:angle) for the polar grid type and complex impedance or admittance for Smith Chart and Inv Smith Chart grid types. The Open View parameters are: Mag, Span, and Trace1. Changing the reference point in the entry field scrolls the part of the graph specified. Zoom is selected by dragging on the graph area. A “rubber band” box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Magnitude vs Phase (Smith) is an object that displays a polar plot of Magnitude versus Phase of a complex spectrum. Waveforms are automatically converted to spectrums via a Fourier Transform. The Radius of each data point is the spectrum’s magnitude; the angle is the spectrum’s phase presented in the chosen trig units. Marker values are displayed in polar format (r:radius, a:angle) for the polar grid

type and complex impedance or admittance for Smith Chart and Inv Smith Chart grid types. The Open View parameters are: Mag, Span, and Trace1. Changing the reference point in the entry field scrolls the part of the graph specified. Polar Reference Location specifies the part of the polar plot displayed. Ref Radius specifies the radius value of the bold graticule circle. Zoom is selected by dragging on the graph area. A "rubber band" box will appear. Zoom will then automatically magnify the display to contain only the rectangular region you select with the pointer. Next Curve resets the pen to display the next curve in a family of curves without clearing the previous curve. Center markers are also available. Much additional information is given in its Help dialog box.

Picture is an object used to display a graphic image on the Detail View or Panel View. You may tile, stretch, and clip the picture. The graphics file name may be added as a data input to the Picture Object. The directory shown is the bitmaps directory under the installation directory. Press the Browse… button to choose a file from another directory. You may add the file name as a data input. Changes are available, such as Actual, Centered, Scaled, Tiled, Preview, Bitmaps (*.BMP), X11 Bitmap (*.ICN), and GIF87a (*.GIF) for machines that have Windows installed. Default bitmaps are located in the bitmaps directory under the VEE Pro program installation directory. Additional information is given in its Help dialog box.

Label is an object that is used to place a text label on the Panel View. It is not executed. Its background color blends into the background color of the Panel View by default. In the Open View, Label's title is changed via Properties in the object menu. Its Help dialog box is opened via the mouse right button. Additional information is given in its Help dialog box.

Beep is an object that generates an audible tone of a specified frequency, duration, and volume. The Beep Object is hardware dependent; the computer must have a speaker.

Note Pad is an object that displays a block of text to document your program, label the Panel View, or write notes.

Appendix D
Definition of Technical Terms

Introduction

This appendix has been extracted from the applicable Agilent- staff-documents as noted in the Bibliography. The definitions have been modified where appropriate. For further information, select from the latest version of the VEE Pro CD ROM, www.agilent.com or go to:

Menu Bar => Help => Contents and Index => Reference => Glossary

in your VEE Pro program.

Technical Terms

Activate – The action that resets a UserObject's or UserFunction's internal controls each time before it operates.

Array – A data shape that contains a predefined arrangement of data that may require one or more dimensions.

Auto Execute – The operation of certain types of objects that occurs when one or more of its options are selected; it operates as a Start button.

Bitmap – A pattern or picture that can be displayed on an icon that can be modified by the programmer.

Buffer – A portion of memory where data are temporarily stored.

Button – A graphical object in VEE Pro that simulates a momentary switch or selection button; it appears to pop out from your screen. When you "press" a button by clicking on it with the mouse, an action occurs. (The instructions may be referring to either 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 menus and dialog boxes that allows you to select a setting by clicking on the box; a checkmark will appear in the box; it indicates that a selection has been made. The setting may be canceled by clicking on the box again.

Click – To press, then release, a mouse button. Clicking usually selects a menu feature or object in a VEE Pro window. See also Double-Click and Drag.

Clone – A menu item on the VEE Pro object menus that duplicates objects and their interconnections. It places a copy of them in the Paste buffer. Clone then copies all of the attributes of the cloned Objects including pins, parameters, and size. Cloning large groups of objects or UserObjects is not recommended; use UserFunctions for a more reliable design.

Compiled Function – A user-defined function that links a program written in a programming language; they can be called via the Call Function Object.

Component – A single instrument function or measurement value in a VEE Pro 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 that 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.)

Composite Data Type – A data type with a uniquely defined shape.

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.

Control Pin – An input pin that controls an object without waiting for the object's input pin(s) that contain data. As an example, when clearing a counter or AlphaNumeric, the object does not execute when the control pin fires.

Coupling – A relationship between two or more functions within an instrument such that changing the value of one of the functions automatically changes the value of another function. This is used in VEE instrument panel drivers.

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 Tool Bar button: Select Menu Bar => Edit => Paste

Data Container – The data package that is transmitted over lines and is processed by objects. Each data container contains data, the data type, the data shape, and mappings (if any).

Data Field – The field within a transaction that can be specified either by a WRITE or READ transaction.

Data Flow – The flow of data through and between VEE Pro objects. Data flows from left to right through objects. An object does not execute until it has data on all of its data-input pins. Data is then 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 Pro program.

Data Input Pin – A connection point, usually on the left side of an object, that propagates data flow through that object and to the next object.

Data Output Pin – A connection point, usually on the right side of an object, that propagates data flow from that object to the next object while passing the result(s) of the first object's operation onto the next object.

DataSet – A collection of Record containers saved into a file for later retrieval.

Data Shape – A structure that defines how data are grouped together, such as in an array.

Data Type – A structure that determines how data are organized and processed by VEE Pro.

DDE (Dynamic Data Exchange) – A means of communication that allows VEE Pro to communicate with other applications and programs. This technology has been replaced with ActiveX in (COM) and .NET.

Default – A setting that is automatically selected whenever an object is opened.

Default Button – A button with a recessed border that is activated by default if depressed.

Demote (to) – To convert from one data type that contains a certain amount of information to a second data type that contains less information.

Description Box – An area where the program description is explained; similar to the Note Pad.

Detail View – The view of a VEE Pro program that displays all the objects and how they are interconnected.

Device – An electronic instrument, such as a sensor, meter, or oscilloscope, that can be attached to a VEE Pro interface.

Dialog Box – An object internal window that indicates information selections that must be chosen prior to program continuation.

Direct I/O Object – A technique that allows VEE Pro to directly control an attached instrument.

DLL (Dynamically Linked Library) – A collection of functions (written in C) that can be called from VEE Pro.

DMA (Direct Memory Access) – A method of transferring data from an external board to internal memory without that data passing through the Central Processing Unit of your computer.

Double-Click – The act of rapidly depressing and releasing a mouse button twice.

Drag – The act of depressing and continuing to hold down a mouse button while moving the object or area involved.

Driver – Programs or routines that allow VEE Pro to communicate with other software or hardware.

Driver Files – Those files contained within VEE Pro that allow the creation of virtual instrument panel used in Panel Drivers. Current Driver Files are VXI *Plug&Play* and IVI-COM that do not have panels.

Drop-Down List – A list of selections obtained by clicking on the arrow to the right of a selection field.

Editing Area – The white space within an object that can be changed via typing characters. Portions of the editing area that cannot be changed become a light-gray color. For the Formula Object, the white space is also known as the Input Field or Expression.

Entry Field – A field that is typically part of a dialog box or an editable object used for data entry. An entry field is editable when its background is white.

Error Message – A presentation of information within the dialog box of an object; it indicates that an error has occurred.

Error Pin – A pin that displays the error number of an error message; it occurs in place of an error message.

Execute – The action that causes a program to run.

Execution Flow – The sequence in which objects operate whenever a program is run.

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.

Feature – An item on a menu that, when selected, causes a prespecified action to occur.

Feedback – A continuous-thread path that uses values from a previous execution to change values in the presently occurring execution.

Font – VEE Pro allows you to change the "font" – the size and style of type – used to display text for various VEE Pro objects, titles, and so forth.

Function – The behavior of an object where its output is a function of its input; the function is selected for an object via special sub-menus.

Global Variable – A globally-set variable that can be called by name. Variables can also be communicated Local to Library or Local to Context (local to a UserObject or UserFunction) by using the Declare Variable Object.

Grayed Feature – A menu feature that is displayed in gray rather than in black. This indicates that the feature is either not active or is 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.

Highlight – The band or shadow around an object that indicates the status of that object.

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 on such text, related information is presented.

Icon – See Iconic View

Iconic View – A small, graphical representation of a VEE Pro object, such as an instrument, control, or display;

or:

Iconic View – a small, graphical representation of an application, file, or folder in the Microsoft Windows and HP-UX (with VUE) operating systems.

Input Field – Another name for the Formula Object Editing Area white space.

Instrument Panel – An object that forces all the function settings of an attached instrument to match the settings in the displayed object control panel.

Interrupt – A signal designed to call for an action on the part of the program user.

Label – The text area or name on an icon or button that identifies that icon or button.

Library – A collection of objects that can be called during the design of another program.

Line – A link between two VEE Pro objects that transmits data containers that are being processed.

Main Menu – Those menus located within the VEE Pro Menu Bar; they are opened by clicking and dragging on the appropriate menu titles.

Main Window – A window that contains the primary Work Area in which you develop a VEE Pro program. The Work Area for this window resides in the workspace for the VEE Pro window.

Main Work Area – The area where all programs are initiated and designed.

Mapping – An associated set of independent values within an array that are identified in such a way that they can be easily accessed.

Maximize – To enlarge a window so it occupies all the Work Area available to it.

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 space.

Menu – A collection of alternative features that are provided in a list format.

Menu Bar – The bar along the top of the VEE Pro window that displays the titles of the pull-down menus from which you can select commands and objects.

Menu Title – The name of a sub-menu within a Menu Bar.

Minimize – To reduce an object or Window to its smallest size where it is then displayed as an icon.

Minimize Button – The button on an object, or on the VEE Pro window, that converts the object or the VEE Pro window to an icon.

Mouse – A piece of hardware that allows you to move across your screen so you may either move or select one or more objects or Work Areas.

Mouse Button – One of the two buttons on your mouse that allows you to perform a predefined action.

Mouse Pointer – The cursor that appears on the screen and, by moving the mouse, can be made to touch any desired object or area on the screen.

Network – Devices linked together so they can share data.

Object – 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. An object can be displayed as either an iconic view (icon) or as an open view.

Object Menu – the menu associated with an object that contains features which operate on the Object (for example: moving, sizing, copying, and deleting the object). To obtain the object menu, click on the object button on the upper-left corner of the object Title Bar;

or:

click on the right mouse button with the pointer over the object.

Object Menu Button – The button in the upper-left corner of an object, or on the VEE Pro window, that causes the object's Object Menu to appear.

Open View – A small, graphical representation of a VEE Pro Object that provides more detail than does the Iconic View; in this view you can change the object title and modify its operation.

Operate – The action of an object that is processing data; the result is an output of the processed data.

Outline Box – A wire-frame rectangular box that appears in your Work Area to indicate where the selected object will be placed.

Palette – A set of colors and fonts that allows objects and displays within those objects to be changed so they can be more easily observed.

Panel – The Work Area within a UserObject; the information displayed within an object's Open View.

Panel View – The view of a VEE Pro program that presents only those objects that a user is allowed to manipulate during the running of that program.

Pin – An external connection point on an object to which a connection line can be attached.

Pointer – The image that follows the movement of your mouse; they are available in different shapes and sizes so you can better distinguish their use.

Pop-Up Menu – A menu that can be accessed by clicking the right-hand mouse button in the object of interest.

PostRun – The set of actions that are performed when a program is stopped.

PreRun – The set of actions that resets your program and checks for errors before you run that program.

Priority Thread – A thread that blocks all other threads from executing until the priority thread executes.

Program – A set of objects connected with lines that represent one solution to a programmed problem.

Promote – An action that converts from one data type containing less information to a data type that contains more information.

Propagation (rules) – The data-flow rules that a program to be run must follow.

Pull-Down Menu – A menu that can be accessed via the menu bar.

Radio Button – A button within a dialog box that allows you to select a predefined setting; all radio buttons within a given dialog box must be mutually exclusive.

Record – A data type, such as a scalar, array, or other record container, that has named data fields containing multiple values.

Restore – The returning of a minimized window or icon to its full size.

Run – To initiate a program so it operates to completion if no programming errors exist.

Save – To write a program, or portion of a program, to a diskette, hard drive, or server for later access.

Scalar – A data shape that contains a single value.

Schema – The structure or framework of a record that defines that record; it includes the field name, type, shape, dimension size, and mapping information.

Screen Dump – A printout of a series of objects or programs that are presently displayed on your screen.

Scroll – The use of a scroll bar to allow you to examine a Work Area, a listing of data files, sub-menus, or other material displayed via a dialog box.

Scroll Arrow – An arrow that, when clicked, displays your movement through a list of data files, the Work Area, or a dialog box.

Scroll Slider – A rectangular bar that, when dragged, displays your movement through a list of data files, the Work Area, or a dialog box.

Select – To choose an object, an action to be performed, or a menu item by clicking the mouse.

Select Code – A number that identifies the address of a hardware interface.

Selection – The highlighting of one or more objects within a window.

Sequence Input Pin – The pin located on the top of an object.

Sequence Output Pin – The pin located on the bottom of an object.

Sequencer – An object that controls execution flow via a series of transactions; it can also call a UserFunction or Compiled Function.

Sleeping Object – An object that is waiting for an operation or time interval to be completed or for an event to occur.

Step – The activation of one object or operation at a time.

Terminal – The (internal) representation of a pin which displays information about that pin and its data container.

Thread – A set of objects that are connected by solid lines within a VEE Pro program.

Title Bar – The top rectangular bar on a window or on an object Open View; the title of the window or object may be displayed.

Tool Bar – The second rectangular bar on a window which provides the buttons that control VEE Pro programs.

Transaction – The specification for input and output used by certain objects; it appears as phrases in the Open View of these objects.

UserFunction – An object that can be created from a UserObject; it exists in the background but provides the same functionality as the original UserObject; it can be saved in a library.

User Interface – That portion of an application that allows the user and that application to communicate with each other so pre-specified tasks can be implemented.

UserObject – An Object that contains a group of interconnected objects; it can be saved in a library and reused.

UTC (Universal Time Coordinated; also known as Coordinated Universal Time) – The clock within VEE Pro, expressed in seconds, starting with AD 0001 January 01; it appears on the screen in scientific notation (powers of ten). It is also known as Greenwich Mean Time (GMT).

Wait – See “Sleeping Object”.

Window – A rectangular area on your screen that can contain your designed program.

Wire Frame Outline – The blank rectangle that appears on the Work Area when an object has been selected from the Menu Bar.

Work Area – The area within a VEE Pro window, or within the Open View of a UserObject, where objects are grouped together.

XEQ Pin – That pin which forces the operation of an object even though the data pins or sequence input pins have not been activated.

Appendix E
Introducing Objects, Icons, and Features

Lesson 1 Pre-lab

The Mouse
The Mouse controls a marker whose shape depends upon its application. The marker can be a:

- movable arrow that indicates where the marker is located on the screen,
- blinking "I" beam or a blinking vertical line that indicates your location in the text and allows you to select portions of a text,
- square containing four smaller squares that allows you to move the contents of an entire screen simultaneously,
- angled arrow when you want to "size" an object,
- magnifying glass when you are on a connecting line,

It takes other forms that will be explained as VEE Pro details are presented.

The conventions for Mouse left-button usage are:

- "Move" means to move the cursor across the screen with neither Mouse button depressed.
- "Click" means to click and immediately release the Mouse button.
- "Click and drag" or "drag" means to depress and hold down the Mouse button.
- "Double-Click" means to click rapidly the Mouse button twice.

Specific instructions will indicate when to click the right Mouse button, such as "Click the Mouse right button".

Opening the VEE Pro Program
Click Start; go to Programs; select Agilent VEE Pro 6.

Note: The first time that VEE Pro is run, you should get the "Welcome" window.

The VEE Pro Opening Development Screen
The four bars below have a series of icons similar to the layout on the Windows screen.

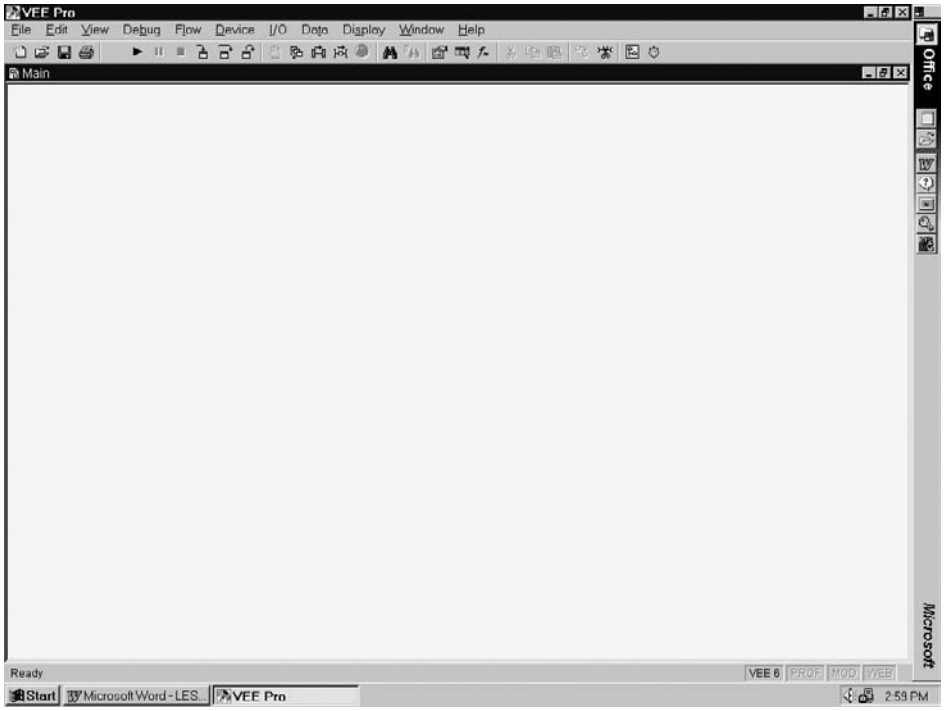


Figure 1.1. The VEE Pro Development Screen

Title Bar

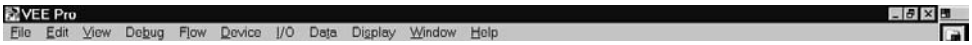


Figure 1.2. Title Bar

The Title Bar of any window, whether Microsoft™ Windows or VEE Pro, contains words, the program (Lab) name and, on the right, the three Window-Sizing buttons.

The three Window-Sizing buttons are:

1. The Minimize (_) button which will convert the program to a bar at the bottom of the screen,

2. The double-square Toggle button which will toggle the Work Area to either a full-screen size or to a lesser size, and
3. The Close (X) button which will eliminate the object; however, the oblect is not lost and is still available via the Program Explorer.

The program name (VEE Pro) is on the upper-left corner, next to the Agilent VEE Pro icon.

Menu Bar

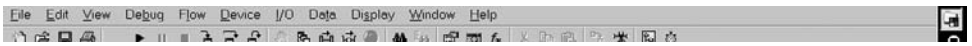


Figure 1.3. Menu Bar

The Menu Bar contains the pull-down menus to activate the program segments and open most of the components of VEE Pro. The Menu Bar contains the words File, Edit, View, Debug, Flow, Device, I/O, Data, Display, Window, and Help. Active items (words) are shown in bold type; the inactive words are shown in gray type. In some situations, the pull-down menus have layers to their right. The first four menu words are instructional; the next five are the components and operating instructions for the program; the last two organize the work area and provide Help if all else fails.

- The underlined letter informs you how to access that menu by using the Alternate (Alt) key.
- Each function level consists of a hold-down menu accessed via the mouse left button. The hold-down menu causes that function level to remain displayed on the screen while you access your desired function level.

Additional details for all words in the Menu Bar are given in Appendix A.

Tool Bar

Figure 1.4. Tool Bar

On the Tool Bar are icons for the frequently used items (shortcuts) from the Menu Bar above. Again, active icons are sharp and dark; the inactive icons are grayed. Icon descriptions are displayed in a yellow tooltip that appears within a few seconds after the icon is touched.

Example: Place the mouse pointer over the icon that represents a printer. That button will display a yellow tooltip that states the task that the button will perform: “Print Screen”. Additional Tool Bar details are given in Appendix A.

Note: The word “tooltip” is the standard “windows” name that refers to this yellow “pop-up”.

Status Bar(s)

Figure 1.5. Status Bar(s)

The Status Bar(s) are at the bottom of the screen. The top bar is the VEE Pro generated bar; the bottom bar is the Windows generated bar. (The lower-right icons are functions on the PC.)

The bar above the Status Bar indicates the status of the VEE Pro. Upon opening the program, the word "Ready" will be displayed on the left. The version of VEE Pro that is running (execution mode or compatibility mode) will be displayed on the right of this Status Bar.

The VEE Pro Screen

The VEE Pro Screen, labeled "Untitled" and "Main", consists of two areas: the Program Explorer on the left and the Work Area (named **Main**) on the right.

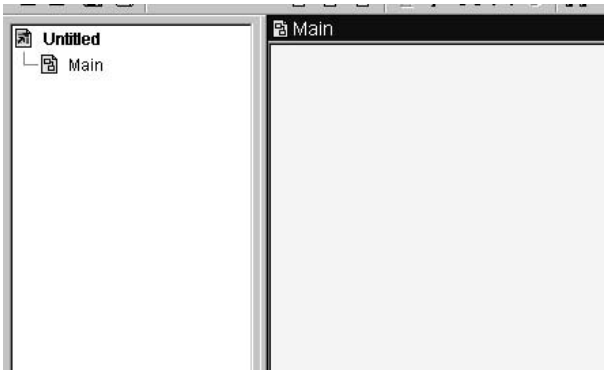


Figure 1.6. The VEE Pro Screen

Program Explorer

Figure 1.7. The Program Explorer

Program Explorer is accessed via the Menu Bar => View menu. If a check mark is displayed, the Program Explorer area will appear on the left side of the screen. Eliminating the checkmark will cause the Program-Explorer area to hide (go into the background). This is a toggle operation. The Program Explorer is a graphical and hierarchical diagram of the program being examined.

Work Area

Figure 1.8. The Work Area

This is the area (Main) where programs are developed and displayed. It is located to the right of the Program Explorer. Previously prepared programs can be displayed in this area so they can be run or modified. In the upper-right corner, there are buttons that will minimize, maximize, or close the Work Area.

In order to distinguish the Menu Bar from the Tool Bar, the notation: Menu Bar => is used. The Menu Bar contains the titles: File, Edit, View, Debug, Flow, Device, I/O, Data, Display, Window, and Help.

Example: Select Menu Bar => Device => Virtual Source => Noise Generator indicates that Noise Generator is the sub-sub-item within the sub-item Virtual Source within the item labeled Device on the Menu Bar.

Note: Each object will appear initially as a blank rectangle that must be placed in the Work Area. A single click will anchor and display that icon. Its default title and other properties can be accessed by double-clicking on its top horizontal bar, which is a shortcut to the Properties box.

Menu Bar => Display

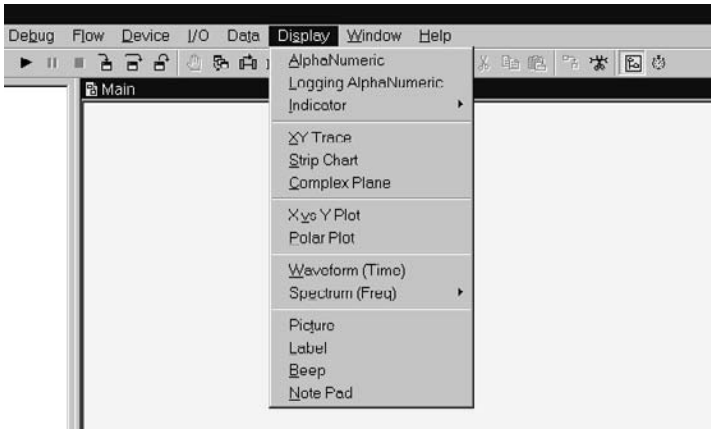


Figure 1.9. The Menu Bar => Display

From the Menu Bar, click sequentially on the following items:

Menu Bar => Display => Note Pad

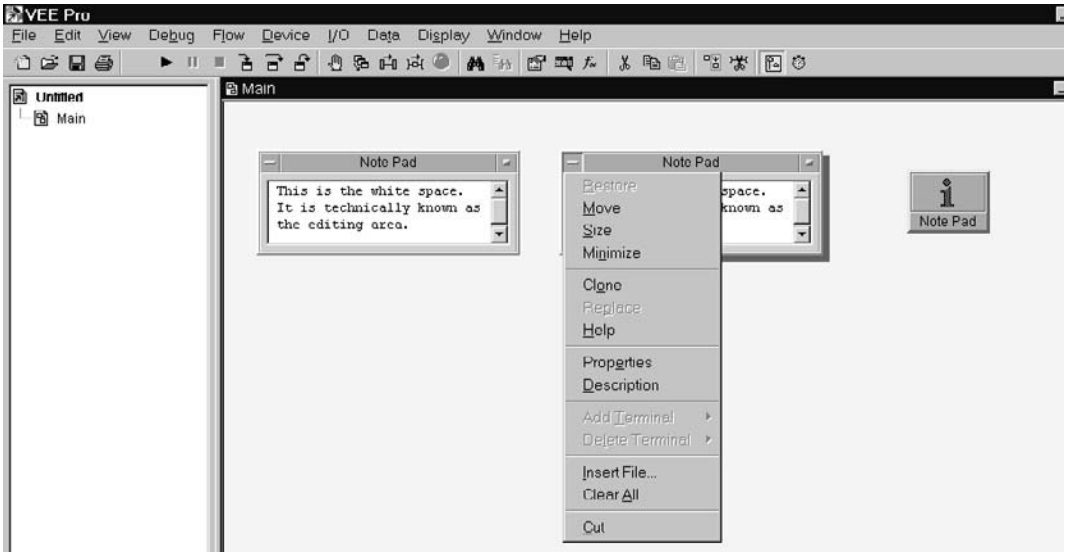


Figure 1.10. The Menu Bar => Display => Note Pad

Once the Note Pad is anchored, you may click inside the white space (edit area"). You will see a cursor, shown as a blinking I-bar. The Note Pad enables you to write notes that explain the program in its editing (white) area the left object in Figure 1.10.

The Note Pad upper-left symbol (**-**) can be used to access additional features (Restore, Move, Size, Minimize, Clone, Replace, Help, Properties, Description, Add Terminal, Delete Terminal, Insert File, Clear All, and Cut) the center object. These features will be described where appropriate. The "dot" symbol displayed in the upper right of this object reduces the Note Pad to its Iconic View – the right-hand object in Figure 1.10.

Note 1: The object left (**dash**) symbol is the object menu; the object right (**dot**) symbol converts the object to an icon. These symbols apply to all objects.

Note 2: The darkened features are the ones accessible in this situation.

Menu Bar => Display => Waveform (Time)

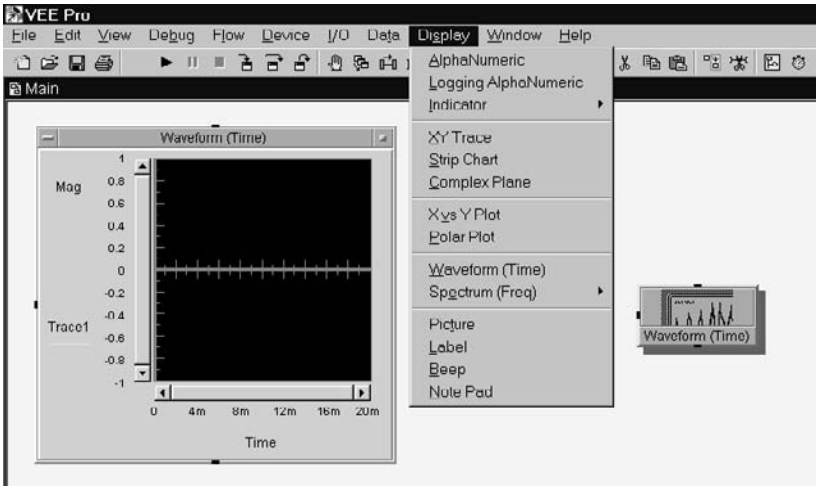


Figure 1.11. The Menu Bar => Display => Waveform (Time)

This object represents an oscilloscope; it displays magnitude (Mag) versus time (Time). The Waveform (Time) input (Trace 1) is on the left side of its object. The Waveform (Time) features are selected by clicking the upper left "–". These features are: Restore, Move, Size, Minimize, Clone, Replace, Help, Properties, Description, Add Terminal, Delete Terminal, Auto Scale, Zoom, Clear Control, Center Markers, Add Right Scale, Plot, and Cut. "Mag" characteristics can be changed by clicking on "Mag". The name "Mag" can be changed; also, its scale range and other parameters can be changed to match the application. The name, color, line type, and point type of "Trace1" can be changed by clicking on "Trace1". Also, additional traces can be added via the "Add Terminal => Data Input" that can be accessed via the upper-left symbol. The name "Time" and its parameters can be changed by clicking on "Time".

Note 1: Most terminals on an object will be explained by a tooltip banner (flag) that will appear if the mouse is held over that terminal, but not clicked, for approximately two seconds. If a square is displayed, it indicates that you can make a "wire" connection from that terminal to another terminal. A double-click will remove the initiated connection.

Note 2: Experience will show you those terminals whose name you can modify by clicking on that terminal.

Menu Bar => Display => AlphaNumeric

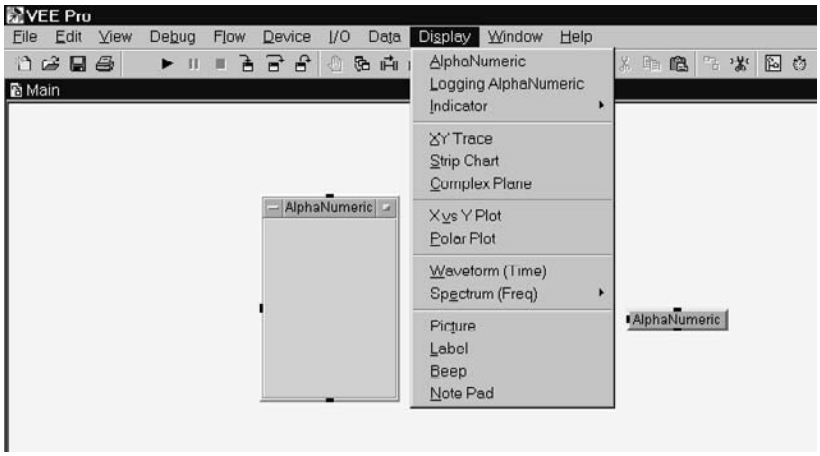


Figure 1.12. The Menu Bar => Display => AlphaNumeric

This box receives input from its left side. The characteristics are set by clicking the upper-left (-) symbol. Sizing of the area for data can be adjusted by clicking the lower right and moving the arrow. The upper right square converts it to an Iconic View.

