



1. Using Variables

Local versus Global Variables

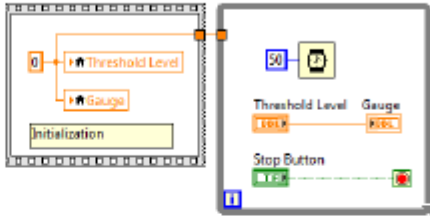
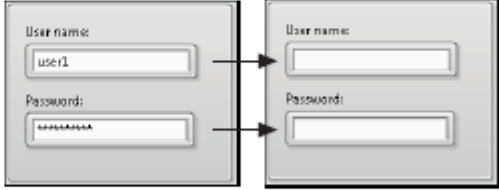
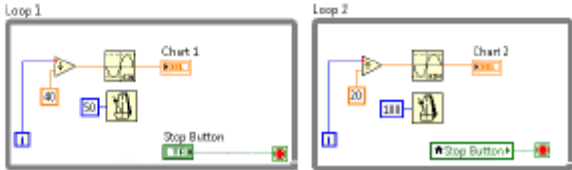
Local and global variables pass information between locations in the application that you cannot connect with a wire. The following table outlines the differences between local and global variables.

Local Variable	Global Variable
	
Store data in front panel controls and indicators.	Store data in special repositories that you can access from multiple VIs.
Use to access front panel objects from more than one location in a single VI.	Use to access and pass data among several VIs.
Has a one-to-one relationship with a control or indicator.	Can contain one or more variable data types.

When to Use Local Variables

The following table highlights use cases for local variables.

Table 1-1. Use Cases for Local Variables

<p>Initializing front panel controls or indicators.</p> <p>For example, you need to set data in a terminal in one location, but access the terminal from another location on the block diagram. Because each front panel object has only one block diagram terminal you can wire, you need to use variables.</p>	
<p>Writing to controls.</p> <p>For example, to clear a log in screen you need to modify the front panel control while the VI is running. You use variables to programmatically write empty strings to the log in controls after a user enters their information.</p>	
<p>Keeping parallel loops parallel.</p> <p>For example, if you pass the data using a wire, the loops are no longer parallel. Variables let you pass data between multiple loops without creating a data dependency.</p>	

Drawbacks of Variables

Local and global variables are inherently not part of the LabVIEW dataflow execution model. The following table highlights some of the potential programming issues with using variables.

Table 1-2. Potential Programming Issues with Variables

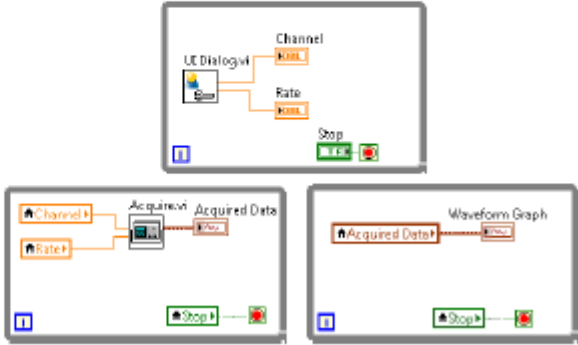
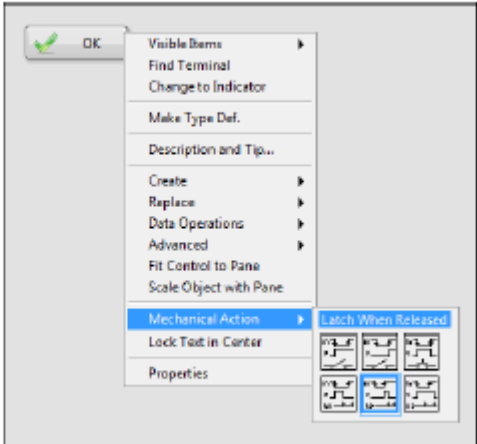
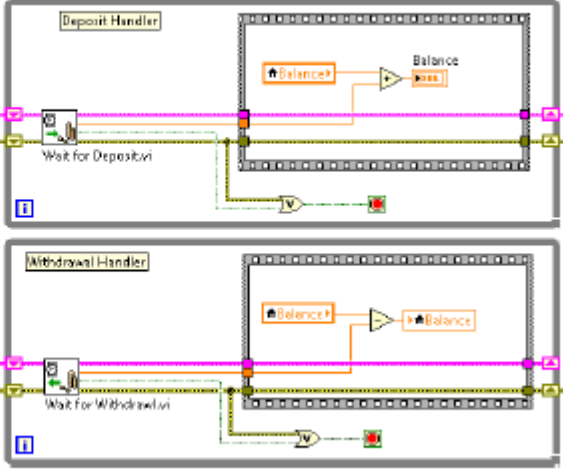

