Learn LabVIEW™ 2013/2014 Fast

A Primer for Automatic Data Acquisition

Douglas Stamps, Ph.D.

SDC PUBLICATIONS

Better Textbooks. Lower Prices.
www.SDCpublications.com

Chapter 1 LabVIEW for Data Acquisition

1.1 What is Automatic Data Acquisition?

In a broad sense, data acquisition (DAQ) is the measurement or generation (control) of a physical phenomenon. It may be performed manually by a person or automatically by a computer. There are numerous reasons to automate your data measurement and generation.

Automatic data acquisition has advantages over manual data acquisition when changes occur either very quickly or very slowly in the physical phenomenon or when there are many inputs and/or outputs associated with the phenomenon. For example, a person would not be able to record the pressure inside an airbag during deployment or the strain in the structural members in the frame of the car during an automotive test crash because the event would occur too quickly to record any data. Likewise, events that occur over a long period of time, like measuring meteorological data, are better performed automatically with a computer. Even if the event occurs at a manageable pace and period of time for a human, a person would be limited in the number of switches or relays that could be controlled or the number of sensors from which data could be recorded simultaneously. Automatic data acquisition also has the advantage over manual acquisition in that the data can be recorded without the possibility of human error and be viewed as it is collected, which permits experimental methods to be corrected if problems arise before the experiment is completed. Otherwise, with manual data acquisition, the experiment is typically completed before the data is entered and plotted in a spreadsheet.

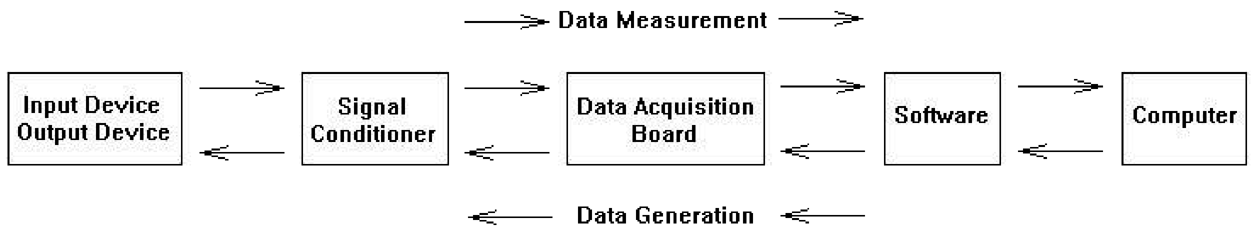


Figure 1.1.1 Components of a data acquisition system

A data acquisition system is composed of several components as shown in Figure 1.1.1, which may take on different configurations depending on the type of system used, but perform similar functions regardless of the system. A typical data acquisition system would have an input or output device; a signal conditioner, which, for the purposes of this introduction, will be broadly interpreted as a device that alters, modifies, or manipulates a signal; a data acquisition (DAQ) board that can convert analog to digital signals or vice versa; a computer; and software to allow the computer to communicate with the DAQ board. The specific components used depend on the type of data and the flow of data, that is, if data is measured or generated.

For data measurement (data direction defined by the arrows pointing towards the computer in Fig. 1.1.1), the input device could be a sensor or transducer, like a thermocouple or pressure transducer, that is detecting a physical phenomenon, like temperature or pressure, and outputting an electrical signal, such as a voltage or current. The signal may pass through a signal conditioner, which is a device that may, among other things, attenuate, amplify, filter, or linearize a signal. Signal conditioners are needed, for example, when the sensor's output is outside of an acceptable range of the other DAQ hardware, the signal has too much electrical noise relative to the output associated with the physical phenomenon, or it is convenient to have the signal in a linear form for unit conversion. The signal is then read by a DAQ board and converted into a digital signal that can be interpreted by a computer. Sometimes DAQ boards are called A/D boards since they are often used to measure an analog input signal and convert it into a digital signal. Software is required for the computer to communicate with the DAQ board. LabVIEW software interfaces with the computer to analyze, store, and display data and with driver software, called NI-DAQmx, which configures data channels and measures data.

Data generation (data direction defined by the arrows pointing away from the computer) operates in the opposite direction as to data measurement. In this case, data is typically generated by LabVIEW to control an output device, such as a switch or motor. LabVIEW works with the NI-DAQmx software to configure the DAQ board to generate the proper electrical signal based on the data generated by LabVIEW. If the output device requires an analog signal, the DAQ board must convert the digital signals used by the computer into an analog signal required by the output device, that is, a digital to analog conversion. In some cases, the DAQ board can provide sufficient current within an acceptable voltage range to drive the output device. However, this is normally not the case as the DAQ boards are typically limited to a voltage range of ±10V and a few milliamps of current. To prevent damage to the DAQ board, another device, such as a solid state relay or an integrated circuit hardware driver, must be connected between the DAQ board and the output device to condition the signal that drives the output device. Loosely speaking, it is analogous to the signal conditioner used during data measurement. Examples of output devices used in example programs found in this primer include stepper motors and DC motors.

Components of a data acquisition system are shown in Figure 1.1.2 for a desktop PC. LabVIEW and driver software are installed on the computer. A data acquisition board is fitted into a PCI slot in the computer. However, the pins on the data acquisition board are too small to make direct connections with sensors or signal conditioners. A terminal connector block is attached by cable to the data acquisition board to provide room to connect the signal conditioners or the input/output devices. A close up of a terminal connector block connected to a data acquisition board is shown in Fig. 1.1.3. The connector block has screw slots for all channels on the data acquisition board and system grounds.

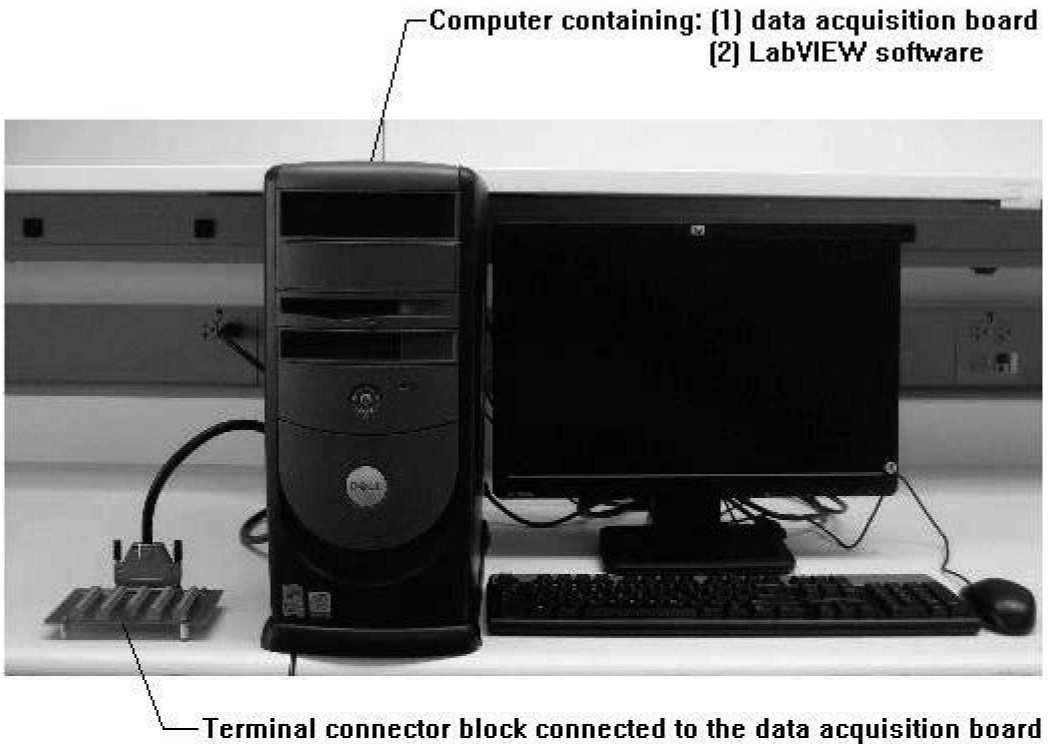

Computer containing: (1) data acquisition board
(2) LabVIEW software
Terminal connector block connected to the data acquisition board

Figure 1.1.2 Components of a data acquisition system

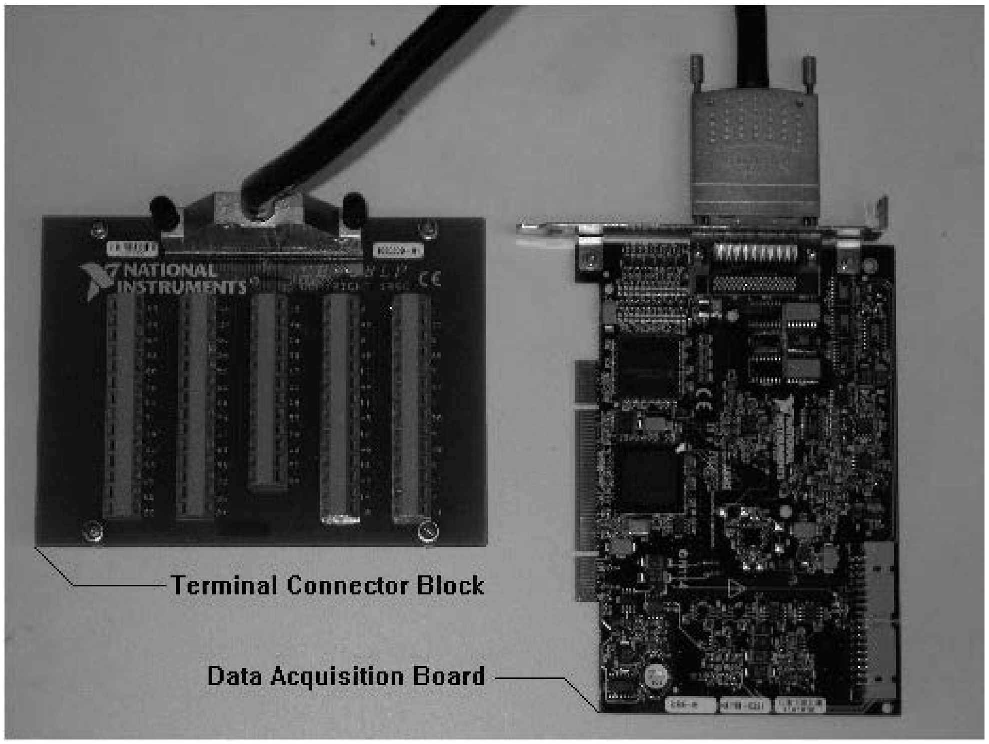


Figure 1.1.3 Terminal connector block connected to a data acquisition board

1.2 What is LabVIEW?

LabVIEW is a software development environment created by National Instruments that allows your computer to interface with data acquisition hardware with appropriate hardware drivers. Early in its development, a focus of LabVIEW was to allow the user to develop virtual instruments to acquire and display data without the aid of an equivalent hardware instrument, like an oscilloscope or a multimeter. The software's name, LabVIEW (**Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench), and the name given to programs developed by LabVIEW, which are called Virtual Instruments (VI) and have a ".vi" file extension, are a result of this early focus of the software. LabVIEW programs are still typically referred to as VIs even though the purpose of the program may not relate to a virtual instrument.

LabVIEW has traditionally been used for automated measurement and control of hardware and processes, which continue to be important application areas. LabVIEW software permits the quick development of VIs to measure, process, analyze, display, and store data as well as to generate data (voltages or currents) to control instruments and hardware, such as motors, valves, or switches. Continued development of the LabVIEW software by National Instruments has allowed it to become a general programming tool, permitting the development of algorithms, mathematical analyses, and communication tasks that extend beyond data acquisition.

LabVIEW is software built on a graphical programming language, known as G code, and the concept of data flow to control program execution. The G programming language is represented by function icons connected by virtual wires, which permit data to flow between the function icons. Functions can have any number of input and output terminals. A function does not execute until data arrives at all input terminals. The execution of the program, therefore, is controlled by the flow of the data. This is conceptually different from text-based programming languages where the execution of the program is determined primarily by the order of the program statements.

1.3 The LabVIEW Environment

The purpose of this section is to introduce you to the LabVIEW environment, which includes the main LabVIEW windows, called the front panel and block diagram, menus and shortcuts that provide options for working with LabVIEW VIs, toolbars to manage objects within the LabVIEW windows, and the palettes that contain the objects that will be used to develop your programs and the user interfaces of the programs. A detailed discussion of all of the features will not be provided in this section as it is difficult to remember the details without actually applying them. However, the hope is that you will become aware of features that exist and where they are located so as to be familiar with them when they are discussed in more detail throughout the examples. So launch LabVIEW and explore the LabVIEW environment as you read this section.

1.3.1 Starting LabVIEW

If your version of LabVIEW was installed using the default installation procedure, launch LabVIEW by selecting **All Programs>>National Instruments>>LabVIEW 2013 (or LabVIEW 2014)>>LabVIEW 2013 (or LabVIEW 2014)** from the Start menu. Or, if available, double click on the LabVIEW shortcut icon on the desktop. A LabVIEW window will appear as shown in Figure 1.3.1. Click on "Create Project" and select "Blank VI" to open a new file for this exercise. Later, you may select a previously used file in the panel located under "Open Existing" or click on "Open Existing" to browse for an existing file not shown.