Menu Bar => Device
From the Menu Bar, click on each of the items below.

Menu Bar => Device => Virtual Source => Function Generator

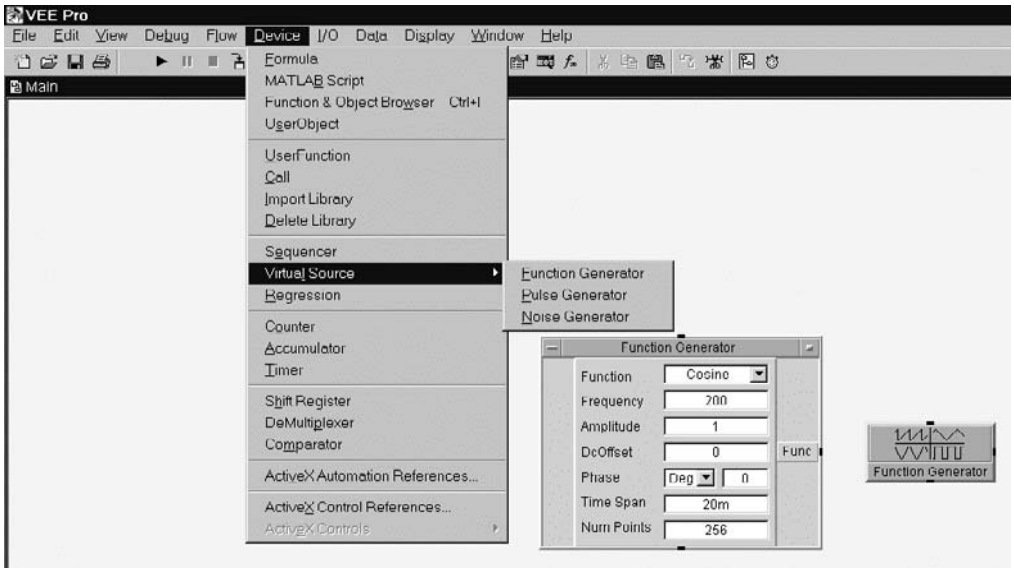


Figure 1.13. The Menu Bar => Device => Virtual Source => Function Generator

The Function Generator is the first Virtual Source. Its features are selected by clicking the upper left "–". These features are: Restore, Move, Size, Minimize, Clone, Replace, Help, Properties, Description, Add Terminal, Delete Terminal, and Cut. Generator parameters are altered within the Generator white (editing) space. There are seven functions available within the Generator Function pull-down menu. There are three Phase units available within the Phase pull-down menu. The Generator has an Output Terminal labeled "Func" on the right side for connecting the Generator to other objects. Double-clicking on Func causes a box labeled "Output Terminal Information" to appear. This box is valuable when troubleshooting after running a program.

Menu Bar => Device => Virtual Source => Noise Generator

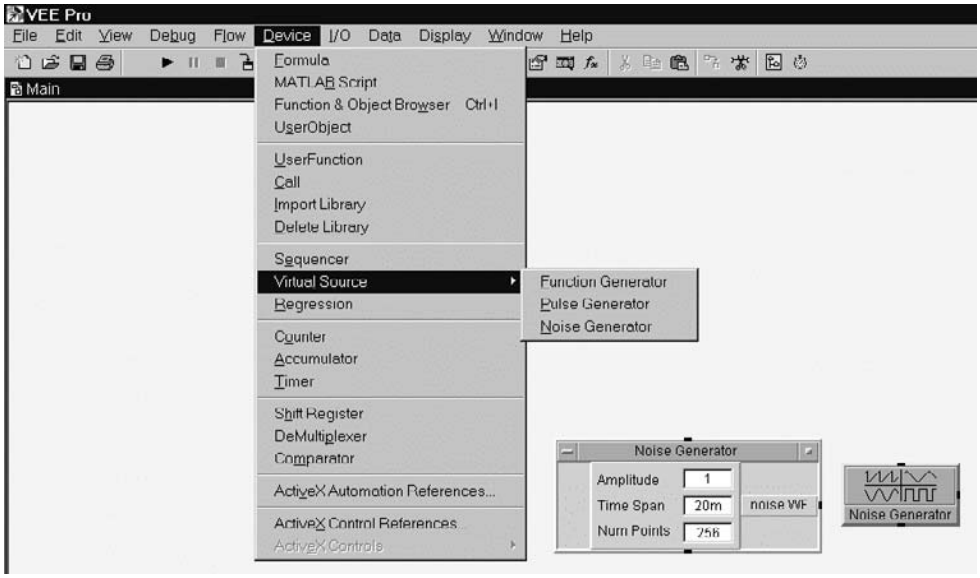


Figure 1.14. The Menu Bar => Device => Virtual Source => Noise Generator

The Noise Generator has the same features as the Function Generator. However, the name of its Output Terminal, initially "noise WF", can be changed if desired.

> **Note:** Hold down the mouse over any unused portion of the Work Area. Four squares within a square will appear. This different cursor icon allows the entire Work Area to be moved for better object positioning.

Menu Bar => Device => UserObject

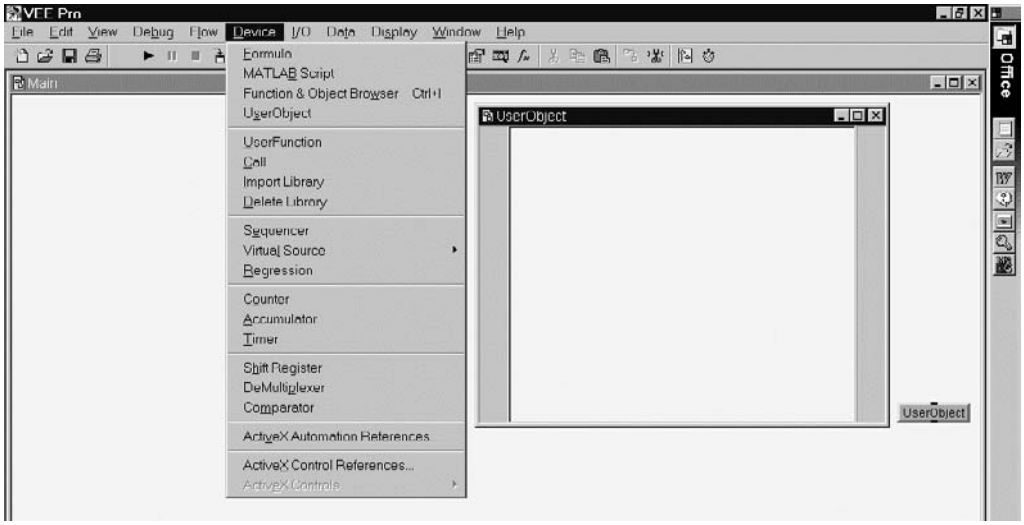


Figure 1.15. Menu Bar => Device => UserObject

The UserObject initially appears as a closed object within Main. It is opened by double-clicking on its Iconic View. The UserObject can contain one or more of interconnected objects that can be saved as a library for future use. Its title can be changed by double-clicking on its top horizontal bar (below the Tool Bar). Its Trig Mode can be changed; its Colors, and Fonts can be displayed.

The UserObject initially contains neither input nor output terminals. Examine the top three squares in the UserObject upper-left corner:

- Click on the minimize (_) button; the UserObject Iconic View will appear just above the Status Bar.
- Click on the overlapping double-square (maximize) button; the UserObject will shrink to a smaller area within Main.
- Click on the X button; the Open View of the UserObject will disappear. You can bring it back by double-clicking on the icon.

If the object totally disappears, then go to View => Program Explorer and toggle its title. Then double-click on the UserObject title; its window will reappear in Main.

UserObject features are selected by clicking on the (-) symbol in its upper-left corner. These features are: Restore, Move, Size, Minimize, Maximize, Cut, Copy, Clone, Help, Properties, Description, Add Terminal, Delete Terminal, Make UserFunction, Unpack, Save Secured Version, Find, Print, Locate, Calls, Create Panel, Delete Panel, and Close. The names for each added terminal may be changed by double-clicking on that terminal.

Menu Bar => Device => Formula

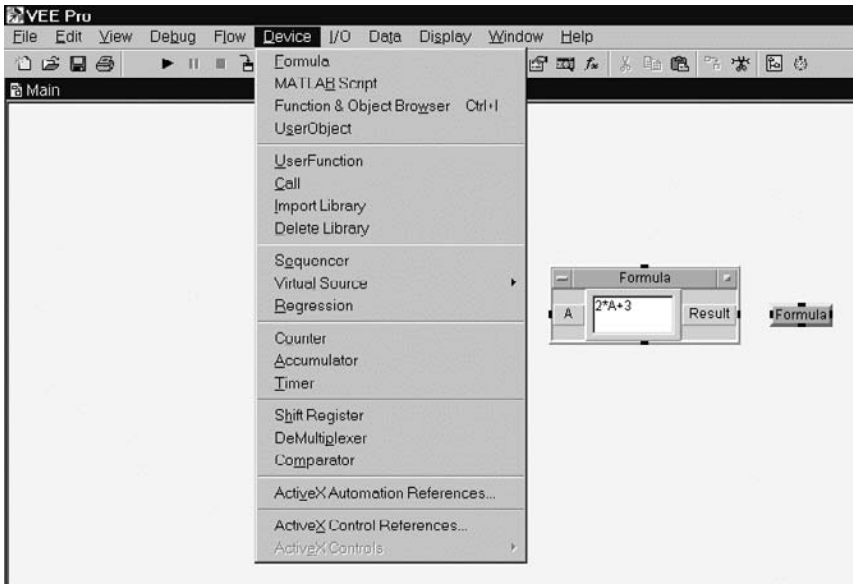


Figure 1.16. The Menu Bar => Device => Formula

The Formula Object initially appears with the default formula "2*A+3". This formula can be modified as desired. Its features are selected by clicking the upper left (-) symbol. These features are: Restore, Move, Size, Minimize, Clone, Replace, Help, Properties, Description, Add Terminal, Delete Terminal, and Cut. Terminals may be added or deleted. The input and output terminal default names can be changed by double-clicking on the terminal. The name of the terminal corresponds directly to the variable name in the Formula expression. The input terminal Required Type and Required Shape can also be changed by double-clicking on the terminals.

Menu Bar => Device => Function & Object Browser

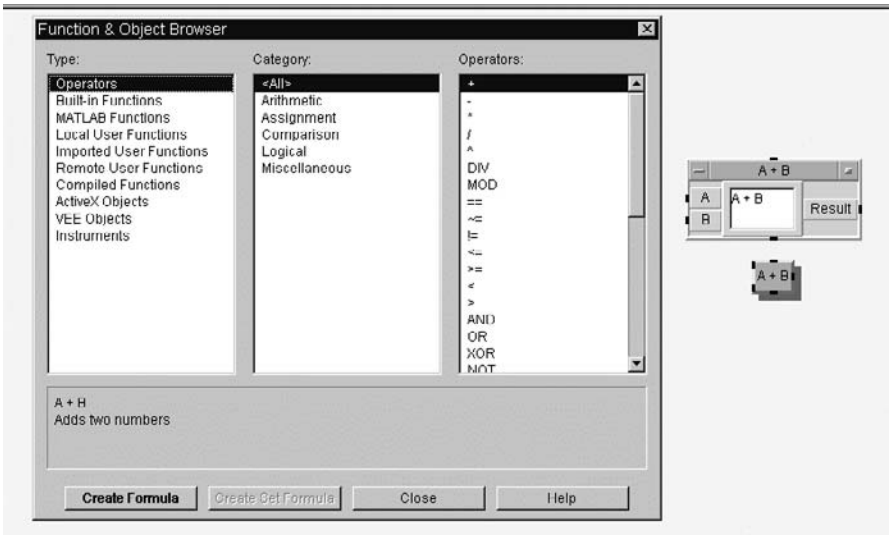


Figure 1.17. The Menu Bar => Device => Function & Object Browser Object

All of the operators available to help build formulas can be accessed via the Type window.

- When Operators is selected under Type, then Category and Operators windows appear.
- When a Function is selected under Type, then Category and Functions windows appear.
- Also, for the ActiveX objects, then Library, Class, and Members appear.
- When "VEE objects" is selected under Type, then Category and Members appear.
- When Instruments is selected under Type, then Instrument Category, Configured Instruments, and Members appear.

There is a gray box across the bottom of Function & Object Browser. Further information regarding each selected item will appear in this box.

> **Note:** "A+B Adds two numbers" is shown above. When Create Formula is clicked, then the Formula Object appears with A+B as its title. The title A+B remains displayed in the Iconic View.

Overall Reminders

Program Explorer

The Program Explorer is a Work Area that requires a significant amount of space on your screen. It is very helpful whenever you want to examine how pieces of your program are connected. The Program Explorer will automatically track your program structure and display it graphically in a hierarchical manner. Double-clicking on a Program Explorer icon will immediately take you to that icon's object. Also, you can provide more room for your Main Work Area:

Menu Bar => Select View => Program Explorer
There is a checkmark next to Program Explorer. Program Explorer can be toggled to select or deselect it. Observe the effect on the screen display. Similarly, you can select and deselect the Tool Bar and the Status Bar. Also, the width of the Program Explorer can be modified by holding the mouse over the right-hand border of the Program Explorer window. It will change into a "splitter" cursor. Drag this cursor to change the Program Explorer width.

Cloning Objects
Clicking on the upper left (-) symbol of an object will display a pull-down menu that includes Clone. Cloning an object causes an exact duplicate of the original object. Its title and parameters can be changed once it is placed in the Work Area. Cloning a given object may be repeated. Cloning large groups of objects is not recommended. See the topic: UserFunctions.

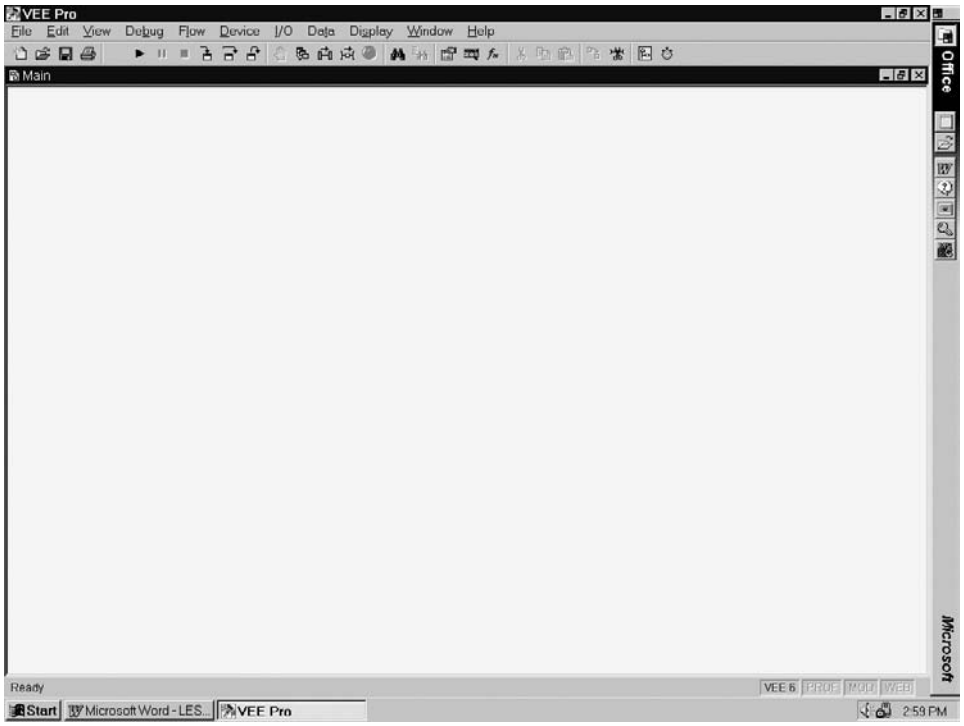
Open View versus Iconic View
The Open View of an object displays all its available and controllable parameters and terminals. The Iconic View provides only a visual description and the name of that object. However, all terminals are accessible and can be connected. Closing an object requires clicking on its upper-right (•) symbol. Opening an object requires double-clicking on any portion of that object. Program development is usually performed in the Open View; programs are usually run in the Iconic View.

Program Development
Program Development is performed by connecting the appropriate objects in the Work Area. The techniques and procedures are described in each lab. A cross-referenced index is given in Appendix B.

Run-Time Errors
After the program is developed, it is run. A VEE Run Time Error occurs when a program is prepared that will not operate properly. Errors are noted after the Run arrow (sixth button from left on Tool Bar) is clicked and an error box appears. The VEE Run Time Error box indicates the type of error and where it occurred. That object will have a red border. The Go To button To indicates which object is at fault. The Help button will provide a description of the error; it is not always helpful! The accessed Help dialog box may be printed if desired. The last error can also be retrieved from the Menu Bar => View => Last Error menu selection.

> **Note:** Click on the black square in the Tool Bar to stop the program. Corrections to the program may now be performed.

Lesson 2 Pre-lab



The VEE Pro Opening Development Screen

Figure 2.1. The VEE Pro Development Screen

Bitmap Transferring
Bitmaps in the form of pictures (art) may be converted from VEE Pro to other Windows applications, via the clipboard. The applications must support Bitmap. Microsoft Word™ supports Bitmap. (The clipboard cannot be viewed.) Its applications will be presented later. To copy an object: select one or more objects, choose Copy from the Edit menu, switch to the other application, and choose Paste in the Edit menu.

Other ways to generate bitmaps from VEE Pro include the little savePanelImage (function) and the Web Server capability. See the VEE Pro documentation for details.

Menu Bar => Device => Function & Object Browser
The categories Function & Object Browser (Random):
Select the titles of interest. As an example, click, in sequence:

- Type – Built-in Functions
- Category – <All>
- Function – random

Click on Create Formula; the object "random(low,high)" will appear containing two input terminals and one output terminal. Examine Help if you want to learn more.

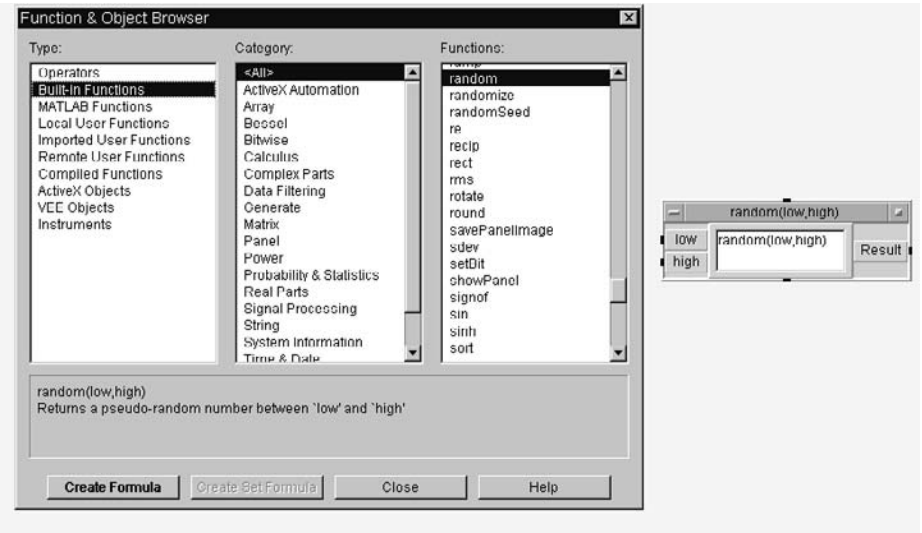


Figure 2.2. Function & Object Browser "Random" Object

Menu Bar => Data => Constant => UInt8
UInt8 is an object that provides a one-byte, unsigned (U) integer scalar – an absolute value without sign. For this lesson, enter the desired integer (0 or 1) within the object white space (editing area).

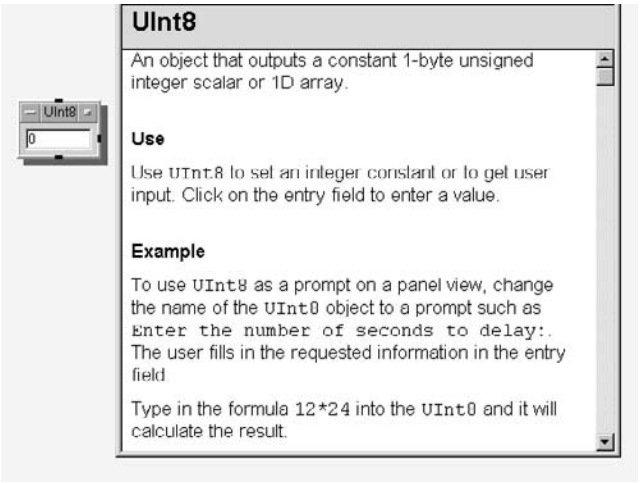


Figure 2.3. Data => Constant => UInt8

The other constant integer objects are explained below.

Object Title Bar

Any object title-bar name (which is in the center of the title bar) may be changed so its title is more descriptive. Double-click on the object title bar; the object Properties box will appear with its existing title highlighted. Type in the desired title; click OK.

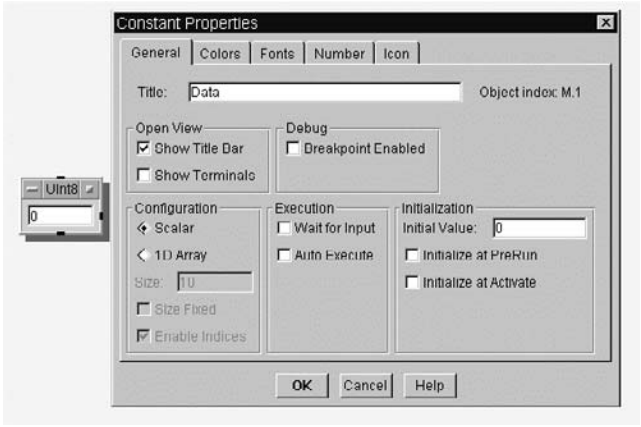


Figure 2.4. Changing object title name

Main Title Bar; Enter Description

Click on the Main title bar with the right-hand mouse button; a pull-down menu will appear; select "Description"; a dialog box – Description of "Main" – will appear. Enter the information that you wish to appear in the documentation of that program.

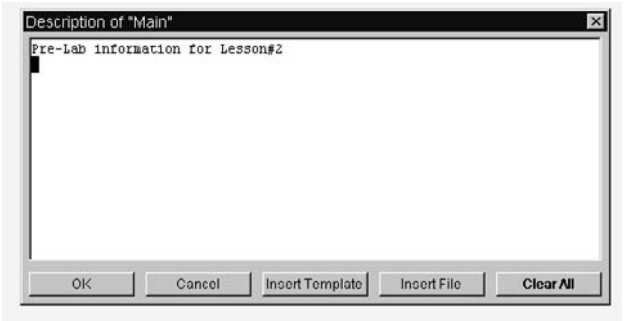


Figure 2.5. Adding Main Title Bar description

Program Documentation
Select Menu Bar => File => Save Documentation. The ".vee" extension will change to a ".txt" extension on the end of the program title. The program documentation is a description of the combined program elements. VEE Pro steps through the program components and prints the properties of each object.

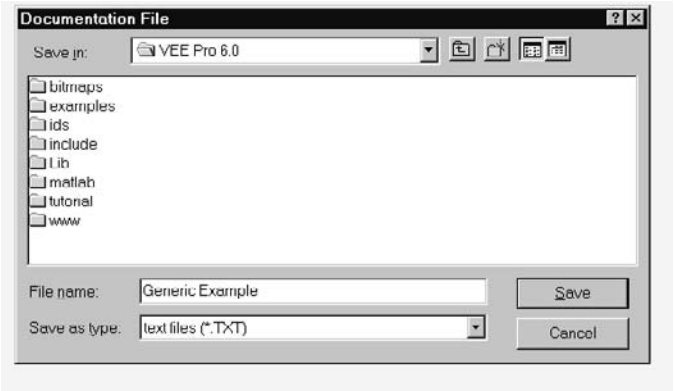


Figure 2.6. "Save Documentation File" window

Menu Bar => Data => Continuous => Real64 Slider (or Knob)
Select Menu Bar => Data => Continuous => Real64 Slider.

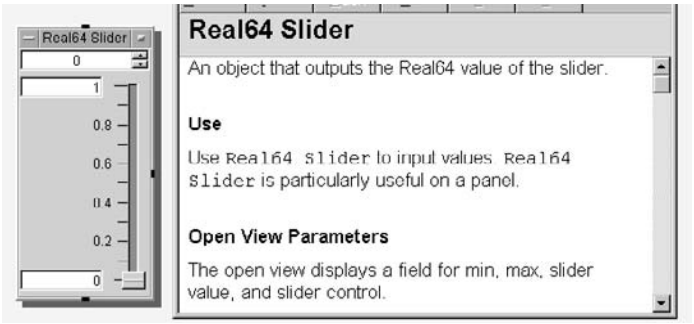


Figure 2.7. Real Slider Continuous-Data Object

Select Menu Bar => Data => Continuous => Real64 Knob.

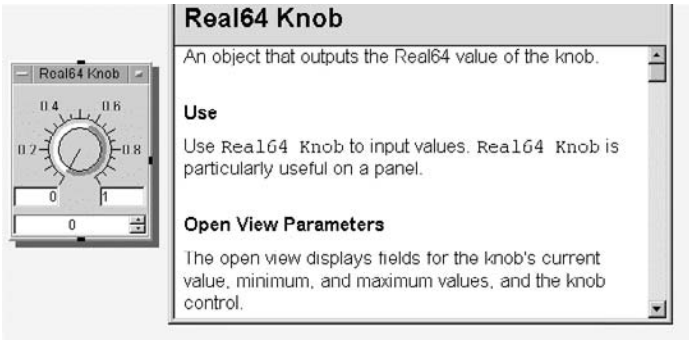


Figure 2.8. Knob Continuous-Data Object

Menu Bar => Data => Dialog Box => Int32 Input
Select Menu Bar => Data => Dialog Box => Int32 Input.
The Prompt/Label field typically contains “Enter Integer Value”; change this text if desired.

Enter a Default Value that is within the Value Constraint range; otherwise, an error message will appear. The Error Message box constraints should agree with the Value Constraint numbers.

There are two types of dialog boxes: the Modal dialog box will cause a program “pause”. A UserObject or UserFunction can be used to create a Non-Modal dialog box (with a panel) that will allow the program to continue to run.

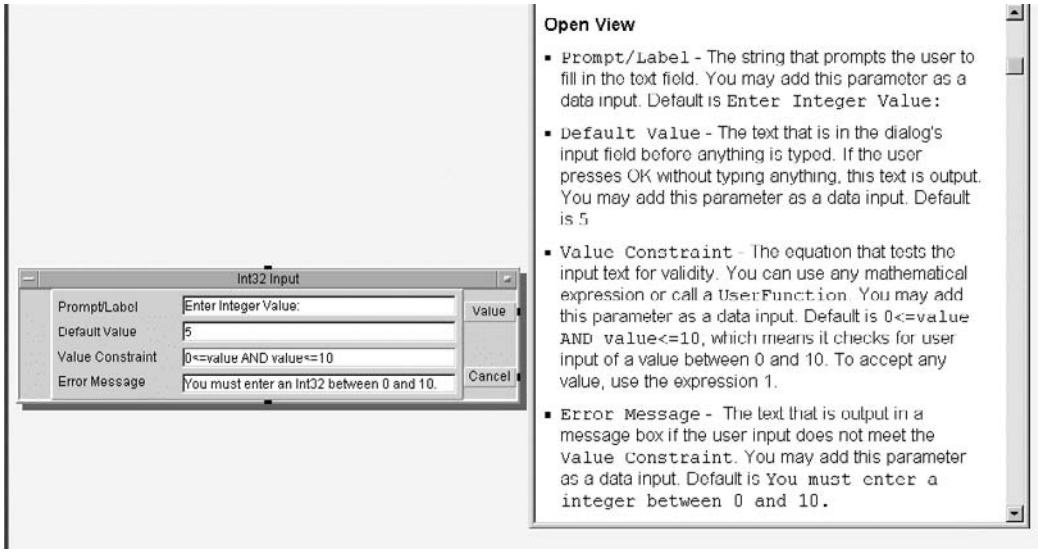


Figure 2.9. Data => Dialog Box => Int32 input and descriptors

Menu Bar => I/O => To => File
Select Menu Bar => I/O => To => File.
Program results may be stored by using the ToFile Object; this object provides a location where the program data (results) are stored. Input terminals can be added to send the specific data to a file; transactions can be added by double-clicking the (<Double-click here>". Data are taken from the input-terminal pins (when they exist) and written to the specified file via transactions inside the object.

The data-location name "myFile" should be changed to be more specific so it will identify the stored data location. When it is clicked, the query: "Write data into what file?" will appear.

- That file-location name may be chosen from the provided list; otherwise, a new file location may be entered.
- If that file-location name already exists, then it will be highlighted when the "new" name is typed into the editing area.
- If that new-location file name is to be added to the list, then click OK on the dialog box.

