➢ LabVIEW

➢ How to call WIN32 Applications

© by
Ralf Engels, Heinz Rongen
Forschungszentrum Jülich GmbH
Zentrallabor für Elektronik
D-52425 Jülich

Ralf Engels | **Forschungszentrum Jülich, ZEL** | **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 | Fax: +49-(0)2461-61399 | Email: R.Engels@FZ-Juelich.de

LabVIEW

Ralf Engels
Forschungszentrum Jülich GmbH
Zentrallabor für Elektronik
D-52425 Jülich

Contents

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1 INTRODUCTION TO LABVIEW

1.1 Chapter Information

Each chapter begins with a section like the one that follows, listing the learning objectives for that chapter.

You Will Learn:

- What LabVIEW is.
- What a Virtual Instrument (VI) is.
- How to use the LabVIEW environment (windows and palettes).
- How to operate VIs.
- How to edit VIs.
- How to create VIs.

1.2 What Is LabVIEW?

LabVIEW is a program development application, much like various commercial C or BASIC development systems, or National Instruments LabWindows. However, LabVIEW is different from those applications in one important respect. Other programming systems use *text-based* languages to create lines of code, while LabVIEW uses a *graphical* programming language, G, to create programs in block diagram form.

You can use LabVIEW with little programming experience. LabVIEW uses terminology, icons, and ideas familiar to scientists and engineers and relies on graphical symbols rather than textual language to describe programming actions.

LabVIEW has extensive libraries of functions and subroutines for most programming tasks. For Windows, Macintosh, and Sun, LabVIEW contains application specific libraries for data acquisition and VXI instrument control. LabVIEW also contains application-specific libraries for GPIB and serial instrument control, data analysis, data presentation, and data storage. LabVIEW includes conventional program development tools, so you can set breakpoints, animate program execution to see how data passes through the program, and single-step through the program to make debugging and program development easier.

1.3 How Does LabVIEW Work?

LabVIEW includes libraries of functions and development tools designed specifically for instrument control. For Windows, Macintosh, and Sun, LabVIEW also contains libraries of functions and development tools for data acquisition. LabVIEW programs are called *virtual instruments (VI*s) because their appearance and operation imitate actual instruments. However, they are analogous to functions from conventional language programs. VIs have both an interactive user interface and a source code equivalent, and accept parameters from higher-level VIs. The following are descriptions of these three VI features.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

- VIs contain an interactive user interface, which is called the *front pane*l, because it simulates the panel of a physical instrument. The front panel can contain knobs, push buttons, graphs, and other controls and indicators. You input data using a keyboard and mouse, and then view the results on the computer screen.
- VIs receive instructions from a *block diagra*m, which you construct in G. The block diagram supplies a pictorial solution to a programming problem. The block diagram contains the source code for the VI.
- VIs use a hierarchical and modular structure. You can use them as top-level programs, or as subprograms within other programs or subprograms. A VI within another VI is called a *subV*I. The *icon and connector pane* of a VI work like a graphical parameter list so that other VIs can pass data to it as a subVI.

With these features, LabVIEW promotes and adheres to the concept of *modular programmin*g. You divide an application into a series of tasks, which you can divide again until a complicated application becomes a series of simple subtasks. You build a VI to accomplish each subtask and then combine those VIs on another block diagram to accomplish the larger task. Finally, your top-level VI contains a collection of subVIs that represent application functions.

Because you can execute each subVI by itself, apart from the rest of the application, debugging is much easier. Furthermore, many low-level subVIs often perform tasks common to several applications, so that you can develop a specialized set of subVIs suited to applications you can construct.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1.4 Tools Palette

LabVIEW uses a floating **Tools** palette, which you can use to edit and debug VIs. You use the <Tab> key to tab through the commonly used tools on the palette. If you have closed the **Tools** palette, select **Windows»Show Tools Palette** to display the palette.The following Illustration displays the **Tools** palette.

	**Operating tool**	Places **Controls** and **Functions** palette items on the front panel and block diagram
	**Positioning tool**	Positions, resizes, and selects objects
	**Labeling tool**	Edits text and creates free labels
	**Wiring tool**	Wires objects together in the block diagram
	**Object pop-up menu tool**	Brings up on a pop-up menu for an object
	**Scroll tool**	Scrolls through the window without using the scrollbars
	**Breakpoint tool**	Sets breakpoints on VIs, functions, loops, sequences, and cases
	**Probe tool**	Creates probes on wires.Chapter 1 Introduction to LabVIEW
	**Color copy tool**	Copies colors for pasting with the Color tool
	**Color tool**	Sets foreground and background colors

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

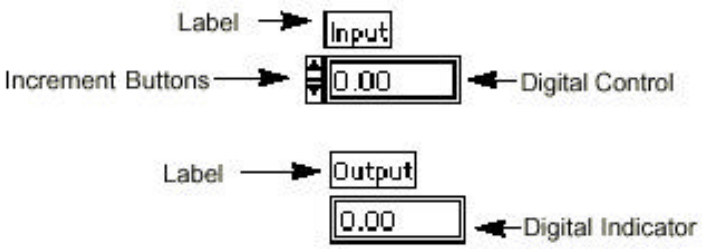
1.5 Controls Palette

The **Controls** palette consists of a graphical, floating palette that automatically opens when you launch LabVIEW. You use this palette to place controls and indicators on the front panel of a VI. Each top-level icon contains subpalettes. If the **Controls** palette is not visible, you can open the palette by selecting **Windows»Show Controls Palette** from the front panel menu. You can also pop up on an open area in the front panel to access a temporary copy of the **Controls** palette. The following illustration displays the top-level of the **Controls** palette.

1.6 Controls and Indicators

1.6.1 Numeric Controls and Indicators

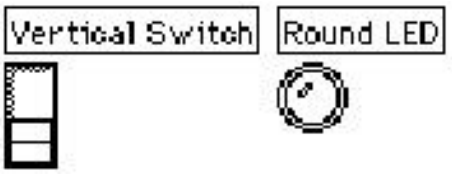
You use, numeric controls to enter numeric quantities, while numeric indicators display numeric quantities. The two most commonly used numeric objects are the *digital control* and the *digital indicator*.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

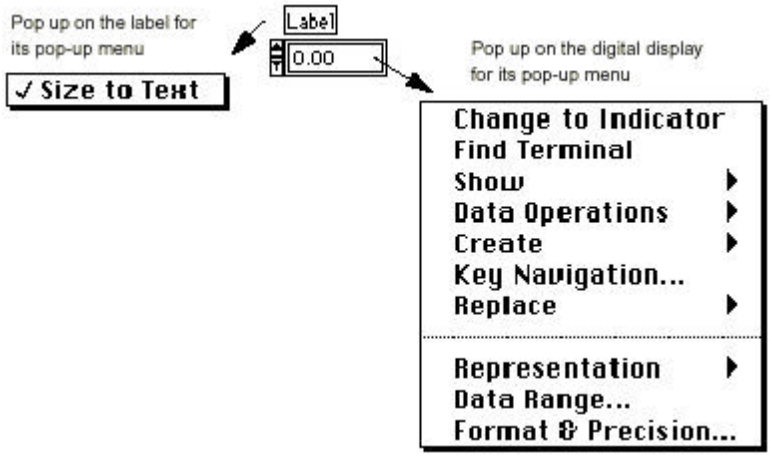
1.6.2 Boolean Controls and Indicators

You use Boolean controls and indicators for entering and displaying Boolean (True/False) values. Boolean objects simulate switches, buttons, and LEDs. The most commonly used Boolean objects are the *vertical switch* and the *round LED.*



1.6.3 Configuring Controls and Indicators

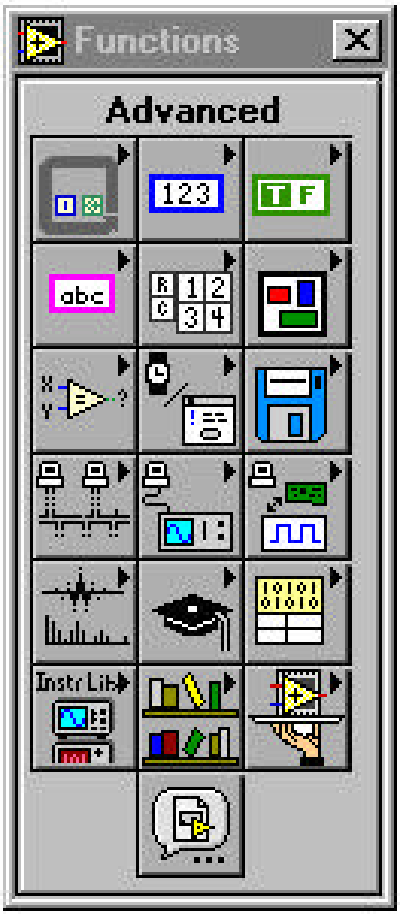
You can configure nearly all the controls and indicators using options from their pop-up menus. *Popping up on individual components of controls and indicators displays menus for customizing those component*s. An easy way to access the pop-up menu is to click the Object pop-up menu tool, shown at left, on any object that has a pop-up menu. The following picture illustrates this display method for a digital control.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1.7 Functions Palette

The **Functions** palette consists of a graphical, floating palette that automatically opens when you switch to the block diagram. You use this palette to place nodes (constants, indicators, VIs, and so on) on the block diagram of a VI. Each top-level icon contains subpalettes. If the **Functions** palette is not visible, you can select **Windows»Show Functions Palette** from the block diagram menu to display it. You can also pop up on an open area in the block diagram to access a temporary copy of the **Functions** palette. The following illustration displays the top-level of the **Functions** palette.



1.8 Building a VI

To build a VI that simulates acquisition of a temperature reading. ***Make sure you have clicked on the*** *Explore LabVIEW for your own applications* ***option in the LabVIEW Demo VI before you start this exercise.***

You will use the Demo Voltage Read VI to measure the voltage, and then multiply the reading by 100.0 to convert the voltage into a temperature (in degrees F).

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1.8.1 Front Panel

1. Open a new front panel by selecting **File»New** or choosing the **New VI** button in the dialog box. For Windows and UNIX, if you have closed all VIs, select **New VI** from the LabVIEW dialog box.

Note*: If the Controls palette is not visible, select Windows»Show Controls Palette to display the palette. You can also access the Controls palette by popping up in an open area of the front panel. To pop up, right-click on your mouse (<command>-click on Macintosh).*

2. Select a Thermometer indicator from **Controls»Numeri**c, and place it on the front panel by dragging the indicator on to the panel.
3. Type `Temp` inside the label text box and click on the enter button on the toolbar.

Note*: If you click outside the text box without entering text, the label disappears. You can show the label again by popping up on the control and selecting Show»Label.*

4. Rescale the thermometer control to display the temperature between 0.0 and 100.0.

- Using the Labeling tool, double-click on `10.0` in thermometer scale to highlight it.
- Type `100.0` in the scale and click the mouse button anywhere outside the display window. LabVIEW automatically scales the intermediary increments.The temperature control should now look like the following illustration.

1.8.2 Block Diagram

1. Open the block diagram by choosing **Windows»Show Diagra**m. Select the block diagram objects discussed below from the **Functions** palette. For each object that you want to insert, select the icon and then the object from the top-level of the palette, or choose the object from the appropriate subpalette. When you position the mouse on the block diagram, LabVIEW displays an outline of the object.

Note*: If the Functions palette is not visible, select Windows»Show Functions Palette to display the palette. You can also access the Functions palette by popping up in an open area of the block diagram.*

Place each of the following objects on the block diagram. The Demo Voltage Read VI **(Functions»Tutoria**l) simulates reading a voltage from a plug-in data acquisition board. Multiply function **(Functions»Numeri**c). In this exercise, the function multiplies the voltage returned by the Demo Voltage Read VI by 100.0. Numeric Constant **(Functions»Numeri**c). You need two numeric constants: one for the scaling factor of 100 and one for the device constant. For the first numeric constant, type 100.0 when the constant first appears on the block diagram.

2. Create the second numeric constant using a shortcut to automatically create and wire the constant to the Demo Voltage Read VI.
   a. Using the Wiring tool, pop up on the input marked `Board ID` on the Demo Voltage Read VI and select **Create Constant** from the pop-up menu. This option automatically creates a numeric constant and wires it to the Demo Voltage Read VI.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

b. Type `1` when the constant first appears on the block diagram. This changes the default value of zero to one.

Note*: You do not have to change to the Labeling tool to insert the value when using this feature, because the cursor is already in place.*

c. Pop up on the constant and choose **Show»Label**. Using the Labeling tool, change the default label (`Board ID`) to `Device`.

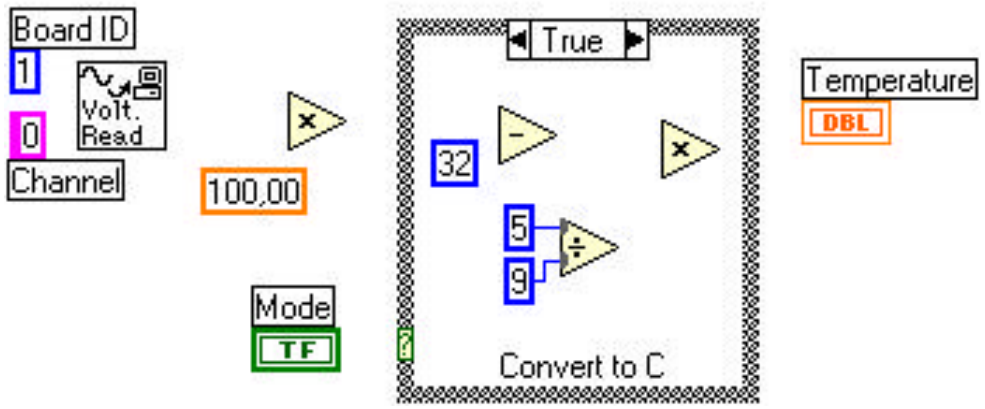
In this example, the two numerics represent the constant 100.0 and the device for the multiply function.

3. Place a String Constant **(Functions»String**) on your block diagram.
4. Using the Wiring tool, pop up on the input marked `Channel`, at the bottom left of the Demo Voltage Read VI and select **Create Constant** from the pop-up menu. This option automatically creates a string constant and wires it to the Demo Voltage Read VI.
5. Type `0` when the constant first appears on the block diagram. Pop up on the constant and choose **Show»Label**. Notice that in this instance, `Channel` appears in the default label so you do not have to change the label.

In this example, you use the string constant to represent the channel number.

Note*: You can create and wire controls, constants and indicators with most functions. If these options are not available for a particular function, the Create Control, Create Constant and Create Indicator options are disabled on the pop-up menu. For more information on this feature, see the Create & Wire Controls, Constants, and Indicators section later in this chapter.*

You should have pulled down all of the objects shown in the following illustration on to your block diagram.

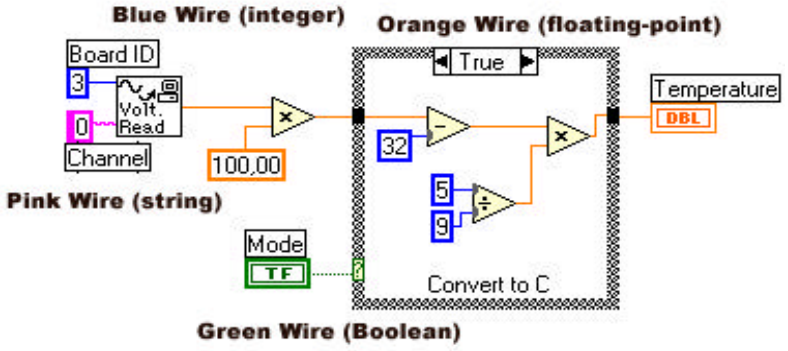


Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6. Using the Wiring tool, wire the remaining objects together as explained in the Wiring Techniques section later in this chapter.

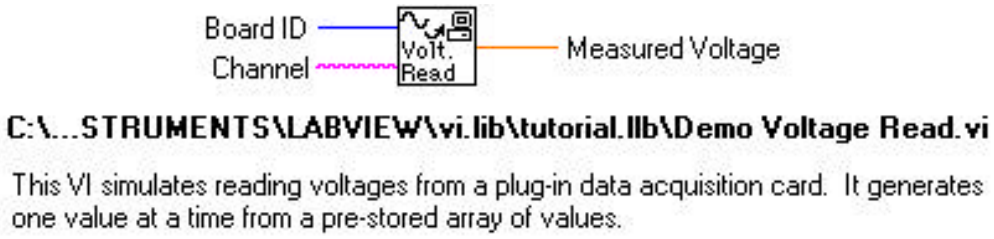
Note*: To move objects around on the block diagram, click on the Positioning tool in the Tools palette.*

LabVIEW color keys wires to the kind of data each wire carries. Blue wires carry integers, orange wires carry floating-point numbers, green wires carry Booleans, and pink wires carry strings.



You can activate the Help window by choosing **Help»Show Hel**p. Placing any of the editing tools on a node displays the inputs and outputs of that function in the Help window. As you pass an editing tool over the VI icon, LabVIEW highlights the wiring terminals in both the block diagram and the Help window. When you begin to wire your own diagrams, this flashing highlight can help you to connect your inputs and outputs to the proper terminals.

The Demo Voltage Read VI simulates reading the voltage at Channel 0 of a plug-in board providing artificial data to the **Measured Voltage** output. This data represents the real temperature divided by 100. The VI then multiplies the voltage by 100.0 to convert it to a temperature in °F.



1.8.3 Wiring Techniques

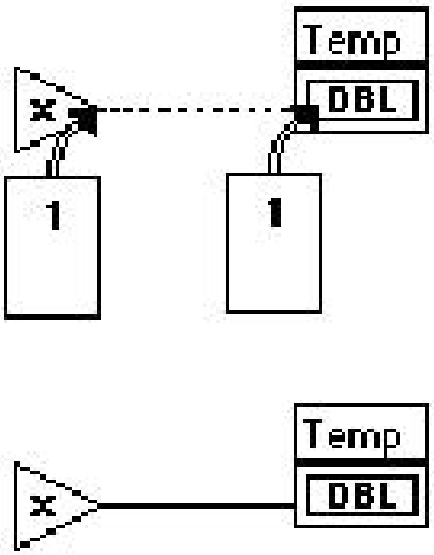
In the wiring illustrations in this section, the arrow at the end of this mouse symbol shows where to click and the number printed on the mouse button indicates how many times to click the mouse button.

The hot spot of the tool is the tip of the unwound wiring segment.

To wire from one terminal to another, click the Wiring tool on the first terminal, move the tool to the second terminal, and click on the second terminal. It does not matter at which terminal you start.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

When the Wiring tool is over a terminal, the terminal area blinks, to indicate that clicking connects the wire to that terminal. Do not hold down the mouse button while moving the Wiring tool from one terminal to another. You can bend a wire once by moving the mouse perpendicular to the current direction. To create more bends in the wire, click the mouse button. To change the direction of the wire, press the spacebar. Click with the mouse button, to tack the wire down and move the mouse perpendicularly.



1.8.4 Tip Strips

When you move the Wiring tool over the terminal of a node, a tip strip for that terminal pops up. Tip strips consist of small, yellow text banners that display the name of each terminal. These tip strips should help you to wire the terminals. The following illustration displays the tip strip (Measured Voltage) that appears when you place the Wiring tool over the output of the Demo Voltage Read VI.

Note: ***When you place the Wiring tool over a node, LabVIEW displays wire stubs that indicate each input and output. The wire stub has a dot at its end if it is an input to the node.***

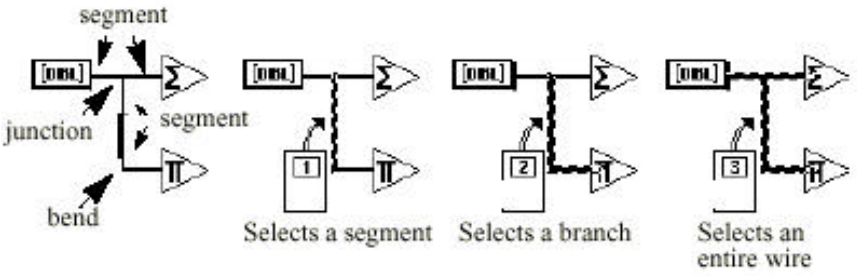
1.8.5 Showing Terminals

It is important that you wire the correct terminals of a function. You can show the icon connector to make correct wiring easier. To do this, pop up on the function and choose **Show»Terminal**s. To return to the icon, pop up on the function and again select **Show»Terminal**s.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

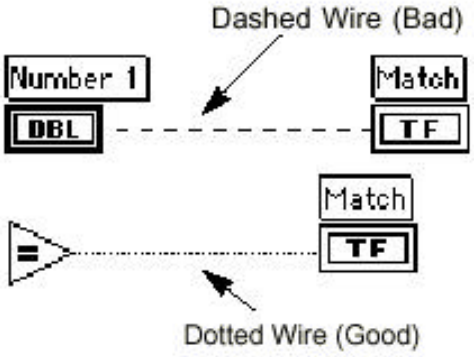
1.8.6 Selecting and Deleting Wires

You may accidentally wire nodes incorrectly. If you do, select the wire you want to delete and then press <Delete>. A wire segment is a single, horizontal or vertical piece of wire. The point where three or four wire segments join is called a junction. A wire branch contains all the wire segments from one junction to another, from a terminal to the next junction, or from one terminal to another if there are no junctions in between. You select a wire segment by clicking on it with the Positioning tool. Double-clicking selects a branch, and triple-clicking selects the entire wire. Selects a segment Selects a branch Selects an entire wire



1.8.7 Bad Wires

A dashed wire represents a bad wire. You can get a bad wire for a number of reasons, such as connecting two controls, or connecting a source terminal to a destination terminal when the data types do not match (for instance, connecting a numeric to a Boolean). You can remove a bad wire by clicking on it with the Positioning tool and pressing <Delete>. Choosing Edit»Remove Bad Wires deletes all bad wires in the block diagram. This is a useful quick fix to try if your VI refuses to run or returns the Signal has loose ends error message.