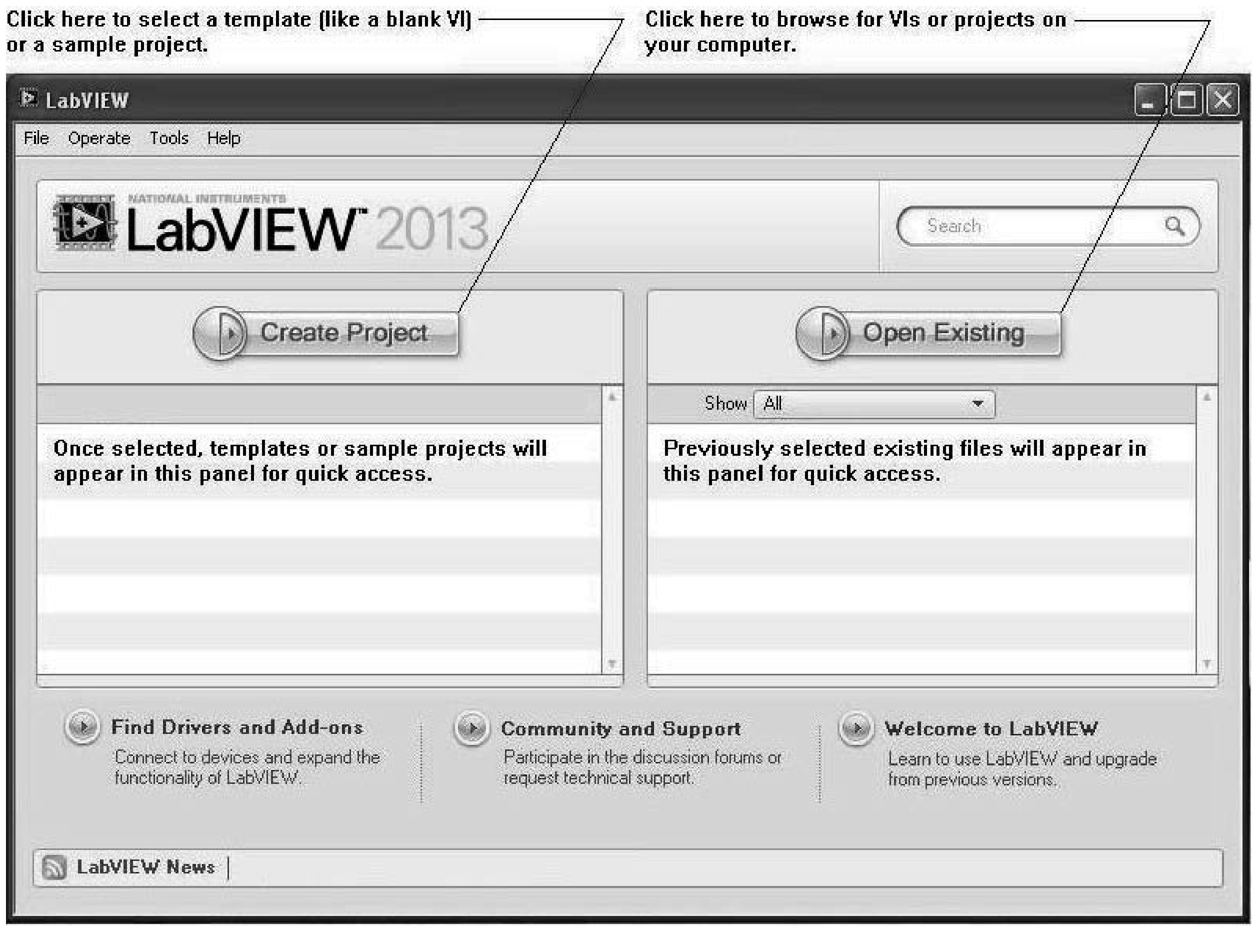

Click here to select a template (like a blank VI) or a sample project.
Click here to browse for VIs or projects on your computer.
LabVIEW
File Operate Tools Help
NATIONAL INSTRUMENTS LabVIEW 2013
Search
Create Project
Open Existing
Show All
Once selected, templates or sample projects will appear in this panel for quick access.
Previously selected existing files will appear in this panel for quick access.
Find Drivers and Add-ons
Connect to devices and expand the functionality of LabVIEW.
Community and Support
Participate in the discussion forums or request technical support.
Welcome to LabVIEW
Learn to use LabVIEW and upgrade from previous versions.
LabVIEW News

Figure 1.3.1 LabVIEW startup window.

1.3.2 Front Panel and Block Diagram

Your first view will be two overlapping windows: the gridded front panel on top and the plain block diagram beneath it.

The front panel window displays controls (user input) and indicators (data output). It can be configured to appear like the instrument panel on measurement equipment. For example, you can place virtual knobs and switches (controls) and display charts, graphs, or virtual LEDs (indicators) in this panel. It is the graphical user interface. This panel provides input to the block diagram through the controls and displays the output of the block diagram through the indicators. The controls and indicators also appear as terminals in the block diagram. Terminals pass data between the front panel and the block diagram, either from the controls or to the indicators. This is the only panel you need to see when your VI is running.

The block diagram window contains the source code of the program and displays the interconnected objects of the graphical programming language. This is the panel where the program is developed and debugged. The block diagram consists of nodes, which are objects that have inputs and outputs and perform some type of operation when the VI executes. Nodes comprise functions, subVIs, Express VIs, and structures. Functions are built-in elements that perform specific operations, subVIs are self-contained sections of code like subroutines in text-based programming languages, Express VIs are configurable VIs, and structures control VI execution. Wires transfer data between the nodes in the block diagram. Wires take on different color, thickness, and texture depending on the type of data they carry.

1.3.3 Pull-Down Menus and Keyboard Shortcuts

There are a number of pull-down menus to help you manage your files, edit your program, manage the LabVIEW windows, and get help, among other things. The pull-down menus are located just under the title of each window. Specific pull-down menu options will be introduced as needed throughout the primer although this is a good time to browse the options to see what is available on each menu. Notice that some commands and options have keyboard shortcuts. For example, if you select the “Edit” pulldown menu, you will notice that <Ctrl+B> is a shortcut for “Remove Broken Wires.” If you find yourself using an option from one of the pull-down menus regularly, such as removing broken wires, you can reduce program development time using the keyboard shortcuts listed to the right of the command or option.

You may find it easier to work with the LabVIEW examples in this primer if you click on “Window” in the menu bar and then select “Tile Up and Down” from the pull down menu as shown in Figure 1.3.2. LabVIEW will place the front panel window on the top half of the screen and the block diagram window on the bottom half.

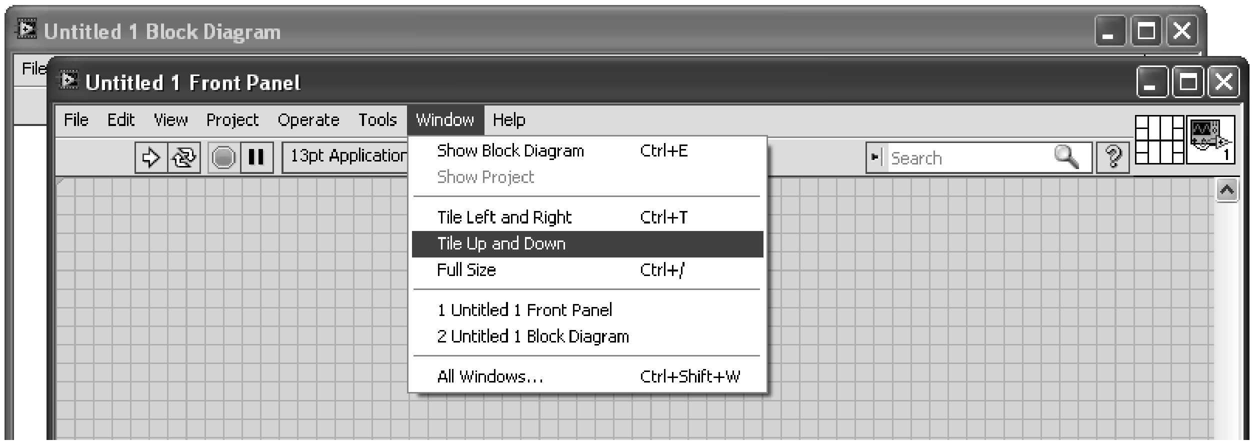

Untitled 1 Block Diagram
Untitled 1 Front Panel
File Edit View Project Operate Tools Window Help
13pt Application
Search
Show Block Diagram Ctrl+E
Show Project
Tile Left and Right Ctrl+T
Tile Up and Down
Full Size Ctrl+/
1 Untitled 1 Front Panel
2 Untitled 1 Block Diagram
All Windows... Ctrl+Shift+W

Figure 1.3.2 Untitled VI showing the “Window” pull-down menu options.

1.3.4 Toolbars

The front panel and the block diagram have toolbars that contain commands, some of which are in the pull-down menus, that manipulate the objects to provide good housekeeping, provide means to debug the VI, get information on VI objects, and run the VI.

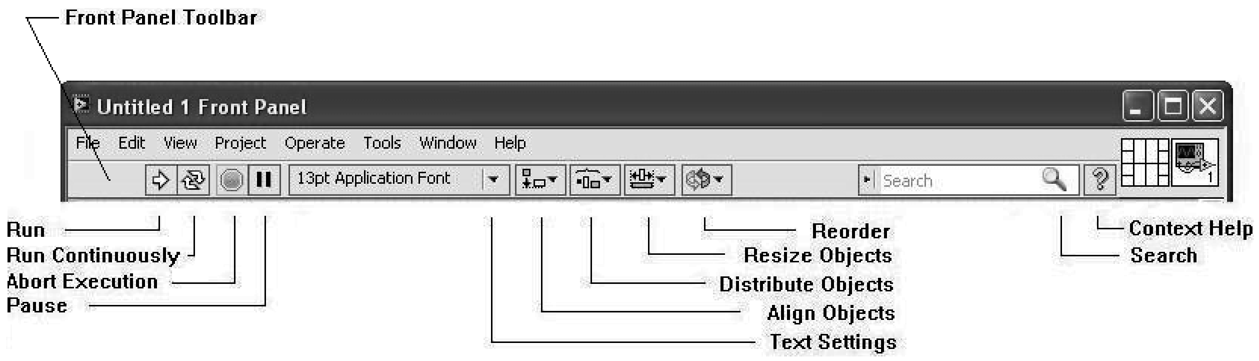


Figure 1.3.3 Front panel toolbar

The front panel toolbar is shown in Fig. 1.3.3. The four buttons on the left of the toolbar control program execution. The “Run” button takes on different appearances depending on the status of the VI, as shown by the top four buttons in Fig. 1.3.4. The other buttons control the execution of the program. When selected, the “Run Continuously” button runs the VI over and over again until you abort or pause the execution. For example, one could write a program to take one data measurement and then select the “Run Continuously” button to take multiple measurements. However, there are LabVIEW structures, such as the While Loop, that perform this type of task more efficiently. The “Run Continuously” button should not be a substitute for this type of programming. Likewise, the “Abort

Execution" button stops the VI but a better practice is to include a stop control in the front panel. The next five buttons in Fig 1.3.3 provide options for general housekeeping for front panel objects and labeling. A VI is easier to operate if the controls and indicators in the front panel are aligned and grouped according to common functions. The last two buttons provide information on LabVIEW objects.

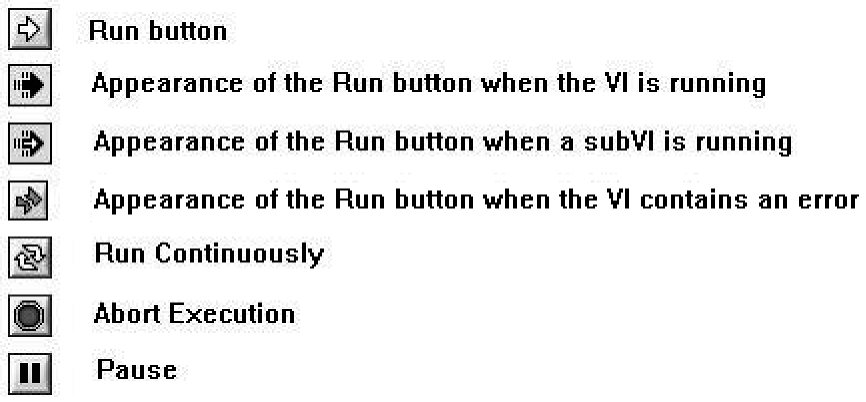

Run button
Appearance of the Run button when the VI is running
Appearance of the Run button when a subVI is running
Appearance of the Run button when the VI contains an error
Run Continuously
Abort Execution
Pause

Figure 1.3.4 Buttons that control the execution of the VI or show its status

Many of the buttons in the block diagram toolbar, shown in Fig. 1.3.5, perform the same functions as described for the front panel toolbar. However, the block diagram toolbar also contains buttons to help debug the VI. The "Highlight Execution" button and the "Step" buttons animate the data flow, provide step-by-step control over program execution, and provide the means to see data values at each node. The middle set of buttons on the toolbar provides options for general housekeeping. A neat and organized set of objects in the block diagram, especially when it pertains to new LabVIEW users, reduces the chances for wiring errors and makes debugging easier.

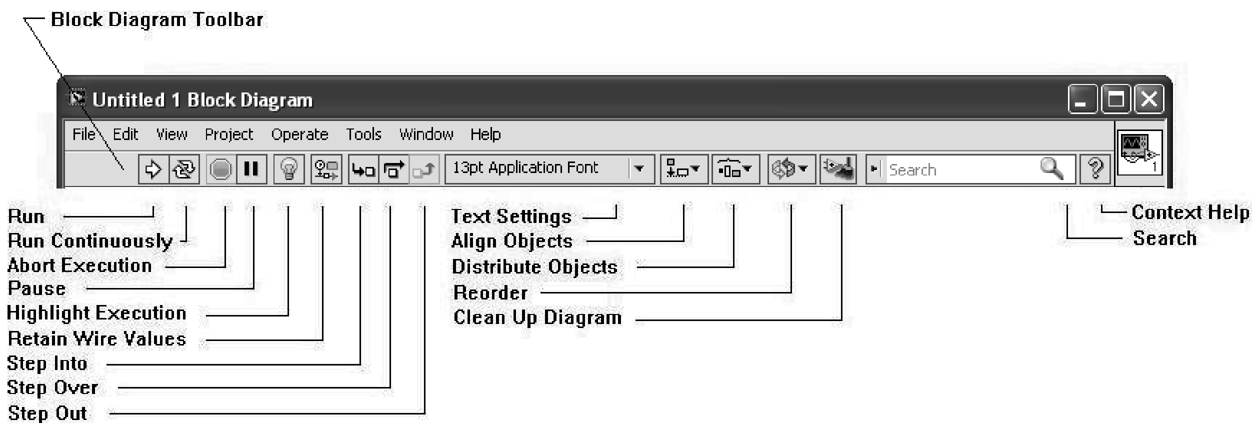


Figure 1.3.5 Block diagram toolbar

1.3.5 Palettes

There are three important palettes: “Tools,” “Controls,” and “Functions.” The “Controls” palette is available only in the front panel, the “Functions” palette is available only in the block diagram, and the “Tools” palette is available in both.

The “Tools” palette, as shown in Fig. 1.3.6, contains special cursors that enable you to perform different functions, like typing alphanumeric characters, entering values, and wiring, selecting, resizing, and positioning objects. A functional name for each tool is given in Fig. 1.3.6 along with the name assigned in the “Tools” palette in parentheses. This palette may already be displayed in the front panel. If not, click on “Tools Palette” from the “View” pulldown menu. At the top of the “Tools” palette, there is an automatic tool selection button. When this button is highlighted green, LabVIEW tries to anticipate what cursor you will need. You may find that this feature improves your programming efficiency although use of the automatic tool selection is a personal preference. If you find that you don’t like it, you can disable the feature by clicking on any of the buttons in the “Tools” palette and the automatic feature will be disabled. You will then need to manually select the tool needed until you click on the “Automatic Tool Selection” button at a later time.

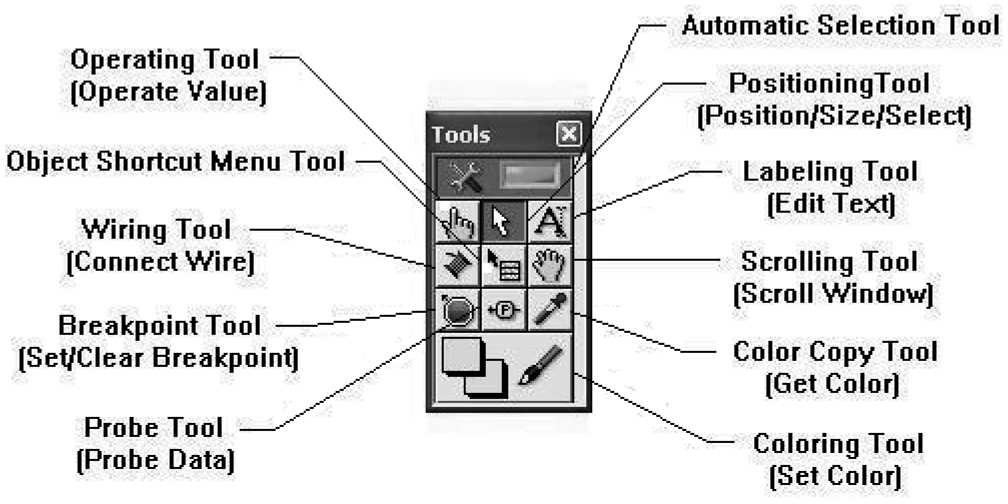

Automatic Selection Tool
Operating Tool (Operate Value)
PositioningTool (Position/Size/Select)
Tools
Object Shortcut Menu Tool
Labeling Tool (Edit Text)
Wiring Tool (Connect Wire)
Scrolling Tool (Scroll Window)
Breakpoint Tool (Set/Clear Breakpoint)
Color Copy Tool (Get Color)
Probe Tool (Probe Data)
Coloring Tool (Set Color)

Figure 1.3.6 The “Tools” palette

Four of the tools that you will frequently use in this primer include:

- Operating
- Positioning
- Labeling
- Wiring

The “Operating” tool allows you to enter data or change the value of a control. The “Positioning” tool allows you to select, move, or resize an object. The “Labeling” tool allows you to create a free label or edit an existing one. The “Wiring” tool allows you to establish data flow between nodes in the block diagram by wiring them together.

