Debugging and Handling Errors: Detailed Explanation

If LabVIEW cannot run your VI, it informs you by changing the **Run Arrow** to a broken icon and the **Error List** window lists the specific reasons why the VI is broken. The second type of bug is typically harder to track down, but LabVIEW features several tools to help you watch your code as it executes, which makes the process much easier.

Fixing Broken VIs



If a VI does not run, it is a broken, or non-executable, VI. The **Run Arrow** button appears broken when the VI you are creating or editing contains errors.

Generally, this means that a required input is not wired, or a wire is broken. Press the broken **Run Arrow** button to access the **Error List** window. The **Error List** window lists each error and describes the problem. You can double-click an error to go directly to the error.

Finding Causes for Broken VIs

Warnings do not prevent you from running a VI. They are designed to help you avoid potential problems in VIs. Errors, however, can break a VI. You must resolve any errors before you can run the VI.

Click the broken **Run** button or select **View >> Error List** to find out why a VI is broken. The Error list window lists all the errors. The Items with errors section lists the names of all items in memory, such as VIs and project libraries that have errors. If two or more items have the same name, this section shows the specific application instance for each item.

The errors and warnings section lists the errors and warnings for the VI you select in the Items with errors section. The **Details** section describes the errors and in some cases recommends how to correct the errors. Click the **Help** button to display a topic in the LabVIEW Help that describes the error in detail and includes step-by-step instructions for correcting the error.

Click the **Show Error** button or double-click the error description to highlight the area on the block diagram or front panel that contains the error.

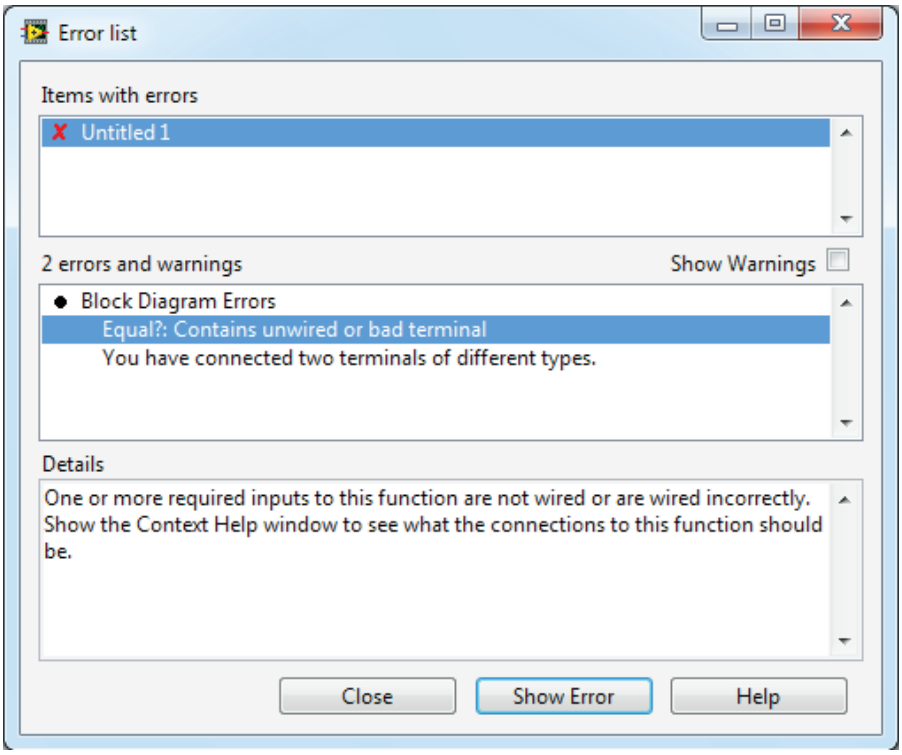


Figure 1. Example of the Error List Dialog Box

Common Causes of Broken VIs

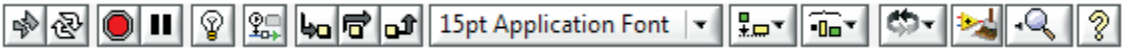
The following list contains common reasons why a VI is broken while you edit it:

- The block diagram contains a broken wire because of a mismatch of data types or a loose, unconnected end. Refer to the Correcting Broken Wires topic of the LabVIEW Help for information about correcting broken wires.
- A required block diagram terminal is unwired. Refer to the Using Wires to Link Block Diagram Objects topic of the LabVIEW Help for information about setting required inputs and outputs.
- A subVI is broken or you edited its connector pane after you placed its icon on the block diagram of the VI.