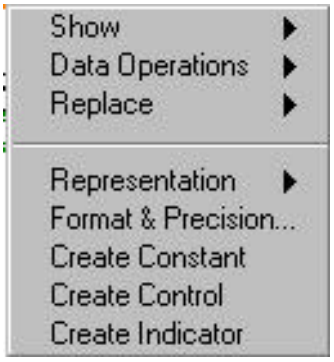


Note*: Do not confuse a black, dashed wire with a dotted wire. A dotted wire represents a Boolean data type, as the following illustration shows.*

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1.8.8 Create & Wire Controls, Constants, and Indicators

For terminals acting as inputs on the block diagram, LabVIEW has two features that you can use to create and wire a control or constant. You access these features by popping up on the terminal and choosing **Create Control** or **Create Constant**. LabVIEW automatically creates and wires the correct control or constant type to the terminal input. The following illustration shows an example pop-up menu.



For a terminal acting as an output on the block diagram, you can choose a **Create Indicator** feature to create and then wire an indicator to the terminal. You access this feature by popping up on the terminal and choosing **Create Indicator**. LabVIEW automatically creates and wires the correct indicator type to the output of a terminal.

Note: Once you choose Create Indicator, you must switch to the front panel and use the Positioning tool to select and delete the indicator.

1.8.9 Run the VI

1. To make the front panel active by clicking on the window title bar or by choosing **Windows»Show Panel**. In Windows and on the Macintosh, you can also make the front panel active by clicking anywhere on it.
2. Run the VI by clicking on the run button in the toolbar of the front panel.

Notice ***that you have to rerun the VI each time. If you want to repeatedly run the VI, you must click on the continuous run button.***

3. Click on the continuous run button in the toolbar.
4. Click on the continuous run button again to deselect it. The VI then completes execution and quits.

Note*: The continuous run button is not the preferred method for repeating block diagram code. You should use a looping structure. This is covered in, Loops and Charts, of this demonstration guide.*

1.8.10 Saving and Loading VIs

As with other applications, you can save your VI to a file in a regular directory. With LabVIEW, you can also save multiple VIs in a single file called a *VI library*. The tutorial.llb library is an example of a VI library.

If you are using Windows 3.1, you should save your VIs into VI libraries because you can use long file names (up to 255 characters) with mixed cases.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

You should not use VI libraries unless you need to transfer your VIs to Windows 3.1. Saving VIs as individual files is more effective because you can copy, rename, and delete files more easily than if you are using a VI library.

Even though you may not save your own VIs in VI libraries, you should be familiar with how they work. For that reason, you should save all VIs that you create during this demonstration guide into VI libraries to become familiar with using them.

Save your VI in a VI library.

1. Select **File»Save As...**. If you are using UNIX, specify a location in the file system where you have write privileges. For example, you might select your home directory.
2. Name the VI and save it in `mywork.llb`. Look at the name in the ring control at the top of the dialog box. Make sure it is `mywork.llb`. If it is not, click on `mywork.llb` in the directory list to make sure you save your VI in the right place.
   a. Type `My Thermometer.vi` in the dialog box.
   b. Click on **O**K.
3. Close the VI by selecting **File»Clos**e..

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

2 CREATING A SUBVI

You Will Learn:

- What a subVI is.
- How to create the icon and connector.
- How to use a VI as a subVI.

2.1 Understanding Hierarchy

One of the keys to creating LabVIEW applications is understanding and using the hierarchical nature of the VI. After you create a VI, you can use it as a *subVI* in the block diagram of a higher-level VI. Therefore, a subVI is analogous to a subroutine in C. Just as there is no limit to the number of subroutines you can use in a C program, there is no limit to the number of subVIs you can use in a LabVIEW program. You can also call a subVI inside another subVI.

When creating an application, you start at the top-level VI and define the inputs and outputs for the application. Then, you construct subVIs to perform the necessary operations on the data as it flows through the block diagram. If a block diagram has a large number of icons, group them into a lower-level VI to maintain the simplicity of the block diagram. This modular approach makes applications easy to debug, understand, and maintain.

2.2 Creating the SubVI

To make an icon and connector for the My Thermometer VI you created in Chapter 1 and use the VI as a subVI. To use a VI as a subVI, you must create an icon to represent it on the block diagram of another VI, and a connector pane to which you can connect inputs and outputs.

2.2.1 Icon

Create the icon, which represents the VI in the block diagram of other VIs. An icon can be a pictorial representation of the purpose of the VI, or it can be a textual description of the VI or its terminals.

1. If you have closed the My Thermometer VI, open it by selecting **File»Open...** or by clicking on the Open VI button in the dialog box. Open the `mywork.llb`. In Windows, you can find this library in the temporary directory or in `windows\temp`. On the Macintosh, you can find this directory in the `Temporary Folder` in the `System Folder`. In Unix, the `mywork.llb` is in the `/tmp` directory.
2. Select `My Thermometer.vi` from `mywork.llb`.
3. Invoke the Icon Editor by popping up in the icon pane in the upper right corner of the front panel and choosing **Edit Icon**. As a shortcut, you can also double-click on the icon pane to edit the icon.

2.2.2 Icon Editor Tools and Buttons

The tools to the left of the editing area perform the following functions:

Pencil tool Draws and erases pixel by pixel.

Line tool Draws straight lines. Press `<Shift>`and then drag this tool to

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

draw horizontal, vertical, and diagonal lines.

Dropper tool Copies the foreground color from an element in the icon.

Fill bucket tool Fills an outlined area with the foreground color.

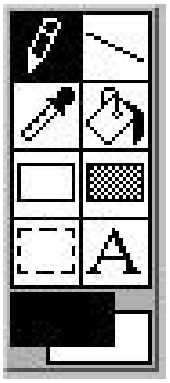
Rectangle tool Draws a rectangular border in the foreground color. Double-click on this tool to frame the icon in the foreground color.

Filled rectangle tool Draws a rectangle bordered with the foreground color and filled with the background color. Double-click to frame the icon in the foreground color and fill it with the background color.

Select tool Selects an area of the icon for moving, cloning, or other changes.

Text tool Enters text into the icon design.

Foreground/ Background Displays the current foreground and background colors. Click on each to get a color palette from which you can choose new colors.



Close the Icon Editor by clicking on **OK** once you complete your icon. The new icon appears in the icon pane in the upper right corner of the front panel.

2.2.3 Connector

Now, you can create the connector.

1. Define the connector terminal pattern by popping up in the icon pane on the front panel with the left mouse button and choosing **Show Connecto**r, as the following illustration shows.

Because LabVIEW selects a terminal pattern based on the number of controls and indicators on the front panel, there is only one terminal—the thermometer indicator.

2. Assign the terminal to the thermometer.
   a. Click on the terminal in the connector. The cursor automatically changes to the Wiring tool, and the terminal turns black.
   b. Click on the thermometer indicator. A moving dashed line frames the indicator, as the following illustration shows.

If you click in an open area on the front panel, the dashed line disappears and the selected terminal dims, indicating that you have assigned the indicator to that terminal. If the terminal is white, you have not made the connection correctly. Repeat the previous steps if necessary.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3. Save the VI by choosing **File»Sav**e. On the Macintosh, if you are using the native file dialog box to save into a VI library, you must click on the **Use LLBs** button before selecting the VI library.

This VI is now complete and ready for use as a subVI in other VIs. The icon represents the VI in the block diagram of the calling VI. The connector (with one terminal) outputs the temperature.

Note*: The connector specifies the inputs and outputs to a VI when you use it as a subVI. Remember that front panel controls can be used as inputs only; front panel indicators can be used as outputs only.*

4. Close the VI by choosing **File»Clos**e.

2.3 Using a VI as a SubVI

You can use any VI that has an icon and a connector as a subVI in another VI. In the block diagram, you select VIs to use as subVIs from **Functions»Select a VI...**. Choosing this option produces a file dialog box, from which you can select any VI in the system. If you open a VI that does not have an icon and a connector, a blank, square box appears in the calling VI's block diagram. You cannot wire to this node.

A subVI is analogous to a subroutine. A subVI node (icon/connector) is analogous to a subroutine call. The subVI node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes calls the same subVI several times.

To build a VI that uses the My Thermometer VI as a subVI. The My Thermometer VI you built returns a temperature in degrees Fahrenheit. You will take that reading and convert the temperature to degrees Centigrade.

2.3.1 Front Panel

1. Open a new front panel by selecting **File»New** or by clicking on the New VI button in the dialog box.
2. Choose the thermometer from **Controls»Numeri**c. Type `Temp in deg C` to label it. If you have clicked outside of the thermometer before typing in your label, it will disappear. To show the label again, pop up on the thermometer and choose **Sho**w»**Label** and then type in your label.
3. Change the range of the thermometer to accommodate the temperature values. With the Operating tool, double-click on the lower limit, type `20`, and press `<Enter>` on the numeric keypad. You do not have to type the decimal and trailing zeroes. LabVIEW adds them automatically when you enter the value. Similarly, change the upper limit of the thermometer to 40 and press `<Enter>` on the numeric keypad. LabVIEW automatically adjusts the intermediate values.

Each time you create a new control or indicator, LabVIEW creates the corresponding terminal in the block diagram. The terminal symbols suggest the data type of the control or indicator. For example, a DBL terminal represents a double-precision, floating-point number; a TF terminal is a Boolean; an I16 terminal represents a regular, 16-bit integer; and an ABC terminal represents a string.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

2.3.2 Block Diagram

1. Select **Windows»Show Diagram**.
2. Pop up in a free area of the block diagram and choose **Functions»Select a VI...**.A dialog box appears. Locate and open the `mywork.llb` library. (In Windows, you can find this library in the temporary directory or `windows\temp`. On the Macintosh, this directory is in `System Folder\Temporary Folder`. In Unix, the `mywork.llb` is in the `/tmp` directory.) Double-click on `My Thermometer.vi` or highlight it and click on **Open** in the dialog box. LabVIEW places the My Thermometer VI on the block diagram.
3. Add the other objects to the block diagram as shown in the following illustration. Numeric Constant **(Functions»Numeric**). Add three numeric constants to the block diagram. Assign the values of 32.0, 5.0, and 9.0 to the constants by using the Labeling tool.

Note: **You can tell the type of constant the number is by its color. Blue numeric constants are integers, and orange constants are double-precision numbers. LabVIEW automatically converts numbers to the appropriat format when necessary.**

Note: **Remember, you can use the pop up on functions and choose Create Constant to automatically create and wire the correct constant to a function.**

The Subtract function **(Functions»Numeric**) subtracts 32 from the Fahrenheit value for the conversion to Centigrade. The Divide function **(Functions»Numeric**) computes the value of 5/9 for the temperature conversion. The Multiply function **(Functions»Numeric**) returns the Centigrade value from the conversion process.

4. Wire the diagram objects as shown in the previous block diagram illustration.

Note*: A broken wire between the Thermometer icon and the Temp in deg C terminal might indicate that you have assigned the subVI connector terminal to the front panel indicator incorrectly. Review the instructions in the Creating the SubVI section earlier in this chapter. When you have modified the subVI, you may need to select Relink to SubVI from the icon pop-up menu. If necessary, choose Edit»Remove Bad Wires.*

5. Return to the front panel and click on the run button in the toolbar. Block Diagram Toolbar

The block diagram contains additional options not included on the front panel toolbar.

2.3.3 Some Debugging Techniques

The thermometer should display a value in the selected range. However, suppose you want to see the Fahrenheit value for comparison and debugging. LabVIEW contains some tools that can help you. In this exercise, you examine the probe and execution highlighting features.

1. Select **Windows»Show Diagram**.
2. Select the Probe tool from the **Tools** palette. Click with the Probe tool on the temperature value (wire) coming out of the My Thermometer subVI. A Probe window pops up with the title `Temp 1` and a yellow glyph with the number of the probe, as shown in the following illustration. The Probe window also appears on the front panel.
3. Return to the front panel. Move the Probe window so you can view both the probe and thermometer values as shown in the following illustration. Run the VI. The temperature in degrees Fahrenheit appears in the Probe window.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Note*: The temperature values that appear on your screen may be different than what is shown in this illustration. Refer to the Numeric Conversion section in Chapter 3, Loops and Charts, for more information*

4. Close the Probe window by clicking in the close box at the top of the Probe window title bar.

   Another useful debugging technique is to examine the flow of data in the block diagram using LabVIEW's execution highlighting feature.

5. Return to the block diagram of the VI by choosing **Windows»Show Diagram**.

6. Begin execution highlighting by clicking on the hilite execute button, in the toolbar, shown at left. The hilite execute button changes to an illuminated light bulb.

7. Click on the run button to run the VI, and notice that execution highlighting animates the VI block diagram execution. Moving bubbles represent the flow of data through the VI. Also notice that data values appear on the wires and display the values contained in the wires at that time, as shown in the following block diagram, just as if you had probed the wire.

Notice the order in which the different nodes in LabVIEW execute. In conventional text-based languages, the program statements execute in the order in which they appear. LabVIEW, however, uses *data flow* programming. In data flow programming, a node executes when data is available at all of the node inputs, not necessarily in a top-to-bottom or left-to-right manner.

The preceding illustration shows that LabVIEW can *multitask* between paths 1 and 2 because there is no data dependency, that is, nothing in path 1 depends on data from path 2, and nothing in path 2 depends on data from path 1. Path 3 must execute last, however, because the multiply function is dependent upon the data from the Subtract and Divide functions.

Execution highlighting is a useful tool for examining the data flow nature of LabVIEW.

You can also use the single stepping buttons if you want to have more control over the debugging process.

8. Begin single stepping by clicking on the step over button, in the toolbar. Clicking on this button displays the first execution sequence in the VI. After LabVIEW completes this portion of the sequence, it highlights the next item that executes in the VI.

9. Step over the divide function by clicking on the step over button, in the toolbar. Clicking on this button executes the Divide function. After LabVIEW completes this portion of the sequence, it highlights the next item that executes in the VI.

10. Step into the My Thermometer subVI by clicking on the step into button, in the toolbar. Clicking on this button opens the front panel and block diagram of your thermometer subVI. You can now choose to single step through or run the subVI.

11. Finish executing the block diagram by clicking on the step out button, in the toolbar. Clicking on this button completes all remaining sequences in the block diagram. After LabVIEW completes this portion of the sequence, it highlights the next item that executes in the VI. You can also hold down the mouse button when clicking on the step out button to access a pop-up menu. On this pop-up menu, you can select how far the VI executes before pausing. The following illustration shows your finish execution options in the pop-up menu of the step out button.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

12. Select **File»Save as** and save the VI in `mywork.llb`. Name the VI `Using My Thermometer.vi`, and then close it.

2.3.4 Opening, Operating, and Changing SubVIs

You can open a VI used as a subVI from the block diagram of the calling VI. You open the block diagram of the subVI by double-clicking on the subVIs icon or by selecting **Project»This VI's.**

SubVIs. You then open the block diagram by selecting **Windows»Show Diagra**m.

Any changes you make to a subVI alter only the version in memory until you save the subVI. Notice that the changes affect all calls to the subVI and not just the node you used to open the VI.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3 LOOPS AND CHARTS

You Will Learn:

- How to use a While Loop.
- How to display data in a chart.
- What a shift register is and how to use it.
- How to use a For Loop.

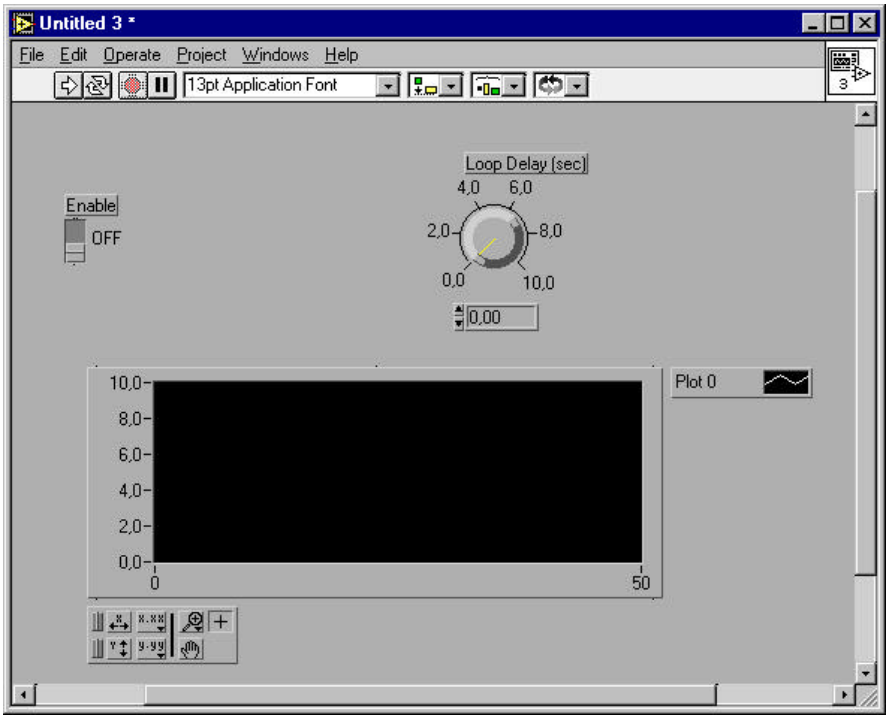
Structures control the flow of data in a VI. LabVIEW has four structures: the While Loop, the For Loop, the Case structure, and the Sequence structure. This chapter introduces the While Loop and For Loop structures along with the chart and the shift register. The Case and Sequence structures are explained later

3.1 Using While Loops and Charts

To use a While Loop and a chart for acquiring and displaying data in real time.

You will build a VI that generates random data and displays it on a chart. A knob control on the front panel will adjust the loop rate between 0 and 2 seconds and a switch will stop the VI. You will learn to change the mechanical action of the switch so you do not have to turn on the switch each time you run the VI. Use the front panel in the following illustration to get started.

3.1.1 Front Panel



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

1. Open a new front panel by selecting **File»New** or by clicking on the New VI button in the dialog box.
2. Place a vertical switch **(Controls»Boolean**) in the front panel. Label the switch `Enable`. You use this switch to stop the acquisition.
3. Use the Labeling tool to create the free label for ON and OFF.You can create these labels by clicking on the Labeling tool and then on your front panel and typing in the label text. Use the Color tool to make the free label border transparent. Click on the Color tool and select the `T` in the bottom left corner of the color palette to make the label transparent.
4. Place a waveform chart **(Controls»Graph**) in the front panel. Label the chart `Random Signal`. The chart displays random data in real time.

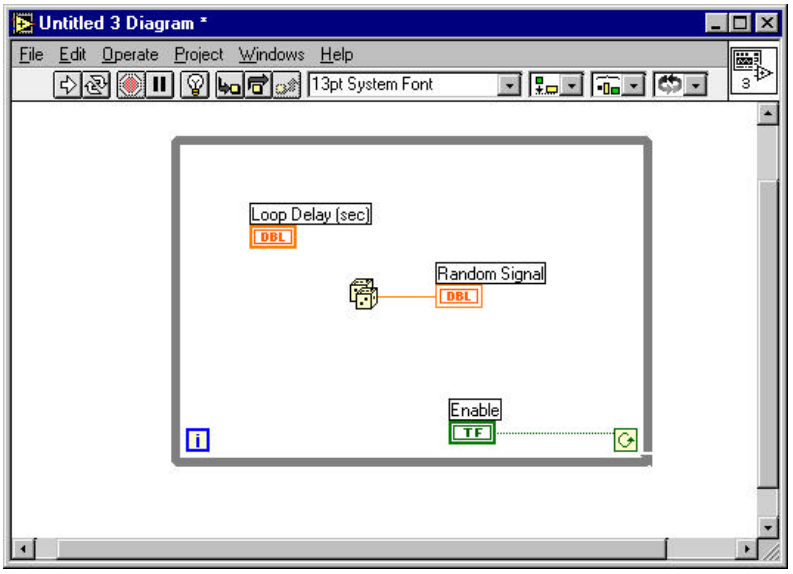
Note*: Make sure that you select a waveform chart and not a waveform graph. In the Graph palette the waveform chart appears closest to the left side of the palette.*

5. Pop up on the chart and choose **Show»Digital Displa**y. The digital display shows the latest value.
6. Using the Labeling tool, double-click on 10.0 in the chart, type 1.0, and click outside the label area. The click value. You can also press `<Enter>` (Windows); `<return>` (Macintosh); `<Return>` (Sun); or `<Enter>` (HP-UX) to input your change to the scale.
7. Place a knob **(Controls»Numeri**c) in the front panel. Label the knob `Loop Delay (sec)`. This knob controls the timing of the While Loop later in this exercise. Pop up on the knob and deselect **Show»Digital Display** to hide the digital display that shows by default.
8. Using the Labeling tool, double-click on `10.0` in the scale around the knob, type 2.0, and click outside the label area to enter the new value.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.1.2 Block Diagram

1. Open the block diagram.
2. Place the While Loop in the block diagram by selecting it from **Functions»Structure**s. The While Loop is a resizable box that is not dropped on the diagram immediately. Instead, you have the chance to position and resize it. To do so, click in an area above and to the left of all the terminals. Continue holding down the mouse button, and drag out a rectangle that encompasses the terminals. A While Loop is then created with the specified location and size.
3. Select the Random Number (0-1) function from **Functions»Numeri**c.