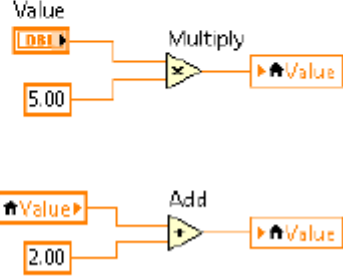
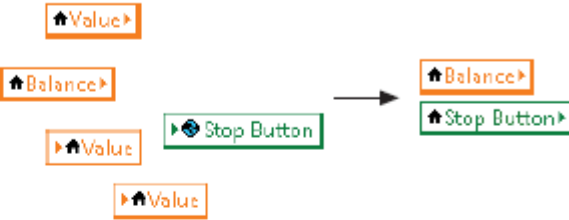
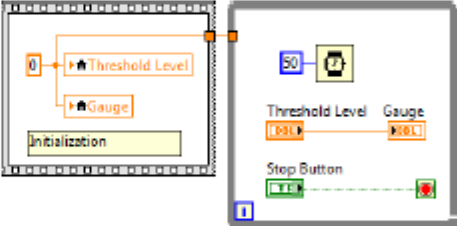
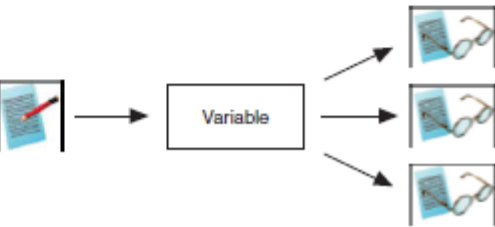
<p>Less readable block diagram code</p> <p>Because variables break the dataflow model, you cannot use wires to follow the flow of data.</p>	
<p>Limited Boolean mechanical actions</p> <p>Boolean controls associated with variables must use a switch mechanical action. If a Boolean control has an associated local variable, it cannot use a latch mechanical action because the first local variable to read the Boolean control with latch action would reset its value to the default, which is not the expected behavior.</p>	

Table 1-2. Potential Programming Issues with Variables (Continued)

<p>Unexpected behavior in VIs</p> <p>Using variables instead of a connector pane or using variables to access values in each frame of a sequence structure is a bad practice and can cause unexpected behavior in VIs.</p> <p>Because variables only contain the latest value, you could have potential data loss if you fail to synchronize data operations properly.</p>	
<p>Slow performance</p> <p>Overusing local and global variables, such as using them to avoid long wires across the block diagram or using them instead of dataflow, can slow performance because each instance of a local or global variable makes a copy of the data in memory.</p>	
<p>Race conditions</p> <p>Refer to the <i>Race Conditions</i> section for more information.</p>	

Avoiding Race Conditions

The following table illustrates some ways to avoid race conditions.

<p>Reducing the use of variables.</p> <p>Use subVIs, connector panes, and wires to transfer data whenever possible.</p>	 <p>The diagram shows four orange boxes labeled 'Value' on the left. Wires from three of these boxes connect to a green box labeled 'Stop Button'. A wire from the 'Stop Button' connects to another green box labeled 'Stop Button', which then connects to an orange box labeled 'Balance'.</p>
<p>Properly sequencing instructions by specifying an execution order.</p> <p>Use wires, sequences, or other methods to specify data flow.</p>	 <p>The diagram shows a sequence of operations. On the left, a box labeled 'Threshold Level' is connected to a box labeled 'Gauge'. Below them is a box labeled 'Initialization'. On the right, a box labeled 'Stop Button' is connected to a box labeled 'Gauge'. Wires connect the 'Threshold Level' box to the 'Gauge' box, and the 'Stop Button' box to the 'Gauge' box.</p>
<p>Controlling and limiting shared resources.</p> <p>Minimize shared resources and the number of writers to the remaining shared resources. In general, it is not harmful to have multiple readers for a shared resource. However, try to use only one writer or controller for a shared resource to avoid race conditions.</p>	 <p>The diagram shows a single box labeled 'Variable'. A wire from a document icon on the left connects to the 'Variable' box. Three wires from the 'Variable' box connect to three separate document and glasses icons on the right, representing multiple readers.</p>



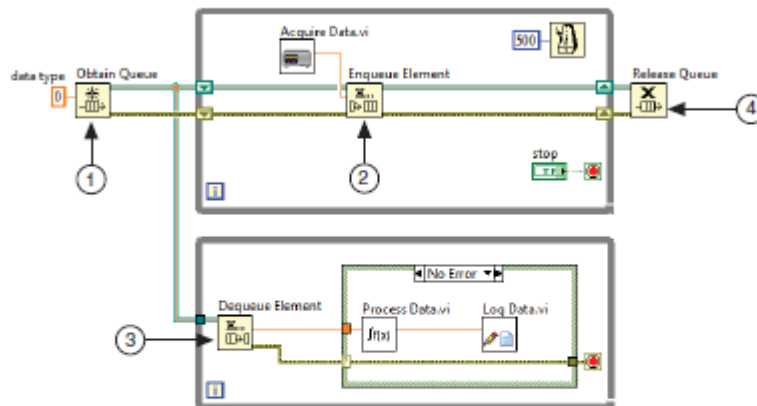
Activity 1-1: Lesson Review - Answers

1. You should use variables in your VI whenever possible.
 - a. True
 - b. False
2. When controlling resources, which combination(s) of writers and readers reduces the chance of race conditions?
 - a. One writer, one reader
 - b. One writer, multiple readers
 - c. Multiple writers, one reader
 - d. Multiple writers, multiple readers

2. Communicating data between parallel loops

Communicating Data Between Parallel Loops - Queues

Use queues to transfer every point of data between parallel loops or VIs.



- 1 Obtain Queue—Creates the queue before the loops begin and defines the data type for the queue.
- 2 Enqueue Element—Adds data to the queue.
- 3 Dequeue Element—Removes data from the queue. The loop does not execute until data is available in the queue.
- 4 Release Queue—Releases the queue. When the queue releases, the Dequeue Element function generates an error, which stops the consumer loop even if more elements are still left in the queue. If you want different functionality for stopping the loops and releasing the queue, you can use different logic and timing than this example.