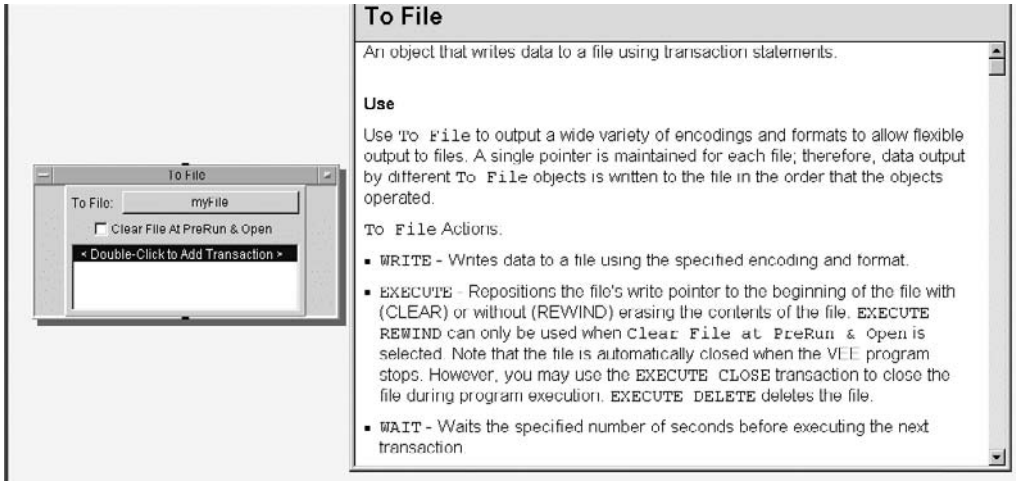


Figure 2.10. I/O => To => File Information (partial)

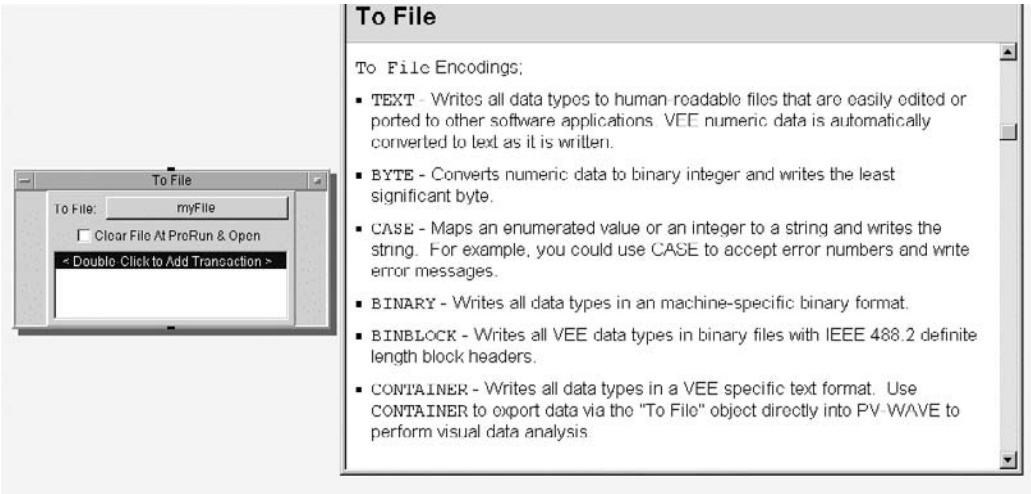


Figure 2.11. I/O => To => File Information (continued)

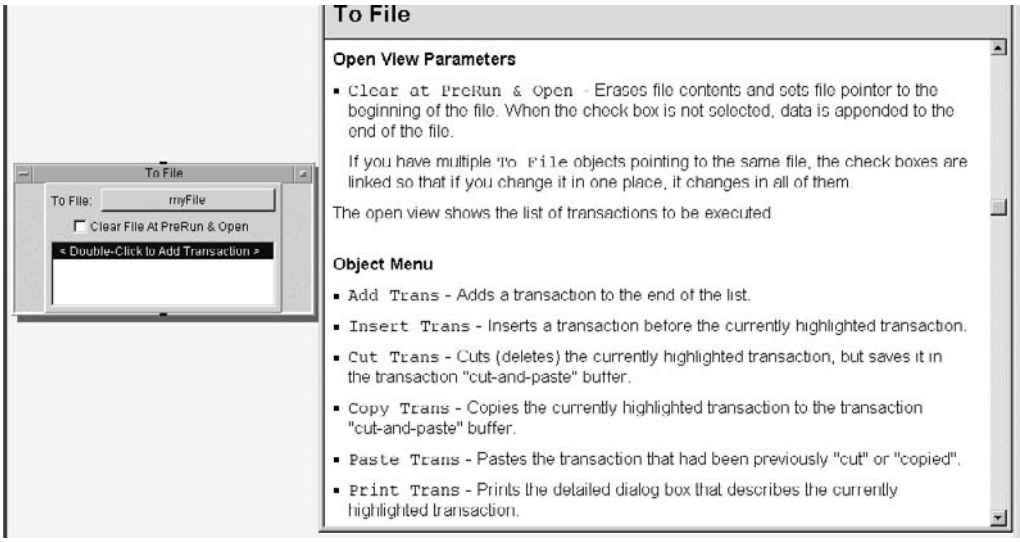


Figure 2.12. I/O => To => File Information (final)

Creating an Operator Interface

An Operator Interface can be created from an existing program. The Operator Interface allows any operator to run a program without being able to change the connections and any other parameters that are to be unavailable to the operator.

The Operator Interface is created by first highlighting those objects (with the Ctrl key) to be included in the operator's Panel View. Then the mouse is placed on the Main menu white space and the mouse right-hand button is depressed. "Add to Panel" is selected. An Operator Interface is then created. Only those objects are displayed that can be modified; no connections are shown.

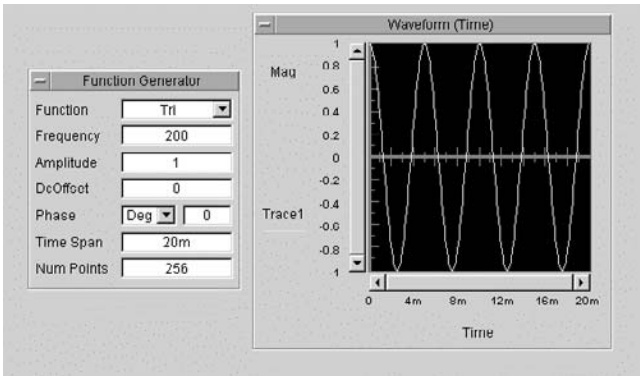


Figure 2.13. Panel View of an Operator Interface

In this example, the Function Generator is set to the Tri function which the operator can change. Once "run" is depressed, the waveform will change from the displayed cosine to the newly selected Triangle function.

Switching between the Detail View and the Panel View

The Main title bar will display two rectangles on its left-hand side. Touch the left rectangle to access the Panel View; touch the right rectangle to access the Detail View. See Figure 2.14.

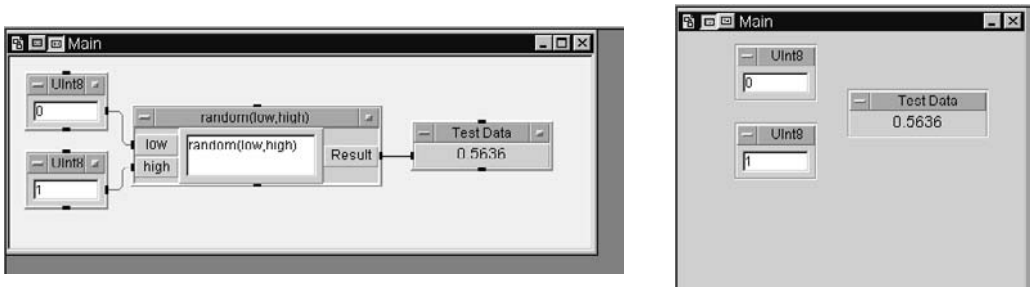


Figure 2.14. Switching between Detail View and Panel View

Changing colors and fonts on Panel

Return to Panel View. From the Object Menu, (the left object on the gray Panel View Title Bar), select Properties, then Color. Click on the rectangular button to the right of "Background". Select the color you desire. See Figure 2.15.

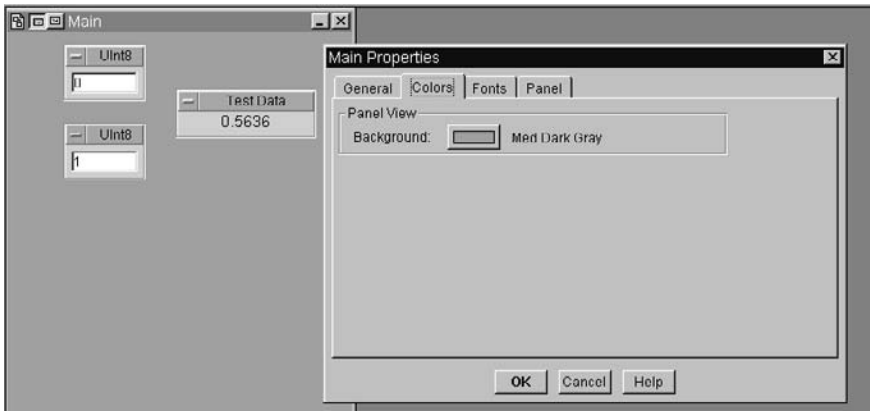


Figure 2.15. Changing Panel-View colors

Changing colors and fonts on Objects

Double-click on a UserObject Title Bar; the Properties box will appear. Click on Colors; your choices will include

- Panel View Background,
- Pop-Up Panel Title Background, and
- Pop-Up Panel Title Text.

This procedure is similar to "Changing colors and fonts on Panel" above. Color-coordination information may be useful in visually "connecting" a variety of related objects.

Menu Bar = > I/O => To => File

The ToFile Object provides access to storing program data. The data are accessed by double-clicking on the ToFile "A" input terminal. See Figure 2.16.

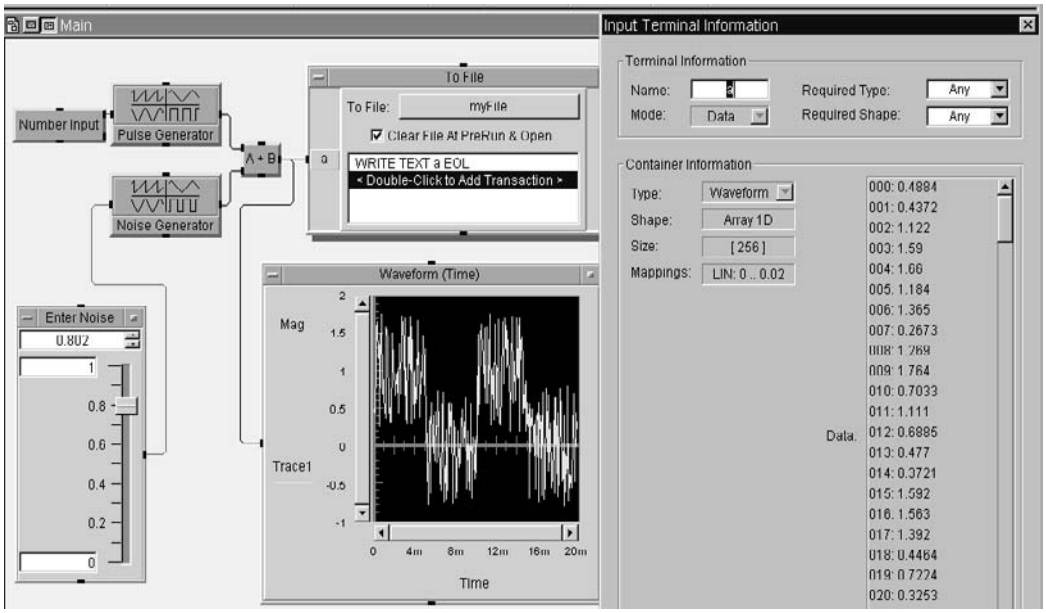


Figure 2.16. Example of ToFile contents

Menu Bar = > I/O => From => File

Select Menu Bar = > I/O => From => File. The FromFile Object provides access to processed program data. After the program is run, the data are accessed by double-clicking on the FromFile "X" output terminal. See Figure 2.17.

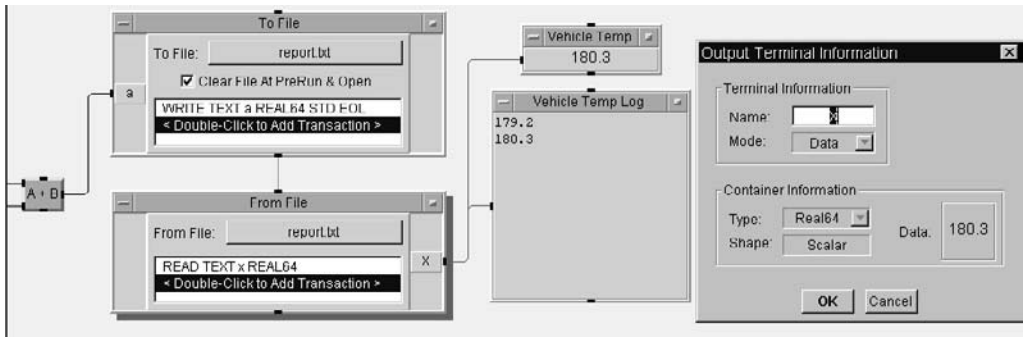


Figure 2.17. Example of FromFile content

Alphanumeric Display Selection
The Alphanumeric Display Object can display text or numbers. The numbers must be in scalar (a data shape that contains a single value), one-dimensional array, or two-dimensional array format. The Alphanumeric Display Object will display only the last data sent via the input terminal.

A logging alphanumeric display will provide displays as text or numbers when a test is repeated more than once – when data is sent to the object multiple times, such as in a loop. The logging alphanumeric input data must be in scalar or in one-dimensional array shape.

Print Screen
Select Menu Bar => File => Print Screen. A dialog box will appear from which the selected printer and number of copies can be selected. See Figure 2.18.

Note: VEE Pro will not allow a "Print Screen" if another Modal dialog box is already open.

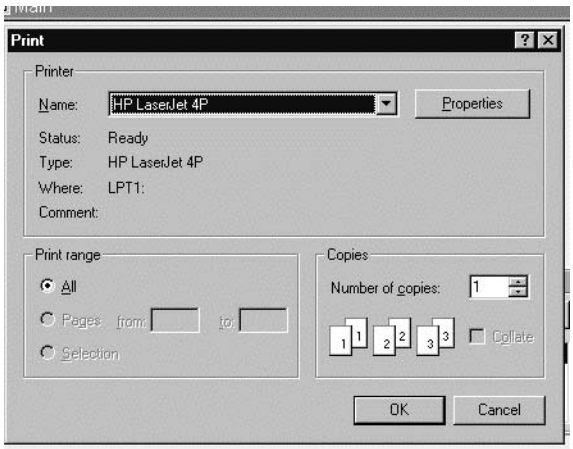


Figure 2.18. The Print Screen Dialog Box

Deleting unneeded objects and connecting lines
Two types of scissors are available from the Menu Bar. See Figure 2.19. The left scissors are used to cut unwanted objects whenever that object is highlighted. Any connecting lines are also deleted. The right scissors are used to cut connecting lines between objects.

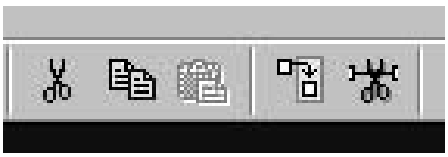


Figure 2.19. The Two VEE Pro Scissors

Deleting unneeded terminals

Input and output terminals (pins) may be deleted. Click on an object upper-left (-) square. As an example, select Delete Terminal; then select the type of (input) terminal you wish to delete. That terminal will be automatically deleted, including its title. See Figure 2.20.

Note: A shortcut for deleting a terminal – hold the cursor over the terminal and depress Ctrl + D.

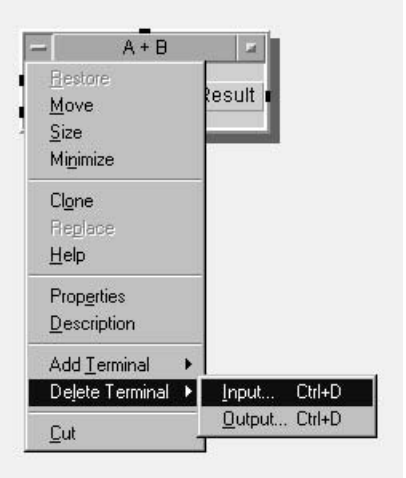


Figure 2.20. Deleting an Object terminal

Open or Save a File

As an example, select Menu Bar => File => Open. A File Directory dialog box (Open File) will appear with a list of files that are available to be opened.

Note: It may be necessary to search the hierarchy of file folders to locate the desired file. This approach is identical to that used in any PC-Windows application.

Save File works in a similar manner:

Menu Bar => File => Save

If a new name is to be given the file, then select:

Menu Bar => File => Save As….

Lesson 3 Pre-lab

Controlling and configuring instruments
There are many ways to control and configure instruments:

- Panel Drivers provide a simple user interface (or "front panel") to control instruments from the computer screen. Changing parameters in the VEE Pro panel driver Front Panel on the screen changes the corresponding state of the physical instrument. Component drivers are a subset of panel drivers.
- Direct I/O Object allows for transmitting of commands and reception of data over a variety of supported interfaces.
- VXI*plug&play* drivers control instruments over VXI, GPIB, and other interfaces. The VXI*plug&play* standard specifies how the drivers are written by the instrument vendor. Therefore, their "look and feel" is similar.

For more information regarding controlling instruments, see the Lesson 4 Pre-lab below.

Menu Bar => I/O => Instrument Manager…
The Instrument Manager allows adding one or more instruments and devices to your VEE Pro program. Properties of the instruments are set via the Properties button on the right side under "Instrument". Click "Add" under Instrument… . This displays the Device Configuration Dialog Box, which presents the following fields. See Figure 3.1.

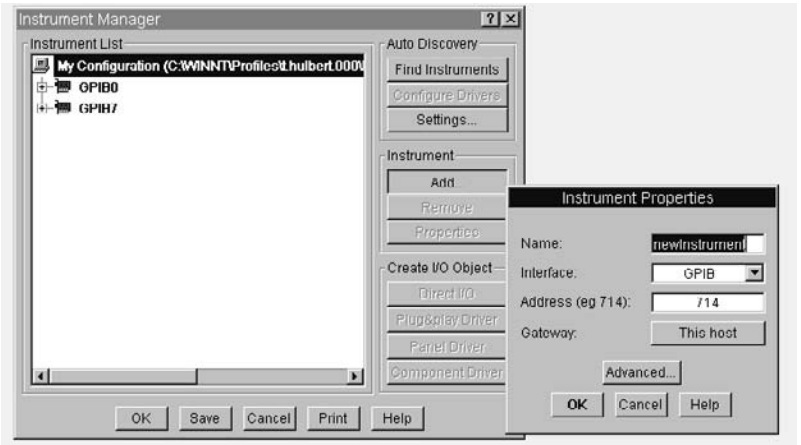


Figure 3.1. Instrument Manager Dialog Box with (Add) Instrument Properties

This configuration of instruments and settings is stored in a file (or can be saved within each .vee program). My Configuration shows the path where the file is saved. Configurations for instruments can be added for both real (physical) instruments and for instruments that you may not have connected live at this time. The non-connected instruments can be configured as "NOT LIVE". This allows you to create your configuration and write your program. NOT LIVE instruments are usually given an address or zero.

Under Instruments, Add…, four fields are given:

- Name (of instrument),
- Interface (to reach either the GPIB, Serial, GPIO, or VXI bus),
- Address (where **0** represents the address of an instrument not present (NOT LIVE) and **7xx** represents the address of the GPIB bus),
- Gateway: ("This host" is used to control instruments locally)

The Advanced Instrument Properties dialog box allows for the selection of Direct I/O, *Plug&Play* Driver, and Panel Driver configurations.

Menu Bar => Flow => Start
The Start button activates the program. It is an alternative to the Run button when you are writing small prototype programs. It is not recommended for large programs and runtime versions. The Start button can only be moved by right-clicking on the Start-button Object.

Menu Bar => Flow => Do
The "Do" Object creates a branch point to control the flow of the execution of a thread.

Menu Bar => Flow => Repeat => For Count
For Count is an object that activates data being processed a specified number of times. The output of For Count can be used as a value to activate succeeding objects. The three objects are shown in Figure 3.2.

Figure 3.2. Three "Flow" objects

Menu Bar => Data => Collector
Collector is used to create a one-dimensional array from scalar input data. It can create an output array from collected input arrays. When all data have been gathered, the XEQ pin informs the output that the collected array of data is available. The Collector has two inputs: one for input (Data), the other is an output trigger (XEQ).

Output Shape There are two selections. See Figure 3.3.

- 1 Dim Array - The output signal shape, regardless of the input signal, is a 1D Array.
- n+1 Dim Array - The input signals (arrays of n dimensions) are collected; the output is an array of n+1 dimensions.

Figure 3.3. The Collector Object; both settings

Menu Bar => Device => Timer
An object whose output is the difference (in seconds) between its activation time (top-left pin) and its completion time (bottom-left pin). The elapsed time is indicated in its display space. See Figure 3.4.

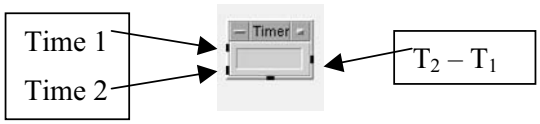


Figure 3.4. The Timer Object

Menu Bar => Device => UserObject
The UserObject initially appears as a closed Iconic View within Main or any other window. It is opened by double-clicking on its icon. The UserObject can contain one or more interconnected objects that can be saved as a library for future use. See Figure 3.5 for its three possible locations (views).

Figure 3.5. Three views of UserObject

Its title can be changed by double-clicking on its top horizontal bar (below the Tool Bar). Its Trig Mode, can be changed; its Colors and Fonts can be displayed.

UserObject features are selected by clicking on the (**-**) symbol in its upper-left corner. These features are: Restore, Move, Size, Minimize, Maximize, Cut, Copy, Clone, Help, Properties, Description, Add Terminal, Delete Terminal, Make UserFunction, Unpack, Save Secured Version, Find, Print, Locate, Calls, Create Panel, Delete Panel, and Close. The names for each added terminal may be changed by double-clicking on that terminal.

Installing your own program within a UserObject is known as encapsulation. Installing one UserObject within another is known as nesting. These are the building blocks of your more sophisticated programs; it is known as achieving "top-down design" by programmers.

Rather than cloning a UserObject, it is preferable to apply a UserFunction that can be called and re-used in multiple locations for any of your programs. The UserFunction is explained below.

Lesson 4 Pre-Lab

Additional information and updates are available via the Agilent Web site at http://www.agilent.com/find/vee.

The telephone contacts for support are:
In the US, 800-452-4844.
In Canada, 877-894-4414
Outside the US and Canada, contact your country's Agilent support organization.

A list of contact information for other countries is available on the Agilent website:
http://www.agilent.com/find/assist

Download the Latest Version of the Agilent IO Libraries from:
http://www.agilent.com/find/iolib

"Hotfixes" must be obtained from Microsoft because Microsoft does not allow Agilent to distribute Windows 2000 hotfixes.

Instrument Control Object
The VEE Pro User's Guide section entitled: Easy Ways to Control Instruments contains extensive, detailed descriptions regarding configuring and controlling instruments.

Instrument Control => Object Menu => Add Terminal => Control
The VEE Pro Object Control adds an asynchronous control input terminal to an object. Use Control Input to add asynchronous inputs, such as Reset or Clear, to an object. After selecting Control Input, choose the control to add information from a dialog box. See Figure 4.1.

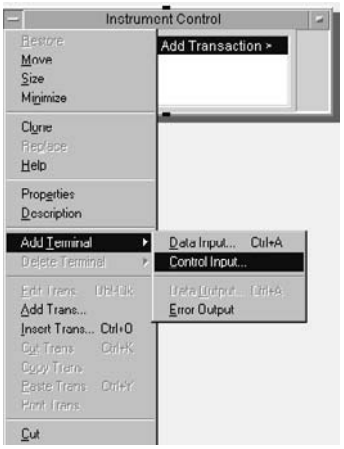


Figure 4.1. Accessing the object control input terminal

Activation of a control input pin does not cause the object to operate; it only forces a particular action to happen, such as Clear.

Instrument I/O Data Types

VEE Pro provides many data types used for instrument I/O. It manipulates and stores all integer values as Int32 data types and all real numbers as Real64 data types. However, instruments generally support 16-bit integers or 8-bit bytes; some instruments support 32-bit real data. VEE Pro provides the following data types for general programming in the Menu Bar => Data => Constant and Instrument Manager "instrument I/O". See Table 4.1.

Table 4.1. VEE Pro Instrument Data Types

VEE Data Type Constant	Description
Byte	An 8-bit two's complement byte (-128 to +127). Byte is used to READ BINARY, WRITE BINARY, and WRITE BYTE instrument I/O instructions only. UInt8 is the default data type.
Int16	A 16-bit two's complement integer (-32768 to +32767).
Int32	A constant 32-bit integer scalar or ID Array. As an example: Use Int32 to set an integer constant or to get a user input. If Global Format is off, specify the number format for integer numbers to be displayed on this object. The choices are Decimal (e.g. 42), Octal (e.g. #Q52), Hexadecimal (e.g. #H2a), and Binary (e.g. #B00101010). Default is Decimal. The Default Value input allows a change in the current value(s). This input requires an integer value, either a Scalar or an Array 1D. The Constant is reconfigured to match the shape of the incoming data. The Initial Value field is a scalar, even if Int32 is configured to be an array.
Real32	A 32-bit real number that conforms to the IEEE 754 standard: (+/-3.40282347 E+/-38)
Real64	A 64-bit real number. The Global Format specifies the number format for real numbers to be displayed. Typical choices are Fixed (98.6000), Scientific (9.8600E+01), Engineering (98.6000E0), Standard (98.6), and Time Stamp (Wed 01/Jan/2003 17:01:38). Default is Standard.
UInt8	A constant 1-byte unsigned integer, scalar or 1D array. As an example: Use UInt8 as a prompt on a panel view, change the name of the UInt8 Object to a prompt such as "Enter the number of seconds to delay:". If Global Format is off, specify the number format for integer numbers to be displayed. The choices are Decimal (42), Octal (#Q52), Hexadecimal (#H2a), and Binary (#B00101010). Default is Decimal. The Initial Value field is always a scalar, even if UInt8 is configured to be an array.

Notes for Table 4.1:

Int32:

Wait for Input – When the check box is selected, the object does not operate until the object's value is changed. Thus, the constant acts like a "Confirm" button. When the check box is not selected, the object operates as soon as any constraints are met, and the currently selected value is the output. Default is off.

Auto Execute – When the check box is selected, the object operates and starts the rest of the program when a value is selected. Thus, the constant acts like a "Start" button. If the program is already running, Auto Execute is ignored. Default is off. Auto Execute is not recommended for large programs.

UInt8:

Initialize is most often used for initializing values inside a UserObject or UserFunction. The other method for setting initial values is the Default Value control pin available on most data constants. The Default Value input allows you to change the current value(s). This input requires an integer value, either a Scalar or an Array 1D. The Constant is reconfigured to match the shape of the incoming data.

Common Instrument Data Type Characteristics

The following Menu Bar => Data => Constant objects have common characteristics, including: Number Format, Default Value Control Input, Wait for Input, Auto Execute, and Initialize at Prerun & Activate. Data constants only support scalar and 1D arrays. VEE Pro usually supports arrays up to ten dimensions. Use Alloc Array or Formula [I.a.b] type of syntax to create larger dimensional arrays that are greater than ten dimensions.

Note: Instruments may require ASCII, binary Real32, and binary Int32. Binary will run programs much faster than ASCII. VXIplug&play requires memory allocation for direct I/O; there is a tab on each *plug&play* driver panel to allocate memory.

Installing Hardware

Consult the Agilent Web site to determine the latest information regarding level of Windows software support for your hardware installation. Also, before you install your IO Libraries software, you should install any new hardware in your computer. After the hardware is installed, turn the computer on and allow it to boot. If you are adding plug&play hardware, when you log on, Windows may display a Wizard to guide you through the process of installing drivers for new hardware. Hardware examples include:

82350 PCI GPIB card

The 82350A card is self-configuring. It will “Plug & Play”. After installing this card into the PCI or ISA slot, the I/O libraries are then installed. “Automatically Configure” is selected following software installation.

E8491 VXI interface

If you install a 1394 OHCI card and want to configure an E8491, you should install the 1394 card in the computer, connect the 1394 cable between the card and the E8491 and power up the VXI card cage before booting the computer. Below are the configuration notes. For information on Windows XP, consult the Agilent Web site.

Configuration notes:

1. If, on Windows 98 or Window 2000, you attempt to configure an E8491 card but none were found on your system, verify that all your VXI mainframes are powered on and that your 1394 cables are connected before running IO Config. If you are still having trouble, refer to the E8491 Troubleshooting Guide.
2. Windows 2000 support for the following interfaces has been added: E8491 IEEE 1394 to VXI interface 82341C and 82350 GPIB Cards.
3. Fixes: In certain topologies, the E8491 hardware is not properly identified on the 1394 bus. This will result in inconsistent behavior when the VXI resource manager tries to configure instruments in the VXI card cages. Behavior may include timeout errors while attempting to communicate with a particular instrument, and/or instruments appearing in cages where they are not actually located. Try reversing the cable connections to the ports on the host interface

card. In general, cable connections with longer chains of E8491's should be on lower host ports. Don't re-run setup too quickly. After the IO Libraries Setup program exits, it takes several seconds to clean up and unload itself from memory. If Setup is re-run before this process is complete, you may see an error message or find that the re-installation does not behave as expected.

Windows 2000

OHCI Card Driver: If the 'Found New Hardware Wizard' appears and prompts you to install the driver for a 'Texas Instruments OHCI Compliant IEEE 1394 Host Controller', you will need to get and install the drivers for this card from the Windows 2000 CD that came with your computer. These drivers may already be installed and in that case, the 'Found New Hardware Wizard' for this device will not appear. E8491 Driver: If the 'Found New Hardware Wizard' appears and prompts you to install the driver for a 'Hewlett-Packard HP8491A', you can find the driver for this device (1394ipt.inf) in the 'Windows 2000' subdirectory of the Agilent IO Libraries CD.

Windows NT 4.0

Because Windows NT is not a Plug&Play OS, all necessary driver installation and configuration is performed when you install the IO Libraries.

Windows 98

OHCI Card Driver: If the 'Add New Hardware Wizard' appears and prompts you to install the driver for a 'Texas Instruments OHCI Compliant IEEE 1394 Host Controller', you will need to get and install the drivers for this card from the Windows 98 CD that came with your computer. E8491 Driver: If the 'Add New Hardware Wizard' appears and prompts you to install the driver for a 'Hewlett-Packard HP8491A', you can find the driver for this device (1394ipt.inf) in the 'Windows 2000' subdirectory of the Agilent IO Libraries CD.

82335B Card

This real-instrument interface card is difficult to configure. If this card must be used, install the card "as is" and then contact Agilent technical support for real-time troubleshooting assistance.

82341D Card

This real-instrument interface card is self-configuring. It will "Plug & Play". After installing this card into the PCI or ISA slot, the I/O libraries are then installed. "Automatically Configure" is selected following software installation.

Note: Some personal computers have 32-bit EISA backplanes. If you installed an Agilent 82340 or 82341 GPIB interface in an EISA slot, you may wish to run the EISA configuration utility provided with your computer. The EISA configuration utility will assign hardware resources to the interface to avoid system conflicts. When you ran the Agilent IO Libraries installation Setup program, the EISA .CFG files were placed in the EISACFG directory under the SICL base directory (for example, under C:\SICL95\EISACFG or C:\SICLNT\EISACFG if you installed the SICL software in the default location). When you run the EISA configuration utility and assign resources, be sure to use the same settings as used with the Agilent IO Libraries configuration utility (IO Config).

RS 232 Serial Ports: The following are formatted for RS-232 Serial Ports:

E2050 LAN to GPIB

E2070 GPIO Cards

SICL and VISA LAN Client

SICL LAN Server

E8491 Support on Windows 98 has been added.

