

# **LabVIEW™ Basics I**

## **Course Instructor Manual**

**March 1998 Edition  
(Course Software Version 4.0)**

**Part Number 320804E-01**

### **Copyright**

Copyright © 1994, 1998 by National Instruments Corporation, 6504 Bridge Point Parkway, Austin, TX 78730-5039. (512) 794-0100. Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

### **Trademarks**

LabVIEW™ and The Software is the Instrument™ are trademarks of National Instruments Corporation.

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

**National Instruments Corporate Headquarters**

6504 Bridge Point Parkway

Austin, TX 78730-5039

(512) 794-0100

Fax-on-Demand Support: (512) 418-1111

**Branch Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,  
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725  
11, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092,  
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466,  
Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70,  
Switzerland 056 200 51 51, Taiwan 02 377 1200, United Kingdom 01635 523545

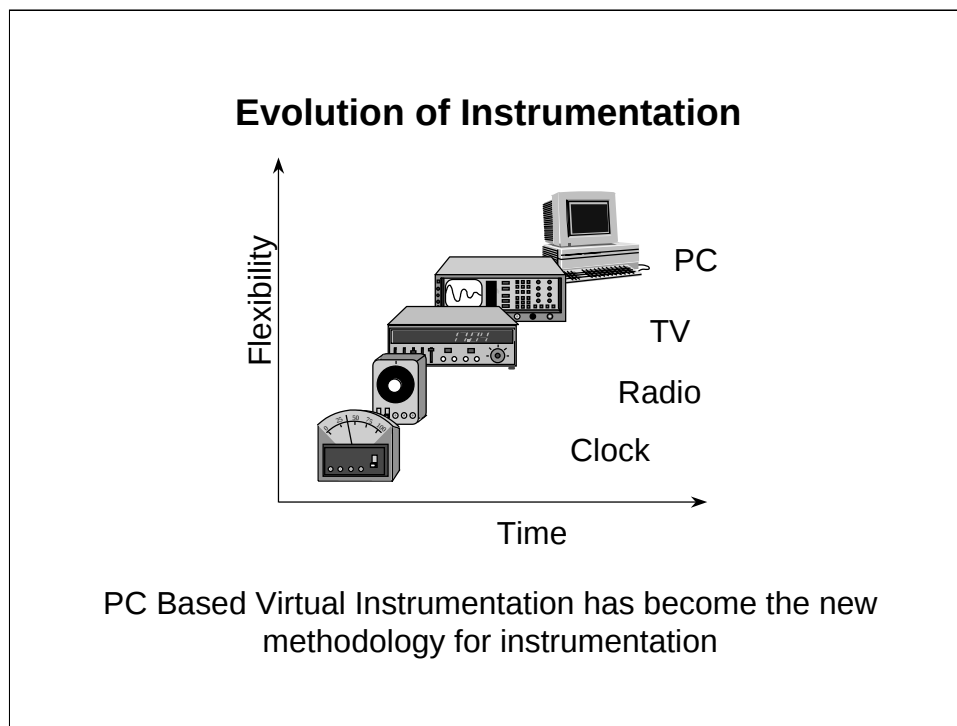
# **LabVIEW™ Basics I Course**

**National Instruments  
6504 Bridge Point Parkway  
Austin, Texas 78730-5039  
(512) 794-0100**

Display this slide when students start arriving so that they know they are in the correct place.

When you begin the class, make sure to cover the following points:

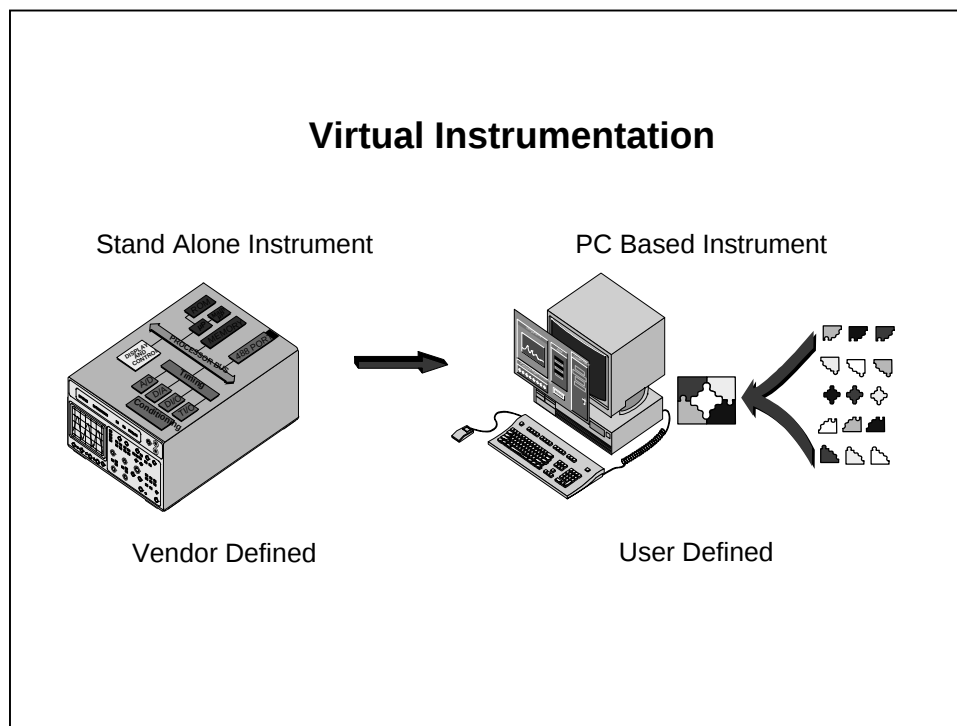
- Introductions - Introduce yourself and give your credentials. Ask the students to introduce themselves to the class, describe what kind of applications they will be using LabVIEW for, and tell how long they have worked with LabVIEW (have they done the Tutorial?) This information helps you pace the course and know what concepts to stress.
- Discuss when class begins and ends each day. Also, explain lunch plans and other breaks.
- Explain where the restrooms, phones, and smoking areas are located.
- Poll the students about their hardware interests. Then you will know how much time to spend on the DAQ and Instrument Control lessons. You may also want to encourage the students to sit near others who have the same hardware or type of application.



Instrumentation helps science and technology progress. Scientists and engineers around the world use instruments to observe, control, and understand the physical universe. Our quality of life depends on the future of instrumentation – from basic research in life sciences and medicine to design, test and manufacturing of electronics, to machine and process control in countless industries.

The slide above shows the evolution of instrumentation over the last 100 or so years. It is important that instruments have always leveraged off widely-used technology. In the 19th century, the jeweled movement of the clock was first used to build analog meters. In the 1930s, the variable capacitor, the variable resistor, and the vacuum tube from radios were used to build the first electronic instruments. Display technology from the television has contributed to modern oscilloscopes and analyzers. And finally, modern personal computers contribute high-performance computation and display capabilities.

In the past, data acquisition devices have been traditionally reserved for the stand alone instrument. As you will soon see, advancements in the personal computer area have shifted data acquisition to the personal computer, which offers greater flexibility, ease of use, and cost-effective analysis.



Instruments have evolved in terms of both flexibility and the degree to which they integrate into systems. The first-generation instruments were analog instruments manually controlled from their front panels. Measurements from these instruments had to be recorded by hand. The users had no flexibility in user interface design, measurement capabilities, or computational features.

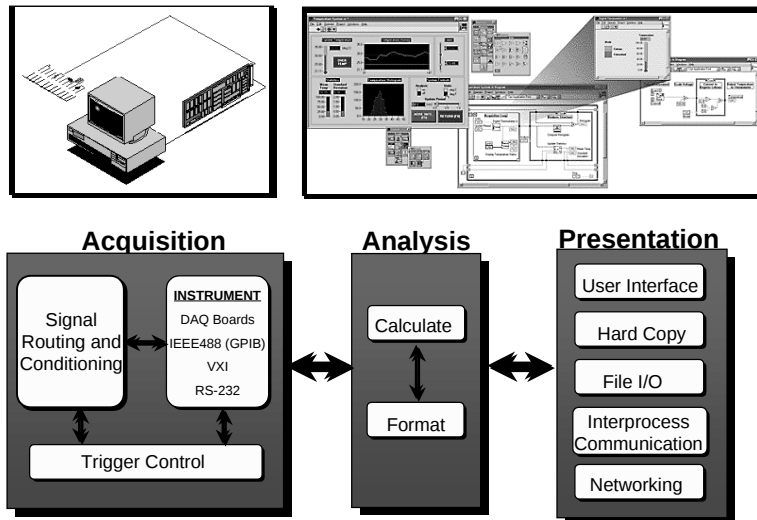
With the invention of the GPIB and digital instruments, users could control systems both programmatically and manually. Each GPIB instrument was designed for a specific measurement, and users “racked and stacked” a number of instruments to create a complete measurement system.

Now millions of scientists and engineers around the world use PCs to automate their research, design, and manufacturing tasks. Increased performance and capabilities of PCs now rival those of sophisticated workstations at a fraction of the cost. Easy-to-use but powerful software and application development environments allow you to develop more capable applications faster.

With the advent of the PC, a new term has become popular in the instrumentation community. The term “virtual instrument” describes the combination of programmable instruments with general-purpose PCs.

Historically, architectural limitations have resulted in hard boundaries between vendor-defined instruments and user-defined functionality in an instrumentation system. The PC revolution has equipped users with powerful processing and display capabilities of their own. If you enhance the fixed capabilities of a traditional instrument with flexible user-defined capabilities of a computer, you have a virtual instrument. The computer enhances instrument functionality in any of the three element areas—data acquisition, analysis, and/or presentation.

## Key Elements of Virtual Instruments

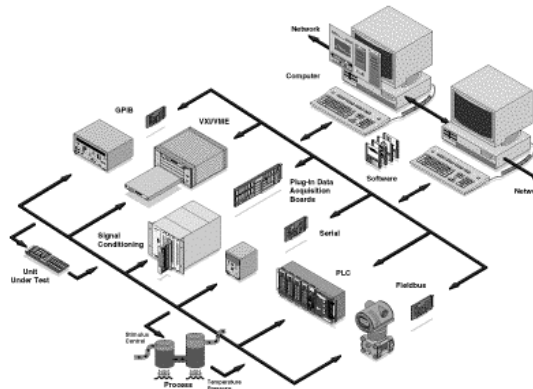


A virtual instrument consists of three key elements: acquisition, analysis, and presentation. Acquisition is the means by which physical signals (such as voltage, current, pressure, temperature) are converted into digital formats and brought into the computer. Analysis, such as a frequency response, is performed by specifically optimized software routines available in a wide selection of development environments. The presentation of the acquired and analyzed signal is accomplished using graphical display methods common on computers today.

The four popular methods for acquiring data are plug-in data acquisition boards, GPIB instruments, VXI instruments, and RS-232 instruments. Data analysis transforms raw data into meaningful information. Data presentation is the means for communicating with your system in an intuitive, meaningful format. The components listed are but a few examples of the data presentation tools available.

## The Virtual Instrument

Industry-standard  
components  
Flexible  
Scalable  
Connectivity  
Compatibility  
Increased productivity  
Reduced cost



With virtual instrumentation, you can build your own instrumentation systems with standard computers and cost-effective hardware. These software-centered systems leverage off the computational, display, and connectivity capabilities of popular computers to give you the power and flexibility to build each of your instrumentation functions. You can mix and match your choice of data acquisition and instrument control hardware, including all of of your existing instruments, to create virtual instrumentation systems that exactly meet your needs.

Virtual instrumentation saves you time and money because you can build “user-defined” systems in a fraction of the time it takes with traditional approaches. By combining LabVIEW with standard data acquisition and instrument control devices, you can create virtual instruments and use them in many applications. Unlike traditional instruments, which are limited by the design of the manufacturer, a LabVIEW virtual instrument can operate a variety of devices, such as a temperature monitor, voltmeter, strip chart recorder, digitizer, and signal analyzer.

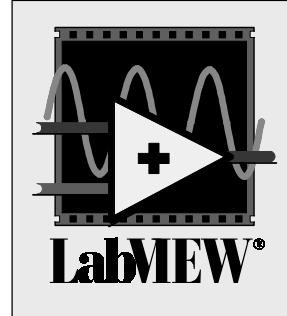
## **LabVIEW**

### **The Industry-Standard Virtual Instrumentation Software**

#### **Graphical Programming for Virtual Instrumentation**

- front panel graphical user interfaces
- graphical block diagram source code
- compiler for optimized execution

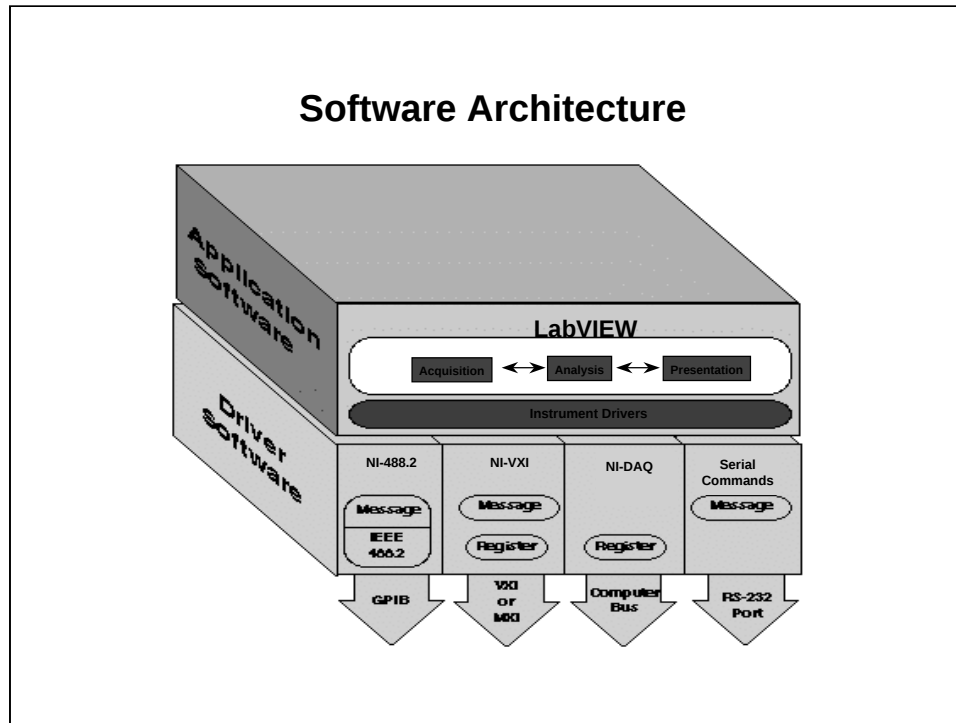
**Standalone executables for easy  
distribution**



LabVIEW is an easy-to-use, powerful graphical programming application that also provides a compiler with the ability to create standalone executable applications.

LabVIEW is ideal for creating virtual instruments. It features a well-defined strategy for constructing both hardware and software modules that are easy to understand and maintain. You can rapidly combine, interchange, and share virtual instruments to build custom applications. Different hardware acquisition options can feature drop-in replacement virtual instruments for truly modular application development.

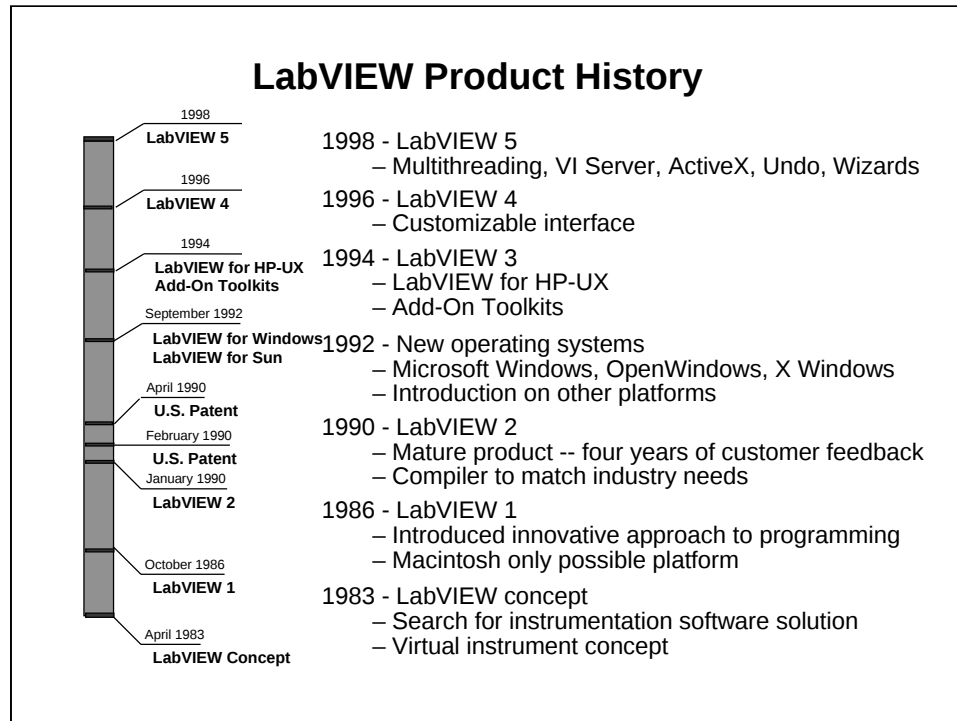
## Software Architecture



To account for the varying needs of measurement and instrumentation applications, software must integrate all of the elements in a manner that delivers ease of use, power, and flexibility. National Instruments software architecture combines low-level device drivers for our hardware and high-level application development software. With this combination, you have a scalable development system with the flexibility to handle a very wide range of measurement and instrumentation applications.

To build virtual instruments, you need software that easily integrates any or all of the hardware instrumentation options in a single system. At the lowest level, with standardized driver software packages you can control the four acquisition hardware options. The driver packages include function libraries for programming your hardware. Our NI-488.2 driver is the industry standard for programming GPIB. NI-VXI is our driver software for programming VXI. NI-DAQ is our driver software for programming our plug-in data acquisition boards. Most computers include RS-232 driver software. NI-VISA is a new standard for combining the different instrumentation systems. Our driver software packages are standardized across all popular computers and a wide variety of operating systems.

You can use our stand-alone driver software to program using the language and operating system of your choice. If you need assistance with programming, our application software is the answer. With our application software, LabVIEW, you can build your own instruments. LabVIEW works with all aspects of an instrumentation system, including data acquisition and control, data analysis, and data presentation. LabVIEW also features modular, reusable, high-level instrument drivers for specific instruments and applications that dramatically reduce your software development task. By using our application software packages, you are free to choose the components necessary to meet your application cost and performance requirements. You now have the opportunity to build your own instruments to exactly meet your particular application needs.



In 1983, National Instruments began its search for an instrumentation software solution that would minimize the time needed to program instrumentation systems. Through its effort, National Instruments developed the LabVIEW virtual instrument concept – intuitive front panel user interfaces combined with an innovative block diagram programming methodology.

In 1986, National Instruments introduced LabVIEW 1 on the Macintosh. Although the Macintosh was not widely used for measurement and instrumentation applications, National Instruments chose it because the operating system’s graphical nature best accommodated the LabVIEW technology until the more popular operating systems could accommodate it.

By 1990, National Instruments had completely rewritten LabVIEW, combining new software technology with years of customer feedback. More importantly, LabVIEW 2 featured a compiler that made execution speeds of virtual instruments comparable with programs created in the C programming language.

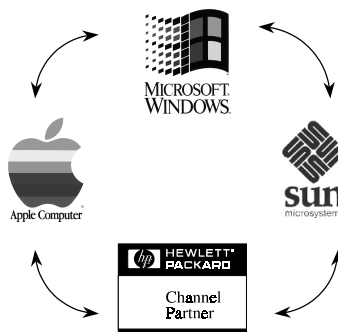
As new graphical operating systems came of age, National Instruments had the opportunity to introduce the now mature LabVIEW technology on the other industry-standard controller platforms – PCs and workstations. In 1992, National Instruments introduced LabVIEW for Windows and LabVIEW for Sun based on the new portable architecture.

In 1996, LabVIEW 4 introduced a fully user-defined environment. The user could completely redesign graphical Controls and Functions palettes.

In 1998, LabVIEW 5 is the most important release since the port to the Windows and Unix platforms. With the power to make it simple, LabVIEW 5 includes multithreading, instrument and DAQ wizards, ActiveX container support, automation servers, distributed computing tools, translation tools, documentation tools, graphical differencing tools, programmatic menu bars, and UNDO!

## Multiplatform Compatibility

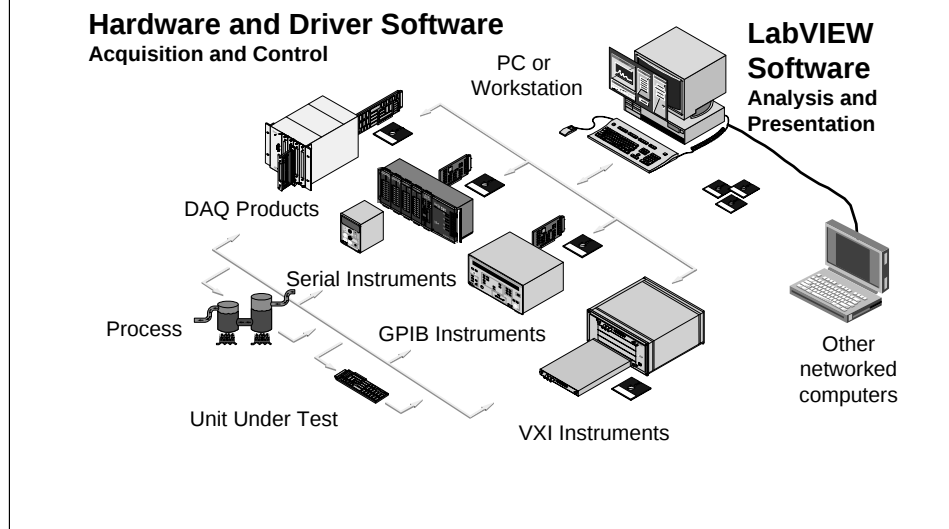
- Platform neutral
- Leverage common technology
- Migrate applications between platforms
- Also available on Concurrent PowerMAX



Because LabVIEW is platform independent, you are free to choose the computing platform to exactly meet your application needs. And the VIs you develop on one platform can easily be migrated to other platforms. The LabVIEW products for the different platforms are identical to each other. Additionally, each product also has features to take advantage of the computer platforms. For instance, LabVIEW for Windows uses ActiveX technology for interprocess communication. LabVIEW for Macintosh uses Apple Events and Program to Program Control (PPC).

LabVIEW does not lock you into one single computer platform or operating system. As your application needs change, you can easily port VIs from one platform to another. LabVIEW is designed so that we can easily port it to other platforms and operating systems in the future, thus assuring that your investment in LabVIEW remains secure as new computer architectures emerge.

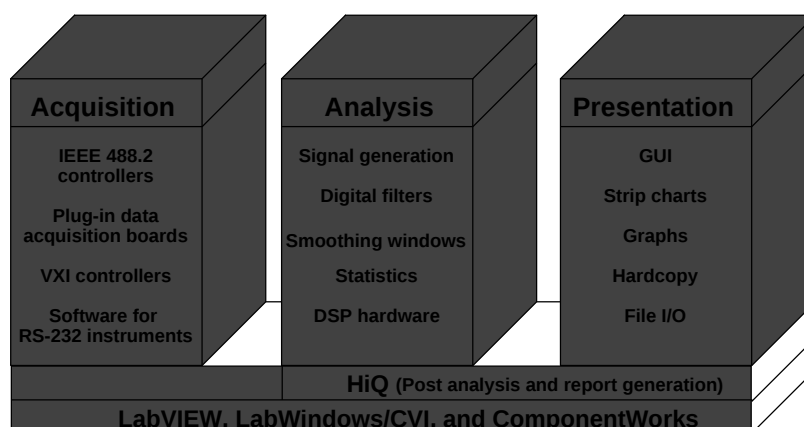
## Integrating Your System



By unbundling the key instrumentation functions, virtual instruments represent a fundamental shift from traditional, hardware-centered instruments to software-centered systems that exploit the computation, display, productivity, and connectivity capabilities of powerful and cost-effective standard desktop computers.

The computer can connect to serial, GPIB, and VXI instruments, as well as data acquisition (DAQ) products, to take a real measurement or control a process. And the computer adds another dimension to data analysis and presentation: Your data can be analyzed, saved to a file, printed, exported to another program, and published or viewed on the web with ease - which traditional instruments just cannot do.

## Who Is National Instruments?



National Instruments is the leading vendor for GPIB control products worldwide. Our GPIB ASICs represent the leading edge of GPIB control technology. We manufacture products for a wide variety of powerful industry-standard computers, including IBM PCI, PC/XT/AT, PS/2, EISA, Macintosh, SPARCstation, IBM RISC System/6000, and DECStation.

We have a wide selection of plug-in data acquisition boards, from high-performance analog input boards to low-cost multifunction boards, for all PC platforms, even laptops.

National Instruments leads the VXI control market with a complete line of VXI controllers and operating systems. We also invented the MXIbus high-speed cable standard to give general purpose computers control of VXI systems with the same power as embedded VXI controllers.

The foundation for the instrumentation system is the software – LabWindows/CVI, ComponentWorks, and LabVIEW. Each software package accommodates all aspects of measurement and instrumentation, including data acquisition and control, data analysis, and data presentation. LabVIEW, LabWindows/CVI, and ComponentWorks are designed to simplify the construction of measurement and instrumentation systems.

National Instruments is driven by a vision for the future. We are changing the world by changing the way the world uses instrumentation. By taking advantage of high-performance industry-standard computers, we are leading a revolution in which scientists and engineers will use these computers running powerful new software to build their own virtual instruments. These modular instruments are more cost effective, more flexible, and adapt more readily to changing needs than the instruments of the past. We are leading the way to the future of instrumentation, a future in which *The Software is the Instrument*.

## National Instruments Technical Support Options

- **Electronic Support**

Fax-on-Demand - 24-hour information retrieval system (512) 418-1111

WWW,BBS, & FTP - collection of updates, instrument drivers, additional files, and documents to answer common questions

www ..... <http://www.natinst.com>

BBS ..... (800) 327-3077 or (512) 794-5422

FTP ..... <ftp.natinst.com>

login: anonymous

password: your e-mail address

- Fax ..... (800) 328-2203 or (512) 794-5678

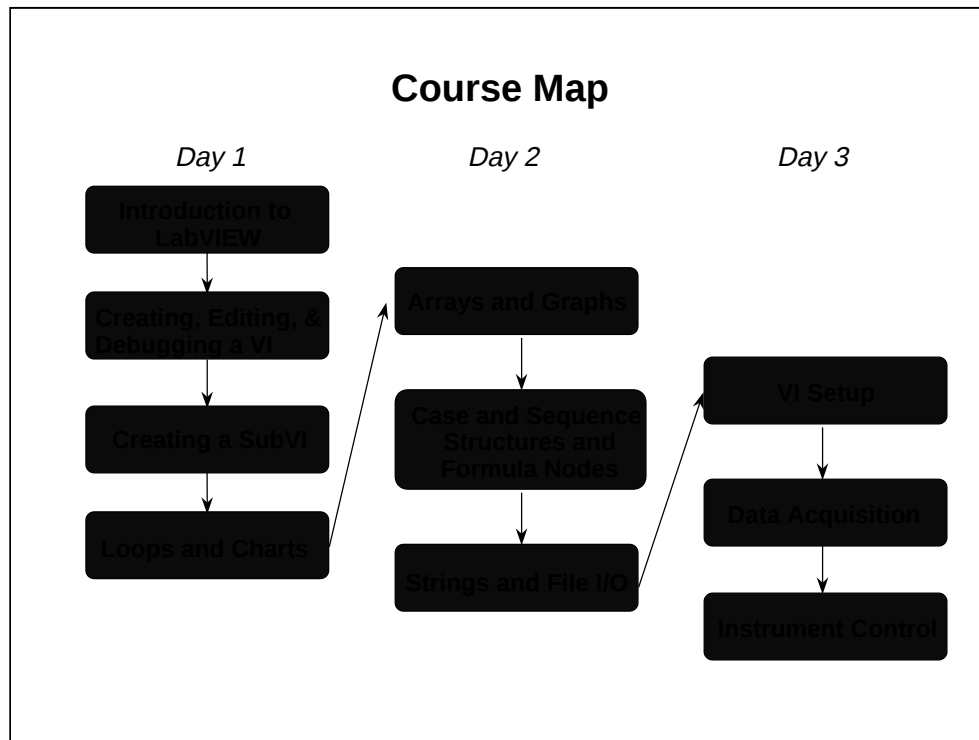
- E-mail ..... [support@natinst.com](mailto:support@natinst.com)

- Telephone ..... (512) 795-8248

Technical Support Form ..... LabVIEW » Help Menu

enter all info regarding system and support issue

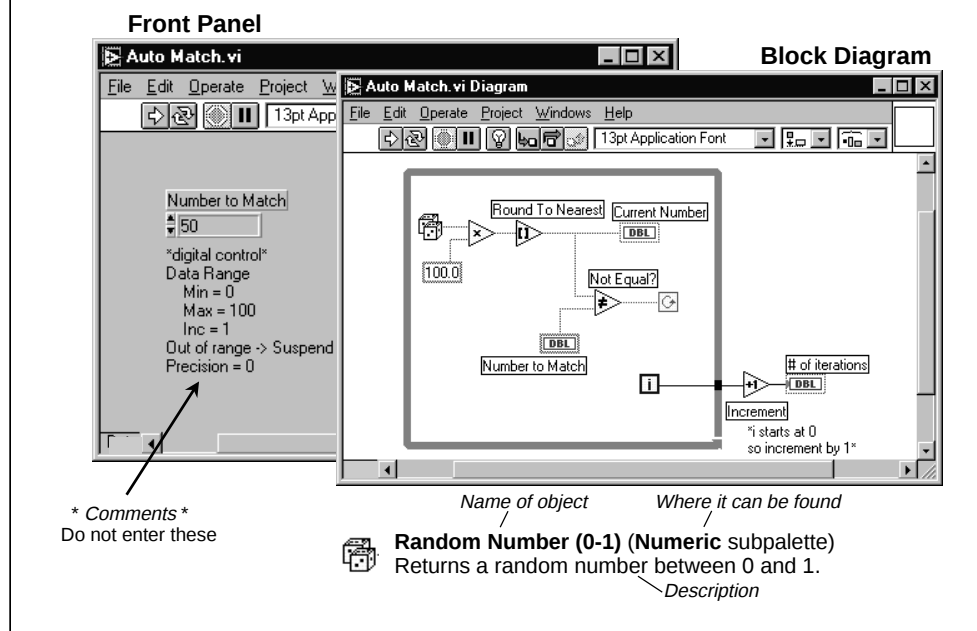
- Be sure to emphasize that the customers should try to answer their questions first by accessing WWW, Fax-on-Demand, BBS, or FTP. If those sources do not provide an answer, then contact the applications engineers through fax, e-mail, or phone.
- Demonstrate the Technical Support Form located in the **LabVIEW » Help** menu.
  - Save info in TSF file; once entered and saved, the TSF automatically accesses the TSF file for future reference.
  - You can save the form as a text file, so information saved in an ASCII file can be e-mailed or faxed to tech support.
- E-mail support is also provided for National Instruments product lines at [support@natinst.com](mailto:support@natinst.com).



## Page SG-9:

- Give a rough time frame of topics to be covered:
  - Day 1 -
    - Introduction to LabVIEW
    - Creating, Editing, and Debugging a VI
    - Creating a SubVI
    - Loops and Charts
  - Day 2 -
    - Arrays and Graphs
    - Case and Sequence Structures
    - Strings and File I/O
  - Day 3 -
    - \* (Find out if anyone is leaving early on the last day. If so, you may need to switch the order of the last two lessons.)
    - VI Setup
    - Instrument Control
    - Data Acquisition

## Course Exercises Notation



### Page SG-10:

- Point out the different conventions in the LabVIEW VIs and in the course manual.
- Comments are for information only; students do *not* need to type these comments during the exercise.
- Point out the hardware symbol in the course manual conventions. If students plan to do exercises after the class, they may need additional hardware.
- Discuss the function explanations that appear in the exercises. Use the **Random Number (0-1)** function on this slide as an example.

## **Lesson 1**

### **Introduction to LabVIEW**

#### **You Will Learn:**

- A. What a virtual instrument (VI) is
- B. The LabVIEW environment
- C. LabVIEW Help Options

#### **Page 1-1:**

- Find out how long the students have been using LabVIEW. This will give you an idea of how slowly to cover this chapter. Remember that it is a *Basics I* course, so expect some new users.
- Discuss what will be covered in Lesson 1.

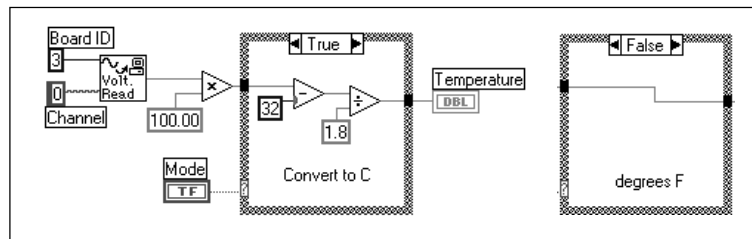
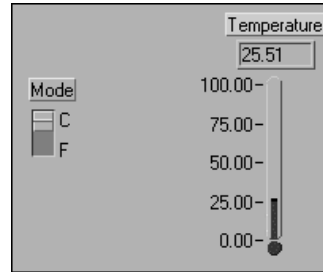
## Virtual Instruments (VIs)

### Front Panel

- Controls = Inputs
- Indicators = Outputs

### Block Diagram

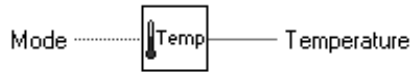
- Accompanying “program” for front panel
- Components “wired” together



## Page 1-2:

- LabVIEW programs are called virtual instruments (VIs).
- Stress that controls equal inputs, indicators equal outputs.
- Each VI contains three main parts:
  - Front Panel – How the user interacts with the VI.
  - Block Diagram – The code that controls the program.
  - Icon/Connector – Means of connecting a VI to other VIs.