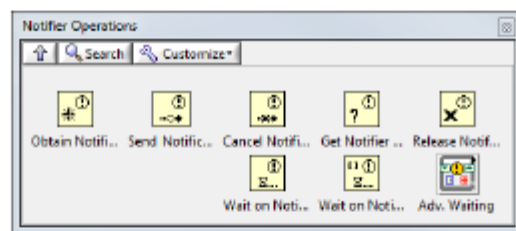
C. Notifiers

Objective: Learn how to create code that broadcasts the latest data to waiting parallel loops using notifiers.

What is a Notifier?



Notifier functions Functions that suspend the execution of a block diagram until they receive data from another section of the block diagram or from another VI running in the same application instance.



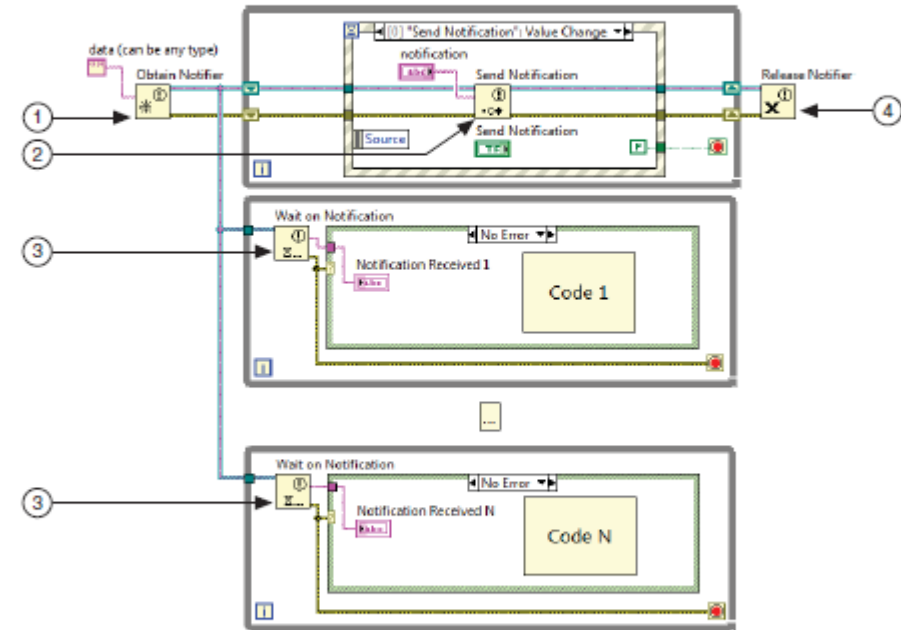
Notifiers vs. Queues

	Notifiers	Queues
Buffer data?	No	Yes
Broadcast data to multiple loops?	Yes	No

Broadcast Data to Parallel Loops

Use Notifiers to communicate between parallel loops when you want one notification to trigger one or more processes and the waiting processes need to receive only the latest notification and data.

The following figure shows how notifiers can be used to broadcast the same data to multiple waiting loops.



- 1 Obtain Notifier—Creates the notifier before the loops begin and defines the data type for the notifier.
- 2 Send Notification—Sends a message to all functions waiting on the notifier.
- 3 Wait on Notification—Waits until the notifier receives a message. The Wait on Notification function will only receive the latest message.
- 4 Release Notifier—Releases the notifier. When the notifier releases, the Wait on Notification functions in the bottom loops generates an error, which stops those loops.

	Suspend execution in reader loop?	Buffer data?	Data can be read by multiple loops?	Use case
Local/global variables			Yes	Transfer latest data
Queues	Yes	Yes		Transfer every point of data
Notifiers	Yes		Yes	Transfer latest data to multiple loops that are waiting on notification

1. Which of the following buffer data?

- a. Queues
- b. Notifiers
- c. Local Variables

2. Match the following:

- | | |
|------------------|---|
| Obtain Queue | b. Assigns the data type of the queue |
| Get Queue Status | d. Determines the number of elements currently in the queue |
| Release Queue | a. Destroys the queue reference |
| Enqueue Element | c. Adds an element to the back of a queue |

3. Which of the following are valid data types for queues and notifiers?

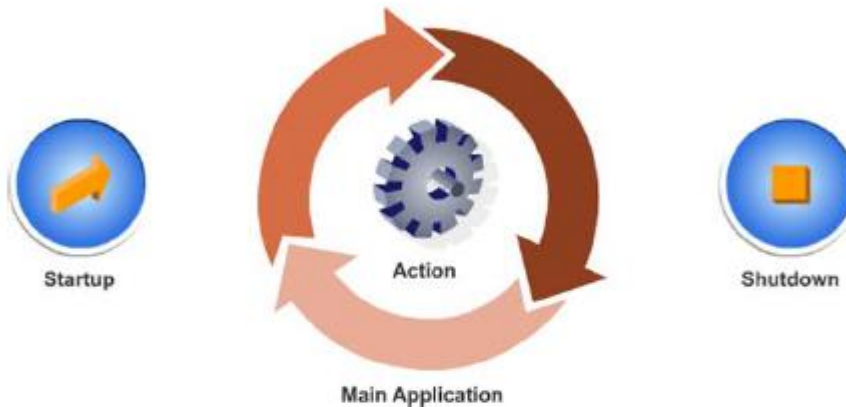
- a. String
- b. Numeric
- c. Enum
- d. Array of Booleans
- e. Cluster of a string and a numeric