4. Wire the diagram as shown in the opening illustration of this *Block Diagram* section, connecting the Random Number (0-1) function to the Random Signal chart terminal, and the Enable switch to the conditional terminal of the While Loop. Leave the Loop Delay terminal unwired for now.
5. Return to the front panel and turn on the vertical switch by clicking on it with the Operating tool. Run the VI.

The While Loop is an indefinite looping structure. The diagram within its border executes as long as the specified condition is true. In this example, as long as the switch is on (TRUE), the diagram continues to generate random numbers and display them on the chart.

6. To stop the loop, click on the vertical switch. Turning the switch off sends the value FALSE to the loop conditional terminal and stops the loop.
7. The chart has a display buffer that retains a number of points after they have scrolled off the display. Give the chart a scrollbar by popping up on the chart and selecting **Show»Scrollba**r. You can use the Positioning tool to adjust the size and position of the scrollbar.

To scroll through the chart, click and hold down the mouse button on either arrow in the scrollbar. To clear the display buffer and reset the chart, pop up on the chart and choose **Data Operations»Clear Char**t.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Note*: The display buffer default size is 1,024 points. You can increase or decrease this buffer size by popping up on the chart and choosing Chart History Length....*

3.1.3 Adding Timing

When you ran the VI, the While Loop executed as quickly as possible. However, you may want to take data at certain intervals, such as once per second or once per minute.

The While Loop, shown in the following illustration, is a resizable box you use to execute the diagram inside it until the Boolean value passed to the *conditional terminal* (an input terminal) is FALSE. The VI checks the conditional terminal at the *end* of each iteration; therefore, *the While Loop always executes at least onc*e. The *iteration terminal* is an output numeric terminal that contains the number of times the loop has executed. However, the iteration count always starts at zero, so if the loop runs once, the iteration terminal outputs 0. The While Loop is equivalent to the following pseudo-code:

Do

Execute Diagram Inside the Loop (which sets the condition)

While Condition is TRUE

LabVIEW's timing functions express time in milliseconds (ms), however, your operating system may not maintain this level of timing accuracy. The following list contains guidelines for determining the accuracy of LabVIEW's timing functions on your system.

- • **(Windows 3.1)** The timer has a default resolution of 55 ms. You can configure LabVIEW to have 1 ms resolution by selecting **Edit»Preferences...**, selecting Performance and Disk from the Paths ring, and unchecking the Use Default Timer checkbox. LabVIEW does not use the 1 ms resolution by default because it places a greater load on your operating system.
- **(Windows 95/NT)** The timer has an resolution of 1 ms. However, this is hardware dependent, so on slower systems, such as an 80386, you may have lower resolution timing.
- **(Macintosh)** For 68K systems without the QuickTime extension, the timer has an resolution of 16 2/3 ms (1/60th of a second). If you have a Power Macintosh or have QuickTime installed, timer resolution is 1 ms.
- **(UNIX)** The timer has a resolution of 1 ms.

You can control loop timing using the Wait Until Next ms Multiple function **(Functions»Time & Dialo**g). This function ensures that no iteration is shorter than the specified number of milliseconds.

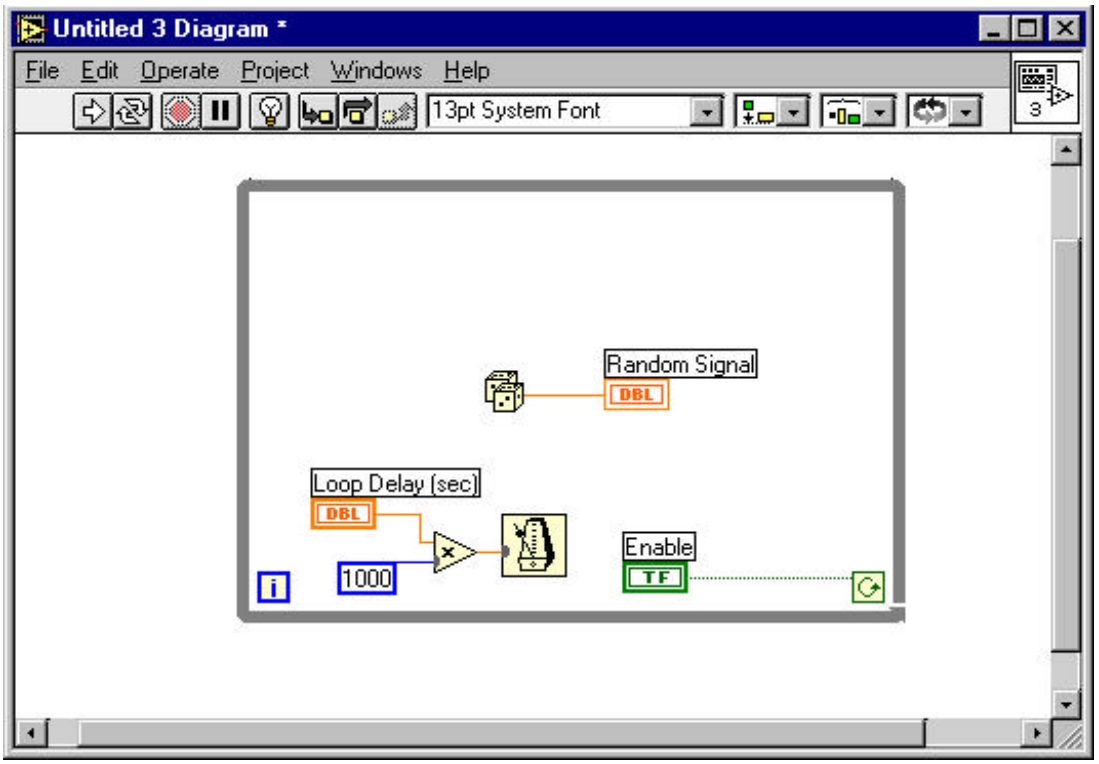
Modify the VI to generate a new random number at a time interval specified by the knob, as shown in the preceding diagram.

Wait Until Next ms Multiple function **(Functions»Time & Dialog**). In this exercise, you multiply the knob terminal by 1000 to convert the knob value in seconds to milliseconds. Use this value as the input to the Wait Until Next ms Multiple function. Multiply function **(Functions»Numeri**c). In this exercise, the multiply function multiplies the knob value by 1000 to convert seconds to milliseconds.

Numeric Constant **(Functions»Numeri**c).The numeric constant holds the constant by which you must multiply the knob value to get a quantity in milliseconds. Thus, if the knob has a value of 1.0, the loop executes once every 1000 milliseconds (once a second).

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Run the VI. Rotate the knob to get different values for the number of seconds.
Save and close the VI in mywork.llb. Name it My Random Signal.vi.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.2 For Loop

You place the *For Loop* on the block diagram by selecting it from **Functions»Structure**s. A For Loop (see preceding illustration) is a resizable box, like the While Loop. Like the While Loop, it is not dropped on the diagram immediately. Instead, a small icon representing the For Loop appears in the block diagram, and you have the opportunity to size and position it. To do so, first click in an area above and to the left of all the terminals. While holding down the mouse button, drag out a rectangle that encompasses the terminals you want to place inside the For Loop. When you release the mouse button, LabVIEW creates a For Loop of the correct size and in the position you selected.

The For Loop executes the diagram inside its border a predetermined number of times. The For Loop has two terminals:

- the *count terminal* (an input terminal) The count terminal specifies the number of times to execute the loop.
- the *iteration terminal* (an output terminal). The iteration terminal contains the number of times the loop has executed.

The For Loop is equivalent to the following pseudo-code:

```
For i = 0 to N-1
Execute Diagram Inside The Loop
```

The example in the following illustration shows a For Loop that generates 100 random numbers and displays the points on a chart.

3.2.1 Numeric Conversion

Until now, all the numeric controls and indicators that you have used have been double-precision, floating-point numbers represented with 32 bits. LabVIEW, however, can represent numerics as integers (byte, word, or long) or floating-point numbers (single-, double-, or extended-precision). The default representation for a numeric is a double-precision, floating-point.

If you wire two terminals together that are of different data types, LabVIEW converts one of the terminals to the same representation as the other terminal. As a reminder, LabVIEW places a gray dot, called a coercion dot, on the terminal where the conversion takes place.

For example, consider the For Loop count terminal. The terminal representation is a long integer. If you wire a double-precision, floating-point number to the count terminal, LabVIEW converts the number to a long integer. Notice the gray dot in the count terminal of the first For Loop.

Note: ***When the VI converts floating-point numbers to integers, it rounds to the nearest integer. If a number is exactly halfway between two integers, it is rounded to the nearest even integer. For example, the VI rounds 6.5 to 6, but rounds 7.5 to 8. This is an IEEE Standard method for reading numbers. See the IEEE Standard 754 for details.***

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.2.2 Using a For Loop

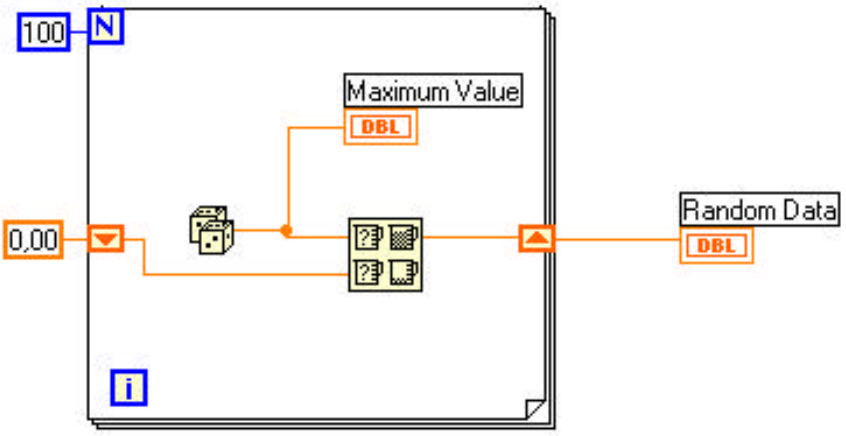
To use a For Loop and shift registers to calculate the maximum value in a series of random numbers. You will use a For Loop (N = 100) instead of a While Loop.

3.2.3 Front Panel

1. Open a new front panel and add the objects shown in the preceding illustration to it.
   a. Place a digital indicator on the front panel and label it `Maximum Value`.
   b. Place a waveform chart on the front panel and name it `Random Data`. Change the scale of the chart to range from 0.0 to 1.0.
   c. Pop up on the chart and choose **Show»Scrollbar** and **Show»Digital Displa**y. Pop up and disable the **Sho**w»**Palette** option if it is selected.

3.2.4 Block Diagram

1. Open the block diagram.
2. Add the **For Loop (Functions»Structure**s).
3. Add the shift register by popping up or right clicking on the right or left border of the



   For Loop and choosing **Add Shift Registe**r. You can learn more about shift registers in the next section.
4. Add the other objects to the block diagram.

Random Number (0-1) function **(Functions»Numeri**c) to generate the random data.

Numeric Constant **(Functions»Numeri**c). The For Loop needs to know how many iterations to make. In this case, you execute the For Loop 100 times.

Numeric Constant **(Functions»Numeri**c). You set the initial value of the shift register to zero for this exercise because you know that the output of the random number generator is from 0.0 to 1.0.

You must know something about the data you are collecting to initialize a shift register. For example, if you initialize the shift register to 1.0, then that value is already greater than all the expected data values, and is always the maximum value. If you did not initialize the shift register, then it would contain the maximum value of a previous run of the VI. Therefore, you could get a maximum output value that is not related to the current set of collected data.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Max & Min function **(Functions»Comparison)** takes two numeric inputs and outputs the maximum value of the two in the top right corner and the minimum of the two in the bottom right corner. Because you are only interested in the maximum value for this exercise, wire only the maximum output and ignore the minimum output.

5. Wire the terminals as shown. If the Maximum Value terminal were inside the For Loop, you would see it continuously updated, but because it is outside the loop, it contains only the last calculated maximum.

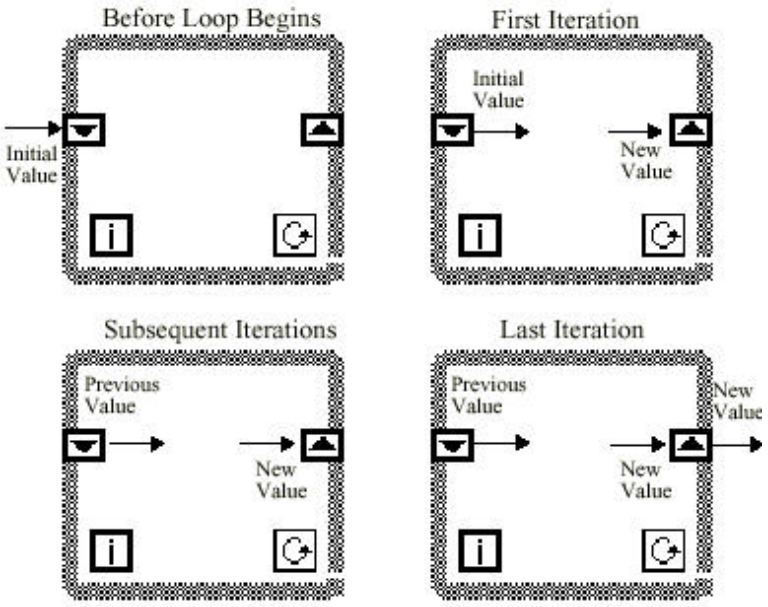
Note*: Updating indicators each time a loop iterates is time-consuming and you should try to avoid it when possible to increase execution speed.*

6. Run the VI.
7. Save the VI. Name the VI `My Calculate Max.vi`.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.3 Shift Registers

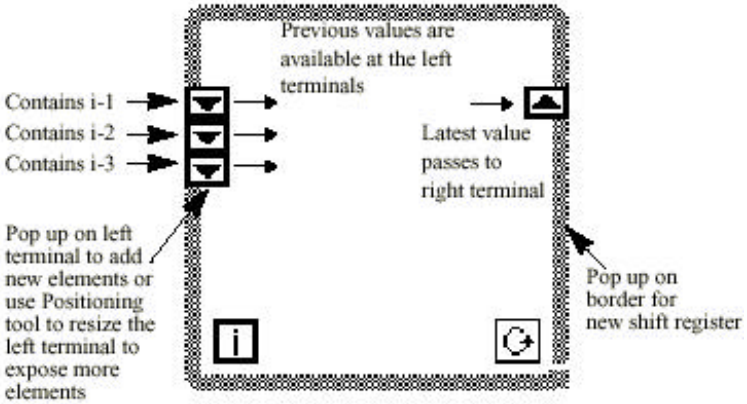
Shift registers (available for While Loops and For Loops) transfer values from one loop iteration to the next. You create a shift register by popping up on the left or right border of a



loop and selecting **Add Shift Register**.

The shift register contains a pair of terminals directly opposite each other on the vertical sides of the loop border. The *right* terminal stores the data upon the completion of an iteration. That data shifts at the end of the iteration and appears in the *left* terminal at the beginning of the next iteration (see the following illustration). A shift register can hold any data type–numeric, Boolean, string, array, and so on. The shift register automatically adapts to the data type of the first object that you wire to the shift register.

You can configure the shift register to remember values from several previous iterations. This feature is useful for averaging data points. You create additional terminals to access values from previous iterations by popping up on the *left* or *right* terminal and choosing **Add Element**. For example, if a shift register contains three elements in the left terminal, you can



access values from the last three iterations.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.3.1 Using Shift Registers

You will build a VI that displays two random plots on a chart. The two plots should consist of a random plot and a running average of the last four points of the random plot.

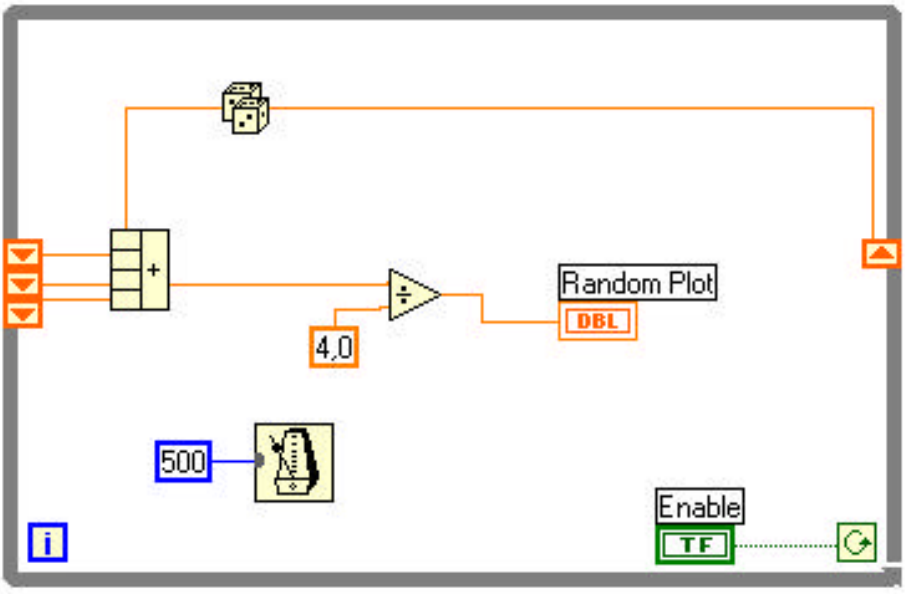
3.3.1.1 Front Panel

1. Open a new front panel and create the front panel shown in the preceding illustration.

2. After you add the waveform chart to the front panel, change the scale to range from 0.0 to 2.0.
3. After adding the vertical switch, pop up on the button on the front panel and select **Mechanical Actio**n»**Latch When Pressed** and set the ON state to be the default by choosing **Operat**e»**Make Current Values Defaul**t.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.3.1.2 Block Diagram



1. Add the **While Loop (Functions»Structures)** in the block diagram and create the shift register.
   a. Pop up on the left or right border of the While Loop and choose **Add Shift Register**.
   b. Add an extra element by popping up on the *left* terminal of the shift register and choosing **Add Element**. Add a third element in the same manner as the second.
2. Build the block diagram shown in the previous illustration.

Random Number (0-1) function **(Functions»Numeric)** generates raw data. Compound Arithmetic function **(Functions»Numeric)**. In this exercise, the compound arithmetic function returns the sum of random numbers from two iterations. To add more inputs, pop up on an input and choose **Add Input** from the pop-up menu.

Divide function **(Functions»Numeric)**. In this exercise, the divide function returns the average of the last four random numbers.

Numeric Constant **(Functions»Numeric)**. During each iteration of the While Loop, the Random Number (0-1) function generates one random value. The VI adds this value to the last three values stored in the left terminals of the shift register. The Random Number (0-1) function divides the result by four to find the average of the values (the current value plus the previous three). The average is then displayed on the waveform chart.

Wait Until Next ms Multiple function **(Functions»Time & Dialog)**, ensures that each iteration of the loop occurs no faster than the millisecond input. The input is 500 milliseconds for this exercise. If you pop up on the icon and choose **Show»Label**, the label `Wait Until Next ms Multiple` appears.

3. Pop up on the input of the Wait Until Next ms Multiple function and select **Create Constant**. A numeric constant appears and is automatically wired to the function.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

4. Type `500` in the label. The numeric constant wired to the Wait Until Next ms Multiple function specifies a wait of 500 milliseconds (one half-second). Thus, the loop executes once every half-second.