## Icon/Connector



icon

- An icon represents a VI in other block diagrams

terminals



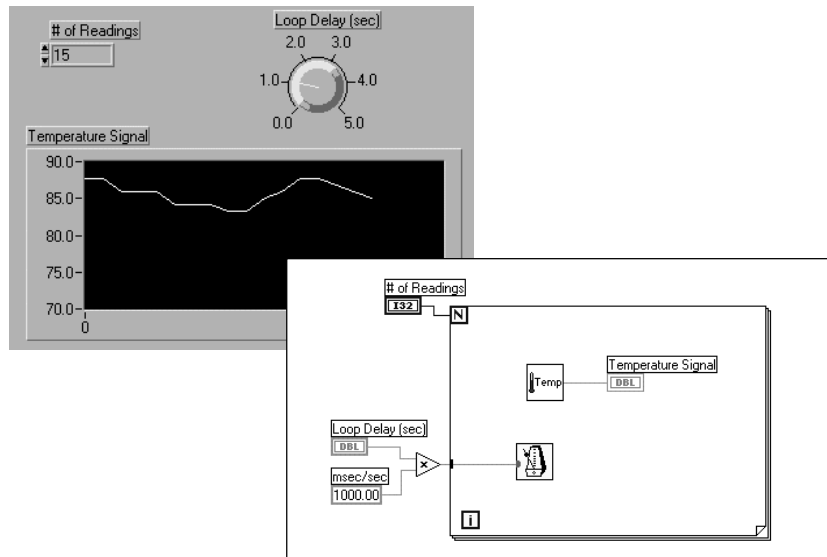
connector

- A connector passes data to and receives data from a “subVI” through terminals

## Page 1-3:

- Icon:
  - Means of turning a VI into an object that can be used in other programs (compare it to a subroutine).
  - Graphically represents the VI in the block diagram of other VIs.
- Connector:
  - Terminals define where to wire inputs and outputs.
  - Analogous to parameters of a subroutine.
  - Terminals correspond to controls and indicators on the front panel.
  - Hidden under the icon unless the user chooses to view it.

## Example: Temperature VI



### Pages 1-3 to 1-4:

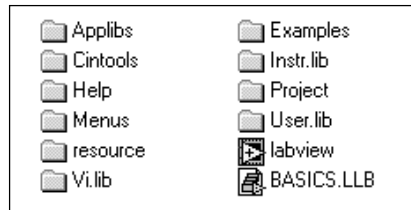
- LabVIEW is hierarchical in nature:
  - You can have many layers of VIs.
  - Very modular, easy to debug and change.
  - This is why LabVIEW is so powerful.
- Explain the example in the slide:
  - Show the relationship between front panel objects and their corresponding terminals on the block diagram.
  - *# of Readings*: control for entering number of measurements to be read.
  - *Loop Delay*: amount of time between measurements.
  - *Temperature Signal*: indicator that displays the temperatures that have been read.

## LabVIEW Files

### Windows 95

Start menu (task bar) » Programs » LabVIEW » LabVIEW

### Macintosh

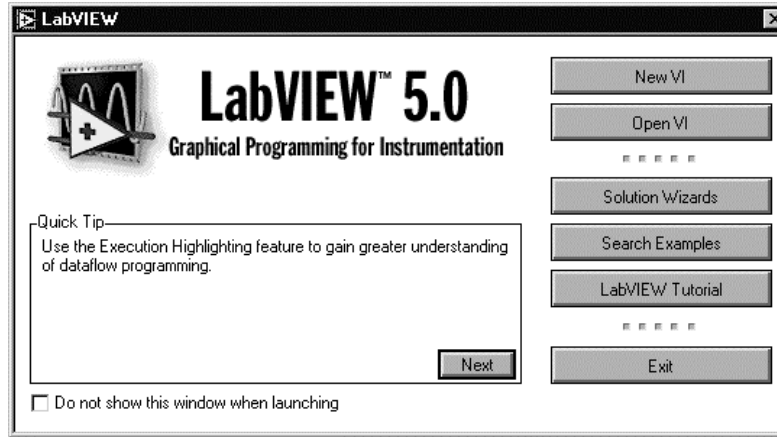


- Keep **vi.lib** in the LabVIEW directory
- Save all course work in the **BASICS.LLB** library
- Place VIs or VI Libraries in **User.lib** or **Instr.lib** to have them appear in the LabVIEW Control and Functions Palettes

## Pages 1-5 to 1-6:

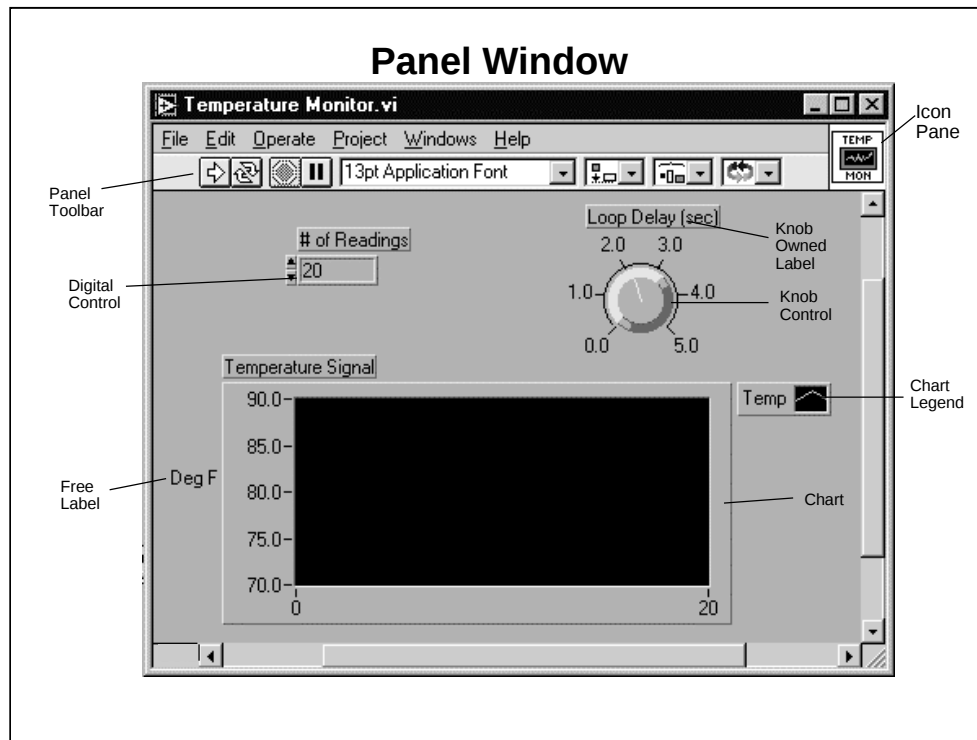
- Explain that the course is for mixed platforms, so some slides may have three sets of information.
- For all platforms, the LabVIEW system consists of the LabVIEW application and associated files. Describe these files and directories: **vi.lib**, **Examples**, **Help**, **user.lib**, **instr.lib**, and **BASICS.LLB**.
- Windows 95 - LabVIEW Group
  - LabVIEW: Starts the LabVIEW program.
  - NI-DAQ Configuration Utility: DAQ configuration utility.
- Macintosh - LabVIEW Folder and System Folder.
  - LabVIEW: LabVIEW and other associated files.
- Sun - LabVIEW Directory
  - LabVIEW: Starts the LabVIEW program.
  - Other associated files.
- HP-UX - Same directory structure and contents as Sun except for GPIB.
  - There are several other supporting programs.
  - **ibconf** and **ibic** do the same thing as on the Sun.

## LabVIEW Startup Screen



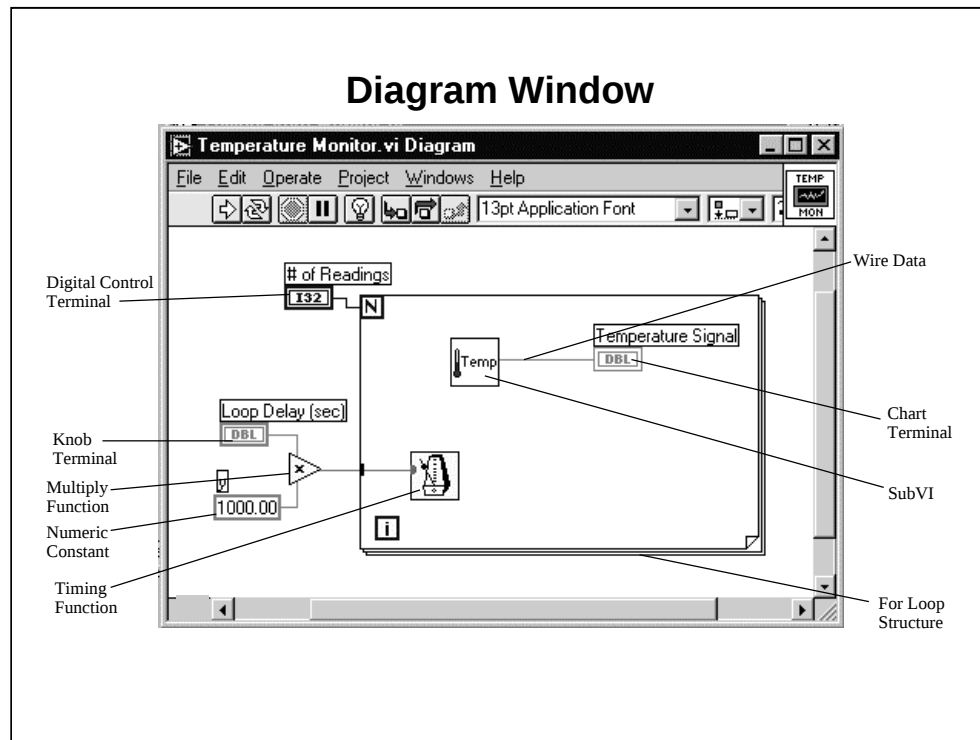
### Pages 1-6 to 1-7:

- This slide shows the startup screen for LabVIEW 5.0
- Explain the various buttons on the screen as follows:
  - **New VI** -Creates a new VI.
  - **Open VI** -Opens an existing VI.
  - **Solution Wizards** - Launches an interactive utility that allows users to make custom data acquisition and instrument control applications.
  - **Search Examples** - Opens a utility that lists and opens the LabVIEW example VIs the user selects.
  - **LabVIEW Tutorial** - Launches the interactive online tutorial. You need the LabVIEW distribution CD to access the tutorial.
  - **Exit** - Quits the LabVIEW application
- The screen also includes quick tips.
- The switch at the bottom of the screen allows the user to select this version or the abbreviated version of this screen.
- Instructor: After LabVIEW has been launched, the checkbox at the bottom of this window is replaced by a switch that gives the option of the small or large dialog box.



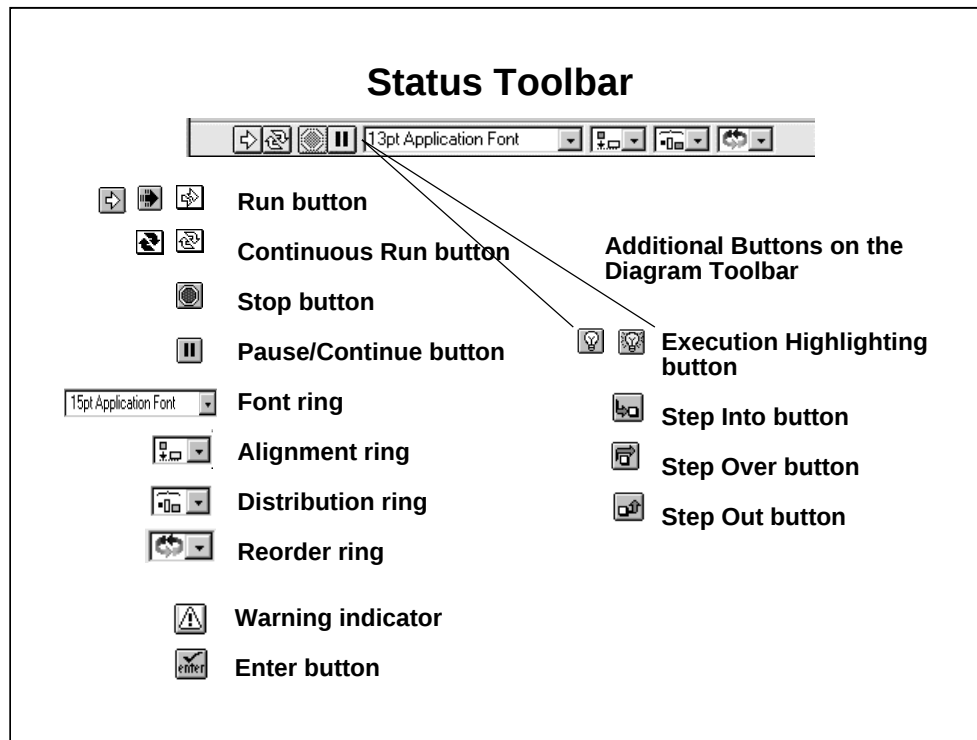
## Pages 1-7 to 1-8:

- This slide represents the front panel of a typical VI.
- Point out the elements in the Panel window:
  - Panel Toolbar: contains tools and command buttons used to control the VI.
  - Discuss the controls, indicators, and labels.



## Pages 1-7 to 1-8:

- This slide shows the block diagram for the front panel on the last slide.
  - Point out front panel object terminals, constants, and nodes. (Note the **Wait Until Next ms Multiple** function, **Multiply** function, subVI, and wires.)
  - Reinforce front panel object-terminal relationship.



## Pages 1-9 to 1-10:

- Launch LabVIEW and discuss the buttons using the computer:
- Tools used to create, modify, debug, and execute VIs:
  - Panel Toolbar: Located in the Panel window.
    - Explain each button as described in the manual.
    - Broken Run: Click on it to find errors.
    - Continuous Run: Use for testing sections of code.
    - Stop: Not a clean way to terminate the program. Avoid using it. Use a switch on the panel.
  - Diagram Toolbar: Additional tools to debug the VI.
    - Explain each button as described in the manual.
- When running a VI, the font, alignment, and distribution rings disappear because they are editing tools.

## Menus

- Pull Down Menus



- LabVIEW Pop-Up Menus

*Windows*

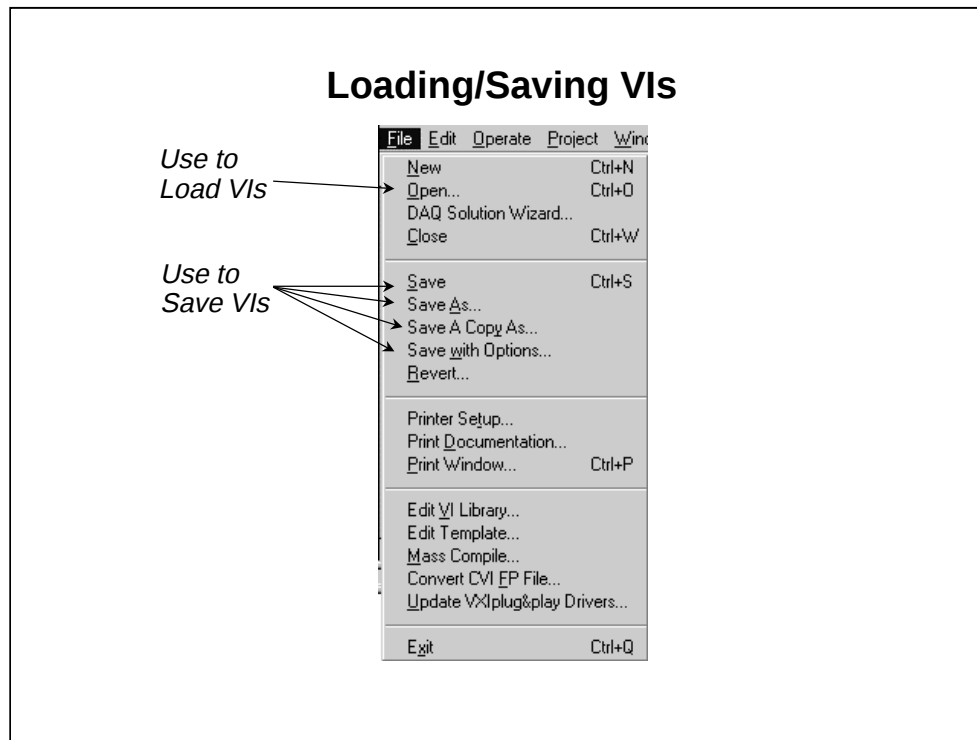
*Sun*

*HP-UX* - Click on object  
with right mouse  
button

*Macintosh* - Hold down open-apple  
and click with mouse  
button

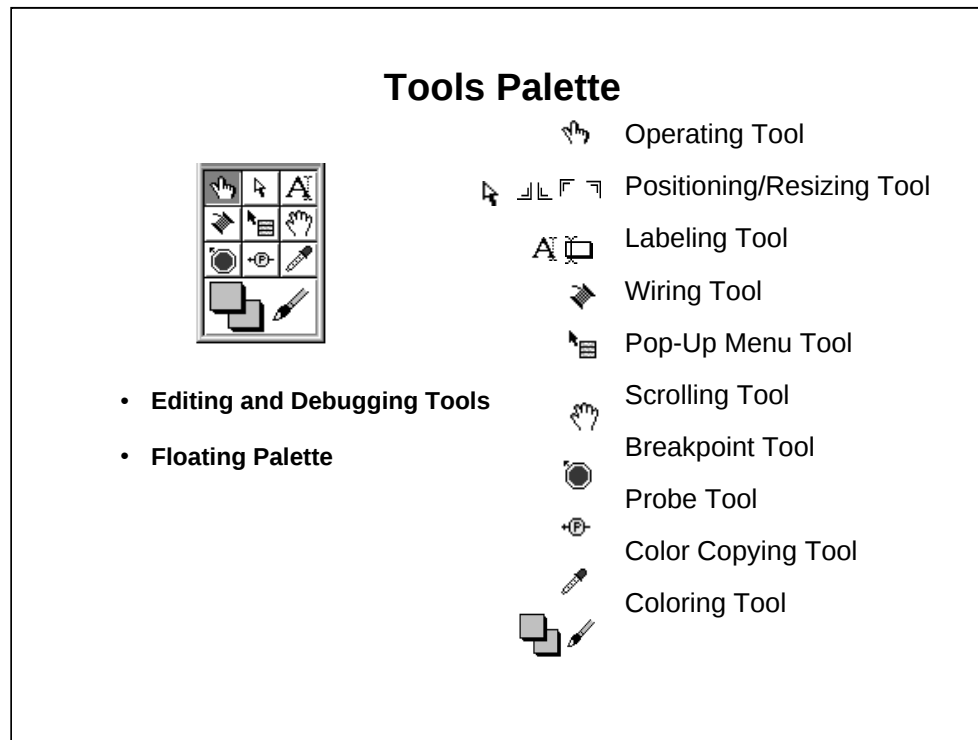
## Pages 1-10 to 1-14:

- Pull-down menus contain options common to most Windows applications, as well as others that are particular to LabVIEW.
- Explain the menus using the computer:
  - Don't go into great detail. Discuss the "high points."
    - \* **Convert CVI fp and Update VXIp&p driver (File)** converts LabWindows/CVI DLL and fp files and VXI*plug&play* DLLs to LabVIEW VIs or VI libraries.
    - \* **Show Profile Window (Project)** - Can be used for benchmarking.
    - \* **Edit Control & Functions palette (Edit)** - Can be used to design your own palettes.
    - \* **Find (Project)** - Find LabVIEW VIs, functions, and objects in memory.
  - After you discuss the **Help** menu, point out that the Online Reference includes all menu details.
  - Point out the Technical Support Form and Search Examples in the **Help** menu.
- Show how to **Pop up** on an object.



## Page 1-11:

- Use the **File** menu to open (load) and save VIs or save VIs into the VI library.
- Point out the difference in appearance of directories, VI libraries, and files.
- LabVIEW will use native OS file dialogs by default. To use LabVIEW dialogs, uncheck the option in **Edit » Preferences » Miscellaneous**.
- Instructor: Explain to the students in the issues regarding saving VIs in directories instead of libraries. Libraries have a higher probability of turning corrupt. Advantage only for Windows limitation for eight character file names.



## Pages 1-14 to 1-15:

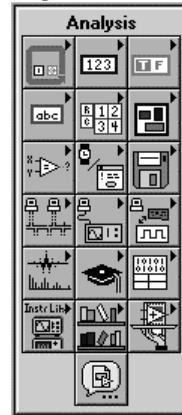
- Graphical, floating palettes.
- Tools can be used for editing, debugging, and operating VIs.
- You can hide/show palettes by enabling or disabling them via the **Window** menu.
- Shortcut key (shift-pop up - Windows)
- Shortcut key (Command-shift-pop up - Mac)

## Controls and Functions Palette

**Controls Palette  
(Panel Window)**



**Functions Palette  
(Diagram Window)**



**Graphical, floating palettes**

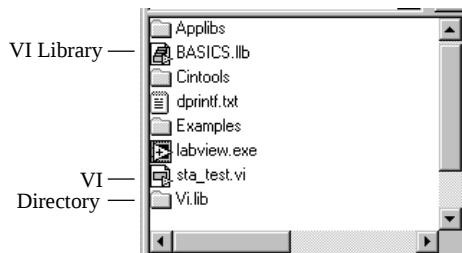
**Subpalettes can be converted to floating palettes**

Pages 1-15 to 1-19:

- Graphical, floating palettes.
- You can access the **Controls** palette *only* from the Panel window.
- You can access the **Functions** palette *only* from the Diagram window.
- If palettes are not displayed, you can access them via the **Window** menu and select the **Show Functions/Controls** palette.
- OR, pop up on open area of panel or diagram window to display **Controls** and **Functions** palettes, respectively.
- Tack down by clicking on the pushpin on the top left corner of the palette.
- Click on subpalettes to access functions, controls, VIs. You can tack down subpalettes using the pushpin.

## VI Libraries

- Similar to directories or folders
- One file – easier to transport VIs between computers
- Save VIs inside a VI library
- Not hierarchical – can't create libraries within libraries
- Compress VIs
- Names up to 255 characters including .vi extension



### Pages 1-19 to 1-21:

- You can save a VI in a directory or you can save several VIs in a single file called a VI library.
  - Not hierarchical: You cannot have libraries within libraries.
  - Point out different folder icons: Directory vs. VI library.
- Advantages:
  - Single file is easier to transport between computers and creates platform-independent filenames.
  - Compresses VIs.
  - Platform independent.
  - VI names up to 31 characters long.
  - Created for the Windows platform; not as necessary for other platforms because file naming conventions are not as restrictive as in Windows.
- Disadvantages:
  - If one file gets corrupted, you could lose a lot of work (so make frequent backups).
  - Recommend saving major applications in *directories* instead of *libraries*.
  - *Always make backups!*

## Moving VIs Across Platforms

- LabVIEW automatically translates and recompiles VIs
- VI libraries simplify VI transfer from one platform to another
  - Transfer many VIs in one VI library
  - Use long filenames for the VIs even if a platform restricts filename length
- File transfer utility mounts a disk from another platform
  - *Windows* : MacDisk and TransferPro
  - *Macintosh* : DOS Mounter and Apple File Exchange
  - *Sun* : PC File System (PCFS)
  - *HP-UX* : doscp command.

**Note: Certain operating system-specific VIs are not portable – for example, DDE, OLE and AppleEvents**

### Pages 1-22 to 1-23:

- Explain that the user will need a file transfer utility only to move VIs from one platform to another; all LabVIEW translations are done automatically.
- If VIs are not placed in VI libraries, then their filenames are limited to 8 characters in Windows 3.1, 255 characters in Windows 95 and NT, 31 characters on Macintosh, and 31 characters on Sun.
- Explain that operating system-specific VIs do not transfer from one platform to another; CINs also do not transfer because each platform uses a different C compiler.

## Exercise 1-1 on page 1-24

Students open and run Frequency Response.vi

Time to complete: 15-20 min.

### Pages 1-24 to 1-26:

- Step the class through launching LabVIEW, opening the Examples Wizard, and opening the Frequency Response.vi.
- Demonstrate how to operate controls.
- Point out the difference between the controls and indicators on this panel.
- Ask students to read through the exercise and play with the VI on their own.
- Common questions:
  - **Instructor:** This is a new exercise. Please enter any questions and answers into the CAR database.

## Help Options

### Help Window (Help menu)

- Simple/Complex Diagram Help
- Lock Help
- Online Help

### Online Reference (Help menu)

- All menus online
- Pop up on functions in diagram to access online info directly

## Pages 1-27 to 1-28:

- Use the Frequency Response example from the first exercise to demonstrate the Help window and online help.

## **Exercise 1-2 on page 1-29**

Students open and read selections from the  
**Online Reference**

Time to complete: 10 min.

### **Pages 1-29 to 1-30:**

- Demonstrate how to open the online reference on your computer.  
Do not go through all the details of the text.
- Common questions:
  - **How do I open the underlined subject titles to see the text?**  
The underlined phrases are hypertext; click on them with the mouse.

## Summary

- Panel and Diagram Toolbars contain tools for operating, editing, and debugging in the windows
- Menu options allow you to access different features in LabVIEW
- Use pop-up menus to customize any object in LabVIEW. Right-click on Windows 95/NT/3.x, Sun, HP/UX, or shift-Command-click for Macintosh
- Floating Palettes
  - Tools Palette
  - Controls Palette (only when Panel Window is active)
  - Functions Palette (only when Diagram Window is active)
- There are help utilities including the Help Window and Online Reference...

## Page 1-31:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What are the two main windows in the LabVIEW environment?
  - What do the Panel and Diagram Toolbars contain?
  - What are the differences between controls and functions? Where are each located?
  - What are the three palettes in the LabVIEW environment?
  - What are pop-up menus? How are they used?
  - How do you get help?
- Review slide: The purpose of Lesson One was to introduce the LabVIEW environment and VI concepts.
- *When in doubt, pop up!!*

## **Lesson 2**

### **Creating, Editing, & Debugging a VI**

#### **You Will Learn:**

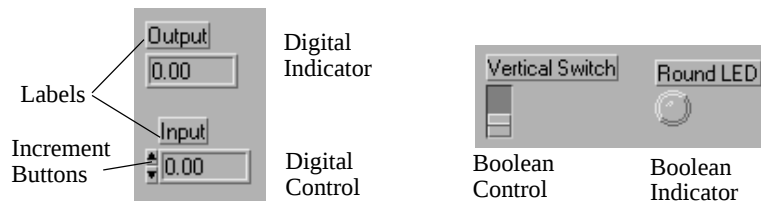
- A. How to Create VIs
- B. How to Edit VIs
- C. How to Debug VIs

#### **Page 2-1:**

- Briefly explain the topics to be discussed in this lesson.
- Point out that there are useful tips and tricks at the back of the chapter.

## Creating a VI Front Panel

- Numeric controls and indicators
- Boolean controls and indicators
- Configuring controls and indicators
  - Use pop-up menus
  - Parts have different menus



### Page 2-2:

- Creating the front panel (user interface):
  - Built with controls and indicators:
    - Controls: Supply data to the VI.
    - Indicators: Display data generated by the VI.
    - Point out the difference of control vs. indicator terminal borders in the diagram window (controls have a thicker border)
    - Also, they can find out if it is a control or indicator by checking the pop up menu of the object and referring to the first item (**Change to Indicator** or **Change to Control**)
  - Explain graphics on slide:
    - Numeric controls: Point out increment buttons.
    - Numeric indicators: No increment buttons.
    - Boolean objects: Ask class which Boolean is a control and which is an indicator.

## Accessing Pop-Up Menus

Pop up on the label for its pop-up menu.

Pop up on the digital display for its pop-up menu.

✓ Size to Text

Label  
0.00



*Windows*

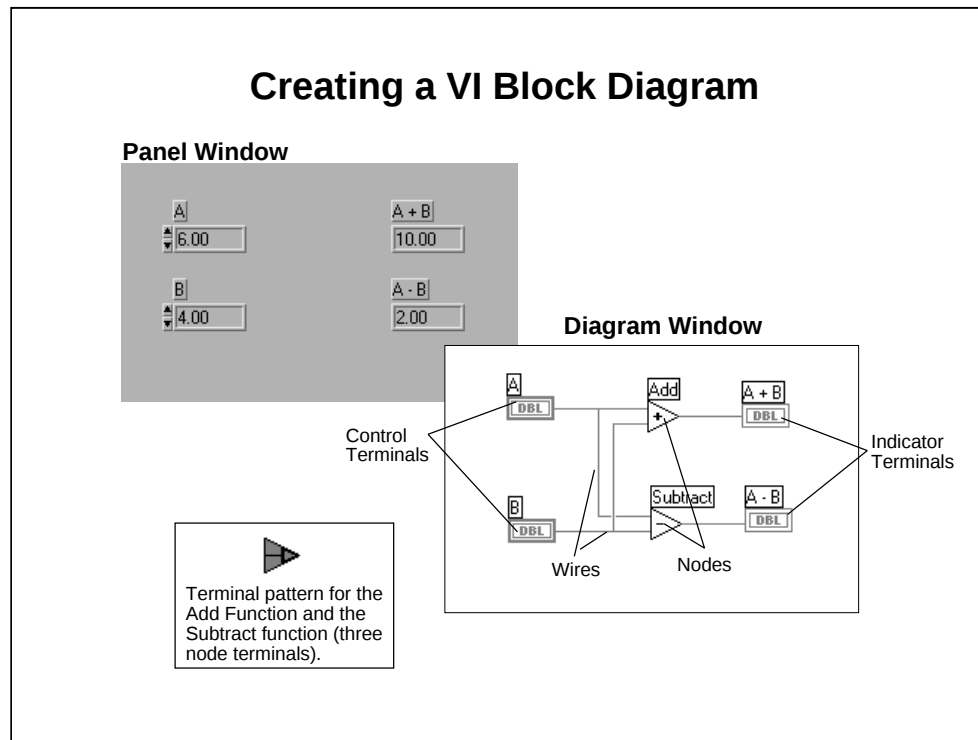
*Sun*

*HP-UX* - Click on object with  
right mouse button

*Macintosh* - Hold down open-apple and  
click with mouse button

## Pages 2-2 to 2-3:

- Pop-up menus: Most often used LabVIEW menus.
  - Explain the different ways to pop up on each platform.
  - Most LabVIEW objects have these menus for modification of attributes pertaining to that object.
  - **Functions/Controls** palettes are available as pop-up menus.
  - Instructor: Explain the difference between a label and a caption for a control/indicator. A caption is used for localization and documentation features, and a label is the LabVIEW reference between a control/indicator and its terminal.

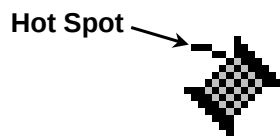


## Pages 2-3 to 2-4:

- The block diagram is the “code” for the VI.
- **Note:** Wires can cross each other. You will see a small gap in the wire. You can have bubbles at wire junctions by modifying **Preferences (Edit menu) » Block Diagram** and selecting **Show dots at wire junctions**.
- Composed of nodes, terminals, and wires:
  - **Nodes:** Program execution elements (analogous to statements, functions, and subroutines).
  - **Functions:** Built-in nodes for elementary functions.
  - **SubVIs:** VIs you design and later call from the diagram of another VI.
  - **Structures:** Control the program flow.
  - **CINs (Code Interface Nodes):** Interface to user-supplied C code.

## Wiring a VI Block Diagram

|                | Scalar                                                                            | 1D Array                                                                          | 2D Array                                                                          |                                           |
|----------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------|
| <b>Numeric</b> |  |  |  | Orange (floating point)<br>Blue (integer) |
| <b>Boolean</b> |  |  |  | Green                                     |
| <b>String</b>  |  |  |  | Pink                                      |



### Tools to Help Wiring

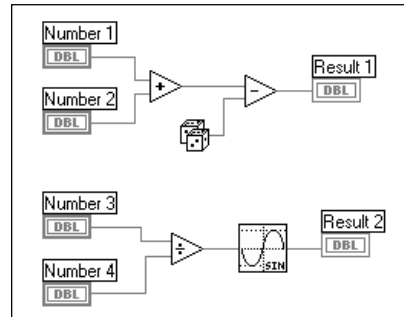
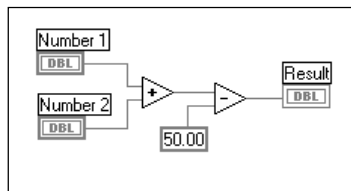
- Tip Strips
- Pop up on terminals and select **Show Terminals**
- **Help Window**

Pages 2-4 to 2-5:

- Explain the difference in colors.
- Explain that arrays and strings will be covered later in the course.
- Demonstrate wiring by placing two controls, one indicator, and an **Add** function.
  - Use the Help window (point out how terminals in the Help window flash when the tool passes over the actual function terminal).
  - Pop up on the **Add** function and select **Show Terminals**.
  - Point out Tip Strips.

## Dataflow Programming

- Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done



### Pages 2-6 to 2-7:

- *Data flow* is the principle that governs how LabVIEW programs execute.
- Explain the diagram on this slide with emphasis on data flow.
  - The node executes when data is available at *all* of its input terminals.
  - The node supplies data on output terminals when it finishes execution.
  - *Not* executed left to right nor top to bottom.
- If you have time, build the above diagrams on your computer and demonstrate the data flow using execution highlighting.

## Exercise 2-1 on page 2-8

Students build **Convert C to F.vi**

\* This VI will be used in a later exercise

Time to complete: 20 min.

### Pages 2-8 to 2-10:

- When students finish, review the following points:
  - Discuss how the diagram corresponds to the front panel.
  - Point out that control terminals have thicker borders than indicator terminals.
  - Demonstrate how to find corresponding terminals by popping up on front panel objects and selecting **Show » Terminal**.
  - Show that when typing labels:
    - <Return> acts as a linefeed.
    - To have <Return> “enter” data, choose **Edit » Preferences » Miscellaneous** menu.
  - Demonstrate the difference between clicking once, twice, and three times on a wire.
  - Explain how to use **Remove Bad Wires**. Emphasize its key shortcut: <Ctrl-B | Meta-B | Option-B>.