3. Implementing Design Patterns.

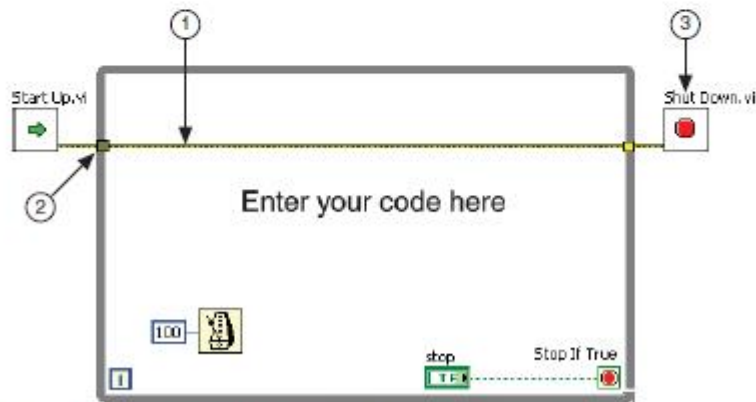
General VI Pattern

The General VI pattern has three main phases:

- **Startup**—Initializes hardware, reads configuration information from files, or prompts the user for data file locations.
- **Main Application**—Consists of at least one loop that repeats until the user decides to exit the program or the program terminates for other reasons such as I/O completion.
- **Shutdown**—Closes files, writes configuration information to disk, or resets I/O to the default state.



General VI Framework

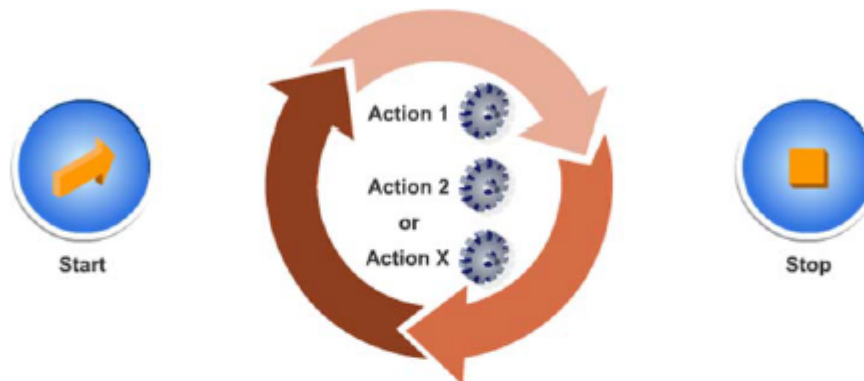


- 1 The error cluster wires control the execution order.
- 2 The While Loop does not execute until the Start Up VI finishes running and returns the error cluster data.
- 3 the Shut Down VI cannot run until the main application in the While Loop finishes and the error cluster data leaves the loop.

State Machine Pattern

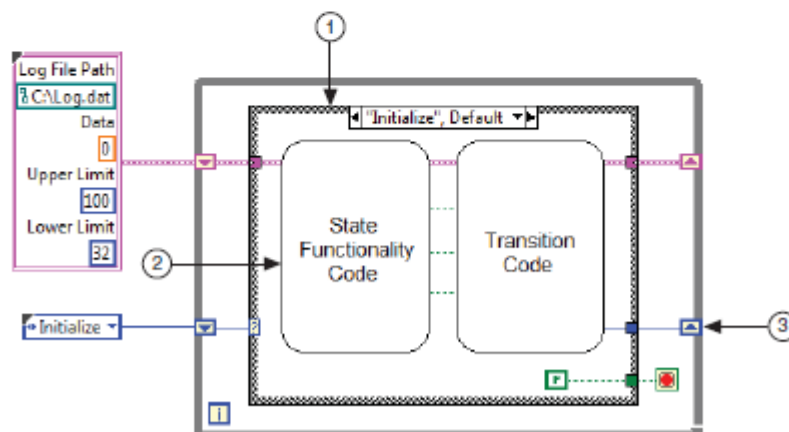
Use this design pattern for VIs that are easily divided into several simpler tasks, such as VIs that act as a user interface.

The state machine pattern has a start up and shut down phase. The main application runs different code each time the loop executes, depending upon some condition.



State Machine Framework

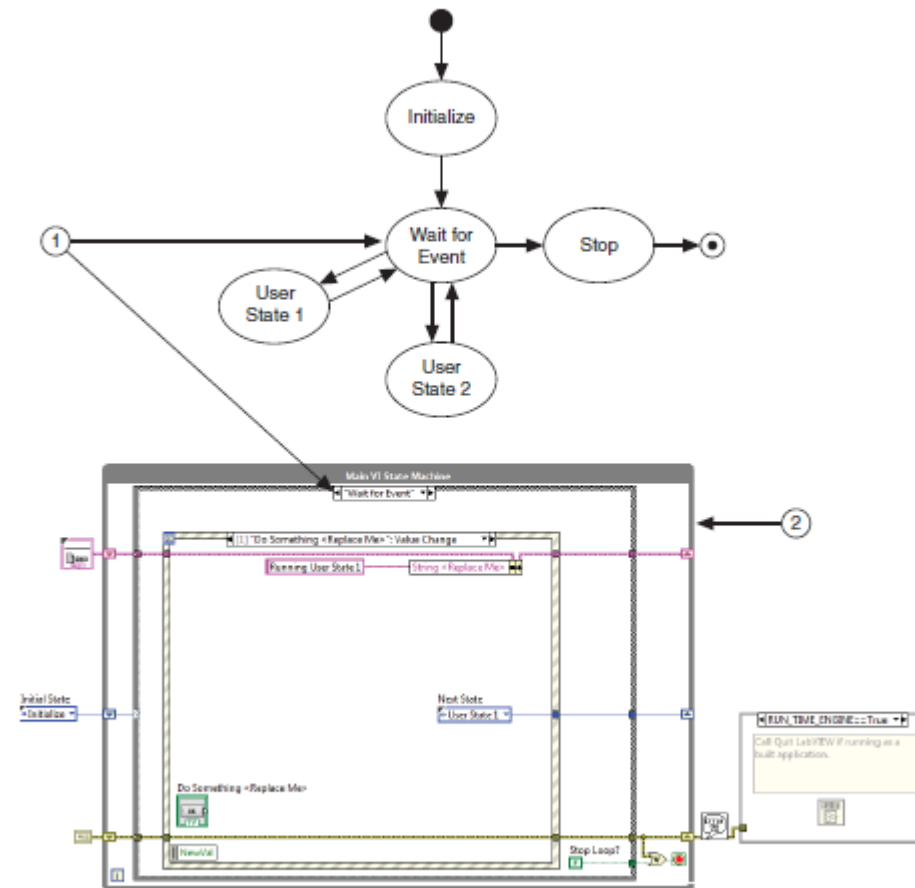
The state machine framework consists of a While Loop, a Case structure, and a shift register.