Fixing Incorrect Behavior

If your VI runs but produces incorrect results, then you have a functional error in your code. LabVIEW has several tools to help you analyze where your VI is not behaving as expected.

Block Diagram Toolbar



When you run a VI, buttons appear on the block diagram toolbar that you can use to debug the VI. The following toolbar appears on the block diagram.



Click the **Highlight Execution** button to display an animation of the block diagram execution when you run the VI. Notice the flow of data through the block diagram. Click the button again to disable execution highlighting.

Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Use execution highlighting with single-stepping to see how data values move from node to node through a VI.

Note: *Execution highlighting greatly reduces the speed at which the VI runs.*

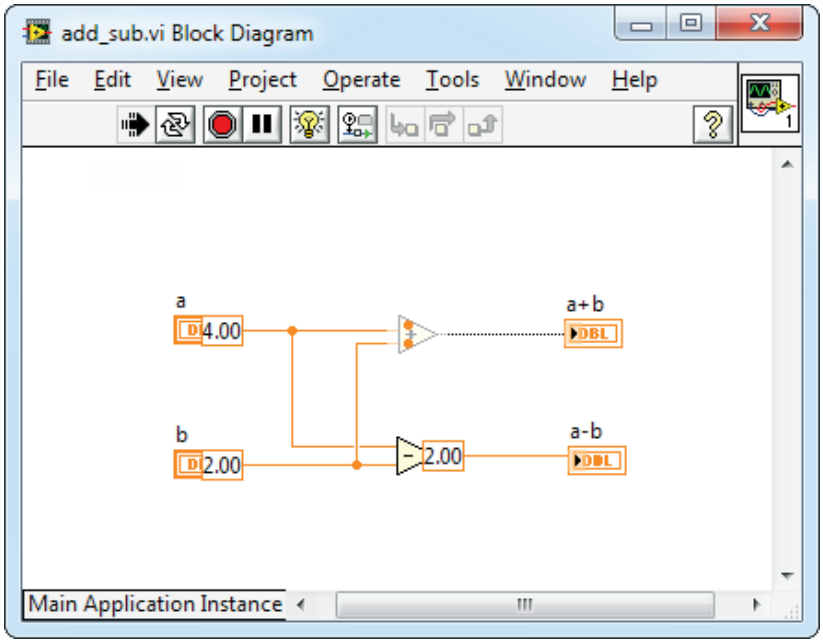


Figure 2. Example of Execution Highlighting in Use



Click the **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on the wire you can immediately retain the most recent value of the data that passed through the wire. You must successfully run the VI at least once before you can retain the wire values.



Click the **Step Into** button to open a node and pause. When you click the **Step Into** button again, it executes the first action and pauses at the next action of the subVI or structure. You also can press the <Ctrl> and down arrow keys. Single-stepping through a VI steps through the VI node by node. Each node blinks to denote when it is ready to execute.



Click the **Step Over** button to execute a node and pause at the next node. You also can press the <Ctrl> and right arrow keys. By stepping over the node, you execute the node without single-stepping through the node.



Click the **Step Out** button to finish executing the current node and pause. When the VI finishes executing, the **Step Out** button is dimmed. You also can press the <Ctrl> and up arrow keys. By stepping out of a node, you complete single-stepping through the node and navigate to the next node.



The **Warning** button appears if a VI includes a warning and you placed a checkmark in the Show Warnings checkbox in the Error List window. A warning indicates there is a potential problem with the block diagram, but it does not stop the VI from running.



Use the **Probe** tool to check intermediate values on a wire as a VI runs. Also use the Probe tool if you have a complicated block diagram with a series of operations, any one of which might return incorrect data. Use the Probe tool with execution highlighting, single-stepping, and breakpoints to determine if and where data is incorrect. If data is available, the probe immediately updates and displays the data in the **Probe Watch Window** during execution highlighting, single-stepping, or when you pause at a breakpoint. When execution pauses at a node because of single-stepping or a breakpoint, you also can probe the wire that just executed to see the value that flowed through that wire.

Handling Errors in LabVIEW

No matter how confident you are in the VI you create, you cannot predict every problem a user can encounter. Without a mechanism to check for errors, you know only that the VI does not work properly. Error checking tells you why and where errors occur.

Automatic Error Handling

Each error has a numeric code and a corresponding error message.

By default, LabVIEW automatically handles any error when a VI runs by suspending execution, highlighting the subVI or function where the error occurred, and displaying an error dialog box.

To disable automatic error handling for the current VI, select **File >> VI Properties** and select Execution from the **Category** pull-down menu. To disable automatic error handling for any new, blank VIs you create, select **Tools >> Options** and select **Block Diagram** from the **Category** list. To disable automatic error handling for a subVI or function within a VI, wire its error out parameter to the error in parameter of another subVI or function or to an **error out** indicator.