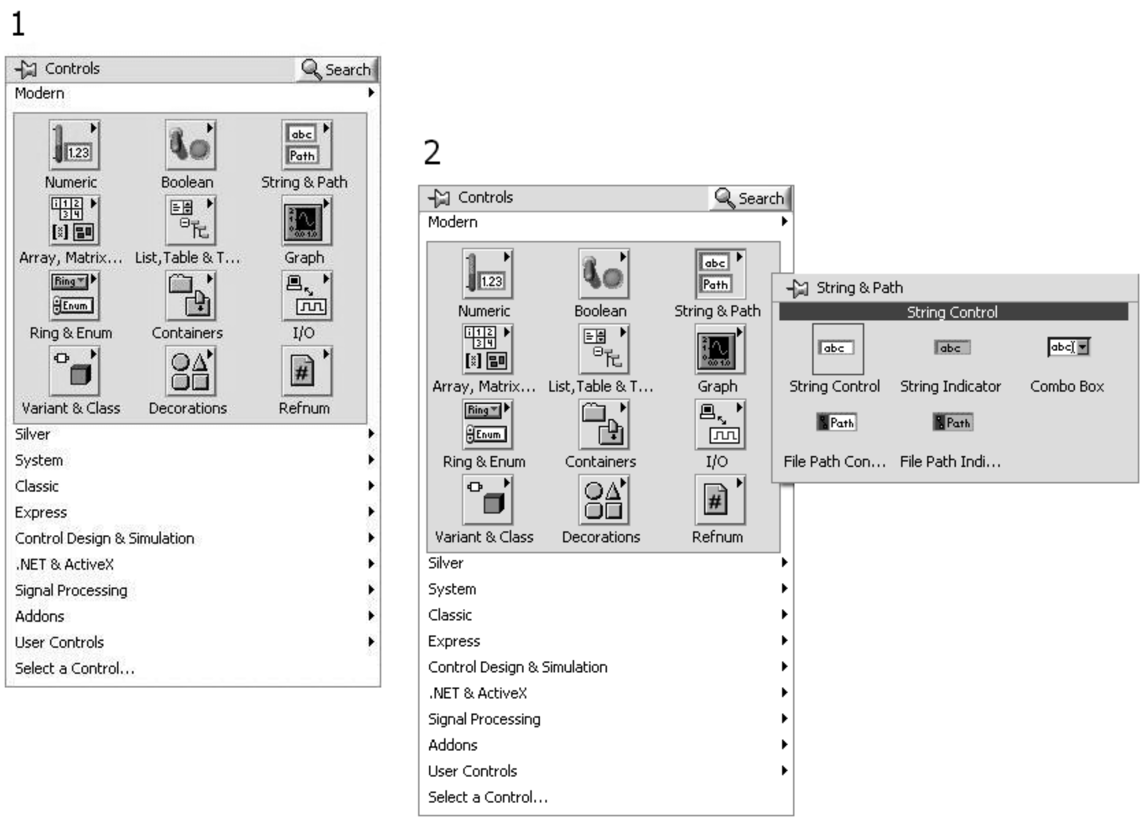
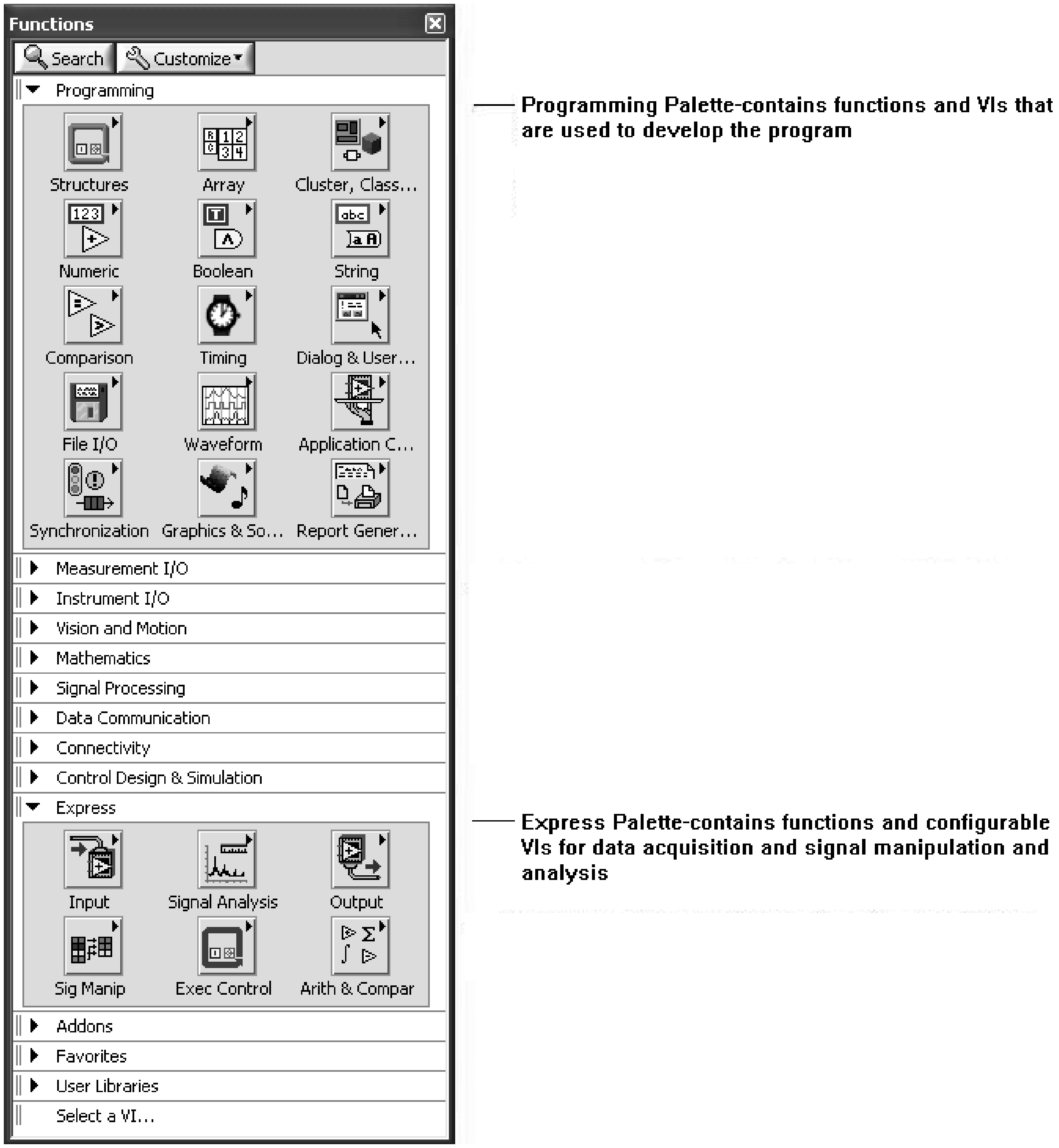


Figure 1.3.7 The “Controls” palette

The “Controls” palette contains different controls and indicators for the front panel (virtual instrument panel). If the “Controls” palette is not visible in the front panel, there are two options to retrieve the palette. One option is to select “Controls Palette” from the “Views” menu and the “Controls” palette will remain with the front panel as you use the selected icons. You may have to expand the palette by clicking on the button with the right facing arrow if a view similar to the one in Fig. 1.3.7 does not appear. The other option is to place the mouse over an open area in the front panel and right click. Using this approach, a view similar to the first image in Fig. 1.3.7 should appear. The “Controls” palette disappears after the icon is selected, which frees up space on the front panel. If you retrieve the palette by right-clicking on the front panel and want the palette to remain, you may also tack down the palette by clicking on the thumbtack in the upper left corner. Whether the palette remains on the front panel for ready access or disappears to free up working space is a matter of personal preference.

Since there are a number of different styles and categories of controls and indicators, most of the palettes are collapsed showing only the category heading name. To expose other palettes, place the cursor over the palette category name or click on it. You can expose subpalettes the same way, as shown in steps 1 and 2 in Fig. 1.3.7. You may have to expand the palette by clicking on the button with the right facing arrow. You may then click on the object of interest and drag it to the front panel.

The "Functions" palette contains all of the graphical programming functions that may be used to develop your program in the block diagram. The term function is applied loosely to functions, VIs, and Express VIs found in the Functions palette. Functions have inputs and outputs to perform specific tasks, like arithmetic or logic operations. VIs on the "Functions" palette are typically LabVIEW programs with a specific purpose that are referred to as subVIs when used in another VI. VIs whose parameters can be configured through a dialog box are referred to as Express VIs. An advantage of the Express VI is that input parameters can be configured interactively, which is usually a benefit for new LabVIEW users. The two functions palettes that will be used primarily in this primer are shown in Fig. 1.3.8. The "Programming" palette contains the building blocks for developing source code. Data acquisition Express VIs in the "Express" palette will be used in Part 2 of this primer to measure and generate data.



Programming Palette-contains functions and VIs that are used to develop the program

Express Palette-contains functions and configurable VIs for data acquisition and signal manipulation and analysis

Figure 1.3.8 The “Functions” palette

1.4 An Experiential Introduction to LabVIEW

This section describes how to write a relatively simple analog input VI to introduce LabVIEW and develop your skills. The VI can be used to record either a finite set of analog measurements or record measurements continuously from multiple channels. The idea is that it will be easier to learn and retain key LabVIEW concepts by applying the concepts as you learn them. Analog input means to acquire data from devices with voltages that vary continuously. This VI is most appropriate when you acquire data at relatively low sampling rates and the length of time to record data is uncertain. For example, this VI would be appropriate to measure strain sensed by a strain gage affixed to a structural member with a time-varying load or to measure the temperature, as sensed by a thermocouple, of a cooling object.

This VI employs nonbuffered data acquisition and software timing. Nonbuffered data acquisition means that samples are acquired one at a time and are stored temporarily within memory on the DAQ board. LabVIEW can then read the sample from the DAQ board and use it in the VI or store the sample on a permanent storage device, such as a hard drive or flash drive. Software-timed intervals are controlled by LabVIEW software timing functions, which depend on the computer's CPU clock. Software timing can produce irregular sample intervals while data is collected, especially if the requested time intervals are small or the CPU has large demands for resources, such as a graphic-intensive task like moving a window on the screen. From a practical point of view, this VI can sample at rates up to approximately 200-500 samples per second, although the maximum rate is limited by the ability of the computer's hardware to execute the LabVIEW software. This VI could also be used to sample at very low rates, such as one sample per hour.

Within this section, guided steps are interwoven among concepts that allow you to learn LabVIEW while you are developing a practical VI. The section is formatted as follows:

- A new LabVIEW concept is first introduced and a brief overview is provided to familiarize you with its function,
- Steps are then provided to help you implement the new concept into the development of the VI,
- Additional information about the new concept may follow the steps so that you may explore more general features of LabVIEW using your VI, which will provide a better foundation for subsequent VI development.

The material discussed in this section assumes that LabVIEW Professional Version or Student Version software, NI-DAQ software, and a data acquisition board have been installed on your computer.

Goals

1. Become acquainted with basic LabVIEW concepts that will be used throughout this primer.
2. Become acquainted with the LabVIEW DAQ Assistant, a means to create tasks to acquire and generate data.
3. Learn about ways to display data including charts and graphs and the ability to write data to a spreadsheet.
4. Develop the software-timed analog input VI shown in Figs. 1.4.1 and 1.4.2, which can acquire either a finite set of measurements or acquire measurements continuously until stopped by the user.

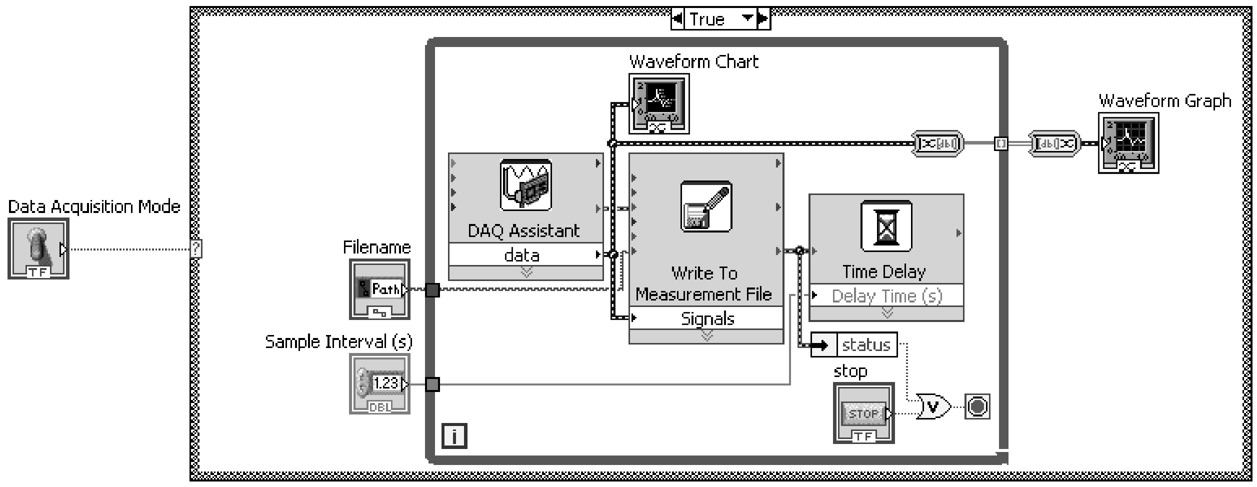


Figure 1.4.1 The general analog input VI showing the option to measure data continuously

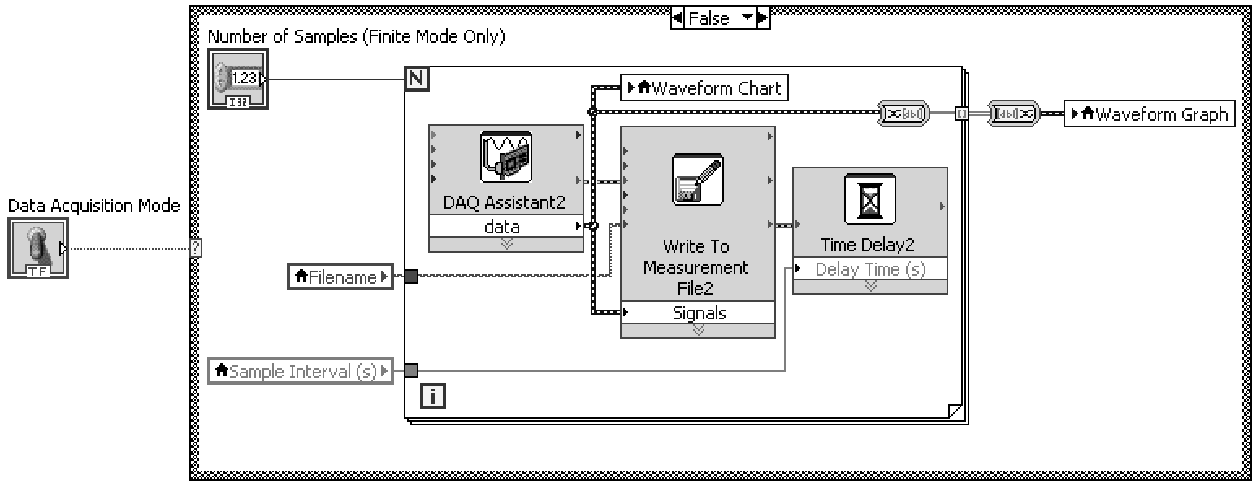


Figure 1.4.2 The general analog input VI showing the option to measure a finite set of data

1.4.1 Case Structures

Overview of Case Structures

The Case Structure executes a portion of code contained within its borders that corresponds to a condition, or case, among two or more possible case options. The Case Structure consists of a border, which encompasses the code, a selector label on the top of the border, and a selector terminal on the left side of the border. The case is identified in the selector label. The portion of code that executes is determined by the case input to the selector terminal. The case may be determined by the user through a control or by other code in the block diagram that is external to the Case Structure.