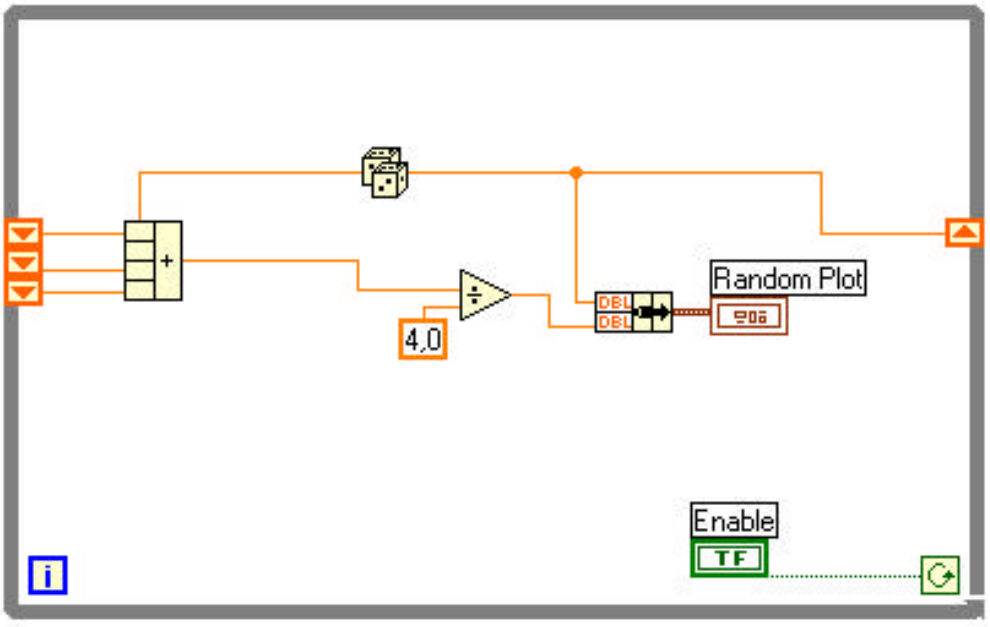
Notice that the VI initializes the shift registers with a random number. If you do not initialize a shift register terminal, it contains the default value or the last value from the previous run. In this case, the first few averages would be meaningless.

5. Run the VI and observe the operation. LabVIEW only plots the average on the graph.

Note*: Remember to initialize shift registers to avoid incorporating old or default data into your current data measurements*

3.3.2 Multiplot Charts

Charts can accommodate more than one plot. You must bundle the data together in the case of multiple scalar inputs.



You should modify the block diagram to display both the average and the current random number on the same chart.

1. Modify the block diagram as shown in the previous illustration.

Bundle function **(Functions»Cluste**r). In this exercise, the Bundle function *bundle*s, or groups, the average and current value for plotting on the chart. The bundle node appears as shown at left when you place it in the block diagram. If you pop up on the bundle and choose **Show»Labe**l, the word `Bundle` appears in the label. You can add additional elements by using the Resizing cursor (accessed by placing the Positioning tool at the corner of the function) to enlarge the node.

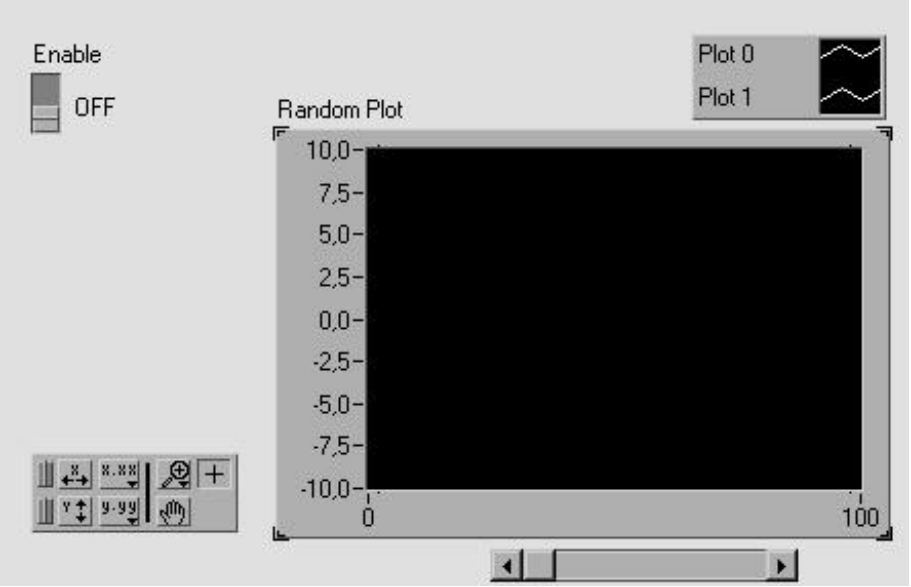
Note*: The order of the inputs to the Bundle function determines the order of the plots on the chart. For example, if you wire the raw data to the top input of the Bundle and the average to the bottom, the first plot corresponds to the raw data and the second plot corresponds to the average*

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

2. Run the VI. The VI displays two plots on the chart. The plots are overlaid. That is, they share the same vertical scale. Try running the VI with execution highlighting turned on to see the data in the shift registers. Remember to turn off the hilite execute button, in the toolbar, when you finish so the VI can execute at full speed.

3.3.3 Customizing Charts

You can customize charts to match your data display requirements or to display more information. Features available for charts include: a scrollbar, a legend, a palette, and a digital display.



On the chart, the digital display has been enabled. Notice that a separate digital display exists for each trace on the chart.

1. If the scrollbar is present, hide it by popping up on the chart and deselecting **Show»ScrollBa**r.
2. Customize the Y axis.
   a. Use the Labeling tool to double-click on 2.0 in the Y scale. Type in 1.2 and press `<Enter>` (Windows); `<return>` (Macintosh); `<Return>` (Sun); or `<Enter>` (HP-UX).
   b. Again using the Labeling tool, click on the second number from the bottom on the Y axis. Change this number to 0.2, 0.5, or something other than the current number. This number determines the numerical spacing of the Y axis divisions.

Note*: The chart size has a direct effect on the display of axis scales. Increase the chart size if you have trouble customizing the axis.*

3. Show the legend by popping up on the chart, and choosing **Show»Legend**. Move the legend if necessary.

You can place the legend anywhere relative to the chart. Stretch the legend to include two plots using the Resizing cursor. The Positioning tool changes to the Resizing cursor to

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

indicate that you can resize the legend. Rename `0` to `Current Value` by double-clicking on the label with the Labeling tool and typing in the new text. You can change plot 1 to `Running Avg` in the same way. If the text disappears, enlarge the legend text box by resizing from the *left* corner of the legend with the Resizing cursor. You can set the plot line style and the point style by popping up on the plot in the legend.

You can set the plot line width by popping up on the plot in the legend. Using this pop-up menu, you can change the default line setting to one that is larger than 1 pixel. You can also select a hairline width, which is not displayed on the computer screen, but is printed if your printer supports hairline printing.

If you have a color monitor, you can also color the plot background, traces, or point style by popping up on what you want to change in the legend with the Color tool. Choose the color you want from the color palette that appears.

4. Show the chart pop-up palette by popping up on the chart and choosing **Show»Palett**e.

With the palette, you can modify the chart display while the VI is running. You can reset the chart, scale the X or Y axis, and change the display format at any time. You can also scroll to view other areas or zoom into areas of a graph or chart. Like the legend, you can place the palette anywhere relative to the chart.

5. Run the VI. While the VI is running, use the buttons from the palette to modify the chart.

You can use the X and Y buttons to rescale the X and Y axes, respectively. If you want the graph to autoscale either of the scales continuously, click on the lock switch to the left of each button to lock on autoscaling.

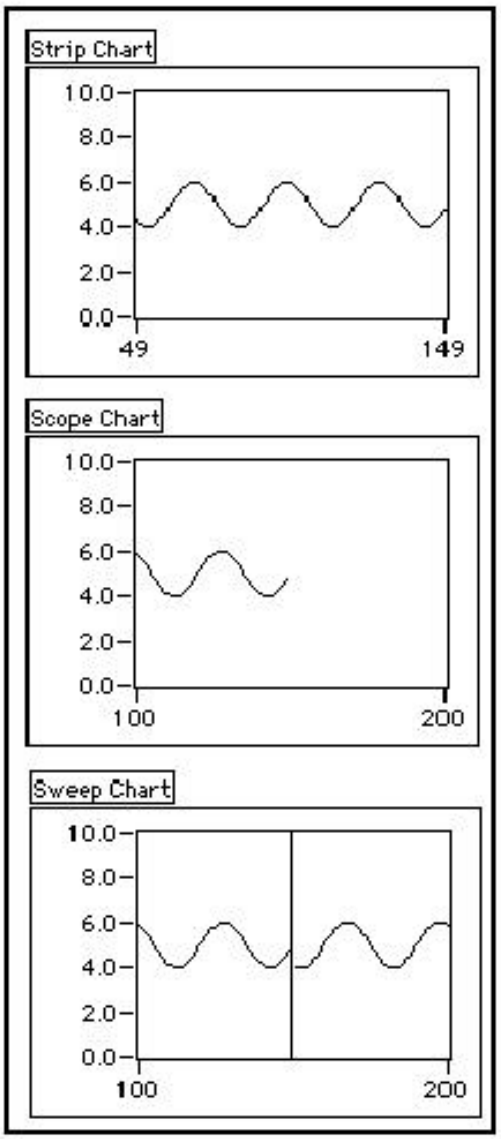
You can use the other buttons to modify the axis text precision or to control the operation mode for the chart. Experiment with these buttons to explore their operation, scroll the area displayed, or zoom in on areas of the chart.

Note*: Modifying the axis text format often requires more physical space than was originally set aside for the axis. If you change the axis, the text may become larger than the maximum size that the waveform can correctly present. To correct this, use the Resizing cursor to make the display area of the chart smaller.*

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

3.3.4 Different Chart Modes

The following illustration shows the three chart display options available from the **Data Operations»Update Mode**: strip chart, scope chart, and sweep chart. The default mode is strip chart. (If the VI is still running, the **Data Operations** submenu is the pop-up menu for the chart.)



The *strip chart* mode scrolling display is similar to a paper tape strip chart recorder. As the VI receives each new value, it plots the value at the right margin, and shifts old values to the left.

1. Make sure the VI is still running, pop up on the chart, and select **Data Operations»Update Mode»Scope Chart**.

The *scope chart* mode has a retracing display similar to an oscilloscope. As the VI receives each new value, it plots the value to the right of the last value. When the plot reaches the right border of the plotting area, the VI erases the plot and begins plotting again from the left

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

border. The scope chart is significantly faster than the strip chart because it is free of the overhead processing involved in scrolling.

2. Make sure the VI is still running, pop up on the chart, and select **Data Operations»Update Mode»Sweep Char**t.

The *sweep chart* mode acts much like the scope chart, but it does not go blank when the data hits the right border. Instead, a moving vertical line marks the beginning of new data and moves across the display as the VI adds new data.

3. Stop the VI, and save it. Name it `My Random Average.vi`.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

4 ARRAYS, CLUSTERS, AND GRAPHS

You Will Learn:

- About arrays.
- How to generate arrays on loop boundaries.
- What polymorphism is.
- About clusters.
- How to use graphs to display data.
- How to use some basic array functions.

4.1 Arrays

An array consists of a collection of data elements that are all the same type. An array has one or more dimensions and up to elements per dimension, memory permitting. Arrays in LabVIEW can be any type (except array, chart, or graph). You access each array element through its index. The index is in the range 0 to n-1, where *n* is the number of elements in the array. The following one-dimensional array of numeric values illustrates this structure. Notice that the first element has index 0, the second element has index 1, and so on.

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

4.1.1 Array Controls, Constants, and Indicators

You create array controls, constants, and indicators on the front panel or block diagram by combining an *array shell* with a numeric, Boolean, string, or cluster. The array element cannot be another array, chart, or graph.

4.1.2 Graphs

A *graph indicator* consists of a two-dimensional display of one or more data arrays called *plot*s. LabVIEW has three types of graphs:

*XY graph*s, *waveform graph*s, and *intensity graphs* (see the *Additional Topics* section at the end of this chapter for information on intensity graphs).

The difference between a graph and a chart (discussed in Chapter 3, *Loops and Charts*, in this demonstration guide) is that a graph plots data as a block, whereas a chart plots data point by point or array by array.

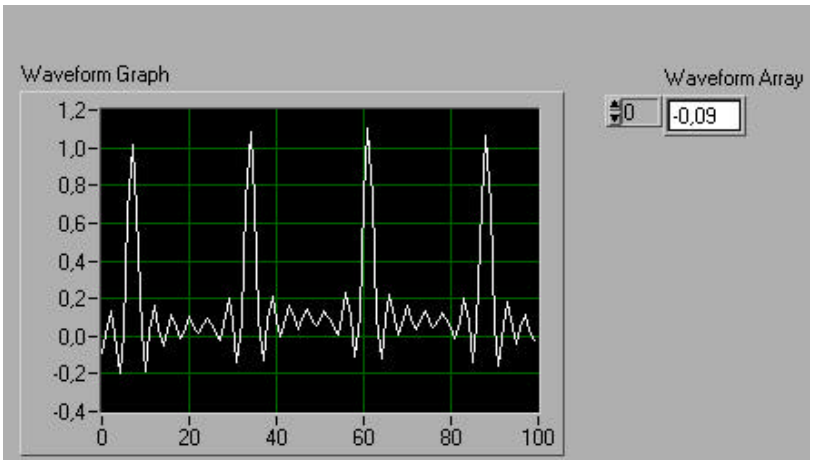
4.2 Creating an Array with Auto-Indexing

To create an array using the auto-indexing feature of a For Loop and plot the array in a waveform graph.

You will build a VI that generates an array using the Generate Waveform VI and plots the array in a waveform graph. You will also modify the VI to graph multiple plots.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

4.2.1 Front Panel



Open a new front panel.

Place an array shell from Controls»Array & Cluster in the front panel. Label the array shell `Waveform Array`.

Place a digital indicator from Controls»Numeric inside the element display of the array shell, as the following illustration shows. This indicator displays the array contents.

As stated previously, a *graph indicator* is a two-dimensional display of one or more data arrays called *plot*s. LabVIEW has three types of graphs: *XY graph*s, *waveform graph*s, and *intensity graph*s.

Place a waveform graph from Controls»Graph in the front panel. Label the graph `Waveform Graph`.

The waveform graph plots arrays with uniformly spaced points, such as acquired time-varying waveforms.

Enlarge the graph by dragging a corner with the Resizing cursor.

By default, graphs *autoscale* their input. That is, they automatically adjust the X and Y axis scale limits to display the entire input data set.

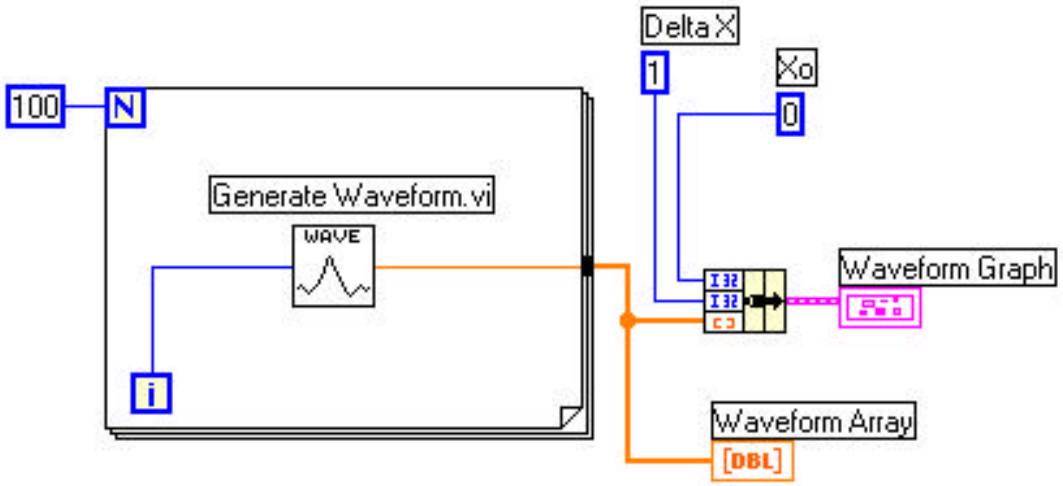
Disable autoscaling by popping up on the graph and deselecting Y Scale»Autoscale Y.

Modify the Y axis limits by double-clicking on the scale limits with the Labeling tool and entering the new numbers. Change the Y axis minimum to -0.5 and the maximum to 1.5.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

4.2.2 Block Diagram

1 Build the block diagram shown in the preceding illustration.



The Generate Waveform VI (Functions»Tutorial) returns one point of a waveform. The VI requires a scalar index input, so wire the loop iteration terminal to this input. Popping up on the VI and selecting Show»Label displays the word `Generate Waveform` in the label.

Notice that the wire from the Generate Waveform VI becomes thicker as it changes to an array at the loop border.

The For Loop automatically accumulates the arrays at its boundary. This is called *auto-indexing*. In this case, the numeric constant wired to the loop count numeric input has the For Loop create a 100-element array (indexed 0 to 99).

Bundle function (Functions»Cluster) assembles the plot components into a cluster. You need to resize the Bundle function icon before you can wire it properly. Place the Positioning tool on the lower right corner of the icon. The tool transforms into the Resizing cursor shown at left. When the tool changes, click and drag down until a third input terminal appears. Now, you can continue wiring your block diagram as shown in the first illustration in this section.

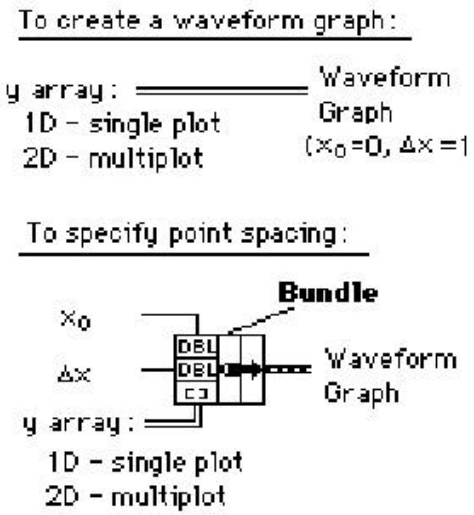
A cluster consists of a data type that can contain data elements of different types. The cluster in the block diagram you are building here groups related data elements from multiple places on the diagram, reducing wire clutter. When you use clusters, your subVIs require fewer connection terminals. A cluster is analogous to a record in Pascal or a struct in C. You can think of a cluster as a bundle of wires, much like a telephone cable. Each wire in the cable would represent a different element of the cluster. The components include the initial X value (0), the delta X value (1), and the Y array (waveform data, provided by the numeric constants on the block diagram). In LabVIEW, use the Bundle function to assemble a cluster.

Note*: Be sure to build data types that the graphs and charts accept.*

As you build your block diagram, be sure to check your data types by taking the following steps:

- Open the Help window by choosing Help»Show Help.
- Move the Wiring tool over the graph terminal.
- Check the data type information that appears in the Help window. For an example, see the following illustration.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de



Numeric Constant (Functions»Numeric). Three numeric constants set the number of For Loop iterations, the initial X value, and the delta X value. Notice that you can pop up on the For Loop count terminal, shown at left, and select Create Constant to automatically add and wire a numeric constant for that terminal.

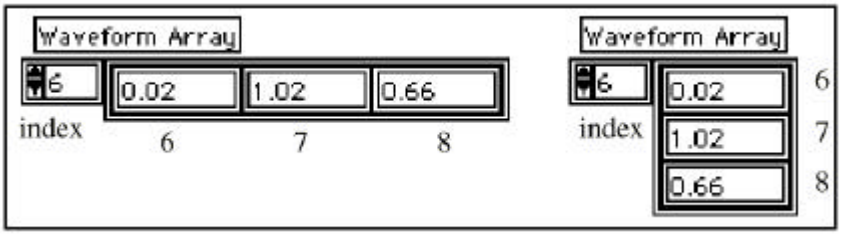
Each iteration of the For Loop generates one point in a waveform that the VI stores in the waveform array created automatically at the loop border. After the loop finishes execution, the Bundle function bundles the initial value of X (Xo), the delta value of X, and the array for plotting on the graph.

3. Return to the front panel and run the VI. The VI plots the auto-indexed waveform array on the waveform graph. The initial X value is 0 and the delta X value is 1.
4. Change the delta X value to 0.5 and the initial X value to 20. Run the VI again.

Notice that the graph now displays the same 100 points of data with a starting value of 20 and a delta X of 0.5 for each point (see the X axis). In a timed test, this graph would correspond to 50 seconds worth of data starting at 20 seconds. Experiment with several combinations for the initial and delta X values.

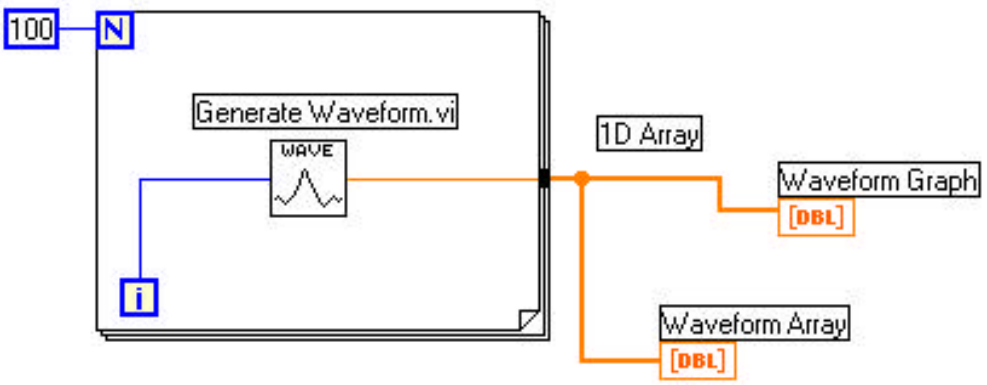
5. You can view any element in the array by entering the index of that element in the index display. If you enter a number greater than the array size, the display dims, indicating that you have not defined a value for that index.