## Editing Techniques

Creating Objects from Diagram Window

Selecting Objects

Deleting Objects

Undo and Redo

Free vs. Owned Labels

Wiring Techniques

Changing Fonts and Text Colors

Copying Objects

Using Color

### Pages 2-11 to 2-15:

Demonstrate the following on your computer.

- Selecting objects - Use the Positioning tool.
- Moving objects - Select and drag with mouse, or select and move with arrows (holding <Shift> restricts movement to horizontal or vertical).
- Deleting objects - Select and press <Delete> or use **Cut** or **Clear** in **Edit** menu.
- Labeling objects - Explain the differences between owned and free labels.
- Changing text - Highlight and use the Font Ring.
- Resizing objects - Use the Positioning tool, which changes to a frame at the object corner.
- Aligning and distributing objects - Select objects and use the ring controls in toolbar.
- Copying objects - **Copy** and **Paste**.
- Using color - Use Coloring (Paintbrush) tool and pop up on the object (T for transparent).
- Undo and Redo

## Exercise 2-2 on page 2-16

Students modify Editing Exercise.vi

Time to complete: 20 min.

### Pages 2-16 to 2-19:

- When the students finish, review the following points:
  - Ask if they had any problems editing the front panel.
  - Point out the difference between moving an object and its owned label and moving only the owned label.
  - Show how to switch between foreground and background color in the **Color** palette by pressing keys.
  - Explain how to make a label transparent. (Select **T** with the **Coloring** tool).
  - Demonstrate popping up and how students can create constants, controls, and indicators from the block diagram.
- Common questions:
  - **How do I resize a circle without changing it to an ellipse?**  
Hold down <Shift> as you resize. You can keep the center point of the circle stationary by holding down <ctrl> as you resize it.
  - **When I align buttons, why do they go on top of each other?**  
You aligned them vertically instead of horizontally. Select **Undo** from the Edit menu and then use **Vertical Centers** to align the buttons.
  - **What is the difference between “Align” and “Distribute”?**  
**Align** puts the edges of selected objects on a specified line.  
**Distribute** affects the spacing of objects.

## Debugging Techniques

- **Finding Errors**



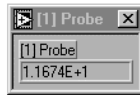
Click on broken Run button  
Window showing error appears

- **Execution Highlighting**



Click on Execution Highlighting button; it changes to   
Data flow is animated using bubbles. Values are displayed on wires.

- **Probe**



Pop up on wire to display probe – shows data as it flows through wire segment



Or, select Probe tool from Tools palette and click on wire

- **Breakpoint**



Select Breakpoint tool from Tools palette and click on wire or item where you want execution to pause

### Pages 2-20 to 2-21:

- When your VI is not executable, a broken arrow is displayed in the Run button in the palette.
- Demonstrate debugging techniques on the computer:
  - **Finding Errors:** To list errors, click on the broken arrow. To locate the bad object, click on the error message.
  - **Execution Highlighting:** Animates the diagram and traces the flow of the data, allowing you to view intermediate values.  
Click on the **light bulb** on the toolbar.
  - **Probe:** Used to view values in arrays and clusters.  
Click on wires with the **Probe** tool or pop up on the wire to set probes.
  - **Breakpoint:** Set pauses at different locations on the diagram.  
Click on wires or objects with the **Breakpoint** tool to set breakpoints.
- Use **Debug Demonstrate** VI from BASICS . LLB to demonstrate the options and tools.

## Debugging Techniques

- **Step Into, Over, and Out buttons for Single Stepping**
  -  Click on Step Into button to enable single stepping. Once Single Stepping has begun, the button will step into nodes.
  -  Click on Step Over button to enable single stepping or to step over nodes.
  -  Click on Step Out button to step out of nodes.

### Page 2-20:

- *Single stepping*: executes the diagram node by node.
- You can access single stepping from the diagram toolbar by using the step functions found on the diagram toolbar.
- Use Step Into/Step Over to begin single stepping.
- Step Into: Steps into a node. If there is a subVI, it will then bring up the subVI diagram and enable single stepping through it.
- Step Over: Executes nodes but visually does not single step through nodes.
- Step Out: Steps out of nodes, if the diagram has completed execution; click on Step Out to terminate single stepping mode.
- Demonstrate with **Debug Demonstrate VI**.

## Exercise 2-3 on page 2-22

Students modify and run Debug Exercise (Main).vi  
(Uses Debug Exercise (Sub).vi)  
Time to complete: 20 min.

### Pages 2-22 to 2-25:

- When the students finish, review the following points:
  - Ask if they had any problems using the debugging tools.
- Common questions:
  - **Why doesn't Step Over work on the subVI?** If you had previously **Stepped Into** the subVI, then a breakpoint is set on that subVI's panel. Disable the breakpoint by unselecting the **Pause** button in the toolbar. Now when you run the main VI and **Step Over**, it will work.
  - **Why do the numeric values in the panel not come out even?** The numeric indicators are only showing 2 digits of precision, but the actual values have many more significant digits. To get more precision, pop up on the digital indicator and select **Format and Precision** and increase the Precision value.

## Summary

- Two windows to create a VI -- Front Panel and Block Diagram
- Control terminals have thicker borders than indicator terminals.
- All LabVIEW objects have pop-up menus
- Wiring
  - Mechanism to control dataflow and produce LabVIEW programs
- Broken Run arrow = nonexecutable VI
- Various debugging tools and options available such as setting probes and breakpoints, execution highlighting, and single stepping

## Page 2-26:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What are the three main parts of a VI?
  - What is a Boolean control? (Give an example.)
  - What are the two main windows in the LabVIEW environment?
  - What is the purpose of wiring?
  - How can you tell if a VI is nonexecutable?
  - What are pop-up menus? How are they used?
  - What are some of the debugging tools available in the LabVIEW environment?
- Review slide: The purpose of Lesson Two was to introduce the LabVIEW editing and debugging techniques.
- *When in doubt, pop up!!*
- See the tips and tricks and the end of the chapter to practice while doing exercises.

## Tips

- Tip 1 – Command key shortcuts

| <i><b>Windows</b></i> | <i><b>Sun</b></i> | <i><b>HP-UX</b></i> | <i><b>Macintosh</b></i> |                                  |
|-----------------------|-------------------|---------------------|-------------------------|----------------------------------|
| <Ctrl-R>              | <◆-R>             | <M-R>               | <⌘-R>                   | Run a VI                         |
| <Ctrl-F>              | <◆-F>             | <M-F>               | <⌘-F>                   | Find object                      |
| <Ctrl-H>              | <◆-H>             | <M-H>               | <⌘-H>                   | Activate Help window             |
| <Ctrl-B>              | <◆-B>             | <M-B>               | <⌘-B>                   | Remove all bad wires             |
| <Ctrl-W>              | <◆-W>             | <M-W>               | <⌘-W>                   | Close the active window          |
| <Ctrl-E>              | <◆-E>             | <M-E>               | <⌘-E>                   | Toggle btwn Diagram/Panel Window |

- Tip 2 – Accessing Tools Palette with <shift>-popup.
- Tip 3 – Use <Tab> to select tools; use space bar to toggle between the two commonly used tools
- Tip 4 – Increment/Decrement faster
- Tip 5 – Preferences menu selection - set options in LabVIEW
- Tip 6 – VI Setup (Icon Pane) sets execution/window options
- Tip 7 – Project menu - many features to aid in VI management
- Tip 23 – Undo and Redo

## Pages 2-27 to 2-31:

- Demonstrate these techniques so that students can practice them throughout the course.
- Highlight the following tips:
  - Hot keys (ctrl -b, ctrl-e, etc.)
  - Accessing functions and controls palettes by popping up
  - Accessing tools palette temporarily by shift - popping up (Windows) or shift - command - pop up (Mac)
  - Tab to toggle through toolbars
  - Spacebar to toggle between positioning and operating tool (panel toolbar)
  - Spacebar to toggle between positioning and wiring tools (diagram toolbar)
  - Wiring: spacebar to change direction
  - Wiring: click with right mouse click to terminate wiring once started
  - Undo and Redo in the Edit menu

## Additional Exercises on page 2-32

2-4 -- Students build Compare.vi

Time to complete: 15 min.

2-5 -- Students build Divide.vi

Time to complete: 20 min.

### Page 2-32:

- Only those students working ahead should do this exercise. Do not spend time for the entire class to complete any additional exercises. If students get ahead later in the class, suggest that they either skip to the end of the current lesson and do its additional exercises, or return to previous lessons and finish those exercises.
- If a few students have questions about an additional exercise, use your judgment about how you should answer them. If the whole class could benefit without getting confused, take the time to explain the issue. If not, explain it to those interested during a short break.
- Common questions:
  - **How do I keep the numbers from being divided if the input is 0?** In this exercise, you divide by 0 anyway and turn on an LED to alert you to the problem. We will discuss conditional programming in a later lesson.

## Lesson 3 Creating a SubVI

### You Will Learn:

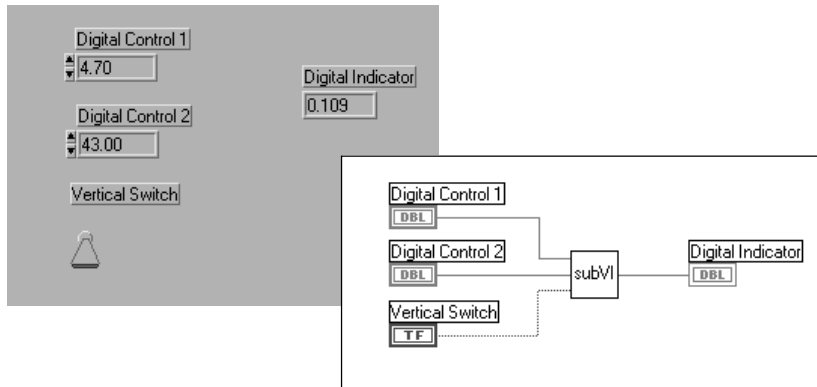
- A. What a SubVI is.
- B. How to create the icon and connector.
- C. How to use a VI as a subVI.
- D. How to use the *Create SubVI* menu option.

### Page 3-1:

- This lesson introduces the icon/connector of a VI and explains how you can use a VI as a subVI.
- The key to creating a LabVIEW application is understanding and using the hierarchical nature of the VI.
- After you create a VI, you can use it as a subVI in block diagram of a higher-level VI.
- This lesson covers:
  - What a subVI is.
  - How to create a VI icon and connector.
  - How to use a VI as a subVI.
  - The **Create SubVI** option.

## SubVIs

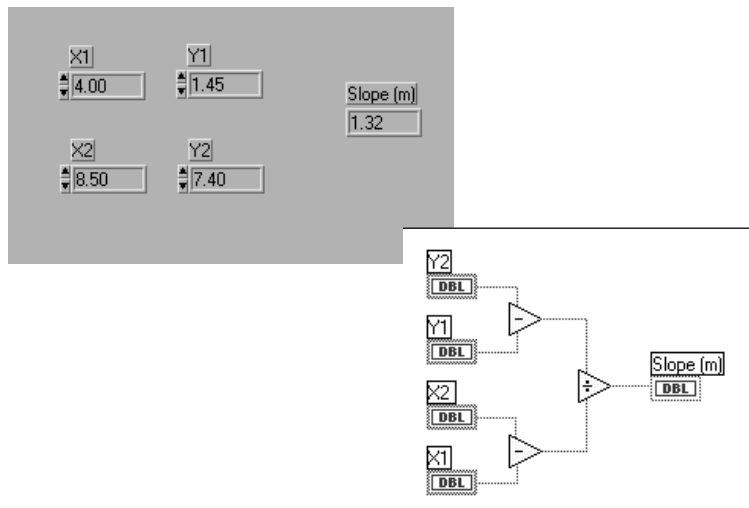
- Means of using a VI in the block diagram of a higher-level VI
- Requires icon and connector



### Pages 3-2 to 3-3:

- A subVI is analogous to a subroutine.
- When creating a VI, you should:
  - Start with the top-level VI and define input and output.
  - Construct subVIs as necessary to implement the desired functionality in the block diagram.
- The modular approach makes applications easier to debug and maintain.
- Explain diagram. (The functionality of the subVI does not matter for this example. The important point is the passing of two numeric inputs and one numeric output.)

## SubVI Example – Calculating Slope

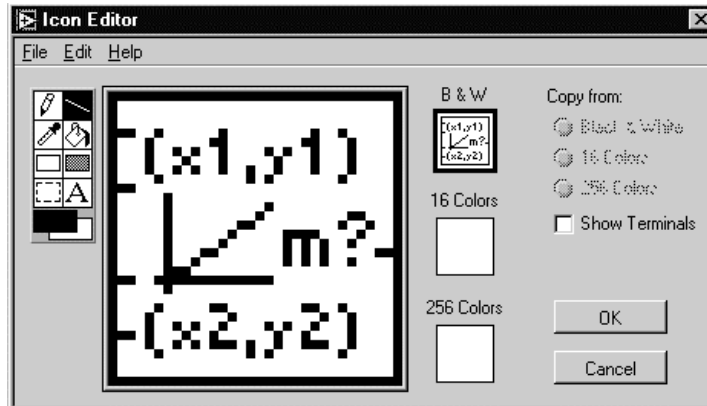


Page 3-3:

- **Instructor:** Build this example subVI **slope.vi** on your computer and demonstrate creating an icon/connector and calling the VI from another VI.

## Creating the Icon

- Pop up in the icon pane (Panel or Diagram)
- Always create black and white icon

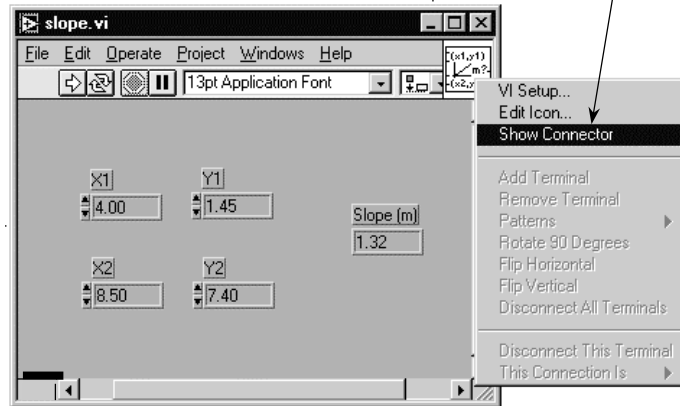


## Pages 3-4 to 3-6:

- Every VI has its icon displayed in the upper right corner of the Panel and Diagram windows. (The default is a LabVIEW icon.)
- Use Icon Editor to design the icon.
  - From the Panel or Diagram window, pop up on the icon and select **Edit Icon...**
  - Design separate icons for monochrome, 16-bit color, and 256-bit color modes.
  - Use the **Copy From** command to make a copy from one color mode to another.
  - Double-click on **Select** tool and press delete to clear icon
  - Double-click on **Rectangle** tool to add border around icon
- Using your computer, demonstrate each tool in the Icon Editor and draw a simple icon.

## Creating the Connector

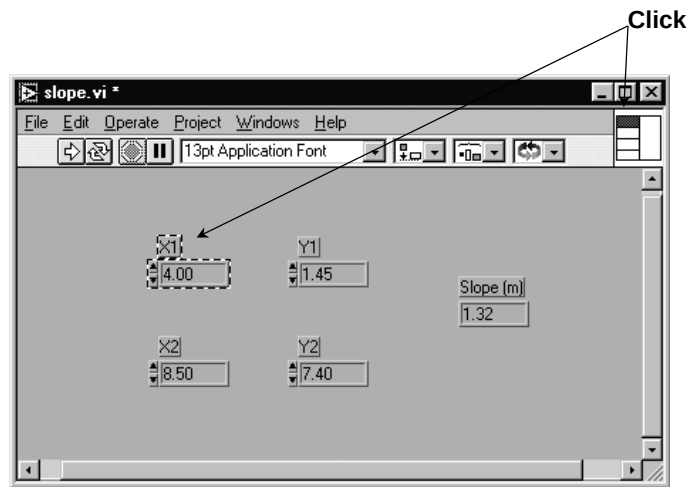
Pop up in the icon pane (Panel only)



Pages 3-6 to 3-9:

- The connector is a programmatic interface to a VI:
  - Data passes from controls and to indicators via terminals on the connector.
  - The connector is defined by assigning Front Panel objects to input and output terminals.
- Demonstrate the **Show Connector** option:
  - A Connector Pane replaces the icon.
  - Pattern selection is based on objects on the front panel.
  - To select a different pattern, pop up on the connector and choose **Patterns**.
  - Note the various ways to rotate and flip patterns.

## Creating the Connector - cont.

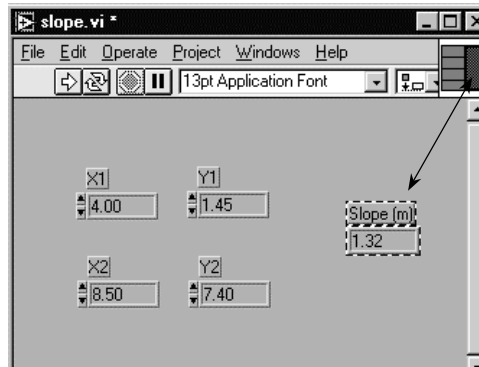


### Page 3-9:

- Demonstrate how you can illustrate which terminal is attached to which control/indicator. Click on a terminal or control.

## The Connector Pane

- The terminal colors match the data types to which they are connected
- Click on the terminal to see its associated front panel object



Page 3-9:

- Connector continued.

### Exercise 3-1 on page 3-10

Students build icon and connector for  
**Convert C to F.vi**

\*This VI will be used in a later exercise.

Time to complete: 15-20 min.

### Pages 3-10 to 3-11:

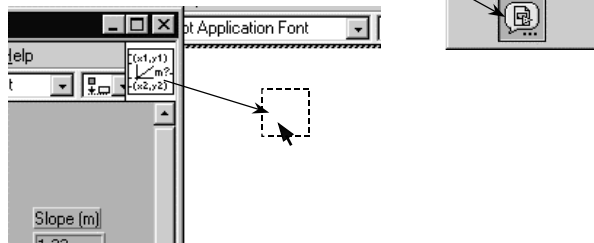
- Common questions:
  - **Why are there black-and-white, 16-color, and 256-color icons?** If you will distribute a VI to different target machines, you can make an icon in all three modes. Then, depending on the target machine's video driver, the appropriate icon can be used. For functions to appear in the **Edit VI Library** screen, you must make a black-and-white icon. You use the color version in your diagrams by selecting that icon before pressing OK in the Icon Editor.
  - **What is Copy From used for in the Icon Editor?** After making a black-and-white icon, you can click on the 16-color version and then use **Copy From Black and White**. Or, if you have a 16-color icon, you can click on another version and use **Copy From 16 Color**. This option reduces icon development time.

## Using a VI as a SubVI

Changes made to subVI saved in memory until saved to disk

Calling subVIs

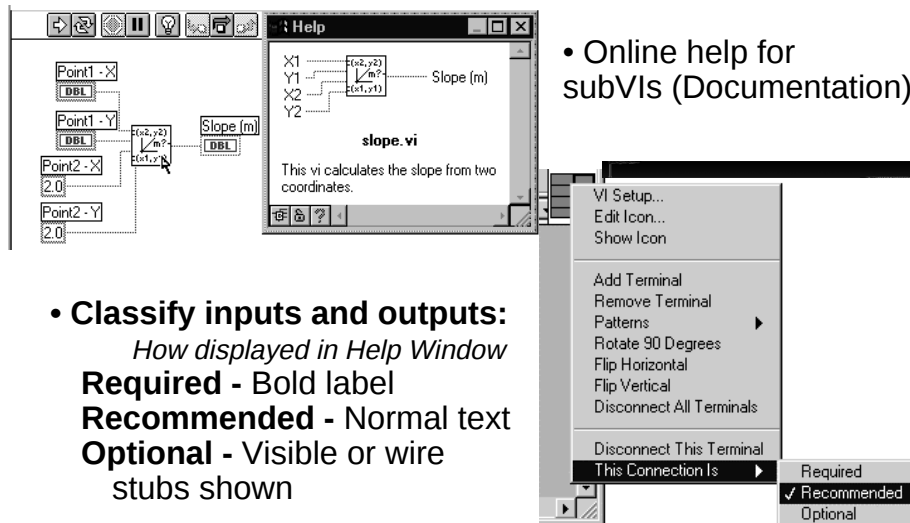
- Functions » Select a VI...
- OR
- Drag icon onto target diagram



## Page 3-12:

- You can use any VI that has an icon and connector as a subVI in the block diagram of another VI.
  - Drag the icon pane to the diagram of the target VI.
  - Select the **Select a VI...** option from the **Functions** palette.
  - The file dialog box is displayed. Select any VI file.
  - Add commonly used VIs to the **Functions** palette by placing them in `User.lib` in the LabVIEW directory.
  - SubVIs are analogous to a subroutine:
    - A node is not a subVI itself, just a subroutine call.
    - Identical nodes in the diagram call the same subVI.
    - To open, double-click the left mouse button on the icon.
- Changes to a subVI:
  - Saved in memory until the subVI is saved to disk.
  - Affect all calls to it, not just the node used to open it.

## Help and Classifying Terminals

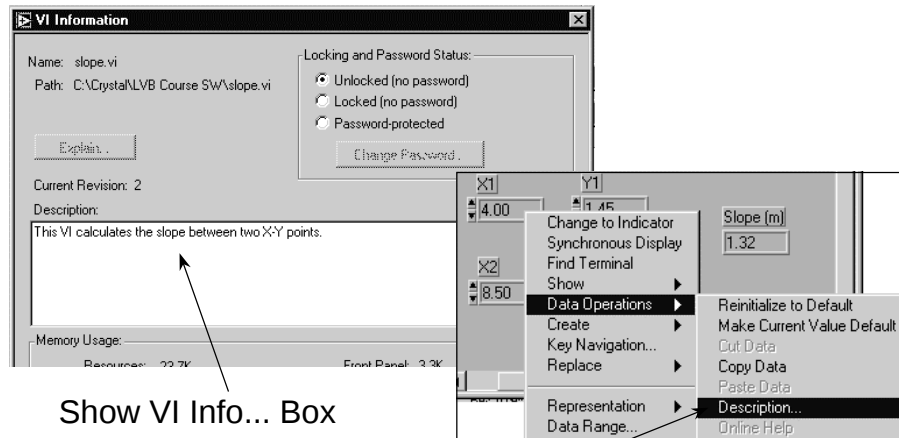


## Pages 3-12 to 3-14:

- Classify controls or outputs as required, recommended, or optional to remind users to wire inputs or outputs.
  - Classify by being in the connector pane and selecting **This connector is >>**.
  - Help windows show classifications as shown above (for example, bold for a required terminal).
  - Demonstrate using a DAQ VI
  - **Required** classification - If input is not wired, VI will be broken. Will be displayed in bold text in Help Menu
  - **Recommended** classification - Displayed as plain text in Help menu
  - **Optional** classification - hides description or hides the description in the Simple View of the Help window.

## Documenting the VI

- Document VIs - File » Show VI Info...
- Document objects - Data Operations » Description



Show VI Info... Box

Description option

### Pages 3-16 to 3-18:

- In addition to free labels you might add to the front panel and block diagram, you can give an overall description of the VI by choosing **Show VI Info...** from **Window** menu and then filling in the **Description** box.
- The Help window displays documentation.
- You can document objects on the front panel by popping up on the object and choosing the **Description...** option.

## Exercise 3-2 on page 3-15

Students build **Thermometer.vi**

**\*This VI will be used in a later exercise.**

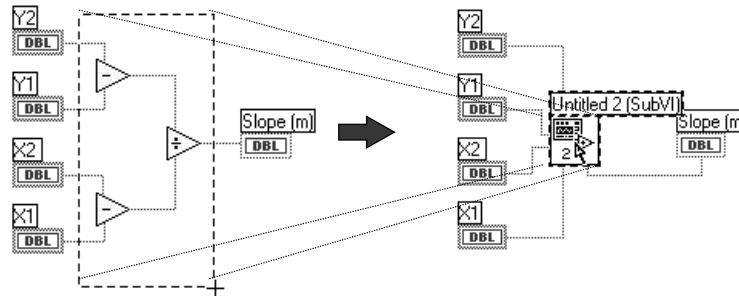
Time to complete: 30-40 min.

### Pages 3-15 to 3-22:

- Common questions:
  - **Why can't I enter text in the Description box?** You must be in edit mode.
  - **What is the difference between documenting a VI with Show VI Info and with Description?** **Show VI info** documents the entire VI. **Description** documents only a single object.
  - **Where is the Read Voltage VI?** In the **User Libraries » Basics Course** subpalette of the **Functions** palette.
  - **Why can't I wire to the channel?** Make sure that it is a string constant.
  - **Why is the channel address a string?** Non-numeric characters often appear in channel addresses; for example, a list of channels is 1,2,3.

## The Create SubVI Option

Enclose area to be converted into a subVI  
Select Create SubVI from the Edit Menu



### Page 3-23:

- Can modularize VIs after the fact by using the **Create SubVI** option.
- Creates front panel, wires diagram, and creates Connector with a default icon for you. All you need to do is save once modularized.
- Two steps:
  1. Select portion of diagram to be encompassed in subVI.
  2. Select **Create SubVI** from the Edit menu.
- Some hooks to be aware of - check User Manual

## Summary

- VIs can be used as subVIs after you make:
  - Icon
  - Connector
- Icon created using Icon Editor
- Connector defined by choosing number of terminals
- Load subVIs using the Select a VI... option in the Functions menu or dragging the icon onto a new diagram
- Online help for subVIs using the Show Help Window option
- Descriptions document functionality
- Use SubVI From Selection feature to easily modularize block diagram

## Page 3-24:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - A VI used as a subVI must have what two parts?
  - How do you create an icon?
  - How do you define a connector?
  - How do you load a subVI into a LabVIEW diagram?
  - How can you modularize the program in your diagram?
- Review the slide:
  - The purpose of Lesson 3 was to introduce the hierarchical nature of LabVIEW and the concept of subVIs.
  - The ability to call VIs as subVIs facilitates modular block diagrams. Applications are more flexible and easier to understand, debug, and modify.
  - Online help provides information about each node.
  - Modularize whenever possible.

### **Additional Exercise on page 3-25**

3-3 -- Students build Slope.VI

Time to complete: 25-30 min.

#### **Page 3-25:**

- Only those students working ahead should do this exercise. Do not spend time for the entire class to complete any additional exercises. If students get ahead later in the class, suggest that they either skip to the end of the current lesson and do its additional exercises, or return to previous lessons and finish those exercises.

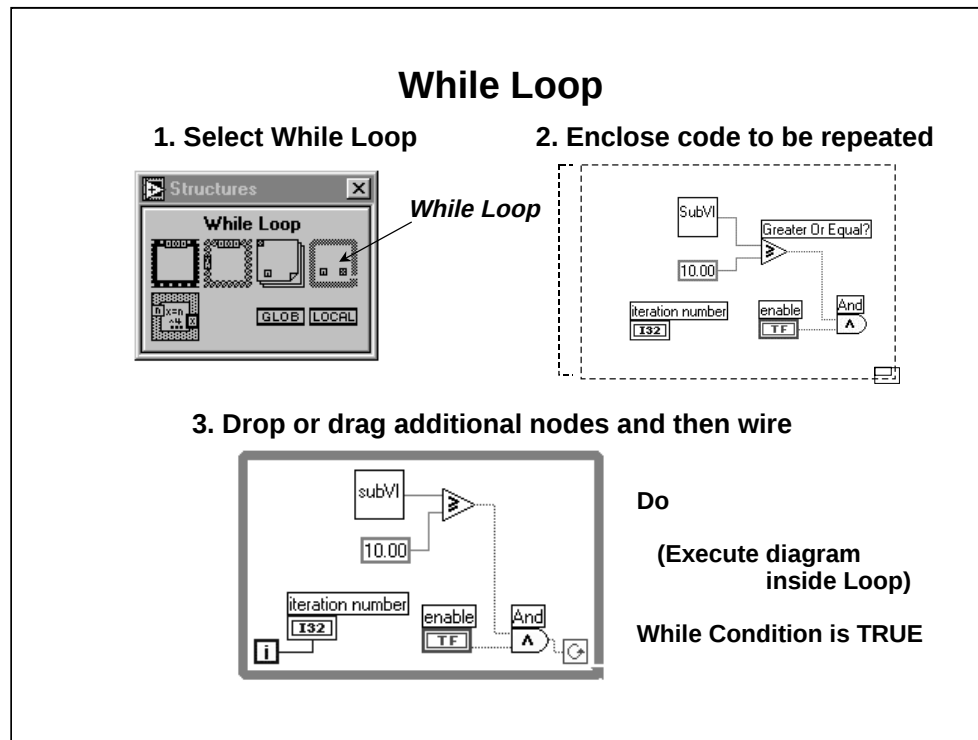
## **Lesson 4 Loops and Charts**

### **You Will Learn:**

- A. While Loops
- B. Waveform charts
- C. Shift registers
- D. For Loops

### **Page 4-1:**

- This lesson introduces LabVIEW structures. Structures control the flow of data in a VI.
- This lesson covers:
  - While Loops
  - Strip Charts
  - Shift Registers
  - For Loops
- Other structures are covered in Lesson 6.

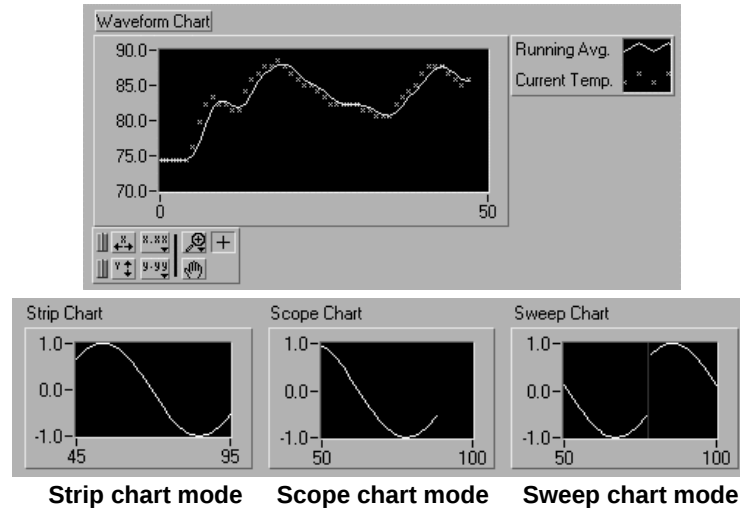


Pages 4-2 to 4-3:

- While Loops execute a sequence of events while a condition is True (analogous to a “Do...While” statement).
- Place loops in your diagram by selecting them from the **Structures** palette of the **Functions** palette (demonstrate):
  - When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.
  - When you release the mouse button, the While Loop boundary is created around the selected code.
  - Drag or drop additional nodes in the While Loop if needed.
- While Loop functionality:
  - A While Loop executes the code contained inside it until a Boolean False is passed to the conditional terminal. Thus, it always executes at least once.
  - The iteration terminal contains the number of executions (zero-indexed).

## Waveform Charts

- Selected from the Controls » Graph subpalette

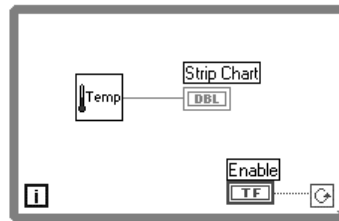


### Page 4-4:

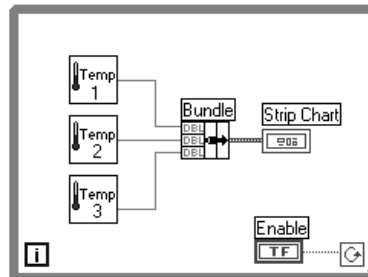
- LabVIEW has three types of charts:
  - *Strip chart*: Scrolling display.
  - *Scope chart*: Retracing display. The plot is erased after reaching the right border of the plotting area.
  - *Sweep chart*: Like the scope chart, but the plot is not erased. Instead, a vertical line marks new data and moves across the display as data is added.
  - Select modes by popping up on the chart.
- Place on the front panel by selecting the **Graph** subpalette of the **Controls** palette.
- Because less overhead is involved in retracing the plot, scope and sweep charts are much faster. To increase the speed of the strip chart, disable the X axis.
- Describe the graphic: A chart may have one trace or multiple traces.

## Wiring to Charts

- **Single-Plot Chart**



- **Multiple-Plot Chart**



### Page 4-5:

- For a single plot, wire from the numeric control to the chart terminal.
- For multiple traces, bundle each number together and then wire to the strip chart.
- Open the chart example with the Example Wizard. It is important that students know that these examples are shipped with the package. If they have questions about how to wire to a strip chart, they have a good resource that shows all possible ways to display data on a chart.