Installing IO Libraries and Configuring Interfaces
Refer to Table 4.2 and “Agilent IO Libraries Installation and Configuration Guide for Windows”. Also, refer to the Agilent web page for the latest installation information.

Table 4.2. Agilent IO Library Versions

Version Installed	Action	First See “Getting Started”
None	Install a new version of the Agilent IO library	Installing new IO libraries
H.01.03 or earlier	Upgrade existing IO libraries to the latest version	Upgrading existing IO libraries
J.01.00 or later	Modify, repair, or remove the same version of the existing IO libraries	Maintaining existing IO libraries

After installing any hardware and drivers needed for IO, you should then install the Agilent IO Libraries by running 'Setup' from the Agilent IO Libraries CD. Before an interface can be used with SICL or Agilent VISA, IO Libraries must configure the interface. This is normally done automatically during the installation or it can be done manually using the 'IO Config' utility located in the 'Agilent IO Libraries' program folder.
The I/O Configuration program contains the following 'Options' selections in its menu:

VISA Logging
Off (Default)
Message Viewer: Log to the Message Viewer application if available.
Event Viewer: Debug Window Log using Windows OutputDebugString(). (This is useful in debuggers.)
SICL Yield (Windows 98)
False (Default): SICL does not yield the CPU during I/O.
SICL Yield must be 'False' for proper operation of some VXI*plug&play* drivers. This will prevent a potential deadlock from within a Windows message handler.
True: SICL yields the CPU during I/O.

The following I/O interfaces are supported for 32-bit applications; 16-bit applications are not supported:

Agilent E8491 IEEE 1394 ("Fire Wire") to VXI Interface
SICL and VISA
Agilent E985x VXI Pentium (R) Controller (VXI and GPIB)
SICL and VISA
Agilent E623x VXI Pentium (R) Controller (VXI and GPIB)
SICL and VISA
Agilent 82340 GPIB Card (ISA) &
Agilent 82341A GPIB Card (ISA) &
Agilent 82341B GPIB Card (ISA) &
Agilent 82341C GPIB Card (ISA) &
Agilent 82350 GPIB Card (PCI)
SICL and VISA
For Agilent USB to GPIB interface cards, see www.agilent.com

Built-in LAN Interface, client and server
SICL and VISA
VISA GPIB-VXI functionality (VXI communications from a GPIB interface)
VISA only
Support for command modules from many vendors.

Using Device Symbolic Names in SICL

This version of SICL supports device symbolic names. You can assign a symbolic name to a device and then use that name in place of the device's address when making an iopen() call. The advantage of doing this is that you don't have to hard-code device addresses in your programs.

Device symbolic names can be defined by editing the registry. First, run the 'regedit' utility. Select 'HKEY_LOCAL_MACHINE', then 'Software', 'Agilent', 'IO Libraries', 'CurrentVersion' (Note: if you have upgraded from an earlier version, it may be located at 'HKEY_LOCAL_MACHINE', then 'Software', 'Agilent', 'SICL', 'CurrentVersion'). Under 'CurrentVersion', add a new key called 'Devices'. This is accomplished by selecting Edit then New then Key from the registry entry's menu. For each symbolic name you want to define, add a new string value in the 'Devices' key. This is accomplished by highlighting the Devices entry and then selecting Edit then New then String Value from the menu. Enter the symbolic name in the name field. Edit the value by double clicking on the name field. Enter the value without quote marks. For example, the following 'Devices' entries would result in two device symbolic names being defined - dmm refers to a device at hpib7,0 and scope refers to a device at gpib7,9.

Name	Data
Dmm	"gpib7,0"
Scope	"gpib7,9"

After adding these entries to the registry, a program could open sessions to the 2 devices by using iopen("dmm") and iopen("scope") instead of iopen("gpib7,0") and iopen("gpib7,9"). Agilent does not recommend that users modify the registry because of its accumulative effect on other software that is contained in the Windows environment.

Tables 4.3 and 4.4 provide related documentation for programming in SICL and VISA.

Table 4.3. SICL documentation

Document	Description
Agilent SICL User's Guide for Windows	Using Agilent SICL and its language reference
SICL Online Help	Windows help information
SICL Example Programs	Online example application programs
VXIbus Consortium Specifications (when using VISA over LAN)	TCP/IP Instrument Protocol and VXIbus Specifications VXI-11 and 11.1 TCP/IP IEEE 488.1 Interface Spec: VXI-11.2 and IEEE 488.2 Instrument Interface Spec: VXI-11.3

Table 4.4. VISA Documentation

Document	Description
Agilent VISA User's Guide	Using Agilent VISA and its language reference
VISA Online Help	Windows help information
VISA Example Programs	Online example application programs
VXI *Plug&Play* System Alliance VISA Library Specification 4.3	Specifications for VISA
IEEE Standard Codes, Formats, Protocols, and Common Commands	ANSI/IEEE Standard 488.2-1992
VXIbus Consortium Specifications (when using VISA over LAN)	TCP/IP Instrument Protocol and VXIbus Specifications VXI-11 and 11.1 TCP/IP IEEE 488.1 Interface Spec: VXI-11.2 and IEEE 488.2 Instrument Interface Spec: VXI-11.3

Additional Visual BASIC Programming Notes

You can now pass any data type instead of just strings. This allows you to read and write binary data with Visual BASIC for the following commands: iread, Ifread, Iwrite, Ifwrite, iswap, ileswap, and ibeswap.

The isetbuf function is now supported with Visual BASIC.

The igetlulist function is not implemented. It currently returns to Visual BASIC with error I_ERR_NOTIMPL.

The igetaddr function is not implemented. It currently returns to Visual BASIC with error I_ERR_NOTIMPL.

The ivxirminfo function does not correctly return the last two arrays in the structure (int_handler() and interrupter()). All other fields are correctly returned.

The igetluinfo function does not correctly return the hwargs() array. All other fields are correctly returned.

Sample files and a vbreadme.txt file documenting VISA calls from Visual BASIC 4.0 have been included in the following directory (assuming you have installed in the standard location) for Windows NT:
c:\vxipnp\winnt\hpvisa\samples\vb

For all other versions, consult the Agilent Web site.

Warnings

LAN

The LAN server on Windows NT does not start automatically after a reboot as the manual says. You need to either start it manually by double clicking its icon in the HP I/O Libraries folder, or you can create a shortcut to it in your StartUp folder.

VISA viLock()/viUnlock() does not work on devices accessed via LAN. This implementation supports the TCP/IP Instrument Protocol but it is known to differ from the specifications in the following ways:

The device_abort RPC is not supported.

Rules B.4.17 and B.4.18 of VXI-11.1 are not supported by the current implementation. As noted in observation B.5.1 of VXI-11.1, changes are anticipated to VXI-5. Due to these possible changes and the fact that the current revision of VXI-5 does not require that any particular VXI-5 commands be implemented, the current LAN server implementation does not support VXI-5 operations.

Fast Data Channel (FDC)

This version of VISA uses a separate DLL to support the FDC protocol. This DLL is not supplied by Agilent Technologies. Contact the supplier of your FDC capable hardware for information on how to obtain and install the FDC DLL. If you attempt to set FDC-related attributes and the FDC DLL has not been installed, you will receive a VI_ERROR_NSUP_ATTR error.

I-SCPI

I-SCPI was implemented ONLY for BACKWARD COMPATIBILITY and migration purposes. It should not be used for new designs. Additional I-SCPI information is available the file:

c:\vxipnp\winnt\bin\iscpinfo.txt (Windows NT)

or

c:\vxipnp\win95\bin\iscpinfo.txt (Windows 95 and Windows 98)

Beginning with version G.02.00 of the I/O Libraries, the 32 bit I-SCPI drivers have changed from a SICL to a VISA based implementation. This change should be invisible to you. Programs which use I-SCPI should continue to function without modification unless a VISA based I-SCPI driver is not available for an instrument you are using. There will be no new development on the SICL based I-SCPI drivers.

Note that even though the I-SCPI drivers in this implementation are VISA based, I-SCPI may only be used with SICL programs. VISA programs cannot use I-SCPI.

The Agilent E8491 IEEE-1394 to VXI product will only work with the VISA based I-SCPI drivers.

Currently VISA based I-SCPI drivers are not available for the DDCC-1553 and E1450. If you are using I-SCPI with these devices, you will need to use the SICL based I-SCPI drivers which are not available on this release of the I/O Libraries. If you need to use the SICL based I-SCPI drivers, do not upgrade to this version of the I/O Libraries. Version G.02.02 was the last version of the Agilent I/O Libraries to contain support for SICL based I-SCPI.

Lesson 5 Pre-Lab

Statistical Formulas

Statistical formulas are accessed via the Function & Object Browser via

Type: Built-in Functions and

Category: Probability and Statistics.

Statistics Formulas

Maximum – max(x)

The maximum of all the values in "x" where "x" is assumed to contain an array of values.

Minimum – min(x)

The minimum value of all the values in "x".

Mean – mean(x)

$$\text{mean (average)} = \frac{\text{the sum of the value of all readings}}{\text{the number of readings summed}} \qquad \text{eq 5.1}$$

Median – median(x)

median = the center value of all processed points arranged in an ascending sequence. eq 5.2

If there are an odd number of points, then it is the middle value. If there are an even number of points, then it is the mean-average of the two values either side of the center.

Mode – mode(x) = It is the largest quantity of readings that have the same value.

RMS – rms(x)

$$\text{RMS} = \sqrt{\frac{R_1^2 + R_2^2 + R_3^2 + ... + R_n^2}{n}} \qquad \text{eq 5.3}$$

where R is value of the reading

Deviation from the (mean)-average

deviation from the average = (average of all readings) - (each reading value) eq 5.4

It is a measure of the departure of each measurement from the mean-average, noting whether the departure is a positive or negative value. Each deviation must be computed separately.

Average value of a deviation

$$\text{average value of deviation} = \frac{\text{sum of all deviation magnitudes}}{\text{number of deviations summed}} \qquad \text{eq 5.5}$$

Standard deviation – sdev(x)

$$\text{standard deviation} = \sqrt{\frac{\text{sum of all squared reading deviations}}{\text{number of readings less one}}} \qquad \text{eq 5.6}$$

Variance – vari(x)

variance = (standard deviation)^2 eq 5.7

VEE Pro Data Types

VEE Pro includes various types of data in containers to simplify information transfer. Each container holds a type of formatted data. Each type of data has a unique programming task.

The data shape is either a scalar or an array. A scalar is a single alphanumeric value; an array is a collection of identical data types.

VEE Pro provides 14 data types that can perform a wide variety of functions; See Table 5.1.

Table 5.1. VEE Pro data types

Data Type	Description
UInt8	UInt8 is an 8-bit two's complement unsigned integer (0 to 255).
Int16	Int16 is a 16-bit two's complement integer (-32768 to 32767).
Int32	A 32-bit two's complement integer (-2147483648 to 2147483647); a whole number whose decimal equivalent has no decimal portion, such as either 1 or 321.
Real32	A 32-bit real that conforms to the IEEE 754 standard (±3.40282347E±38).
Real64	A 64-bit real that conforms to the IEEE 754 standard; its decimal equivalent (+/- 1.797693138623157 E308) has a decimal portion, such as 3.2, 12.56.
PComplex	A Polar Complex 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 Real. For example, the complex number 1 + 2i is represented by (1,2).
Waveform	A composite data type of time domain values that contains the Real values of evenly-spaced, linear points and the total time span of the waveform assumed to start at time zero. The data shape of a Waveform must be a one-dimensional array.
Spectrum	A composite data type of frequency domain values that contains the PComplex values of points and the frequency start and stop times.
Coord	A composite data type that contains at least two components in the form (x,y,…). Each component is Real64. The data shape must be a Scalar or a 1D Array.
Enum	A group of text strings; each has an associated integer value; the Unum has only one current value. You can access the integer value with the ordinal (x) function.
Text	A string of alphanumeric characters, such as abc123, gef, or 798.
Record	A composite data type with a field for each type. Each field has a name and a container, which can be of any type and shape (including Record).
Object	Used for ActiveX Automation and Controls, a reference to an ActiveX control or a reference returned if from an Automation call. Can also refer to a named VEE object and "get" and "set" properties via the program.

VEE Pro Analysis Capabilities

The math functions are given in the Menu Bar => Device => Function & Object Browser menu: choices for Built-in Functions and Operators. To create your own math functions, you can

- create the function with the Formula Object
- write the function in a compiled language such as C and link it to VEE Pro
- communicate with another software application from VEE, such as MATLAB®

Multiline Formula Techniques

Multiline formulas can be entered into the Formula Object editing area (white space). When explanatory text is desired, then each line of explanatory text must be preceded by two forward slashes (//). The actual instructions are separated with a semicolon (;). See Figure 5.1.

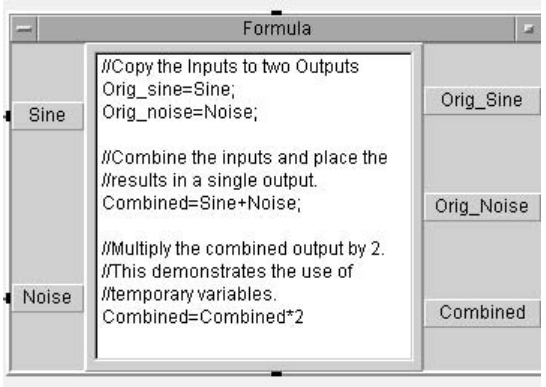


Figure 5.1. A Formula Object containing text and instructions

Operators

Each Operator is accessed via Function & Object Browser. See the Help menus and Appendix C.

Arithmetic: +, -, *, /, ^, DIV, MOD

Assignment: objectvalue=1, a[2]=b, a[2,*,5:6,4]=b, rec.field=b, rec.field[2]=b, rec[2].field=b, globalA=2*b+3, tmp=a, a=b, b=tmp; tmp=a[2], a[2]=a[3], a[3]=tmp

Comparison: ==, ~=, !=, <=, >=, <, >

Logical: AND, OR, XOR, NOT

Miscellaneous: object.value, [1,2,3], [a,b,c], a[2], a[5,3:4,*], rec.field, PI, TRUE, FALSE, (a?b:c)

Built-in Functions

The Function & Object Browser built-in function details are given in their related Help menus and Appendix C. The specific built-in functions are:

ActiveX Automation	Power (Functions)
Array	Probability & Statistics
Bessel	Real Parts
Bitwise	Signal Processing
Calculus	String
Complex Parts	System Information
Data Filtering	Time & Date
Generate	Trig (Functions)
Matrix	Type Conversion
Panel	

Note: All the above functions can be nested.

Display Capabilities

There are fourteen display capabilities accessed via Menu Bar => Display =>

- AlphaNumeric
- Logging AlphaNumeric
- Indicator => Meter, Thermometer, Fill Bar, Tank, Color Alarm
- XY Trace
- Strip Chart
- Complex Plane
- X vs Y Plot
- Polar Plot
- Waveform (Time)
- Spectrum (Freq) => Magnitude, Phase, Magnitude vs Phase (Polar & Smith)
- Picture
- Label
- Beep
- Note Pad

The details are given in their related Help menus and in Appendix C.

Lesson 6 Pre-Lab

There are four specific operations that will be either discussed or referenced in Lesson 6:

- using the Collector and Concatenator objects
- using the To/From File objects for I/O Transactions
- sending a Real Array to a File
- retrieving data using the FromFile Object

Menu Bar => Data => Collector
The collector is an object that can collect data from multiple iterations from a loop and generate a single array of data. It can accept multiple types of data, but only one type of data will be the output. The first piece of data determines the output type; if other input data in the loop is another type, then it is converted to the first output type. The collector can take a stream of scalar values in a loop and concatenate them to form an array.

Menu Bar => Data => Concatenator
The Concatenator can also be used. It gathers elements from multiple sources and combines them into a data container. The Concatenator, or its formula expression concat ( ), allows you to join elements from several inputs. When all inputs to a Concatenator have data, then the Concatenator combines these inputs together into a single data container. It then offers these data points as a one-dimension array.

> **Note:** The Collector is preferred when you are gathering data in a loop – each iteration of the loop generates a piece of scalar or array data and you want to gather all the data from each loop iteration. You may then use the data at the end of the loop. The Concatenator is preferred when you have multiple pieces of data and want to join them together.

To/From Files
The ToFile and FromFile objects were first used in Lesson 2. They are examined in greater detail below. There are some basic concepts that you should understand about these objects.

- A data file is opened on the first READ or WRITE transaction. When the program ends, VEE Pro closes any open files automatically.
- VEE Pro 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 is to be written.
- You can append data to an existing file or overwrite previous data.

If the Clear File at PreRun setting is checked in the Open View of the ToFile Object, the write pointer starts at the beginning of the file.

If the Clear File at PreRun setting is unchecked, the write 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 the file contents.
- A read pointer starts at the beginning of a file and advances through the data depending upon the READ transactions. You may perform an EXECUTE REWIND in the FromFile Object to move the pointer back to the beginning of the file without erasing the file contents.

To use FromFile, you need to know the format and location of the stored data.

Introduction to Understanding I/O Transactions

I/O Transactions are used to communicate with instruments, files, strings, the operating system, interfaces, other programs, and printers. The example of Figure 6.1 shows a ToFile Object sending (writing) the data of variable "a" on input pin **a** to a file named myFile. The highlighted line is known as a transaction line. It contains the default transaction statement: WRITE TEXT a EOL. When you double-click it, an I/O Transaction dialog box appears (Figure 6.1) which configures your specific transaction statement. You can add inputs to accept data from your program.

Note: If a Direct I/O channel address is active, then a real instrument must be connected to it.

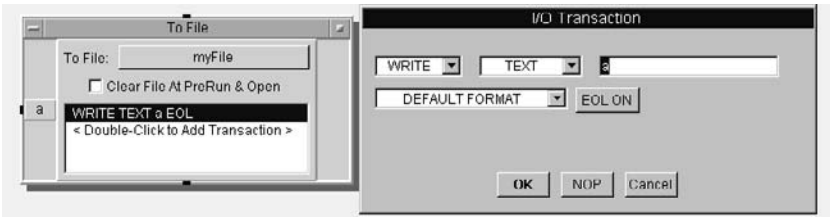


Figure 6.1. The ToFile Object with its I/O Transaction Dialog Box

Different objects call different forms of this dialog box which share certain common elements. We want to stress: the "actions", the "encodings", the "expression list", the "format", and the "end-of-line" (EOL) sequence. These elements form the structure for writing data, which usually takes the following format:

<action><encoding><expression list><format><EOL>

It is important for you to know that the <action>, <encoding>, and <format> elements provide selection via their individual drop-down menus. The <EOL> button toggles the End-of-Line character ON and OFF.

Only the <expression list> field requires keyboard entry.

The tables and comments that follow will provide you a basic understanding of these elements and the actions they perform. The most common actions available are READ, WRITE, EXECUTE, and WAIT. See Table 6.1.

Table 6.1. Actions and their explanations

Action	Explanation
READ	Reads data from the specified source using its encoding and format.
WRITE	Writes data to the specified target using its encoding and format.
EXECUTE	Executes a specific command. For example, EXECUTE REWIND repositions a file's 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: You can also examine a number of actions for Menu Bar =>
I/O => Advanced I/O Operations by exploring objects in that menu.

Encoding and formats refer to the way data are packaged and sent. For instance, a TEXT encoding sends data in the form of ASCII characters. The TEXT encoding could be formatted in several ways. Suppose, for example, you want to send a string of numbers and letters to a file. A WRITE TEXT STRING transaction would send the entire string – represented as ASCII characters. On the other hand, a WRITE TEXT REAL transaction would extract only the real numbers from the string of numbers and letters, and send those using ASCII characters for the individual digits. Table 6.2 provides explanations of encodings:

Table 6.2. Encoding explanations

Encoding	Explanations
TEXT	Reads or writes all data types in a human-readable form (ASCII) that can be edited or ported easily to other software applications. VEE Pro numeric data is automatically converted into text.
BYTE	Converts numeric data to binary integers. Sends/receives the least significant byte.
CASE	Maps an enumerated value or an integer to a string and reads or 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 Pro data types in binary files. CONTAINER Uses VEE Pro-specific text format with all data types.
CONTAINER	Uses VEE Pro specific text format with all data types.

In a write transaction, an "expression list" is simply a comma-separated list of expressions that will be evaluated to yield the data sent. The expression may consist of a mathematical expression, an input pin name, a string constant, a VEE Pro function, a UserFunction or a Global Variable, or any combination of them. The expression is very similar to a formula; however, the assignment of "a=b*3" syntax is not allowed. In a read transaction, the "expression list" should consist of a comma-separated list of output-terminal names indicating where the data to be read in should be placed.

Data types and formats were presented in Lesson 3, including reading data from instruments. Most of those formats apply to I/O transactions.

The EOL (end-of-line sequence of characters) may be turned on or off. You can specify the EOL sequence by opening the object menu of most of the "To" objects under Menu Bar => I/O => To => objects and selecting Properties…. Then select Data Format and make your changes under Separator Sequence.
The details of these operations are given below along with introductions to arrays and the To/From Files.

Introduction to Arrays
An array is a collection of like data types such as integer, real, and text. Arrays store multiple values of data, storing each data value in a contiguous memory location. This allows VEE Pro to accommodate programs performing repetitive computations.

An M-by-N array consists of M rows and N columns. Each row and each column illustrates an array dimension; in a 2-by-2 array the number of elements in each row determines the number of columns in the array. In theory, an array can contain an unlimited number of dimensions. See Figure 6.2 which is an example of a 3 row, 3 column array.

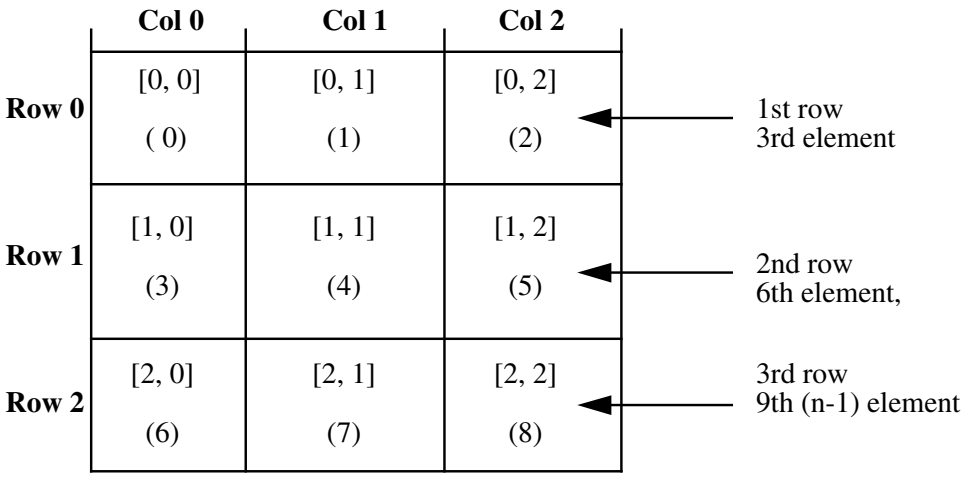


Figure 6.2. A Three-Row, Three-Column Array

In practice, three-dimension arrays (rather than four-dimension arrays) are used because they are easy to visualize and process. Arrays whose dimensions are four, five, and beyond are devised and applied primarily by mathematicians.

Indexing for arrays is zero-based; that is, the first element is numbered zero, the second element is numbered one, and the nth element numbered n - 1. Brackets [ ] are used to indicate the position of each array element.

Example: If the array A holds the elements [4, 5, 6], then A[0] = 4, A[1] = 5, and A[2] = 6.

The brackets either state the position of an array element or declare the value stored in the array. The syntax <arrayName>[index] is used to refer to that element in an array, known as the subarray syntax. The [<value>,<value>, …] syntax is used to create an array from the given values.

A one-row, three-column, one-dimension array is shown in Figure 6.3; the method of addressing is the same as for a multi-row array.

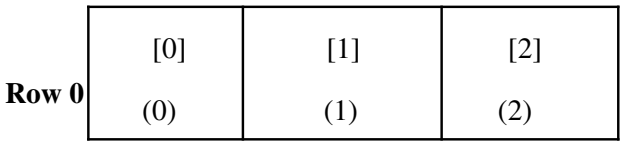


Figure 6.3. A one-row, three-column, one-dimension array

Accessing arrays requires the following notation:

The colon ":" is used to indicate a range of elements.

Example: A[0:2] = [4, 5, 6] where 0, 1, 2 are the data addresses or indices and 4, 5, 6 are their data values (in the same order).

The asterisk "*" is a wild card used to specify all elements in a particular array dimension.

Example: A[*] will access all elements of array A for further processing.

In the subarray syntax, commas are used to separate array dimensions. This syntax (to access elements of an array) can be used in the Formula Object or in any expression field, such as in the To/From File box Object.

Example: If B is a two- dimension array with three elements in each dimension, then B[1,0] returns the first element (which is 0) in the second row of B (which is 1).

The Collector Object is used to form an array by collecting a series of test-data results in a loop. It accepts data input until its XEQ is pinged (activated). Then it combines the data containers into an array and sends that array as one container. The Collector considers the data elements to be "in series", and collects them into a single data container.

Example: Use the For Count Object to simulate nine readings from an instrument. Place the readings in an array. See Figure 6.4.

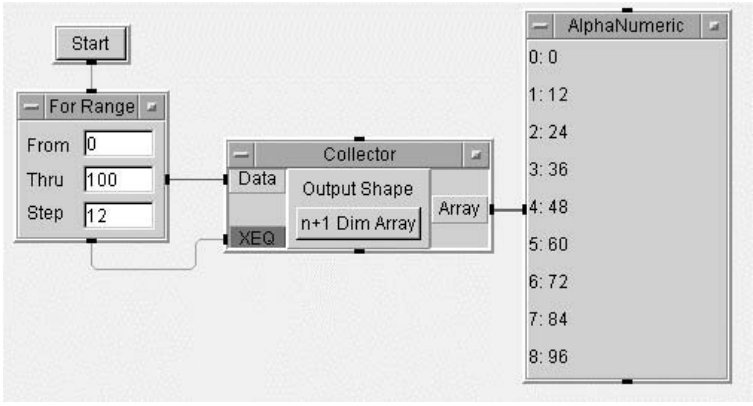


Figure 6.4. Creating arrays with the Collector

Note: This example uses an alphanumeric display that displays only one data container at a time. All of the numbers displayed reside in one container.

The Concatenator Object can be used to combine data in parallel as shown in Figure 6.5.

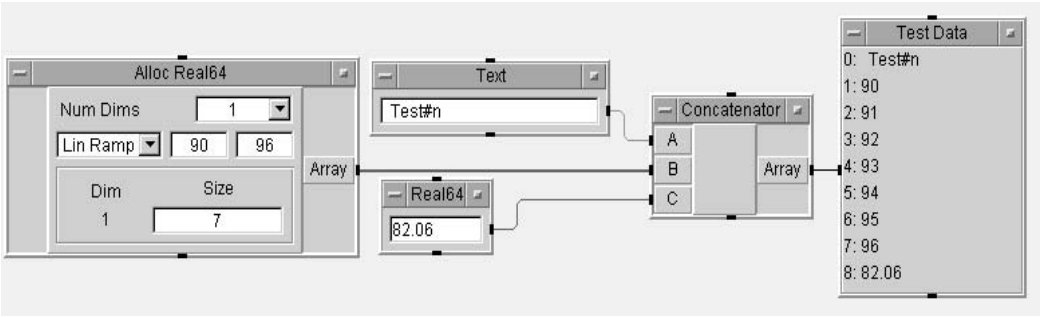


Figure 6.5. Creating arrays with the Concatenator

Lesson 7 Pre-Lab

Creating and Manipulating Records
The Record Object can store different data types and shapes in a single container. Elements in a Record Object are stored as fields and accessed via dot notation: Rec.Name would access the field Name within the Record Rec.

If you have created an array of records, Rec[*].Name would access all Name fields in array Rec; Rec[2].Name would address the Name field in the third record (element number 2) of the array. Remember: Array indexing is zero-based and * is a wild-card symbol that selects all elements in the record.

The Record Constant
The Menu Bar => Data => Constant allows you to create a Record that defines the record field names and values. It does not allow you to create a Record whose field is a Record. See Build Record Object.

The Build-Record Object
The Build Record Object enables you to build a record consisting of any number of data types (fields) you desire. This object requires you to define the shape of the output Record container. The object's Output Shape button lets you toggle between Scalar and Array 1D shapes. Later labs will demonstrate the editing and display of these multiple data-type inputs.

The Set Field and Get Field Objects
The Set Field Object lets you modify the content of an existing Record by writing an expression (actually an instruction) in its formula editing space. The object accepts Record-type data input and additional data inputs as required to modify the Record field(s). The Set Field Object output is a modified version of its Record data-type input; it complies with the expression in the Set Field formula editing space. The Set Field Object is actually merely a formula without a Result output; its expression input and output terminal names are preset.

You can extract Record fields using one Get Field Object for each data input and typing the target-field name using the dot notation described above. Note that each field name must match its target-label name in the Build Record Object. The format of the instruction you type in the Get Field formula white space is either

<input terminal name>.<field name>
or
<record variable name>.<field name>.

Note: The Get Field Object is also a formula with a preset expression.

The Unbuild-Record Object
You can extract all the fields of the record plus a listing of the field names and their data types via the Unbuild Record Object. Again, each field name (output terminal name) in the Unbuild Record Object must match the output field name of the Build Record Object (Record).

Lesson 8 Pre-Lab

Data Sets
A DataSet is an array of records stored in a file. To create a DataSet, you first create an array of records, then you specify the fields within each record. As an example, these could be a test name, a Real Scalar that could be a time stamp, and an array of Reals. You then store this array of records in a DataSet. From this DataSet, you can retrieve all the records with their specified fields, display and manipulate them as desired. The Record retrieval also has some search capabilities to retrieve only specific records.

Test Data Bases
A test database is a custom database that allows you to search and sort data. From these data you will be able to compute, display, and store a variety of statistics that you select via menu or devise on your own.

Operator Interfaces
An operator interface is a method of presenting the data or operations that you, as the program designer, will allow a program user (operator) to manipulate. The user will be able to modify only those choices that you specify. You can

- provide a test menu that will allow the user to select a particular test from which all related test data will be available,
- display the specified test results with the fields and values labeled; the user can then interact with the display to gain more detailed information, and
- provide clear instructions that allow the user to operate the program.

These three items become the program specifications.

Secured Run Time Version
You can create a Secured Version of your VEE program. This is known as a Secured Run Time Version. By default, it is saved as a ".vxe" extension instead of the usual ".vee" extension. **Caution** – there is no way to unsecure a secured .vxe program. Therefore, be certain to save and archive your **.**vee program prior to creating a Secured Run Time Version.