If you want to view more than one element at a time, you can resize the array indicator. Place the Positioning tool on the lower right corner of the array. The tool transforms into the Resizing cursor shown at left. When the tool changes, drag to the right or straight down. The array now displays several elements in ascending index order, beginning with the element corresponding to the specified index, as the following illustration shows.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

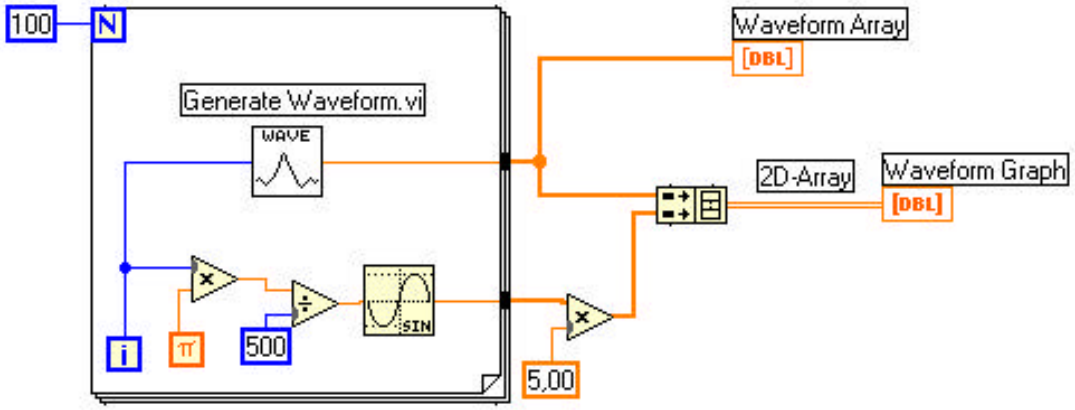
In the previous block diagram, you specified an initial X and a delta X value for the waveform. Often, however, the initial X value is zero and the delta X value is 1. In these instances, you can wire the waveform array directly to the waveform graph terminal, as the following illustration shows.



6. Return to the block diagram. Delete the Bundle function and the numeric constants wired to it. To delete the function and constants, select the function and constants with the Positioning tool then press <Delete>. Select Edit»Remove Bad Wires. Finish wiring the block diagram as shown in the previous illustration.
7. Run the VI. Notice that the VI plots the waveform with an initial X value of 0 and a delta X value of 1.

4.2.3 Multiplot Graphs

You can create multiplot waveform graphs by building an array of the data type normally passed to a single-plot graph.



Continue building your block diagram as shown in the preceding diagram.

Sine function from (Functions»Numeric»Trigonometric). In this exercise, you use the function in a For Loop to build an array of points that represents one cycle of a sine wave.

Build Array function (Functions»Array). In this exercise, you use this function to create the proper data structure to plot two arrays on a waveform graph, which in this case is a two-dimensional array. Enlarge the Build Array function to create two inputs by dragging a corner with the Positioning tool.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Pi constant (Functions»Numeric»Additional Numeric Constants).

Remember that you can find the Multiply and Divide functions in Functions»Numeric.

Switch to the front panel. Run the VI.

Notice that the two waveforms plot on the same waveform graph. The initial X value defaults to 0 and the delta X value defaults to 1 for both data sets.

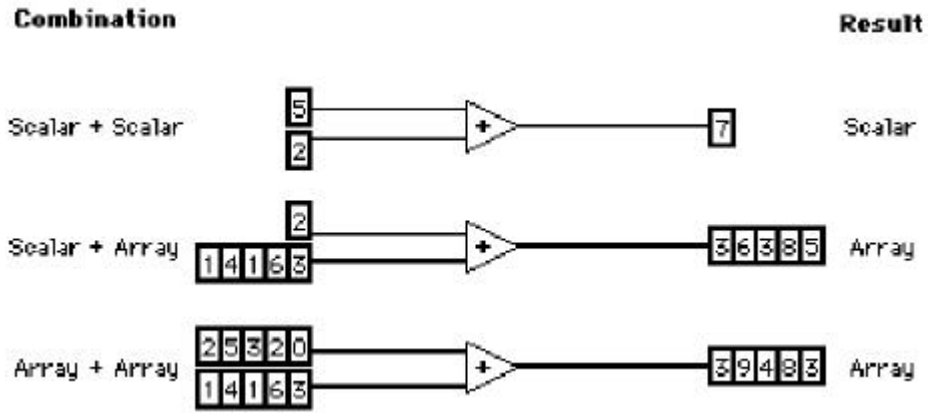
Note*: You can change the appearance of a plot on the graph by popping up in the legend for a particular plot. For example, you can change from a line graph to a bar graph by choosing Common Plots»Bar Graph.*

Save and close the VI. Name it `My Graph Waveform Arrays.vi`. Be sure to save your work in `mywork.llb`.

4.3 Polymorphism

Polymorphism is the ability of a function to adjust to input data of different types, dimensions, or representations. Most LabVIEW functions are polymorphic. The previous block diagram is an example of polymorphism. Notice that you use the Multiply function in two locations, inside and outside the For Loop. Inside the For Loop, the function multiplies two scalar values; outside the For Loop, the function multiplies an array by a scalar value.

The following example shows some of the polymorphic combinations of the Add function. In the first combination, the two scalars are added together, and the result is a scalar. In the second combination, the scalar is added to each element of the array, and the result is an array.



In the third combination, each element of one array is added to the corresponding element of the other array. You can also use other combinations, such as clusters of numerics, arrays of clusters, and so on.

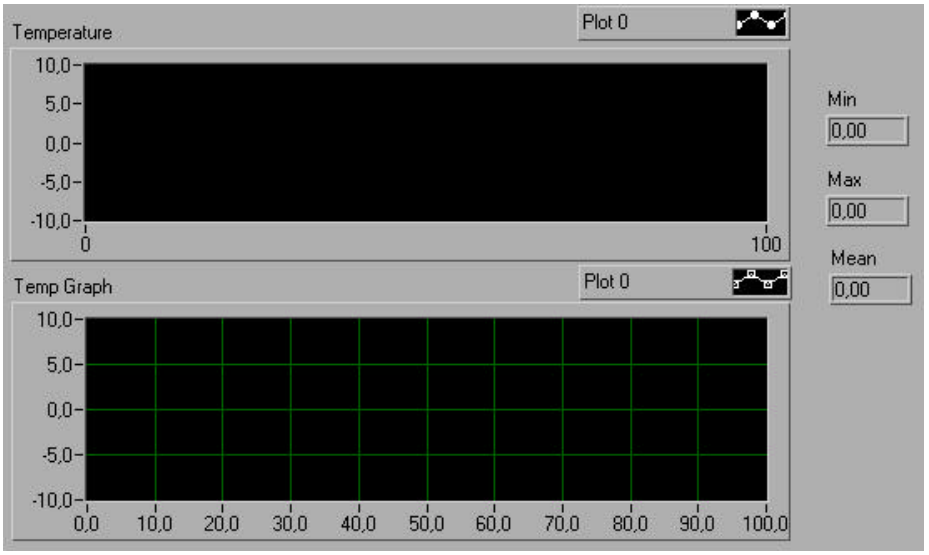
These principles can be applied to other LabVIEW functions and data types. LabVIEW functions may be polymorphic to different degrees. Some functions may accept numeric and Boolean inputs, others may accept a combination of any data types.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

4.4 Using the Graph and Analysis VIs

You will build a VI that measures temperature every 0.25 seconds for 10 seconds. During the acquisition, the VI displays the measurements in real time on a strip chart. After completing the acquisition, the VI plots the data on a graph and calculates the average, maximum, and minimum temperatures.

4.4.1 Front Panel



Open a new front panel and build the front panel shown in the preceding illustration. You can modify the point styles of the waveform chart and waveform graph by popping up on their legends.

The Temperature waveform chart displays the temperature as it is acquired. After acquisition, the VI plots the data in Temp Graph. The Mean, Max, and Min digital indicators display the average, maximum, and minimum temperatures.

4.4.2 Block Diagram

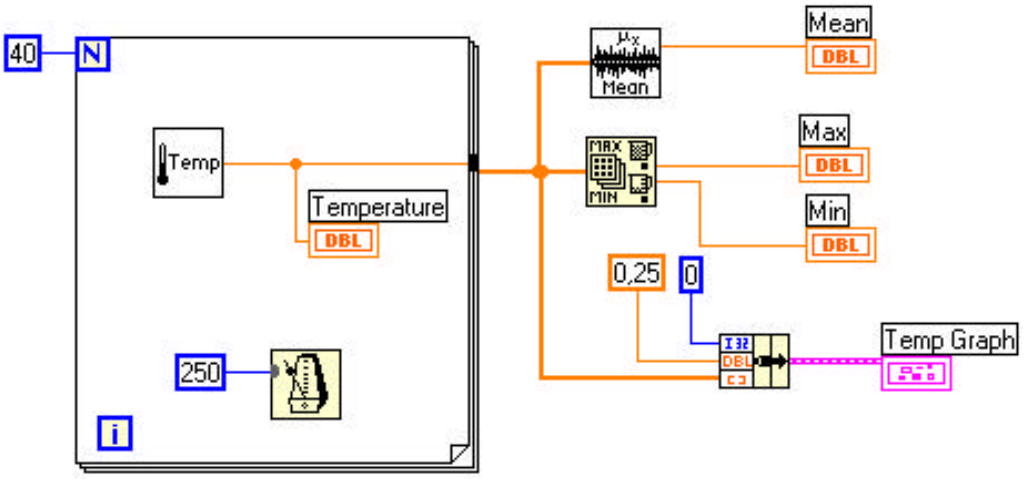
Build the block diagram shown in the previous illustration, using the following elements:

The Digital Thermometer VI (Functions»Tutorial, or you can use the VI you built in Chapter 2 by choosing Functions»Select a VI... and selecting My Thermometer VI. Returns one temperature measurement.

Wait Until Next ms Multiple function (Functions»Time & Dialog). In this exercise, this function ensures the For Loop executes every 0.25 seconds (250 milliseconds).

Numeric constant (Functions»Numeric). You can also pop up on the Wait Until Next ms Multiple function and select Create Constant to automatically create and wire the numeric constant.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de



Array Max & Min function (Functions»Array). In this exercise, this function returns the maximum and minimum temperature measured during the acquisition.

The Mean VI (Functions»Analysis»Probability and Statistics) returns the average of the temperature measurements.

Bundle function (Functions»Cluster) assembles the plot components into a cluster. The components include the initial X value (0), the delta X value (0.25), and the Y array (temperature data). Use the Positioning tool to resize the function by dragging one of the corners.

The For Loop executes 40 times. The Wait Until Next ms Multiple function causes each iteration to take place every 250 milliseconds. The VI stores the temperature measurements in an array created at the For Loop border (auto-indexing). After the For Loop completes execution, the array passes to various nodes.

The Array Max & Min function returns the maximum and minimum temperature. The Mean VI returns the average of the temperature measurements.

Your completed VI bundles the data array with an initial X value of 0 and a delta X value of 0.25. The VI requires a delta X value of 0.25 so that the VI plots the temperature array points every 0.25 seconds on the waveform graph.

Return to the front panel and run the VI.

Save the VI in `mywork.llb` as `My Temperature Analysis.vi`.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

5 CASE AND SEQUENCE STRUCTURES AND THE FORMULA NODE

You Will Learn:

- How to use the Case structure.
- How to use the Sequence structure.
- What Sequence Locals are and how to use them.
- What a Formula Node is and how to use it.

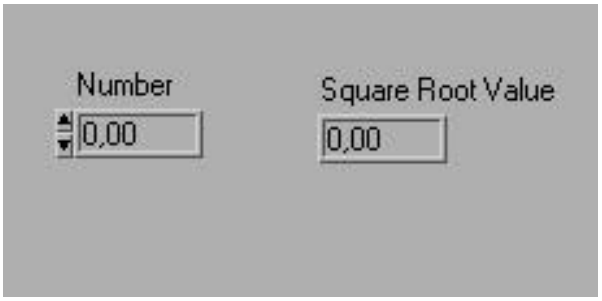
5.1 Using the Case Structure

You will build a VI that checks a number to see if it is positive. If the number is positive the VI calculates the square root of the number; otherwise, the VI returns an error.

5.1.1 Front Panel

1. Open a new front panel and build the front panel as shown in the previous illustration.

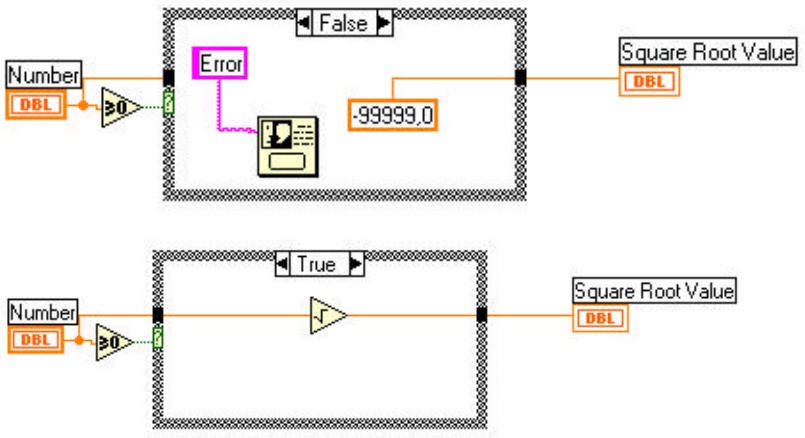
The Number control supplies the number. The Square Root Value indicator displays the square root of the number. The free label acts as a note to the user.



5.1.2 Block Diagram

1. Open the block diagram.
2. Place a Case structure **(Functions»Structure**s) in the block diagram. Enlarge the Case structure by dragging one corner with the Resizing cursor.

By default, the Case structure is Boolean and it has only two cases: True and False. A



Boolean Case structure is analogous to an if-then-else statement in text-based, programming

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

languages. It Selection Terminal automatically changes to numeric when you wire a numeric control to the selection terminal.

You can display only one case at a time. To change cases, click on the arrows at the top of the Case structure.

3. Select the other block diagram objects and wire them as shown in the block diagram illustration.

Greater Or Equal To 0? function **(Functions»Comparison**). In this exercise, the function determines whether the number input is negative. The function returns a TRUE if the number input is greater than or equal to 0.

Square Root function (**Functions»Numeric**). In this exercise, the function returns the square root of the input number.

Numeric Constant **(Functions»Numeric**).

One Button Dialog function **(Functions»Time & Dialog**). In this exercise, the function displays a dialog box that contains the message `Error...Negative Number`.

String Constant **(Functions»String**). Enter text inside the box with the Labeling tool.

In this exercise, the VI executes either the True case or the False case. If the number is greater than or equal to zero, the VI executes the True case and returns the square root of the number. The False case outputs `-99999.00` and displays a dialog box with the message `Error...Negative Number`.

Note*: You must define the output tunnel for each case. When you create an output tunnel in one case, tunnels appear at the same position in all the other cases. Unwired tunnels appear as white squares.*

Be sure to wire to the output tunnel for each unwired case, clicking on the tunnel itself each time. In this exercise, you assign a value to the output tunnel in the False case because the True case has an output tunnel. If you do not want to assign the output in all cases to a value, then you must put the indicator in that case or use a global or local variable.

4. Return to the front panel and run the VI. Try a number greater than zero and a number less than zero by changing the value in the digital control you labeled `Number`. Notice that when you change the digital control to a negative number, LabVIEW displays the error message you set up in the False case of the case structure.

5. Save and close the VI. Name it `My Square Root.vi`.

5.1.3 VI Logic

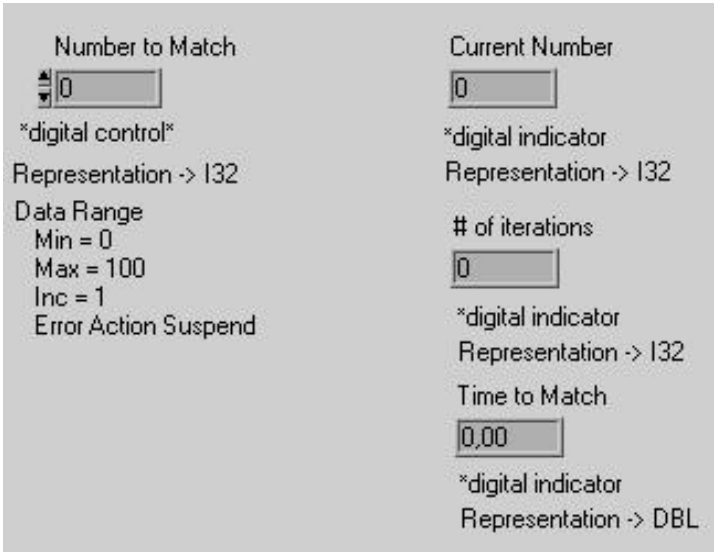
```
if (Number >= 0) then
Square Root Value = SQRT(Number)
else
Square Root Value = -99999.00
Display Message "Error...Negative Number"
end if
```


Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

5.2 Using the Sequence Structure

You will build a VI that computes the time it takes to generate a random number that matches a given number.

5.2.1 Front Panel



1. Open a new front panel and build the front panel shown in the following illustration. Be sure to modify the controls and indicators as described in the text following the illustration.

The `Number to Match` control contains the number you want to match. The `Current Number` indicator displays the current random number. The `# of iterations` indicator displays the number of iterations before a match. `Time to Match` indicates how many seconds it took to find the matching number.

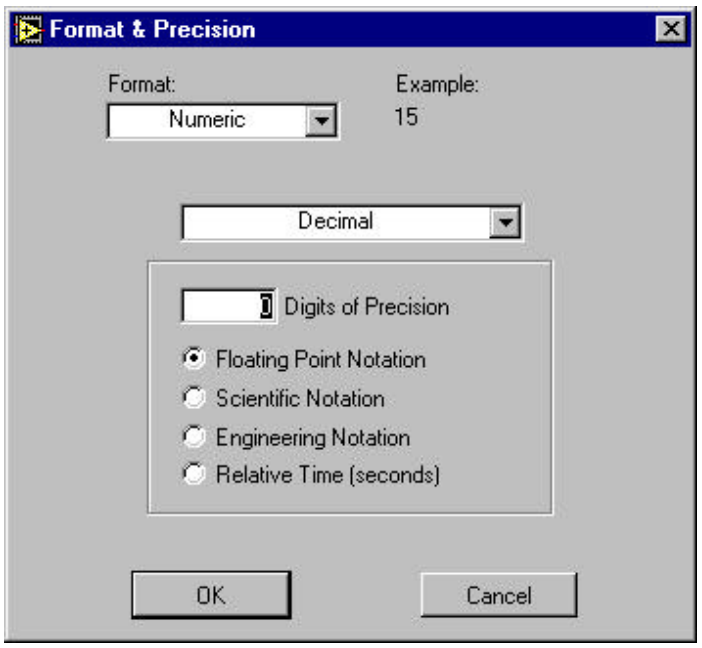
5.2.2 Modifying the Numeric Format

By default, LabVIEW displays values in numeric controls in decimal notation with two decimal places (for example, 3.14). You can use the **Format & Precision...** option of a control or indicator pop-up menu to change the precision or to display the numeric controls and indicators in scientific or engineering notation. You can also use the **Format & Precision...** option to denote time and date formats for numerics.

1. Change the precision on the `Time to Match` indicator.
   a. Pop up on the `Time to Match` digital indicator and choose **Format & Precision...**. You must be in the front panel to access the menu.
   b. Enter a `3` for Digits of Precision and click on **O**K.
2. Change the representation of the digital control and two of the digital indicators to long integers.
   a. Pop up on the Number to Match digital control and choose **Representation»Lon**g.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

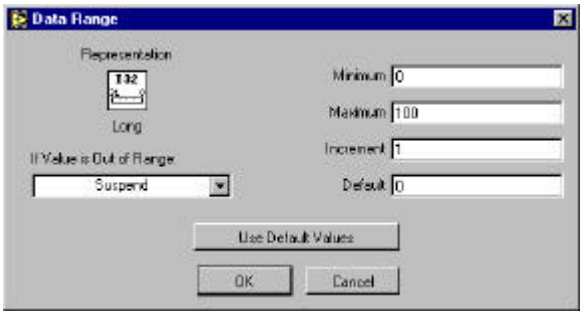
b. Repeat the previous step for the Current Number, and the # of iterations digital indicators.