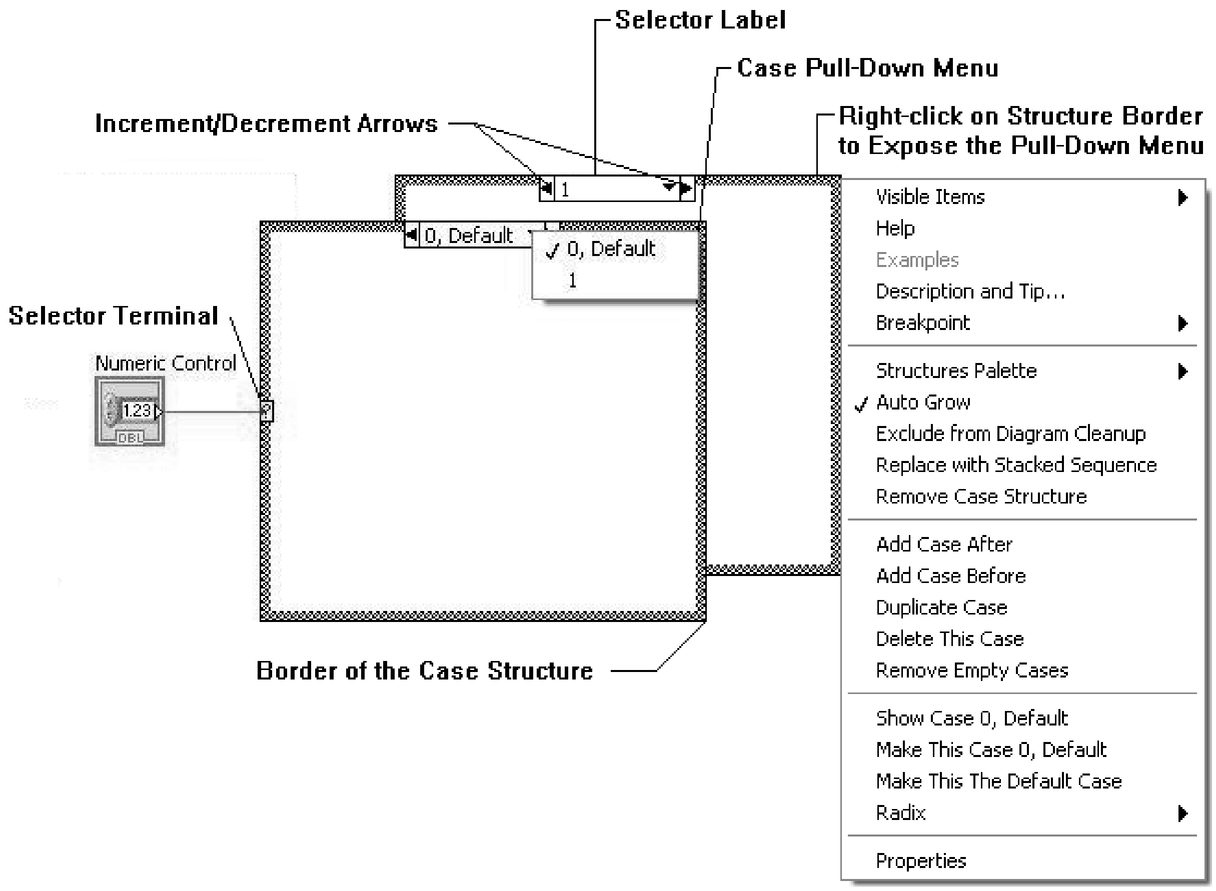


Figure 1.4.3 LabVIEW Case Structure

Two cases of a Case Structure are shown in Figure 1.4.3. The subdiagram, or code, that is to be conditionally executed is contained within the border of each case of the Case Structure. Each case will have a different subdiagram. Cases are stacked and show only one subdiagram at a time, unlike Fig. 1.4.3, which includes an offset second case for illustration purposes.

The case is determined by the data type wired to the selector terminal. A numeric control is wired to the example shown in Fig. 1.4.3, which shows two number cases, 0 and 1. The number case "0" has been selected as the default case. The default case will be executed if a wired value does not match any of the other cases. The default case might include an error message, for example. The example in Fig. 1.4.3 shows that a control is wired to the selector terminal, which would allow the user to directly select the case.

Input and output data may pass through tunnels in the border of the Case Structure. The tunnels are depicted by squares on the border. Input data that passes through tunnels is available to all cases. If output data is wired to the border of one case of the Case Structure, all cases must output a value or else the "Run" arrow on the toolbar will remain broken. An output tunnel appears as a hollow square until data is provided from all cases, at which point the tunnel appears as a solid square.

Steps 1-5: Creating a Case Structure

The general analog input VI that is to be developed in this problem is designed to allow the user to select finite or continuous measurement of data. For this VI, the user will provide input through the front panel to select a measurement case for data acquisition: either continuous or finite. A Case Structure (represented by the outermost border in Figs. 1.4.1 and 1.4.2) will be used to determine what case will be executed. When the data acquisition mode is set to "True," data is taken continuously. Likewise, when the data acquisition mode is set to "False," a finite set of data is taken. Two cases (True and False) of the same Case Structure are shown in Figs. 1.4.1 and 1.4.2.

1. If you haven't already done so, launch LabVIEW by selecting **All Programs>>National Instruments>>LabVIEW 2013 (or LabVIEW 2014) >>LabVIEW 2013 (or LabVIEW 2014)** from the "Start" menu.

Note: Read Section 1.3 to get the necessary background on the LabVIEW environment, if you haven't already done so.

2. Select "Blank VI" to open a new file for this exercise.

Tip: Use the keyboard shortcut <Ctrl T> to tile the windows with the front panel above and the block diagram below.

3. In the Functions palette, place the cursor over the Express palette and then over the Execution Control subpalette. Depress the left mouse key on the Case Structure icon in the Execution Control subpalette and drag the structure to the block diagram. This

procedure will be referred to as **Express>>Execution Control>>Case Structure** in the remainder of this primer. If you have problems with this or any other step, you can remove the Case Structure and start over using the "Undo" feature on the "Edit" pull-down menu.

Note: The Case Structure can also be found in **Programming>>Structures>>Case Structure**.

4. Resize the Case Structure to be large enough to contain the functions, structures, and VIs shown in Figs. 1.4.1 and 1.4.2.

Tip: The initial size is not critical since the Case Structure can be resized at any time by clicking on the border and dragging the border with the mouse on one of the blue "handles."

5. Place the cursor over the selector terminal (box containing the question mark on the left border), right click, and select "Create Control."

Note: A Boolean push button control appears simultaneously in the front panel and a terminal appears in the block diagram. This control will allow a user to determine if a finite set of data will be measured or if the data will be measured continuously.

Additional Information about Case Structures

Different data types, such as Boolean (True or False) and string (text), can be wired to the selector terminal and case values will be shown at the top of the border in the Selector label area. You can select a case by cycling through the available cases using the increment and decrement arrows or by using the pull-down menu by selecting the down arrow in the Selector label.

A number of options for the Case Structure are available if you right-click on the structure border, as shown in Fig. 1.4.3. For example, you can add or delete a case. If you add a case, you can change the value in the selector label using the Edit Text (letter A) cursor.

Case Structures are part of a larger class of structures that control the execution of data flow in a VI. Some of the other structures used in this primer are listed below.

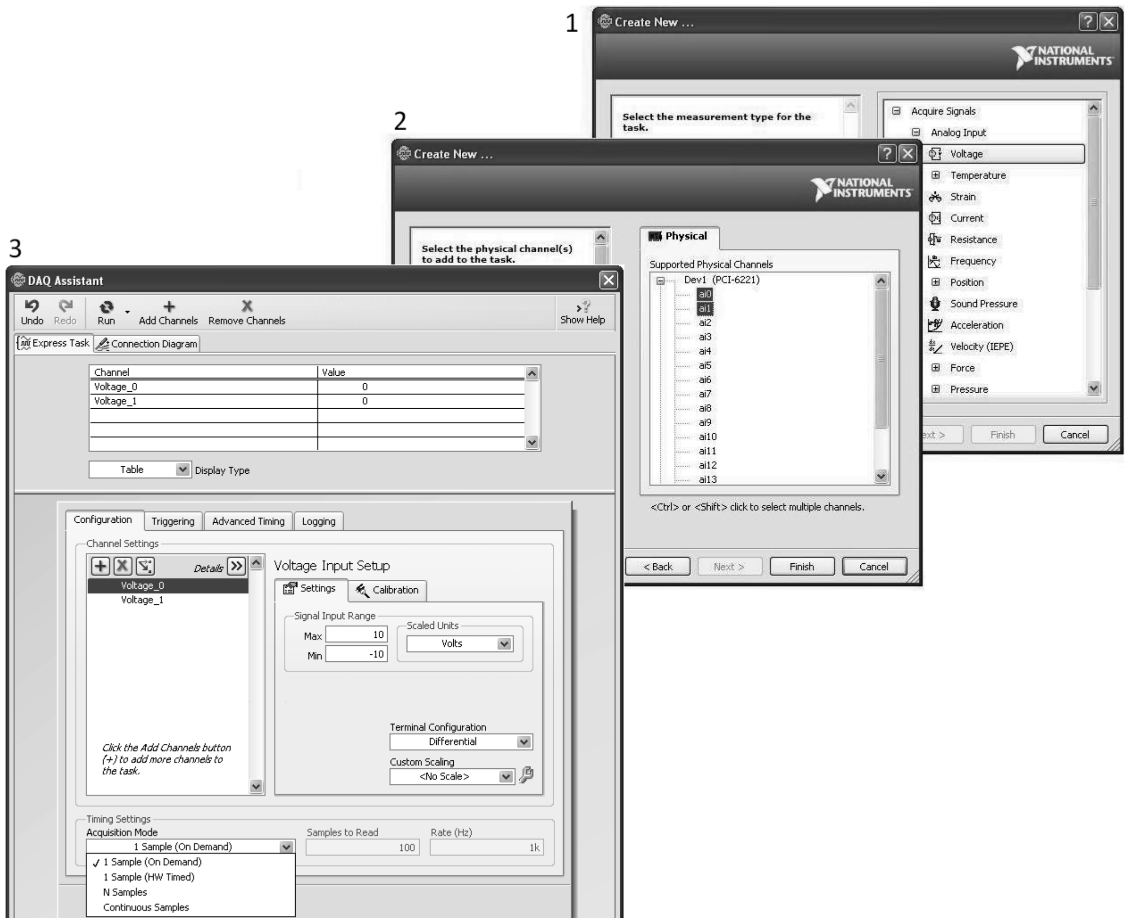
- The While Loop continuously executes a portion of code within its borders, called a subdiagram, until a condition is met;
- the For Loop executes a subdiagram a finite number of times;

- and the Sequence Structure executes one or more subdiagrams in a sequential order.

1.4.2 Data Acquisition: The DAQ Assistant

Overview of the DAQ Assistant

The DAQ Assistant is a configurable Express VI that can create, edit, or test a data measurement or generation task. A task contains information on the timing, triggering, and configuration of one or more channels. The DAQ Assistant graphical user interface allows the user to configure channels and set data acquisition timing and triggering conditions. An advantage of the DAQ Assistant is that the graphical user interface guides the user to properly configure data acquisition tasks, which is beneficial for new users.



Select the measurement type for the task.

Select the physical channel(s) to add to the task.

Supported Physical Channels

Dev1 (PCI-6221): ai0, ai1, ai2, ai3, ai4, ai5, ai6, ai7, ai8, ai9, ai10, ai11, ai12, ai13

<Ctrl> or <Shift> click to select multiple channels.

Channel	Value
Voltage_0	0
Voltage_1	0

Display Type: Table

Channels: Voltage_0, Voltage_1

Click the Add Channels button (+) to add more channels to the task.

Signal Input Range: Max 10, Min -10

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Acquisition Mode: 1 Sample (On Demand)

Samples to Read: 100

Rate (Hz): 1k

Figure 1.4.4 Using the DAQ Assistant to configure an analog input measurement task

The DAQ Assistant guides the user through a series of windows to configure the data acquisition task as shown for an analog input measurement task in Fig. 1.4.4. The DAQ

Assistant automatically launches when placed in the block diagram. The “Create New…” window sets up the measurement type for the data acquisition task. You select if the data will be acquired or generated for the measurement type (this view is not shown in Fig. 1.4.4). If you expand the list, you can see that analog, digital, and counter modes are available for both “acquire” and “generate.” By further expanding the list, you can see what types of measurements are supported for each mode. For example, the different types of analog input measurements are shown in the first view of Fig. 1.4.4. Once you select the type of measurement you want, for example, a voltage analog input measurement was selected in Fig. 1.4.4, the DAQ Assistant then lists the channels that are available for that type of measurement based on the DAQ board in your computer. This is the second view in Fig. 1.4.4. After you select the channel(s) that you want, a DAQ Assistant window opens, as shown in the third view of Fig. 1.4.4, which allows you to configure the channel(s).

Steps 6-9: Creating a Measurement Task using the DAQ Assistant

In the following steps, you will create an analog input measurement task to sample data from two different channels when called by the software. You will use the DAQ Assistant to create the measurement task.

6. Select **Express>>Input>>DAQ Assistant** from the Functions palette in the block diagram and drag it inside the Case Structure.

7. Click on “Acquire Signals,” “Analog Input,” and “Voltage” as shown in Fig. 1.4.4 to create a measurement task that can sample a continuously varying voltage signal.

8. Click on the hardware device to show the channels that are available to measure analog input signals on your DAQ board. Select analog input channel 0 (“ai0”) and channel 1 (“ai1”) by depressing the control key, <Ctrl>, on the keyboard while selecting the channels with the left mouse key. After you select “Finish,” the DAQ Assistant window appears.