You can run secured programs in VEE RunTime, which is a run-only version of VEE Pro. You may distribute your program and VEE RunTime to an unlimited number of computers.

If the user or operator must observe or interact with the program as it executes, you must create Panel View(s) in the form of a Main Panel View or Pop-Up Panel(s). If you want your program to run without user intervention, then Panel Views are not necessary.

To DataSet
An object that allows the user to write records into a file. Use To DataSet to collect records that have the same format into a file for later retrieval. See Figure 8.1.

Figure 8.1. To DataSet Object

The following control pins can be added via "Add Terminal".

File Name – This control pin allows the user to change the name of the file into which the records will be written.

Clear File At PreRun – This check box "rewinds" the file and makes it available for rewriting with an entirely new record type.

Close – This control pin closes the file.

Open View Parameters

To DataSet – Click on the myFile field to display a dialog box, then select the name of the file that contains the DataSet.

Clear File At PreRun – If this check box is selected, the entire DataSet file is cleared and rewritten the first time that To DataSet executes (when it receives the first record). Because the DataSet is completely erased, records with a new format can be sent; the first record establishes the new format.

The records going into the To DataSet file must have the same data shape and type. The first record written to the file will define the format specification for all subsequent records. If the new record is an array, every record element is included. The number of fields, and each field name, type, and shape must be the same for every record. However, the shape of the record, either scalar or array, is not important.

From DataSet – An object that allows the user to conditionally retrieve records from a data set. Use From DataSet to find and retrieve from a data set either one record or all records that meet the constraints defined in the Search Specifier. If the record retrieved meets the constraints of that expression, then the record becomes available on the Rec output pin. The default expression in the Formula field is 1, (true). Thus, either the first record found (One), or all records in the data set (All) will be available at the output (Rec) pin. If you want to test values in record fields, then you must use the form Rec.A for field A, Rec.B for field B, and so forth.

For example, examine the expression: Rec.A<10.

This expression tests each record to see if field A is less than ten.

If Get records: One is specified, the first record in the data set with field A less than 10 will be available on Rec.

If Get records: All is specified, all records with field A less than 10 will be available on Rec.

The following control pins may be added. See Figure 8.2.

Figure 8.2. From DataSet Object

File Name – This control pin allows the user to change the name of the file from which the records will be read.

Rewind – This control pin "rewinds" the From DataSet file so that it can be re-read from the beginning of the file.

Formula – This control pin allows the user to change the formula used to determine which records read from the file will appear at the output of the device.

Close – This control pin closes the file.

Open View Parameters

From DataSet – Click on the "myFile" field to display a dialog box, then select the name of the file that contains the DataSet.

Get Records – Select One or All:

If One is selected, the first record found in the data set that meets the constraints of the selection formula will be its output. When One is selected, the output shape is always a scalar. Normal usage of "One" is to add a data output for "EOF" (end of file) and execute the From DataSet in a loop to get one matching record at a time. When EOF is reached, the EOF data output will generate and display the data values instead of the "Rec" output.

If All is selected, all records found in the data set that meet the constraints of the selection formula will be output. The output shape will always be an array. The entire data set file is searched to find all matching records.

Search Specifier

Enter a mathematical expression to test the records in the data set. Either the first One, or All records that satisfy the expression, will be sent to the output, depending on the Get Records selection. The default expression is 1 (true).

You may use any valid mathematical expression in the conditional formula. You may add additional inputs for use in the expression, as you would for a Formula Object. If you want to test values in record fields, then express them as Rec.A, Rec.B, and so forth. The expression can be fairly complex. For example: Rec.A > Rec.B AND Rec.C < 2.3

For any relational test (for example, equality between two operands), if one operand is an array, the other must be either a scalar or an array of the same size and shape. If the first operand is equal to the second, the result is True; otherwise it is False.

If both operands are of type Coord, they must have all of their independent variables and dependent variables match exactly for the result to be True. If independent variables do not match, an error is returned. Complex, PComplex, and Spectrum operands must have both parts match for the operation to return True. Enums are converted to Text for comparison.

Arrays must have all the values of both operands equal for the operation to return True.

You may wish to bring in a DataSet (an array of records) from a file, edit it, and then save it back to the file. This can be accomplished with a From DataSet, a Record Constant using the Default Value input pin, and a To DataSet. See Record Constant for more information.

You may add an EOF output pin to this object. When the object executes and there are no records meeting the query expression constraints, or you are already at the end of the DataSet, this pin will fire. Otherwise, an End of File with "no-data-found" error will result.

If you have multiple From DataSet objects reading from the same file, they will share the file pointer. That is, all of the objects read sequentially through the file, no matter how many objects there are reading the file.

Record Constant

An object that provides a constant format for the record data type. Use the Record Constant Object to define and build a constant record format, or to interactively edit an existing record or an array of records. See Figure 8.3.

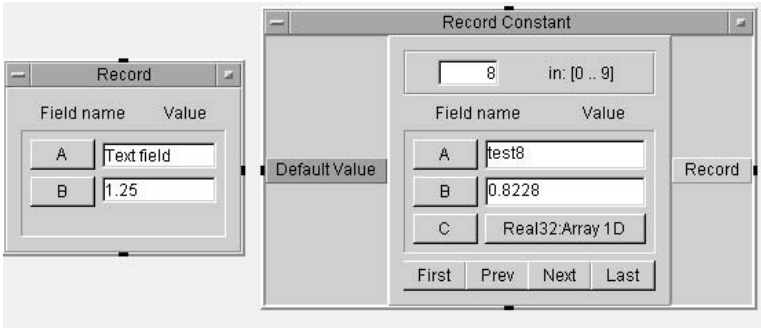


Figure 8.3. The Record Constant Objects (default object on left)

Open View Parameters

There is a Field name button and a Value button for every field in the record. The default Record Constant has two fields, named "A" and "B". You can add and delete fields; change the field types, etc. The buttons on the left ("A") and ("B") indicate the field names; the fields on the right contain the current values for these fields.

If the record is an Array 1D, then there is an additional field in the open view for the index into the array (0, 1, etc.). To step through the array values, First, Prev, Next, and Last buttons are provided.

Field name – Pressing the Field name button causes a dialog box to pop up. The user may re-name the record field, and change its type and shape. (An error will occur if you attempt to rename a field to an existing field name.) Field names are not case sensitive. When changing the data type, VEE Pro will attempt to change the data value into the new data type, subject to the existing rules regarding changing data types.

Value – The Value fields allow you to edit the values in the field. If the field is a scalar container of a data type other than Enum, then Value is a "type-in" field. (That is, you can click on the field and type in the desired value.) If the field is an Enum or Array 1D container, then the value field is a

button that displays a dialog box and allows you to select the values. If the field is an array of two or more dimensions, then the values may not be edited.

Object Menu – see Figure 8.4

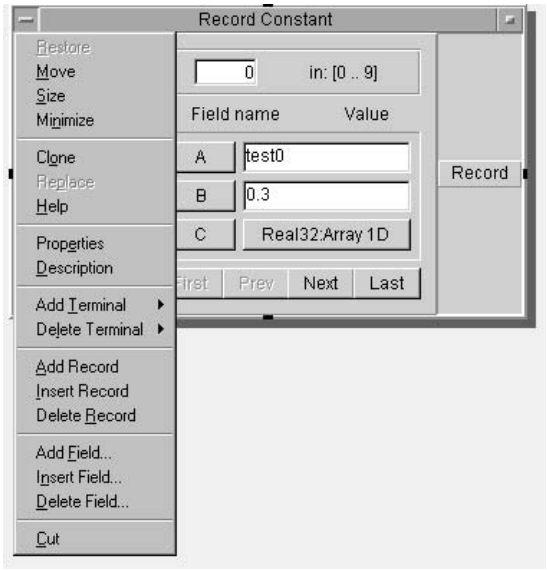


Figure 8.4. Object Menu for Record Constant

Add Record: Adds a record at the end of the record array. It is a copy of the last record in the array. This menu selection is not active for record scalars.

Insert Record: Inserts a record at the position that you are currently viewing. It is a copy of the record that you were viewing. This menu selection is not active for record scalars.

Delete Record: Deletes the record at the position that you are currently viewing. For record arrays, the selection is active only if there is more than one record in the array. This menu selection is not active for record scalars.

Add Field: Adds a new field to the record at the end of the field list. A dialog box will allow you to define the name, type, and shape of the new field.

Insert Field: Inserts a new field to the record at a specified position. A dialog box allows you to define the name, type, and dimension of the new field. A second dialog box allows you to specify the insertion point.

Delete Field: Allows you to delete record fields. A dialog box will list the available fields to delete. This selection is available only if the record has more than one field.

Record Constant Properties

See General; Configuration; Figure 8.5:

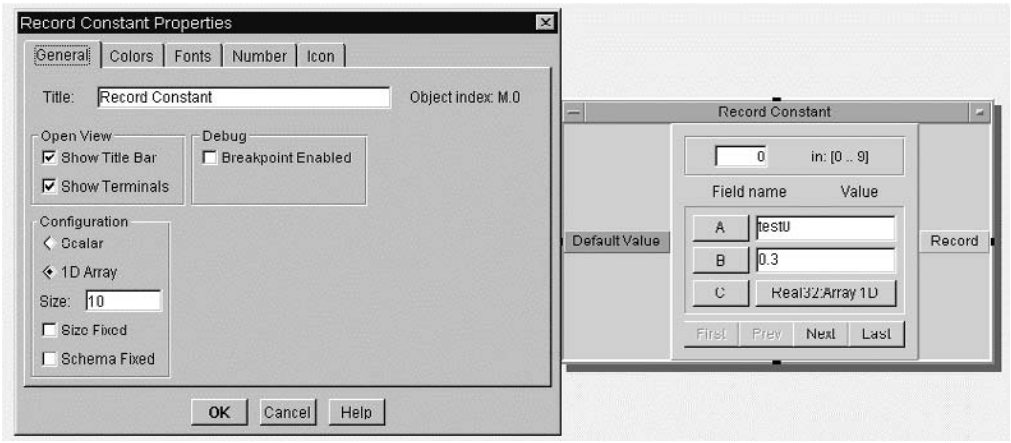


Figure 8.5. Properties: General Dialog Box

1D Array – Sets the initial data shape of the value(s) – the output of this object.

Size – When 1D Array is selected, Size sets the length of the array.

Size Fixed – When this check box is selected, the array cannot have lines added or deleted. This selection is only available if 1D Array is selected.

Schema Fixed – When this check box is selected, the format ("schema") of the record is fixed, and may not be modified. This is useful when allowing users to modify Record values interactively on a pop-up panel.

See Number; Figure 8.6:

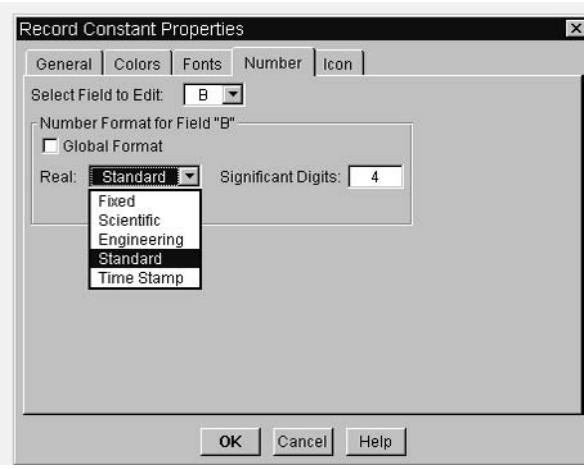


Figure 8.6. Record Constant properties: Number

Select Field to Edit – Allows you to select the field name corresponding to the number-format field.

Global Format – This is default format.

Integer – If Global Format is off, specify the number format for integer numbers to be displayed on this object. The choices are Decimal (default), Octal, Hexadecimal, and Binary.

Real – If Global Format is off, specify the number format for real numbers to be displayed on this object. The choices are Fixed, Scientific, Engineering, Standard (default), and Time Stamp.

Control inputs (Figure 8.7) may be added for Default Value, Reset, Enable Editing, and Disable Editing.

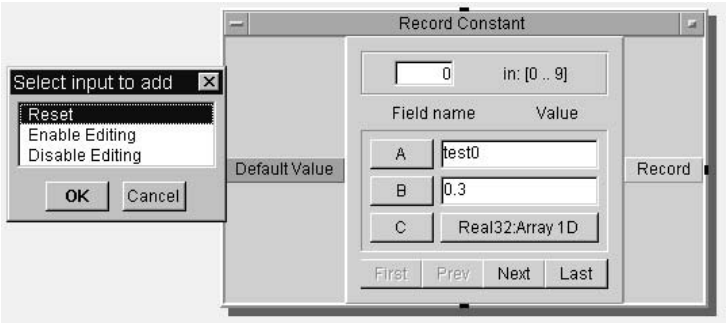


Figure 8.7. Record Constant control input options

The Default Value input allows you to change the current value(s). This input requires a Record value, either a Scalar or an Array 1D. The Constant is reconfigured to match the shape of the incoming data. The Default Value pin is useful for editing large record arrays (for example, a record array "pulled in" with the From DataSet Object).

The Reset input allows you to set the value(s) to zero. If the Constant is configured as an Array 1D, all values are set to zero. (Numeric fields become zero and Text fields become null strings.) The actual data on the Reset input is ignored.

The Enable Editing and Disable Editing inputs allow you to control whether the value can be changed in the object's open view.

Note that the Record Constant Object does not allow you to create a record of records as do the Build Record and Set Record objects.

Lesson 9 Pre-Lab

VEE Pro 6: ActiveX References

Figure 9.1 displays the Microsoft set of ActiveX Automation References as installed on a typical PC. ActiveX replaces DDE wherever possible. However, DDE is still supported by VEE Pro for so it can be used in earlier versions of VEE. ActiveX can generate test and measurement program reports. ActiveX is more flexible than DDE. All installed libraries and programs are listed in the entire ActiveX Automation Reference Box.

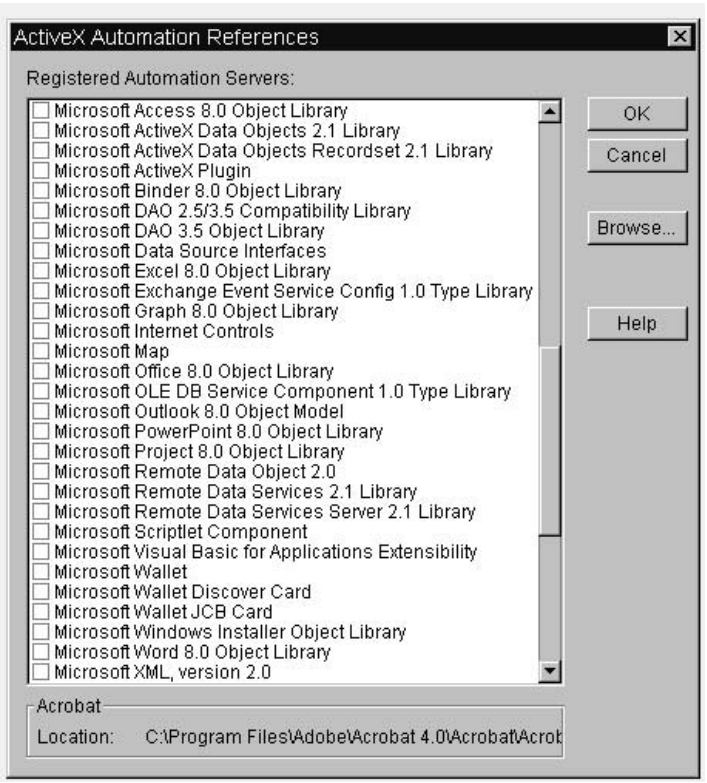


Figure 9.1. Microsoft ActiveX Automation References

Microsoft Excel™ Column and Row Notation

Designers of Microsoft Excel™ chose to label each two-dimensional entry as follows:

Columns: Letters starting with A

Rows: Numbers starting with 1

An entry example would be: B11 – the second column and eleventh row. Each entry location is referred to as a "cell" whose content can be alpha, numeric, or alphanumeric. Where data processing is involved, it is customary to limit the entries of a column to only one of these three types of data.

Microsoft Excel™ Cell Access

Cells are accessed either via an external address, such as from VEE Pro, or via an internal address entered within the Excel™ spreadsheet by the user. Once all data are entered, then that data can be processed by math that is available either externally, such as from VEE Pro, or by equations written using the f_x icon available to the left of the Excel™ white expression space. Excel™ can be used as a spreadsheet, connected to VEE Pro via the ActiveX program. Therefore, our first focus is on ActiveX, how it operates, and how it can transfer data first to Excel™ and later to Word™.

The ActiveX Object Variable

An object is a pointer to something or some data held by the Automation Server. It can point to a worksheet within Excel™ or to a specific cell inside of that worksheet.

The ActiveX Object Discovery

Different installations offer different libraries. For example, having Microsoft Office™ versions 8, 9, or 10 installed will show different libraries (ActiveX automation servers) available. In these libraries, there are various applications that may or may not be useful to you. You will have to sort through these applications to decide which ones to use. Your computer system may not have enough room for all of them. Experience in their use will guide you to those that are most useful to your applications.

Declare Variable

The Declare Variable Object allows you to declare variables with specified name, time, and scope. This is useful for sharing data between various UserFunctions in your program. Advanced users can also use Declare Variable to enable "catching" of events from external ActiveX automation servers, such as "catching" an event from Excel™ or Word™.

Lesson 10 Pre-Lab

The Dynamic I/O Automation Server

The VEE Pro Dynamic I/O Automation Server lets you find and identify instruments programmatically. In previous versions of VEE, instrument bus addresses in the VEE I/O configuration file were static (fixed). This provides new levels of flexibility. For example, you can find all your instruments and then use Programmable Instrument Properties to set up Direct I/O objects.

Because this utility is a stand-alone ActiveX automation server, it can also be used from other applications, such as C++ and Visual Basic. For more information for using this server in languages similar to Visual Basic, refer to the VEE dynamic I/O server "Help".

ActiveX Automation References

The type library for an ActiveX application must be made available to VEE Pro before you can create expressions which control that application (Device => ActiveX Automation References…). The ActiveX Automation References dialog box displays ActiveX automation libraries available for use in VEE Pro. Use the ActiveX Automation References dialog box to select the ActiveX automation libraries for VEE Pro to use to control other applications. Libraries appear in this dialog box if their associated applications have been installed so the Windows Registry recognizes them. When you select the libraries you want, press OK to load them into memory for use in VEE Pro, and to search for their object classes, dispatch interfaces, and exported events. You must select type-libraries only if you plan to handle events generated by them

After referencing type libraries, use the built-in functions CreateObject() or GetObject() to connect with an automation object. Open VEE Pro's Function & Object Browser (Device => Function & Object Browser) and select ActiveX objects in the Type area. This will generate expressions in Formula objects to manipulate the objects.

Dialog Information

Registered Automation Servers contain the list of registered ActiveX automation libraries. Click on a name to highlight it and see information in the status area below the list. Click in the box beside a name, and click OK to select it for use in VEE Pro.

If a library exists on your system that does not appear in the list, click Browse to search for it in the Add ActiveX Automation Reference dialog box. When you find and open the library, VEE Pro adds it to the list, and attempts to register it. To use it in VEE Pro, be sure its box is checked, and click OK.

Refer to "Using ActiveX Automation Objects and Controls" in the VEE Pro Advanced Techniques manual.

Lesson 11 Pre-Lab

Merging Objects
VEE Pro provides several library objects (pre-built UserObjects) and complete library programs that you can merge into your own VEE Pro program.

Select File => Merge. The Merge dialog box will appear. By default, the Merge dialog box looks for library objects and programs in the current directory.

You can also merge any VEE Pro program file into the work area by changing directories. Click on the file name of the library file that you want to open, and then click on Open. The object is merged into your workspace. You can connect it into your program like any VEE object.

UserObjects
A UserObject provides the means for you to encapsulate a group of objects that perform a particular task into a single, custom object. This encapsulation allows you to do the following:

You can create a UserFunction by creating a UserObject and then executing Make UserFunction (refer to UserObject for details). You can also create a UserFunction by selecting a group of objects and using Edit => Create UserFunction....

You can use Edit UserFunction to edit the UserFunction once it has been created. UserFunctions exist in the "background" and can be called with Call Function or from certain expressions.

An advantage of making a UserFunction out of a UserObject is that the same UserFunction can be called multiple times within the program, but it exists as source code only in one place.

Use modular design techniques in building your VEE program. This allows you to solve a complex problem through an organized approach. UserObjects allow you to use top-down design techniques to create a more flexible and maintainable program.

Build user-defined objects that you can save in a library for later re-use. Once a UserObject is created and saved, you can merge it into other programs.

A UserObject can be used to create a UserFunction. A UserFunction can be created from a UserObject or from a group of selected objects.

You can create a UserFunction by creating a UserObject and then executing Make UserFunction (refer to UserObject for details). You can also create a UserFunction by selecting a group of objects and using Edit => Create UserFunction....

You can use Edit UserFunction to edit the UserFunction once it has been created. UserFunctions exist in the "background" and can be called with Call Function or from certain expressions. An advantage of making a UserFunction out of a UserObject is that the same UserFunction can be called multiple times within the program, but it exists as source code only in one place.

User Functions

There are three types of user-defined functions in VEE Pro.

UserFunction – It is a reusable segment of VEE code. There is only one copy of a particular UserFunction in a VEE program. Therefore, you can access (call) it from various places within your program. Changing a portion of your UserFunction automatically modifies all applications of that UserFunction. You can call a VEE Pro UserFunction by using Menu Bar => Device => Call (myFunction) or by using an expression within an object; an example would be from Formula. UserFunctions can be either local or imported. You can also create libraries of UserFunctions for use within other programs. You would import the UserFunction library. However, an imported UserFunction cannot be edited. An imported UserFunctions can be called in a manner similar to the calling of a local UserFunction.

Compiled Functions – A Compiled Function is a function written in another language, such as C or C++. It is compiled into a dynamic linked library (DLL). You can load this library of compiled functions into VEE Pro via the Import Library Object. Then you can call a compiled function via a CALL Object or by writing an expression within a VEE Pro object. Note that, since these compiled functions are loaded into VEE Pro, any errors or calling exit ( ) from a compiled function can cause VEE Pro to crash. When there is insufficient memory allocation or memory is overwritten, it can cause VEE Pro to randomly crash.

Remote Functions – A Remote Function is a VEE UserFunction that is running on another computer. You can call a remote function the same way that you would call a UserFunction. However, the remote function runs on a remote host computer that is connected to your network. This Remote Function is specified via an Import Library Object.

Your new programs can be easily merged with existing VEE Pro programs using the Merge… command. This command is in the File menu. When you use the Merge… command, you will see a list box that displays the library (lib) subdirectory. This subdirectory is in the VEE Pro home directory. Merge is used to copy an entire program into another program. Choose the UserFunction whenever you want to reuse code. Thus, only one copy is retained. To copy UserFunctions from one program to another program, use Menu Bar => File => Merge Library.

Lesson 12 Pre-Lab

The Instrumentation Amplifier

The Instrumentation Amplifier (IA) amplifies the voltage difference across its input terminals. The IA ideally applies no gain to identical signals that are simultaneously applied to both inputs; these signals are known as common-mode signals. Common-mode voltages are caused by excessive return current from various parts of the overall system via the IA signal-return conductor. The resulting error voltages can easily mask the small output signals generated by many types of sensors. Proper (single-point) grounding techniques are the usual solution.

The circuit of Figure 12.1 shows a conceptual model of the Instrumentation Amplifier. This circuit is used in the next lab to show that the basis for common-mode rejection is simple subtraction. The Instrumentation Amplifier is a differential amplifier with two identical gain channels of opposite sign. If the same voltage is applied to both inputs, these voltages will cancel each other before arriving at the IA output. See Figure 12.1 for the circuit wiring that is simulated in the laboratory experiment.

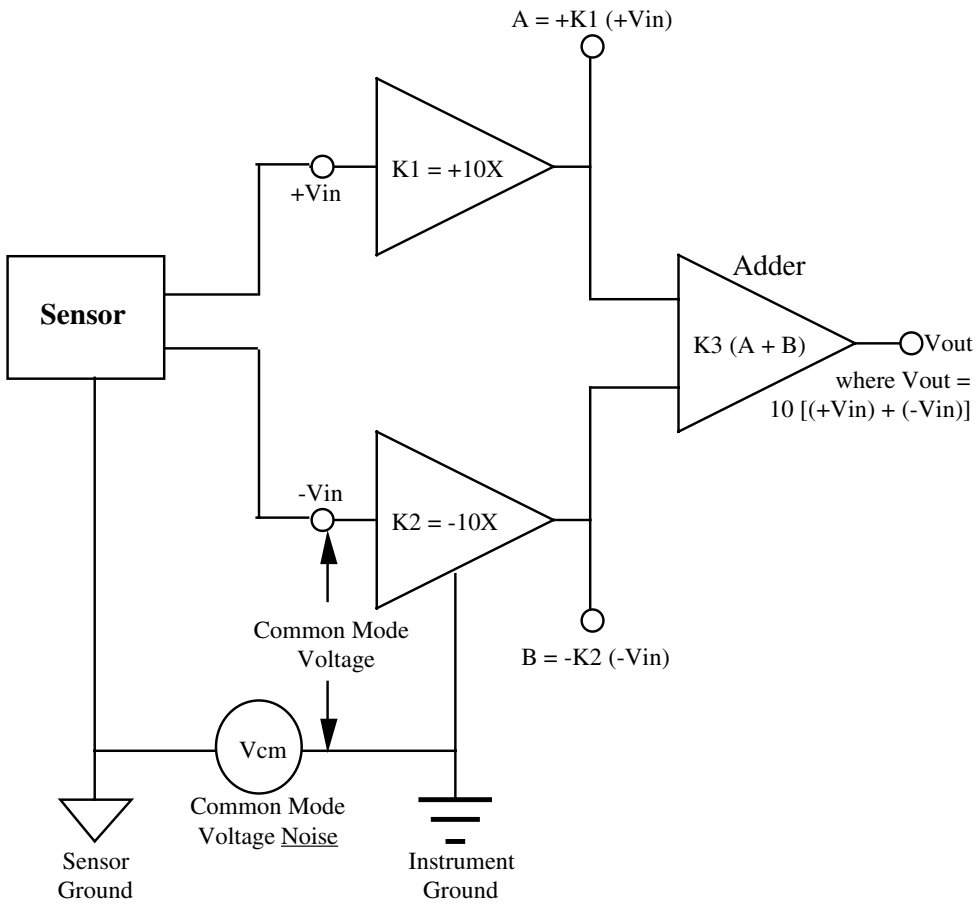


Figure 12.1. Instrumentation amplifier simulation showing common mode rejection

The instrumentation amplifier is widely used for interfacing sensor outputs to the next portion of a data acquisition subsystem. It provides both temperature-stable, steady-state (dc) and dynamic (ac) performance in addition to its high input impedance. This high impedance minimizes loading of the signal source and instrument-amplifier gain. You can easily simulate the instrumentation amplifier in VEE Pro. You can then use your simulation to explore the operation of both an ideal and non-ideal instrumentation amplifier.

Understanding a Strain Gauge

Strain gauges indirectly determine strain by measuring the resistance change (ΔR) on the structure being monitored. As the force on the structure changes, the strain gauges on the same surface of the structure as the applied force will increase in value while the strain gauges on the opposite surface will decrease in value. See Figure 12.2.

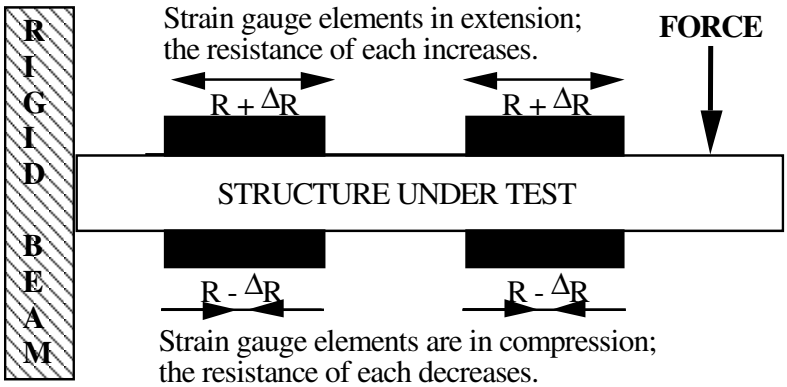


Figure 12.2. The four strain gauge elements on a structure being deformed

The strain-gauge resistors (ΔR) are connected electrically as a Wheatstone Bridge. The bridge excitation voltage (V_{EXC}) is the electrical input. The bridge electrical output (V_{BRIDGE}) is the operational amplifier (op amp) input. The relating equation is given in Figure 12.3. The op amp replaces a meter; its two inputs are either above (+) or below (-) system ground.

The strain-gauge resistors compensate for the structure temperature. The stretching and shrinking of all four resistors provide the differential-resistance change (ΔR) that represents strain. The op amp provides both circuit isolation and gain (K). The output voltage is $V_{OUT} = K \times V_{BRIDGE}$. The output of the op amp is connected to an n-bit analog-to-digital converter. The information provided to the data-acquisition system is a digital word.

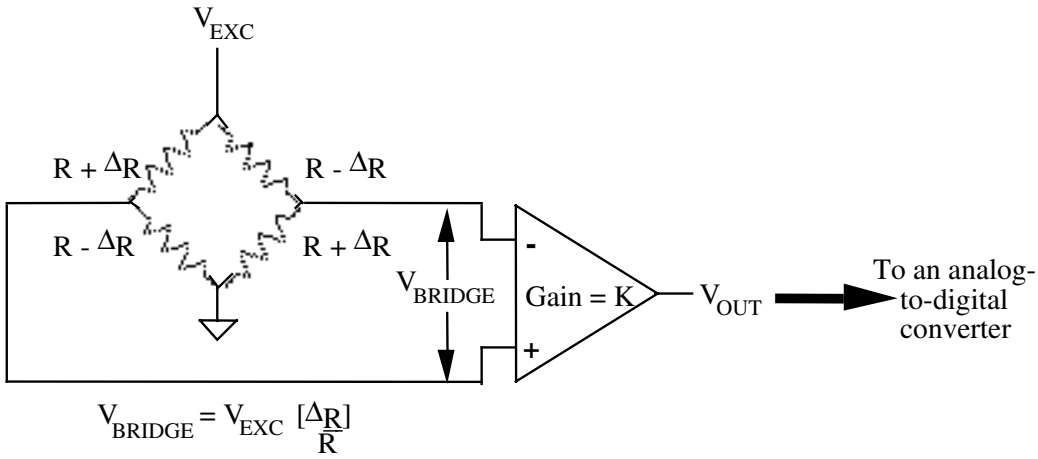


Figure 12.3. A four-element strain gauge network

The VEE Pro-provided application programs

VEE Pro applications are located on the hard-drive location in the following sequence:

The Program Files folder,
The Agilent folder,
The VEE Pro 6.0 folder,
The examples folder, and
The Applications folder.

