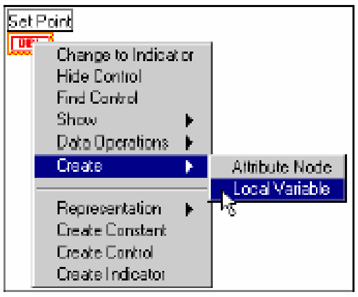
A. Local Variables

Up until now, you have read data from or updated a front panel object using its terminal on the block diagram. However, a front panel object has only one terminal on the block diagram, and you may need to update or read a front panel object from several locations on the diagram. Using local variables, you can access front panel objects in several places and pass data between structures that cannot be connected by a wire.

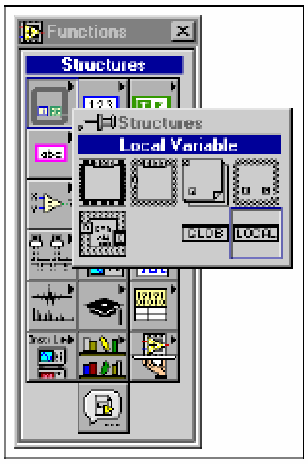
Creating Local Variables

There are two ways to create local variables on the block diagram. If you have already created a front panel object, you can create a local variable by popping up on the object or its terminal and selecting **Create » Local Variable** from the pop-up menu. You can use this method on the front panel or the block diagram. A local variable icon for the front panel object appears next to the terminal on the diagram. When you pop up on a terminal to create a local variable, the local variable refers to the object you popped up on to create the icon.



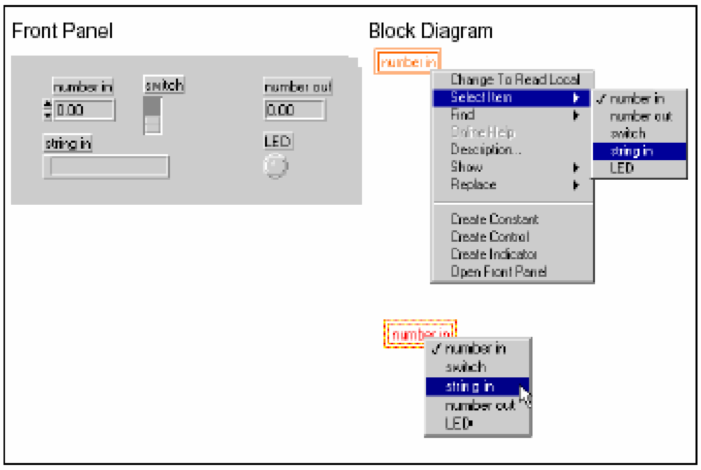
Local variable node

Another way to create a local variable is to select **Local Variable** from the **Structures** palette. A local variable node appears on the block diagram. The node shown at left appears if none of the controls or indicators on the front panel have an owned label. Otherwise, the local variable appears with the name of the first front panel object you created. You can select the front panel object you want to access by popping up on the variable node and selecting an object from the **Select Item** menu. This menu lists the owned labels for the front panel controls and indicators. *Thus, you should always label your front panel controls and indicators with descriptive names, using owned labels.*



Local variable icon with "number in" selected

For example, if the first object you create on the front panel is labeled "number in," the local variable icon appears as shown at left. After you place the local variable on the block diagram, you can select the appropriate front panel object by either clicking on the variable using the Operating tool, or popping up on it and choosing the front panel object from the **Select Item** menu.

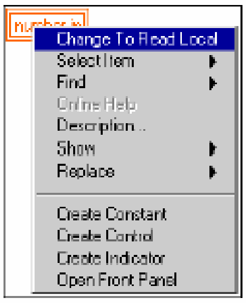


Read Locals and Write Locals

You can either read data from a local variable or write data to it. After you place the local variable on your diagram, you must decide how you will use it.

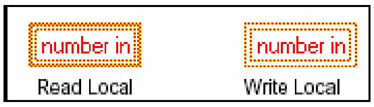
By default, a local variable assumes that it will *receive* data. Thus, this kind of local variable acts like an indicator and is a *write local*. When you write new data into the local variable, the associated front panel control or indicator updates to contain the new data.

You can change the configuration of a local variable so that it acts as a data source, or a *read local*. To do this, pop up on the variable and select **Change To Read Local**. On the diagram, a *read local* icon behaves just like a control. When this node executes on the diagram, your program reads the data in the associated front panel control or indicator.