Manual Error Handling

You can choose other error handling methods. For example, if an I/O VI on the block diagram times out, you might not want the entire application to stop and display an error dialog box. You also might want the VI to retry for a certain period of time. In LabVIEW, you can make these error handling decisions on the block diagram of the VI.

Use the LabVIEW error handling VIs and functions on the **Dialog & User Interface** palette and the **error in and error out** parameters of most VIs and functions to manage errors. For example, if LabVIEW encounters an error, you can display the error message in different kinds of dialog boxes. Use error handling with the debugging tools to find and manage errors.

When you perform any kind of input and output (I/O), consider the possibility that errors might occur. Almost all I/O functions return error information. Include error checking in VIs, especially for I/O operations (file, serial, instrumentation, data acquisition, and communication), and provide a mechanism to handle errors appropriately.

Use the LabVIEW error handling VIs, functions, and parameters to manage errors. Or you can fix the error programmatically and then erase the error by wiring the **error out** output of the subVI or function to the error in input of the Clear Errors VI. Use error handling with the debugging tools to find and manage errors.

Error Clusters

Use the error cluster controls and indicators to create error inputs and outputs in subVIs.

The error in and error out clusters include the following components of information:

- **status** is a Boolean value that reports TRUE if an error occurred.
- **code** is a 32-bit signed integer that identifies the error numerically. A nonzero error code coupled with a status of FALSE signals a warning rather than an error.
- **source** is a string that identifies where the error occurred.

Error handling in LabVIEW follows the dataflow model. Just as data values flow through a VI, so can error information. Wire the error information from the beginning of the VI to the end. Include an error handler VI, such as the Simple Error Handler.vi, at the end of the VI to determine if the VI ran without errors. Use the error in and error out clusters in each VI you use or build to pass the error information through the VI.

As the VI runs, LabVIEW tests for errors at each execution node. If LabVIEW does not find any errors, the node executes normally. If LabVIEW detects an error, the node passes the error to the next node without executing that part of the code. The next node does the same thing, and so on. At the end of the execution flow, LabVIEW reports the error.

Explain Error

When an error occurs, right-click within the cluster border and select **Explain Error** from the shortcut menu to open the Explain Error dialog box. The **Explain Error** dialog box contains information about the error. The shortcut menu includes an Explain Warning option if the VI contains warnings but no errors.

You also can access the **Explain Error** dialog box from the **Help >> Explain Error** menu.

Using While Loops With Error Handling

You can wire an error cluster to the conditional terminal of a While Loop or a For Loop with a conditional terminal to stop the iteration of the loop. If you wire the error cluster to the conditional terminal, only the TRUE or FALSE value of the status parameter of the error cluster passes to the terminal. If an error occurs, the loop stops. In a For Loop with a conditional terminal, you also must wire a value to the count terminal or auto-index an input array to set a maximum number of iterations. The For Loop executes until an error occurs or until the number of set iterations completes. If you wire an error cluster to the conditional terminal, the shortcut menu items Stop if True and Continue if True change to Stop on Error and Continue while Error.

In **Figure 3**, the error cluster and a stop button are used together to determine when to stop the loop. This is the recommended method for stopping most loops because the loop stops when you ask it to, or when something goes wrong (and produces and error), whichever comes first.

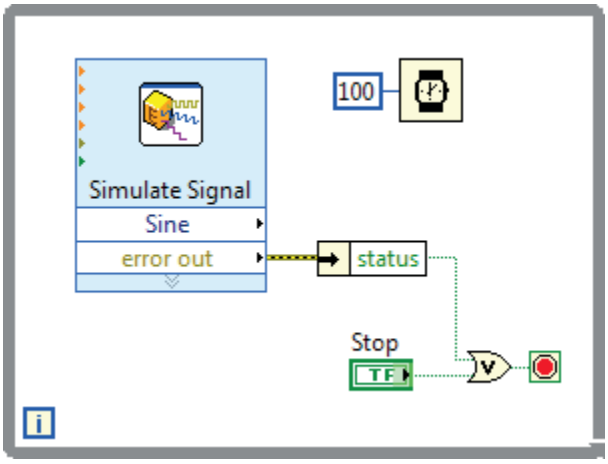


Figure 3. Stopping a While Loop

©2011 National Instruments. All rights reserved. LabVIEW, National Instruments, NI, and ni.com are trademarks of National Instruments.
Other product and company names listed are trademarks or trade names of their respective companies.