Note: All of the default settings are acceptable for this example problem except the timing settings of the acquisition mode. The default signal input range is ±10 V, the maximum range allowed by the data acquisition board. You may modify this parameter at a later time if you measure a signal having a different voltage range. Differential mode is the default configuration of the input terminals. This means that positive and negative leads must be connected to the data acquisition board. Other types of terminal configurations are described in Section 2.1.1. No custom scaling is selected, but this feature allows you to create a scale that converts voltage data into physically meaningful units, like temperature or pressure. Without any custom scale, the “Scaled Units” parameter shows “Volts.”

9. Since this VI uses software timing, select “1Sample (On Demand)” from the “Acquisition Mode” pulldown menu and then OK at the bottom right corner of the window.

Tip: You can double-click on the DAQ Assistant icon to edit the configurations at a later time, if needed.

Note: The DAQ Assistant can be displayed as an icon or an expanded node by dragging the icon by the “handle” at the bottom of the icon.

Additional Information about the DAQ Assistant

The DAQ Assistant can, among other things:

- create and edit data measurement and generation tasks;
- create and configure channels in the tasks;
- create and edit scales that convert voltages into physically meaningful units, and
- test and save your data measurement or generation configuration.

Specifications in the data measurement or generation task include the acquisition mode and timing. As seen at the bottom of the window in Fig. 1.4.5, there are four acquisition modes: two of which take single samples, a finite set of N samples, and continuous acquisition by repetitively taking blocks of a finite set of N samples. The timing can be either software timing controlled by the computer’s CPU clock, which occurs when the LabVIEW software calls a subVI to acquire data, or by hardware timing, which occurs when a clock on the DAQ board or an external hardware device controls the data acquisition.

The first acquisition mode, “1 Sample (On Demand),” employs software timing since the sample is not acquired until a LabVIEW subVI demands the sample. It is referred to as software timing since the execution of the LabVIEW software is controlled by the CPU clock. This sample mode can also permit continuous data acquisition if the calling subVI is placed in a loop. However, the time spacing between VI calls depends on the execution time of the program and the CPU clock, which has other priorities as well as LabVIEW. This can result in uneven time spacing, especially for fast sampling rates, and ultimately limits how fast data can be acquired.

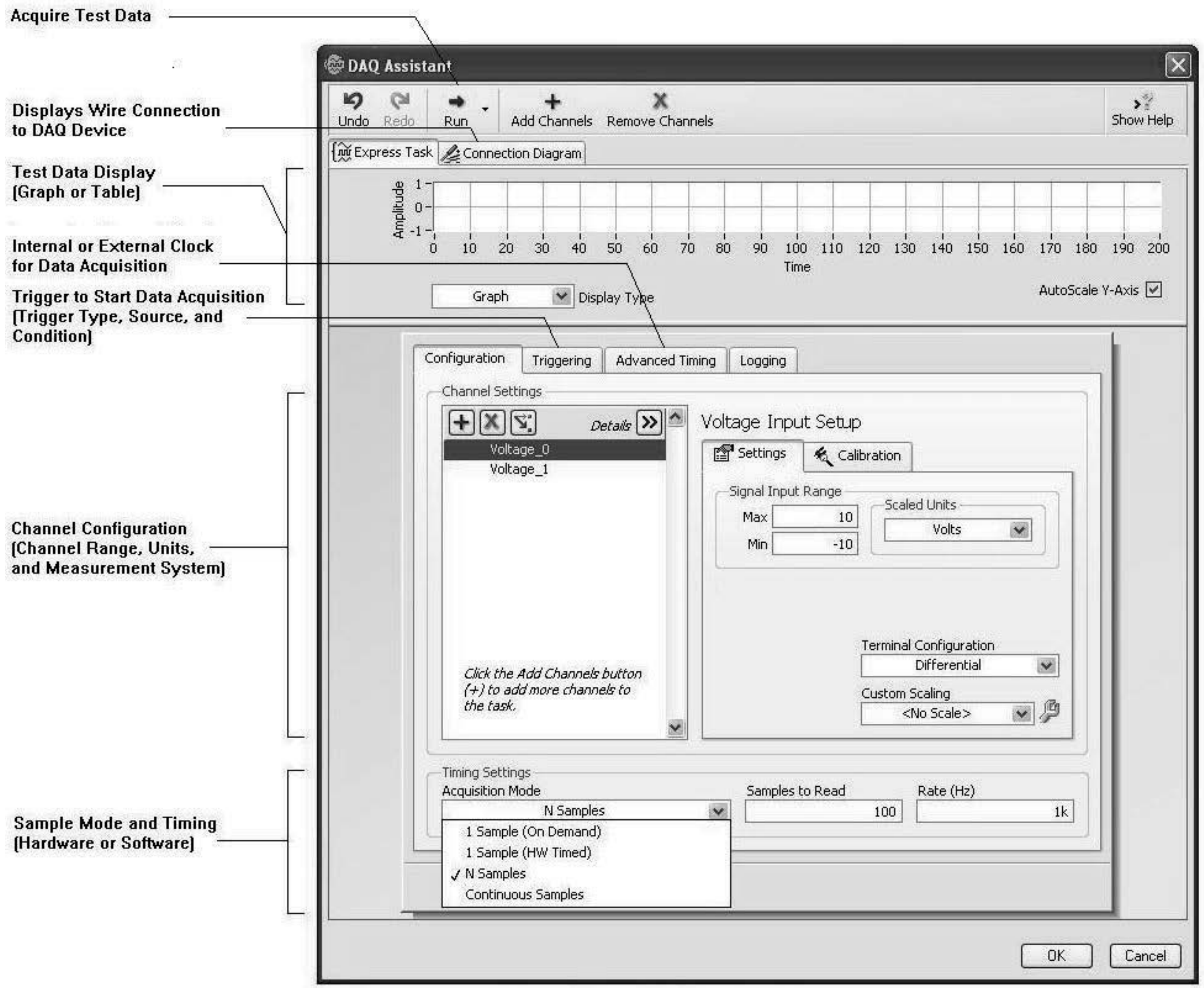


Figure 1.4.5 DAQ Assistant features

The second acquisition mode, “1 Sample (HW Timed),” takes one sample, whose acquisition is controlled through a clock on the DAQ board or an external timing device. Multiple samples can be taken by repetitive triggers, typically using a train of digital pulses from an external timing device.

The last two acquisition modes use hardware timing via a clock on the DAQ board or an external hardware timer. You can specify which one through the “Sample Clock Type” on the “Advanced Timing” tab. “Internal” and “External” refer to the DAQ board or external hardware device, respectively. The control of data acquisition is transferred to the DAQ board when using hardware timing. It ensures uniform time spacing and offers the possibility of significantly higher sampling rates compared to software timing, depending on the DAQ hardware installed on your computer.

Another specification in a data measurement or generation task includes triggering. The data measurement or generation task will be executed as soon as it is called by the LabVIEW subVI, unless triggering is employed using options listed in the "Triggering" tab. Depending on the capabilities of your hardware, data acquisition may be triggered by an analog or digital signal from, for example, a sensor or a relay. Triggering is essential to acquire data when an event will occur rapidly yet the onset of the event is unknown. A rapid event would dictate a high sampling rate, yet copious quantities of data would need to be stored if the data measurement was not triggered. The rupture of a pressure vessel is a good example since a high sampling rate is required to capture the pressure history yet the timing of the rupture is unknown.

Virtual channels can be created and configured using the DAQ Assistant. Virtual channels are required for data measurement or generation tasks. A virtual channel is comprised of:

- a physical channel,
- the type of measurement or generation system,
- the voltage range for the channel, and
- scaling information.

A physical channel is part of the DAQ board and manifests itself as a terminal or pin at the terminal connector block that can make a wired connection to an input or output device.

You specify how the input or output device is connected using the "Terminal Configuration" options, shown in the middle of the window in Fig. 1.4.5. The option shown, a differential measurement system, reads the potential difference between two terminals. The other two options (not shown in Fig. 1.4.5) include both referenced and nonreferenced single-ended measurement systems. The referenced single ended (RSE) measurement system measures the signal with respect to the DAQ hardware (system) ground. The nonreferenced single-ended (NRSE) measurement system measures the signal with respect to a common reference, for example, a shared power supply ground that is not the system ground. A description of these measurement systems is given in Section 2.1.1. Depending on the measurement system selected, the DAQ Assistant shows how to connect the wires of the input or output device to the DAQ board's terminal connector block through the "Connection Diagram" tab, shown near the top of the window in Fig. 1.4.5.

The "Signal Input Range" determines the voltage range accepted by the DAQ board for that channel and is based on the anticipated minimum and maximum voltages for the input signal. A physical channel can also be scaled to convert a voltage to a physically meaningful unit. A sensor's calibration curve, typically a linear relationship, can be entered through the "Custom Scaling" options.

Finally, once the data measurement or generation task has been created and configured, the DAQ Assistant allows test data to be taken using the "Run" button, which is shown at the top of the window in Fig. 1.4.5. You can display the data in tabular or graphical form. This is a very useful feature of the DAQ Assistant since you can check data from your measurement system with an independent method, such as a multimeter or an oscilloscope, to verify that everything is connected and configured correctly.

1.4.3 Writing to a Measurement File

Overview of the Write to Measurement File Express VI

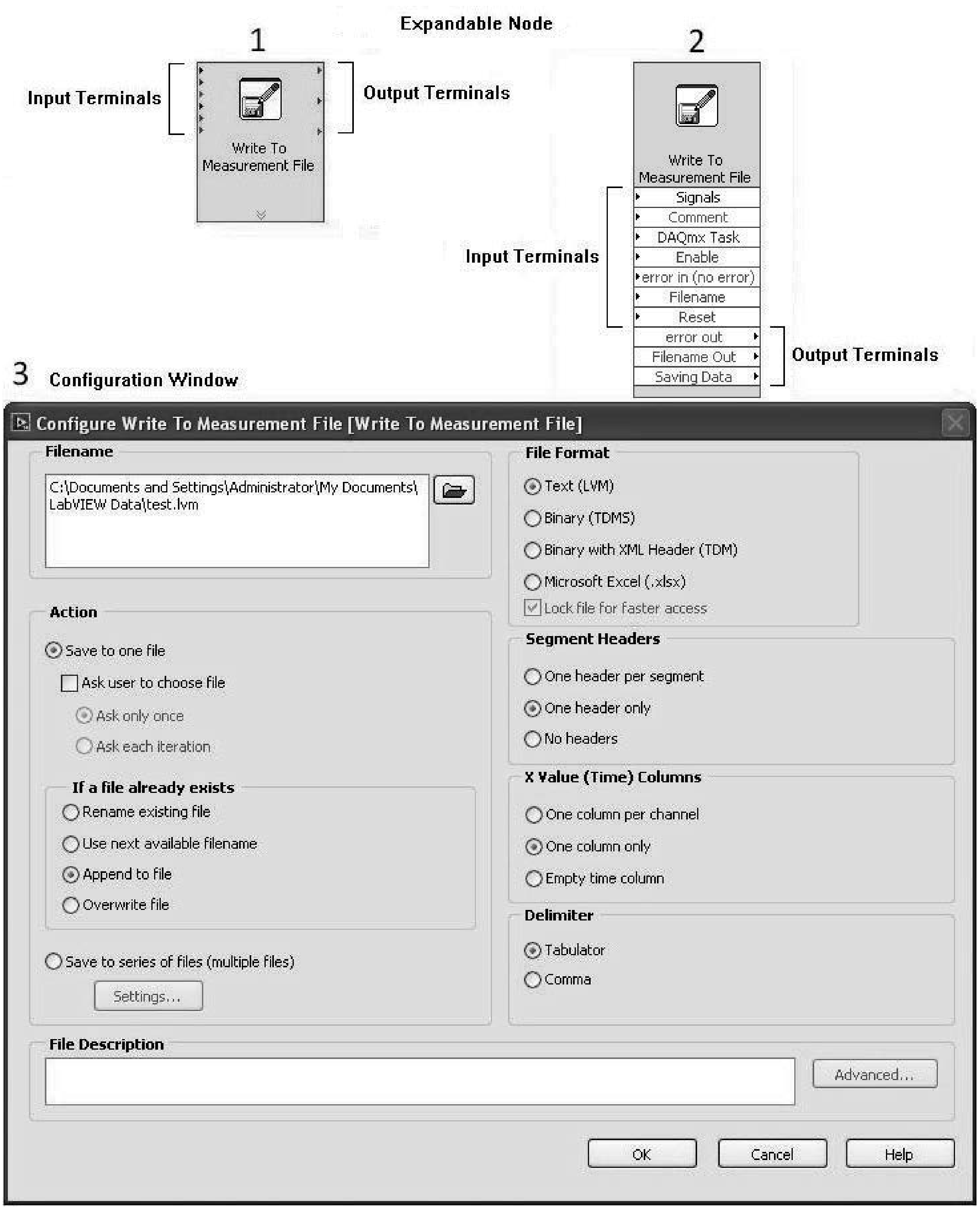
The general analog input VI that we are developing collects data continuously, one data point per channel per iteration. Since the data measured through a measurement task (created and configured by the DAQ Assistant) is stored in temporary RAM memory, it must be written to a permanent file to archive it. This can be accomplished through the "Write to Measurement File" Express VI.

The Write to Measurement File Express VI writes numerical data to a text-based measurement file with a ".lvm" (**L**ab**V**IEW **M**easurement) extension. The data in a text-based file is human readable, separated by a delimiter like a tab or comma, and can be read by a spreadsheet or word processing application for later analysis, plotting, or printing.

This Express VI is an expandable node as are most Express VIs, such as the DAQ Assistant. The VI appears as the icon shown in the first view of Fig. 1.4.6 when placed in the block diagram. The VI may also be expanded by placing the "Position" (arrow) cursor over one of the top or bottom blue "handles" and dragging the handle until the VI appears similar to the second view in Fig. 1.4.6. This has the advantage of making it easier to wire the input and output terminals although at the expense of space in the block diagram.

Steps 10-19: Writing Data to a Measurement File

In the following steps, you will configure a file for permanent storage of the measurement data. For this to be completed, you will use the Write to Measurement File Express VI.



Configure Write To Measurement File [Write To Measurement File]

Filename

C:\Documents and Settings\Administrator\My Documents\LabVIEW Data\test.lvm

File Format

[✓ Text (LVM)]
Binary (TDMS)
Binary with XML Header (TDM)
Microsoft Excel (.xlsx)
[✓ Lock file for faster access]

Action

[✓ Save to one file]
Ask user to choose file
[✓ Ask only once]
Ask each iteration

If a file already exists

Rename existing file
Use next available filename
[✓ Append to file]
Overwrite file

Save to series of files (multiple files)

Segment Headers

One header per segment
[✓ One header only]
No headers

X Value (Time) Columns

One column per channel
[✓ One column only]
Empty time column

Delimiter

[✓ Tabulator]
Comma

File Description

Figure 1.4.6 Write to Measurement File Express VI

10. Select **Express>>Output>>Write to Measurement File** from the "Function" palette in the block diagram and drag the icon inside of the Case Structure and to the right of the DAQ Assistant.

Note: You may skip the next step if the default filename provided by LabVIEW is acceptable.

11. Type the path of a file in the "Filename" dialog box or select an existing one by clicking on the folder to the right of the default file path to browse the directory on your computer or external storage device. LabVIEW will create the file if the filename does not exist.