- 1 Each state of the state machine is a separate case in the Case structure.
- 2 Place VIs and other code that the state should execute within the appropriate case.
- 3 A shift register stores the state that should execute upon the next iteration of the loop.

Event-Based State Machine

The event-based state machine combines the powerful user interaction of the user interface event handler with the transition flexibility of the state machine.



- 1 The Wait for Event state contains an Event structure that waits for front panel changes. The Wait for Event state is the only one that recognizes user input.
- 2 Only one state executes at a time, and the single While Loop means all tasks execute at a single rate.



Activity 3-2: Design Patterns Job Aid

Use the table below to determine the best uses for the design patterns described in this lesson.

Design Pattern	Use	Advantage	Disadvantage
Simple	Standard subVIs Calculations/algorithms; modular processing LabVIEW equivalent of a subroutine in other programming languages	Allows for modular applications	Not suitable for user-interface design or top-level VIs
General	Standard control flow Good for quick prototypes or simple, straight-forward applications that will not grow in complexity	Distinct initialize, run, and stop phases	Unable to return to a previous phase

Design Pattern	Use	Advantage	Disadvantage
State Machine (Polling)	Controls the functionality of a VI by creating a system sequence	Controls sequences Code maintenance is easy because new states can be added quickly For simple applications, do not have to manage both event and state machine diagrams	Polling-based UI is not scalable as application grows This design pattern is not inherently parallel.
State Machine (Events-based)	Controls the functionality of a VI by creating a system sequence	Controls sequences Code maintenance is easy because new states can be added quickly Use of event structure more efficient than polling controls	This design pattern is not inherently parallel.
Producer/Consumer (Data)	Processes or analyzes data in parallel with other data processing or analysis	Buffered communication between application processes	Does not provide loop synchronization Limited to one data type, although data can be clustered
Producer/Consumer (Events)	Responds to user interface with processor-intensive applications	Separates the user interface from processor intensive code	Does not integrate non-user interface events well
Functional Global Variable	Use as a subVI that needs to hold global data and perform actions on that data	Holds data as long as VI is in memory Executes operations based on input selection Good way to protect critical sections of code to eliminate race conditions	Not suitable for reentrant VIs Problematic when duplicating or scaling global data with multiple copies and performing actions on the copies

E. Error Handlers

Objective: Use error handlers in design patterns to manage code execution when an error occurs.

Examples of Error Handlers



Error handler

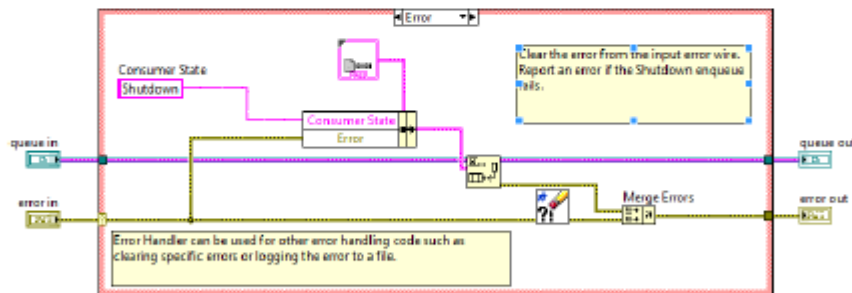
VI or code that changes normal flow or program execution when an error occurs.

Simple Error Handler VI		Displays a dialog box with error information when an error occurs.
General Error Handler VI		Same functionality as the Simple Error Handler VI, except this VI allows you to define custom errors as well.
State machine error handler	Next State 	Transitions the state machine to an error or shutdown state when an error occurs.



Demonstration: Producer Consumer Error Handler VI

Navigate to <Exercises>\Producer Consumer - Error\ and explore how errors are handled in Error Handler.vi.



F. Generating Error Codes and Messages

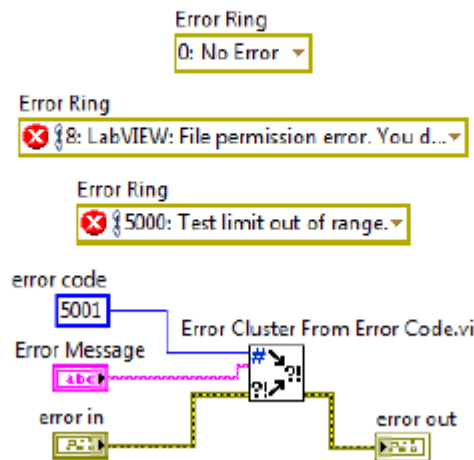
Objective: Use error codes and messages in design patterns.