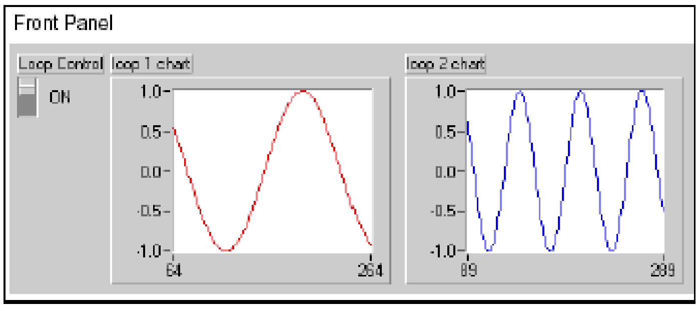
To change a *read local* to a *write local*, pop up on the variable and select **Change To Write Local**. The local variable will change its "data direction" so that it *receives* data instead of providing data.

On the diagram, you can visually distinguish read locals from write locals just as you distinguish controls from indicators. A read local has a thick border, emphasizing that it is a data source, similar to a control. A write local has a thin border, because it acts like a data sink, similar to an indicator. In the figure below, both local variables refer to the same item on the front panel.



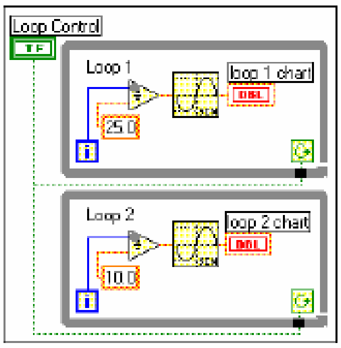
Local Variable Example

As an example of when you might need a local variable, consider how you would use a single front panel switch to control two parallel While Loops. Parallel While Loops are not connected by a wire, and they execute simultaneously. First, we will study two unsuccessful methods using one terminal for the switch on the block diagram (Methods 1 and 2). Then, we will show how a local variable accomplishes the task (Method 3). The front panel of this example VI appears below.



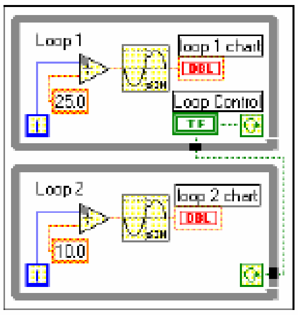
Method 1 (Incorrect)

As a first attempt, we place the Loop Control terminal outside of both loops and wire it to each conditional terminal. In this case, the Loop Control terminal is read only once, before either While Loop begins executing. Recall that this happens because LabVIEW is a *dataflow* language, and the status of the Boolean control is a data *input* to both loops. If a True is passed into the loops, the While Loops run indefinitely. Turning off the switch does not stop the VI because the switch is not read during the iteration of either loop. This solution does not work.



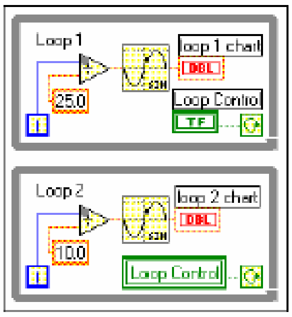
Method 2 (Incorrect)

Now we move the Loop Control terminal inside Loop 1 so that it is read in each iteration of Loop 1. Although Loop 1 terminates properly, there is also a problem with this approach. Loop 2 does not execute until it receives all its data inputs. Remember that Loop 1 does not pass data out of the loop until the loop stops. Thus, Loop 2 must wait for the final value of the Loop Control, available only after Loop 1 finishes. Therefore, the loops do not execute in parallel. Also, Loop 2 executes for only one iteration because its conditional terminal receives a False value from the Loop Control switch in Loop 1.



