Using Local and Global Variables Carefully

Local and global variables are advanced LabVIEW concepts. They are inherently not part of the LabVIEW dataflow execution model. Block diagrams can become difficult to read when you use local and global variables, so you should use them carefully. Misusing local and global variables, such as using them instead of a connector pane or using them to access values in each frame of a sequence structure, can lead to unexpected behavior in VIs. Overusing local and global variables, such as using them to avoid long wires across the block diagram or using them instead of data flow, slows performance.

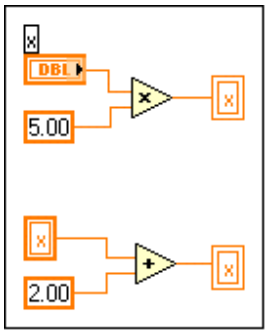
Initializing Local and Global Variables

Verify that the local and global variables contain known data values before the VI runs. Otherwise, the variables might contain data that cause the VI to behave incorrectly.

If you do not initialize the variable before the VI reads the variable for the first time, the variable contains the default value of the associated front panel object.

Race Conditions

A race condition occurs when two or more pieces of code that execute in parallel change the value of the same shared resource, typically a local or global variable. The following block diagram shows an example of a race condition.



The output of this VI depends on the order in which the operations run. Because there is no data dependency between the two operations, there is no way to determine which runs first. To avoid race conditions, do not write to the same variable you read from.

Memory Considerations when Using Local Variables

Local variables make copies of data buffers. When you read from a local variable, you create a new buffer for the data from its associated control.

If you use local variables to transfer large amounts of data from one place on the block diagram to another, you generally use more memory and, consequently, have slower execution speed than if you transfer data using a wire. If you need to store data during execution, consider using a shift register.

Memory Considerations when Using Global Variables

When you read from a global variable, LabVIEW creates a copy of the data stored in that global variable.

When you manipulate large arrays and strings, the time and memory required to manipulate global variables can be considerable. Manipulating global variables is especially inefficient when dealing with arrays because if you modify only a single array element, LabVIEW stores and modifies the entire array. If you read from the global variable in several places in an application, you create several memory buffers, which is inefficient and slows performance.