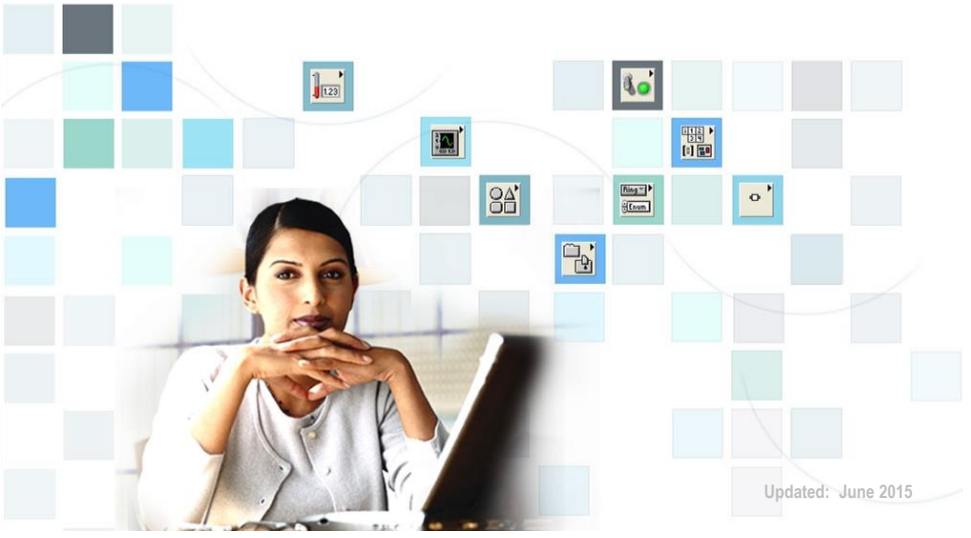
NATIONAL INSTRUMENTS

LabVIEW Core 2



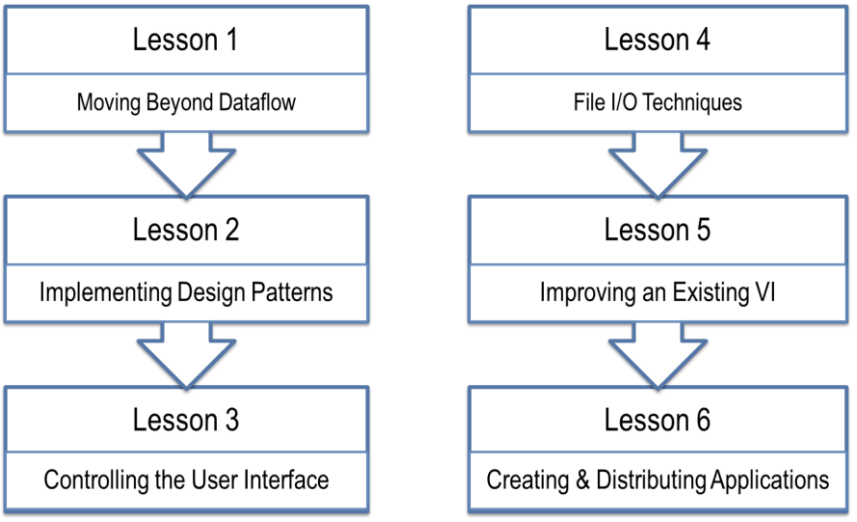
Updated: June 2015

1

Getting the Most out of this Course

- Ask questions!
- Experiment with hands-on exercises to understand the methods used.
- Explore solutions. Remember that implementations explore one possible solution. You may find a better one!

Course Learning Map



Lesson 1
Moving Beyond Dataflow

TOPICS

A. Asynchronous Communication

B. Queues

C. Event-Driven Programming

A. Asynchronous Communication

Asynchronous Communication

LabVIEW is a dataflow language.

- Functions depend on other functions for data.
- Dependent functions do not execute until their dependencies have completed execution.
- Wires transfer data between functions.

However, sometimes you need break dataflow and program using asynchronous communication.

Asynchronous Communication

- Asynchronous communication is between the following:
  - Parallel loops
  - UI and Block Diagram
  - VIs
  - Application instances (LabVIEW projects, executables, etc.)
- Information includes the following:
  - Data
  - Notification that something happened

B. Queues

Queues

Queue Operations

Producer/Consumer (Data) Design Pattern

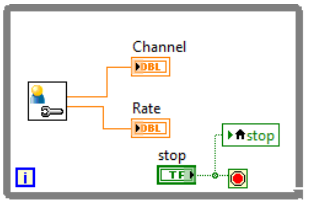
Queues

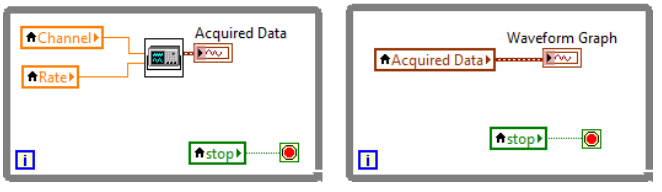
- Are used for communicating data between parallel loops.
- Store multiple pieces of data (i.e. buffer data).
- Work in a FIFO (first in, first out) manner by default.
- Can hold data of any type.

Drawbacks of Variables

Drawbacks of using variables to communicate between loops:

- It's possible to read duplicate data.
- It's possible to miss data.
- You can create read-modify-write race conditions.