There are sixteen VEE Pro-supplied application simulations:

beam.vee*	chaos.vee
coilspr.vee*	convecoef.vee*
convert.vee	digfilt.vee
fluidflw.vee	lissajou.vee
mfgtest.vee*	payback.vee
pid.vee	simp_npv.vee
telecomm.vee	torsion1.vee
torsion2.vee*	wizbang.vee

The ones with the asterisk (*) are the ones explored in Lesson 12.

MATLAB®

VEE Pro 6 and higher access MATLAB® Script. This object is located in Menu Bar => Device. MATLAB® programs can be written within the MATLAB® Script Object expression field. See Figure 12.4.

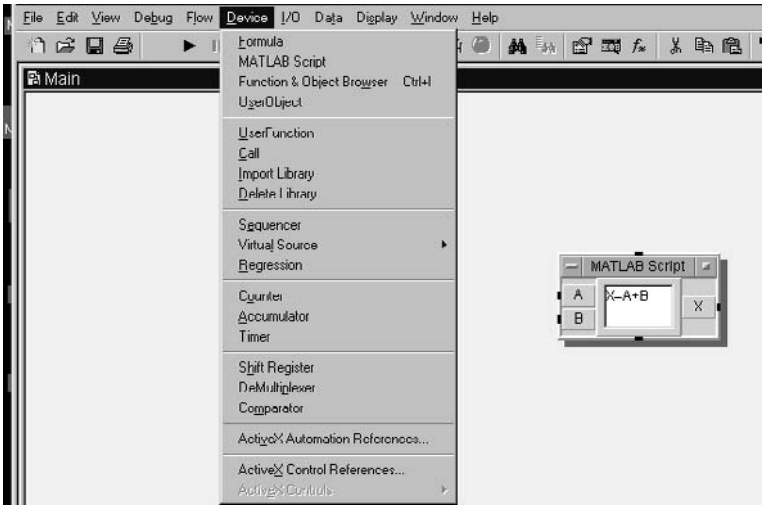


Figure 12.4. MATLAB® Script Object

Also, via the Function & Object Browser, MATLAB® functions can be accessed via the Type (left-hand) column. Their major categories are listed in the Category (middle) column; the programs within each category are located in the Function (right-hand) column. See Figure 12.5.

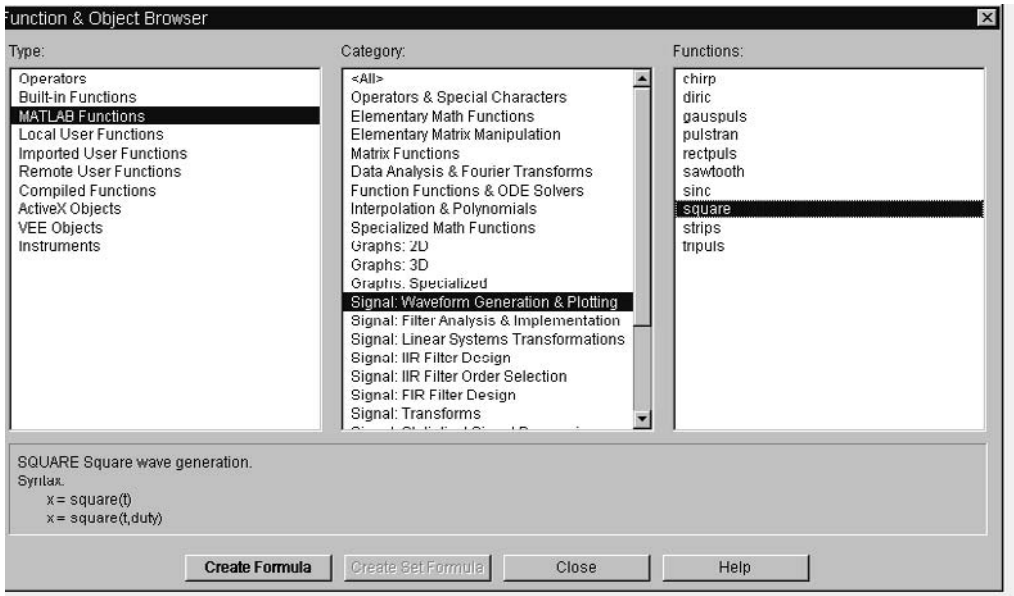


Figure 12.5. MATLAB® Function & Object Browser

Lesson 13 Pre-Lab

Menu Bar => Display Spectrum (Freq)
There are four subheadings listed:

- Magnitude Spectrum
- Phase Spectrum
- Magnitude vs Phase (Polar)
- Magnitude vs Phase (Smith)

The subheading investigated in this lab is the Magnitude Spectrum icon. Its horizontal scale is frequency (Freq). Its vertical scale is magnitude (Mag). Additional input terminals can be added to provide other traces.

Horizontal variables are: scale name, auto scaling (on or off), maximum and minimum range, label spacing, mapping (linear or logarithmic), log cycles, and scale color. These variables are accessed by a single mouse click on Freq or via the Properties menu.

Vertical variables are: scale name, color, line type, and point type. These variables are accessed by a single mouse click on Trace 1 or via the Properties menu.

The Time and Frequency Domains
Today, there are four generally accepted classes of waveforms that vary versus time:

- continuous
- discrete
- pulsating
- alternating

These waveforms may be periodic or non-periodic.

A continuous waveform varies from one moment to another; it does not change suddenly. Therefore, it does not have a disruption in its waveform. Continuous waveforms are considered to be functions of time because, for every value of time, they have only one value of voltage, current, or power.

A discrete waveform changes suddenly from its present value to another value at one or more points in time. An example would be a series of steps or a “sawtooth” waveform.

A pulsating waveform changes quickly and then returns to its original value. Such a waveform may contain one or more pulses. An impulse is an extreme example of a pulsating waveform.

Alternating waveforms vary from one moment to another; they may, or may not, change suddenly at any moment of time. However, they “alternate” from a positive value to a negative value as time changes. These waveforms may be functions versus time if they have only one vertical value for any given value of time. They may be relations versus time if they may have more than one value of voltage, current, or power for any given value of time.

When a sine-shaped waveform represents a voltage, the equation that describes this waveform is

$$e = E_m \sin \alpha \qquad \text{eq 13.1}$$

where E_m represents the maximum value (or peak) of the sine-shaped waveform and α represents the angular displacement from the positive-X axis.

When the circular turning of any device is examined, it will generate a sine-shaped waveform. A sine-shaped waveform displayed in the time domain can be graphed by projecting the tip of a line that rotates with a circular motion. The circular display is graphed in polar coordinates; the projection of the tip of the line is graphed in cartesian coordinates.

The mechanical velocity **v** of a device is usually expressed in revolutions per minute (rpm). It is often converted to a frequency **f** whose unit is the cycle per second or Hertz. The relation between cycle per second (Hertz) and angular velocity ω requires trigonometry:

$$\omega = \frac{2\pi \text{ radian}}{1 \text{ cycle}} \cdot f \frac{\text{cycle}}{\text{second}} = 2\pi f \frac{\text{radian}}{\text{second}} \qquad \text{eq 13.2}$$

The unit of angular velocity is the radian per second. As a math review, there are 2π radians or 360 degrees in one mechanical revolution or one electrical cycle. Therefore

$$\omega = 2 \cdot \pi \cdot f \qquad \text{eq 13.3}$$

The frequency **f** of a sine-shaped waveform is related to its period **T**.

$$f \frac{\text{cycle}}{\text{second}} = \frac{1}{T} \frac{\text{second}}{\text{cycle}} \qquad \text{eq 13.4}$$

For the sine-shaped waveform whose base is zero, its one-period average height $\mathbf{E_{avg}}$ is zero because there is as much area under the positive part of the waveform as there is under the negative part of the waveform. However, the average value for one-half of a sine-shaped waveform can be derived. The result is:

$$E_{avg} = \frac{2}{\pi} E_m \qquad \text{eq 13.5}$$

The $2/\pi$ value is exact; many texts provide this value as $0.637\ E_m$, which is approximately $2/\pi$.

Power is the product of voltage **E** and current **I**. It continues to be the product of the instantaneous values of voltage **e** and current **i**. Assume a voltage **e** and current **i** where

$$e = E \sin (\omega t + \alpha_e) \qquad \text{eq 13.6}$$

$$i = I_m \sin (\omega t + \alpha_i) \qquad \text{eq 13.7}$$

where α_e and α_i are different initial phase angles for voltage (α_e) and current (α_i). Apply the trigonometric identity:

$$\sin A \cdot \sin B \equiv \frac{1}{2} \cos (A - B) - \frac{1}{2} \cos (A + B) \qquad \text{eq 13.8}$$

where $A = \omega t + \alpha_e$ and $B = \omega t + \alpha_i$. The result becomes:

$$p = e \cdot i = \frac{E_m I_m}{2} \cos (\alpha_e - \alpha_i) - \frac{E_m I_m}{2} \cos (2 \omega t + \alpha_e + \alpha_i) \qquad \text{eq 13.9}$$

In the late 1880s, oscilloscopes did not exist. Thus, those who were experimenting with resistors, capacitors, and inductors (such as Thomas Edison and Charles Steinmetz) could not observe the phase differences. After many comparison experiments, Dr Steinmetz suspected there was a phase shift between capacitor and inductor voltages and currents. He used calculus to investigate and verify the voltage-current phase shift and other interactions. Interpretation is required as suggested by Dr Steinmetz:

- The first term is a constant and independent of time; time does not appear in this term. He named it the average power value.
- The second term is time-dependent; its repetition rate is at twice the frequency of the original voltage and current; its phase shift is the sum of α_e and α_i.

He then suggested that the first term be redefined by splitting the product of voltage and current into two parts:

$$\frac{E_m}{\sqrt{2}} \cdot \frac{I_m}{\sqrt{2}} = E \cdot I \qquad \text{eq 13.10}$$

where E would be known as the effective value of voltage and I would be known as the effective value of current. The word effective refers to the product of E and I; it would effectively represent the value of useful power transferred from one location to another.

For Cartesian coordinates, the X axis is considered to be the **domain** of a function and the Y axis is considered to be the **range** of that same function. (Recall that, for a function, there can be only one Y value for each corresponding X value.) Therefore, the expression "time domain" indicates that the X-axis (domain) label is replaced with the parameter: **time**. The Y axis (range) label may be replaced with either a voltage – which is the most common – a current, or (seldom) a power. These displayed waveforms are usually sine-shaped waveforms; they can be displayed on an oscilloscope.

The expression "frequency domain" indicates that the X-axis label is replaced with the parameter: **frequency**. The Y axis (range) label is most often power because the display is usually associated with spectrum analyzers. The frequency-domain waveform is usually one or more vertical lines, representing the amount of power being transferred at that frequency. These vertical lines are usually displayed on a spectrum analyzer.

A comparison of a sine wave displayed in the time domain (on an oscilloscope) and its frequency-domain equivalent (displayed on a spectrum analyzer) is shown in Figure 13.1.

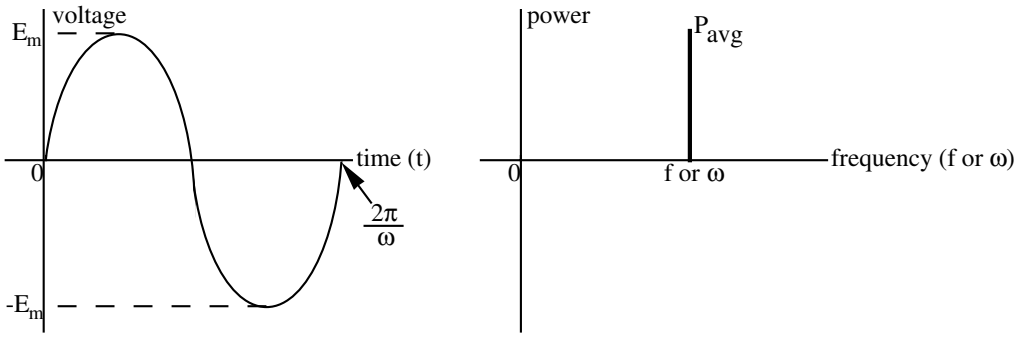


Figure 13.1. A Comparison of Time-Domain and Frequency-Domain waveforms

As noted earlier, the relation between the waveform period **T** and frequency **f** is:

$$f = \frac{1}{T} \qquad \text{eq 13.13}$$

where **f** is measured in Hertz (cycles/second) and the period **T** is in seconds/cycle.

Include the entire unit for period. Most persons state the unit of period as time (in seconds). The unit that belongs with a period computation is actually time/cycle or seconds/cycle. The importance of keeping the entire and correct unit for a period will become more apparent whenever the Laplace Transform is applied.

The phase of the sine wave shown in Figure 13.2 is zero. (The frequency-domain display does not normally indicate phase.) Dr Steinmetz "froze" the rotating line at t = 0. He devised a method for correcting this shortcoming regarding phase difference; it also avoided calculus because he discovered that engineers (and other technical people) in the United States were not comfortable working with calculus. He examined mathematically the relations among voltage, current, and power, and how these relations could be simplified if the calculus were reduced to algebra and trigonometry. He named the method: **phasor algebra**, which is a shortened version of phase angle vector. He purposely avoided the word vector because the rules for vector computations do not totally apply to phasor computations.

When working with phasor algebra, recall that:

$$A \angle\alpha = A \cdot \varepsilon^{j\alpha} \qquad \text{eq 13.14}$$

Therefore, phasor lengths are multiplied; phasor angles are added algebraically. The left side of this equation is known as the polar form of a phasor; the right side of this equation is known as the exponential form.

The relation between time and angle is

$$\alpha = \omega t = 2 \pi f t \qquad \text{eq 13.15}$$

where **α** is measured in radians, **ω** in radians/second, **t** in seconds, and **f** in Hertz (which replaces the cycle/second).

Note that **ω** is the frequency given in radians/second. Therefore, its relation with **f** in cycles/second (Hertz) is

$$\omega = 2 \cdot \pi \cdot f \qquad \text{eq 13.16}$$

Once the computations (at t = 0) involving phasors were complete, the phasor line would be allowed to rotate again. The resulting waveforms of voltage, current, and power would be the same. The phasor approach included angle computations, which are not displayed on most spectrum analyzers. This was an enormous step forward in the analysis of circuits involving resistors, capacitors, and inductors and, today, electronic circuitry and operational amplifiers.

The use of the term **frequency domain** for phasor algebra is not quite correct. Dr Steinmetz described how to compute the frequency-domain relations among voltage, current, and power. Then, after the analysis was complete, he let the phasors rotate at those angular velocities that represented their original frequencies. (Recall from above that the frequency of the power phasor would be at twice the frequency of the voltage and current phasors.) As the voltage and current angular relations would remain the same, his trigonometric analysis would apply for all of that time represented by the voltage and current phasors.

As the angle-domain (X-axis) should be shown horizontal, then the **y = cos α** waveform is rotated so its angle axis is horizontal. The cosine-wave function is then also rotated. This is the way that the cosine-shaped wave is usually displayed. The initial phase angle, α_1, agrees with the positive-cosine reference axis. Again, note that the phasor always rotates in a counterclockwise (CCW) direction.

The reference axes, used primarily by mathematicians and physicists, were suggested by Dr Steinmetz. A positive angular motion is counterclockwise (CCW) from the phasor's initial position. A negative angular motion is clockwise (CW) from the phasor's initial position. As more clocks and watches become digital, these standards will become more difficult to remember. Use your imagination to visualize a car that is passing in front of you from right to left. Its wheels will be rotating counterclockwise.

Periodic Waveforms and Their Fourier Series

In 1822 March 21, Jean Baptiste Joseph Fourier (1768-1830) published Analytical Theory of Heat where he described how to analyze heat conduction in solids in terms of an infinite series of sine and cosine terms, now known as a Fourier series. After reading that publication, Pierre Simon de Laplace (1749-1827) criticized the work because it did not include relations as well as functions. Most mathematicians, at that time, considered the omission to be trivial. Today, as technical personnel, we realize that its omission can lead to difficulties in the analysis of certain periodic waveforms.

A non-sine-shaped periodic waveform can be expressed as a combination of sine and cosine waveforms. Their frequency, amplitude, and phase angles will be directly related to the fundamental frequency of the non-sine-shaped periodic waveform.

The most commonly expressed waveform is the "square" wave. For the square wave that starts (at t = 0), its Fourier-series equation is:

$$\text{square wave}(\omega) = \sin \omega + \frac{1}{3} \sin(3\omega) + \frac{1}{5} \sin(5\omega) + \ldots + \frac{1}{n} \sin(n\omega) \qquad \text{eq 13.17}$$

where **n** is an odd integer.

Note that the leading and trailing edges are multi-valued. They are not functions; they are relations. Therefore, it is impossible to simulate a square wave exactly. Laplace was right!

Figure 13.2 further indicates that the frequency spectrum of a perfect sine-shaped wave is a function; it has one vertical value for every horizontal time value. What will be the Fast Fourier Transform frequency-spectrum magnitude presentation?

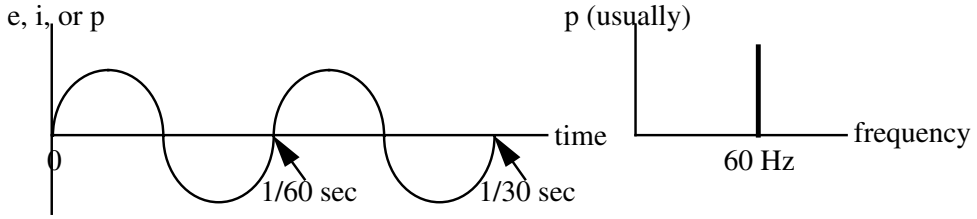


Figure 13.2. The Time and Frequency Display for a sine-shaped wave

The Fast Fourier Transform (FFT) of this waveform will be displayed as shown in the "virtual" diagram of Figure 13.3 because the FFT can only display functions, not relations.

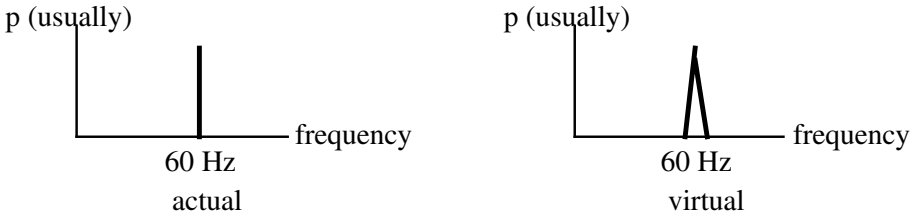
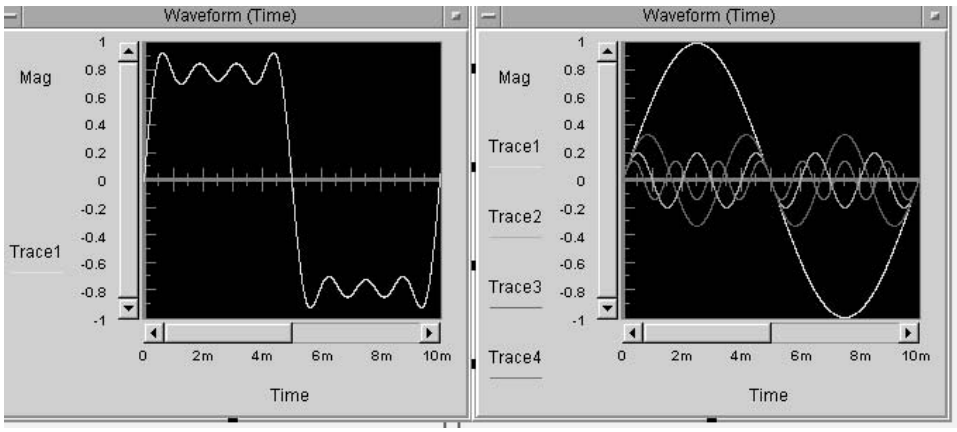


Figure 13.3. A Frequency-Domain Display on a simulated spectrum analyser

Observe three of the square-wave individual sinusoidal harmonics prior to being added to approach the simulated square wave. See Figure 13.4.



Harmonics Added Harmonics Separated

Figure 13.4. Sinusoidal harmonics

Examine the right-hand oscilloscope presentation. It represents the separate first four waveforms: harmonics 1, 3, 5, and 7, prior to their addition. (Reminder: The first harmonic is often referred to as the "fundamental".) The left-hand oscilloscope presentation represents the algebraic sum of these first four frequencies.

Other commonly-applied waveforms in science, engineering, and technology include the periodic triangular wave and the trapezoidal wave. The Fourier-series equation for the triangular wave is:

$$\text{tri wave}(\omega) = \cos(\omega) + \frac{1}{3^2}\cos(3\omega) + \frac{1}{5^2}\cos(5\omega) + \ldots + \frac{1}{n^2}\cos(n\omega) \qquad \text{eq 13.18}$$

The Fourier-series equation for the trapezoidal wave is:

$$\text{trap wave}(\omega) = \cos(\omega) - \frac{1}{3^2}\cos(3\omega) - \frac{1}{5^2}\cos(5\omega) + \ldots \pm \frac{1}{n^2}\cos(n\omega) \qquad \text{eq 13.19}$$

where the signs of the terms alternate every second term.

Other periodic waveforms that are considered applicable to science, engineering, and technology-oriented problems are discussed in Tilton (1986) and Wei and Zhang (2000).

Lesson 14 Pre-Lab

Using Library Functions
The VEE Pro library functions are very flexible. You can

- use libraries via the VEE Pro UserFunctions,
- devise and merge a library of UserFunctions,
- devise a top-level program without programming the details of the UserFunctions,
- devise a report-generation program by building a VEE Pro program within that program,
- devise a new program that merges your newly created library of UserFunctions into it,
- import and delete the specific libraries that you select, and
- devise a program that will allow you to monitor several tests and note where any given test exceeds previously established limits.

This flexibility will save you many hours of repetitious work.

Generating Lissajous Patterns
Lissajous patterns are generated by applying a reference sine-shaped wave to the X input of an X vs Y display and applying a comparison sine-shaped wave to the Y input of that same display. When the two frequencies are exact integers, then the Lissajous pattern becomes stationary. The frequency ratio is determined by comparing the number of times the pattern is tangent to a horizontal line divided by the number of times the pattern is tangent to a vertical line.

The amount of relative phase shift between the reference wave and the wave being compared changes only the location of the tangent points but not their ratio. The phase shift can be computed from the equation: $\sin \theta = B/A$, where

θ = the phase shift,
B = the value of the Y intercept (where X = 0), and
A = the value of the X intercept (where Y = 0).
The sign of the phase angle depends upon the quadrant in which it occurs.

The Junction (JCT) Object
The Junction Object offers two or more input terminals so the active input to a terminal may be passed on to another object. The program must be such that only one Junction input terminal at a time is activated. It can be used in feedback loops for data initialization.

The Call Object
This object executes a previously defined UserFunction, Compiled Function, or Remote Function.

The If/Then/Else Object
The If/Then/Else Object allows a branching of execution flow based upon the value on its input "If" terminal. The input condition is tested to see if the input-pin information has not been met. If it has not been met, then the Else terminal is activated with the zero value of the input expression. Additional "Else

If' can be added. This action is different from a Comparison Object; the Comparison Object has an output of either a one or a zero.

The Break Object
The Break Object terminates the current iteration loop. It usually operates along with the Until Break Object.

The Until Break Object
This object causes execution to repeat until the Break Object is encountered.

Lesson 15 Pre-Lab

The VEE Pro Sequencer allows you to document tests, compare them against previously developed specifications, and compare test runs with each other.

Accessing logged test data
The Sequencer output is a Record of Records. In the Sequencer record, each test uses the test name as its field name. The fields within each test are named according to your logging configuration.

If you use the default configuration with the fields Name, Result, and Pass, you could access the result in all of the test2 column with the notation Log[*].Test2.Result; see Figure 15.1.

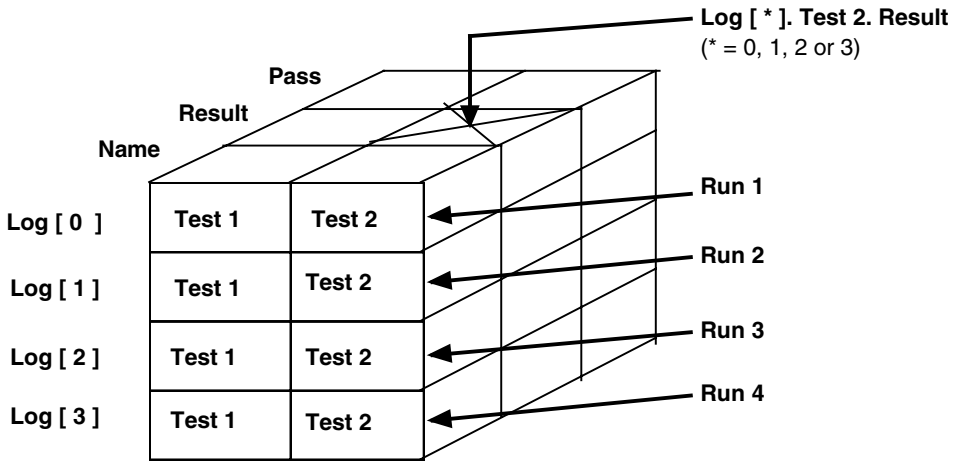


Figure 15.1. A logged array of Record of Records

The Sequence Transaction dialog box
The following fields are available in the Sequence Transaction dialog box:

Sequence Transaction Field	Explanation
TEST:	The TEST: button toggles to EXEC. The TEST: field holds the unique name used to reference this test in the Sequencer. The default names start with test1 and increment with each test.

or:

Sequence Transaction Field	Explanation
	Toggle TEST: to EXEC. The test will be executed without a comparison of the test result value; there is no logging of this test.
ENABLED	ENABLED provides four menu choices: ENABLED executes the test under all conditions. ENABLED IF: executes the test if the stated expression is TRUE; used for audit test control. DISABLED is the opposite of ENABLED. DISABLED IF: is the opposite of ENABLED IF. Toggle the EXEC button to TEST.
SPEC NOMINAL:	This provides the expected value from the test.
RANGE:	RANGE signifies the range of PASS test values. LIMIT uses just one test-data value for a comparison. Available are >, >=, <, <=, ==, !=
TOLERANCE	Tolerance states the passing range of values by adding or subtracting the specified tolerance to the SPEC NOMINAL value. %TOLERANCE % Tolerance states the passing range of values by adding or subtracting a percent tolerance to the SPEC NOMINAL value.
FUNCTION:	The FUNCTION: FIELD specifies which test to run. You may call UserFunctions Compiled Functions Remote Functions or: You can write an expression to be evaluated. Functions can be combined or nested.

Note: Advanced users should refer to the Sequencer Help menu and explore additional variables.

Sequence Transaction Field	Explanation
LOGGING ENABLED	If logging is enabled, then this test logs a record. If logging is disabled, then this test is not logged. For the LOGGING ENABLED mode, you can choose the fields that you want logged if you (a) close the Sequence Transaction dialog box, (b) open the Properties dialog box of the Sequencer Object, and (c) select the Logging tab, choosing from: Name Result Nominal High Limit Low Limit Pass Time Stamp Description Name, Result, and Pass are the default selections.
IF PASS	IF PASS tells VEE Pro to branch, according to your selection, to THEN CONTINUE.
IF PASS CALL:	IF PASS CALL tells VEE Pro to call the specified function (or any expression) if your test passes and THEN CONTINUE.
IF FAIL	IF FAIL tells VEE Pro to branch, according to your selection, and THEN CONTINUE.
IF FAIL CALL	IF FAIL CALL tells VEE Pro to call the specified function (or any expression) if your test passes and THEN CONTINUE.

Note: The fields "THEN CONTINUE" are available for IF PASS, IF PASS CALL, IF FAIL, and IF FAIL CALL.

THEN CONTINUE	THEN CONTINUE executes the next test configured in the Sequencer. THEN RETURN tells VEE Pro 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 Pro 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. THEN EVALUATE: calls the specified User Function (or any expression) which must return a string that states a branching menu option. Valid string results are: "Continue" "Return <expr>" "Goto <name>" "Repeat <expr>" "Error <expr>" where <expr> is any valid VEE Pro expression and <name> is the name of the test in the sequence. This powerful option allows you, via the program, to ask the user what to do next.
DESCRIPTION:	Holds your Text comments; they appear on the Sequencer transaction bar and can be stored with your test record by using the Logging tab in the Properties dialog box.

Declaring Globals

Globals can be declared as a specific type and shape or you can use the SetVariable Object to create an undeclared global.

- Declared variables allow you to set their scope so they will be truly global. They can also be local to a UserFunction, UserObject, or local to a library of UserFunctions.
- Declared variables also allow the VEE compiler to perform type checking; it will run slightly faster.

Using undeclared globals is recommended for the situations where declared variables are not supported, such as a Record of Records.

Lesson 16 Pre-Lab

Data Selection Control
There are six Data Selection Control icons accessed via Menu Bar => Data => Selection Control. See Figure 16.1; each provides a method of accepting operator input where a single value is chosen from a list of possible values. Their display formats vary; however, they all execute in the same manner.

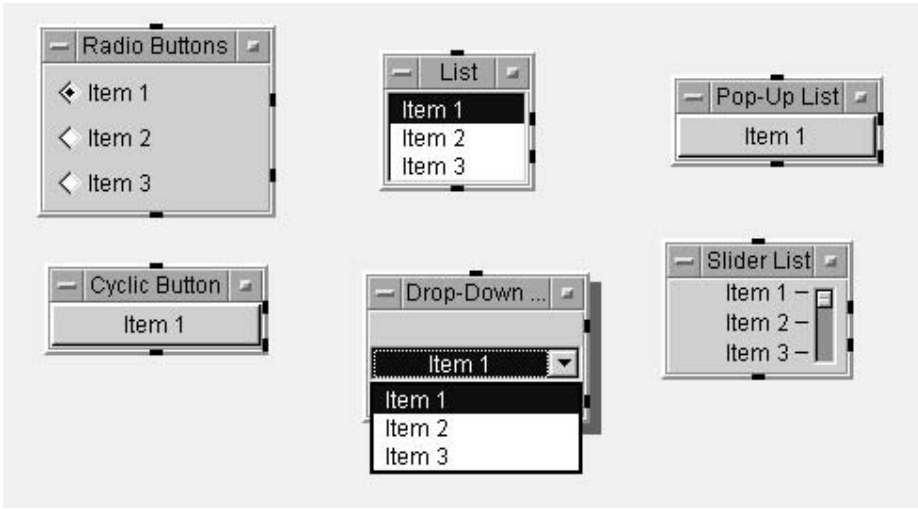


Figure 16.1. The six Data Selection Control objects

Use Selection Control objects either to set a constant or to receive a user input from a list of choices. The first item in the list is assigned ordinal position 0; the nth item in the list is assigned ordinal position n-1. To use a Selection Control as a prompt, change the name of its object to a prompt such as "Make a Selection:" and place it on or add it to an existing user panel.