12. Make sure the action is to "Save to one file" but *do not* check the box "Ask user to choose file."

Note: The user was not asked to choose a file to avoid the potential of delaying data acquisition. Based on the execution of the block diagram in Fig. 1.4.1, the Write to Measurement File Express VI executes after the measurement task is created and samples are taken. If the user was asked to choose a file, the execution of the program would be suspended until a filename was provided by the user. Potential measurements could be missed if the physical event occurred quickly.

13. Select "Append to file" under the heading "If a file already exists."

Note: Since the VI will take data continuously in the current example, a file will be created on the first iteration and data should be appended to that file on subsequent iterations. Otherwise, a different file name would be required every iteration. Any of the other options may be suitable if a finite number of data points are taken and data are recorded after the measurement task is complete.

14. Select "Text (LVM)" under "File Format."

Note: A text format is chosen so that the data can be viewed in a spreadsheet at a later time. Binary measurement files cannot be read by humans and are used to transfer data efficiently between software.

15. Select "One header only" under "Segment Headers."

Note: The header contains information like the date and time the data was measured. One header was chosen in this example. Otherwise, there would be a header for every data point if one header per segment was selected since this Express VI is executed every iteration.

16. Select “One column only” under “X Value (Time) Columns.”

Note: The time the sensor data was measured relative to the first data point can also be included with the measured sensor data. One X (time) column was selected, which will show time in the first column followed by each analog input channel in subsequent columns in the order listed in the DAQ Assistant channel settings. “One column per channel” means there will be a time-variable pair of columns for every channel (variable measured).

17. Select “Tabulator” under “Delimiter” and click on OK. Tabs are used so that commas do not appear in a word processor.

Tip: You can double-click on the “Write to Measurement File” icon to edit the configurations at a later time, if needed.

18. Place the cursor over the “filename” input terminal (it’s the bottom terminal on the left side of the unexpanded node-see view number 1 of Fig. 1.4.6), right-click the mouse, and create a control. The control will appear in the front panel and the corresponding terminal will appear on the block diagram.

Tip: If you are having difficulty locating the “filename” input terminal, there are two ways to easily find it if the node is not expanded. The first way is to place the “Connect Wire” (solder spool) cursor over the terminals to locate the filename input terminal. As the “Connect Wire” cursor passes over a terminal, the terminal name pops up. The second way is to use the “Context Help” window. Select “Show Context Help” from the “Help” pull-down menu to show labeled terminals. If you are using the “Connect Wire” cursor, the terminal will blink in the “Context Help” window and light up on the icon with the terminal name displayed in the block diagram.

Note: The filename control will allow a user to enter a filename without having to open the “Write to Measurement File” configuration window every time a different filename is desired.

19. Click on the filename input terminal with the “Position” (arrow) cursor and drag the terminal near the left border of the Case Structure, as shown in Fig. 1.4.1, so that it will be outside of the inner loop.

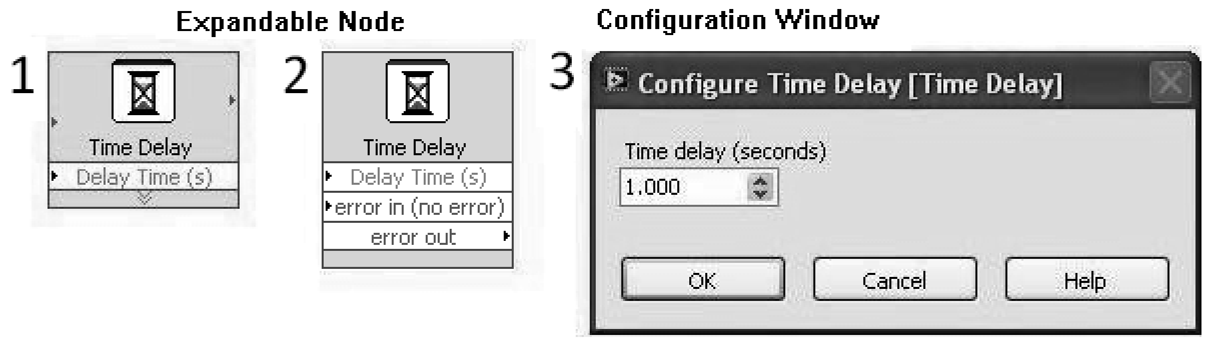
1.4.4 Timing VIs for Control of VI Execution

Overview of Timing VIs

Timing VIs are useful to control the execution of the program. In the development of the current analog input VI, a While Loop will be placed around the DAQ Assistant VI, as shown in Fig. 1.4.1. Without any timing VIs, the program will execute as fast as the computer can process the code. This is undesirable if you want to measure data at a specified rate. The timing VIs provide a means to control VI execution. When applied to data acquisition, this is referred to as software timing, since the timing VI is controlled by the computer's CPU clock.

The timing VIs provide only an approximate means to establish a sampling rate to acquire data. The time delay is added to the time it takes to execute all of the other code in the VI before the next measurement is taken. However, for moderate to slow sampling rates (approximately 1 second/sample or greater), the time for VI execution is typically not significant compared to the time delay. For fast sampling rates (on the order of milliseconds/sample), the time spacing between measurements is generally irregular anyway since the CPU must balance requests from LabVIEW for VI execution with other priorities. Once again, the additional time to execute the VI is not critical. Hardware timing should be employed if the time spacing between samples must be precise. Hardware timing is based on functions that transfer control of the data acquisition to a clock on an external device, like the DAQ board, and is discussed in Section 2.2.

There are both Express and traditional timing VIs in the Functions palette. The Express VIs are located in the **Express>>Execution Control** subpalette and the traditional VIs are located in the **Programming>>Timing** subpalette. The Express VI used in the current example, "Time Delay," inserts a specified time delay each time it is called. The "Time Delay" VI is shown as both an unexpanded and expanded node in the first two views in Fig. 1.4.7 and, in the third view, the configuration window that appears when the VI is placed in the block diagram.



Time delay (seconds)
1.000

Figure 1.4.7 Time Delay Express VI

Steps 20-22: Time Delay VI

In the following steps, the VI is modified to provide a sampling rate for the measurement of the data. This will be accomplished by adding the Time Delay Express VI after data is recorded in the measurement file.

20. Select **Express>>Execution Control>>Time Delay** and drag the VI inside the Case Structure to the right of the "Write to Measurement File" VI in the block diagram.

21. Press OK for the default value of 1 second since it can be changed at a later time by double-clicking on the "Time Delay" icon.

Note: A more convenient means of changing the time delay when executing the VI is through a control in the front panel, which will be employed in the current example VI.

22. Place the cursor over the "Delay Time" input terminal, right click on the mouse, and select **Create>>Control**. Select the "Delay Time (s)" terminal (later renamed "Sample Interval (s)") and drag it near the left border of the Case Structure as shown in Fig 1.4.1.

1.4.5 While Loop

Overview of While Loops

The While Loop, shown in Fig. 1.4.8, is a structure that executes the subdiagram enclosed within its borders until a condition is met. The condition is checked at the end of the iteration. The "Iteration" and "Conditional" terminals appear within the While Loop when it is first placed in the block diagram. The "Iteration" terminal outputs the number of times the loop has iterated beginning with a value of zero for the first iteration. The "Conditional" terminal executes until a condition is met. The default terminal condition is to stop if the input to the terminal is true. However, the condition can be changed to continue if true by using the shortcut menu that appears when you right-click on the conditional terminal as shown in Fig. 1.4.8.

The "Conditional" terminal is an input terminal, which can be satisfied by one or more inputs. The most common input to the "Conditional" terminal is a Boolean control in the front panel that allows the user to control execution of the VI by selecting true or false. However, when the While Loop is used for data acquisition, for example, it is also common to stop the While Loop when an error occurs in one of the data acquisition VIs. Since the "Conditional" terminal can accept only one wire, multiple inputs can be combined with a logical OR function such that the While Loop will stop execution if any of the inputs is true. A Boolean input is provided automatically if the While Loop is selected from the "Express" palette (**Functions>>Express>>Execution Control**) but not if selected from the "Programming" palette (**Functions>>Programming>>Structures**).

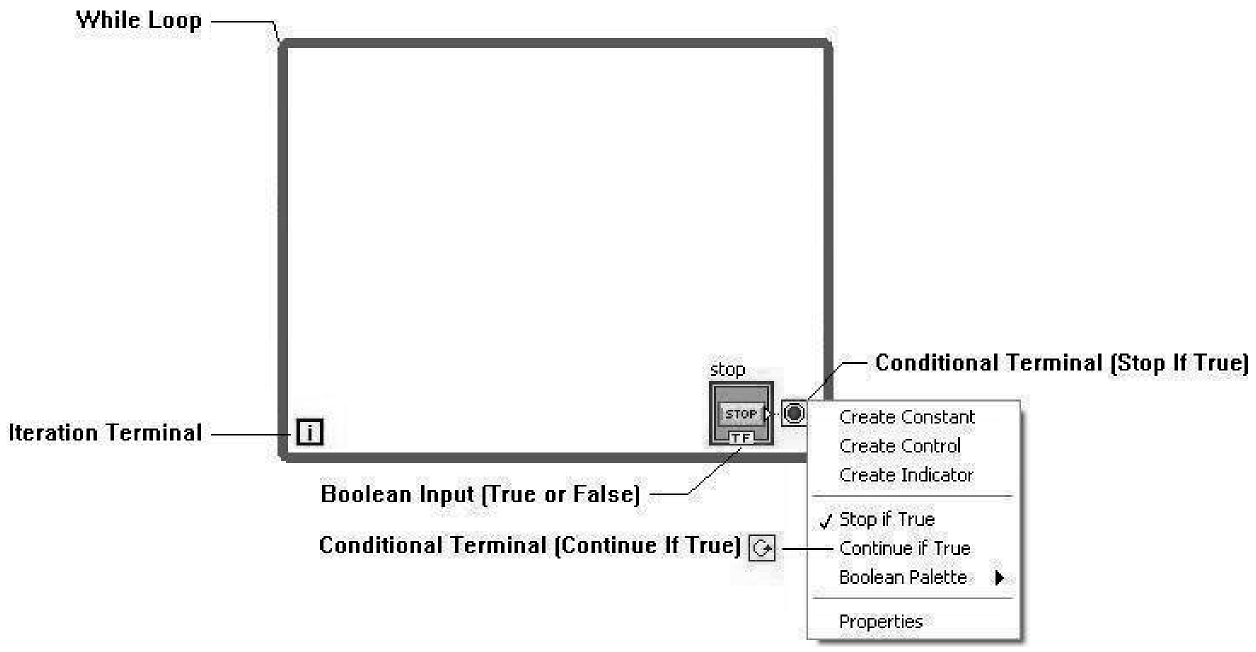


Figure 1.4.8 The While Loop

Step 23: Creating a While Loop

In this step, your program will be modified so that data measurements will be taken continuously, one sample per channel per iteration, until you stop execution. For this to be completed, a While Loop will be placed around the three Express VIs that create a measurement task, write the data to a file, and insert a time delay in the execution of the program.

23. Select **Express>>Execution Control>>While Loop**, place the cursor to the upper left of the three Express VIs, left-click the mouse and hold it down, drag the icon to the lower right to enclose the three Express VIs (but not the controls to the inputs of these Express VIs) as shown in Fig 1.4.9, and release the mouse key to create the While Loop.

Note: You do not have to depress the mouse key when dragging the border of the While Loop. However, you will have to left click the mouse key a second time to set the border if the mouse key does not remain depressed.

Tip: If you make a mistake dragging the While Loop, you can undo the creation of the While Loop using the “Undo” command from the “Edit” pulldown menu and start over.

Tip: You may also resize the Case Structure or While Loop using the blue handles if you need more room.

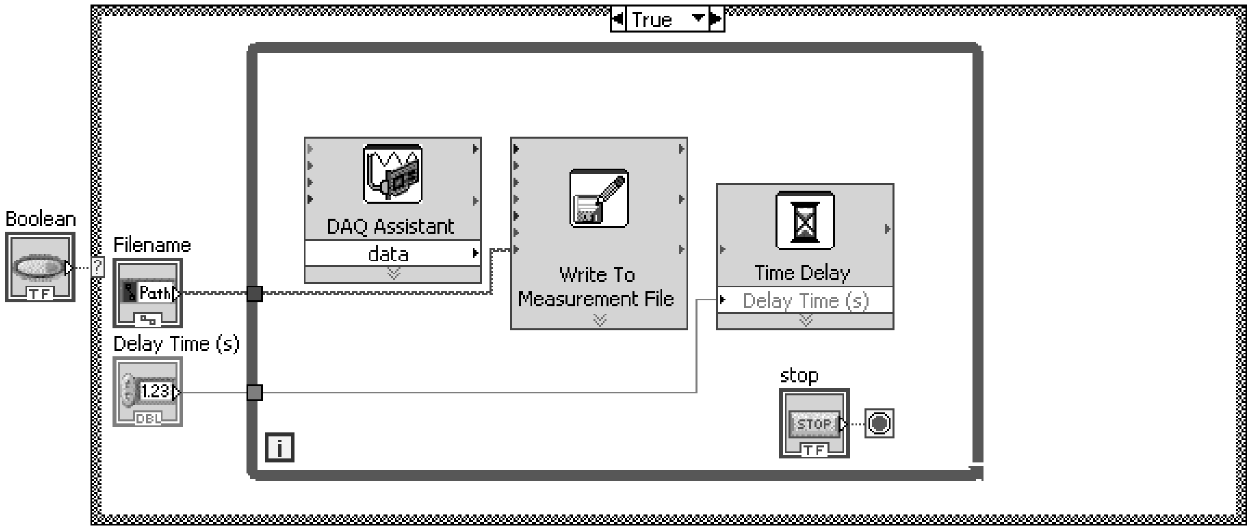


Figure 1.4.9 Intermediate view of the block diagram after the While Loop is created

Note: Two tunnels should appear on the left border of the While Loop showing that the “Filename” and “Time Delay” terminals become inputs. The conditional terminal is connected with a Boolean control and an iteration terminal also appears. You may also need to left-click the mouse on the Boolean control using the “Position” (arrow) cursor to drag it to expose the wire that connects to the conditional terminal.

Additional Information about While Loops

The input to the “Conditional” terminal must be placed inside the While Loop to prevent the possibility of an infinite loop, a condition that occurs when there is no way to stop execution of a repeated section of code. Values of variables that pass through the boundary of the While Loop remain constant during the execution of the While Loop until the loop stops. For example, consider that the value of a Boolean control in the front panel is set to false. If the corresponding Boolean terminal wired to the “Conditional” terminal in the block diagram is outside of the While Loop, then an infinite loop will be established. In this example, the false value is read once when the loop first executes and will not change, even if the user changes the value outside the loop at a later time as the loop is executing. If an infinite loop is established accidently, the VI can be aborted using the “Abort Execution” button on the block diagram toolbar.

The While Loop and “Conditional” terminal are preferred methods to control program execution over the “Run Continuously” and “Abort Execution” buttons on the block diagram toolbar. Additional data analyses or plotting of data may be desirable after a set of data have been measured, which would not be possible with the toolbar buttons.