5.2.3 Setting the Data Range

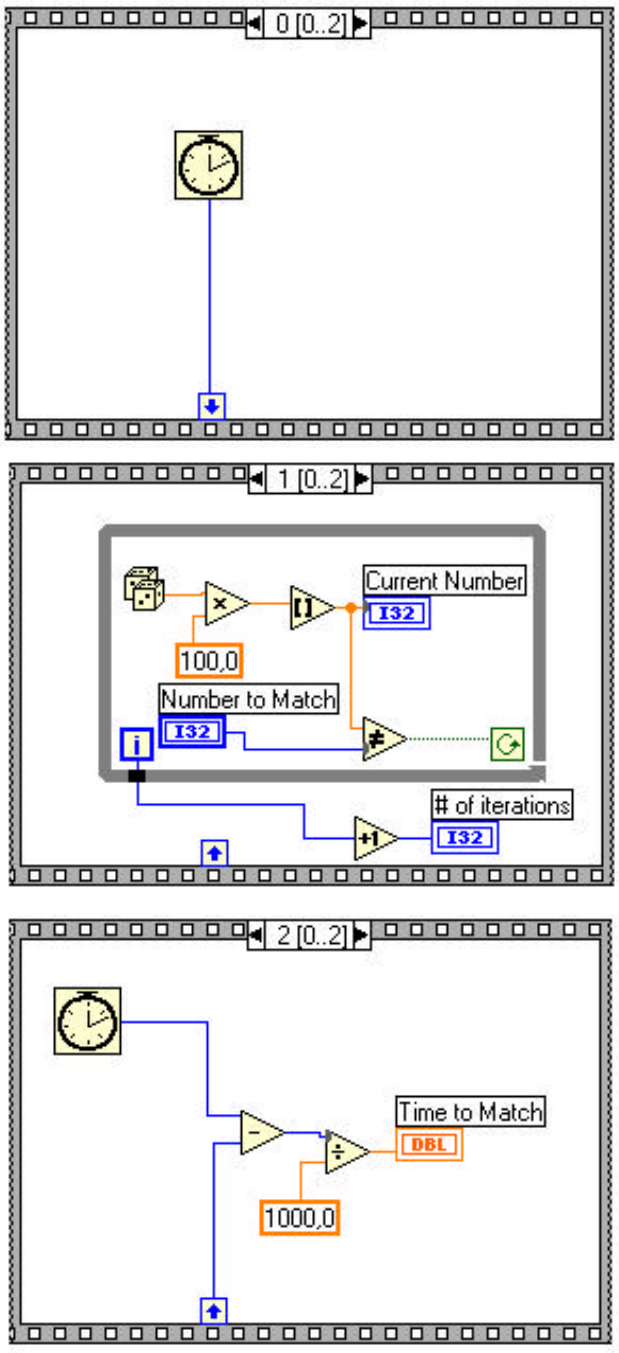
With the **Data Range...** option you can prevent a user from setting a control or indicator value outside a preset range or increment. Your options are to ignore the value, coerce it to within range, or suspend execution. The range error symbol appears in place of the run button, in the toolbar, when a range error suspends execution. Also, a solid, dark border frames the control that is out of range.

1. Set the data range between 0 and 100 with an increment of 1.
   a. Pop up on the Time to Match indicator and choose **DataRange...**.
   b. Fill in the dialog box, as shown in the following illustration, and click on **O**K.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

5.2.4 Block Diagram



1. Open the block diagram.
2. Place the Sequence structure **(Functions»Structure**s) in the block diagram.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

The Sequence structure, which looks like frames of film, executes block diagrams sequentially. In conventional programming languages, the program statements execute in the order in which they appear. In data flow programming, a node executes when data is available at all of the node inputs, although sometimes it is necessary to execute one node before another. LabVIEW uses the Sequence structure as a method to control the order in which nodes execute. LabVIEW places the diagram that the VI executes first inside the border of Frame 0, it places the diagram it executes second inside the border of Frame 1, and so on. As with the Case structure, only one frame is visible at a time.

3. Enlarge the structure by dragging one corner with the Resizing cursor.
4. Create a new frame by popping up on the frame border and choose **Add Frame After**. Repeat this step to create Frame 2.

Frame 0 in the previous illustration contains a small box with an arrow in it. That box is a sequence local variable which passes data between frames of a Sequence structure. You can create sequence locals on the border of a frame. The data wired to a frame sequence local is then available in subsequent frames. However, you cannot access the data in frames preceding the frame in which you created the sequence local.

5. Create the sequence local by popping up on the bottom border of Frame 0 and choosing **Add Sequence Local**.

The sequence local appears as an empty square. The arrow inside the square appears automatically when you wire a function to the sequence local.

6. Finish the block diagram as shown in the opening illustration of the *Block Diagram* section.

Tick Count (ms) function **(Functions»Time & Dialog**). Returns the number of milliseconds that have elapsed since power on. For this exercise, you need two Tick Count functions.

Random Number (0-1) function **(Functions»Numeric**). Returns a random number between 0 and 1.

Multiply function **(Functions»Numeric**). In this exercise, the function multiplies the random number by 100. In other words, the function returns a random number between 0.0 and 100.0.

Numeric Constant function **(Functions»Numeric**). In this exercise, the numeric constant represents the maximum number that can be multiplied.

Round to Nearest function **(Functions»Numeric**). In this exercise, the function rounds the random number between 0 and 100 to the nearest whole number.

Not Equal? function **(Functions»Comparison**). In this exercise, the function compares the random number to the number specified in the front panel and returns a TRUE if the numbers are not equal. Otherwise, this function returns FALSE.

Increment function **(Functions»Numeric**). In this exercise, the function increments the While Loop count by 1.

Subtract function **(Functions»Numeric**). In this exercise, the function returns the time (in milliseconds) elapsed between Frame 2 and Frame 0.

Divide function **(Functions»Numeric**). In this exercise, the function divides the number of milliseconds elapsed by 1000 to convert the number to seconds.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Numeric constant **(Functions»Numeric**). In this exercise, the function converts the number from milliseconds to seconds.

In Frame 0, the Tick Count (ms) function returns the current time in milliseconds. This value is wired to the sequence local, where the value is available in subsequent frames. In Frame 1, the VI executes the While Loop as long as the number specified does not match the number that the Random Number (0-1) function returns. In Frame 2, the Tick Count (ms) function returns a new time in milliseconds. The VI subtracts the old time (passed from Frame 0 through the Sequence local) from the new time to compute the time elapsed.

7. Return to the front panel and enter a number inside the Number to Match control and run the VI.
8. Save and close the VI. Name it `My Time to Match.vi`.

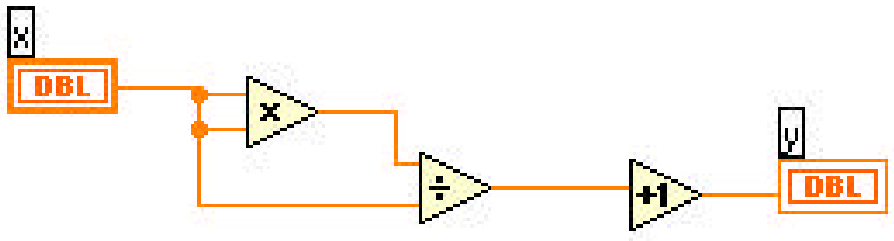
5.3 Formula Node

The *Formula Node* is a resizable box that you can use to enter formulas directly into a block diagram. You place the Formula Node on the block diagram by selecting it from **Function»Structure**s. This feature is useful when an equation has many variables or is otherwise complicated. For example, consider the equation:

$$y = x^2 + x + 1.$$

If you implement this equation using regular LabVIEW arithmetic functions, the block diagram looks like the one in the following illustration.

You can implement the same equation using a Formula Node, as shown in the following illustration.

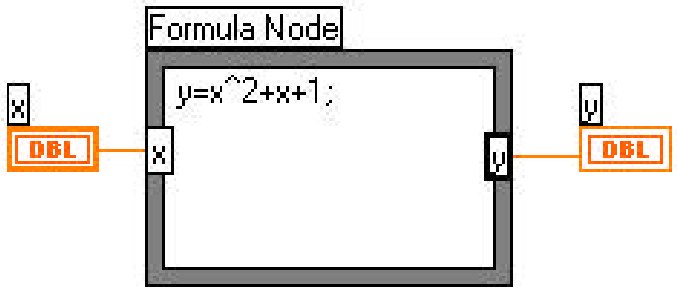


With the Formula Node, you can directly enter a complicated formula, or formulas, in lieu of creating block diagram subsections. You enter formulas with the Labeling tool. You create the input and output terminals of the Formula Node by popping up on the border of the node and choosing **Add Input (Add Output**). Type the variable name in the box. Variables are case sensitive. You enter the formula or formulas inside the box. Each formula statement must end with a semicolon (;).

The operators and functions available inside the Formula Node are listed in the Help window for the Formula Node. A semicolon terminates each formula statement.

The following example shows how you can perform a conditional assignment inside a Formula Node.

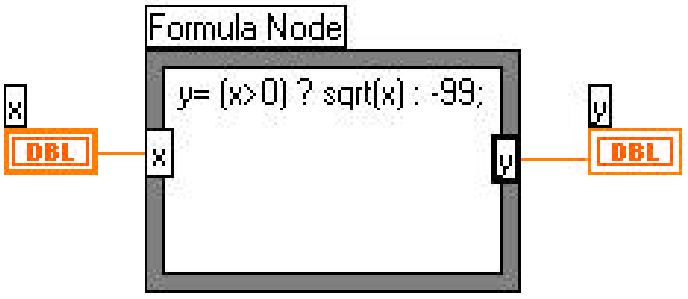
Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de



Consider the following code fragment that computes the square root of *x* if *x* is positive, and assigns the result to y. If *x* is negative, the code assigns -99 to y.

```
if (x >= 0) then
y = sqrt(x)
else
y = -99
end if.
```

You can implement the code fragment using a Formula Node, as shown in the following diagram.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

5.4 Using the Formula Node

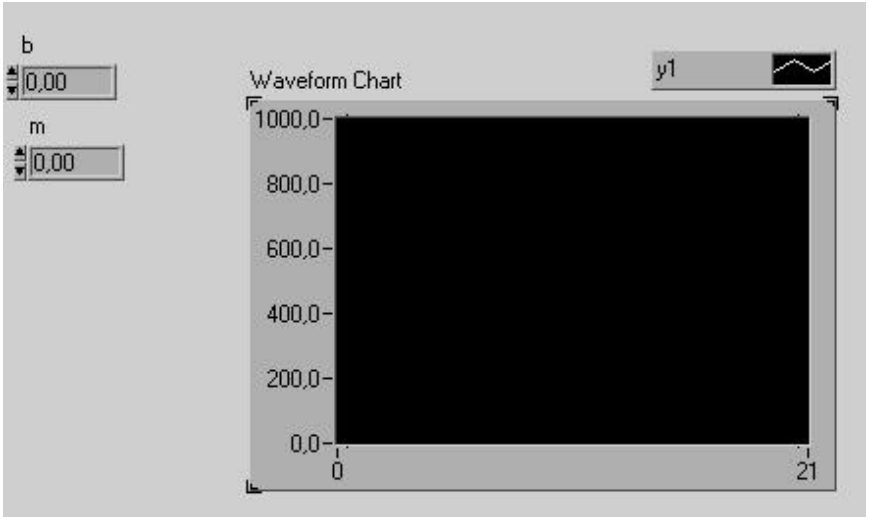
You will build a VI that uses the Formula Node to calculate the following equations.

```
y1 = x 3 - x 2 + 5
y2 = m * x + b
```

where *x* ranges from 0 to 10.

You will use only one Formula Node for both equations, and you will graph the results on the same graph.

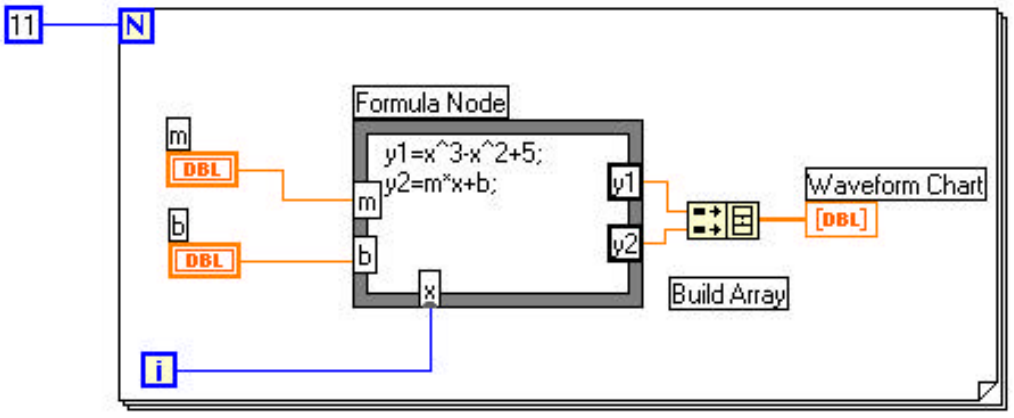
5.4.1 Front Panel



1. Open a new front panel and build the front panel shown in the preceding illustration. The waveform graph indicator displays the plots of the equation. The VI uses the two digital controls to input the values for `m` and `b`.

5.4.2 Block Diagram

1. Build the block diagram shown in the preceding illustration.
2. Place the For Loop **(Functions»Structure**s) in the block diagram and drag the corner to enlarge the loop.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Formula Node **(Functions»Structures**). With this node, you can directly enter formula(s). Create the three input terminals by popping up on the border and choosing **Add Input**. You create the output terminal by choosing **Add Output** from the pop-up menu.

When you create an input or output terminal, you must give it a variable name. The variable name must exactly match the one you use in the formula. The names are case sensitive. That is, if you use a lower case `a` in naming the terminal, you must use a lower case `a` in the formula. You can enter the variable names and formula with the Labeling tool.

Note*: **Although variable names are not limited in length, be aware that long names take up considerable diagram space. A semicolon (;) terminates the formula statement.***

Numeric Constant **(Functions»Numeric**). You can also pop up on the count terminal and select **Create Constant** to automatically create and wire the numeric constant. The numeric constant specifies the number of For Loop iterations. If x range is 0 to 10 including 10, you need to wire 11 to the count terminal.

Because the iteration terminal counts from 0 to 10, you use it to control the *X* value in the Formula Node.

Build Array **(Functions»Array**) puts two array inputs into the form of a multiplot graph. Create the two input terminals by using the Resizing cursor to drag one of the corners.

3. Return to the front panel and run the VI with different values for *m* and b.
4. Save and close the VI. Name the VI `My Equations.vi`..

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6 STRINGS AND FILE I/O

You Will Learn:

- How to create string controls and indicators.
- How to use string functions.
- About file input and output operations.
- How to save data to files in spreadsheet format.
- How to write data to and read data from text files.

6.1 Strings

A string is a collection of ASCII characters. You can use strings for more than simple text messages. In instrument control, you can pass numeric data as character strings and then convert these strings to numbers. Storing numeric data to disk can also involve strings. To store numbers in an ASCII file, you must first convert numbers to strings before writing the numbers to a disk file.

6.1.1 Creating String Controls and Indicators

You can find the string control and indicator, shown at left, in **Controls»String & Table**. You can enter or change text inside a string control using the Operating tool or the Labeling tool. Enlarge string controls and indicators by dragging a corner with the Positioning tool.

6.1.2 Strings and File I/O

If you want to minimize space that a front panel string control or indicator occupies, select **Show»Scrollbar**. If this option is dimmed, you must increase the vertical size of the window to make it available.

6.2 Using String Functions

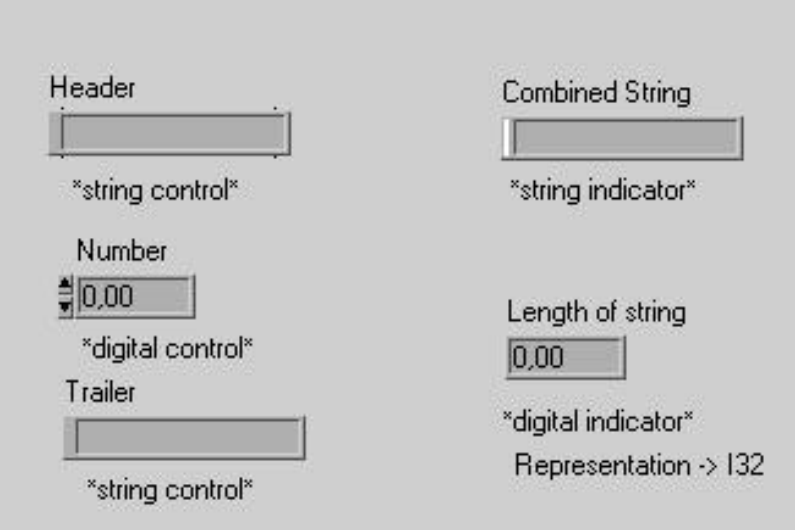
LabVIEW has many functions to manipulate strings. You will find these functions in **Functions»String**. You will build a VI that converts a number to a string and concatenates the string with other strings to form a single output string. The VI also determines the output string length.

6.2.1 Front Panel

1. Open a new front panel and build the front panel shown in the preceding illustration. Be sure to modify the controls and indicators as depicted.

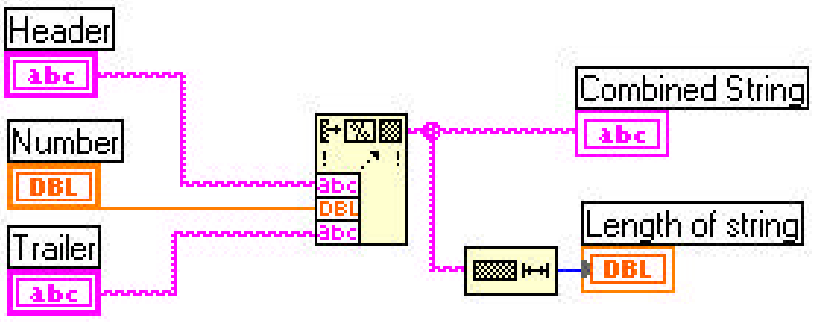
The two string controls and the digital control can be combined into a single output string and displayed in the string indicator. The digital indicator displays the string length.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de



The Combined String output in this exercise has a similar format to command strings used to communicate with GPIB (IEEE 488) and serial (RS-232 or RS-422) instruments.

6.2.2 Block Diagram



1. Build the block diagram shown in the preceding illustration.

Format Into String function **(Functions»String**) concatenates and formats numbers and strings into a single output string. Use the Resizing cursor on the icon to add three argument inputs.

String Length function **(Functions»String**) returns the number of characters in the concatenated string.

2. Run the VI. Notice that the Format Into String function concatenates the two string controls and the digital control into a single, output string.
3. Save the VI as `My Build String.vi`. You will use this VI in the next exercise.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6.3 File I/O

The LabVIEW file I/O functions **(Functions»File I**/O) are a powerful and flexible set of tools for working with files. In addition to reading and writing data, the LabVIEW file I/O functions move and rename files and directories, create spreadsheet-type files of readable ASCII text, and write data in binary form for speed and compactness.

You can store or retrieve data from files in three different formats.

- ASCII Byte Stream. You should store data in ASCII format when you want to access it from another software package, such as a word processing or spreadsheet program. To store data in this manner, you must convert all data to ASCII strings.
- Datalog files. These files are in binary format that only LabVIEW can access. Datalog files are similar to database files because you can store several different data types into one (log) record of a file.
- Binary Byte Stream. These files are the most compact and fastest method of storing data. You must convert the data to binary string format and you must know exactly what data types you are using to save and retrieve the data to and from files.

This section discusses ASCII byte stream files because that is the most common data file format.

6.4 File I/O Functions

Most file I/O operations involve three basic steps: opening an existing file or creating a new file; writing to or reading from the file; and closing the file. Therefore, LabVIEW contains many utility VIs in **Functions»File I**/O. This section describes the nine, high-level utilities. These utility functions are built upon intermediate-level Vis that incorporate error checking and handling with the file I/O functions.

You can also set a delimiter or string of delimiters, such as tabs, commas, and so on, in your spreadsheet. This saves you from parsing your spreadsheet if you used a delimiter other than the default tab to set up the spreadsheet.

The Write Characters To File VI writes a character string to a new byte stream file or appends the string to an existing file. This VI opens or creates the file, writes the data, and then closes the file.

The Read Characters From File VI reads a specified number of characters from a byte stream file beginning at a specified character offset. This VI opens the file beforehand and closes it afterwards.

The Read Lines From File VI reads a specified number of lines from a byte stream file beginning at a specified character offset. This VI opens the file beforehand and closes it afterwards.

The Write To Spreadsheet File VI converts a 1D or 2D array of single-precision numbers to a text string and writes the string to a new byte stream file or appends the string to an existing file. You can optionally transpose the data. This VI opens or creates the file beforehand and closes it afterwards. You can use this VI to create text

files readable by most spreadsheet programs.

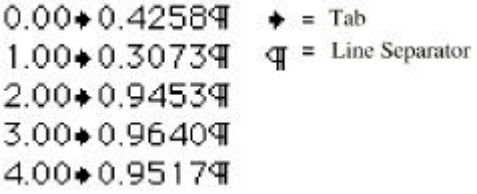
The Read From Spreadsheet File VI reads a specified number of lines or rows from a numeric text file, beginning at a specified character offset, and converts the data to a 2D, single-precision array of numbers. You can optionally transpose the array. This VI opens the

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

file beforehand and closes it afterwards. You can use this VI to read spreadsheet files saved in text format.

6.5 Writing to a Spreadsheet File

One very common application for saving data to a file is to format the text file so that you can open it in a spreadsheet. In most spreadsheets, tabs separate columns and EOL (End of



Line) characters separate rows, as shown in the following figure.

Opening the file using a spreadsheet program yields the following table.