## Modifying Controls and Indicators

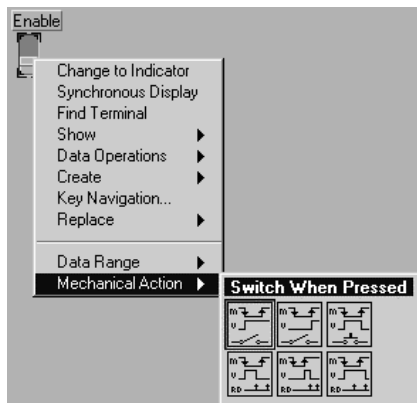
- Mechanical action of Boolean switches



Switch action: Control is toggled until changed by hand



Latch action: Control reverts to default state when read by dataflow

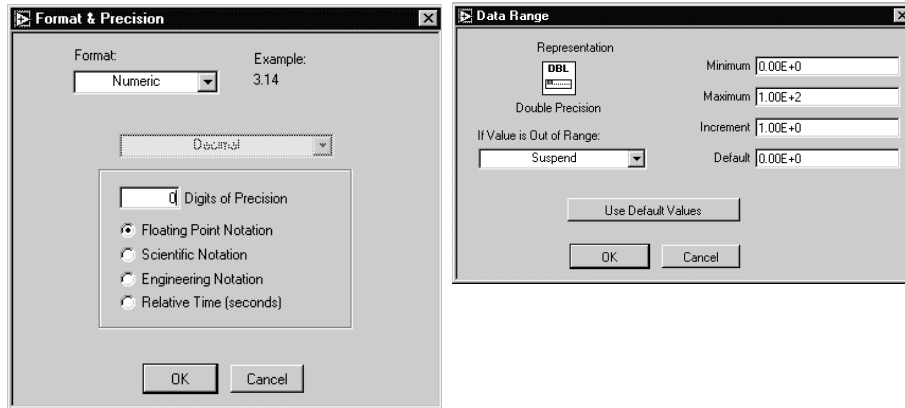


### Pages 4-9 to 4-10 (Part of Exercise 4-1):

- Each time you run Exercise 4-1, you will need to turn the vertical switch on before pressing the Run button.
- Mechanical action of Booleans can be modified:
  - Switches: Change the value when pressed.
  - Latches: Retain the new value until read, and then revert to the default.
- Use the Example Wizard to open the Mechanical Action of Booleans VI to demonstrate each of the different mechanical actions.

## Modifying Numeric Controls & Indicators

- Setting the digits of precision
- Setting the data range



Pages 4-12 to 4-13 (Part of Exercise 4-3):

- **Format and Precision** allows you to change numeric formats and number of digits of precision.
- **Data Range** allows you to specify maximum, minimum, and increment values, as well as the action to take if the value is out of range.

## Exercise 4-1 on page 4-6

Students build Temperature Monitor.vi

\*This VI will be used in a later exercise.

Time to complete: 25-30 min.

### Pages 4-6 to 4-10:

- Common questions:
  - **Why does my While Loop execute only once after changing the switch to a latch?** Assign the default state of the switch to True. Use the **Operate** menu.
  - **How do I change the value of the Boolean constant?** Use the Operating tool.
  - **How would I change from deg F to deg C while the VI is running?** Use a Boolean control on the front panel. Make sure its mechanical action is switch and not latch.
  - **Why can't I find the terminal for the chart/switch?** Pop up on the front panel object and choose **Find Terminal**.
- When students finish, review the following points:
  - **Switch mechanical action** is like a light switch. If you change the switch's position, the switch stays in that position until you change it again.
  - **Latch action** is like a doorbell. The switch stays in the "pressed" (non-default) state only until it is read by the data flow. Then it reverts to its default state.

## Exercise 4-2 on page 4-11 (Optional)

Students build Random Signal.vi

Time to complete: 20 min.

### Page 4-11:

- This exercise is optional and only for the students who finish early.
- Continue to review the following topics as part of the previous exercise:
  - Explain the **Wait Until Next ms Multiple** function:
  - Make sure you wire the **Wait Until...** function in parallel in the diagram; that is, do not connect the function to anything but the wire supplying the ms value. When connected this way, the **Wait Until...** function executes simultaneously with all the other code in the loop.
  - If the other code in the loop takes 12 ms to execute, and 550 is wired to the **Wait Until...** function, the function has 550 - 12 or 538 ms of “sleeping” to do before it completes execution. The loop will execute every 550 ms.
  - However, if the other code in the loop takes 554 ms to execute, the **Wait Until...** function will sleep until the next multiple of 550 appears on the software clock—a total of  $550 * 2$  or 1100 ms. The loop will execute every 1100 ms.

## Exercise 4-3 on page 4-12

Students build Auto Match.vi

\*This VI will be used in a later exercise.

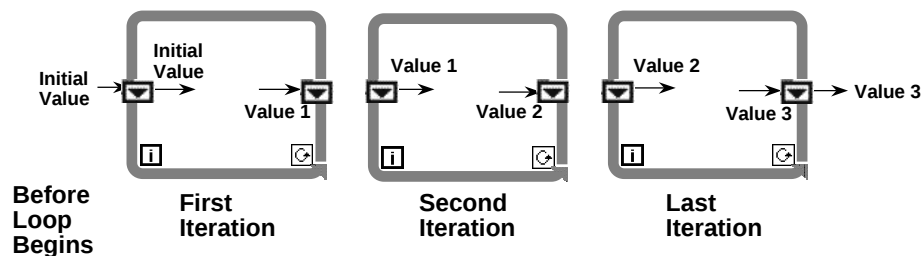
Time to complete: 20 min.

### Pages 4-12 to 4-15:

- Common questions:
  - **Why isn't the number of iterations updated each time through the loop?** Because the While Loop passes out its values only after it terminates. To update at each iteration, place the terminal inside the loop.
  - **Why do I need to increment the iterations by one?** Because the iteration terminal is zero-indexed.
  - **Why is the Not Equal? function wired to the conditional terminal?** The **Not Equal?** function is True until a match is found.

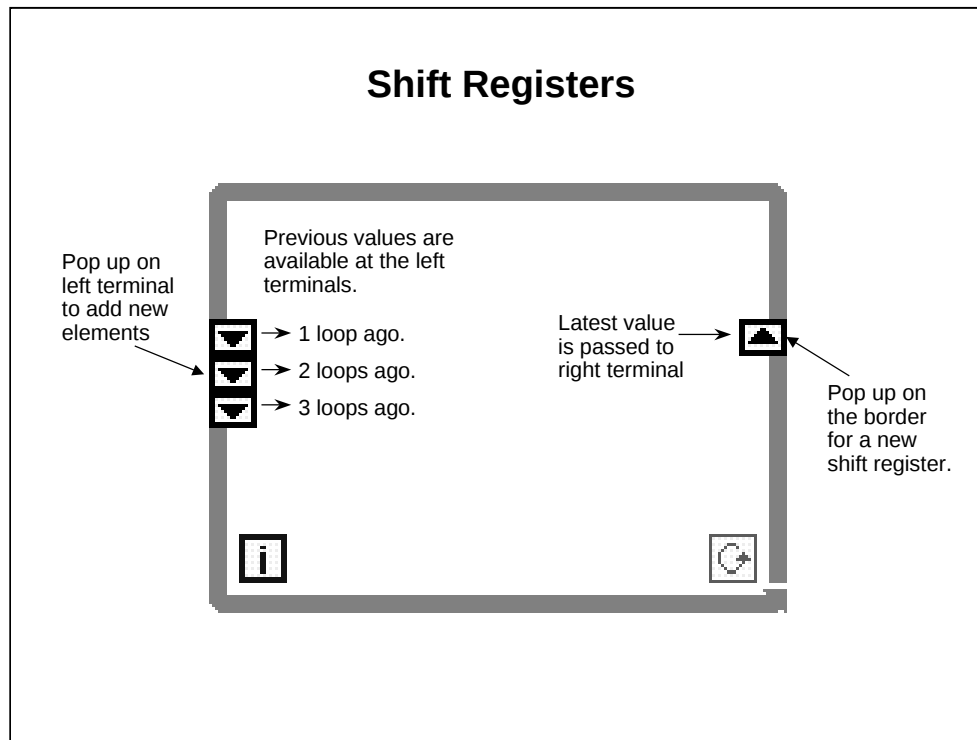
## Shift Registers

- Available at left or right border of loop structures
- Pop up on border and select Add Shift Register
- Right terminal stores data on completion of iteration
- Left terminal provides stored data at beginning of next iteration



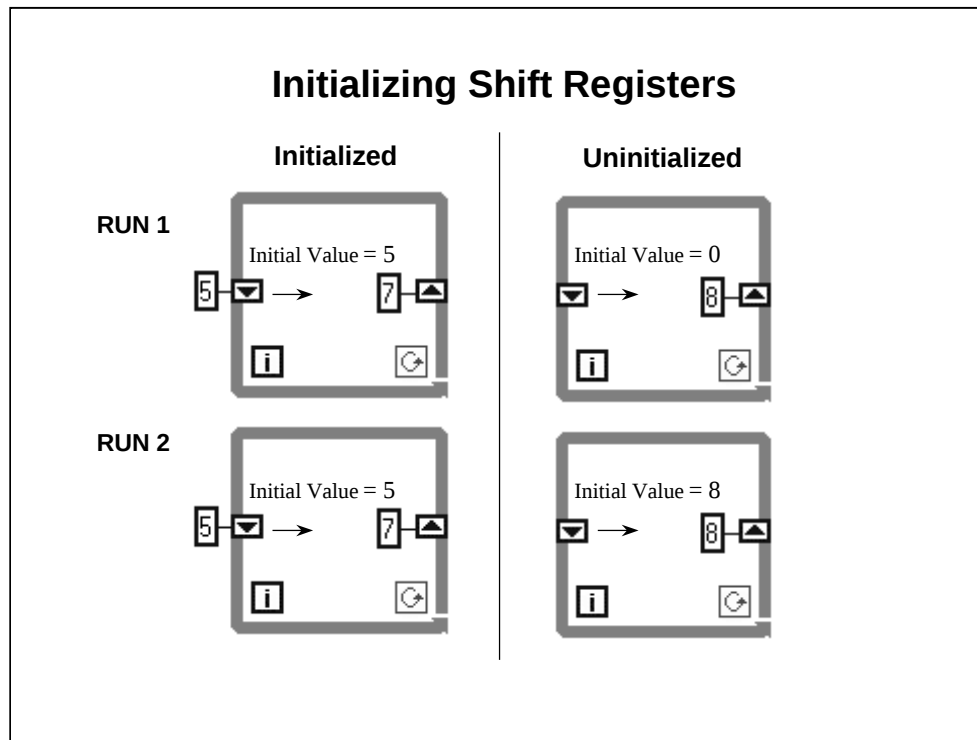
### Page 4-16:

- Shift registers transfer data from one iteration to the next:
  - Pop up on the left or right side of a For Loop or a While Loop and select **Add Shift Register**.
  - The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.
  - A shift register adapts to any data type wired into it.
- Stress *why* you need a shift register—to retain values from one iteration to the next.
- Explain the slide *in detail*. Go slowly and repeat yourself.



## Page 4-17:

- You can configure the shift register to remember values from several previous iterations.
- Pop up on the left terminal and choose **Add Element** to create additional terminals.
- Explain the slide: The shift register allows you to access values from the past three iterations if you have three terminals.
- This method is different from using separate shift registers:
  - One shift register with added terminals is storing multiple previous values of *one* variable.
  - Multiple shift registers maintain a single previous value of *multiple* variables.
  - Demonstrate the difference between adding a shift register vs. adding an element to a shift register.



## Page 4-18:

- Explain the difference between the left column and right columns of loops.
- Left column -
  - Initialize the shift register with a 5 so *every* time the VI is run, the shift register starts out with a value of 5.
- Right column-
  - No initialization, so the first time the VI is run, the shift register has a value of 0. The next time it is run, however, it contains the value at the last iteration of the previous run (in this example, an 8).

### **Exercise 4-4 on page 4-19**

Students examine and run  
**Shift Register Example.vi**

Time to complete: 15-20 min.

### **Pages 4-19 to 4-20:**

- Single step through this VI in highlighting mode on your computer, explaining the use of shift registers.
- Go slowly. Be sure to answer all questions.

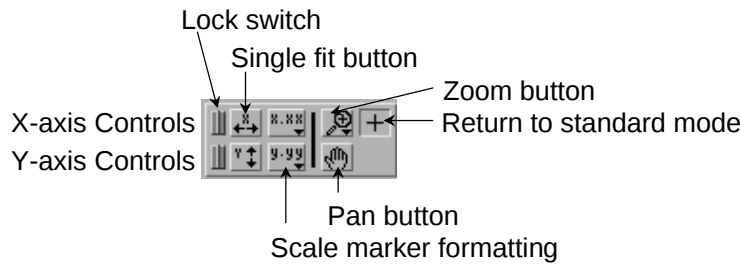
## Customizing Charts and Graphs

Digital displays (from Show menu)

Scroll bars (from Show menu)

Customize the axes (from Formatting... in pop-up menu)

Zooming controls (on palette)



**Demonstrate use of the Chart/Graph palette on the computer**

### Pages 4-24 to 4-25 (Part of Exercise 4-5):

- You can extensively customize charts and graphs.
- Use the **Show** option in the pop-up menu to add **Digital Displays** for the traces and **Scroll Bars** for the X axis.
- You can change the font style and color for the axis labels by highlighting one of the numbers on the axis you want to change with the Labeling tool and then choosing the style and color you want from the **Text** menu.
- **Instructor:** Demonstrate the zooming tools on your computer. Go slowly so the students can absorb the information and practice.

## Exercise 4-5 on page 4-21

Students modify Temperature Monitor.vi (Ex. 4-1)  
and save it as Temperature Running Average.vi

\*This VI will be used in a later exercise.

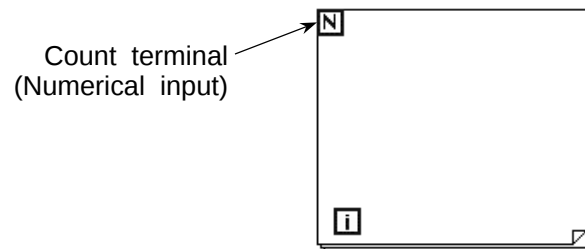
Time to complete: 25-30 min.

## Pages 4-21 to 4-25:

- Common questions:
  - **How do I fit all of my text in the legend box?** Resize the legend from the lower left corner.
  - **How do I change the number of values displayed on the chart?** Double-click with the Operating tool on the upper limit of the X axis and enter the number of points you want to display.
  - **How many points are stored in the buffer?** There are 1,024 points by default. You can increase the points up to as many as you need to store in **Chart History Length** in the chart's pop-up menu.
  - **How do I change the plot style so that there is no line connecting the data points?** Change **Interpolation** to None.
  - **Why can't I use a single Temp subVI?** One subVI initializes the shift registers while the other one inside the loop gathers data repeatedly. Be sure the students use one register with added elements—not two sets of registers.

## For Loop

- In Structures subpalette of Functions palette
- Enclose code to be repeated and/or resize and add nodes inside boundary
- Executes diagram inside of loop a predetermined number of times
- Shift registers can be created at the border

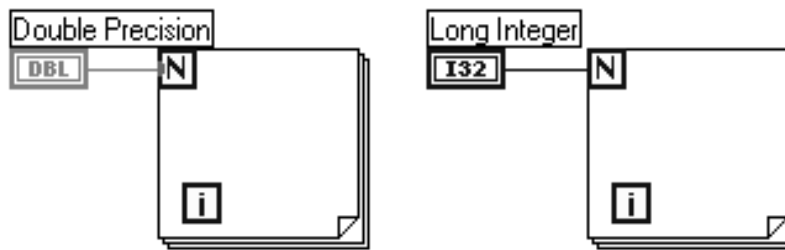


### Page 4-26:

- A For Loop executes the diagram inside its border for a predetermined number of iterations (analogous to a “For...Next” statement).
- Enclose nodes inside the For Loop the same way as in a While Loop.
- $N$  is the count terminal—It specifies how many times the loop will execute.
- $i$  is the iteration terminal—It specifies the number of loops executed at any iteration.
- You can add shift registers if you need to retain values from one iteration to the next.
- Explain the difference between While Loops and For Loops:
  - While Loops run until a condition is met.
  - For Loops run a predetermined number of times.

## Numeric Conversion

- Numeric defaults to double precision (8 bytes) or long integer (4 bytes)
- LabVIEW automatically converts to different representation
- Gray dot on terminal indicates conversion



### Pages 4-27 to 4-28:

- The default representation of a numeric control or indicator is a double-precision floating-point number.
- LabVIEW can represent a number as:
  - Byte (8-bit) signed or unsigned integer.
  - Word (16-bit) signed or unsigned integer.
  - Long (32-bit) signed or unsigned integer.
  - Single-precision (32-bit) floating-point number.
  - Double-precision (64-bit) floating-point number.
  - Extended-precision (96-bit\*) floating-point number.
- \* Depends upon the operating system
- When two terminals of different data types are wired together, LabVIEW will convert one numeric to the same representation as the other. This is signified by a gray “coercion dot.”

## Exercise 4-6 on page 4-29

Students build **Random Average.vi**

Time to complete: 20 min.

### Page 4-29:

- If the class is running behind schedule, have the students complete this exercise by modifying Exercise 4-5. Show students how to pop up on the While Loop, replace it with a For Loop, and replace the Temperature VIs with the **Random Number** function.
- Common questions:
  - Same questions as for Exercise 4-5.
- Ask students how they would determine when to use a For Loop and when to use a While Loop. (A For Loop executes a pre-determined number of times.)

## Summary

- Two structures to repeat execution
  - While Loop
  - For Loop
- Loop timing controlled using Wait Until Next ms Multiple function
- Three modes of waveform charts
  - Strip chart
  - Scope chart
  - Sweep chart
- Charts can be customized – pop up on chart for menus
- Shift registers transfer data values from one iteration to the next
  - Adapt to any data type
  - Additional left terminals may be added to access previous iteration

### Page 4-30:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What structures are used to repeat execution of certain functions?
  - How can you control the speed at which the loop executes?
  - Describe the difference between the three types of charts.
  - Why would you want to use a shift register?
  - How do you retain multiple values from previous iterations?
- Review the slide: The purpose of Lesson Four was to introduce LabVIEW structures. Structures control the flow of data in a VI.

### Additional Exercises on page 4-31

#### 4-7 -- (Challenge) Students build Combo While/For Loop.vi

Time to complete: 20-25 min.

#### 4-8 -- Students build Temperature Limit.vi

Time to complete: 20 min.

#### 4-9 -- Students modify Temperature Limit.vi (Ex. 4-8) and save the new VI as Temp Limit (max/min).vi

Time to complete: 20 min.

### Page 4-31:

- 4-7 Hints:
  - Use a While Loop.
  - Use the **Or** function to check if the button has been pressed or if the loop has reached a specified count.
- 4-8 Common questions:
  - **How do I create a scope chart?** Select **Data Operations » Update Mode**.
  - **How do I create a multiple-plot chart?** Review the diagram on page 4-5.
- Do not spend time for the entire class to finish these exercises. However, because they contain important concepts that many students may use in their applications, discuss the techniques and show the solutions on your computer if there is time.

## **Lesson 5**

### **Arrays and Graphs**

#### **You Will Learn:**

- A. About arrays
- B. Generating arrays on loop boundaries
- C. Some basic array functions
- D. What polymorphism is
- E. Using graphs to display data

#### **Page 5-1:**

- This lesson introduces graphs and arrays.
- Arrays allow you to group like data types together.
- This lesson also introduces graphs. Graphs are useful in plotting a full waveform or plotting a group of X data vs. a group of Y data. In contrast, the previous lesson discussed charts. Charts are used for continuous plotting of points, one or more at a time.
- This lesson covers:
  - Basic array operations and functions
  - Polymorphism
  - How to use graphs

## Arrays

- Collection of data elements that are of the same type
- One or more dimensions, up to  $2^{31}$  elements per dimension
- Elements accessed by their index
- First element is index 0

|                  |     |     |     |     |     |     |     |     |     |     |
|------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| index            | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 10-element array | 1.2 | 3.2 | 8.2 | 8.0 | 4.8 | 5.1 | 6.0 | 1.0 | 2.5 | 1.7 |

|            |   |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|---|
|            | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2D array 0 |   |   |   |   |   |   |   |
| 1          |   |   |   |   |   |   |   |
| 2          |   |   |   |   |   |   |   |
| 3          |   |   |   |   |   |   |   |
| 4          |   |   |   |   |   |   |   |

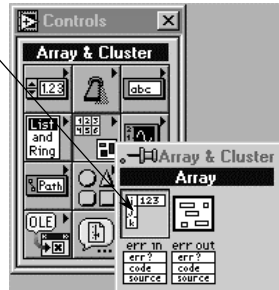
Five-row by seven column  
array of 35 elements

### Page 5-2:

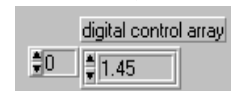
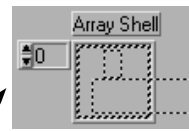
- Arrays are lists of elements of the same data type. They are analogous to arrays in traditional languages.
  - Arrays can have one or more dimensions.
  - Arrays can have up to  $2^{31}$  elements per dimension. Actual array sizes that students can create is limited by memory.
  - Elements are accessed by “index”. The index ranges from 0 to N-1 (N = number of elements in the array).
  - Arrays are zero-indexed (first element is zero) in each dimension.
- A 2D array is analogous to a spreadsheet or table.
  - Example: Temperature readings and time stamp. One column is time values, and the other column is readings.
  - Point out element (2,3) and element (3,2) on the slide.
- Be careful to specify the element you really want. Example: The first element in an array is array (0), not array(1).

## Array Controls and Indicators

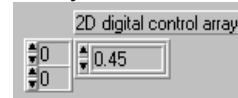
1. Select the **Array** shell from the Controls palette



2. Place **data object** inside shell



Add Dimension  
for 2D arrays

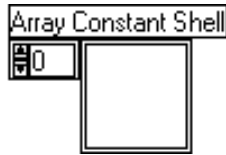


### Pages 5-2 to 5-3:

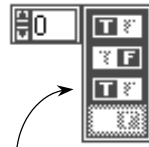
- Creating arrays in LabVIEW:
  - Choose **Array** from the **Array and Cluster** palette
  - You can place any data type in an array shell except an array. You cannot have an array of arrays; use a 2D array instead.
  - Two components: *index* and *data object*.
  - Emphasize that this is a *two-step* process. Students often place only empty array shells on the front panel. Remind them they must place a data type inside the array shell.
- Demonstrate the following on your computer:
  - Create a numeric array.
  - Point out index and data object components.
  - Show how to create a 2D array.
  - Show how to display multiple array elements.
  - Show that index elements always reference the upper-leftmost object in the array display.
  - Show how elements in an array are initially “grayed-out,” indicating that a portion of the array has not been defined.

## Creating Array Constants

1. Select Array Constant shell from Array subpalette of the Functions Palette



2. Pop up on the array shell and select data object

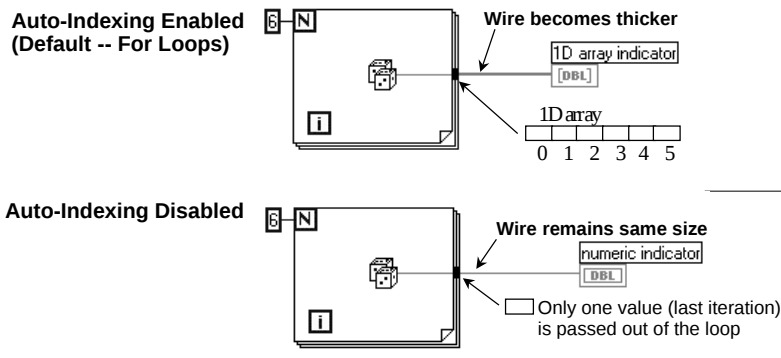


### Pages 5-3 to 5-4:

- Creating array constants uses the same two-step process as a front panel array.
- Select an **Array Constant** from the **Array** subpalette of the **Functions** palette.
- Place a constant data type inside the empty array shell.
- Note again that the data types are “grayed-out” and must be manually defined if the user wants to store values in the array constant.

## Creating and Using Arrays

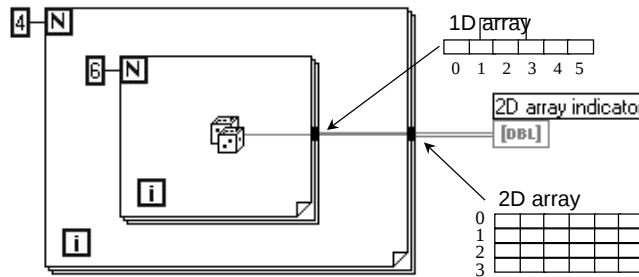
- Loops accumulate arrays at their boundaries – Auto-Indexing
- For Loops auto-index by default; While Loops do not



### Page 5-5:

- For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as *auto-indexing*.
  - The indexing point on the boundary is called a *tunnel*.
  - The For Loop default is auto-indexing enabled.
  - The While Loop default is auto-indexing disabled.
- Examples:
  - Enable auto-indexing to collect values within the loop and build the array. All values are placed in array upon exiting loop.
  - Disable auto-indexing if you are interested only in the final value.
- Auto-indexing can be used on input arrays:
  - For calculations to be performed on each element of array, use auto-indexing.
  - To pass the entire array into a loop, disable auto-indexing.

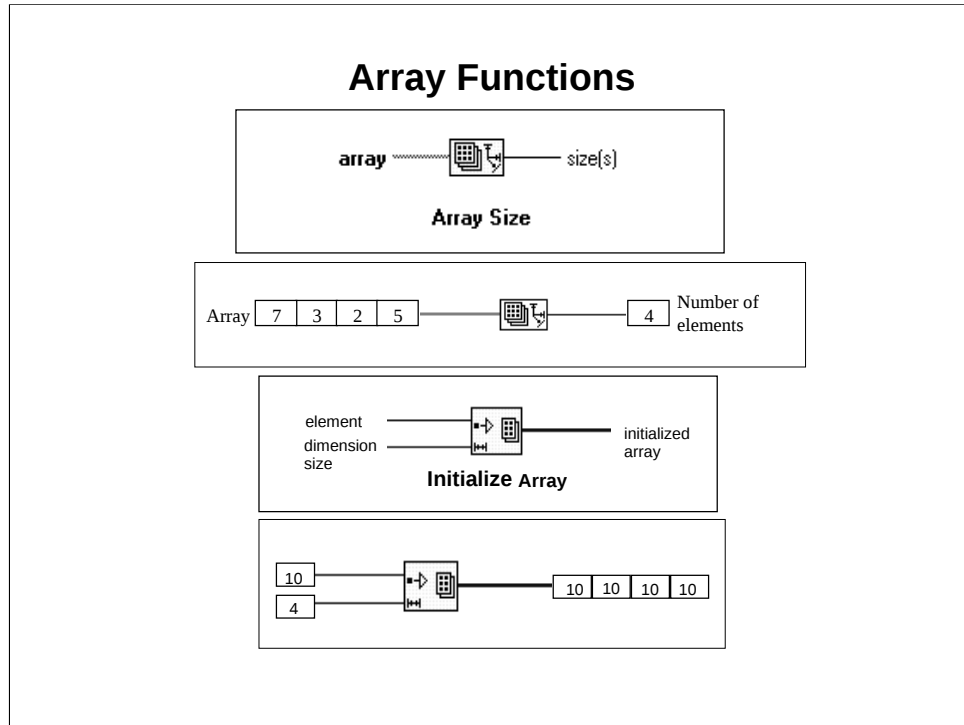
## Creating 2D Arrays



- Inner loop creates column elements
- Outer loop stacks them into rows

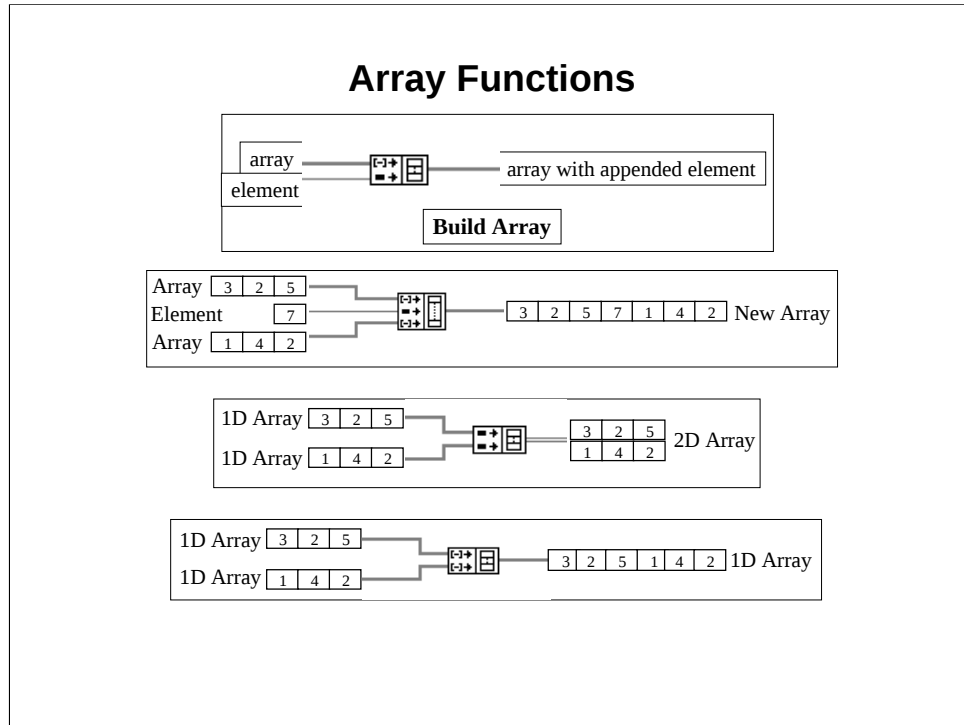
### Page 5-6:

- You can use two nested For Loops to create a 2D array. Auto-indexing must be enabled for both.
  - Inner loop creates column elements.
  - Outer loop creates row elements.
- Explain the three line thicknesses.
- Demonstrate on your computer:
  - How to change indexing and line thickness.
  - **Enable Indexing** in pop-up menu means indexing is currently *disabled*. The menu choice is the opposite of the current index mode. Students get confused about this feature.



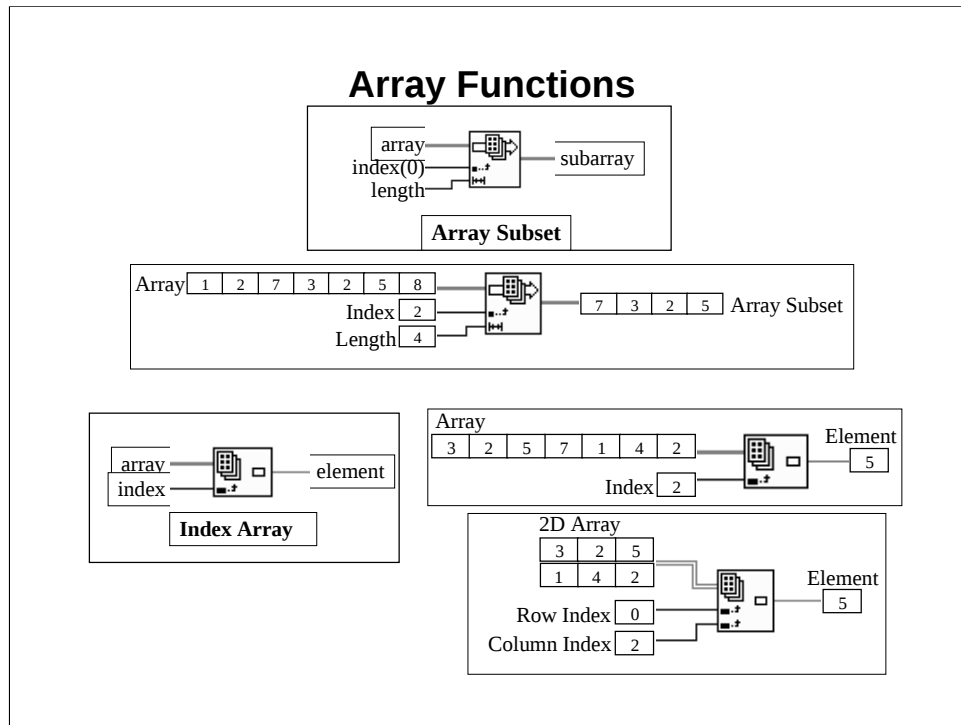
## Page 5-7:

- LabVIEW has many functions to manipulate arrays in the **Array** subpalette of the **Functions** palette.
  - **Array Size:** The number of elements in the input array:
    - 1D array: Output is a numeric.
    - Multidimensional array: Array output with elements signifying size of each dimension. (Example: A 2 x 4 array will output array of two elements, with first element = 2 and second element = 4).
  - **Initialize Array:** Creates an array of n dimensions containing the value tied to the element input.
    - Initialize array can be “stretched out” to add more “dimension size” input terminals. You must have one “dimension size” input terminal for each dimension in the array.



## Pages 5-7 to 5-8:

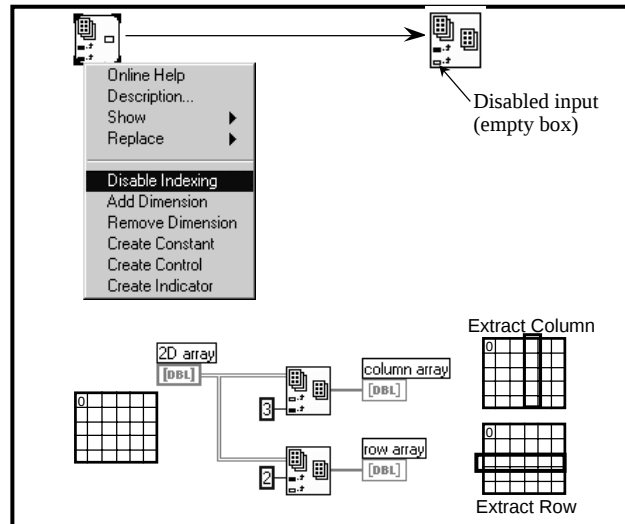
- **Build Array:** Concatenates elements together into one array or concatenates multiple arrays together into arrays of higher dimension.
  - Build array can be “stretched-out” to add additional input terminals
  - Input terminals are set as *Array* input or *Element* input. Popping up on the input terminals allows selection of **Change to Array/Element input**.
  - To concatenate arrays or elements together into one longer array, arrays must be tied to *Array* input terminals and single elements must be tied to *Element* input terminals.
  - To create a higher dimensional array, arrays must be tied to *Element* input terminals.
  - Students often are confused by the difference. Demonstrate on your machine the difference between “Array” and “Element” input.