While loops pass data through tunnels at the loop border. Fig. 1.4.10 shows a simple example with data on the right border that can pass out of the loop. Tunnels also appear for data passing into the loop. Since data arrays are indexed by rows and columns, tunnels can have indexing enabled or disabled. If indexing is enabled, an array of data is *input* one element at a time every iteration starting with the first element. When indexing is disabled, the entire array is passed through the tunnel on the first iteration. When indexing is enabled for *output*, a variable’s value is stored at each iteration as an element of a row array that is then passed out of the loop when execution is completed. The first element of the array is the value from the first iteration. With indexing disabled (Tunnel Mode>>Last Value), only the value from the last iteration is passed out of the loop.

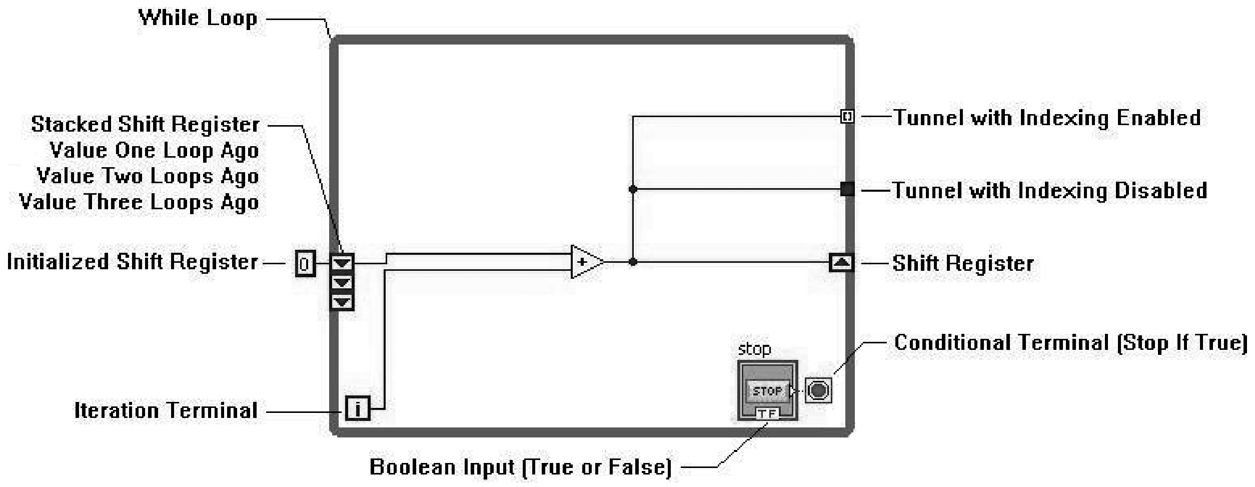

While Loop
Stacked Shift Register
Value One Loop Ago
Value Two Loops Ago
Value Three Loops Ago
Initialized Shift Register
Iteration Terminal
Tunnel with Indexing Enabled
Tunnel with Indexing Disabled
Shift Register
stop
STOP
TF
Conditional Terminal (Stop If True)
Boolean Input (True or False)

Figure 1.4.10 The While Loop with a simple example to show components

Shift registers pass variable values from previous iterations to the next iteration. A pair of terminals appears when a shift register is added. A shift register may be added by right clicking the mouse on the border and selecting “Add Shift Register.” The terminal on the right border marked by the upward arrow stores a value at the end of the most recent iteration. This value then becomes available at the beginning of the next iteration from the corresponding shift register on the left border with a downward arrow. On the first iteration of the While Loop, the initial value may be specified by using a constant or control wired to the shift register. If the While Loop has never executed and nothing is wired to the shift register, a default value for the data type will be used, such as zero for the integer type used in the example in Fig. 1.4.10. If the loop executed previously, stopped, and is to execute again, the initial value is the last value written to the shift register when the loop last executed, if nothing is wired to the shift register. You may also stack shift registers on the left border by right clicking on the register and selecting "Add Element." Each additional element stores values from each previous loop, respectively.

The example in Fig. 1.4.10 will be used with three iterations to illustrate shift registers and tunnels. This simple VI adds the values of the iteration terminal and the shift register and provides output values at the right border of the loop. On the first iteration, both the iteration terminal and the shift register have values of zero so that the shift register on the right border will have a value of zero. On the second iteration, the iteration terminal will have a value of one and the shift register on the left border will have a value of zero (the value at the end of the previous iteration) so that the shift register on the right border will have a value of one. On the third iteration, the iteration terminal will have a value of two and the shift register on the left border will have a value of one (previous iteration value) so that the shift register on the right border will have a value of three. The tunnels on the right border contain values that may pass to another node outside of the While Loop. If the While Loop stops after three iterations, the tunnel with indexing disabled contains the value three, which is the value at the last iteration. The tunnel with indexing enabled will be a one-dimensional array of values from all iterations in row format (0, 1, and 3).

1.4.6 Waveform Chart

Overview of Waveform Charts

The waveform chart is a numeric indicator that can display and continuously update one or more plots. When the chart is filled with data, the plot scrolls from right to left with new data appended from the right. Since the chart is an indicator, it must be selected from the "Controls" palette. The chart is a great way to display the data in real time but the data is not saved after the VI execution ends. Saving data is covered in Section 1.4.3, Writing to a Measurement File.

Often it is convenient to see your data displayed while the experiment is being performed to verify the validity of the data. That way, if something goes wrong with the data measurement, you can perform the experiment again while everything is set up. Charts are most suitable for slow to moderate sampling rates and with single point sampling.

Steps 24-26: Plotting Data in a Chart

In the following steps, the program will be modified to plot data continuously as it is acquired. For this to be accomplished, a Waveform Chart will be added within the While Loop. One data point is acquired per channel every iteration and will be appended to the chart.

24. Place the cursor in the front panel, select **Express>>Graph Indicators>>Waveform Chart** and drag the icon to the front panel. You may also select the waveform chart in the "Modern" subpalette under **Modern>>Graph>>Waveform Chart**.

Note: Two data lines will eventually be displayed on the chart in the current example since we have two channels of analog input data, yet a legend for only one line, Plot 0, is displayed at the upper right corner. Perform step 25 to add another plot legend.

25. Place the "Position" (arrow) cursor over the top middle blue handle of the plot legend and drag the boundary up one more plot legend to add Plot 1.

Tip: Locate the chart terminal in the block diagram by double-clicking on the chart in the front panel. A black border will temporarily appear around the chart terminal. The method of double-clicking on any of the controls and indicators in the front panel may be used to locate the corresponding terminals in the block diagram. Likewise, double-clicking on terminals in the block diagram may be used to locate corresponding controls and indicators in the front panel.

26. Using the "Position" (arrow) cursor, drag the chart terminal in the block diagram within the While Loop border and above the "Write to Measurement File" subVI, as shown in Fig. 1.4.1.

Additional Information on Waveform Charts

There are three modes to update the data displayed on the waveform chart: strip chart, scope chart, and sweep chart. The strip chart mode (default mode) continuously appends data to the right end of a curve. The chart area displays an array of data stored in the chart history. When all of the data points that are held in the chart history are plotted, the curve moves to the left as new data points are added. In the scope chart mode, the data are displayed in the plot area until all of the points in the chart history are plotted and then clears the plot and starts over. The sweep chart mode is similar to the scope chart mode except new data overwrites the oldest data displayed instead of clearing the entire plot.

The update mode can be changed by right-clicking on the plot and selecting **Advanced>>Update Mode** from the shortcut menu as shown in Fig. 1.4.11.

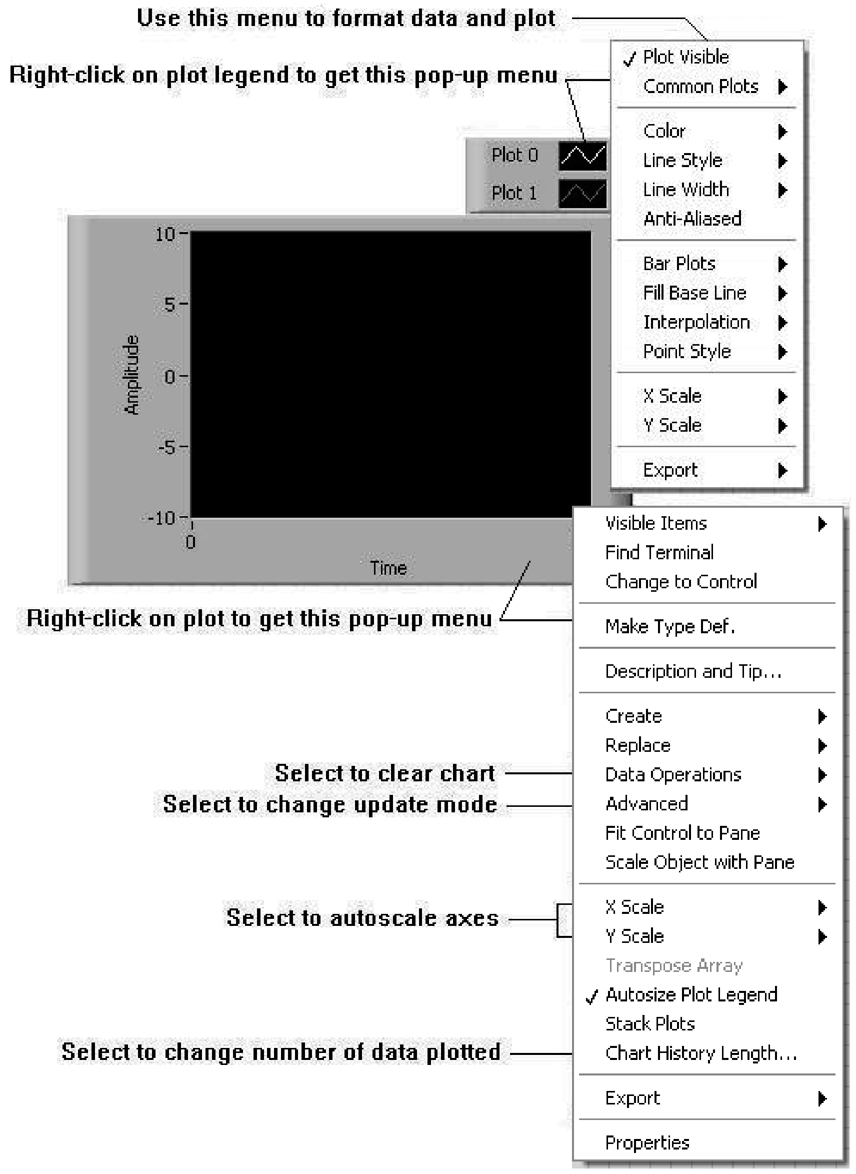


Figure 1.4.11 Waveform chart options

The chart can be modified in a number of ways to improve the viewing of the data as shown by the shortcut menus in Fig. 1.4.11. If you right-click the mouse over the plot legend (top right area of the plot containing the line), the line color, style, and width can be changed among other options on the shortcut menu. Furthermore, a number of options to modify the chart are available by right clicking on the panel in the chart where the data will be displayed. Common modifications include auto-scaling the x- and y-axes, clearing the chart (Data Operations>>Clear Chart), and changing the number of data points plotted through the chart history length. These options are identified in Fig. 1.4.11. If the chart is not an appropriate size for your application, you can use the "Position" (arrow) cursor to resize the chart with the blue handles. Use the "Tools" palette "Edit Text" (letter A) cursor if you want to rename the chart to an appropriate name for your data.

1.4.7 Waveform Graph

Overview of Waveform Graphs

It's a good idea to plot the entire set of data at the end of an experiment to see if anything needs to be repeated. The waveform graph plots one or more arrays of data all at once, unlike the waveform chart, which continually updates the plot. The graph is a great way to display the data after a test, but the data is not saved after the VI execution ends. Saving data is covered in Section 1.4.3, Writing to a Measurement File.

A single plot consists of an array of data in row format. Multiple plots require a 2-D array of data as input, where each plot is a row in the 2-D array. If the graph input consists only of a row of data (Y values), it is assumed that the initial X value, X_0, is zero and the spacing between X values, ΔX, is 1. Other values for the initial value of X and the spacing between X values may also be specified by building a waveform. However, for the purposes of this example VI, only the Y data will be plotted.

Steps 27-28: Graphing Data

In the following steps, the program will be modified to plot the entire set of data at the end of the experiment. This will be accomplished by adding a Waveform Graph outside of the While Loop.

27. Place the cursor in the front panel, select **Express>>Graph Indicators>>Waveform Graph** and drag the icon to the right of the chart. You may also select the waveform graph in the "Modern" subpalette under **Modern>>Graph>>Waveform Graph**.

Tip: The graph may be resized by selecting one of the blue handles using the "Position" (arrow) cursor and dragging the graph to the desired size.

28. Place the graph terminal in the block diagram to the right of the While Loop border but within the Case Structure as shown in Fig. 1.4.1.

Additional Information on Waveform Graphs

The waveform graph has options that are similar to the waveform chart. Right click on the graph to see the available options as shown in Fig. 1.4.12. For example, select "Data Operations" to clear the graph and select "X Scale" and "Y Scale" to auto-scale the axes.

The waveform graph has a useful palette to examine the data in more detail after the experiment. Right click on the graph and select **Visible Items>>Graph Palette** in the shortcut menu. The "Graph Palette" is identified at the bottom left of the graph in Fig. 1.4.12. The first of the three buttons is the "Cursor Movement Tool." This tool permits the cursor to move through the data on the plot, which can be used in conjunction with the "Cursor Legend" to obtain data values. The last button is the "Panning Tool," which grabs the plot and allows it to be moved. The middle button, "Zoom," can be used to magnify the data, zoom in or out, and isolate a small band of data to view.