Each object menu offers **Edit Enum Values**; it allows you to either add valid selections to the enumerated list of choices or edit the names of existing choices. First, add a choice, then press the Tab key to move to the next entry in the list. You can move back up in the list by pressing Shift+Tab or by clicking the mouse to remove the blinking vertical cursor.

The icon image may be changed: Go to the Properties box, select **Icon**, and choose an image from the given list. Two examples are shown in Figure 16.2.

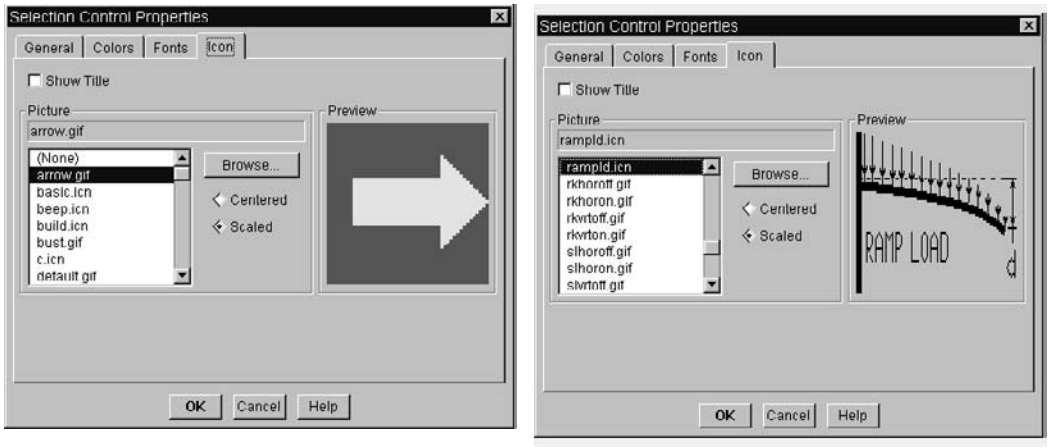


Figure 16.2. Two examples of icon images

In the Properties box, there are four function-option groups:

Format– This list allows the object to be changed to one of the other five Selection Control objects. The Selection Control icon title should also be changed via the Properties box.

Execution – There are two choices:

Wait for Input - When the check box is selected, the object does not operate until the object's value is changed. When the check box is not selected, the object operates as soon as any constraints are met, and the currently selected value is output. Default is off.

Auto Execute - When the check box is selected, the object acts like a Start button and starts the rest of the program when a value is selected. If the program is already running, Auto Execute is ignored. Default is off.

Slider List Layout – The slider layout can be either horizontal or vertical; it sets the orientation of how the slider is displayed. (This selection is only available in the Slider Property box.)

Initialization–Specifies the value to be set. The default value is the first selection from the list. There are two choices:

Initialize At PreRun – When the check box is selected, the Initial Value is set at PreRun time; the default is off.

Initialize At Activate – When the check box is selected, the Initial Value is set at Activate time; the default is off.

Initialize is most often used for initializing values inside a UserObject. The other method for setting an initial value is the Default Value control pin.

Control inputs may be added for the following:

The Default Value input allows you to change the current value. This input requires a Text Scalar value (or any data that can be converted to a Text Scalar, including an Enum). This value must match one of the possible Enum values (case sensitive).

The Reset input allows you to programmatically set the current value to the first (0 ordinal) value in the list. The actual data on the Reset input is ignored.

The Enable Editing and Disable Editing inputs allow you to control whether the values can be changed in the object's open view.

The Enum Values input allows you to set the list of possible values for the Selection Control. This input requires Enum, a Text Array 1D, or any data that can be converted to a Text Array.

Enum Containers are automatically converted into Text for many objects, including Collector, Concatenator, Sliding Collector, and most Formula objects.

Ordinal Value: You may connect a line to the Ordinal output rather than to the selected string, or you can use the **ordinal(x)** function on the Enum.

The six Data Selection Control objects each operate differently:

Radio Buttons – The choices are all displayed. The selection is made when the radio button next to the choice is pressed.

Cyclic Button – The choices cycle as the button field is clicked on. This choice is best used if there are two choices such as On/Off or Stop/Go or in a dialog box where the choice is confirmed by another button.

List – The choices are presented in a list box with scroll bar if all choices cannot be shown at once. The selection is made when the choice is clicked on.

Drop-Down List – The choices are presented in a list box that drops down when the arrow to the right of the field is pressed. The selection is made when the choice is clicked on.

Pop-Up List – The enumerated choices are presented in a list box in a dialog box that is displayed when the button field is pressed. The selection is made when the choice is clicked on and OK is pressed.

Slider List – The choices are presented as "tic" labels on a slider. The selection is made when the slider button or the label is clicked on.

To/From Files may be compared with other To/From Objects:

To/From DataSet – It is a combination of To/From File, To DataSet, and From DataSet. It is simpler to use and less error-prone in retrieving data; however, it is slower than To/From File since more description of field names, field types, and data must be written.

To/From File Read/Write Container – It is simpler to use than To/From File; however, it requires more disk storage space and is slower to execute because of the extra descriptive information required.

Lesson 17 Pre-Lab

Status Panel
A status panel can be created with a UserFunction which has a panel and the built-in functions "hidePanel" and "showPanel". These functions can be accessed from the Function & Object Browser. Select Menu Bar => Device => Function & Object Browser; Type: Built-in Function; Category: Panel. Then select either Show Panel or Hide Panel.

Bitmaps
Bitmaps in the form of pictures (art) may be copied from VEE Pro to other Windows applications, via the Microsoft Windows™ copy/paste clipboard. The applications must support Bitmap format. Microsoft Word™ supports Bitmap. Its applications will be presented later. To copy an object: select one or more objects, choose Copy from the Edit menu, switch to the other application, and choose Paste.

Nested UserFunctions
Nesting in VEE Pro is similar to a DO loop in a FORTRAN program. An example is given in Lab 17.1.

- Independent Panel – A Panel on a UserObject. This method is used when you want to prepare a UserObject for other applications or to simplify segments of your program for other applications without building a Main Panel. See Figure 17.1.

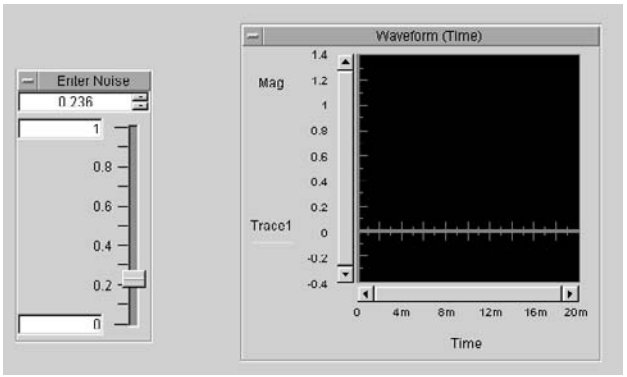


Figure 17.1. An Independent Panel

- Pop-Up Panel – A panel on a UserObject with Show (Panel) on Execute specified via the Properties dialog box. This method is usually used for custom dialog boxes that query for information, and then go away. This type of panel pops up anywhere you desire in a VEE Pro window. See Figure 17.2.

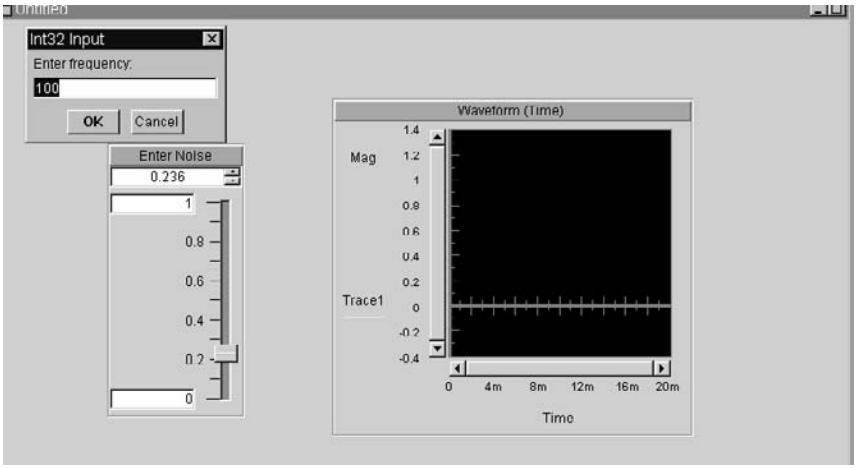


Figure 17.2. The “Enter Frequency” Pop-Up Panel

- A Panel on the Main Panel – A panel on a UserObject that is added to the Main Panel. This method is used to combine subprograms. See Figure 17.3.

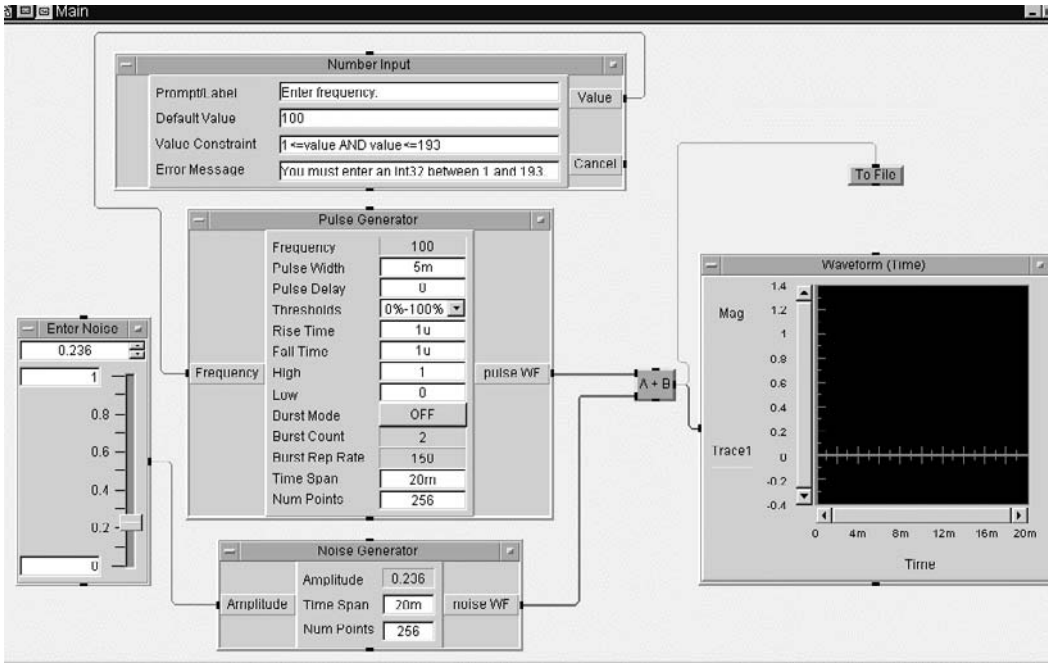


Figure 17.3. A Panel on a Main Panel

Once a UserObject Panel has been added to the Main Panel (or to a parent UserObject Panel), you can still add objects to this panel. However, you must first add these objects to the original UserObject Panel, then adjust sizes and positions of related objects and panels.

Adding Objects to a Nested UserObject Panel – The procedure is to:

1. Click on the Detail button on the left side of the Main Window title bar to place the edit window of the Main Window in its Detail View.
2. Access the edit window of the UserObject either by double clicking on the UserObject in its icon view or, if it is in its restored view, by selecting Menu Bar => Object => Edit UserObject.
3. Click on the Detail button on the upper-left side of the UserObject title bar to place the edit window of the UserObject in its Detail View.
4. Add objects to the Detail View of the UserObject.
5. Highlight the objects you wish to add and click the mouse right button on the UserObject background to get the UserObject Edit menu; select Add To Panel.
6. Resize the panel and rearrange these new objects as desired on the UserObject Panel.
7. Click on the Main Window Panel View button to view the UserObject panel that now contains the new objects; resize the panel if necessary.

Adding a UserObject Panel to the Main Panel:

1. Right-click on the UserObject in the Detail View of the Main Window; select Restore. (The UserObject will expand into the Edit window of the newly created Panel.)
2. Highlight the UserObject to be added to the panel; select Edit; Add To Panel via a right-mouse click. You can nest a UserObject Panel on its UserObject Panel by following the same steps above, except you must use the Edit menu of the identified UserObject.
3. Change its size, location, and other appearance details as desired.
   Note: When you nest a panel on another panel, you are making a copy of its panel contents. The copy is linked to the original panel and is locked. You must make all changes to the original UserObject Panel; the copy is automatically changed.

An Overview of Filter Theory

Filters are the name given to the class of circuits that affect the various frequency components of an input signal in a different manner. Filters are also referred to as "selective circuits" in the literature.

Modern filter (selective-circuit) design requires an understanding of:

- the Transfer Function of a circuit
- the Laplace Transform derivation of that Transfer Function
- the response of a filter to step and impulse input signals
- the frequency-domain response to sine-shaped input signals
- the relation between pulse time delay and sine-wave phase shift

These topics are reviewed below.

Filter Design Procedure

The customary approach to filter design is to

1. Determine the filter transfer function H(s).

The transfer function H(s) is defined as the ratio of the filter output voltage $E_o(s)$ to the filter input voltage $E_i(s)$. It is assumed that there is no loading upon the output caused by the next connected circuit.

$$H(s) = \frac{E_o(s)}{E_i(s)} \quad \text{where } I_o(s) = 0 \qquad \text{eq 17.1}$$

2. Locate all poles and zeros on the "s" plane.

The poles are the denominator zeros; the zeros are the numerator zeros.

3. Convert H(s) to H(jω).

Replace “s” with “jω” everywhere.

$$j = \sqrt{-1} \qquad \text{eq 17.2}$$

$$j^2 = -1 \qquad \text{eq 17.3}$$

$$j^3 = -\sqrt{-1} \qquad \text{eq 17.4}$$

$$j^4 = +1 \qquad \text{eq 17.5}$$

Recall that, from the algebra , these exponents repeat every four integers; the “j” is cyclic.

4. Group separately the real and "j" terms for both the numerator and the denominator.
5. Select component values.
   Choose values that will not load the previously connected circuit.
   Choose values that will not be loaded by the next connected circuit.
6. Compute and graph $|H(j\omega)|$ and $\underline{/H(j\omega)}$.
   The magnitude – $|H(j\omega)|$ – is determined by applying the Pythagorean Theorem separately to the numerator and the denominator if there are both real and imaginary values. The magnitude is the ratio of these two separate computations.
   The angle – $\underline{/H(j\omega)}$ – is the difference between the numerator angle and the denominator angle. Each angle is computed by applying the trigonometric formula:

$$\theta = \text{invtan} \frac{\text{imaginary group of terms}}{\text{real group of terms}} \qquad \text{eq 17.6}$$

   where the proper quadrant for each angle must be included in the solution.
7. Return to H(s) and examine transients, where important.
   Note: Transients prevent correct filter behavior for at least 5τ seconds, where the τ is obtained from the denominator of H(s); it is the largest value of time in the expression.

Filter component-content classifications
Filters are first classified according to the types of components they contain.

a. Passive filters consist of resistors, capacitors, resonators, piezoelectric crystals, and (sometimes) inductors. Passive filters operate best in the mid-audio frequency range. Passive filters that contain inductors may become design-implementation problems because of the inductor physical characteristics. If the resistance of the inductor is to be ignored, then the physical size of the inductor may become too large for the system. If the physical size of the inductor is to be designed to be smaller, then the resistance of the inductor must be included in the design. Then the interwinding capacitance of the inductor may become a problem. Radio-frequency-band (RF) filters consist of small inductors constructed with wire of a very small diameter. The coils are wound so the interwinding capacitance may be ignored. At "radio" frequencies, the inductive reactance of the coil is large when compared to the coil internal resistance.
b. Active filters consist of magnetic amplifiers, transistor amplifiers, or operational amplifiers for the active components; they also consist of resistors and capacitors; they seldom contain inductors unless the active component is a magnetic amplifier. Active filters require at least one power supply for the active components. Feedback connections are usually part of the topology of active filters. Thus an active-filter design may be unstable; it may suddenly oscillate. Active filters operate best from $0 < f < 100$ kHz where this upper value of frequency is increasing annually as operational-amplifier designs improve.

Frequency-range (ideal) classifications and terminology
Filters are next classed according to their effect upon the sine-shaped input frequencies. For ideal filter configurations (topologies) consisting of a large number of (Laplace) poles, either the frequencies are passed with no attenuation, or the frequencies are rejected with maximum ("infinite") attenuation. See Figure 17.4.

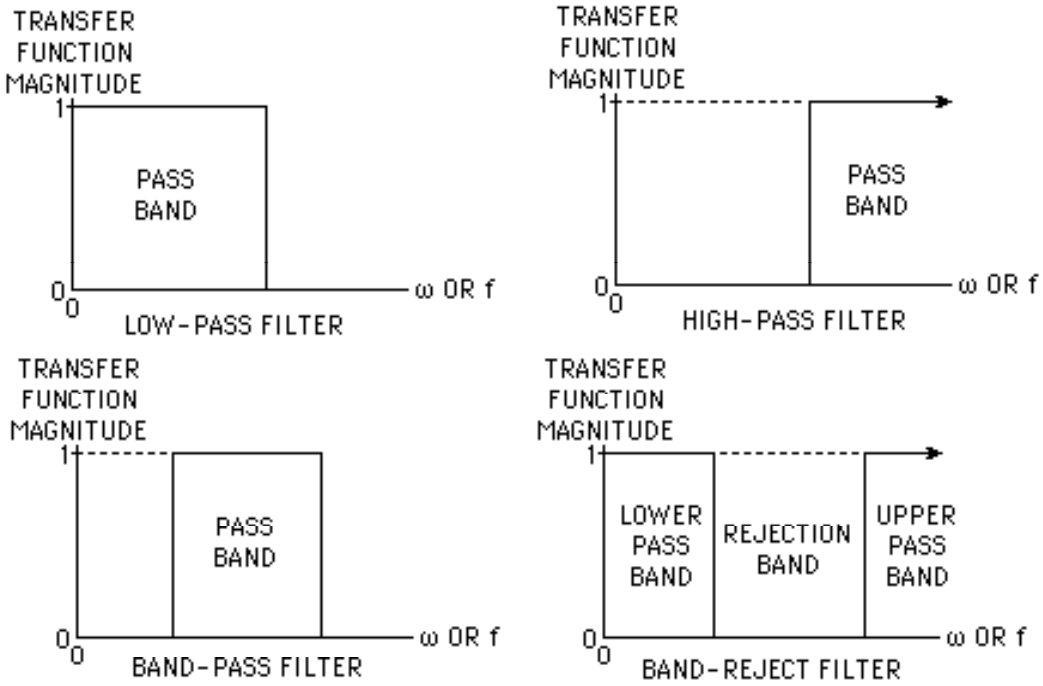


Figure 17.4. Filter ideal transfer functions

At a transition frequency (which is between the pass band and the stop or rejection band), the phase shift for an ideal filter approaches 90°.

On the "s" plane, the frequency classification of filters can be determined and analyzed depending upon the number and location of poles and zeros.

- Low-pass filters consist of “n” poles.
- High-pass filters consist of “n” poles and “n” zeros; the zeros are at the origin.
- Band-pass filters consist of “n” poles and “m” zeros.
- Band-reject (or “band-stop”) filters consist of “n” poles and “m” zeros.

The location of the poles and zeros identifies whether the filter is a band-pass filter or a band-reject filter. The number of poles determines the slope of the transition curve between the “pass” band and the “stop” band.

There are two types of one-pole filters – the low-pass filter and the high-pass filter. These filters are often referred to as “single-pole” filters. There is a gradual transition between the “pass” band and the “stop” band. For single-pole filters, a frequency has been identified for comparison and frequency-identification purposes. This frequency is known by one of four names: half-power frequency, cutoff frequency, corner frequency, or breakpoint frequency. This frequency may be specified in either Hertz (f) or radians per second (ω).

One-pole filters are either high-pass filters or low-pass filters. Examples of the only possible two topologies for one-pole (passive) filters are given in Figure 17.5.

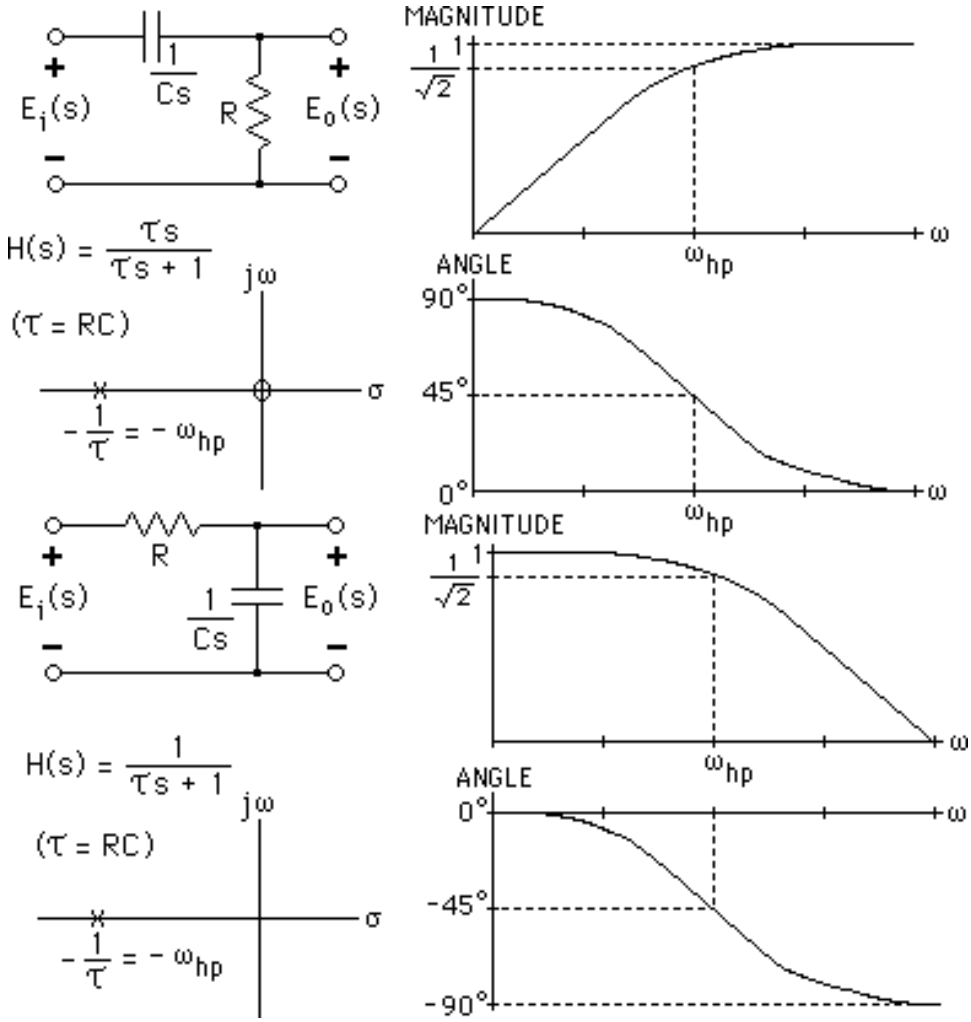


Figure 17.5. One-pole filter options

Note that these one-pole filters can be identified by their Laplace-World descriptions.

- A low-pass filter consists of one pole
- A high-pass filter consists of one pole and one companion zero at the origin

In the frequency domain, the range of the angle is 90 degrees.

Two-pole filters can be low-pass, high-pass, band-pass, or band-reject filters. The only possible four topologies for two-pole (passive) filters are given in Figure 17.6.

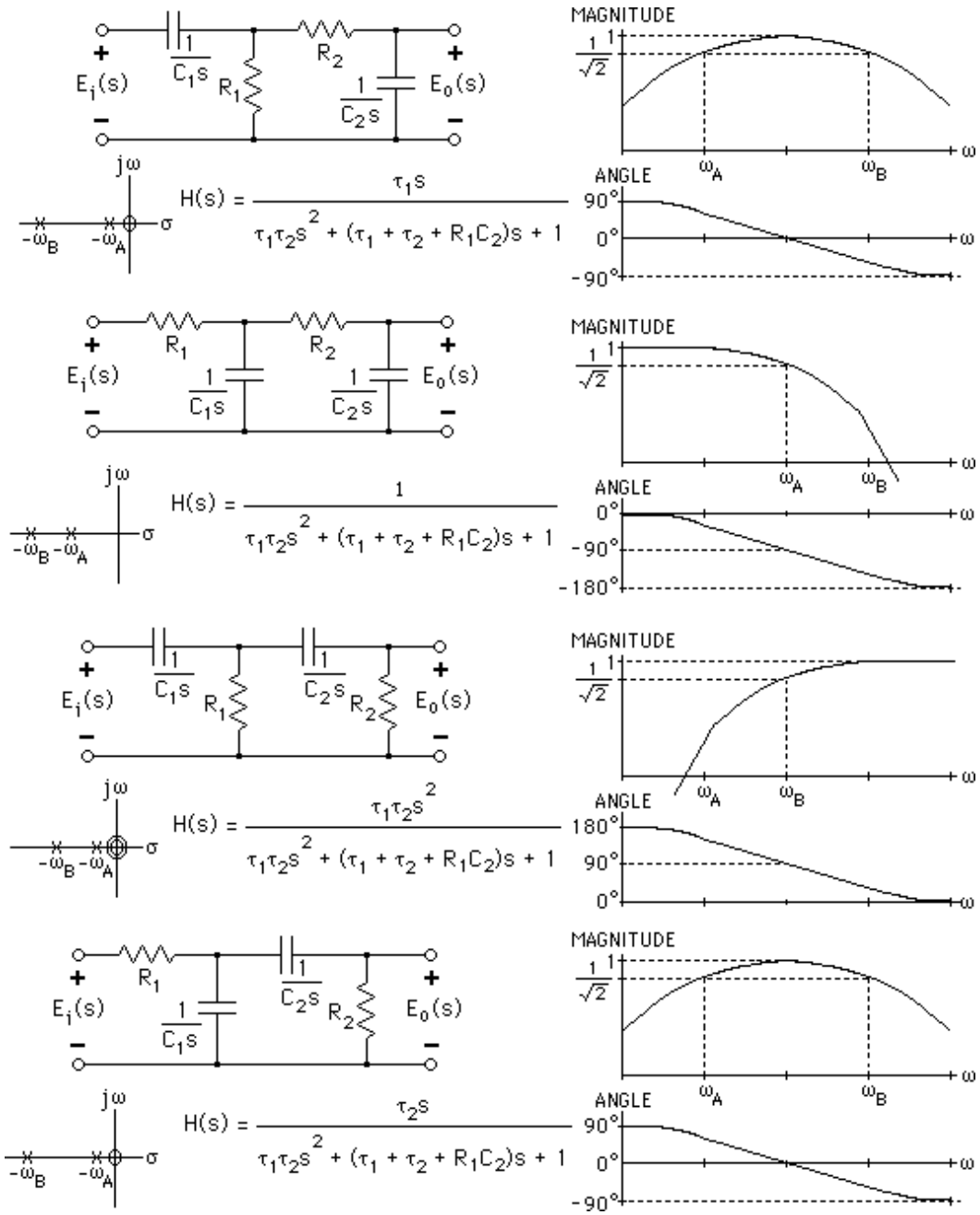


Figure 17.6. Two-pole filter options

Note that these two-pole filters can be identified by their Laplace-World descriptions.

- A low-pass filter consists of two poles.
- A high-pass filter consists of two poles and two companion zeros at the origin.
- A band-pass filter consists of one pole at a low frequency (near the origin), a companion zero at the origin, and one pole at a high frequency.
- A band-reject filter consists of one pole at a high frequency with a companion zero and also one pole at a low frequency.

In the frequency domain, the range of the angle value is 180 degrees for a two-pole filter.

Filters may contain both inductive (L) and capacitive (C) components, or their equivalent – an operational amplifier may be used to simulate an inductor. The math derivations indicate that these filters may have characteristic curves that deviate from the modern conventional RC implementations. This deviation can be explored by examining the solution to the standard quadratic equation for a two-pole filter.

$$as^2 + bs + c = 0 \qquad \text{eq 17.7}$$

where

$$s = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \text{eq 17.8}$$

Whenever the discriminant ($b^2 - 4ac$) is negative, there will be a peak in the magnitude-versus-frequency curve at the "resonant" (ω_0) frequency of the transfer function. This is the frequency at which the angle (or phase shift) of the transfer function is zero.

Filter Specifications

As noted above, a filter can be specified by its number of poles.

- A one-pole filter can be only one of two types – high-pass or low-pass.
- The "sharpness" of its magnitude-versus-frequency curve is gradual when compared to an ideal filter curve.

Increasing the components within a filter can cause the actual filter curve to approach the curve of an "ideal" filter.

The specification of these filters requires additional parameters and their companion definitions.

a. The number of poles and, where applicable, the number of zeros
   The location of these poles and zeros specifies the filter classification and the shape of its magnitude and phase (versus frequency) curves.
b. The half-power (cutoff, corner, or breakpoint) frequency or frequencies
   These frequencies specify a defined gain (or attenuation) level at which one-half the normal pass-band power is transferred from input to output.
c. Attenuation (and gain) levels
   There are three possible attenuation (or gain) specifications.
   1. Absolute Attenuation or Gain
   2. Relative Attenuation or Gain
   3. Attenuation or Gain Floor

d. Pass Band Ripple
   This parameter specifies the amount of allowable deviation of the "magnitude" curve from the desired (or normal) value.
e. Group Delay
   This parameter is the derivative of phase-shift versus frequency.
f. Group Delay Ripple or Tolerance in Group Delay
   This parameter specifies the amount of allowable deviation of the "phase shift" curve from the desired (or normal) value.
g. Step Response
   This parameter specifies the output of a filter to a step input. The step response normally includes deviation amounts or percentages from the specified curve.
h. Impulse Response
   This parameter specifies the output of a filter to an impulse input. The impulse response normally includes deviation amounts or percentages from the specified curve.