Error Reporting Options

Use existing error reporting mechanisms to report error conditions detected with your code, such as the following:

- Invalid inputs to subVIs
- File and resource errors
- LabVIEW-generated messages

You can use either pre-defined errors or user-defined errors. You may want to override LabVIEW-generated error messages to make your code more usable. Use error rings to report pre-defined LabVIEW errors or define your own custom errors. You can also use the Error Cluster From Error Code VI to generate a custom error.



G. Timing a Design Pattern

Objective: Use timing functions in design patterns.

Execution and Software Control Timing

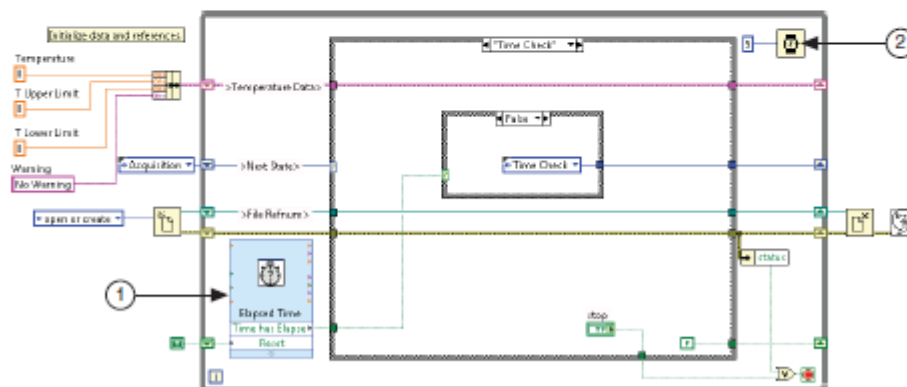


Execution timing

Code that uses timing functions to give the processor time to complete other tasks.

Software control timing

Code that involves timing a real-world operation to perform within a set time period or controls the frequency at which a loop executes.

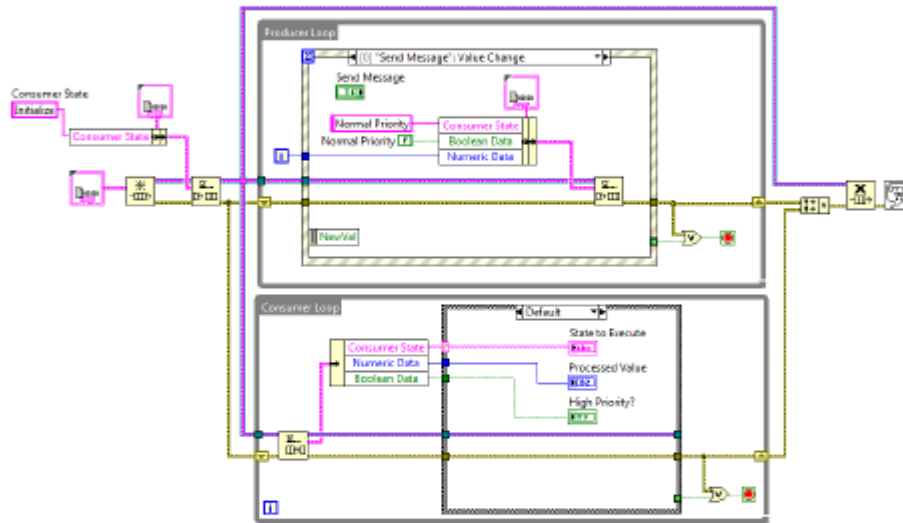


- 1 Software control timing
- 2 Execution timing

Execution Timing

Explicit timing can use a function that specifically allows the processor time to complete other tasks, such as the Wait Until Next ms Multiple function. Execution timing can also be based on events. When the timing is based on event, the design pattern waits for some action to occur before continuing and allows the processor to complete other tasks while it waits.

Use explicit timing for polling-based design patterns such as the producer/consumer design pattern (data) or the polling-based state machine.



Execution Timing Functions

Function Name	Icon
Wait (ms)	
Wait Until Next ms Multiple	

Obtaining Timestamps

If you want to measure the amount of time that passes between two iterations of a loop, use the following functions to calculate relative time durations:

Function	Connector Pane
Tick Count (ms)	millisecond timer value
High Resolution Relative Seconds	relative seconds
Get Date/Time in Seconds	current time



Activity 3-4: Lesson Review - Answers

1. Which of the following are reasons for using a multiple loop design pattern?
 - a. **Execute multiple tasks concurrently**
 - b. Execute different states in a state machine
 - c. **Execute tasks at different rates**
 - d. Execute start up code, main loop, and shutdown code

2. Which of the following are examples of error handling code?
 - a. Displays a dialog box used to correct a broken VI
 - b. Generates a user-defined error code
 - c. **Displays a dialog box when an error occurs**
 - d. **Transitions a state machine to a shutdown state when an error occurs**

4. Controlling the User Interface.

A. VI Server Architecture

Objective: Describe the purpose of the VI Server and the class hierarchy of properties and methods.

VI Server Purpose and Use

The VI Server provides programmatic access to LabVIEW and LabVIEW applications. The VI server performs many functions and in this lesson concentrates on how to use the VI Server to control front panel objects and edit the properties of a VI and LabVIEW.