Method 3 (Correct)

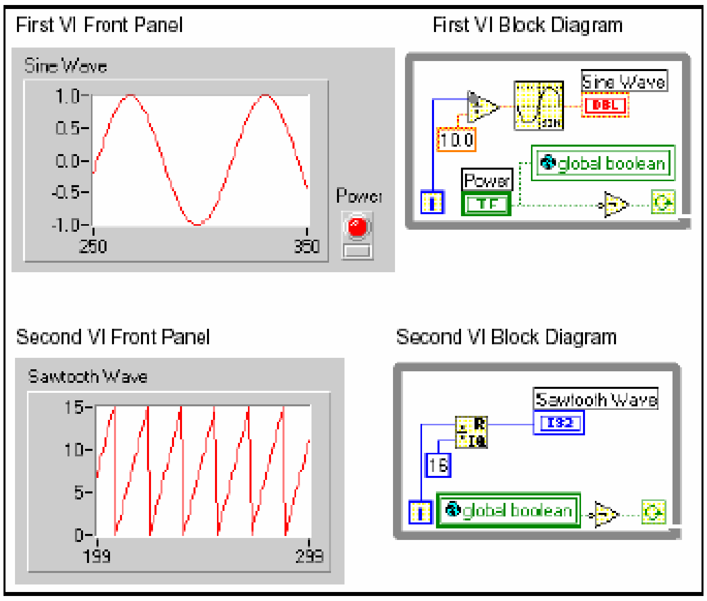
In this example, Loop 1 is again controlled by the Loop Control switch, but this time, Loop 2 reads a local variable associated with the switch. When you set the switch to False on the front panel, the switch terminal in Loop 1 writes a False value to the conditional terminal in Loop 1. Loop 2 reads the Loop Control local variable and writes a False to Loop 2's conditional terminal. Thus, the While Loops run in parallel and terminate simultaneously when the single front panel switch is turned off.



B. Global Variables

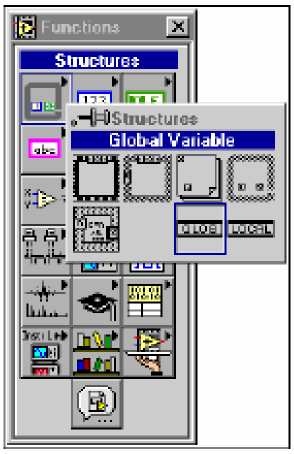
Recall that you can use local variables to access front panel objects at various locations in your block diagram. Those local variables are accessible only in that single VI. Suppose that you need to pass data between several VIs that run concurrently, or whose subVI icons cannot be connected by wires in your diagram. You can use a global variable to accomplish this. Global variables are similar to local variables, but instead of being limited to use in a single VI, global variables can pass values between several VIs.

Consider the following example. Suppose you have two VIs running simultaneously. Each VI writes a data point to a waveform chart. The first VI also contains a Boolean to terminate both VIs. Remember that when both loops were on a single diagram, you needed to use a local variable to terminate the loops. Now that each loop is in a separate VI, you must use a global variable to terminate the loops.



Creating Global Variables

Global variables are built-in LabVIEW objects. They appear as special VIs in the computer's memory. A global variable has a front panel where you can place controls and indicators of any type. However, a global variable has no block diagram. The front panel is simply a container from which you access data from other VIs.



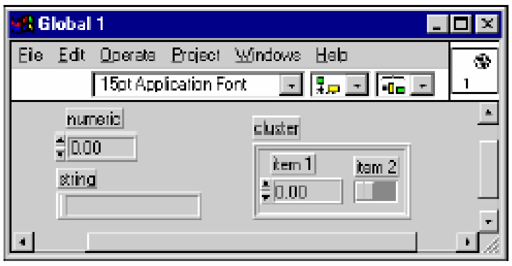
Global variable node

To create a global variable, select **Global Variable** from the **Structures** palette. A global variable node appears on the diagram. The icon for a global variable on the diagram is similar to a local variable icon, except that the border is always black and a small picture of a globe appears to the left of the global variable name.

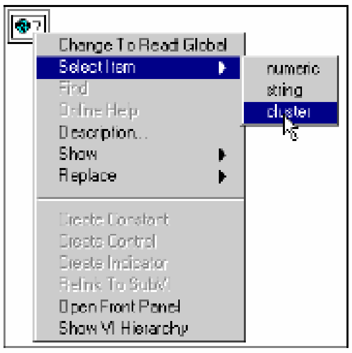
You open the panel of a global variable by double-clicking on the global variable node. You then place controls and indicators on the panel in the same way you place them on a standard VI's front panel.

Note ***You must label each control or indicator with an owned label because a global variable refers to an item by its name.***

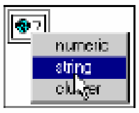
The following example shows a global variable front panel with three objects—a numeric, a string, and a cluster containing a digital and a Boolean control. Notice that the toolbar in the window does not show the same items as a normal Panel window.