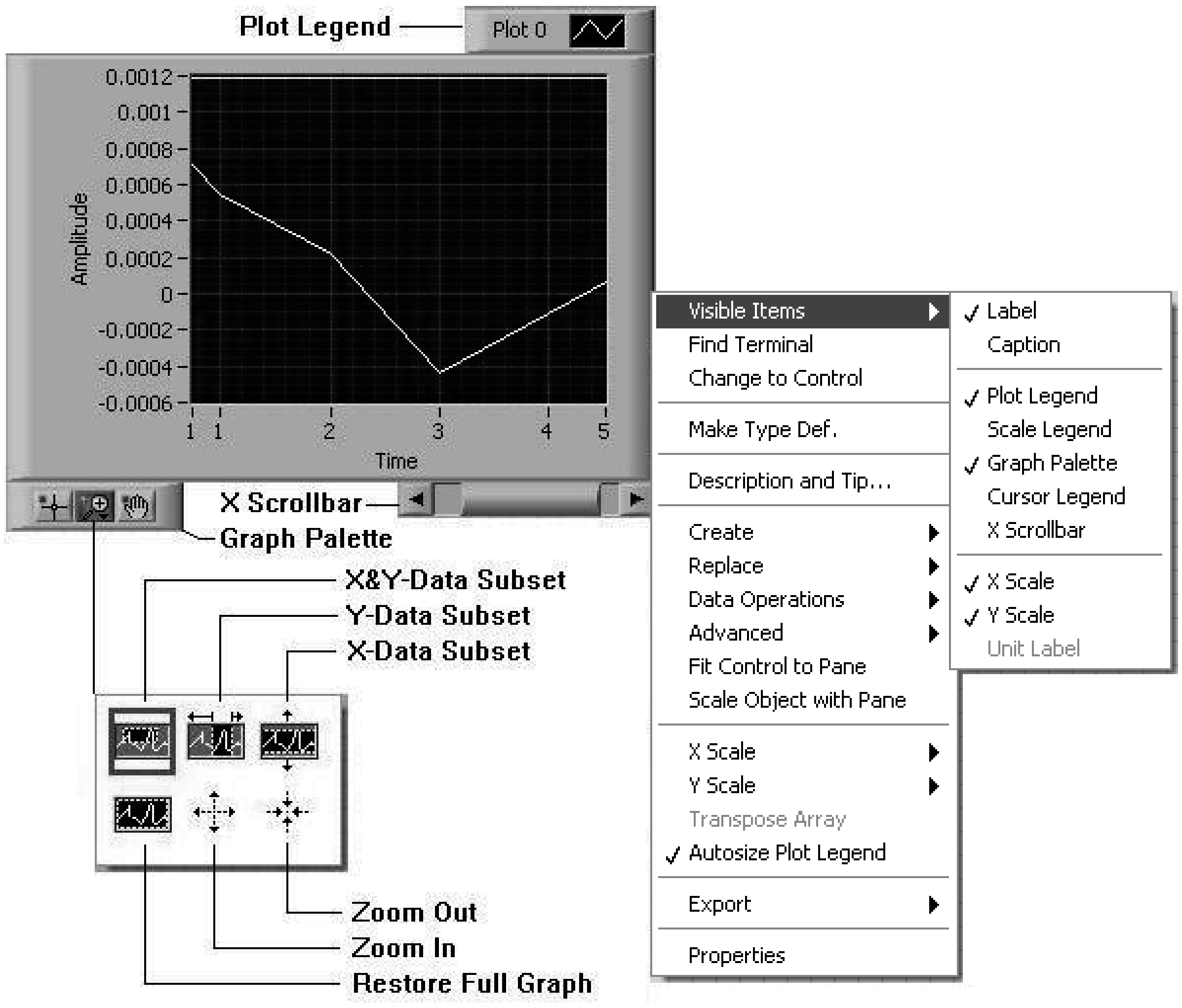


Figure 1.4.12 Waveform graph options

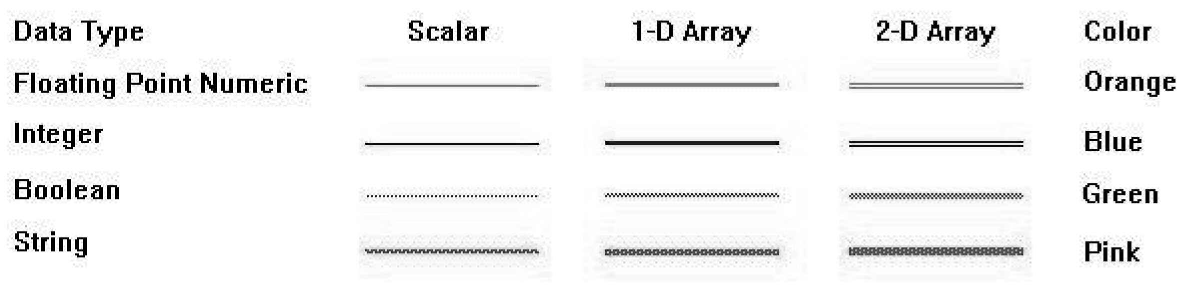
The subpalette at the bottom of Fig. 1.4.12 is displayed when the "Zoom" button is selected using the "Operate Value" (pointing hand) cursor. The top three options on the subpalette show that portions of the data can be enlarged by selecting one of the zoom buttons to expand the data and then dragging the cursor over the data of interest in your graph. You may also zoom in or out about a point with the lower right buttons. Finally, the lower left button auto-scales the x- and y-axes, which restores the plot to the original size.

The "X Scrollbar" is a convenient feature when you zoom in on the plot data. The "X Scrollbar" allows you to scroll through detailed (zoomed in) portions of data that are too magnified to fit on a single graph panel. Add the "X Scrollbar" by selecting **Visible Items>>X Scrollbar** from the shortcut menu.

1.4.8 Data Types

Overview of Data Types

LabVIEW operates under the principle of data flow. This means that a function executes only after it has received all required inputs regardless of its position in the block diagram.



Data Type	Scalar	1-D Array	2-D Array	Color
Floating Point Numeric	─	━	═	Orange
Integer	─	━	═	Blue
Boolean	┈	┉	░	Green
String	〰	▒	▓	Pink

Figure 1.4.13 Wire styles and colors for different LabVIEW data types

Dataflow is accomplished by LabVIEW via data paths, called wires, which connect nodes and terminals in the block diagram. A wire can emanate from one source terminal to one or more sink terminals. The wire's style, thickness, and color indicate the data type it carries. Examples of common wire types are shown in Fig. 1.4.13. A thin wire is displayed if the wire carries a single element, or scalar. A thicker wire will be displayed if the wire carries a 1-D array, for example, a row of data elements. This is typical of a number of data points taken over time on a single measurement channel. Depending on the data type, either an even thicker line or pair of lines will be displayed if the wire carries a 2-D array. An example of a 2-D array would be a number of data points taken over time on multiple measurement channels forming an array with each row representing a different channel. In Fig. 1.4.13, the floating point numeric and the integer data types have a pair of lines and the Boolean and string data types have thick lines.

Steps 29-30: Wiring Block Diagram Objects

In the following steps, you will learn wiring techniques and then wire nodes and terminals within the block diagram.

Note: Fig. 1.4.14 shows the block diagram objects that should appear in the VI you are creating. The exact size and position of the objects are not critical but the type of objects and general placement is. Likewise, your VI should have a "Run" button with an unbroken arrow, that is, the VI is ready to run. If your VI does not generally appear as shown in Fig. 1.4.14, repeat any appropriate step from Steps 1-28 to correct the error.

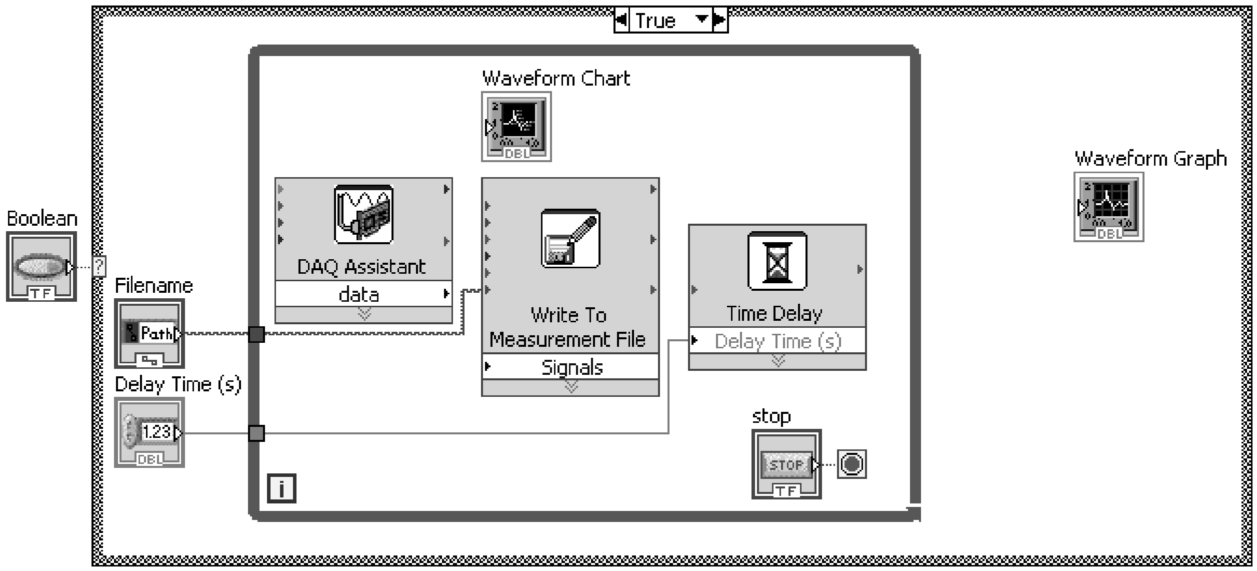


Figure 1.4.14 Intermediate stage of the example VI to continuously measure analog input data based on steps 1-28

Tip: An important aspect of wiring is to connect the correct terminals, and there are a number of aids to help. Wiring is performed using the "Connect Wire" (solder spool) cursor. If this cursor is placed over a terminal, the terminal will highlight the data type color. A tip strip, which is the terminal identifier, also appears. You may also show the object's terminals by opening the "Context Help" window. If you haven't already done so, open this window using the "Help" pull-down menu and selecting **Help>>Show Context Help** or pressing <Ctrl H> on the keyboard. The "Control Help" window shows all of the object's terminals and, when the "Connect Wire" (solder spool) cursor is placed over a terminal in the block diagram, the corresponding terminal in the "Context Help" window blinks.

Tip: There are a number of tips to wiring two terminals together. Wiring may begin from the source terminal to the sink terminal or vice versa. Place the "Connect Wire" cursor over the desired terminal, left click the mouse to tack the wire to the terminal, move the mouse to the second terminal, and left click once again on the blinking receiving terminal. You do not need to hold down the left mouse button as you wire although you can tack down the wire at any point by left clicking on the mouse. As you proceed through this example, you may also notice that LabVIEW will automatically wire objects that have just been placed in the block diagram if the terminal of a close object has a similar name and data type as the one placed next to it. The automatic wiring feature may be disabled by pressing the space bar.

29. Wire terminals together as shown in Fig. 1.4.15. Wire the data output terminal from the DAQ Assistant to the waveform chart and the signals input terminal of the "Write to Measurement File" Express VI. Notice that the waveform chart changes to the dynamic data type when the wire is connected. You can start wiring from a wire that already connects two terminals.

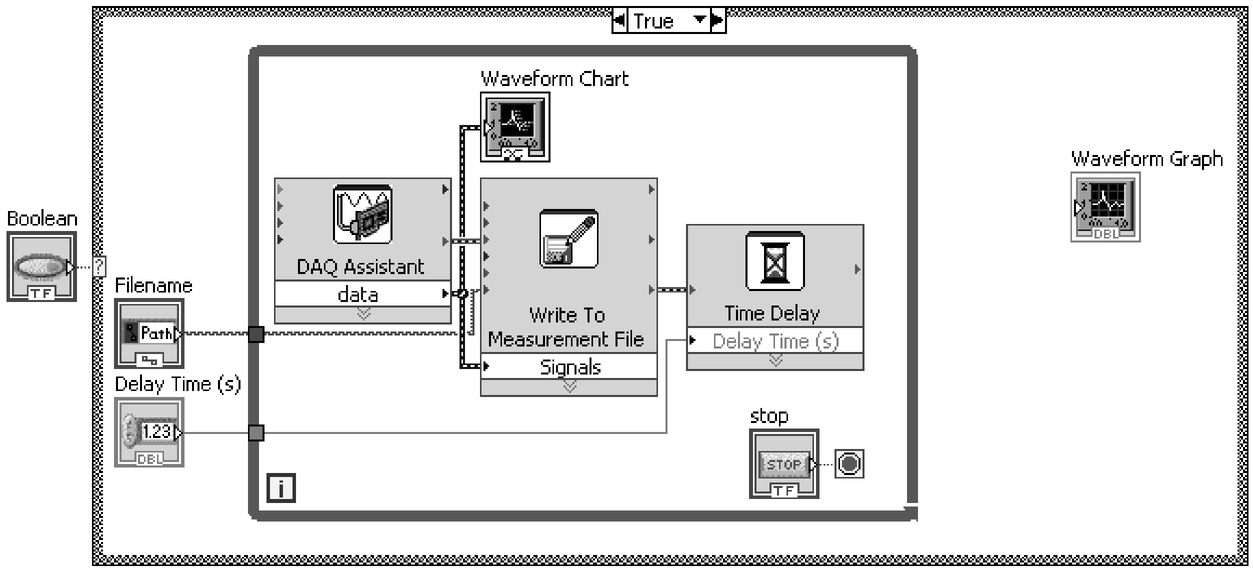


Figure 1.4.15 Example VI to continuously measure analog input data showing wiring

Note: The Express VIs also contain terminals that pass along information about errors that may have occurred before or during the execution of the VI. If an error occurs during the execution of the Express VI, an error message will be generated and pass through the "error out" terminal. Any error that occurs before the execution of the Express VI is passed along from the "error in" terminal to the "error out" terminal. The error message can be displayed with a dialog function, usually at the end of the execution of the VI. Connecting the error terminals between the Express VIs also forces the flow of data and controls the order of execution within the VI.

30. Wire the error input and output terminals between the Express VIs as shown in Fig. 1.4.15.

Additional Information on Data Types

There are a number of different data types. LabVIEW differentiates data types by color in the block diagram. A sample of common data types used by LabVIEW is shown in Table 1.4.1. Each data type will be briefly described.

- A floating point numeric contains a decimal point and may be a real or complex, positive or negative value. It may have a single precision (32 bit), double precision (default precision, 64 bit), or extended precision (128 bit) representation.
- Integers are whole numbers and may be positive only (unsigned) or both positive and negative (signed). Integers may be represented as a byte (8 bit), word (16 bit), long (32 bit), or quad (64 bit) integer.
- The Boolean data type contains two values: logical TRUE and FALSE.
- A string is a sequence of ASCII characters, most commonly alphanumeric characters. For example, a data measurement stored in binary format must be converted to a string of human-recognizable numbers to store in a text or spreadsheet file.
- A cluster is a group of data elements of mixed type. An important cluster is the error cluster, which groups the error status (Boolean, that is, there is an error-TRUE or no error-FALSE), the error code (integer), and the source of the error (string).
- The path data type contains the location of a file or directory.
- Most Express VIs use the dynamic data type, which includes the data and its attributes, such as the signal name or time the data was taken. Other functions and subVIs do not accept this data type.
- The waveform data type contains not only data but the start time and uniform time spacing of the data. A common example of the waveform data type is when you use it as an input to a graph.

Data Type	Color	Representation	Default Value
Floating Point Numeric	Orange	SGL-Single Precision DBL-Double Precision EXT-Extended Precision CSG-Complex Single CDB-Complex Double CXT-Complex Extended	0.0 0.0+0.0i
Integer	Blue	8-bit, 16-bit, 32-bit, or 64-bit Signed or Unsigned	0
Boolean	Green		False
String	Pink		Empty String
Cluster	Brown-Numeric Pink-Non-numeric Yellow-Error code		
Path	Teal Green		Empty Path
Dynamic	Dark Blue		
Waveform	Brown		

Table 1.4.1 Common LabVIEW Data Types

LabVIEW allows different data types to be used in many functions by coercing one of the data types. For example, it can add an integer to a floating point numeric. Data that is coerced will have a small coercion dot placed at the function’s input terminal.

If an attempt is made to wire a source and sink terminal that are not compatible, a dashed line will appear with an X, which is called a broken wire. Examples include attempting to wire two controls or indicators together, wiring a terminal of one data type to a terminal of a different data type, or wiring a scalar to an array. The Run button on the block diagram toolbar will appear as a broken arrow (see Fig. 1.3.4) if there are any broken wires. The error associated with the broken wire can be displayed by left clicking on the “Run” button. You may also use the keyboard shortcut <Ctrl B> to remove broken wires.

1.4.9 Converting Dynamic Data

Overview of Dynamic Data

Since Express VIs generally use the dynamic data type but other functions and subVIs don’t, a means to convert from the dynamic data type to other data types and vice versa is provided by LabVIEW. The “Convert from Dynamic Data” Express VI converts dynamic