You can use the VI Server to perform the following actions:

- Programmatically control front panel objects and VIs
- Dynamically load and call VIs
- Run VIs on a computer or remotely across a network
- Programmatically access the LabVIEW environment and editor (Scripting)

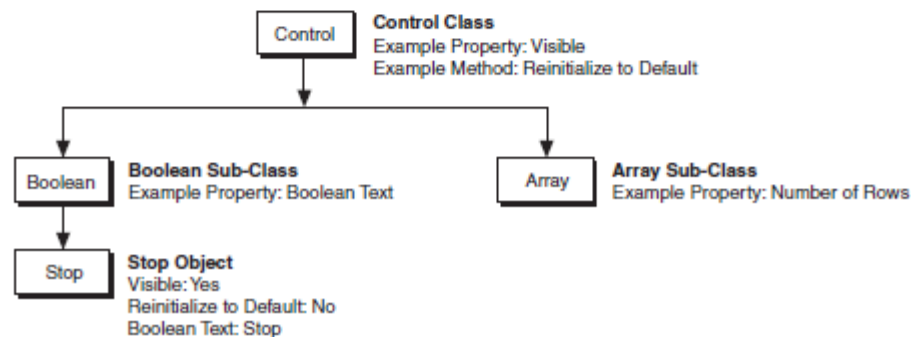
Properties and Methods



Properties Single-valued attributes of the object: read/write, read only, write only
Methods Functions that operate on the object

VI Server—Class Hierarchy

LabVIEW front panel objects inherit properties and methods from a class. When you create a Stop control, it is an object of the Boolean class and has properties and methods associated with that class.



Class Hierarchy

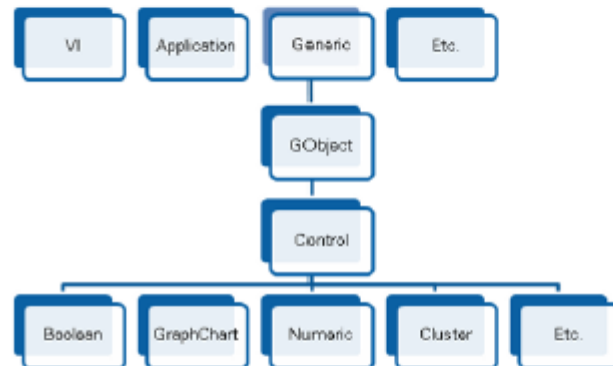


Object

Is a member of a class.

Class

Defines the object type, what an object is able to do, what operations it can perform (methods), and what properties it has.



B. Property Nodes

Objective: Demonstrate how to create property nodes and explain execution order.

Property Nodes

Some of the ways you can use property nodes include the following actions:

- Read and write the properties of an object
- Make modifications programmatically
- Use Context Help to get information about properties

The property nodes in the following table are equivalent approaches. You will learn more about explicitly linked property nodes later in this lesson.

Two Types of Property Nodes	
Implicitly Linked	
Explicitly Linked	

D. Control References

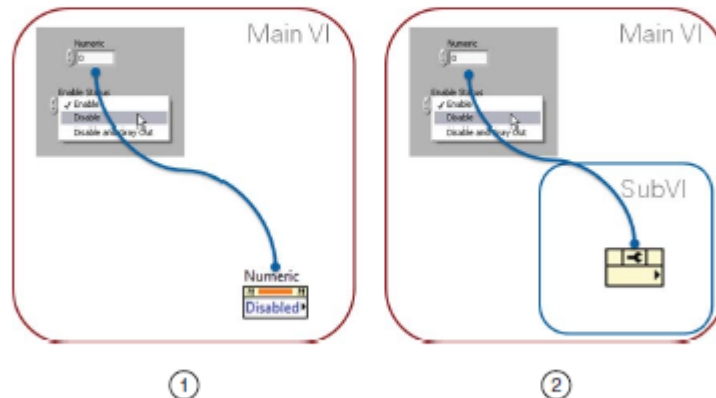
Objective: Practice creating control references and explain the difference between strictly typed and weakly typed control references.



Control reference

A reference to a front panel object.

Implicitly and Explicitly Linked Nodes



- 1 Implicitly Linked Property Node—The Property Node is linked to the front panel object.
- 2 Explicitly Linked Property Node—You must use explicitly linked Property Nodes if the Property Node will be part of a subVI.

Selecting the VI Server Class

You can specify a more generic or a more specific class.



	Advantages	Disadvantages
Specifying a more generic class, such as a Ctl Refnum instead of a Boolean Refnum	Allows the subVI to accept a reference to any type of front panel control.	Limits the properties available.
Specifying a more specific class such as a Num Refnum instead of a Ctl Refnum	Accesses more properties.	Makes the subVI more restrictive. A mismatch of refnum types results in an edit-time error.



Activity 4-1: Lesson Review - Answers

1. For each of the following items, determine whether they operate on a VI class or a Control class.
 - a. Format and Precision: **Control**
 - b. Visible: **Control**
 - c. Reinitialize to Default Value: **Control**
 - d. Show Tool Bar: **VI**

2. You have a Numeric control refnum, shown below, in a subVI. Which control references could you wire to the control refnum terminal of the subVI?



- a. Control reference of a knob
 - b. Control reference of a numeric array
 - c. Control reference of a thermometer indicator
 - d. Control reference of an LED

5. File I/O Techniques.

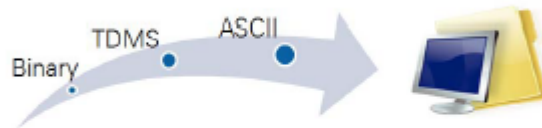
A. File Formats

Objective: Recognize appropriate use cases and the pros and cons of each file type.

Files store data as a series of bits.