After you finish placing objects on the global variable's panel, save the global variable and return to the block diagram of the original VI. You must then select the specific object in the global variable VI that you want to access. To select a specific object, pop up on the global variable node and select the object from the **Select Item** menu. This menu lists the owned labels for all the objects on the panel. Notice that you can also open the front panel containing the objects in the global variable from this pop-up menu.



Alternately, using the Operating tool, you can simply click on the node and choose the global variable you want to access.



Global variable node

After you select the specific global variable object that you want to access, the node changes from the figure shown at left to display the object you have chosen, such as a numeric.

Global variable node displaying a numeric

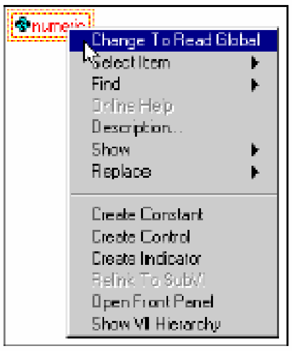
You may want to use this global variable in other VIs. Because a global variable is a special kind of VI, you can place it in other VIs using the **Select a VI...** option in the **Functions** palette. Then, pop up on the node to select the specific object in the global variable you want to access, as described above.

Note ***A global variable panel can contain references to many individual objects that are globally accessible. You do not need to create a separate global variable VI for each item you need to globally access.***

Read Globals and Write Globals

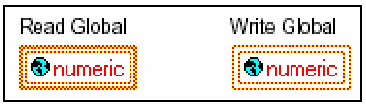
Like local variables, you can either read data from a global variable, or write data into it. By default, a global variable is *write global* when you create it. You can write a new value into the global variable; hence a write global acts like an indicator.

You can change the configuration of a global variable so that it acts as a data source, or a *read global*. To do this, pop up on the variable and select **Change To Read Global**. On the diagram, a *read global* icon behaves like a control. When this node executes on the diagram, your program reads the data in the associated front panel object.



To change a *read global* to a *write global*, pop up on the variable and select **Change To Write Global**. The global variable will change its "data direction" so that it *receives* data instead of providing data. When this node executes on the diagram, your program will send new data into the global variable.

On the diagram, you can visually distinguish read globals from write globals just as you distinguish controls from indicators. A read global has a thick border, emphasizing that it is a data source. A write global has a thin border, because it acts as a data sink. In the figure below, both global variables refer to the same item on the global variable's panel.



C. Important Advice about Local and Global Variables

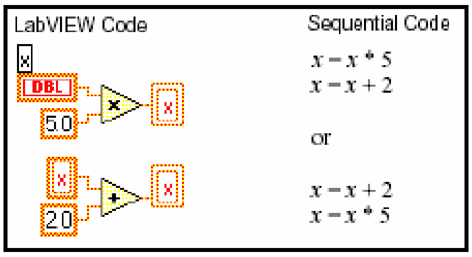
Initialize Local and Global Variables

You should verify that your local and global variables contain known data values before your program begins. Otherwise, the variables may contain data that causes your program to behave incorrectly.

If you do not initialize a variable before reading data from it, it will contain the current value of the control. The first time your program reads a global variable, it contains the default value of the object it reads, unless you have previously initialized the global variable.