## Pages 5-8 to 5-9:

- **Array Subset:** returns a portion of an n-dimensional array starting at *index* in each dimension and containing *length* elements in each dimension.
  - The function can be “stretched-out” to add additional index and length terminals. You must have one pair of these terminals for each dimension in the array.
- **Index Array:** accesses a specific element of the input array.
  - **Index Array** can be “stretched-out” to add more index terminals. You must have one index terminal for each dimension in the input array.
  - The slide example shows accessing a single element of a one-dimensional array.
  - **Index Array** does *not* remove the element from the array.
  - Remind students that arrays start their indices at zero. Index 2 in the example actually accesses the *third* element in the array.

## Array Functions

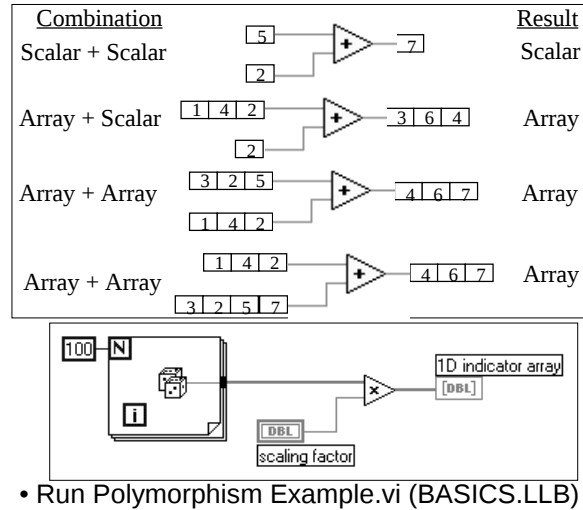


Pages 5-9 to 5-10:

- **Index Array** (cont.):
  - Slide Example: Extract a column and a row from a two-dimensional array. You must select **Add Dimension** and then **Disable Indexing** for the desired dimension.
  - Explain this example carefully, because many students want to know how to do this task. Show an example on your machine.

## Polymorphism

- Function inputs can be of different types
- All LabVIEW arithmetic functions are polymorphic



## Page 5-11:

- LabVIEW arithmetic functions are *polymorphic*:
  - Inputs to these functions can be of different types.
  - The node automatically performs the appropriate function on unlike data.
  - Greatly simplifies array arithmetic.
- Examples of polymorphism in slide:
  - Scalar+Scalar: Scalar addition.
  - Scalar+Array: The scalar is added to each element of array.
  - Array+Array: Each element of one array is added to the corresponding element of other array.
- Remind students that polymorphism does *not* perform matrix arithmetic when inputs are 2D arrays (for example, two 2D array inputs to a multiply function does element by element multiplication, not matrix multiplication).
- Show an example on polymorphism if the class does not understand the concept.

## Exercise 5-1 on page 5-12

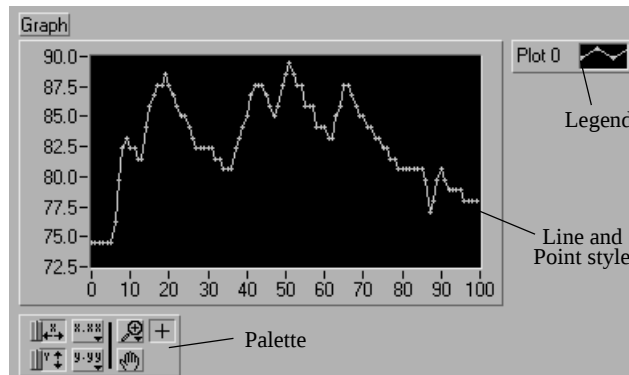
Students build Array Exercise.vi  
Time to complete : 20 min.

Pages 5-12 to 5-13:

- **Instructor:** Please write down any common questions the students may have and enter them into the CAR database.

## Graphs

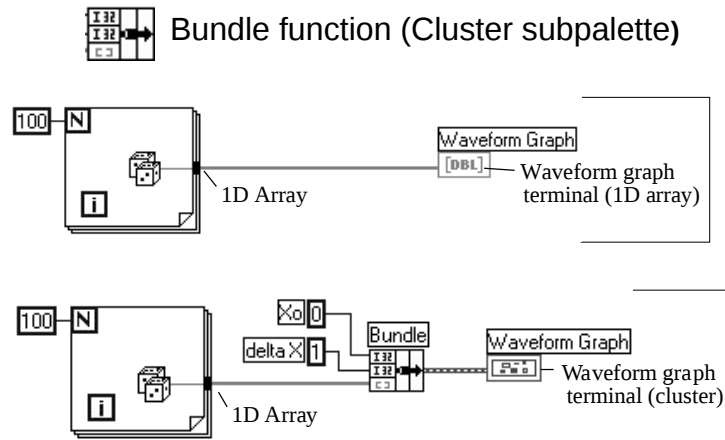
- Selected from the Graph palette of Controls menu
- Waveform Graph – Plot an array of numerics against their indices
- XY Graph – Plot one array against another



### Page 5-14:

- A graph is a 2D display of one or more data arrays. Data arrays are called *plots*.
- Two types of graphs:
  - *Waveform Graph*: Plots array of numerics vs. its index. Ideal for arrays with evenly distributed data points. Used for time-varying waveforms.
  - *XY Graph*: Plots one array vs. another. General purpose. Good for multivalued functions, such as circular shapes or waveforms with a varying timebase.
  - Both look identical on front panel.
- To place a graph on the front panel, choose the appropriate graph from the **Graph** subpalette of the **Controls** palette.

## Single-Plot Waveform Graphs

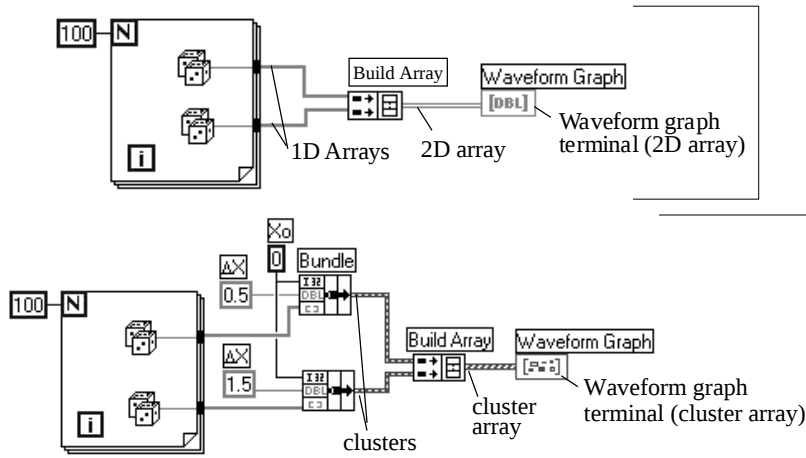


Pages 5-14 to 5-15:

- Because the **Bundle** function is used with graphs, be sure to briefly introduce the idea of clusters. Clusters allow you to group different data types together into one structure. In this lesson, they are useful to control the x-axis scaling attributes.
- **Bundle** function: Used to combine two or more elements of like or different data types. Remind the class that they used this function to bundle arrays in multiplot strip chart.
- There are several different ways to plot a waveform graph:
  - Example 1: Bundle the array with the initial X value ( $X_0$ ) and interval between X values (Delta X). The terminal looks like a cluster of objects because the data was bundled. (Called a *cluster*).
  - Example 2: For simple plots, the array can be passed directly to the waveform graph terminal. Assumes the initial X value is 0 and the X increment is 1. The terminal is an array of double-precision numbers.
  - Note that auto-indexing is used in the above examples.

## Multiple-Plot Waveform Graphs

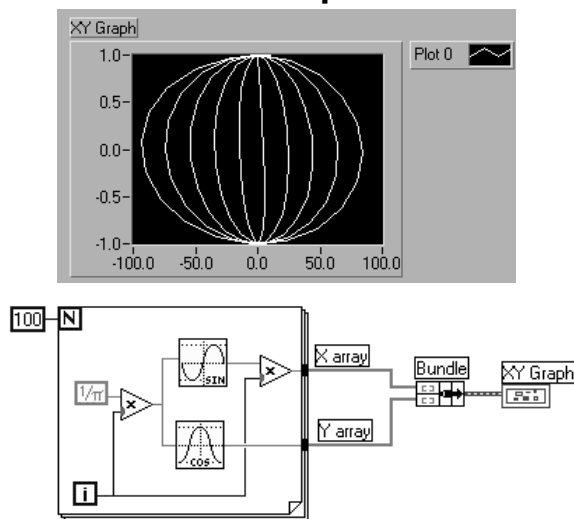
 Build Array function (Array subpalette)



### Page 5-16:

- If a one-dimensional array is wired to a waveform graph, the graph assumes it is one plot. When a 2D array is wired to a waveform graph, the graph assumes *each row* in that array is a *separate* plot.
- Use the **Build Array** function to create a 2D array input for multiplot graphs:
  - Example 1: A combination of two single-plot examples with  $X_0$  and  $\Delta X$  defined (Ex 1 from previous slide). The **Build Array** function creates a cluster array input to the waveform graph. The terminal is an array of clusters, because each graph is a cluster.
  - Example 2: A combination of two single-plot examples without  $X_0$  and  $\Delta X$  defined (Example 2 from the previous slide). The **Build Array** function creates a 2D array input to the waveform graph. The terminal is a 2D array of double-precision numbers.

## XY Graphs



- Run Waveform Graph Example.vi and XY Graph Example.vi

## Page 5-17:

- The XY graph looks identical to the waveform graph on the panel.
  - Does not assume that the X-axis has a constant interval.
  - Array of X values and array of Y values must be provided.
  - More flexible than the waveform graph.
- Explain the slide: Generate two arrays, one for X values and other for Y values. Bundle arrays wire into the XY graph.
- Explain that an X array 1,2,3,4,5 and a Y array 0,1,0,1,0 will result in a graph with points ((1,0)(2,1)(3,0)(4,1)(5,0)).
- Show the class some graph examples using the Example Wizard:
  - Examples of ways to graph charts (**Charts.vi**), waveform graphs (**Waveform Graph.vi**), and XY graphs (**XY Graph.vi**).
  - Very useful reference.

## Exercise 5-2 on page 5-18

Students build Graph Waveform Array.vi  
Time to complete : 30 min.

### Pages 5-18 to 5-24:

- Common questions:
  - **Why do I get a bad wire to the Waveform Array indicator?** Make sure you place a numeric inside the array shell. Also make sure it is an indicator and not a control.
  - **Why do I get a bad wire from the For Loop to the Bundle function?** Make sure that auto-indexing is enabled. Pop up on the output tunnel.
  - **Where is the Process Monitor VI?** In the **Basics Course** menu.
  - **How do I type a delta symbol?** Type an uppercase <D> and change the font to Symbol.
  - **How do I undo the zooming I've done?** Click on the X scaling button, followed by the Y scaling button.

## Exercise 5-3 on page 5-25

Students build Temperature Analysis.vi

\*This VI will be used in a later exercise.

Time to complete: 25-30 min.

### Pages 5-25 to 5-28:

- Common questions:
  - **Why is 40 the number of iterations of the For Loop?** Every 0.25 s multiplied by 10 s = 40 measurements.
  - **Why is 250 wired to the Wait Until... function?** 250 ms = 0.25 s.
  - **Why isn't Array Max & Min working?** Make sure you have wired the maximum and minimum values to the function, not their index values.
  - **How do I delete the X axis on the chart?** Set the X-scale style to none.

## Exercise 5-4 on page 5-29 (Optional)

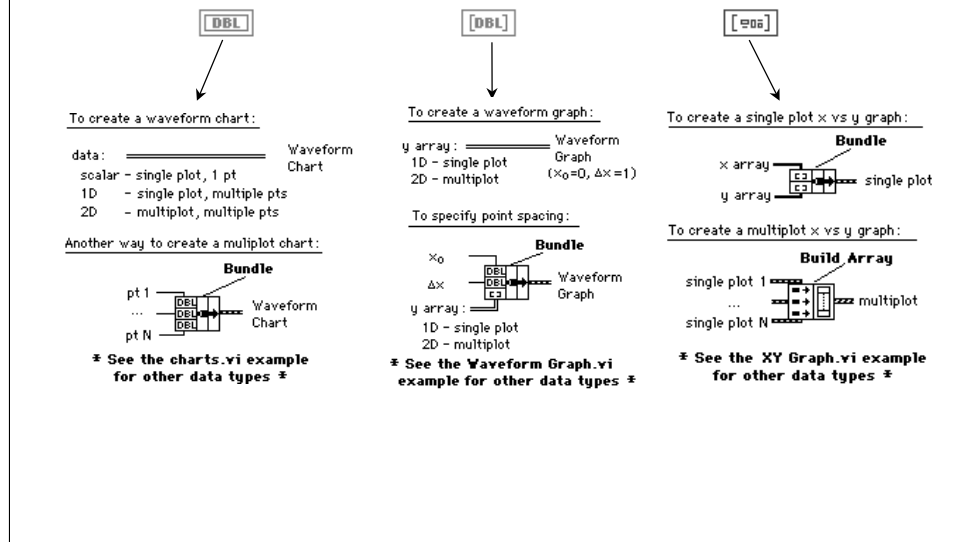
Students build Graph Circle.vi  
Time to complete : 20 min.

### Pages 5-29 to 5-30:

- Common questions:
  - **Why doesn't the graph have a circle on it?** The circle is not generated until the VI runs.
  - **How do I enter  $2 * \pi$ ?** It is a constant in the **Structures** palette **Additional Numeric Constants**.
  - **Why can't I type "Circle" into the legend without part of the word being hidden?** Move the legend to the right and resize it from the lower left corner.

## Chart and Graph Use Summary

- Use the Help Window when wiring charts and graphs



Pages 5-31 to 5-33:

- Go over the examples in detail. Be sure to discuss the purpose of the **Bundle** function.
- Demonstrate to the students how the Help window gives you valuable wiring information about charts and graphs and which one you have.

## Summary

- An array is a collection of elements of same data type
- Arrays can be of any data type – numeric, Boolean, string
- Creating array controls/indicators is a two-step process
  - First, get array shell
  - Second, place desired control/indicator inside the shell
- Loops can accumulate arrays at boundaries – auto-indexing
- Array functions are in Array subpalette of Functions palette
- LabVIEW arithmetic functions are polymorphic – input different data types
- Plot data on graphs
  - Many features to manipulate plotted graph
  - Multiple plots can be plotted on one graph

## Page 5-35:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What is an array? Where might it be used?
  - How would I create a 2D array of numeric controls?
  - How can a loop create an array? Which loops can be used to do so?
  - What does “polymorphic” mean?
  - Describe the two types of graphs available in LabVIEW.
  - What is the difference between the **Build Array** and **Bundle** functions?
- Review the slide: The purpose of Lesson Five was to introduce arrays and graphs.

### Additional Exercises on page 5-35

5-5 -- Students build **Reverse Random Array.vi**

Time to complete: 20 min.

5-6 -- Students build **Subset Random Array.vi**

Time to complete: 20 min.

5-7 -- Students build **Extract 2D Array.vi**

Time to complete: 20 min.

5-8 -- (Challenge) Students build **Die Roller.vi**

Time to complete: 35-40 min.

5-9 -- (Challenge) Students build **Array Pair Multiplier.vi**

Time to complete: 20 min.

### Page 5-36:

- 5-5 Hints:
  - Use a For Loop to build the array of numbers.
  - Then use the **Reverse 1D Array** function.
- 5-7 Hints:
  - Use two For Loops to build the 2D array.
  - Then use three **Index Array** functions and wire each array to a separate waveform graph.
- 5-8 Hints:
  - Multiply a random number by 6 and **Round to Infinity** to get a properly distributed number between 1 and 6.
  - Create an array with six elements, where each element contains the current number of results of a given 1–6 value. You can start with array(0) = the number of ones, array(1) = the number of twos, and so on.
- 5-9 Hints:
  - Modify Exercise 4-3 (**Reverse Random Array.vi**) or Exercise 4-4 (**Subset Random Array.vi**).

## **Lesson 6**

### **Case and Sequence Structures**

#### **You Will Learn:**

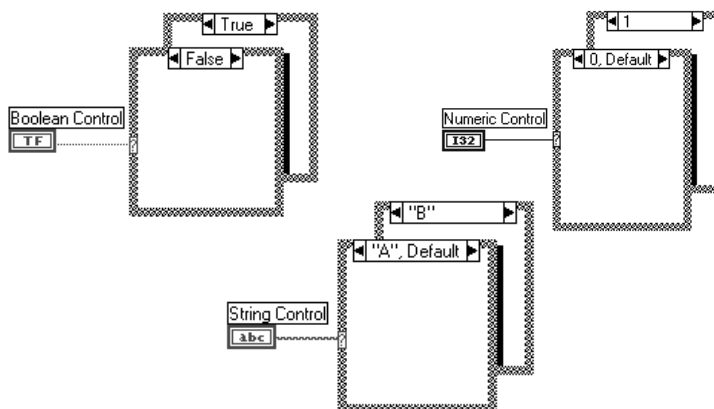
- A. About Case structures
- B. About Sequence structures
- C. About Formula Nodes

#### **Page 6-1:**

- In Lesson Four, we discussed ways to repeat certain functions using the For Loop and the While Loop. This lesson introduces three more structures to control the flow of data in your diagram
- This lesson covers:
  - Case Structures
  - Sequence Structures
  - Formula Nodes

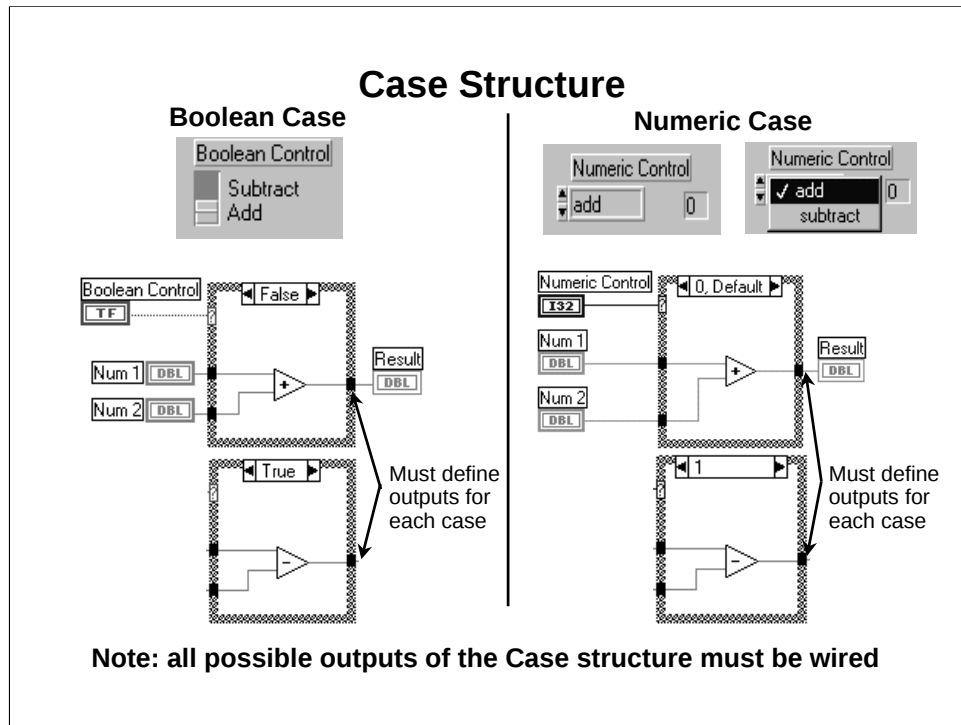
## Case Structures

- In the Structures subpalette of Functions palette
- Enclose nodes or drag them inside the structure
- Stacked like a deck of cards, only one case visible



## Page 6-2:

- The Case structure allows you to choose a course of action depending on an input value.
  - In the **Structures** subpalette of the **Controls** palette.
  - Analogous to a case statement in Pascal or if-then-else statement in other languages.
  - Like a deck of cards. You can see only one case at a time. Customers often think that when separate cases are shown, it means to create two or more case structures. Stress that you need only **one** case structure.
- Explain examples:
  - Example 1: Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.
  - Example 2: Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.
  - Example 3: String input. Briefly discuss the string input for cases.



## Pages 6-3 to 6-5:

- Discuss the examples on the slide. Be sure to mention the difference between the numeric and Boolean Case structures. Also, remind the students that all cases must provide a value for all output tunnels.
- Demonstrate using the **Text Ring** from the **List & Ring** subpalette of the **Controls** palette.
- Demonstrate some of the new features of Case structures:
  - Directly typing into the Cases
  - Setting ranges in the Case
  - Reordering the Cases
  - The Default Case

## Exercise 6-1 on page 6-6

Students build Square Root.vi

Time to complete: 20 min.

### Pages 6-6 to 6-8:

- Common questions:
  - **Why do I need a Case structure?** An error occurs if you take the square root of a negative number. You could use a **Select** function here.

**Note:** Some students will place two separate Case structures on the diagram, as seen on page 5-4. Remind them to use only one. Point out how the structures are stacked like cards. The manual displays two separate structures so that students can see what is in each one.

- **VI Logic:**

```
if (num >= 0) then
    numsqrt = SQRT (num)
else
    numsqrt = -99999
    display error message
endif
```

## Exercise 6-2 on page 6-9

Students modify Temperature Running Average (Ex. 4-5) and save as Temperature Control.vi

\*This VI will be used in a later exercise.

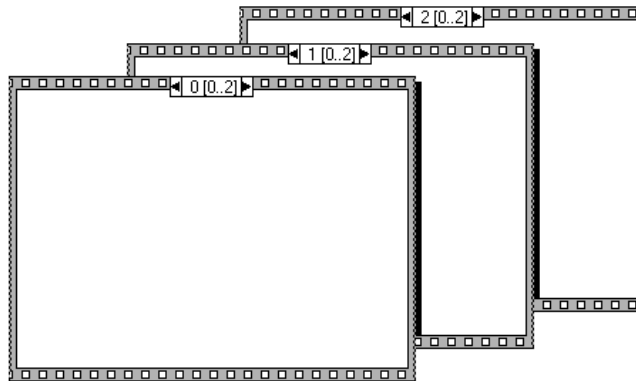
Time to complete: 25 min.

### Pages 6-9 to 6-10:

- Common questions:
  - **Why do I need to place the Beep VI in a Case structure, but not the warning light?** The warning light is a Boolean indicator and registers whatever the Boolean condition is. The **Beep VI** requires no inputs; it beeps when called.
  - **Can I show all the cases of my diagram at once?** Yes. Choose **Preview** under the **Print Documentation** window of the **File** menu.

## Sequence Structures

- In the Structures subpalette of Functions palette
- Executes diagrams sequentially, Frame 0 (0..x), where x is the total number of frames
- Stacked like a deck of cards, only one frame visible

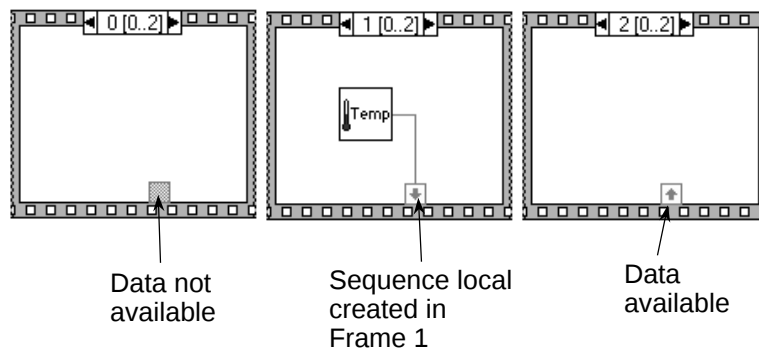


### Page 6-11:

- In a text-based language, program statements execute in the order in which they appear. In data flow, a node executes when data is available at all its input terminals.
- Sometimes it is hard to tell the exact order of execution. Often, certain events must take place before other events.
- **Sequence** structure: Used to control the order in which nodes in a diagram will execute.
  - In the **Structures** subpalette.
  - Looks like a frame of film.
  - Used to execute diagrams sequentially.
  - Like a deck of cards. You can see only one case at a time. Remind students again that when they see multiple sequence frames, it means to create a single sequence structure, not several.

## Sequence Locals

- Pass data from one frame to future frames
- Created at the border of the Sequence structure



### Pages 6-11 to 6-12:

- You cannot directly wire data from one sequence frame to any another frame with the usual wiring techniques.
- Sequence locals are variables that pass data between frames of a Sequence structure.
  - Created on the border of the Sequence frame.
  - Data wired to a sequence local is available only in subsequent frames. It is not available in previous frames.
  - In subsequent frames, you can use a sequence local to output only. If you must read a value from a sequence local, change it, and use it in a later frame, you *must* create another sequence local.
- Applications for Sequence structures:
  - Benchmarking with DAQ counters (Start Timer, Run Benchmark, Stop Timer).

### Exercise 6-3 on page 6-13

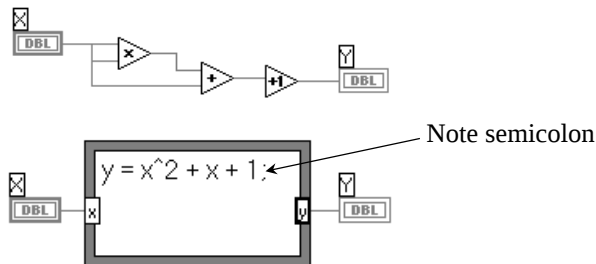
Students build **Time to Match.vi** by modifying  
**Auto Match.vi** (Ex. 4-3)  
Time to complete: 30 min.

#### Pages 6-13 to 6-15:

- Exercise 6-3 uses the **Tick Count** function for the timing of **Auto Match.vi** (this is less accurate).
- Common questions:
  - **Why do I need to use a Sequence structure?** Because you must start timing, find the match, and stop timing, in that exact order. Because it is difficult to use wires to enforce the execution order, a Sequence structure is simpler and easier.
  - **Can I show all Sequences at once?** Yes. Choose **Preview** in the **Print** option of the **File** menu.

## Formula Node

- In the Structures subpalette
- Implement complicated equations
- Variables created at border
- Variable names are case sensitive
- Each statement must terminate with a semicolon (;)
- Help Window shows available functions

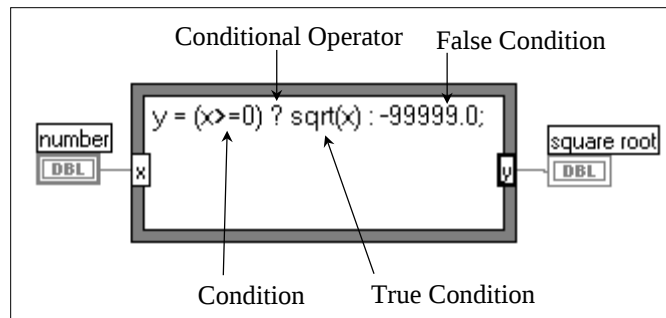


## Pages 6-16 to 6-17:

- Sometimes it is preferable to program mathematical expressions with text-based function calls, rather than icons (which can take up a lot of room on the diagram).
- Formula Node: allows you to implement complicated equations using text-based instructions:
  - Located in the **Structures** subpalette.
  - Resizable box for entering algebraic formulas directly into block diagrams.
  - To add variables, pop-up and choose **Add Input** or **Add Output**. Name variables as they are used in formula. (Names are case sensitive.)
  - Statements must be terminated with a semicolon.
  - When using several formulas in a single formula node, every assigned variable (those appearing on the left hand side of each formula) must have an output terminal on the formula node. These output terminals do not need to be wired, however.
- Compare examples on the slide.

## Conditional Branching in Formula Nodes

```
if (x >= 0) then
    y = sqrt(x)
else
    y = -99999.0
end if
```



Page 6-17:

- Explain the branching structure used for Formula Nodes. This structure is identical to the “?:” branching structure in C.

## Exercise 6-4 on page 6-18

Students build **Formula Node Exercise.vi**  
Time to complete: 20 min.

Pages 6-18 to 6-20:

- Common questions:
  - **How do I create variable names on the sides of a Formula Node box?** Pop up on the border and select **Add Input** or **Add Output**. Variable names here are case sensitive.

## Summary

- Two structures to control flow of data
  - Case structure
  - Sequence structure
- Case structure
  - Boolean or numeric cases – selector determines type
  - Subdiagrams placed inside case structure
  - Output from a Case structure must be defined for all cases
- Sequence structure executes subdiagrams sequentially
- Sequence locals pass data between frames
  - Created at the border of Sequence structure
  - Data available in subsequent frames
- Formula Nodes allow direct entry of equations in the block diagram

## Page 6-21:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What two structures control the flow of data?
  - How do you use a case structure?
  - What is a Sequence structure used for?
  - What is a Sequence local and how is it used?
  - When would it be best to use a formula node?
- Review the slide: The purpose of Lesson Six was to introduce Case structures, Sequence structures, and Formula Nodes.

### Additional Exercise on page 6-22

6-5 -- Students build **Equations.vi**

Time to complete: 20 min.

6-6 -- Students build **Calculator.vi**

Time to complete: 30 min.

6-7 -- Students build **Square Root 2.vi**

Time to complete: 20 min.

6-8 -- (Challenge) Students build **Using Array  
Over Threshold.vi**

Time to complete: 30 min.

### Page 6-22:

- This exercise does not cover any new material. Do not spend too much time discussing this exercise unless several students ask questions.
- 6-6 Hints:
  - Use a Case statement.
  - Wire the slide terminal to the selector box of the Case statement.

If you have time, go over this VI on your computer. Point out that many different applications use a slide, knob, menu ring, and so on, to determine what action should be taken.
- 6-8 Hints:
  - Use the **Array Threshold** function to determine which values are greater than a specified limit.

## **Lesson 7**

### **Strings and File I/O**

#### **You Will Learn:**

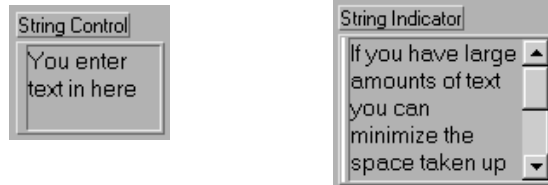
- A. How to create string controls and indicators.
- B. Some string functions.
- C. How to perform file input and output operations.

#### **Page 7-1:**

- This lesson introduces string and ASCII file I/O operations.
- File I/O involves saving collected data to a file or reading data from a file.
- This lesson covers:
  - String controls and indicators
  - String functions and their uses
  - ASCII file input and output operations

## Strings

- A string is a sequence of displayable/nondisplayable characters (ASCII)
- Many uses – displaying messages, instrument control, file I/O
- String control/indicator is in the Controls » String subpalette

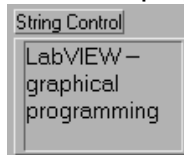


### Page 7-2:

- A string is a sequence of displayable/nondisplayable (ASCII) characters. Strings often are used to send commands to instruments, to supply information about a test (such as operator name and date), or to display results to the user.
- Strings in LabVIEW:
  - String controls and indicators are in the **String** subpalette of the **Controls** palette.
  - Enter or change text by using the Operating or Text tool and clicking in the string control.
  - Strings are resizable.
  - String controls and indicators can have scrollbars: Pop up and select **Show Scrollbar**. The scroll bar will not be active if there is not enough text in the control or indicator.

## String Display Modes

- Normal display



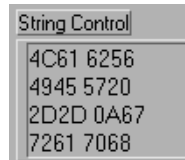
- \ code display



- Password display



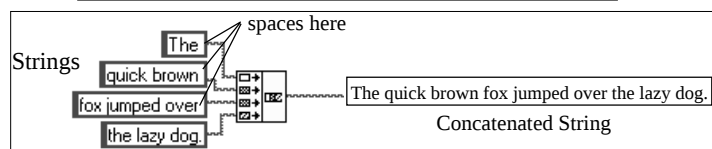
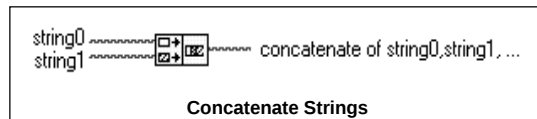
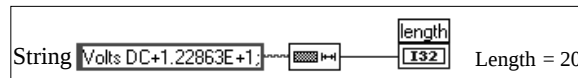
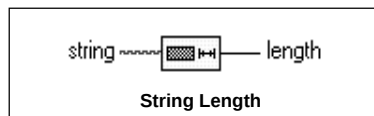
- Hex display



## Pages 7-2 to 7-3:

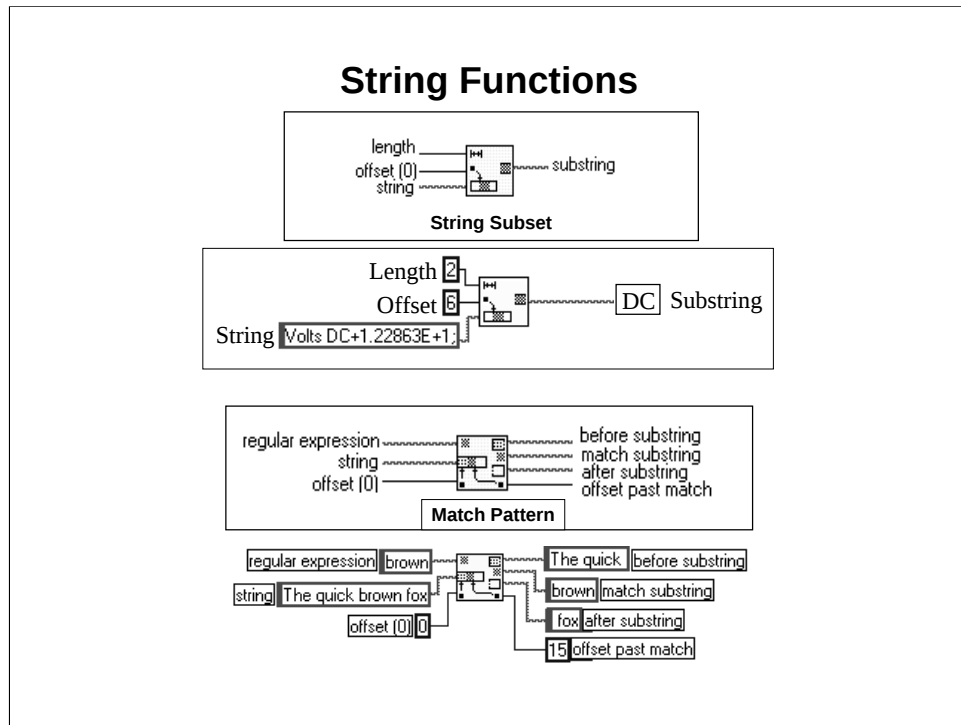
- String controls and indicators can be configured for several different display modes. You can change display modes by popping-up on the control or indicator while in edit mode.
  - **Password Display:** Asterisks replace the string text on the front panel. This is useful for concealing passwords for logging into VI.
  - **\ Codes Display:** Replaces all “unprintable” characters in the string with a ‘\’ followed by a letter. A full list of backslash codes is given on page 7-3. For unprintable characters not on this list, LabVIEW displays a \ followed by the hex value of the unprintable character (for example, “\23” = ASCII table character 23).
  - **Hex Display:** Replaces every character in the string with its ASCII table equivalent. This is not noted on the slide or in the Basics Course manual, although it is useful for serial and GPIB communication.