At their lowest level, all files written to your computer's hard drive are a series of binary bits. However, many formats for organizing and representing data in a file are available. There are three common file types in LabVIEW:

- ASCII file format
- TDMS file format
- Direct binary data storage



Compare File Formats

	ASCII	TDMS	Direct Binary
Numeric Precision	Good	Best	Best
Share Data	Best (Any program easily)	Better (NI programs easily; Excel)	Good (only with detailed format information)
Efficiency	Good	Best	Best
Ideal Use	Share data with other programs when file space and numeric precision are not important.	Store measurement data and related metadata. High-speed streaming without loss of precision.	Store numeric data compactly with ability to random access.

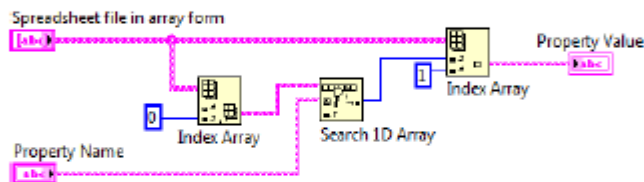


Activity 5-2: Lesson Review - Answers

1. Consider the code shown below. The resulting Log File Path contains a text file path in which folder?



- a. Same folder as the VI that executed the code
 - b. Same folder as the LabVIEW Project
 - c. Current user's AppData Directory
 - d. **Unknown**
2. In the following example, what index value is returned from the Search 1D Array function if Property Name is not found in the input array?



- a. NaN (Not a Number)
 - b. 0
 - c. -1
 - d. Negative Infinity
3. You need to store data which other engineers will later analyze with Microsoft Excel. Which file storage format(s) should you use?
- a. **Tab-delimited ASCII**
 - b. Custom binary format
 - c. TDMS
4. TDMS files store properties at which of the following levels?
- a. File
 - b. **Channel Group**
 - c. Channel
 - d. Value
6. Improving an Existing VI.

A. Refactoring Inherited Code

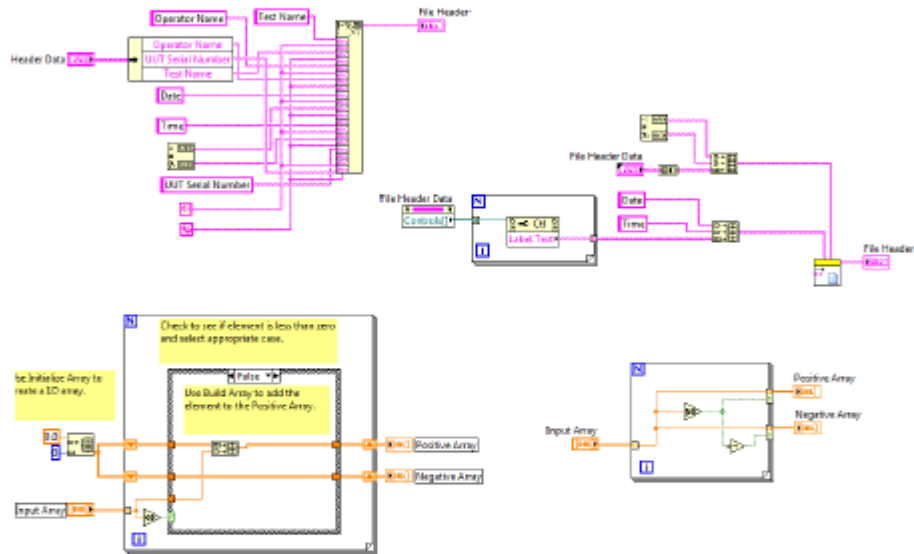
Objective: Explain the refactoring process and identify VIs that would benefit from refactoring.

Definition



Refactoring The process of redesigning software to make it more readable and maintainable so that the cost of change does not increase over time.

Refactoring changes the internal structure of a VI to make it more readable and maintainable, without changing its observable behavior.



When to Refactor

Consider refactoring when you are adding a feature to a VI or debugging it. Refactoring is usually beneficial. However, there is value in a VI that functions, even if the block diagram is not readable.

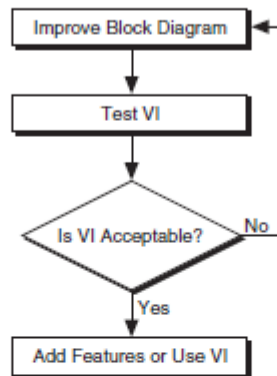


Good candidates for complete rewrites include the following types of VIs:

- VIs that do not function
- VIs that satisfy only a small portion of your needs

Refactoring Process

When you refactor to improve the block diagram, make small cosmetic changes before tackling larger issues. Cosmetic changes to the block diagram can be useful. For example, it is easier to find duplicated code if the block diagram is well-organized and the terminals are well-labeled.



B. Typical Refactoring Issues

Objective: Recognize common code issues that may require you to refactor your code.

Poor Naming

Inherited VIs often contain controls and indicators that do not have meaningful names.

My Acq.vi	Acq Window Temperature.vi	Acq Window Temperature.vi
Poor	Better	Best

By renaming controls and VIs and creating meaningful VI icons, you can improve the readability of an inherited VI.

7. Creating and distributing applications.



Activity 7-1: Lesson Review - Answers

1. You need to use LabVIEW build specifications to distribute which of the following items?
 - a. Installers
 - b. ZIP files
 - c. VIs
 - d. Executables
2. Mark the following statements True or False.

Applications that you create with Build Specifications generally have the same system requirements as the LabVIEW development system used to create the VI or application.	True / False
You create executables to make sure the application behaves the same when distributed to different systems.	True / False
Memory requirements vary depending on the size of the application created.	True / False
You should always include the LabVIEW Run-Time Engine in the application build specification.	True / False