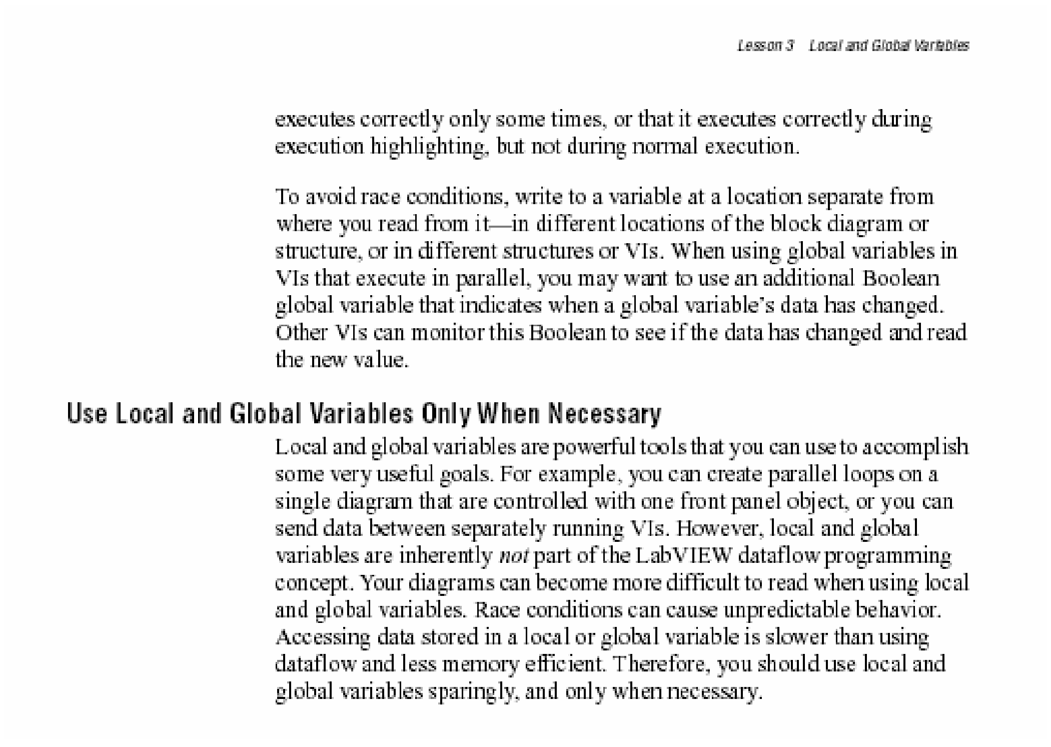
Race Conditions

Local and global variables in LabVIEW *do not* behave like local and global variables in conventional programming languages. Recall that LabVIEW is a *dataflow* language, not a sequential programming language. Overusing local and global variables can easily lead to unexpected behavior in your program because functions may not execute in the order you expect. Consider the simple local variable example below. The LabVIEW code appears next to the equivalent code of a sequential programming language.



When you execute the sequential code, the solution for a given value of *x* is clear because the statements execute from top to bottom. However, the LabVIEW code does *not* follow such a convention because LabVIEW is a *dataflow* language. Each node executes when all of its data is available. There are no data dependencies to guarantee the desired order of execution in the above diagram. In fact, there are several possible solutions, depending on how the VI compiles. You *cannot* assume that the code located at the bottom of the diagram executed after the code above it.

The above example illustrates what is known as a *race condition*. The result of your program depends on the order in which its instructions execute, and the code may not execute in the order you assume. If you use local and global variables, you may have a race condition if you notice that your code



executes correctly only some times, or that it executes correctly during execution highlighting, but not during normal execution.

To avoid race conditions, write to a variable at a location separate from where you read from it—in different locations of the block diagram or structure, or in different structures or VIs. When using global variables in VIs that execute in parallel, you may want to use an additional Boolean global variable that indicates when a global variable's data has changed. Other VIs can monitor this Boolean to see if the data has changed and read the new value.

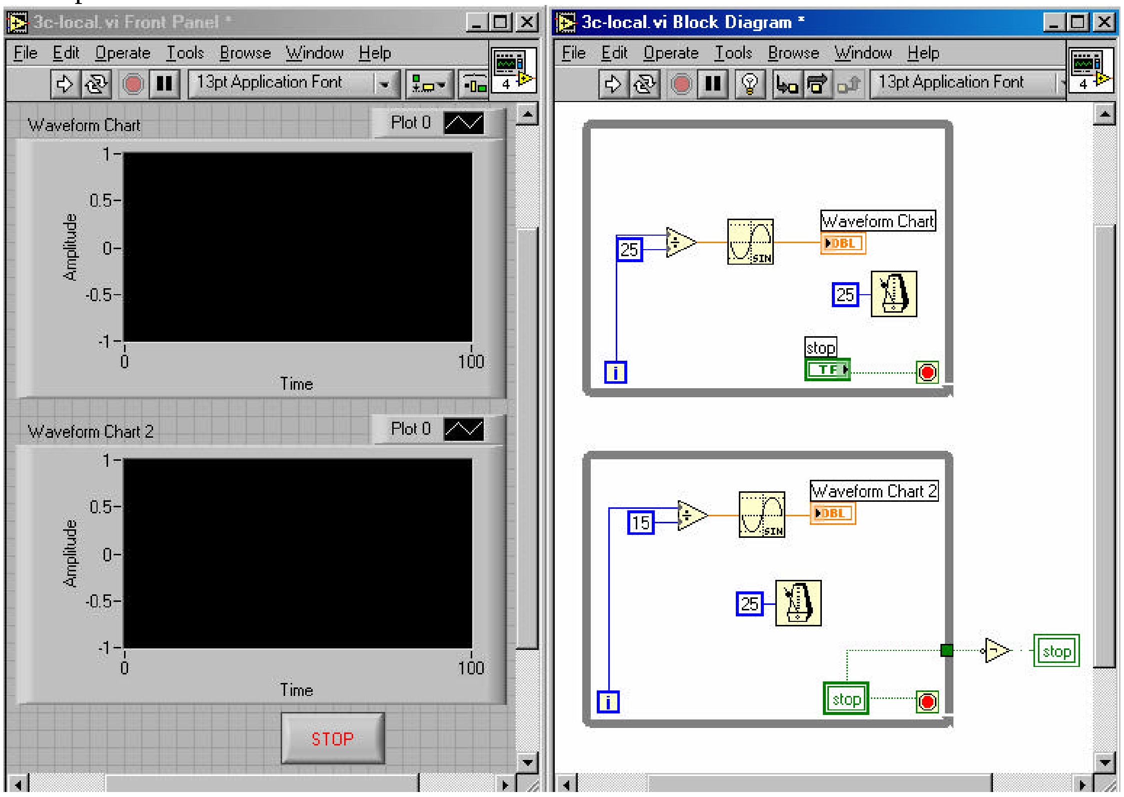
Use Local and Global Variables Only When Necessary