Filter Types

There are several types or classifications of filter designs. The filter types have often been given the name of the designer first associated with the design description as documented in the literature. The most common types are

1. Butterworth
2. Chebyshev (Tschebyschev)
3. Bessel (Maximally Flat Time Delay, or MFTD)
4. Cauer Elliptic
5. Switched Capacitor
6. Paynter
7. Thompson
8. Gaussian
9. Equi-Ripple Group Delay

Information regarding these filter types is available in the bibliography.

Specific Filter Designs

Filter designs require investigation of the location of poles and zeros on the “s” plane and the magnitude and frequency response in the frequency domain. Information regarding the Laplace-World “s” plane and the Steinmetz frequency domain (ω) is given below.

Butterworth Filter Design: A Butterworth filter has a relatively flat magnitude response in its frequency-domain pass band. The pole/zero “s” plane, two-dimensional diagram for a four-pole Butterworth filter design has, as its locus of pole locations, a semi-circle.

Chebyshev (Tschebyschev) Filter Design: A Tschebychev filter has a variable-magnitude (ripple) response in its frequency-domain pass band. The pole/zero “s” plane, two-dimensional diagram for a four-pole Tschebychev filter design has, as its locus of pole locations, an ellipse.

Bessel (Maximally Flat Time Delay, or MFTD) Filter Design: A Bessel filter has a smaller variable-magnitude (ripple) response in its frequency-domain pass band than does the Tschebychev filter. The pole/zero “s” plane, two-dimensional diagram for a four-pole Bessel filter design has, as its locus of pole locations, an ellipse.

Cauer Elliptic: A Cauer Elliptic filter has a relatively flat magnitude response in its frequency-domain pass band. The pole/zero "s" plane, two-dimensional diagram for a four-pole, four-zero Cauer Elliptic filter design also has, as its locus of pole locations, an ellipse.
Note: The zeros of the Cauer elliptic filter are no longer at the origin.
Switched Capacitor Filter Design: A switched-capacitor filter utilizes a clock to achieve an ω_0 that is a variable. The value of ω_0 is controlled by a digital "clock" whose frequency can be accurately controlled.

In the 1950's, R. P. Sallen and E. L. Key devised a filter using an operational amplifier (op amp). Such a filter is referred to as an "active" filter because it contains an active component which, in this implementation, is an op amp.

Typically, each desired filter pole requires one op amp. This one-pole-per-op-amp approach allows for much simpler circuit and system analysis. The op amp acts as a circuit isolator. Thus the location and effect of each pole can be computed separately.

Active filters require no inductors, thus saving large amounts of space. Also, active filters usually perform more efficiently and effectively then do their passive-filter counterparts.

A second-order, Sallen-Key, low-pass filter topology requires one (isolating) op amp. This topology is shown in Figure 13.8. Its transfer function is given in the article by Mark Fortunato using the same component notation as shown in Figure 17.7.

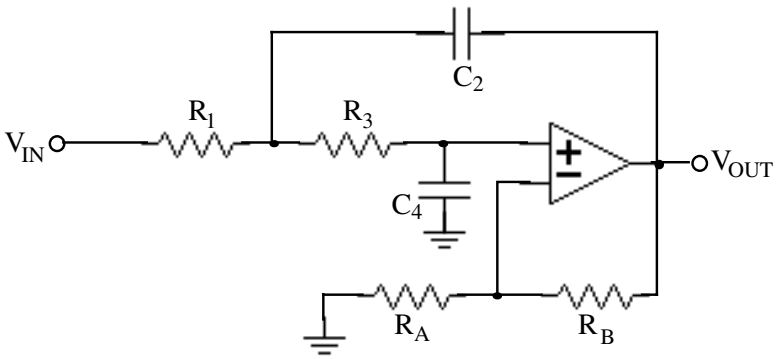


Figure 17.7. A Sallen-Key low-pass filter topology

Resistors with tolerances of 5% can be used with this Sallen-Key topology.

If a high-pass Sallen-Key filter is desired, then resistors R1 and R3 are replaced by capacitors C1 and C3; capacitors C2 and C4 are replaced by resistors R2 and R4. The spread in frequency response for the high-pass filter is much "tighter" than that of the low-pass filter. In practical applications, 1%-tolerance resistors and 5%-tolerance capacitors are used because of their availability. This also keeps the total circuit cost under control.

Active filter circuits can be designed and constructed with ten or more op amps. Because the op amps act as circuit isolators, the circuit analysis requirement seldom exceeds two equations per op-amp stage and often only one equation per stage. The application of one of Kirchhoff's two Laws is all that is required.

The overall transfer function for the filter/op-amp stages is the product of each individual transfer functions derived separately. Thus, the math becomes tolerable and transient analysis can be included. Computer programs that can process the Laplace "s" are not yet available to the designer.

Filters can be designed using easy-to-purchase, off-the-shelf components. Filters can also be purchased as individual components. Because these off-the-shelf designs usually employ inexpensive op amps, they often require two power supplies: +15Vdc and -15Vdc.

It is possible to purchase active filters that require a single (positive) power supply. The filter design can be adapted to the op-amp power-supply requirements that you are using for the remainder of your system. Thus, your spare-parts inventory can be kept to a more reasonable level.

A number of excellent references regarding filters are given in the bibliography.

This completes the pre-labs for the first seventeen lessons. There is no pre-lab for Lesson 18 because it describes how to optimize the labs in the previous seventeen lessons.

Appendix F
Agilent VEE 7.0 Features

VEE 7.0 has some new and unique features. They are described below under the headings: Undo and Redo, Saving Programs, Object Property Editing, Rich Text Format, Visual Studio.NET, Object Property Editing, Panel Edit Features, Using .NET Assemblies, Connectivity for LAN and USB Based Instruments, .NET and IVI-COM Drivers, Error Codes, Terminology, VEE and the .NET Framework, .NET and Communications, and VEE and .NET Security.

Undo and Redo

For Undo, use (Ctrl + Z) or Select Menu Bar => Edit => Undo or click the Task Bar Undo symbol.

One of the most commonly requested features has been Undo. Both Undo and Redo operations will be supported in VEE for certain common operations. Examples include: cut a line, delete a terminal, delete objects from a panel, paste objects, and move an object. Undo/Redo will not be available for every operation in VEE 7.0. The following are available on Undo/Redo:

Panel – Align, delete, and move or group single objects.

- Detail – Cut or paste objects; move, align or delete a line; add, delete or modify a terminal; add or delete a terminal on a UserObject or UserFunction; and autoscroll.
- Undo and Redo taskbar icons have toolkit help that helps you understand what you are undoing or redoing.

You can only undo the most current operations.

For Redo, use (Ctrl + Y) or Select Menu Bar => Edit => Redo or from the Task Bar Redo symbol.
Redo is available only for operations supported by Undo. See Undo and Redo Help menu.

Saving Programs

VEE 7.0 has some new features, such as the Property Grid and Microsoft Visual Studio .NET programmability that cause the VEE 7.0 save format to be unique. Programs saved from VEE 7.0 will **not** load into prior VEE versions.

Object Property Editing (Select Menu Bar => View => Properties)

VEE provides a Properties window where you can easily change properties of your VEE objects. To access the Properties window, click on the Object It also operates when you select multiple objects (providing an "intersection" of properties for the objects if they are different). It works for both the Detail View and Panel View. Any changes made in the Property window in the Panel View affect only the Panel View of that object. It may not be available for all objects. See Figure F.1. See Properties (Object) Help menu for further details.

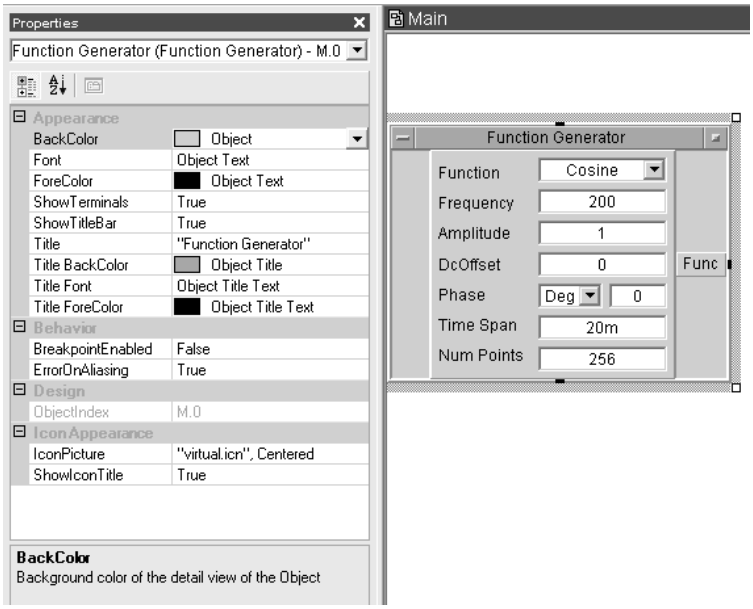


Figure F.1. Properties window

Panel Edit Features

The following are some of the features that have been added: visible grid, alignment of objects, "rubber-band" selection of objects (Ctrl while dragging the objects), undo of most common operations while building a panel, bring to front, send to back, and tab order.

Rich Text Format (RTF) for Note Pad and Description

Using the capability of .NET, RTF has been implemented for the Note Pad and Descriptions. Additionally, the notes and descriptions support automatic hyperlink detection and will start your browser when you click a link. Both of these support Undo (CTRL + Z) and Redo (CTRL + Y) and other editing short-cut

keys. You can cut and paste from the .RTF file directly into a note or description or from the description directly to the .RTF file. Text will word-wrap also.

Visual Studio.NET and the .NET Framework-Based Class Library

Visual Studio .NET is Microsoft's latest development environment. It addresses the needs of both programmers and World Wide Web developers. From a VEE perspective, the strength of .NET is the Framework Class Library. The .NET Framework is distributed with VEE 7.0. There is a description of .NET and the .NET Framework at:

http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx.

The CLR provides common services for .NET Framework applications. Most languages are supported by the CLR. Many time consuming and awkward programming tasks are performed automatically by the CLR. Examples include: memory management, exception handling, and security management.

The .NET Framework also includes a common class library. These object-oriented classes are integrated into the CLR. The standard class library is an asset from which your managed code can derive functionality; the standard type library brings consistency across programming languages. Some of the types available help with tasks such as file and string management.

.NET provides a native COM interop technology. This means that a COM component can be accessed as a .NET object and a .NET object can be accessed as a COM component.

Using .NET Assemblies

For access to .NET capabilities, VEE 7.0 provides sources of .NET classes:

- The Microsoft .NET Framework Library and
- Third-party .NET libraries and .NET classes.

See the .Net (DotNet) examples directory to explore the power of the .NET Framework.

An additional feature is its ability to import namespaces. From the Device => .NET Assembly References menu, mark the "Import namespaces after closing" checkbox. (To use an Assembly, you must add a reference to it.) Choose one or more .NET Assemblies; you will be prompted to select zero or more namespaces. You can also choose Import .NET Namespaces from the Device menu. This avoids performing a namespace import; System.IO.File.Exists() simplifies to File.Exists() when you import System.IO. With a namespace import, your method becomes File.Exists() which reduces typing.

There is a new "type conversion" function known as "asClrType". This function converts a VEE data type to a .NET/CLR data type. These data types include primitive, decimal, DateTime, and String. See examples\datetime.vee in the Agilent VEE Examples programs.

Connectivity for LAN and USB Based Instruments

The VEE I/O subsystem continues to be enhanced to allow for interaction with devices (T&M Industry standard) that are connected via LAN (VXI-II) and the Universal Serial Bus (USB). These connectivities are becoming standard throughout the industry.

.NET and IVI-COM Drivers

IVI-COM is a new standard for instrument compatibility. Established by the IVI Foundation, it encourages instrument manufacturers to design instruments that are interchangeable with their competitors. The IVI Foundation is a consortium established to promote specifications for programming test instruments. For a review of the IVI Foundation, go to http://www.ivifoundation.org/.

To adapt to these new standards, VEE accepts all Agilent IVI-COM drivers as well as third-party IVI-COM drivers. VEE interacts with IVI-COM drivers by finding a Primary Interop Assembly (PIA). An assembly is an .EXE or DLL file and is the fundamental building block of a .NET application. The PIA is a .NET assembly provided by vendors for their COM-based components. Because the VEE IVI-COM driver is a COM-based component, VEE will either search for a PIA or attempt to create one. You can then use the IVI-COM driver just like any other .NET assembly. They will eventually be integrated into the VEE Instrument Manager.

To obtain drivers for your instruments, go to http://www.agilent.com/find/adn. If you are an ADN member, select IVI-COM and VXI*plug&play* drivers can be located in the Downloads area of ADN (Agilent Developers Network). If you are not an ADN member, it is an online source for Agilent drivers, evaluation software, documentation, and white papers. To sign up for this free service, fill out the registration form for new users. Once the IVI-COM drivers are installed and registered on your system, they are available just like any other COM component and, like most COM components, will appear on the COM tab of the Device = > .NET References in VEE. Choose your driver from the assemblies by using the Function & Object Browser with the COM tab. This includes choosing the driver class; constructing the driver in the VEE program; initializing it; and using the properties, events, enumerations, and methods associated with it[0] This is only true for drivers that support the IVI Class Compliant interface. IVI-COM drivers can have two interfaces: the IVI COM Class Compliant and the Instrument Specific Interface.

The Class Compliant one is defined by the IVI Foundation and should work with any instrument of the same type. The instrument specific interface will only work with one specific instrument. The instrument specific interface would support more features available in the specific instrument. For example, if two vendors provide an IVI-COM driver for a DMM to meet the IVI standard, the base drivers must be interchangeable. This means that whichever driver you use on your system, the DMMs from either vendor will work. Any vendor's DMM will work using the class compliant interface; the Agilent DMM will work with the Agilent driver instrument specific interface. Your code is automatically reusable. Via .NET's interop technology, the IVI-COM drivers become available to VEE which then adds the Assembly Reference to the Function & Object Browser's .NET Object Assembly References list.

Error Codes

All VEE 7.0 error codes are available on the Internet and can be downloaded from http://www.agilent.com.

Terminology

Assembly. Microsoft defines an Assembly as follows: “Components are packaged in assemblies. Assemblies are the reusable, versionable, self-describing building blocks of .NET applications. The simplest assembly is a single executable that contains all the information necessary for deployment and versioning. An Assembly is the fundamental building block of the .NET Framework. It takes the form of an executable (.exe) or dynamic link library (.dll) file.”

Primary Interop Assembly (PIA). Microsoft defines a PIA as follows: “A primary interop assembly is a unique, vendor-supplied assembly that contains type definitions (as metadata) of types implemented with COM. There can be only one primary interop assembly, which must be signed with a strong name by the publisher of the COM type library.”

Namespace. Microsoft defines a Namespace as follows: “.NET Framework types use a dot syntax naming scheme that connotes a hierarchy. This technique groups related types into namespaces so they can be searched and referenced more easily. The first part of the full name - up to the rightmost dot - is the namespace name. The last part of the name is the type name. For example, System.Collections.ArrayList represents the ArrayList type, which belongs to the System.Collections namespace. The types in System.Collections can be used to manipulate collections of objects.”

Shared Member. Microsoft defines shared members as follows: “Shared members are properties, procedures, and fields that are shared by all instances of a class. Some programming languages refer to such items as static members. Shared fields and properties are useful when you have information that is part of a class, but is not specific to any one instance of a class. Normal fields and properties exist independently for each instance of a class. Changing the value of a field or property associated with any one instance does not affect the value of fields or properties of other instances of the class. On the other hand, when you change the value of a shared field and property associated with an instance of a class, you change the value associated with all instances of the class. In this way, shared fields and properties behave like global variables that can be accessed only from instances of a class.”

Class. Microsoft defines a Class as follows: “If you are familiar with object-oriented programming, you know that a class defines the operations an object can perform (methods, events, or properties) and defines a value that holds the state of the object (fields). Although a class generally includes both definition and implementation, it can have one or more members that have no implementation. An instance of a class is an object. You access an object's functionality by calling its methods and accessing its properties, events, and fields.”

Instance Member. An instance member is tied to a specific instance of a class. Changes to its value affect the object it is associated with and no other objects.

VEE and the .NET Framework

Before beginning a review of how the .NET Framework interacts with VEE, review the .NET Framework by looking up the ".NET Framework – getting started" in the MSDN index.

Use

Under the VEE 7.0 Device menu there is a new menu option named .NET Assembly References. As you will remember from the definitions, a reference allows you to use an Assembly. Open this menu selection. You will see a window as shown as Figure F.2.

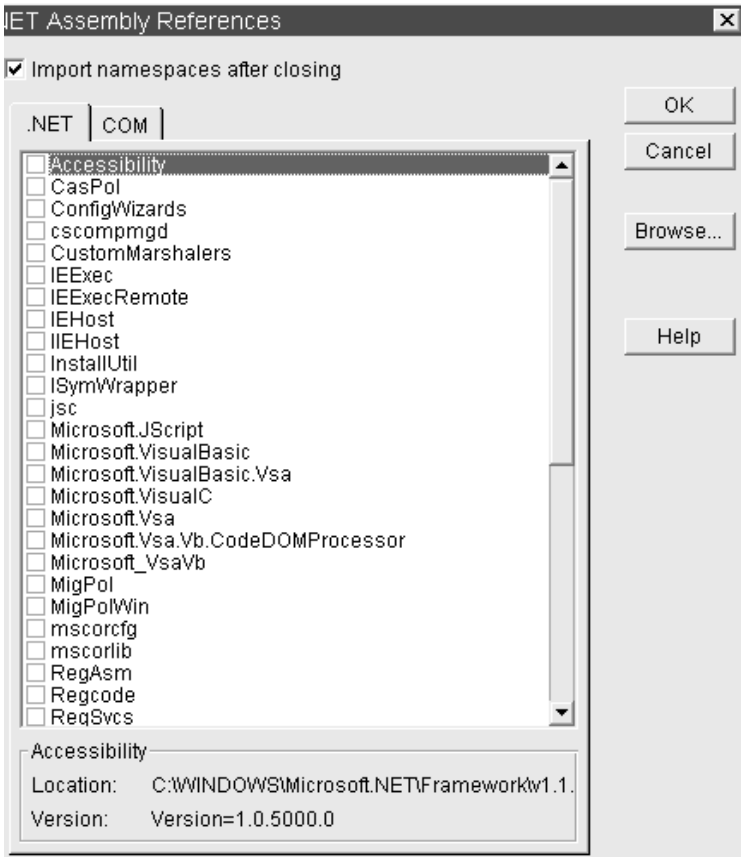


Figure F.2. .NET Assembly References menu

There are a great number of References you can select; the most commonly used Assembly, **mscorlib**, is selected. Select this reference. Consult your MSDN documentation to discover the functionality contained within this and other assemblies.

There is another tab labeled COM where there is a list of all the COM type libraries currently registered with your computer. (This is the same list of type libraries you see when you use the ActiveX automation references dialog box.) Use ActiveX if the library that appears is fully functional and has no Primary Interop Assembly (PIA). This avoids adding another layer, the .NET Interop when using ActiveX. Use the .NET / COM Interop explained in this topic if

- the library does not appear under ActiveX, or
- the functionality is not all imported, or
- the COM library has a PIA

For example, IVI-COM drivers do not implement “Idispatch” and, therefore, do not appear under the VEE ActiveX list. However, they can be imported under COM interop via the .NET Assembly References menu.

When using the .NET Assembly References menu option and selecting a COM type library from the list, VEE tries to import the COM type library as a .NET assembly. If there is a PIA, then VEE uses it. Otherwise, VEE automatically creates an Interop Assembly (IA) or assemblies (if the COM type library references other type libraries) in the directory where the program is saved. After VEE selects the IA, it receives all the type information from that assembly. Finally, select the Microsoft ActiveX Plugin.

From the Device menu, open the Function & Object Browser. One of the Type: menu selections is .NET/CLR Objects. Select this option. Note how the Assemblies window reflects the two selections made earlier (Interop.ActiveXPlugin and mscorlib). See Figure F.3 for an example of the Function & Object Browser window Type content.

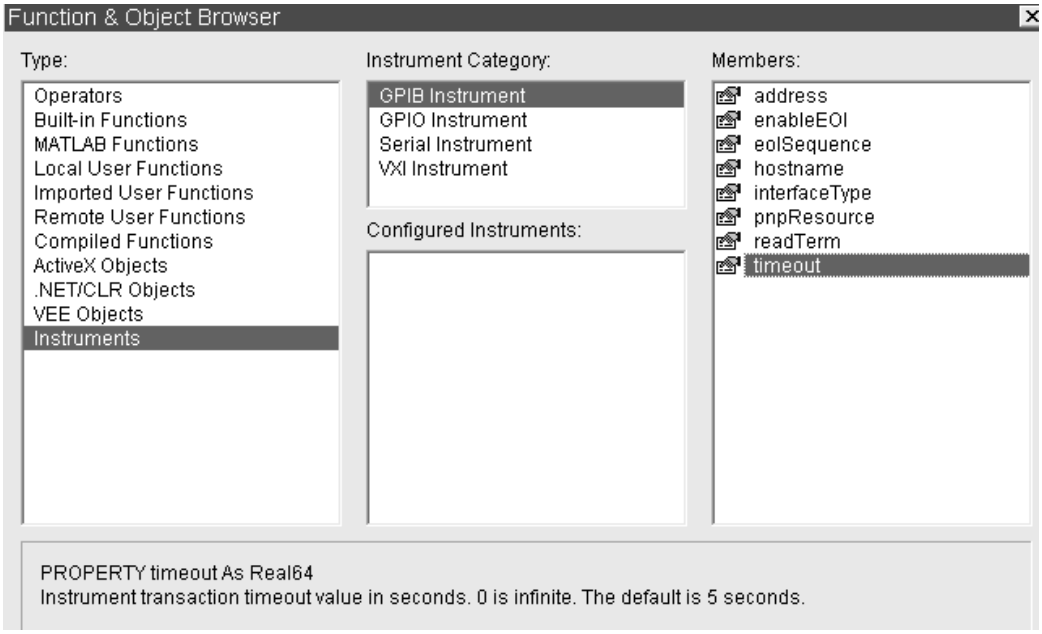


Figure F.3. Function & Object Browser window Type content

All of the classes are listed in the Type window; all the members of the each class are listed in the Members window. Members include properties, methods, constructors (.ctor), enumerations, and events.

For the .NET/CLR Objects type, the individual windows change to Assembly, Namespace, Type, and Members.

If you have used ActiveX in VEE, you will find that selecting a .NET class member is essentially the same process as with ActiveX in VEE 6. The differences include:

1. COM did not have any shared (static) members. If the .NET member is a static member, the help area shows the STATIC keyword.
2. To create a .NET object, select constructor in the member list. As shown in the previous graphic, the Create Instance button becomes available. Selecting this button creates a new Formula Object; the .NET object is not created until the Formula Object executes.
3. You can no longer substitute an integer constant for Enumin .NET the way you could with ActiveX objects. You have to enter the full Enum name or use the Create button; it will save you some typing.

.NET and Communications (COM)

Note that.NET objects and COM objects exist in memory in different ways:

- .NET objects exist in managed memory, which is controlled by the Common Language Runtime (CLR); they can be moved during runtime as needed to improve performance.
- COM objects reside in unmanaged memory; they point to locations in memory that do not move.

Because the two manage memory differently, a wrapper is needed to help them communicate with each other. Figure F.4 shows how .NET makes use of wrappers to make the communication between the two transparent. These two wrappers are the foundation of the .NET Framework's COM interop strategy. For in-depth and understandable reviews of the Runtime Callable Wrapper and the COM Callable Wrapper, see the articles in MSDN by Mike Gunderloy. They are entitled “Calling COM Components from .NET Clients” and “Calling a .NET Component from a COM Component”.

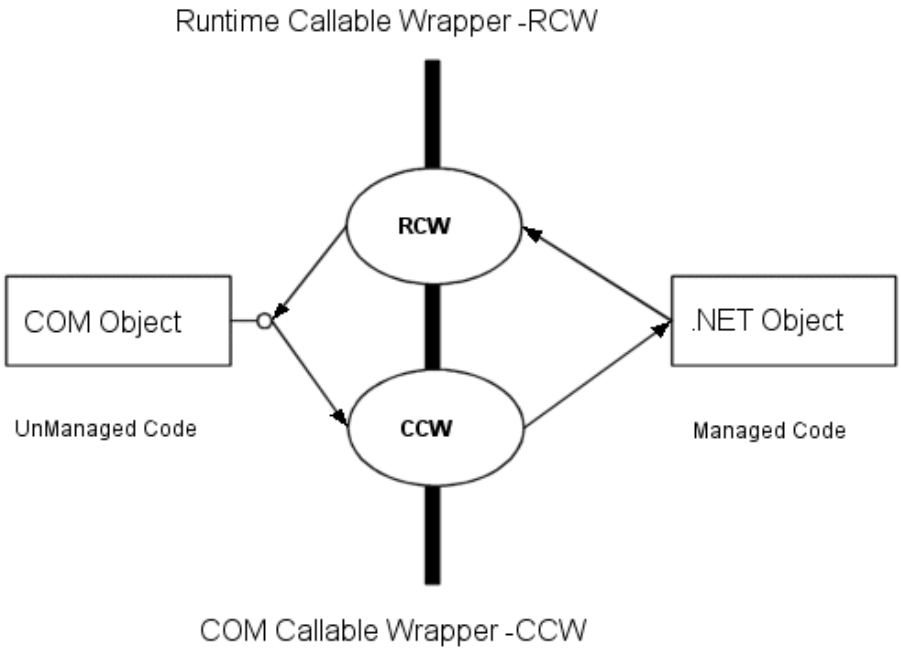


Figure F.4. COM Callable Wrapper (CCW)

VEE and .NET Security

There are security issues to consider if you install VEE on a network drive. The simplest way to address this issue is to open a command prompt. Go to your system drive (usually C).

- Change directory to \[winnt or windows]\Microsoft.NET\Framework\v1.1.4322
- From the prompt, run the following command: Caspol –machine –addgroup "All_Code" –url File://[VEEInstallDir]/* FullTrust

The VEEInstallDir is your VEE installation directory on a network drive. Caspol is included with VEE 7.0. If your VEE program references .NET assemblies, VEE often needs to generate and/or save assemblies to an accessible location. For example, if you open a VEE program with .NET references from email or a network without saving locally first, you might get a warning.

Bibliography

VEE Pro Reference Manuals

Agilent Staff, Advanced Programming Techniques, Agilent Technologies Inc, 2000

Agilent Staff, Agilent IO Libraries Installation and Configuration Guide for Windows, Agilent Technologies Inc, 2000

Agilent Staff, User's Guide, Agilent Technologies Inc, 2000

Technical Papers

Angus, R. B., and Hulbert, T. E., Objectives to Outcomes and Back, Flexible Automation & Intelligent Manufacturing (FAIM) Conference, Dublin, Ireland, 2001

Costlow, T., Robotic Vehicles Will Aid Military Technology, IEEE-USA News and Views, March, 2004

(See also www.todaysengineer.org/Nov03/darpa.asp)

Hulbert, T. E., and Angus, R. B., Preparing a Virtual Engineering Environment Laboratory Instructional Package, ASEE Annual Conference Proceedings, June 2002

Hulbert, T. E., Hansberry, E. W., and Angus, R. B., Just-In-Time Education™: An Idea Whose Time is Overdue ASEE Annual Conference Proceedings, June 1996

Mannix, M., The Virtues of Virtual Labs, ASEE Prism, September, 2000

Periodic Waveforms and Their Fourier Series

Tilton, H. B., Waveforms: A Modern Guide to Nonsinusoidal Waves and Nonlinear Processes, Prentice-Hall, 1986, ISBN0-13-946096-9

Wei, Y., and Zhang, Q., Common Waveform Analysis: A New and Practical Generalization of Fourier Analysis, 2000, Kluwer Academic Publishers, ISBN0-7923-7905-5

An Overview of Filter Theory

Budak, A., Passive and Active Network Analysis and Synthesis, Waveland Press, Inc., 1991, ISBN0-8813-3625-4

Fortunato, M., Reduce Component Sensitivity in Single-Op-Amp Filters, Electronic Design, Analog Applications issue, 1998 June 22, pp10-20

Foster, C. C., Real Time Programming – Neglected Topics, 1981 Edition, Addison-Wesley Publishing Company, ISBN0-201-01937-X for the Sampling Theorem

Ghausi, M. S., and Laker, K. R., Modern Filter Design: Active RC and Switched Capacitor, Noble Publishing, 2003, ISBN1-8849-3238-X

Goldsbrough, P., Lund, T., and Rayner, J., Analog Electronics for Microcomputer Systems, SAMS – The Blacksburg Group, First (1983) Edition, ISBN0-672-21821-6

Heathkit/Zenith Educational Systems, Active Filters, 1979 Edition, Prentice-Hall, ISBN0-13-002063-X

Huelsman, L. P., and Allen, P. E., Introduction to the Theory and Design of Active Filters, McGraw-Hill, 1980, pp156-157, ISBN007-030859-3

Irvine, R. G., Operational Amplifier Characteristics and Applications, Third (1994) Edition, Prentice-Hall, Inc., ISBN0-13-606088-9
Note: Pages 235-323 include excellent three-dimensional graphs

MAXIM Engineering Journal, Volume 2, A Primer on Switched-Capacitor Filters, Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086, tel: (408) 737-7600

Meiksin, Z. H., Complete Guide to Active Filter Design, Op Amps, & Passive Components, 1990 Edition, Prentice-Hall, Inc., ISBN0-13-159971-2

Millman, J., and Grabel, A., Microelectronics, McGraw-Hill Book Company, Second (1987) Edition, ISBN0-07-042330-X
Note: See pages 750-752 on switched capacitor filters

Sedra, A. S., and Smith, K. C., Microelectronic Circuits, Oxford University Press, Fourth (1997) Edition, ISBN0-19-511663-1
Note: See pages 803-809 on switched capacitor filters

Sallen, R. P., and Key, E. L., A Practical Method of Designing Active Filters, IRE Transactions on Circuit Theory, Vol.CT-2, 1955 March, pp74-85

Stanley, W. D., Operational Amplifiers with Linear Integrated Circuits, Prentice-Hall, Fourth (2002) Edition, ISBN0-13-032013-7

Stephenson, F. W., RC Active Filter Design Handbook, 1985 Edition, John Wiley & Sons, Inc., ISBN0-471-86151-0

Steer, R. W., and Wing, C. G., An Introduction to Frequency Devices, Frequency Devices, Inc., Haverhill, MA 01830, tel: (508) 374-0761

Young, T., Linear Systems and Digital Signal Processing, 1985 Edition, Prentice-Hall, Inc., ISBN0-13-537366-2

Zuch, E. L., Data Acquisition and Conversion Handbook, 1979 Edition, Opamp Technical Books, ISBN0-9602-2946-0-0