## String Functions



### Page 7-4:

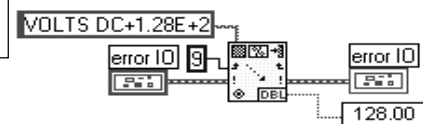
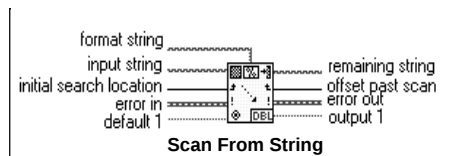
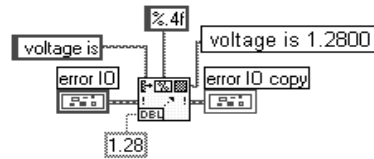
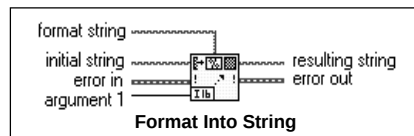
- LabVIEW has many functions to manipulate strings.
  - **String Length:** Returns the number of characters in a string, including unprintable characters.
  - **Concatenate Strings:** Concatenates input strings into a single output string. Resize the function to accommodate multiple inputs.
  - Similar in functionality to LabVIEW array functions.
- Explain the diagrams on the slide.



## Page 7-5:

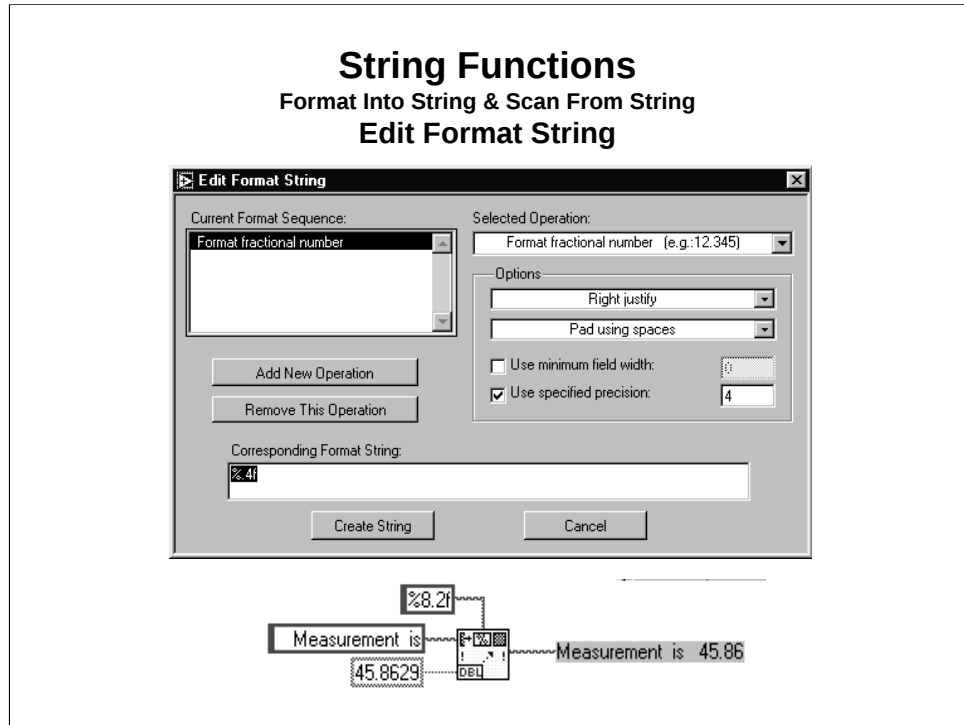
- String functions (cont.):
  - **String Subset:** Returns the substring beginning at *offset* and containing *length* number of characters. The first character is at offset zero. Unprintable characters should be counted when determining the *length* input.
  - **Match Pattern:** Returns the matched substring. The function searches for *regular expression* in the *string* input. If it finds a match, the string is split into three parts: the match, the string before the match, and the string after the match. When no match is found, *offset past match* returns a -1.
- Discuss the examples on the slide.

## String Functions



Pages 7-5 to 7-6:

- String functions (cont.):
  - **Format into String:** Converts the input *argument* into a string based on the *format string* input. The resulting string is the concatenation of the *initial string* input and the argument converted to a string. More arguments can be added by “stretching out” more input terminals.
  - **Scan from String:** Converts the *input string* containing valid numeric characters to individual numbers. Format string can specify multiple outputs with differing data types (numeric, Boolean, etc.). The function can be stretched out to add more *output* terminals.
- Explain the examples on the slide.



### Page 7-7:

- **Format into String** and **Scan from String** have an **Edit Format String** option in their pop-up menus. You can also access the format string editor by simply double-clicking on the string formatting functions.
- Allows users to easily specify the cryptic formatting strings using a simple menu interface.
- Discuss the example on the slide.
- Demonstrate the use of **Edit Format String** on your machine.

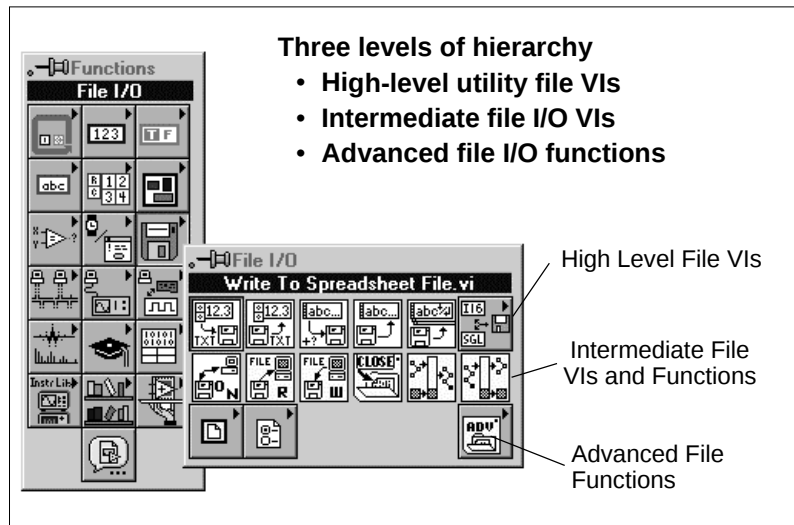
## Exercise 7-1 on page 7-8

Students build Build String.vi  
\*This VI will be used in a later exercise.  
Time to complete: 25 min.

### Pages 7-8 to 7-10:

- Common questions:
  - **Why doesn't my output string look exactly like the one in the manual?** This exercise assumes that the user types in the appropriate spaces in the header and trailer strings so that the combined string has the correct spacing. Show students how to avoid this problem using the following method:
    - Increase the size of the **Concatenate Strings** function so it has two additional inputs. Wire a string constant containing a single space to the terminals between the header and number and the number and trailer.
  - OR
  - Put a space before and after the format string so that it reads <space>%.4f<space>.

## File Input and Output

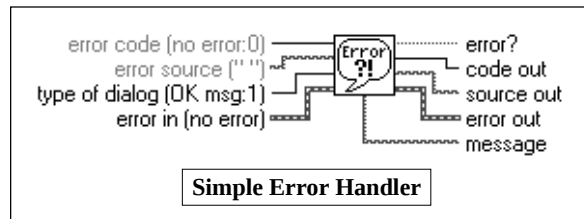


### Pages 7-11 to 7-13:

- There are three levels of file I/O hierarchy: High-Level Utility File VIs, Intermediate File I/O VIs, and the Advanced File I/O Functions.
- The high-level VIs handle all lower level functions transparently and provide a simplified means of error handling. If an error occurs, a dialog box appears showing the error.
- The Intermediate File I/O VIs provide substantially more flexibility while still handling the lowest level of functionality for you. You can handle 80-90% of file I/O operations at this level. Encourage students to use the Intermediate File I/O functions for their applications
- The Advanced File I/O functions manage all file I/O operations explicitly but are the most complex.
- We will study the Intermediate File I/O VIs and some of the High-Level Utility File VIs in this course.

## Intermediate File I/O VIs

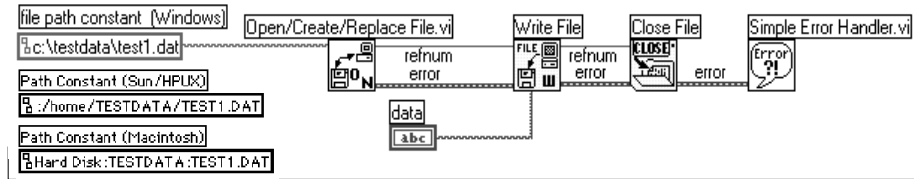
- **Open/Create/Replace file** – opens, creates, or replaces a file
- **Read File** – reads data from the file
- **Write File** – writes data to the file
- **Close File** – closes the file
- **Error handling in file I/O**
  - Time & Dialog subpalette
  - Displays a dialog box if an error occurs



## Page 7-13:

- Use **Online Reference** (Help menu) to show the connection details of each VI.
- **Open/Create/Replace File** opens or replaces an existing file or creates a new one.
- Leave **file path** unwired and you will get a dialog box.
- Read File reads **count** bytes from the file associated with **refnum**.
- Write file writes “data” to the file references by **refnum**.
- Reading begins from the location given by **pos mode**.  
The values for **pos mode** are:
  - 0: Reading starts at the beginning of the file plus **pos offset**.
  - 1: Reading starts at the end of the file plus **pos offset**.
  - 2: Reading starts at the current file points plus **pos offset**.

## Saving Data to a File



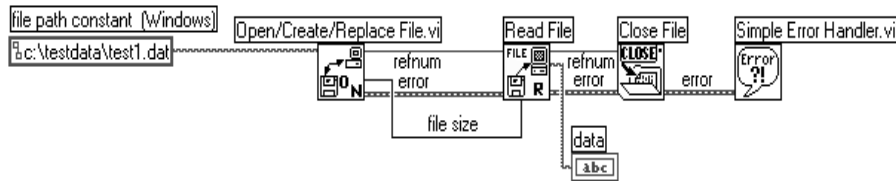
- Open/Create/Replace opens the existing file TEST1.DAT and generates refnum and error cluster
- Write File writes the data
- Close File closes the file
- Simple Error Handler checks for errors

## Pages 7-14 to 7-15:

Explain the block diagram. Stress the following:

- “Hardwiring” the filenames is strictly optional. It’s only really useful when you’re trying to save time. Most of the time you will want to leave **file path** unwired to get a dialog box.
- **refnum** is a file identifier generated by **Open/Create/Replace**. It is used by the subsequent file VIs to identify the file being operated on. The **error** cluster is a bundle of data containing error information. These clusters are a powerful method of handling errors and will be studied in more detail in the LabVIEW Advanced Course.
- Passing the **refnum** and **error** cluster also forces the VIs to execute in order.

## Reading Data from a File



- Open/Create/Replace opens the file
- Read File reads the specified number of bytes from the file
- Close File closes the file
- Simple Error Handler checks for errors

## Page 7-15:

Explain the block diagram. Emphasize the following:

- The user *must* tell the **Read File** VI how many bytes of data to read by wiring that number of bytes to the **count** terminal.
- Because **file size** is wired to **count** from the **Open/Create/Replace** VI, the entire file is read by the **Read File** VI.

## Exercise 7-2 on page 7-16

Students build File Writer.vi

Time to complete: 20 min.

### Pages 7-16 to 7-17:

- Common questions:
  - **What is happening when I run this VI?** Since you didn't wire a specific path name to the Open/Create/Replace File VI, it pops up a dialog box to ask you what file to save the data to.  
  
NOTE: Do not try to save your data file into BASICS.LLB or it will overwrite all the work you have done so far.
- **Instructor:** Make sure the students understand why BASICS.LLB is not a place to save data files.

## Exercise 7-3 on page 7-18

Students build File Reader.vi

Time to complete: 20 min.

### Pages 7-18 to 7-20:

- Common questions:
  - **Why doesn't this VI read the data file?** There are two possible reasons why you aren't getting the returned data string.
    - The File Writer VI didn't actually write the data to file.
    - You forgot to wire the byte count from the Open/Create/Replace File to the Read File.
  - **How do I get those things in the False case?** Make sure the True case is wired first. Then go to the False case and pop up on each white tunnel and select **Create Constant**.

## Creating a Spreadsheet File

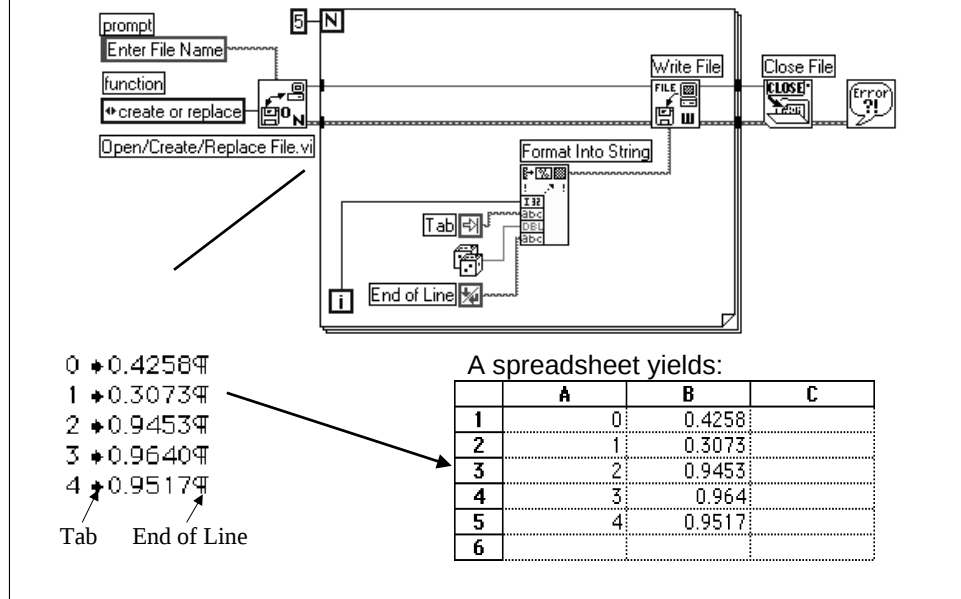
- Spreadsheets are popular tools for data handling and analysis
- There are many formats for spreadsheet data. One of the most popular is *tab-delimited*:
  - Columns are separated by a tab character
  - Rows are separated by an end-of-line character
- Easily created using LabVIEW file I/O VIs

### Page 7-21:

Read from the slide.

- Depending on the spreadsheet application, the user may need to use a different delimiter.
- An example is on the next slide.

## Creating a Spreadsheet File



Pages 7-21 to 7-23:

Explain the block diagram. Emphasize the following:

- Although there are multiple writes to the file, with the Intermediate File I/O VIs, we can make sure the file is opened and closed only once. This saves time.
- **Format into String** converts each numeric to a string and inserts a tab character and end-of-line character. The output is a tab-delimited string compatible with most spreadsheet programs.
- Mention the table structure. A table is a front panel object used to display a 2D array of strings. The table looks much like a spreadsheet.

### **Exercise 7-4 on page 7-24**

Students build Temperature Logger.vi by  
modifying Temperature Running Average.vi

\*This VI will be used in a later exercise.

Time to complete: 25 min.

**Pages 7-24 to 7-27:**

**Instructor:** Please write down common questions and the time required to complete this exercise. Record any information in the CAR database.

## High-level File I/O VIs

- **Write to Spreadsheet File**
- **Read from Spreadsheet File**
- **Write Characters to File**
- **Read Characters from File**
- **Read Lines from File**

Pages 7-28 to 7-29:

- **Write to Spreadsheet File:** Converts a 1D or 2D array of numerics to a single-precision, tab-delimited string and writes the string to an ASCII file. Useful for quickly creating ASCII spreadsheet files as in the last slide.
- **Read from Spreadsheet File:** Reads a tab-delimited ASCII file and converts the data back to a 1D or 2D array of numerics. Useful for reading data created with **Write to Spreadsheet File**, or data saved as an ASCII file from a spreadsheet program.
- **Write Characters to File:** Writes a character string to a file.
- **Read Character from File:** Reads a character string from a file.
- **Read Lines from File:** Read a specified number of lines from an ASCII file.
- All of the high-level VIs take care of opening/creating/replacing the file, reading/writing the file, and closing the file in one VI.

## Exercise 7-5 on page 7-30

Students run **Spreadsheet Example.vi**  
Time to complete: 25 min.

### Pages 7-30 to 7-33:

- Common questions:
  - **Why is the Transpose 2D Array function used?** This way, each different graph is written to a separate column rather than a row. This makes it easier to read on a spreadsheet (paging down rather than across).
- Point out that the **Write to Spreadsheet File** VI opens the file, converts all the numbers into ASCII strings, inserts the appropriate tabs and linefeeds, writes the data to the file, and closes the file.
- If this VI is used in a loop, the file will be opened, written to, and closed during every iteration. This process is much slower than opening the file only once, before the loop, and closing it only once, after the loop.

## **Exercise 7-6 on page 7-34**

Students build Temperature Application.vi  
Time to complete: 30-40 min.

### **Pages 7-34 to 7-35:**

- This exercise uses all of the concepts covered in the class up to this point. It will probably overwhelm slower students.
- Review the following points to help students get started:
  - Encourage faster students to work on this exercise instead of working ahead.
  - Stress that this exercise is good practice for building an application from scratch.
- Suggest that the students plan each step in LabVIEW. You could make it a group participation lesson and have the class help you write the objectives of this exercise on the board, or even help you write the VI on your computer.
- When possible, copy and paste from existing VIs as described in the hints.

## Summary

- String is a collection of ASCII characters – many uses
  - Display messages
  - Instrument control
  - File I/O
- Many functions to manipulate strings – strings palette of Functions menu
- Three levels of file I/O hierarchy
  - High-level file I/O VIs
  - Intermediate file I/O VIs and functions
  - Advanced file I/O functions
- Writing data in spreadsheet format
  - Tab character separates columns
  - End-of-line character separates rows
  - Write/Read from spreadsheet high-level utility file I/O VIs

## Page 7-36:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What are some uses of strings?
  - How are files opened using the Intermediate File I/O VIs?
  - When is it preferable to write data to a file in ASCII format?
  - In binary integer format?
  - What two ways can you write to a spreadsheet file?
- Review the slide: The purpose of Lesson Six was to introduce string and file I/O operations.

### Additional Exercises on page 7-37

7-7 -- (Challenge) Students build Spreadsheet Exercise.vi

Time to complete: 25 min.

7-8 -- (Challenge) Students build Spreadsheet Converter.vi

Time to complete: 30 min.

7-9 -- Students modify Temperature logger.vi (Ex. 7-4) and save the new VI as Temperature Logger 2.vi

Time to complete: 25 min.

### Page 7-37:

- Do not spend class time to finish these exercises. However, because it is an important concept that many students may use in their applications, discuss the technique and show the solution on your computer, if there is time.
- 7-7 Hints:
  - Use the **Write Characters to File** VI to first write the headers. Then use the **Write to Spreadsheet File** VI to write the numbers.
  - To prevent the file dialog box from popping up twice, wire the **file path** output of the **Write Characters to File** VI to the **file path** input of the **Write To Spreadsheet File** VI.
- 7-9 Hints:
  - Modify the **Temperature Logger** VI so that the **pos mode** is 1 and **pos offset** is 0.

## Lesson 8

### VI Customization

You Will Learn:

- A. The use of VI Setup
- B. The use of SubVI Node Setup
- C. How to edit VIs with disabled options
- D. About customizing palettes

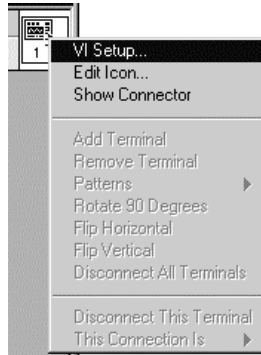
#### Page 8-1:

- Lesson 8 will cover some tools used to customize your panels.
- **VI Setup** is used to customize a given VI.
- **SubVI Node Setup** is used to customize a specific instance of a subVI.
- Sometimes these Setup options can prevent you from editing your VIs. You will learn how to prevent this situation.
- You can fully customize the Controls and Functions palettes in LabVIEW.

## VI Setup

Access VI Setup... by popping up on icon pane

- Affects every instance of that VI in all applications
- VI Setup menu
  - Execution options
  - Window options
  - Documentation

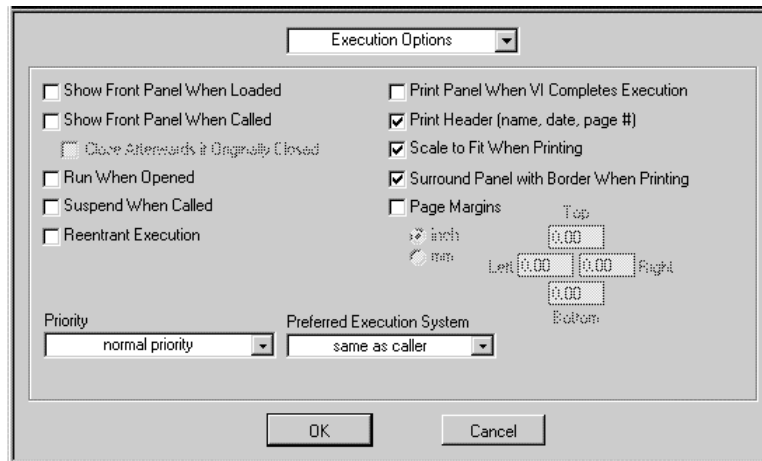


### Page 8-2:

- You can access **VI Setup...** from the panel or diagram window in the icon pane.
- Three options are available:
  - **Execution Options** - Used to set up when the front panel is opened, whether the VI is reentrant, and the priority of the VI.
  - **Window Options** - Allow you to disable various pieces of the panel window to customize the window.
  - **Documentation** - Allows you to modify help and documentation features of the VI. You can associate help files with the VI from this window.

## VI Execution Options

- Only affects VI while in Run mode

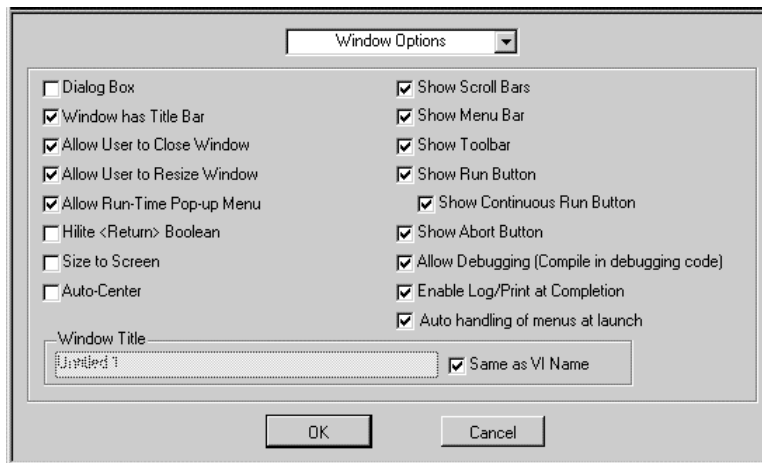


## Pages 8-2 to 8-3:

- Discuss the more commonly used features in the Execution Options window:
  - Use **Show Front Panel...Called** and **Close Afterwards...** to create pop-up panels.
  - Use **Run When Opened** to make the VI automatically run when the user opens its panel.
  - Use **Suspend When Called** for debugging (same as setting breakpoint on tools palette of VI).
  - **Reentrant Execution** provides separate data space for the subVI.
  - Various printer setup options are available for programmatic printing of a VI.
  - **Priority** and **Preferred Execution System** determine low-level multitasking and time sharing aspects of a VI. This is discussed further in the LabVIEW Basics II Course.

## VI Window Options

- Only affects VI while in Run mode

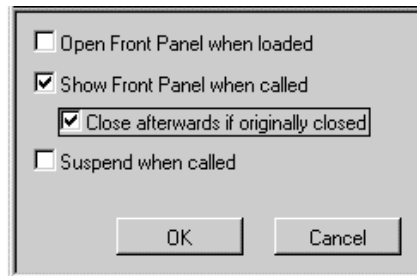


### Pages 8-4 to 8-5:

- Discuss the more commonly used features in the Window Options window:
  - Showing/hiding scroll bars.
  - Showing/hiding the title bar.
  - Showing/hiding menu bars.
  - Showing/hiding toolbar or various toolbar options like debugging, aborting, or continuous running.
  - Making the VI act as a dialog box. (Discuss the implications of being a dialog box and why they may or may not want this option for their VIs.)

## SubVI Node Setup

- Access SubVI Node Setup by popping up on subVI icon on calling VI's diagram
- Affects only that single instance of the subVI

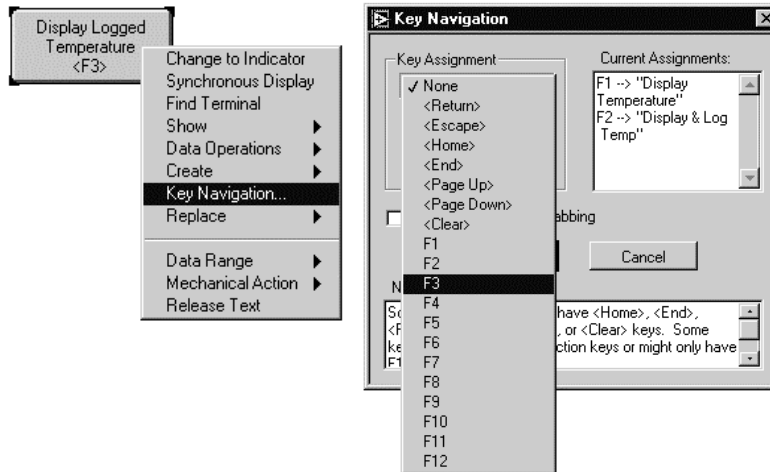


Page 8-8:

- **SubVI Node Setup** is used to set execution options for a single instance of a subVI.
- Does not affect other calls to that same subVI in the diagram. (**VI Setup** options affect every instance of that VI.)

## Key Navigation

- Assigns key assignments to front panel controls



## Page 8-13:

- **Key Navigation** can be accessed from the pop-up menu of any front panel control. The operation of the control can be associated with specific keys or key combinations.
- Not in course but mention if there is time:
  - When running a VI, you can hit the tab key to enable key focus to the front panel controls
  - To key focus inside arrays and clusters - use <ctrl> + down arrow (to step into array or cluster) and <ctrl> + up arrow (to step out of array or cluster). Then use tab to key focus each element.

## Exercise 8-1 on page 8-9

Students build Pop-Up Graph.vi and  
Use Pop-Up Graph.vi

Time to complete: 20-30 min.

### Pages 8-9 to 8-13:

- Common questions:
  - **What is the difference between specifying subVI options in “VI Setup” and “SubVI Node Setup”?** **VI Setup** affects the VI every time it is called. **SubVI Node Setup** affects the VI only when it is called in the location that it was modified.
- Make sure the students understand the purpose of using **SubVI Node Setup** in this exercise—to make a pop-up graph after all of the values are obtained. This method is useful when there is not enough room on the panel to display all of the necessary indicators. Use pop-up panels to display objects when they need to be viewed.
- When students finish, discuss the diagram on your computer.

## Exercise 8-2 on page 8-14

Students build Temperature System.vi

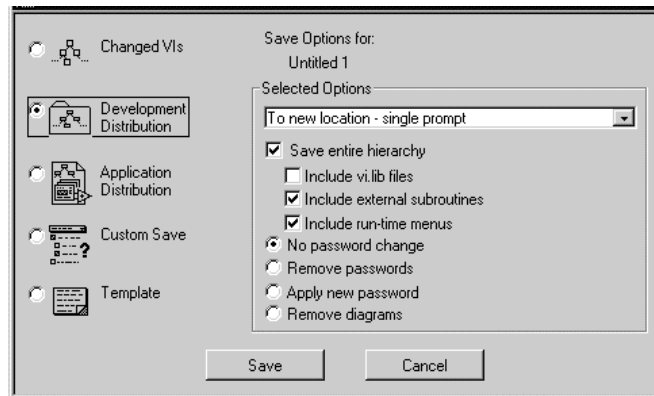
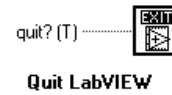
Time to complete: 25-30 min.

### Pages 8-14 to 8-17:

- **Instructor:** You may want to discuss **Key Navigation** before students begin this exercise.
- Common questions:
  - **Why isn't anything happening when I press the Display & Log Temp button?** You must use **SubVI Node Setup** to configure the **Display & Log Temp** subVI to pop open its front panel when called.
- Point out that by disabling the Tools palette, this VI could be used by someone not highly trained. It becomes an easy-to-use menuing system.
- Go over the solution of this exercise when the students are finished and make sure they understand the concepts.

## Combination of VI Setup Options

- Opens, runs, and closes LabVIEW without user intervention
- Save with Options... to make backups



## Pages 8-18 to 8-19:

- Explain to the students that when you change the VI Setup options in a certain combination, they can make it almost impossible to edit their VI again. For example, Run When Opened, all menu bars and toolbars disabled, and Exit LabVIEW function.
- Show the Save with Options window and encourage the students to save a copy of their VIs (and the hierarchy) to a new location before they start modifying the VI Setup options.

### Exercise 8-3 on page 8-20

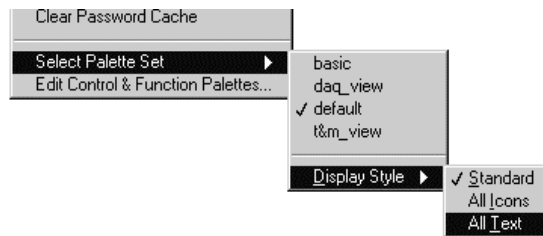
Students modify Edit\_Me.vi  
Time to complete: 25-30 min.

#### Pages 8-20 to 8-21:

- **Instructor:** Run this VI and explain what it is doing before the students begin the exercise.
- Common questions:
  - **This is a new exercise, so please enter any questions and answers into the CAR database.**
- Go over the solution of this exercise when the students are finished and make sure they understand the concepts.

## Customizing LabVIEW Palettes

- Access different palette sets from the Edit menu
- Display either icon or text Controls and Functions palettes

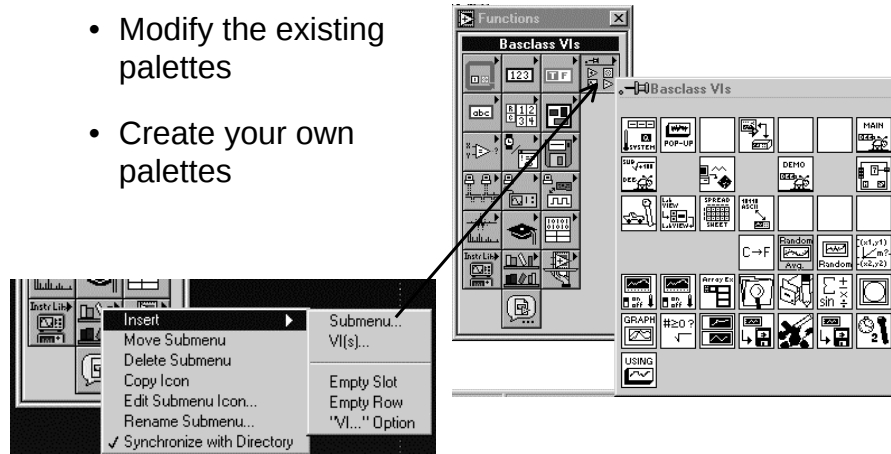


### Page 8-22:

- This section of Lesson 8 is optional. Therefore, only discuss it if the class is ahead of schedule. Otherwise, skip it and tell them to read this section when they have spare time.
- Demonstrate to the students how they can pick other palette sets like T&M View or DAQ View.
- Also show how they can get the all Text menus for the Controls and Functions palettes from the **Display Style** option in **Edit » Select Palette Set**.

## Editing Controls & Functions Palettes

- Modify the existing palettes
- Create your own palettes



### Pages 8-22 to 8-24:

- This section of Lesson 8 is optional. Therefore, only discuss it if the class is ahead of schedule. Otherwise, skip it and tell them to read this section when they have spare time.
- Demonstrate to the students how they can modify the existing palettes using **Edit » Edit Control & Function Palettes** and moving the icons around in the User Libraries » Basic Course palette.
- Demonstrate how the students can make their own palette from **Edit » Edit Control & Function Palettes** and putting BASICS.LLB into the Functions palette. (This is the next exercise.)

### **Exercise 8-4 on page 8-26 (Optional)**

Students modify the Functions Palette  
Time to complete: 20-25 min.

#### **Pages 8-26 to 8-28:**

- This exercise is optional, so only do it if the students are ahead of schedule or are very interested in this particular feature.
- Common questions:
  - **This is a new exercise, so please enter any questions and answers into the CAR database.**

## Summary

- Use VI Setup to set Execution, Window, and Documentation Options for every instance of a VI.
- Use SubVI Node Setup to set execution options for a single instance of a subVI.
- Use the Key Navigation option to assign front panel controls to a keyboard key combination.
- Use the Save with Options... from the file menu to make backups of your VIs.
- You can often edit VIs by aborting them from the diagram of another VI.
- You can fully customize the Controls and Functions palettes from the Edit menu.