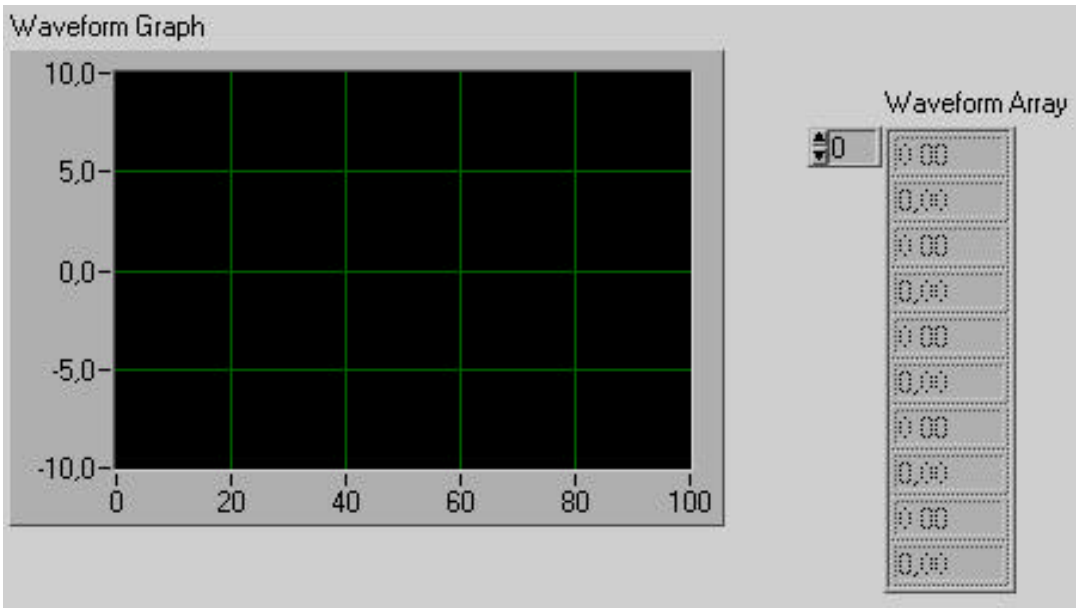
	A	B	C
1	0	0.4258	
2	1	0.3073	
3	2	0.9453	
4	3	0.964	
5	4	0.9517	
6			

You will modify an existing VI to use a file I/O function so that you can save data to a new file in ASCII format. Later you can access this file from a spreadsheet application.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6.5.1 Front Panel

1. Open the `My Graph Waveform Arrays.vi` you built in the *Chapter Arrays, Clusters, and Graphs* of this demonstration guide. As you recall, this VI generates two data arrays and plots them on a graph. You modify this VI to write the two arrays to a file where each column contains a data array.



6.5.2 Block Diagram

2. Open the block diagram of `My Graph Waveform Arrays` and modify the VI by adding the block diagram functions that have been placed inside the oval, as shown in the preceding illustration.

The Write To Spreadsheet File VI **(Functions»File I**/O) converts the two-dimensional array to a spreadsheet string and writes it to a file. If you have not specified a path name, then a file dialog box pops up and prompts you for a file name. The Write To Spreadsheet File writes either a 1-dimensional or 2-dimensional array to file. Because you have a 2D array of data in this example, you do not have to wire to the 1D input. With this VI, you can use a spreadsheet delimiter or string of delimiters, such as tabs or commas in your data.

Boolean Constant **(Functions»Boolea**n) controls whether or not LabVIEW transposes the 2D array before writing it to file. To change the value to TRUE click on the constant with the Operating tool. In this case, you want the data transposed because the data arrays are row specific (each row of the two-dimensional array is a data array). Because each column of the spreadsheet file contains a data array, the 2D array must first be transposed.

3. Return to the front panel and run the VI. After the data arrays have been generated, a file dialog box prompts you for the file name of the new file you are creating. Type in a file name and click on **O**K.

Caution: ***Do not attempt to write data in VI libraries, such as the mywork.llb. Doing so may result in overwriting your library and losing your previous work.***

4. Save the VI, name it `My Waveform Arrays to File.vi`, and close the VI.
5. You now can use spreadsheet software or a text editor to open and view the file you just created. You should see two columns of 100 elements.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

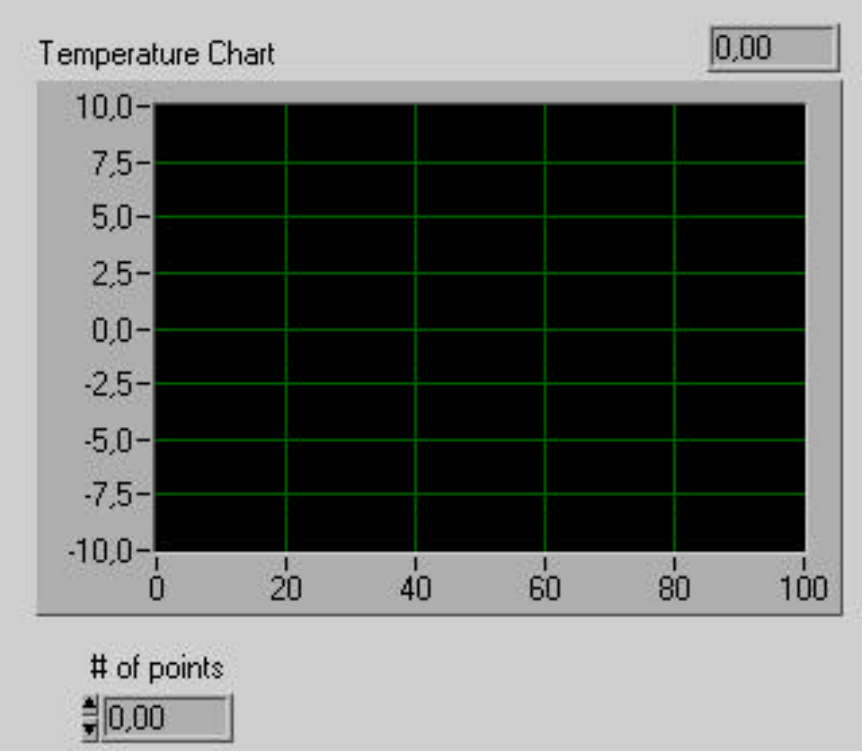
In this example, the data was not converted or written to file until the entire data arrays had been collected. If you are acquiring large buffers of data or would like to write the data values to disk as they are being generated, then you must use a different File I/O VI.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6.6 Appending Data to a File

You will create a VI to append temperature data to a file in ASCII format. This VI uses a For Loop to generate temperature values and store them in a file. During each iteration, you will convert the data to a string, add a comma as a delimiting character, and append the string to a file.

6.6.1 Front Panel



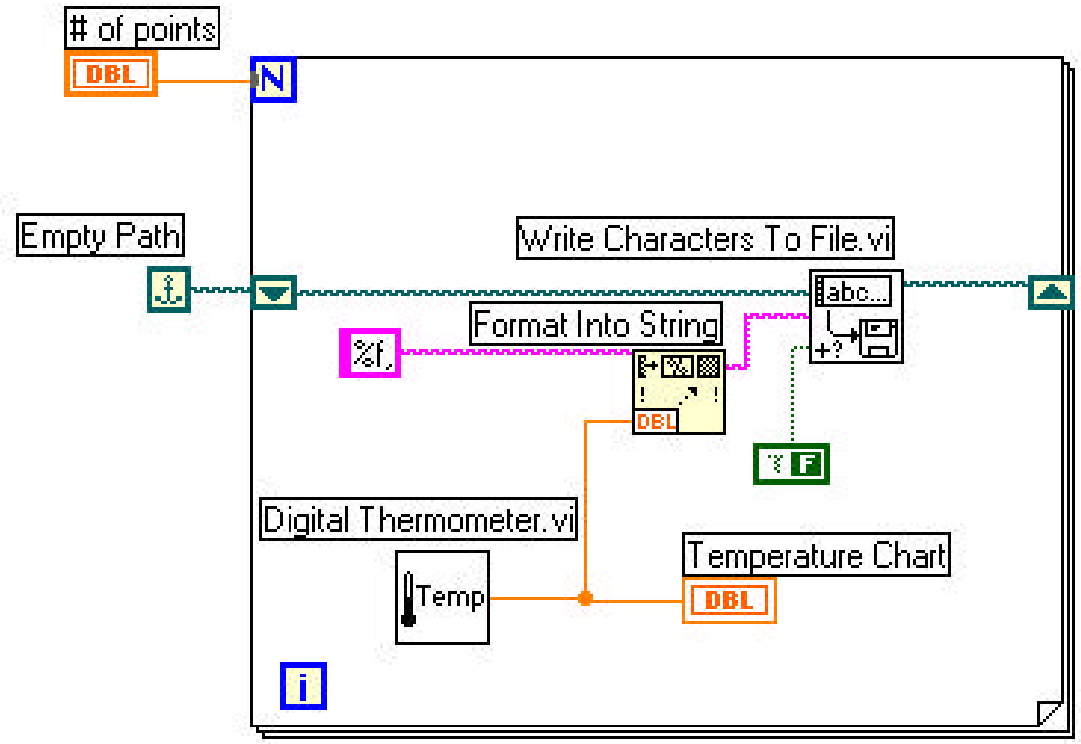
1. Open a new front panel and place the objects as shown in the preceding illustration.

The front panel contains a digital control and a waveform chart. Select **Show»Digital Displa**y. The `# of points` control specifies how many temperature values to acquire and write to file. The chart displays the temperature curve. Rescale the y axis of the chart for the range 70.0 to 90.0, and rescale the x axis for the range 0 to 20.

2. Pop up on the # of points digital control and choose **Representation»Long**.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6.6.2 Block Diagram



1. Open the block diagram.
2. Add the For Loop and enlarge it. This VI generates the number of temperature values specified by the `# of Points` control.
3. Add a Shift Register to the loop by popping up on the loop border. This shift register contains the path name to the file.
4. Finish wiring the objects.

Empty Path constant **(Functions»File I\O»File Constant**s). The Empty Path function initializes the shift register so that the first time you try to write a value to file, the path is empty. A file dialog box prompts you to enter a file name.

The My Thermometer VI you built in Chapter 2 **(Functions»Select a VI...**) or the Digital Thermometer VI **(Functions»Tutoria**l) returns a simulated temperature measurement from a temperature sensor.

Format Into String function **(Functions»String**) converts the temperature measurement (a number) to a string and concatenates the comma that follows it.

String constant **(Functions»String**). This format string specifies that you want to convert a number to a fractional format string and follow the string with a comma.

The Write Characters To File VI **(Functions»File I/O)** writes a string of characters to a file.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Boolean Constant **(Functions»Boolea**n) sets the `append to file?` input of the Write Characters To File VI to True so that the new temperature values are appended to the selected file as the loop iterates. Click the Operating tool on the constant to set its value to True.

5. Return to the front panel and run the VI with the `# of points` set to 20. A file dialog box prompts you for a file name. When you enter a file name, the VI starts writing the temperature values to that file as each point is generated.
6. Save the VI, name it `My Write Temperature to File`.vi, and close the VI.
7. Use any word processing software such as Write for Windows, Teach Text for Macintosh, and Text Editor in Open Windows for UNIX to open that data file and observe the contents. You should get a file containing twenty data values (with a precision of three places after the decimal point) separated by commas.

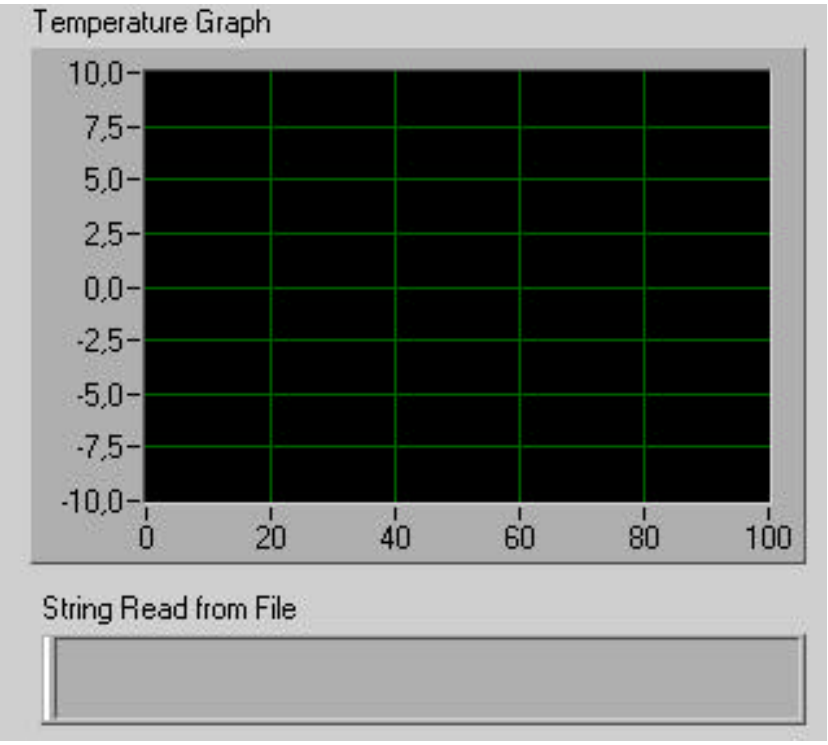
Reading Data from a File

You will create a VI that reads the data file you wrote in the previous example and displays the data on a waveform graph. You must read the data in the same data format in which you saved it. Therefore, since you originally saved the data in ASCII format using string data types, you must read it in as string data with one of the file I/O VIs.

6.6.3 Front Panel

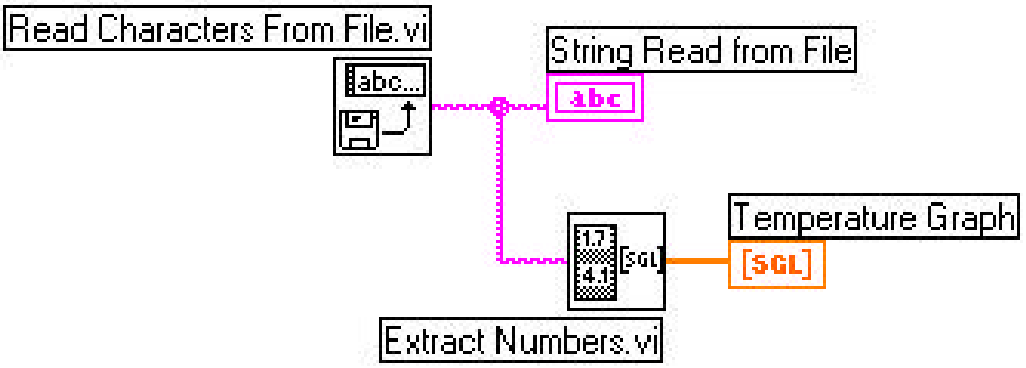
1. Open a new front panel and build the front panel shown in the preceding illustration.

The front panel contains a string indicator and a waveform graph. The String Read from File indicator displays the comma delimited temperature data from the file you wrote in the last example. The graph displays the temperature curve.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

6.6.4 Block Diagram



1. Build the block diagram as shown in the preceding illustration.

The Read Characters From File VI **(Functions»File I**/O) reads the data from the file and outputs the information in a string. If no path name is specified, a file dialog box prompts you to enter a file name. In this example, you do not need to determine the number of characters to read because there are fewer characters in the file than the default (512).

You must know how the data was stored in a file in order to read the data back out. If you know how long a file is, you can use the Read Characters From File VI to determine the known number of characters to read.

The Extract Numbers VI **(Function**s»**Tutoria**l) takes an ASCII string containing numbers separated by commas, line feeds, or other non-numeric characters and converts them to an array of numerics.

2. Return to the front panel and run the VI. Select the data file you just wrote to disk when the file dialog box prompts you. You should see the same data values displayed in the graph as you saw in the My Write Temperature to File VI example.

Save the VI, name it `My Temperature from File.vi`, and close the VI.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

How to Call Win32 Dynamic Link Libraries (DLLs) from LabVIEW

Ralf Engels
Forschungszentrum Jülich GmbH
Zentrallabor für Elektronik
D-52425 Jülich

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Contents

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

INTRODUCTION

LabVIEW is a graphical programming language rich in data acquisition, data analysis, and data presentation capabilities. You assemble software components using the LabVIEW innovative graphical programming environment to create virtual instruments to meet your application needs. LabVIEW includes VIs to acquire data from plug-in data acquisition boards, programmable instruments, and other applications. It also includes VIs that analyze data and present results through graphical user interfaces. In most cases, the VIs included in the LabVIEW Development System meet the needs of users.

However, LabVIEW programmers can access Dynamic Link Libraries (DLLs) through the Call Library Function. DLLs are extremely powerful tools, because you can use them to share code among many applications. By using LabVIEW to access DLLs, you gain access to functions available in third-party libraries, including existing DLLs you or your colleagues may have written.

This application note discusses how to use the Call Library Function in LabVIEW 4.0 to access Win 32 DLLs, giving you access to numerous functions available in the Win32 Applications Programming Interface (API) for increasing the functionality of LabVIEW applications. Win32 is a 32-bit API provided in Windows 95 and Windows NT. This API has numerous changes from the Win16 (16-bit API in Windows 3.1 and Windows for Workgroups). Most functions contained in the Win16 API have equivalent functions in the Win32 API with their parameters changed from 16 to 32-bits.

The Call Library Function in LabVIEW 4.0 can also be used for direct access to shared libraries on Unix operating systems and on Mac OS.

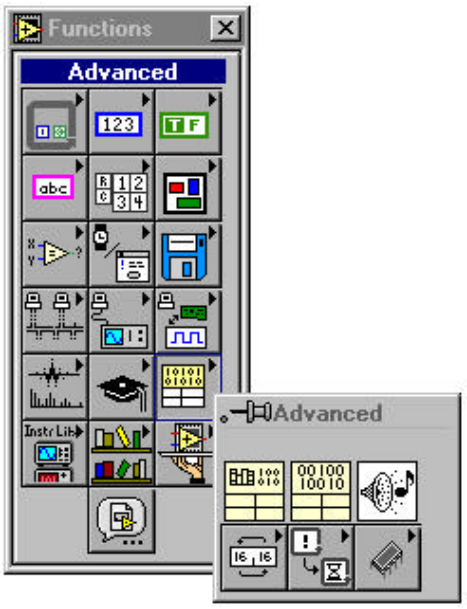
Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

THE CALL LIBRARY FUNCTION

LabVIEW 4.0 features the **Call Library Function** node to offer easy access to your dynamic link libraries.

Some of the important features of the **Call Library Function** in LabVIEW for Windows 95/NT are:

- You can call DLLs that use either the C or the Default(stdcall) calling convention.
- You can pass integer and floating point arrays of arbitrary dimensions.
- You do not have to be concerned about `HUGE`, `NEAR`, or `FAR` pointer type s.
- LabVIEW strings can be passed as C or Pascal string pointers, or as a LabVIEW string handle, depending on the DLL being called.



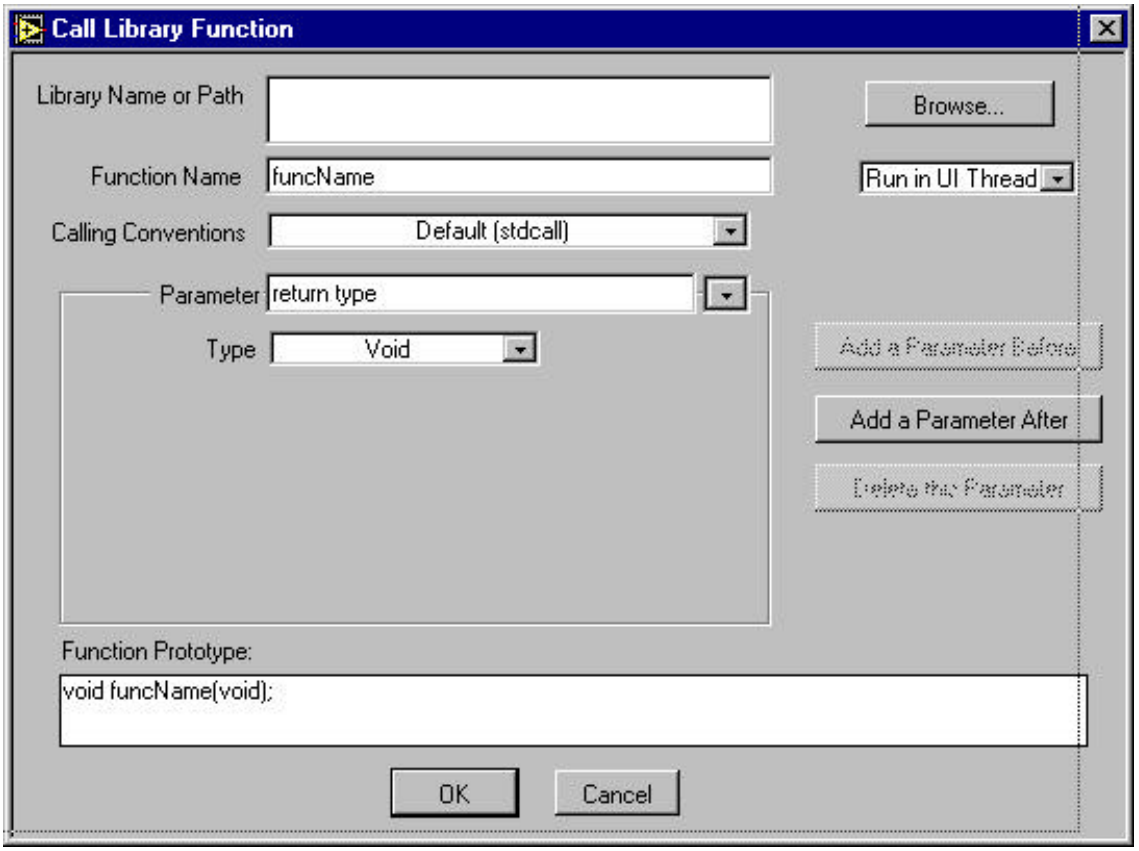
- You can use `void`, `integer`, and `floating point` return types.