Local and global variables are powerful tools that you can use to accomplish some very useful goals. For example, you can create parallel loops on a single diagram that are controlled with one front panel object, or you can send data between separately running VIs. However, local and global variables are inherently *not* part of the LabVIEW dataflow programming concept. Your diagrams can become more difficult to read when using local and global variables. Race conditions can cause unpredictable behavior. Accessing data stored in a local or global variable is slower than using dataflow and less memory efficient. Therefore, you should use local and global variables sparingly, and only when necessary.

Locals and Globals Summary

- You can use global and local variables to access a given set of values throughout your LabVIEW application. These variables pass information between places in your application that cannot be connected by a wire.
- Local variables access front panel objects of the VI in which you placed that local variable.
- When you write to a local variable, you update its corresponding front panel control or indicator.
- When you read from a local variable, you read the current value of its corresponding front panel control or indicator.
- Global variables are built-in LabVIEW objects that pass data between VIs; they have front panels in which they store their data.
- You should always write a value to a global variable before reading from it, so that it has a known initial value when you access it.
- You should write to local and global variables at locations separate from where you read them to avoid race conditions.
- Use local and global variables only when necessary; overuse can cause slower execution and inefficient memory usage in your application.
- Local and global variables do not use dataflow; therefore, if used too frequently, they can make your diagrams more difficult for others to understand. Use locals and globals wisely.

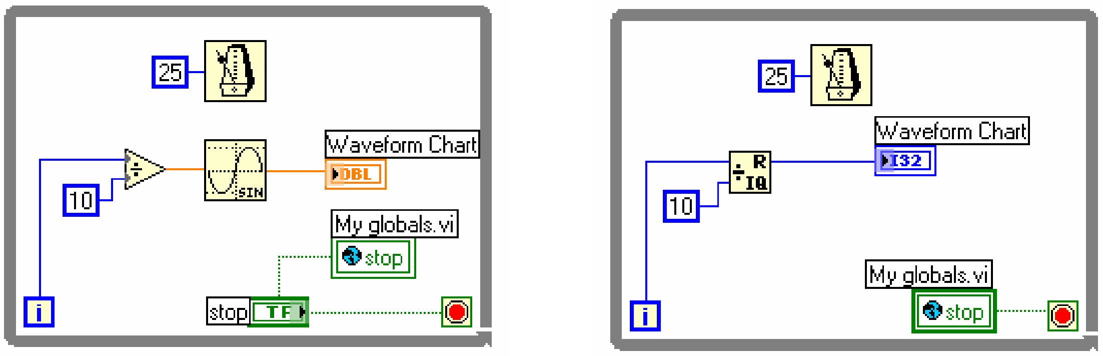
Example 1: Local variable



Example 2: Global Variable

VI – 1

VI -2


25
10
Waveform Chart
DBL
My globals.vi
stop
TF
i
I32
R
IQ