## Page 8-29:

- Do not show this slide immediately.
- Discussion questions:
  - What is the difference between **VI Setup** and **SubVI Node Setup**?
  - How do you make a subVI open its front panel when it is run?
  - Why should you use the Save with Options before modifying your VI Setup options?
  - How can you edit a VI that opens, runs, and closes or is locked?
  - In what ways can you customize the Controls and Functions palettes?
- Review the summary slide.

## **Lesson 9**

### **Data Acquisition**

#### **You Will Learn:**

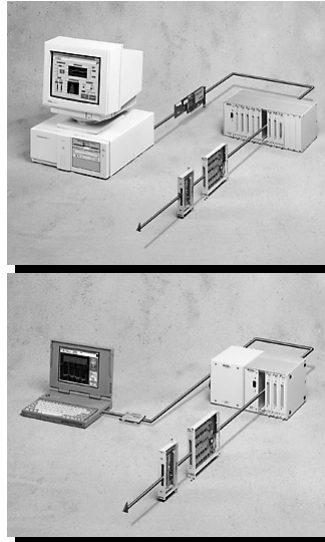
- A. About plug-in data acquisition (DAQ) boards.
- B. About the organization of the DAQ VIs.
- C. How to acquire and display an analog signal.
- D. How to output an analog signal.
- E. How to acquire data from multiple analog channels.
- F. How to drive the digital I/O lines.
- G. About buffered data acquisition.

### **Page 9-1:**

- This lesson introduces data acquisition.
- This lesson covers:
  - A brief discussion of boards and DAQ concepts.
  - The organization of LabVIEW DAQ VIs.
  - How to acquire an analog reading.
  - How to output an analog signal.
  - How to acquire analog readings from multiple channels.
  - How to perform digital I/O.
  - How to acquire data continuously.

## Overview

- DAQ library supports all DAQ boards
- LabVIEW uses the NI-DAQ driver-level software
- DAQ boards for
  - Analog I/O
  - Digital I/O
  - Counter/timer I/O
- Data acquisition system components

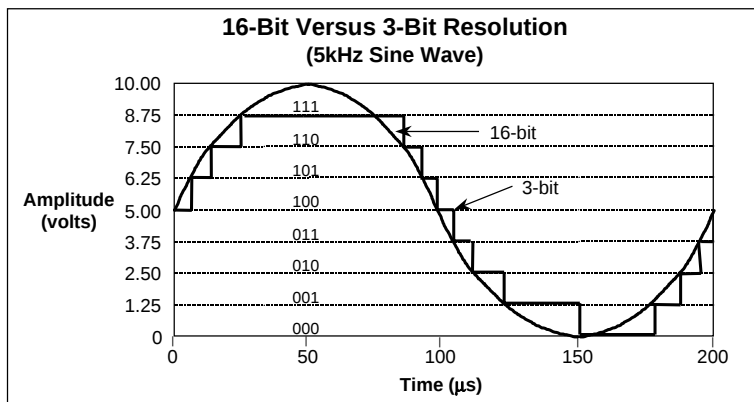


## Page 9-2:

- Discuss components:
  - Transducers - Convert physical measurements to electrical signals.
  - SCXI - Signal conditioning (amplifying, linearization, etc.).
  - DAQ Boards - Talk briefly about the boards listed in the manual (mention DAQ Designer to determine what boards might be needed).
- The LabVIEW DAQ library supports all NI DAQ boards:
  - The same VIs are used regardless of board.
  - Choose a DAQ board that best fits the application.
- LabVIEW uses NI-DAQ software:
  - DAQ functions can be upgraded without needing to replace LabVIEW VIs.
  - Fixes, new functions, and updates are greatly simplified.

## Analog Input Considerations

- Single-Ended vs. Differential
- Resolution
- Range
- Gain



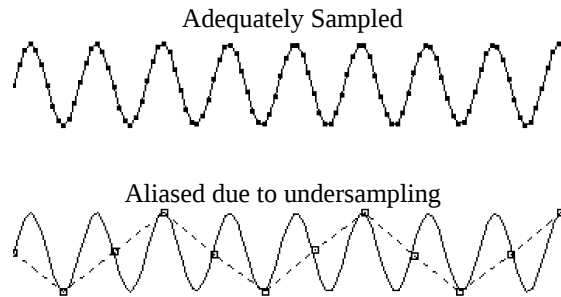
### Pages 9-3 to 9-4:

- When measuring signals with a DAQ board, you must consider several factors.
  - **Single-ended (SE) versus differential (DIFF):**
    - SE inputs are referenced to a common ground.
    - Three conditions must be met for SE inputs:
      1. Input signals are greater than 1 V.
      2. Leads from signal source to analog input hardware are short (less than 15 feet).
      3. All inputs share common ground reference.
    - If your signals do not meet these criteria, DIFF inputs should be used.
    - DIFF inputs each have their own ground source and reduce or eliminate noise errors.
  - **Resolution** - Number of bits that the DAQ board's (A/D) converter can use to represent the analog signal. Point out on slide the difference between 3-bit vs. 16-bit resolution.
  - **Range** - Voltage range that the board's ADC can accept as inputs.
  - **Gain** - Hardware amplification applied to the incoming signal before it is digitized to maximize the number of resolution divisions available to digitize the signal.

## Analog Input Considerations

- Code Width  $\frac{10}{1 * 2^{12}} = 2.4 \text{ mV}$        $\frac{20}{1 * 2^{12}} = 4.8 \text{ mV}$

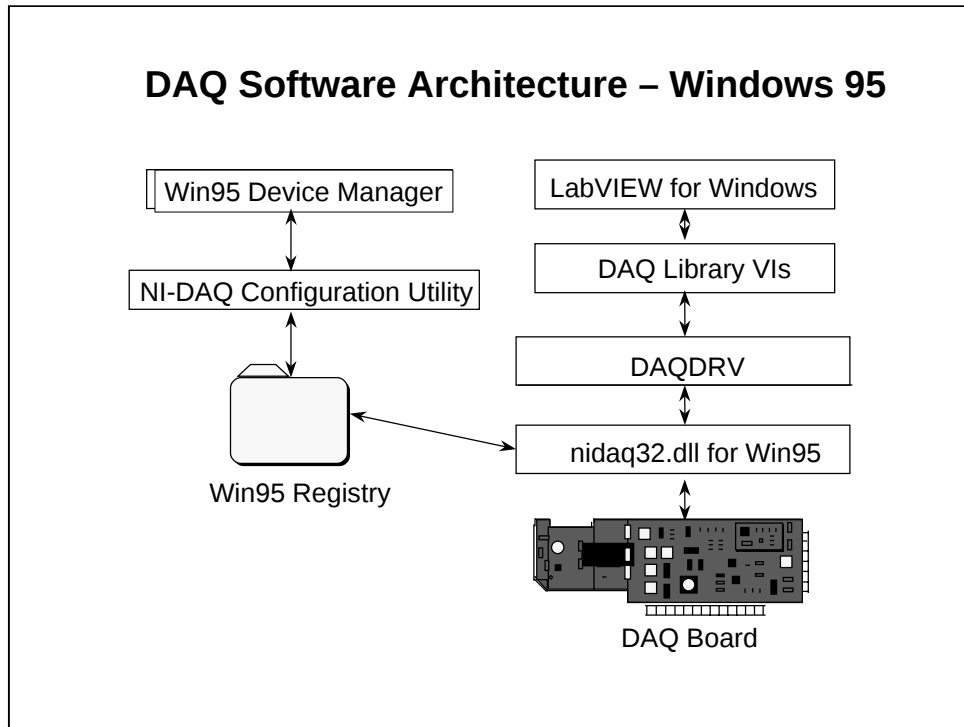
Sampling Rate



Averaging

Pages 9-4 to 9-5:

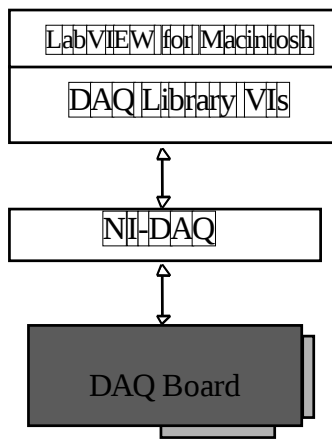
- When measuring signals with a DAQ board, you must consider several factors.
  - **Code Width** equation - Combine the previous quantities of resolution, range, and gain to calculate the minimum voltage at which the DAQ board can digitize the signal (code width).
  - **Sampling Rate** - The number of times per second that the A/D conversion takes place. Fast sampling rates produce a better representation of your signal.
    - It is recommended to use the Nyquist Sampling Theorem: Sample at a frequency at least twice as fast as the highest frequency component of your incoming signal.
    - If the sampling rate is not fast enough, the signal will become aliased.
  - **Averaging** - Take more points and average them to reduce noise.



### Pages 9-5 to 9-9:

- When measuring signals with a DAQ board, you must consider several factors.
  - NI-DAQ can be installed through the LabVIEW installation.
  - Installs DLL and NI-DAQ Configuration Utility and modifies the system registry.
  - System resources can be configured through the Device Manager (**Start » Settings » Control Panel » System**).
  - Board configuration can be done and tested through the **NI-DAQ Configuration Utility** (LabVIEW program group).
  - You must configure the board before you can use LabVIEW DAQ VIs.

## DAQ Software Architecture – Macintosh

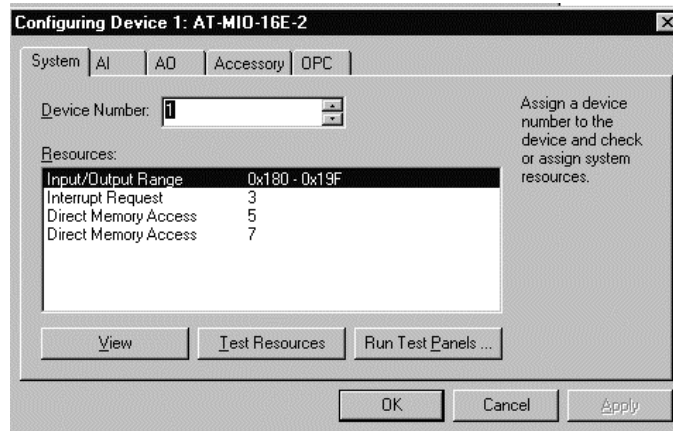


### Page 9-11:

- When measuring signals with a DAQ board, you must consider several factors.
  - NI-DAQ can be installed through the LabVIEW installation.
  - NI-DAQ Configuration Utility in the NI-DAQ folder detects the board and can be used for configuring the board.
  - You must configure the board before using it with LabVIEW.

## DAQ Hardware Configuration

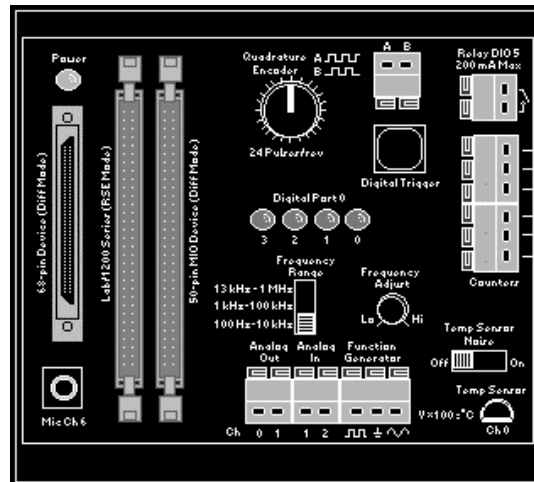
- NI-DAQ Configuration Utility



### Page 9-12:

- Windows 95 - The LabVIEW Setup program copies the required files to your hard disk:
  - NIDAQ32.DLL into Windows directory
  - NI-DAQ Configuration Utility into LabVIEW directory/folder
- NI-DAQ Configuration Utility is used to configure DAQ boards. The utility in conjunction with Windows 95 Device Manager (System Properties) allows configuration of DAQ board
- Macintosh - The LabVIEW installer installs NI DAQ
- You must configure your board before you can use the LabVIEW DAQ VIs.
- **Instructor** -- Run the NI-DAQ Configuration Utility and point out the features. Be sure to show them the Help.

## The DAQ Signal Accessory



### Page 9-12:

- Before the class starts Exercise 9-1, you should explain the following points about using the DAQ Signal Accessory:
  - The 5V line from the DAQ device supplies power to the DAQ Signal Accessory.
  - Analog input channels 1 and 2 are available at the quick connect terminal located in the lower central portion of the top panel. The IC temperature sensor is hard-wired to AI channel 0 and is located in the lower right hand corner of the top panel.
  - Analog output channels 0 and 1 are available at the quick connect terminal located at the lower central portion of the top panel.
  - The function generator is located in the lower central portion of the top panel. It produces a sine wave and a square wave. Use the Frequency Range selector switch to select a frequency range and then use the Frequency Adjust knob next to it to further adjust the frequency.
  - There is a row of four LEDs in the middle of the DAQ Signal Accessory that correspond to Lines 0 to 3 on Port 0 of the MIO board. The four LEDs use inverted logic.
  - Briefly describe the other parts of the DAQ Signal Accessory.
- **Instructor** -- make sure each student has at least one wire with stripped ends.

## Exercise 9-1 on page 9-12

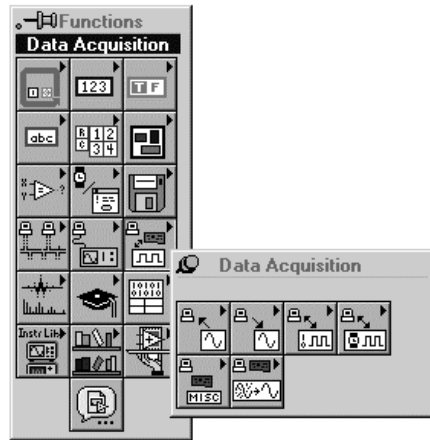
Students examine the DAQ Configuration Utility

Time to complete: 15 min.

### Pages 9-12 to 9-17:

- Common questions:
  - **Why don't the LEDs work properly in the Test panel?** The LEDs on the DAQ Signal Accessory use negative logic.
- **Note:** If students make any changes, they should not save them.

## DAQ Functions Organization

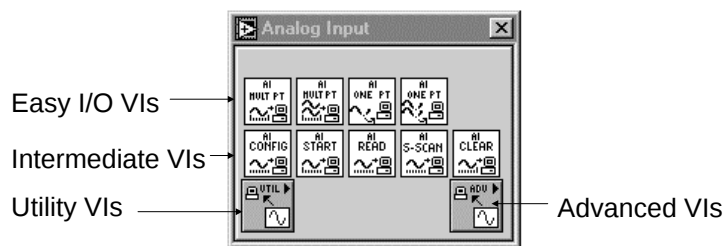


- Analog Input
- Analog Output
- Digital I/O
- Counter
- Calibration and Configuration
- Signal Conditioning

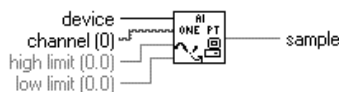
### Pages 9-18 to 9-19:

- LabVIEW DAQ VIs are divided into three categories:
  - Easy I/O: High-level VIs for basic analog operations. Ideal for simple data acquisition applications or getting started with DAQ in LabVIEW.
  - Intermediate: For more hardware functionality, flexibility, and efficiency.
  - Advanced: Low-level calls to the NI-DAQ driver.
- Point out that there is only one set of VIs for *all* National Instruments boards. You do not need to learn a separate set of VIs for each board. (**Note:** The same VIs are used for all platforms.)

## Analog Input VI Organization

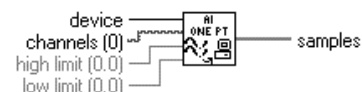


- Single-point VIs
  - AI Sample Channel



**AI Sample Channel.vi**

- AI Sample Channels



**AI Sample Channels.vi**

## Page 9-20:

- The simplest of the Easy I/O VIs. They perform a single A/D conversion on one or more channel:
  - **AI Sample Channel:** Measures a signal from a specified channel and returns a voltage.
  - **AI Sample Channels:** Measures a signal from each of the specified channels and returns an array of voltages (one for each channel sampled.)
- Explain the inputs and outputs:
  - Parameters in **boldface** are required.
  - The channel input is a string and the device input is a numeric.
  - Parameters in regular text are optional.
  - Values in parentheses are default values.
- If an error occurs with any Easy I/O VI, a dialog box displays the error code and gives you the option to abort or continue.

## Exercise 9-2 on page 9-21

Students build Voltmeter.vi

\*This VI will be used in a later exercise.

Time to complete: 20 min.

### Pages 9-21 to 9-22:

- Common questions:
  - **Why do I get a bad wire when I try to connect the Channel control?** You can connect the Channel control only if it is a *string*, not a number.
  - **When I run the VI, why do I get a bad device error?** Make sure that you have set the Device control to 1 and the channel set to a value string such as “0” or “1”.
  - **Why does my meter display approximately 0.26?** You are taking readings from Channel 0, the Temperature channel. If you place your finger on Channel 0 of the DAQ Signal Accessory, you will see the reading increase.

### Exercise 9-3 on page 9-23

Students modify Measurement Averaging.vi

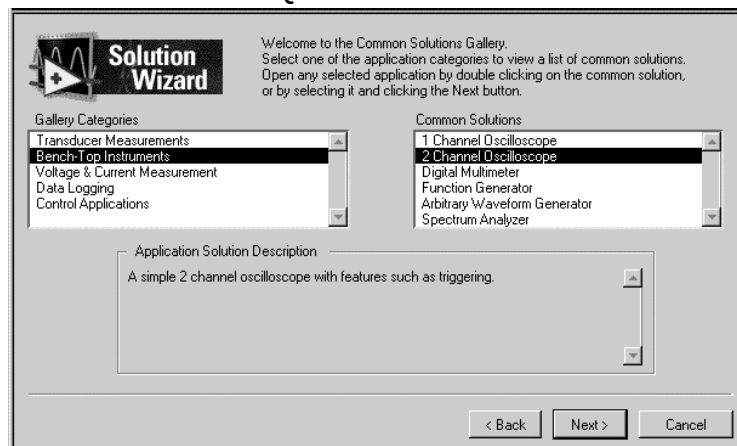
Time to complete: 20 min.

### Page 9-24:

- Common questions:
  - **How do I modify the VI to take 30 readings?** Place a For Loop that executes 30 times in the True case, and place the **AI Sample Channel** VI inside of the loop. Use either the **Mean** function or the **Sum Array Elements** and divide by 30, to take the average of those 30 readings, and plot the average to the chart.

## DAQ Wizards

- DAQ Channel Wizard
- DAQ Solution Wizard



### Page 9-24:

- LabVIEW DAQ Wizards are utilities that help you define the signals and quickly create DAQ applications:
  - **DAQ Channel Wizard:** Consists of several pop-up windows where you select which signals are connected to which channels on your DAQ board. For example, if you specify a channel named “temp” for a thermocouple, fill out what voltage/temperature ranges it supports and what CJC to use, and every time you take a reading from channel “temp” it will automatically be compensated and scaled to the correct temperature scale.
  - **DAQ Solution Wizard:** Has several categories of applications already prebuilt or you can specify a custom DAQ application and a completed VI will open with all the correct DAQ VIs, inputs/outputs, displays, etc.

### **Exercise 9-4 on page 9-25**

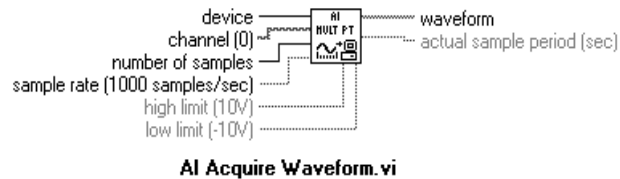
Students use the DAQ Solution Wizard  
Time to complete: 20 min.

Page 9-25:

- Common questions:
  - **This is a new exercise. Please enter any questions and answers into the CAR database.**

## Analog Input Waveform VI

- AI Acquire Waveform
- VIs display a dialog box if an error occurs



### Page 9-29:

- **AI Acquire Waveform:** Measures multiple A/D conversions from a specified channel and returns an array of voltages.
- Explain the inputs and outputs:
  - Parameters in **boldface** are required.
  - The channel input is a string and the device input is a numeric.
  - Parameters in regular text are optional.
  - Values in parentheses are default values.
- If an error occurs with any Easy I/O VI, a dialog box displays the error code and gives you the option to abort or continue.

## Exercise 9-5 on page 9-30

Students Build Acquire Waveform.vi

Time to complete: 25 min.

### Pages 9-30 to 9-32:

- Common questions:
  - **Why can't I wire to the waveform graph?** Make sure that it is not a chart or an XY graph.
  - **Why is the "actual sample period" wired to the "Bundle" function?** So that the X axis on the graph is accurate; it is the delta x value. If you did not use the **actual sample period**, the waveform would look correct, but the information on the X axis might not accurately reflect the sampling rate.
  - **Why is the computer hung or Why don't I get data?** Make sure that the customers enter in realistic values for the # of points to acquire and the sampling rate. They may be asking for 10,000 data points at 100 points/sec.
- Instructor -- make sure the students have signals wired to whatever channel they are sampling on the DAQ Signal Accessory. Also, make sure they don't try to put the data file into BASICS.LLB.

## Exercise 9-6 on page 9-33

Students build Read Waveform From Disk.vi

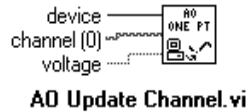
Time to complete: 15 min.

### Pages 9-33 to 9-34:

- Common questions:
  - **Why do I need to enter the Samples/sec value?** You did not save the Samples/sec value when you saved the waveform to a file in Exercise 9-5. So now, to generate a graph with the correct X axis information, you must specify the rate at which the data was taken.
  - **Why do I need to take the reciprocal of the Samples/sec value?** To calculate the actual amount of time between each reading.

## Analog Output VIs

- Single-point VI



- Waveform Generation VI



## Page 9-35:

- The simplest of the Easy I/O VIs. They perform a single D/A conversion or multiple conversions on one channel:
  - **AO Update Channel:** Writes a specified voltage value to the analog output channel.
  - **AO Generate Waveform:** Writes a specified array of voltage values to the analog output channel.
- Explain the inputs and outputs:
  - Parameters in **boldface** are required.
  - Parameters in regular text are optional.
  - Values in parentheses are default values.
- If an error occurs with any Easy I/O VI, a dialog box displays the error code and gives you the option to abort or continue.

## Exercise 9-7 on page 9-36

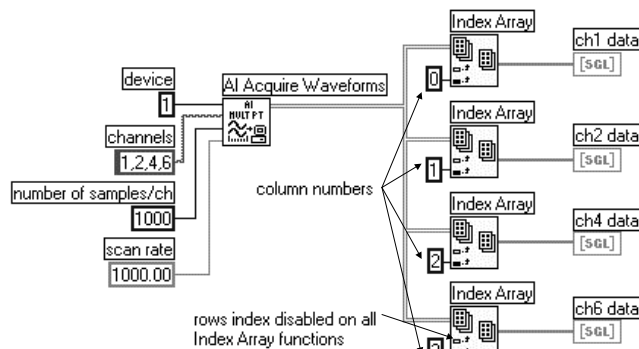
Students examine and run Voltage Output  
Example.vi with Voltmeter.vi

Time to complete: 15 min.

### Pages 9-36 to 9-38:

- Common questions:
  - **Why doesn't my meter read the appropriate values?** You must connect Analog Output CH 0 to Analog Input CH 1 on the DAQ Signal Accessory. Also, you must change the scale on the meter from 0-1 to 0-10 and set the channels to write at channel 0 and read at channel 1.
  - **Why does my meter go back to zero after stepping through the voltages?** Because a 0.0 is written to the Voltage Output indicator after the loop completes.
  - **What is a Local Variable and why is it used here?** A local variable is an additional terminal for a front panel object. Local variables are covered in detail in the LabVIEW Basics II Course. The local is used here to set the Voltage Output indicator back to zero after the loop is finished.

## Scanning Multiple Analog Input Channels



|            | CH1 | CH2 | CH4 | CH6 |
|------------|-----|-----|-----|-----|
| Scan No. 1 | —   | —   | —   | —   |
| Scan No. 2 | —   | —   | —   | —   |
| Scan No. 3 | —   | —   | —   | —   |

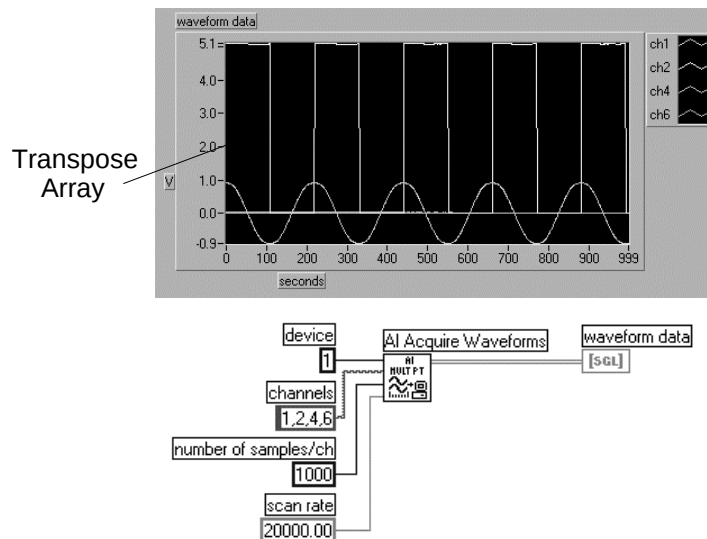
### Page 9-39:

- The last Easy I/O VIs we will discuss acquire multiple samples from multiple channels.
- The *Actual Scan Period* parameter represents time between two adjacent scans:

|            | CH1 | CH2 | CH4 | CH6 |
|------------|-----|-----|-----|-----|
| Scan No. 1 | —   | —   | —   | —   |
| Scan No. 2 | —   | —   | —   | —   |
| Scan No. 3 | —   | —   | —   | —   |

- Index Array VIs strip off columns of an array to retrieve data for individual channels. Col 0 contains data from the first channel scanned, Col 1 contains the second channel, Col 2 contains the third channel, etc. Thus, for channels 0, 2, and 4, you access columns 0, 1, 2, not columns 0, 2, 4.

## Scanned Waveforms and Graphs



### Page 9-40:

- You only need to use the Index Array functions if you want to separate the different channels into single channel arrays. If you want to analyze or display all the channels at once, you can just leave the data as a 2D array and wire it straight to a graph.
- Remember to Transpose array when displaying data on graph because graphs plot each row separately and each channel is in a column when returned from the DAQ VI.
- You can Transpose the array in two ways:
  - Pop up on the graph and select **Transpose Array** from the menu.
  - Use the **Transpose 2D Array** function (**Array** subpalette) between the DAQ VI and the graph on the diagram.

## Exercise 9-8 on page 9-42

Students examine and run Scan Example.vi

Time to complete: 15 min.

### Page 9-42:

- Common questions:
  - **How is the data from each channel being extracted from the array?** Using the technique shown on page 8-39, the row indexing is disabled so that an entire column is extracted at a time. Each column is then plotted on a separate graph.

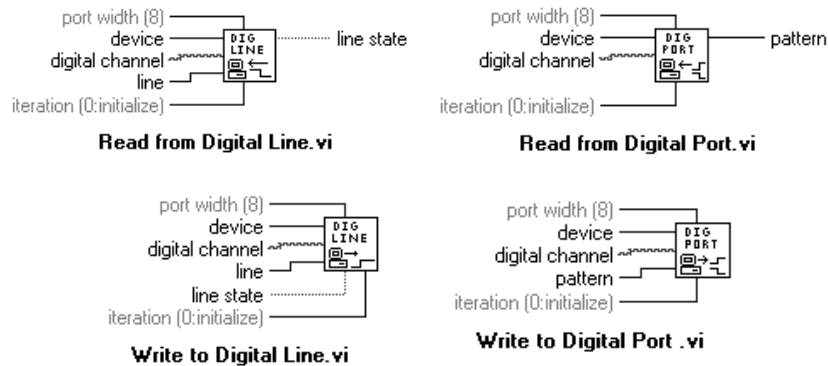
**Exercise 9-9 on page 9-43  
(Optional)**

Students build **Scan Two Waveforms.vi**  
Time to complete: 20-25 min.

**Page 9-43:**

- Hints:
  - This VI should look like the diagram shown on the bottom of page 9-40.

## Digital Input and Output VIs



Pages 9-44 to 9-45:

- Digital I/O VIs are used to read or write digital TTL signals.
  - **Write to Digital Line:** writes a binary value (Boolean) to a single specified digital line.
  - **Write to Digital Port:** writes a binary pattern to an entire digital port.
  - **Read from Digital Line:** reads a binary value (Boolean) from a single digital line.
  - **Read from Digital Port:** reads a binary pattern from a specified digital port.
- Discuss the functions shown on this slide and their inputs/outputs in detail.
- Describe the pattern value (EX: pattern = 8 = 1000 binary; Line 3 is high while Lines 0, 1, and 2 are low.)

### Exercise 9-10 on page 9-46

Students examine and run Digital I/O Example.vi  
Time to complete: 20 min.

#### Page 9-45:

- Common questions:
  - **Why aren't the Binary Pattern and the LED pattern the same?** The least significant bit of the Binary Pattern is on the right. However, the LED Pattern is an array of Booleans. Its "least significant bit" is array(0). Thus, for both patterns to look identical, you need to invert the array using the **Reverse 1D Array** function.
  - **Why do I need to subtract the pattern input from 15?** To account for the negative logic of the DAQ Signal Accessory LEDs.

## Buffered Data Acquisition (Optional)



- Intermediate VIs
  - Highly recommended for most applications
  - Continuous acquisition with triggering available
  - Interval scanning (except on NB boards)
  - Different gains for different channels
  - Streaming data to disk
  - Error handling

### Page 9-46:

- The Intermediate DAQ VIs are covered in detail in the LabVIEW Data Acquisition Course. However, if the students are doing a lot of DAQ or are ahead of schedule, you may want to cover the information covered in this and the next 3 slides.
- The Intermediate DAQ VIs allow you to use more features of the DAQ board such as continuous acquisition, triggering, interval scanning, different gains per channel, and custom error handling.

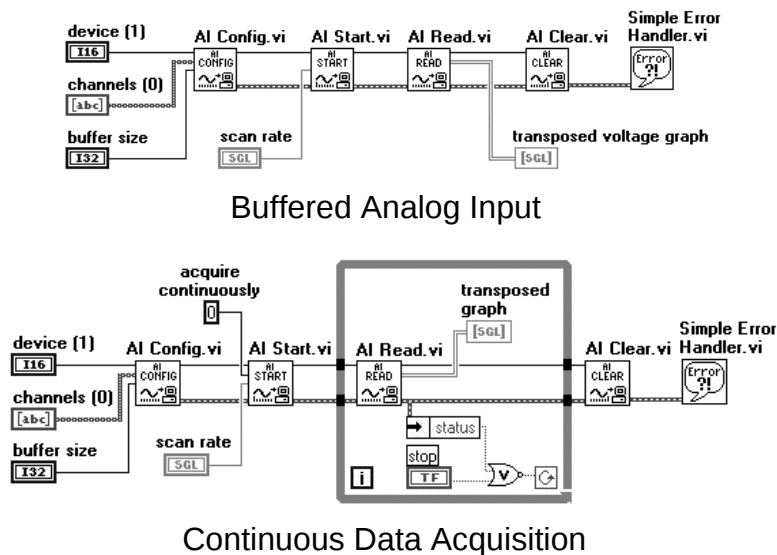
## Analog Input Operations

- Sequence
  - AI Config** -- Configure channels, gains, and computer buffer
  - AI Start** -- Set trigger conditions, program clocks, and start DAQ
  - AI Read** -- Read data from buffer (repeat if continuous)
  - AI Clear** -- Stop DAQ, free board and computer resources
- General VI parameters
  - taskID** - to control execution order
  - Bold** terminals
  - Default values

Pages 9-47 to 9-48:

- Discuss the standard sequence of the analog input operations. Explain how each sequence step translates to an intermediate Analog Input VI.
- Each of these VIs has several inputs and outputs. Show the help window information for each VI and discuss the important connections they will need to make such as **taskID**, required inputs, and **error in/error out**.

## Analog Input Intermediate VIs



Pages 9-49 to 9-52:

Explain the nonbuffered application algorithm. Now explain the buffered application algorithm. Discuss the following points:

- Acquiring the data and reading the data from the DAQ board's buffer are two separate operations.
- Default values are used to optimize the application. Because the start and read steps use the default value of -1 for **number of scans to acquire** and **number of scans to read**, respectively, the **buffer size** sets the number of scans for both steps.
- After acquiring **buffer size** scans, the acquisition is complete. The clear step frees the buffer memory and other resources used.
- The **taskID** "chains" the different VIs together to control the order of execution.
- Error information is passed from one VI to the next through the **error in** and **error out** clusters. The **Simple Error Handler** VI at the end of the sequence reports information on the *first* error that occurred, if any.

Now explain how the continuous buffered acquisition works by simply setting the board to acquire continuously and then reading the buffer in a While Loop.

### **Exercise 9-11 on page 9-53 (Optional)**

Students build Continuous Acquire with MIO.vi

Time to complete: 20-25 min.

#### **Pages 9-53 to 9-55:**

- This exercise is optional, so only have the students work on it if they are ahead of schedule. If they are interested in this topic and there isn't enough time for them to do this exercise, then open the solution and describe how it works.
- Describe the following parameters:
  - **Scan backlog** -- is the number of scans left in the buffer that have not been read yet. If the scan backlog becomes the same size as the buffer length, then you will get an overwrite error.
  - **Read/search position** -- is where in the DAQ buffer you take the readings. You can specify from the beginning of the buffer, where the current read pointer is, or from the end of the buffer.