The **Call Library Function** icon is located in the **Advanced** subpa lette of the **Functions** palette.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

To configure the **Call Library Function** to call a specific function within a DLL, pop up on the icon and select the **Configure...** option, as shown.

You use the following configuration window to specify the DLL, the specific DLL function to call, and the function parameters .



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

EXAMPLE – CALLING A FUNCTION IN USER32.DLL

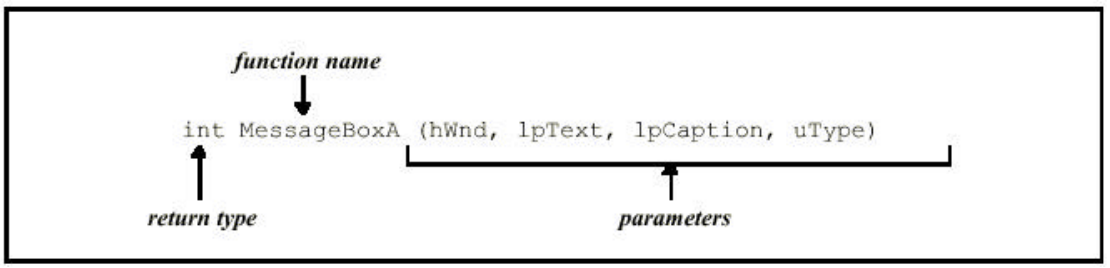
3.1 What Information do You Need?

This example of using LabVIEW 5.1 to call a DLL will involve making a function call to one of the standard DLLs that is part of the Windows 95/NT operating system. You will learn how to configure the **Call Library Function** to call `USER32.DLL,` which resides in the `\WINDOWS\SYSTEM` directory. In `USER32.DLL,` you will call the `MessageBoxA` function to create a three-button dialog box with "Yes", "No", and "Cancel" buttons. Note that you can easily create this type of dialog box in LabVIEW without the use of a DLL, but the DLL is already written for you and using it reduces the development time. This example is used to introduce you to the concepts of the **Call Library Function,** and to demonstrate the use of the Win32 API.

When you call a function in a DLL, you need to know the following information, almost all of which can be obtained from the appropriate Win32 include file (`windows.h, winuser.h`, and so on):

- The data type returned by the function; you can use LabVIEW to call functions that return `void, integer`, *or* `floating point` data types (signed or unsigned 8, 16, and 32-bit integers, or 32 and 64- bit floating point data types).
- The calling convention used; both C and Default(stdcall) conventions are available. The Win32 API uses the Default(stdcall) convention whereas most user written DLLs use the C convention.
- The parameters to be sent to the function, their types, and the order in which they must be passed.
- The location of the DLL on your computer.

To find this information for the `MessageBoxA` function, consult a Windows programming manual that covers the Win32 API. If you have installed a 32-bit Windows compiler such as Borland C++ or Microsoft Visual C++ then you will also have access to the Windows "include" files such as `windows.h, windowsx.h,` and the `winuser.h.` You will find your compiler documentation and the `.h` include files to be invaluable resources in locating information about the Win32 DLL functions (other useful tool s for viewing export functions in a DLL are `QUICKVIEW`, provided with Windows 95, and



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

DUMPBIN, provided with Visual C++). The description of the MessageBoxA function from winuser.h supplies us with the information we need to call the function:

return type

The return type for the function is defined as a 32-bit signed integer:

int 32 bit signed integer

The Win32 API lists the names of the constants for the possible return values for the MessageBoxA function. The actual values of these constants are stored in the winuser.h file. In this example, the possible return values are IDYES, IDNO, and IDCANCEL, which have the decimal values 6, 7, and 2, respectively. If the message box cannot be created due to a lack of memory, zero will be returned.

parameters

The *Microsoft Win32 Programmer's Reference* lists the data types of each of the parameters to the MessageBoxA function; the actual type definitions are all found in the *winuser.h* file.

HWND hWnd Identifies the owner or parent window of the message box to be created. If this parameter is NULL, the message box has no owner window. The HWND data type is a 32 -bit unsigned integer as defined in winuser.h and windows.h.. Es sentially, we can identify which window the message box "belongs to" by passing a valid value for hWnd. However, it is not necessary to define a parent for this window, so we will assign "no parent", or NULL. The constant NULL is defined to be zero.

LPCSTR lpText The LP CSTR type is a 32-bit pointer to a constant character string and is defined as a C-style (NULL terminated) string . This string contains the text we wish to display in the window.

LPCSTR lpCaption This parameter is a C-style constant character string containing the desired name to appear in the title bar of the window.

UINT uType The UINT data type is defined as an unsigned 32-bit integer value. It determines which type of message box is displayed. The Windows API lists the names of valid constants that can be passed to this function, and winuser.h will contain the actual decimal values. In this example, we will create a dialog box with "Yes", "No", and "Cancel" buttons. The name of the constant is MB_YESNOCANCEL, which is defined to have the value **3** in winuser.h. We will pass this value for the uType parameter. The other types of message boxes and their corresponding uType are:

message box button type	uType
OK	**0**
OK CANCEL	**1**
ABORT RETRY IGNORE	**2**
YES NO CANCEL	**3**
YES NO	**4**
RETRY CANCEL	**5**

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

Warning: Do not use different values for `uType` than those listed in the Win32 API or `winuser.h`. You could cause errors in Windows 95/NT which may result in a crash or incorrect behavior! calling convention

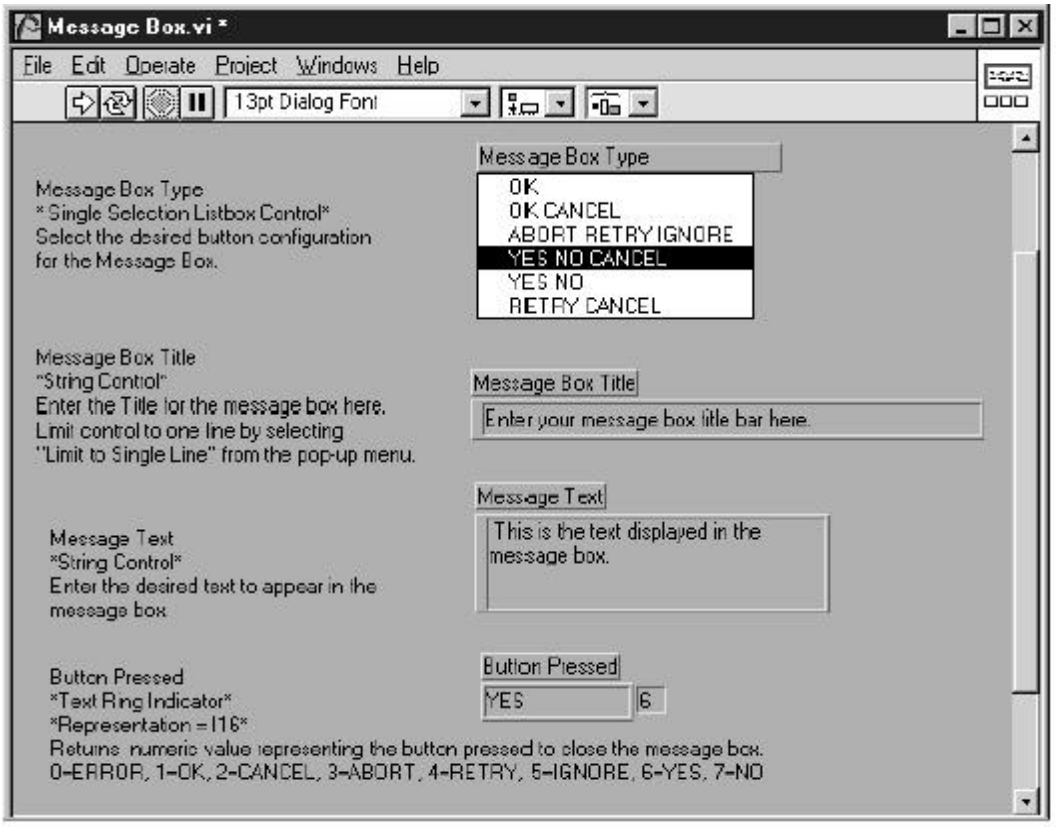
The calling convention for the `MessageBoxA` function can be found in the *winuser.h* file. Searching in `winuser.h` for `MessageBox`A, we find that the function is preceded by the word `WINAPI.` This is defined as `__stdcall` in `windef.h.`

The Default("standard C" or " __stdcall") calling convention is used to call Win32 API functions. P arameters are passed by a function onto the stack from right to left, and are passed by value unless a pointer or reference type is passed. Function arguments are fixed, and a function prototype is required. Functions using this calling convention return values the same way as functions using the C calling convention. The C calling convention is the default calling convention for C and C++ programs. Arguments are passed from right to left; however, a called function pops its own arguments from the stack. Because the stack is cleaned up by the caller, it can have variable argument functions.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

HOW TO CALL THE MESSAGEBOX FUNCTION IN THE USER32.DLL

First, create the following front panel. Note that the comments on the front panel, which are there to explain the controls and indicators, are not necessary.



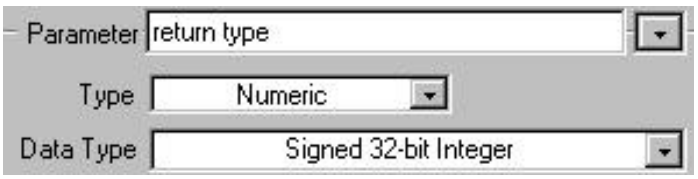
On the block diagram, create the **Call Library Functio**n. To view the on-line help for this function, select **Show Help** from the **Help** menu and move the cursor over the function icon. You can also obtain more information about this function either by selecting **Online Help** from the pop-up menu, or by reading Chapter 24 of the *LabVIEW User Manual for Windows.*

Note that at this point, only one set of terminals appears on the function icon, and they are grayed out. After you configure the **Call Library Function** for the DLL function you wish to use, the appropriate terminals will be available on the icon. Pop up on the **Call Library Function** icon, and select **Configure...** from the pop-up menu. Complete instructions are listed below:

1. Type `USER32.DLL` in the **Library Name or Path** box. You will not need to type in the entire path to the DLL unless the DLL is stored in a location that does not appear in the `PATH` statement in your `AUTOEXEC.BAT` file or the LabVIEW VI Search Path. If you press <enter> on the keyboard, the configuration window will close. You can re-open it by selecting **Configure...** from the pop-up menu of the **Call Library Function** icon or by double-clicking on it.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

2. Next, click in the **Function Name** field, and type the name of the function:



`MessageBoxA`. Function names in general are case sensitive.

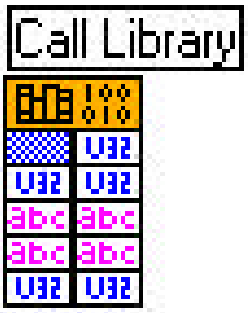
3. You do not need to change the value in the **Calling Conventions** box.
4. At this point, you need to indicate what kind of data the `MessageBoxA` function will return to LabVIEW when it has finished. We know the return value of the function is a 32-bit integer indicating which button was pressed in the dialog box. To set this, we use the **Parameter** and **Type** fields. Observe that the **Parameter** field contains the text "`return type`", and below this, you see that the **Type** is set to `Void`.The `MessageBoxA` function returns a signed 32-bit integer value, so select this data type for the Return value. To do this, click on the selection box next to the **Type** field and select `Numeric` from the pop-up list. You may also change the name of the return type from "`return type`" to something more descriptive, for example "`button pressed`".
5. After setting the return type to `Numeric`, you will see a new field appear, called **Data Type**. The default is `Signed 32-bit Integer`.
6. In addition to defining the return type of the function, we must also define the four arguments to be passed to the function. The first argument of the `MessageBoxA` function is the `hWnd` parameter, which we know to be an unsigned 32-bit integer. Click on the **Add Parameter After** button to add the first parameter. Then, select `Unsigned 32-bit Integer` from the **Data Type** menu. Because the function expects the value, and not a pointer to the value, leave the **Pass** setting unchanged. If you like, you can change the name of the parameter from `arg1` to something more descriptive, such as `hWnd`.
7. From the definition of the `MessageBoxA` function, the second and third arguments of the function are pointers to C-style strings. To add a string to the parameter list as the second argument, first make sure that the first argument appears in the **Parameter** box. You can select an argument by using the selector to the right of the box containing the parameter name. Click the **Add a Parameter After** button. To set the **Type** of the data to a pointer to a string, select `String` from the **Type** menu. When you send a string to a function, you can select whether the pointer to the string points to a C-style (string followed by a `NULL` character), Pascal-style (string preceded by a length byte) string, or as a LabVIEW string handle (four bytes of length information followed by the string data). In this example, the default setting (CÊString Pointer) is correct For more information about pointers and strings, see the Array and String Options section of this document.
8. The third argument passed to this function is another string, which contains the title of the message box window. Setting up this argument is the same as the previous argument.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

9. Finally, you must add the `uType` parameter, which is an `Unsigned 32-bit Integer`. This is the value that determines which type of message box is displayed.

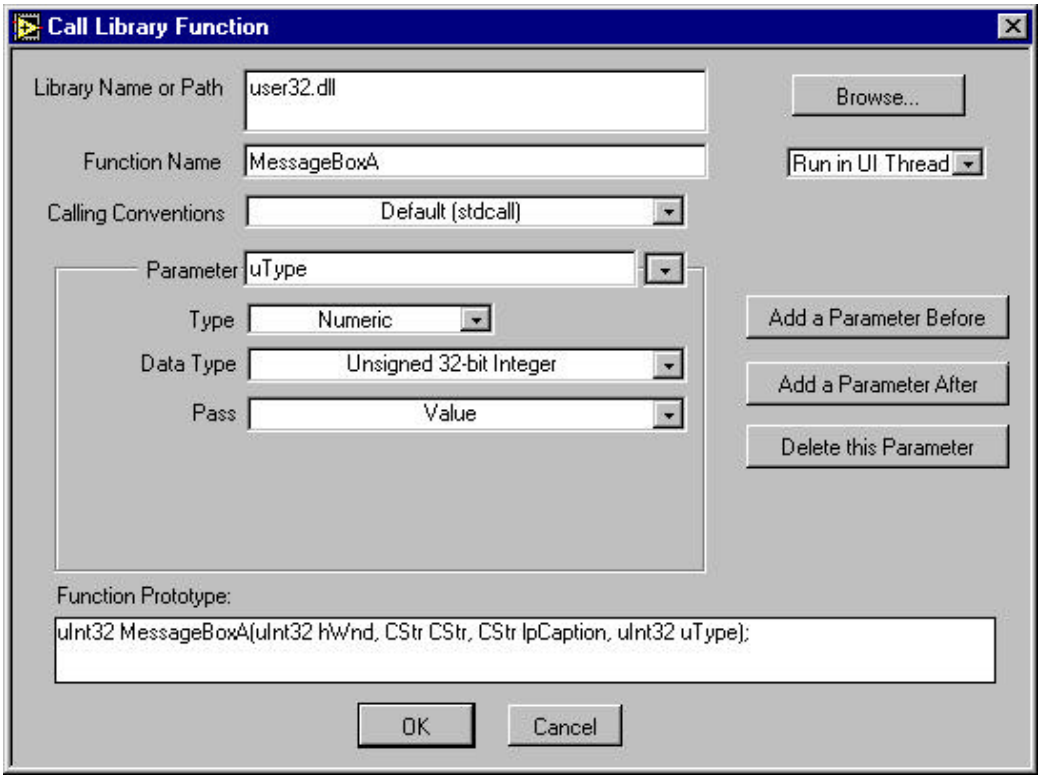
When you have finished configuring the **Call Library Functio**n, you can double-check if your configuration is correct by comparing the **Function Prototype** displayed in the configuration window to that obtained from the documentation of the function. This will help you to be certain that you are passing the correct data types to the function. Note that LabVIEW uses descriptive names for data types. For example, the int32 data type describes a 32-bit signed integer in LabVIEW. In most compilers, this data type is described as int .

Check to see that you have completed the dialog box correctly by studying the figure above. Click the **OK** button to close the configuration window. Notice how terminals have been added to the icon, and the parameters of the function listed from left to right in the function prototype match the data types appearing on the terminals of the icon from top to bottom. The upper left input terminal is disabled because the top output terminal is the return value of the function, not an argument to the function.



Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

To complete the VI, build the following diagram. Remember to make sure to set the



representation of the numeric constants you connect to the **Call Library Function** icon to the correct type.

Note: All input terminals to the Call Library Function must receive data!

Once you have finished constructing the diagram, save your program and run it.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

ADDITIONAL EXAMPLES

1. If you have a sound card with Windows sound drivers installed on your system, you can also investigate Play Sound.vi found in the LabVIEW Examples directory:

   `\LABVIEW\EXAMPLES\DLL\SOUND\PLAYSND.LLB\Play Sound.vi`

   You can use this VI to play Windows `.WAV` sound files on your computer from LabVIEW.

2. If you do not have a sound card you can "beep" the PC by calling the MessageBeep function in User32.DLL.

   The function prototype is:

   ```
   VOID MessageBeep(UINT uType);
   ```

   This function will generate a short tone through your PC speaker.

3. A good example for using LabVIEW string handles can also be found in the LabVIEW Examples directory:

   `\LABVIEW\EXAMPLES\DLL\HOSTNAME\hostname.vi`

   This example will return the host name of your computer.

4. You can programmatically position your cursor anywhere on your monitor using the SetCursorPos function in User32.DLL. The function prototype is:

   ```
   BOOL SetCursorPos( INT x, INT y);
   ```

   x and y are the desired coordinates referenced from the upper left corner of the screen. The return value is TRUE if the function was successful and FALSE if it was not. (The value returned is type BOOL, which is defined in the Win32 API as a 32-bit signed integer with values 0=FALSE and 1=TRUE.)

ARRAY AND STRING OPTIONS

This section briefly reviews some important concepts you should be familiar with when using the **Call Library Function** to work with array and string data

Arrays of Numeric Data

Arrays of numeric data can be any type of integers, or floating point numbers with single (4-byte) or double (8-byte) precision. When you pass an `Array Data Pointer`, you can also set the number of dimensions in the array, but you do *not* include information about the size of the array dimension(s). You will have to pass this information to your DLL in a separate variable unless you are using LabVIEW Array Handles . Because the Win32 API does not use LabVIEW array handles, the function definition of the specific API function you are calling will specify which array parameters are required.

String Data

The **Call Library Function** passes C or Pascal-style string pointers, or LabVIEW string handles. You must select the same type of string pointer as that used in your function, or errors will occur. The C-style string consists of the string followed by a NULL character. The Pascal-style string consists of the string preceded by a length byte. The LabVIEW string handle consists of 4-bytes of length information followed by the string data. Most Win32 API functions use the C-style string pointer.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

IMPORTANT REMINDERS AND QUICK REFERENCE

- Make sure that the path to the DLL file is correct.
- If you get the error message `Function not found in library`, check the spelling, syntax, and case sensitivity of the function name you wish to call.
- Make sure that all the parameters passed to a DLL function have data wired to all of the input terminals of the **Call Library Function** icon. Be sure to configure the function properly for *all* input parameters.
- Know the return types and data types of arguments for your functions and remember to configure the **Call Library Function** to *exactly* match the data types your function uses. Failure to do so may result in crashes.
- Make sure you use the proper calling convention (C or `Default(stdcall)`). The Win32 API uses the `Default(stdcall)` convention.
- Know the correct order of the arguments passed to the function.
- When passing strings to a function, remember to select the correct type of string to pass – C or Pascal string pointers, or LabVIEW string handle. The Win32 API uses the C-style string pointer.
- If you are working with arrays or strings of data, ***ALWAYS*** pass a buffer or array that is large enough to hold any results placed in the buffer by the function.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de

The content of the script is mainly based on the publications noted in the Reference list, whereas some passages were reproduced originally.

REFERENCES

- National Instruments; Application Notes
- National Instruments; LabVIEW User Books
- LabVIEW Graphical Programming; Gary W.Johnson; Mc.Graw Hill-Inc.

Ralf Engels **Forschungszentrum Jülich, ZEL** **52425 Jülich, Germany**
Tel: +49-(0)2461-612878 Fax: +49-(0)2461-61399 Email: R.Engels@FZ-Juelich.de