## Summary

- NI-DAQ Configuration Utility to configure DAQ boards
- DAQ VIs organized into five subpalettes – Analog Input, Analog Output, Digital I/O, Counter, Configuration and Calibration, and Signal Conditioning
- Analog Input and Output subpalette divided into three levels – Easy I/O, Intermediate, Advanced, and Utility VIs
- Easy I/O contains VIs for
  - Single-channel analog input and output
  - Single-channel waveform input and output
  - Multichannel waveform input and output
  - Digital input and output
- Continuous DAQ can be performed using the intermediate VIs of AI Config, AI Start, AI Read, and AI Clear.

## Page 9-56:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - How are DAQ Boards configured?
  - How can you test the configuration of your board?
  - What are the three types of DAQ VIs?
  - Which VIs are used for...
    - One-point I/O on one channel?
    - One-point I/O on multiple channels?
    - Multiple-point I/O on one channel?
    - Multiple-point I/O on multiple channels?
  - What is the difference between **Write to Digital Line** and **Write to Digital Port**?
  - What are the DAQ Wizards and why would you use them?
- Review the slide: The purpose of Lesson Nine was to introduce the LabVIEW DAQ library and supporting functions.

### **Additional Exercises on page 9-57**

**9-12 -- Students build Temp Monitor with LED.vi**

Time to complete: 25 min.

**9-13 -- Students use the DAQ Solution Wizard to make Simple Data Reader.vi**

Time to complete: 25 min.

### **Page 9-57:**

- 9-12 Common question:
  - **What pattern do I need to write to Port 0 to turn on LED 0?** Because the LED is the least significant line, you need to write a 1 to it. However, because the Demo Box uses negative logic, you need to write a  $255-1=254$  to Port 0 to turn on LED 0 and a  $255-0=255$  to LED 0 to turn it off.

## **Lesson 10**

### **Instrument Control**

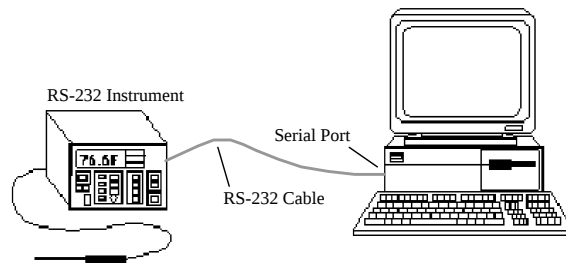
#### **You Will Learn:**

- A. About serial port communication
- B. About GPIB instrument control
- C. About VISA functions for instrument control
- D. About LabVIEW instrument drivers
- E. About waveform transfers

#### **Page 10-1:**

- This chapter introduces Serial and GPIB instrument control using LabVIEW.
- This lesson covers:
  - Serial communication background.
  - Serial VIs in LabVIEW.
  - IEEE 488 background.
  - IEEE 488 control from LabVIEW.
  - Instrument drivers in LabVIEW.
  - ASCII and binary waveform transfer.

## Serial Communication



### Terminology

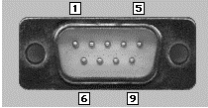
- Baud rate -- bits per second
- Data bits -- inverted logic and LSB first
- Parity -- optional error-checking bit
- Stop bits -- 1, 1.5, or 2 inverted bits at data end
- Flow control -- hardware and software handshaking options

### Pages 10-2 to 10-3:

- **Instructor:** See how many students are going to use or are interested in Serial I/O. Then cover the material in this and the next 4 slides accordingly.
- Go over the following Serial I/O terminology:
  - **Baud rate**—How many bits per second are transferred on the serial cable.
  - **Data bits**—How many bits represent a data value.
  - **Parity**—Optional error checking bit that is added to the data.
  - **Stop bits**—Certain number of bits added to the end of each data transfer.
  - **Flow control**—Optional hardware or software handshaking parameters for communicating with a device.

## Serial Hardware Connection

- RS-232
  - Most PCs
  - DCE or DTE configurations
  - 9-pin or 25-pin

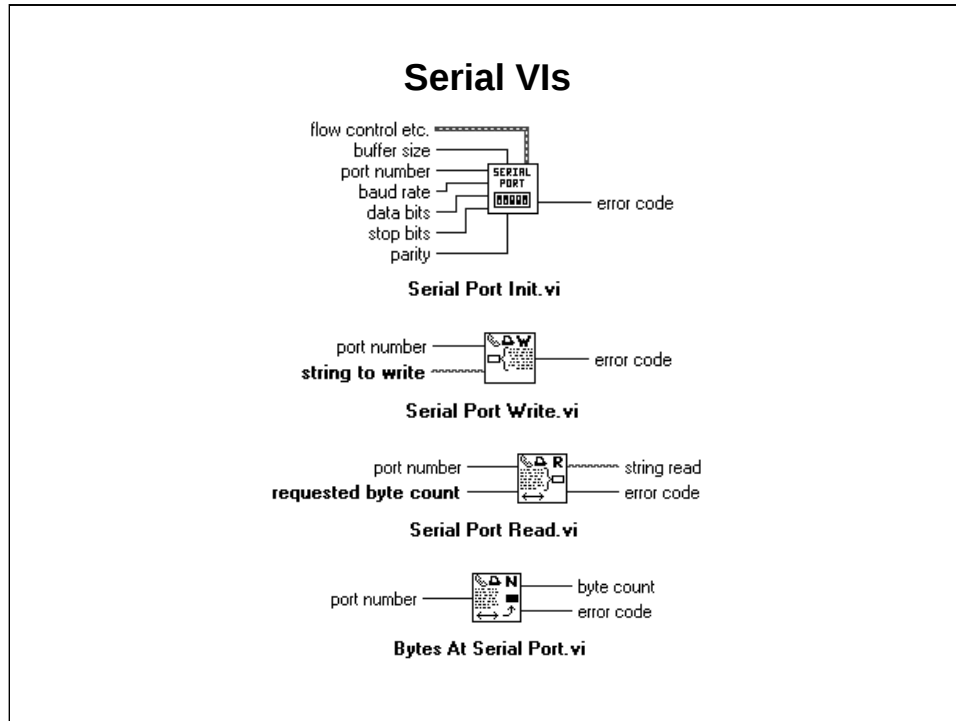


| Pin |     | DTE   | DCE    |
|-----|-----|-------|--------|
| 1   | DCD | Input | Output |
| 2   | RxD | I     | O      |
| 3   | TxD | O     | I      |
| 4   | DTR | O     | I      |
| 5   | Com | -     | -      |
| 6   | DSR | I     | O      |
| 7   | RTS | O     | I      |
| 8   | CTS | I     | O      |
| 9   | RI  | I     | O      |

- RS-422
  - Macintosh
  - 8-pin
- RS-485

### Pages 10-4 to 10-6:

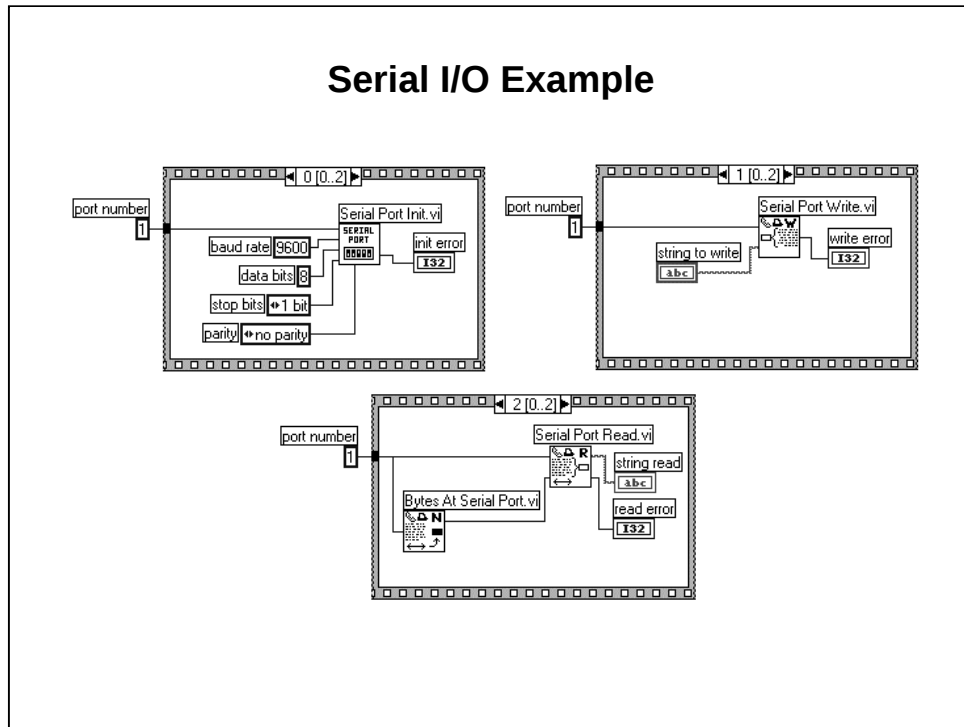
- There are several recommended standards (RS) for serial communication. Each varies in hardware and software specifications. The students must be familiar with their instrument and what connector is used before you can begin controlling that device with their computer.
- There are three main Serial I/O types:
  - **RS-232**—Connector found on most PCs. This is a single-ended communication method where only one device can be connected per port. Two connector types, 9- or 25-pin. Two configurations, DCE or DTE.
  - **RS-422**—Connector found on most Macs. This is a differential communication method. Connector has 8 pins.
  - **RS-485**—Differential and multi-drop serial communication method used mainly in industrial automation. Not covered in this course.



Pages 10-6 to 10-7:

- **Instructor:** Have students examine the VI terminals with the Help window, not with the **Online Reference**.
- The Serial VI library is very simple. These VIs call the standard serial driver that is part of the operating system.
- Serial protocol is bit serial (one bit at a time); GPIB is byte serial (one byte at a time).
- Serial VIs:
  - **Serial Port Initialization**—Used to configure the port number, baud rate, data bits, handshaking, and so on.
  - **Serial Port Write**—Writes an ASCII string to the specified serial port.
  - **Serial Port Read**—Reads the bytes available at the specified port.
  - **Bytes at Serial Port**—Returns the total number of bytes available at the port to be read.

## Serial I/O Example



Pages 10-7 to 10-8:

- Explain the diagram:
  - You must put the functions in a Sequence structure so that initialization occurs before the write, and so on.
  - Notice that an ASCII string must be passed to the **Write** function (possibly a command to the instrument).
  - The **Bytes at Serial Port** VI determines the total number of bytes that should be read in with **Read** function.
  - The **Serial Port Read** function reads the available bytes and returns it to a string.
- **Instructor:** Demonstrate serial communication by using the null modem cable. Connect two computers together and make note of the port of each (most likely port 0). Open the **Serial Example** VI on both computers. Enable the **Read** Boolean on one computer and **Write** on another computer.

### **Exercise 10-1 on page 10-11**

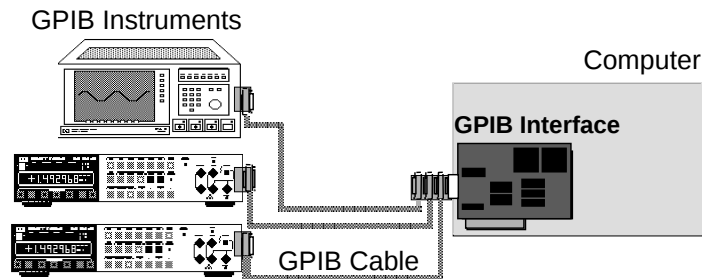
Students examine and run **Serial Write & Read.vi**  
Time to complete: 15-20 min.

#### **Pages 10-11 to 10-13:**

- The Instrument Simulator can communicate in either GPIB or Serial mode depending upon the DIP switch settings. Show the students how to configure the Instrument Simulator for serial communication.
- Make sure the Instrument Simulator is OFF when changing the switch settings.
- **Instructor:** This is a new exercise, so please record common questions and possible answers to the CAR database.

## GPIB Overview

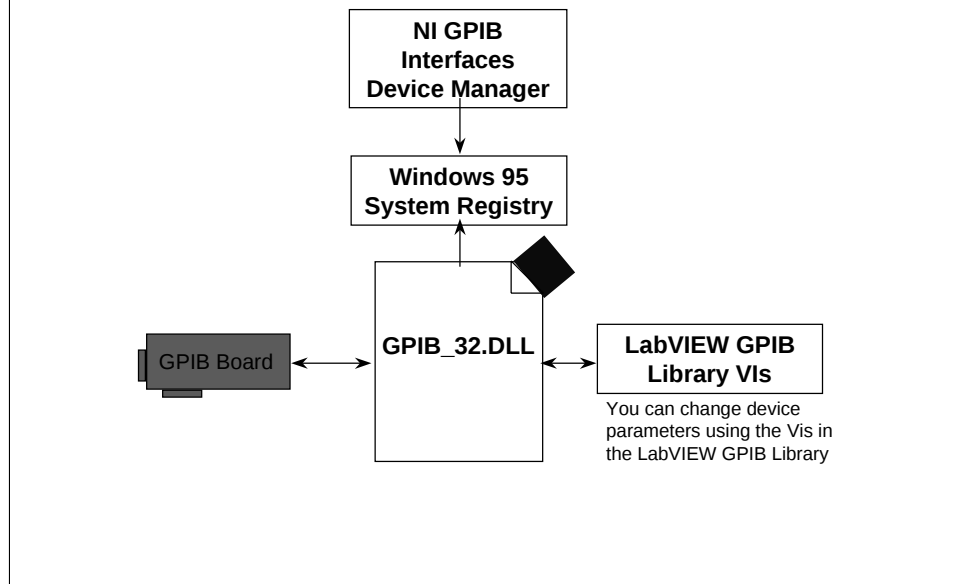
- GPIB library supports all GPIB boards
- LabVIEW uses the NI-488.2 driver-level software
- Traditional GPIB VIs for compatibility
- 488.2 VIs



### Pages 10-14 to 10-15:

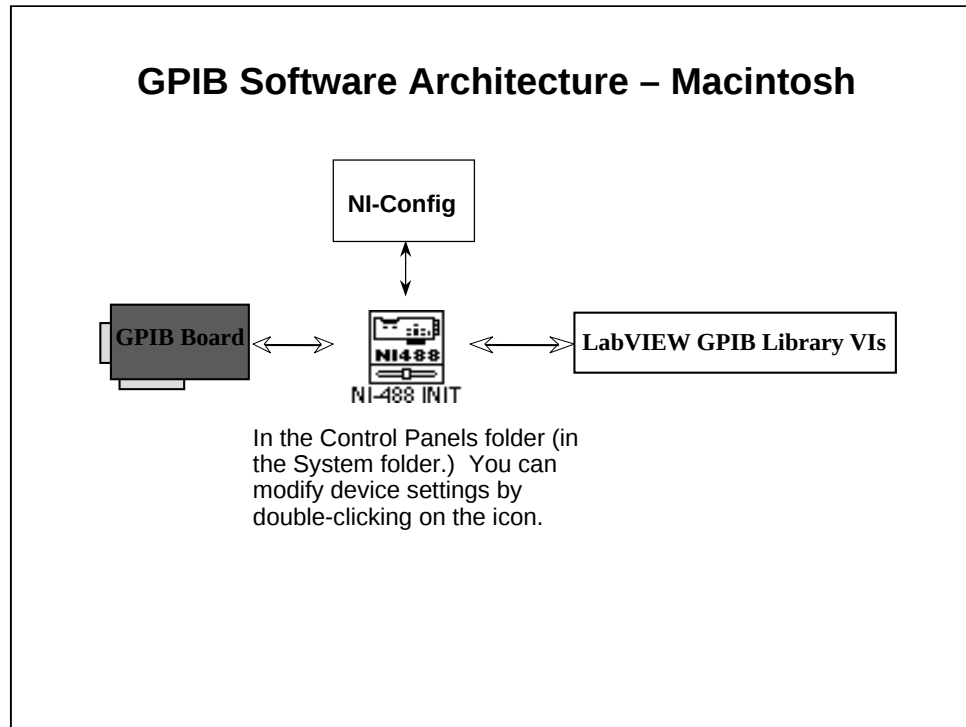
- The LabVIEW GPIB library contains VIs that control National Instruments GPIB interface boards.
- The LabVIEW GPIB VIs use NI-488.2 driver-level software:
  - Old boards (non 488.2) must be upgraded.
  - The library contains 488.2 and traditional VIs. We will focus primarily on the traditional VIs.
- Briefly describe a typical system containing several GPIB devices connected to a computer. Describe that one device is the System Controller and the others are Talkers/Listeners. Each device on the GPIB has a unique address.

## GPIB Software Architecture – Windows



Pages 10-16 to 10-20:

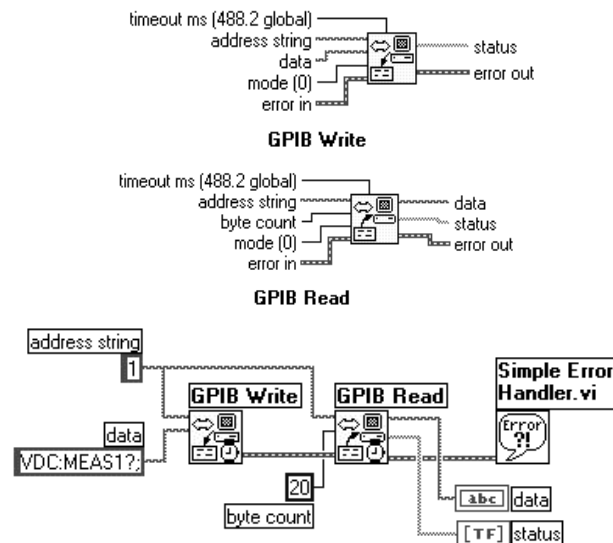
- LabVIEW uses the standard NI-488.2 for Windows 95 32-bit GPIB DLL.
- GPIB parameters can be specified using:
  - Use the System Device Manager as the utility for configuring GPIB. Edits Windows System Registry (which specifies GPIB\_32.DLL parameters).
  - I/O terminals on the LabVIEW GPIB VIs.
  - **Note:** Instrument driver VIs modify GPIB parameters programmatically (there is no need to make any changes using Device Properties).
- Demonstrate the GPIB configuration utilities found in Windows 95 System Properties.
- Refer in-depth questions to technical support or suggest that the students take the GPIB course.



## Page 10-19:

- LabVIEW for Macintosh uses the standard NI-488.2 driver-level software.
- GPIB parameters can be specified using:
  - NI-488 Init: Shows the boards installed in the machine.
  - NI-488 Config: Utility for configuring GPIB.
  - I/O terminals on the LabVIEW GPIB VIs.
  - **Note:** Instrument driver VIs modify GPIB parameters programmatically.
- Refer in-depth questions to technical support or suggest that the students take the GPIB course.

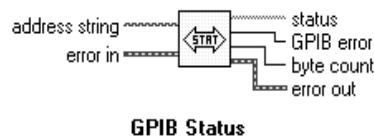
## Traditional GPIB Functions



Pages 10-20 to 10-21:

- LabVIEW has several VIs for GPIB communication. Most GPIB programs involve only writing and reading data found in **Functions> Instrument I/O> GPIB**.
- **GPIB Write:**
  - **GPIB address string**—Not numeric because secondary address might be needed.
  - **data string**—ASCII string to be sent.
  - **error out**—Cluster of 3 fields. Status field is a Boolean and is TRUE when an error occurs. The code field will be a GPIB error code value if the error occurs during a GPIB function. The source field is a string which describes where the error has occurred.
- **GPIB Read:**
  - **GPIB address string**—Same as for **GPIB Write**.
  - **Byte count**—Total number of bytes to be read.
  - Output **data string**—Data read from the instrument.
  - **status** —Same as for **GPIB Write**.

## Traditional GPIB Functions



### Status Bit Array Description

| Bit | Description            |
|-----|------------------------|
| 0   | Device clear state     |
| 1   | Device trigger state   |
| 2   | Listener active        |
| 3   | Talker active          |
| 4   | Attention asserted     |
| 5   | Controller-in-Charge   |
| 6   | Remote state           |
| 7   | Lockout state          |
| 8   | Operation completed    |
| 12  | SRQ detected while CIC |
| 13  | EOI or EOS detected    |
| 14  | Timeout                |
| 15  | Error                  |

Pages 10-21 to 10-22:

- **GPIB Status** VI shows the GPIB Controller status that **address string** indicates after the last operation.
- **GPIB Status** VI returns the number of bytes transferred in the last GPIB operation in the byte count parameter.
- The **GPIB error** parameter contains an error code if the VI detects an error. GPIB error is valid only if element 15 of the status Boolean array is TRUE.
- The **Status** output is an array of 16 Booleans that indicate the state of the GPIB after the last GPIB function call.

## Exercise 10-2 on page 10-23

Students build GPIB Write & Read.vi

Time to complete: 20-25 min.

### Pages 10-23 to 10-25:

- Before students begin this exercise, review the following points.
  - They should reconfigure the Instrument Simulator to communicate in GPIB mode by changing the switch settings on the side of the Simulator.
  - They should turn the Instrument Simulator OFF before they change the switches.
  - The GPIB box accepts ASCII commands and returns binary waveforms
  - They should first write to the GPIB box and then read from it.
  - They should type \*IDN? in the String to Write control.
- **Instructor:** This is a new exercise, so please record common questions and possible answers to the CAR database.

## VISA Overview

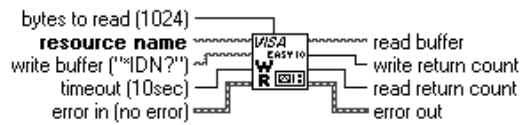
- Virtual Instrument Software Architecture
- Interface to Serial, GPIB, and VXI instruments
- Uses VISA.DLL
- **Resource Name** contains interface info

| Interface | Resource Name Grammar                                                                          |
|-----------|------------------------------------------------------------------------------------------------|
| Serial    | ASRL[board][::INSTR]                                                                           |
| GPIB      | GPIB[board]:: <i>primary address</i> ::INSTR]                                                  |
| VXI       | VXI[board]:: <i>VXI logical address</i> ::INSTR]                                               |
| GPIB-VXI  | GPIB-VXI[board][:: <i>GPIB-VXI primary address</i> ]<br>:: <i>VXI logical address</i> ::INSTR] |

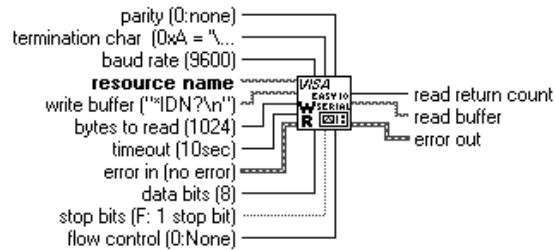
### Pages 10-26 to 10-27:

- **VISA:** Virtual Instrument Software Architecture. A protocol built upon 488.2 VIs to meet the industry needs for having a way to easily interface with multiple I/Os and have all manufacturers of instruments and instrument drivers follow a protocol.
- VISA created by the VXIplug&play Alliance which is composed of the top 35 instrument manufacturers such as HP. National Instruments is a leading member of the alliance.
- The **resource name** contains information on the type of I/O interface and the device address.

## VISA Functions



**Easy VISA Write & Read.vi**



**Easy VISA Serial Write & Read.vi**

Pages 10-29 to 10-30:

- The VISA subpalette contains the same hierarchy that you saw with the File I/O and DAQ VIs. The following are the high-level VISA VIs:
  - **Easy VISA Write & Read:** sends a command string out to the device specified by resource name and then reads the response back from the instrument.
  - **Easy VISA Serial Write & Read:** configures a serial connection and parameters, sends a data string to the serial device, and reads the response from the instrument.
- Open the diagrams of the above high-level VIs and discuss the intermediate VISA functions and their parameters. Show that the diagrams for the two high-level VIs are similar except for the serial configuration with the **Property Node** function.

### **Exercise 10-3 on page 10-31**

Students build VISA Write & Read.vi

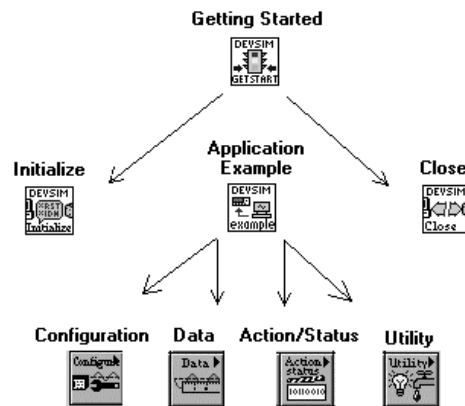
Time to complete: 20-25 min.

#### **Pages 10-31 to 10-32:**

- Before students begin this exercise, review the following points.
  - The Instrument Simulator will remain in GPIB communication mode. They should specify the resource name as GPIB::2::INSTR.
  - The Simulator accepts ASCII commands and returns binary waveforms
  - They should first write to the GPIB box and then read from it.
  - They should type \*IDN? in the String to Write control.
- **Instructor:** This is a new exercise, so please record common questions and possible answers to the CAR database.

## Instrument Drivers

- Over 600 instruments supported in the instrument library



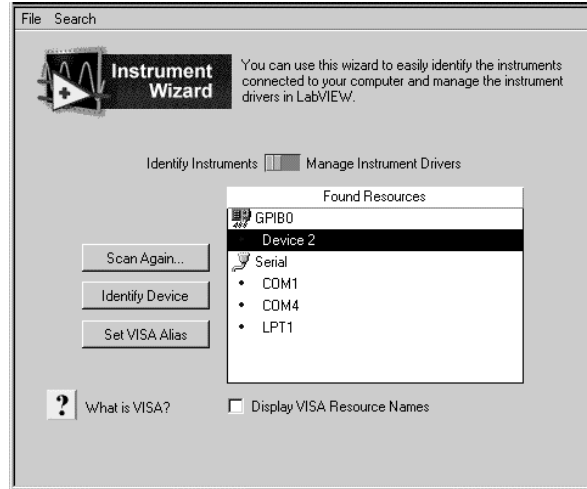
Instrument Driver VI Tree

## Pages 10-33 to 10-34:

- An instrument driver is a software module that allows remote control of an instrument (GPIB and serial).
- It is easy to modify a GPIB instrument driver for a serial instrument with similar command set and the VISA VIs.
- LabVIEW is ideally suited for creating instrument drivers:
  - The front panel simulates the operation of an instrument front panel.
  - The block diagram sends necessary commands to control an instrument.
- You no longer need to remember the instrument's command set. You need only to specify inputs on the front panel.
- More than 600 instrument drivers are available.
- All instrument drivers in the library have the same VI Tree structure. Therefore, once you learn to use one instrument driver, all others have the same basic hierarchy.

## The Instrument Wizard

- Automatically identifies connected devices
- Installs opens selected instrument drivers



### Page 10-34:

- The Instrument Wizard is a utility that will:
  - Automatically detect connected Serial, GPIB, VXI, or computer-based instruments.
  - Install and open selected instrument drivers.
- **Instructor:** Open the Instrument Wizard and go through the different parts.

### **Exercise 10-4 on page 10-35**

Students use the Instrument Wizard and examine  
the Device Simulator Instrument Driver

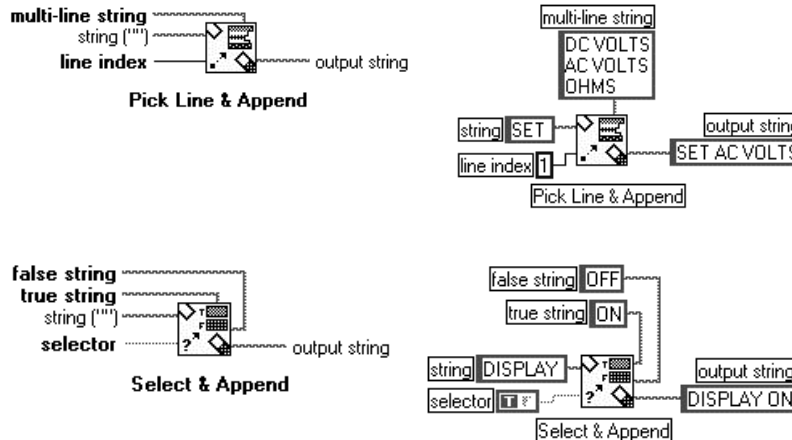
Time to complete: 15-20 min.

Pages 10-35 to 10-38:

- **Instructors:** Please write down any common questions the students may have and enter them in the CAR database under Customer Education.

## Instrument Driver Functions

- String subpalette - building, parsing, modifying strings



Page 10-39 :

- **Note:** Have students examine string function terminals with the **Online Reference**.
- Discuss all string functions thoroughly:
  - **Pick Line and Append** chooses a line from **multi-line string** and appends that line to **string**. Line index selects the line.
  - **Select and Append** chooses a string according to a Boolean **selector** and appends that string to **string**.

**Reminder:** Use **Scan From String** and **Format Into String** for formatting purposes.

## Exercise 10-5 on page 10-40

Students build Build Command String.vi

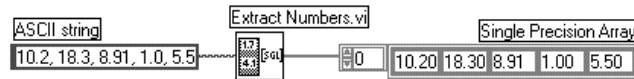
Time to complete: 20 min.

### Pages 10-40 to 10-41:

- Common questions:
  - **How do I add text labels to the vertical slide?** Pop up on the slide and choose **Text Labels**, then pop up on the Text Display to add items.
- When students finish, stress that these functions are important for writing instrument drivers.

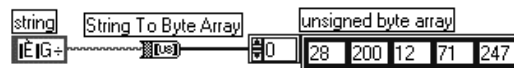
## Waveform Transfers

- ASCII Waveforms

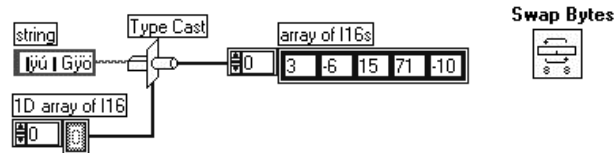


- Binary Waveforms

- 1-byte integers



- 2-byte integers



## Pages 10-42 to 10-44:

- Most instruments return a waveform as either an ASCII string or a binary string. A binary transfer is faster and requires less memory than an ASCII string transfer.
- ASCII Waveforms:
  - With 1,024 points, each between 0 and 255, you need a maximum of 4 bytes per point, or 4,096 bytes, plus any header and trailer information.
  - Use the **Extract Numbers** VI to convert to an array.
- Binary waveforms encoded as 1-byte integers:
  - Use the **String to Byte Array** function to convert to an array.
- Binary waveforms encoded as 2-byte integers:
  - The same waveform requires 2,048 bytes, plus header/ trailer information.
  - Use the **Typecast** function to convert to an I16 array and then **Swap Bytes**. The LabVIEW Advanced course covers typecasting in more detail.

## Exercise 10-6 on page 10-45 (Optional)

Students examine and run Waveform Example.vi

Time to complete: 20 min.

### Pages 10-45 to 10-47:

- Common questions:
  - **When extracting a binary waveform string, why are the String Subset, String to Byte Array, and Subtract functions used?** The **String Subset** function extracts the header information so that only the data remains. **String to Byte Array** converts the string to an array of unsigned integers. The VI subtracts 127 from each element of the array to correctly convert the data points as the oscilloscope manual specifies.

**Exercise 10-7 on page 10-48  
(Optional)**

Students build Binary Waveform.vi  
Time to complete: 20-25 min.

**Page 10-48:**

- If time permits, discuss the diagram on your computer. Otherwise, you can skip this exercise because it does not cover any new material.

## Summary

- Serial library contains functions for serial communication
- GPIB library contains functions for GPIB boards
- Common GPIB functions
  - GPIB Status
  - GPIB Write
  - GPIB Read
- VISA a standard protocol for using multiple types of I/O and instrument driver development
- Instrument Library – more than 600 instruments supported
- The Instrument Wizard detects connected devices and installs appropriate instrument drivers
- String functions ideal for creating instrument drivers
- You need to know the format of the returned data string in order to convert it to the correct numbers.

## Page 10-49:

- Do not immediately display this slide.
- Suggested questions for class participation:
  - What GPIB boards does LabVIEW support?
  - What is the difference between GPIB and serial?
  - What type of GPIB instruments will LabVIEW support? (488.2 and 488 instruments)
  - What is VISA and why use it?
  - What is the purpose of an instrument driver?
  - What is the Instrument Wizard and how do you use it?
- Review the slide: The purpose of Lesson Ten was to introduce the LabVIEW serial library and the LabVIEW GPIB library and supporting functions.

### **Additional Exercises on page 10-50**

10-8 -- Students build Simulator Voltmeter.vi

Time to complete: 20-25 min.

10-9 -- Students modify a previous VI and save it as  
VISA Write & Read 2.vi

Time to complete: 20-25 min.

10-10 -- Students use the Instrument Wizard to  
communicate in serial mode with a device

Time to complete: 20-25 min.

### **Page 10-50:**

- **Instructors:** Please write down any common questions the students may have and enter them in the CAR database under Customer Education.

## **LabVIEW Add-On Toolkits**

Enhance your development flexibility and productivity with LabVIEW Add-On Toolkits!

- Application Builder
- Test Executive
- SQL
- Statistical Process Control (SPC)
- Professional G Developers
- Unit Test and Validation Procedures
- Automation Symbols
- Picture Control
- Internet Developers

- Describe each toolkit. Elaborate if there is interest!

### **LabVIEW Add-On Toolkits**

- Joint Time-Frequency Analysis
- Wavelet and Filter Bank Design
- Digital Filter Design
- Third-Octave Analysis
- G Math
- Image Acquisition (IMAQ) Vision
- PID Control Toolkit
- Fuzzy Logic Control Toolkit
- Control and Simulation Toolkit
- Motion Control Toolkit

- Describe each toolkit. Elaborate if there is interest!

## **Additional Information**

Application Notes

Technical Notes

Instrument Driver Library

Course Evaluation

### **Conclusion:**

- Have the students fill out their Customer Education Student Profile forms to get Application Notes and demos.
- Tell the students how to access Application and Technical Notes and how to get to the instrument driver library on the NI Web page.
- Explain that instrument driver and technical notes order forms will be processed and the students' requests will be sent to them in a few weeks.