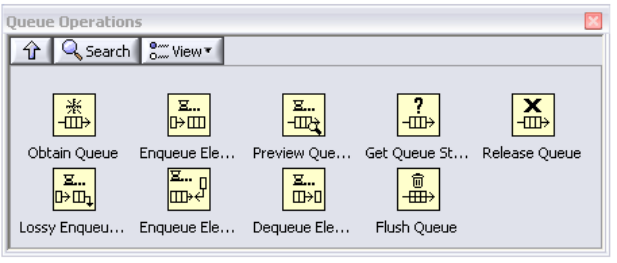




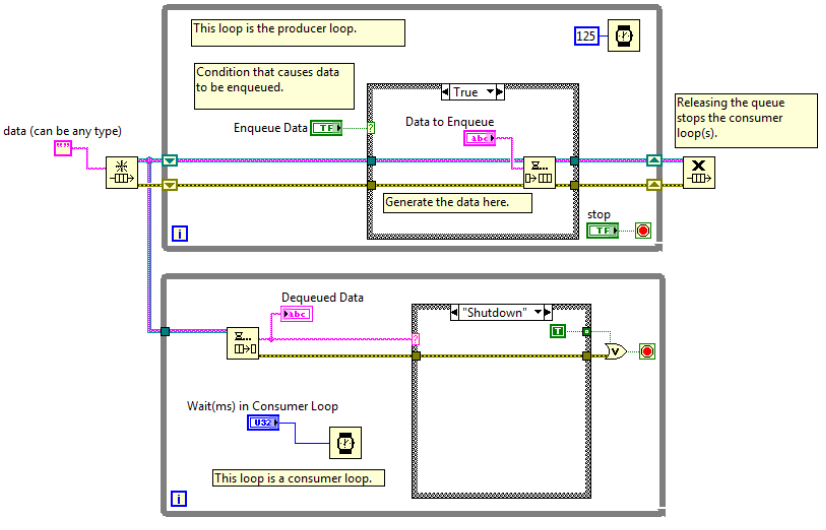
Queue Operations

Use the Queue Operations functions to create and use a queue for communicating data between:

- Different sections of a VI.
- Different VIs.



Producer/Consumer Design Pattern (Data)



C. Event-Driven Programming

Events – Definition

Event-Driven Programming – Definition

Polling Versus Event Structures

Parts of an Event Structure

Configuring the Event Structure

Caveats and Recommendations

Events

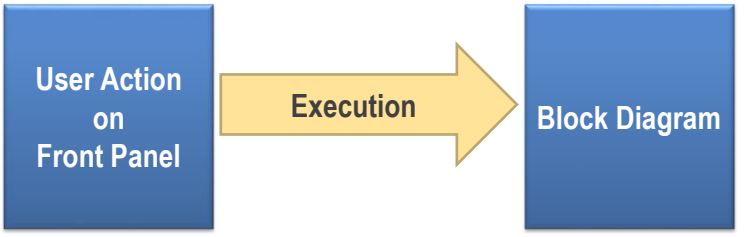
Event — An asynchronous notification that something has occurred

- Events originate from the user interface, external I/O, or other parts of the program.
- Events do things TO event sources.

–Example: Value change happens TO front panel controls.

Event-Driven Programming

Event-Driven Programming — Method of programming where the program waits for an event to occur before executing one or more functions



Polling versus Event Structures

Polling

- Method of event-based programming where a loop must continually run code to check if changes have occurred.
- Polling the front panel requires a significant amount of CPU time.
- Polling can fail to detect changes if they occur too quickly.

Event Structures

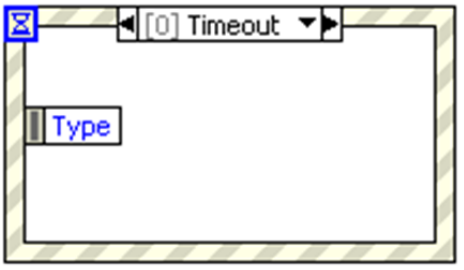
- Events in Event structures eliminate the need to poll the front panel.
- Benefits of using Event structures:
  - –Reduces the CPU requirements of the program.
  - –Simplifies the block diagram code.
  - –Guarantees that the block diagram can respond to all interactions the user makes.

Using Event Structures for Event-Driven Programming

An Event structure works like a Case structure with a built-in Wait on Notification function.

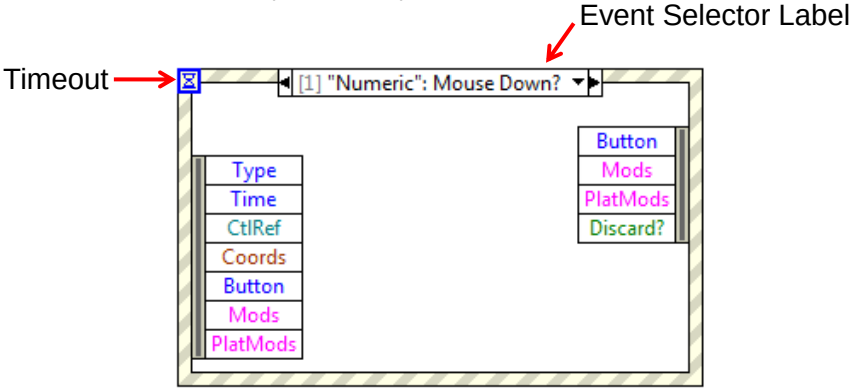
Use an Event structure to handle user-interface (static) events such as:

- Pressing a button on the mouse.
- Pressing a key on the keyboard.
- Changing the value of a numeric control.



Parts of an Event Structure

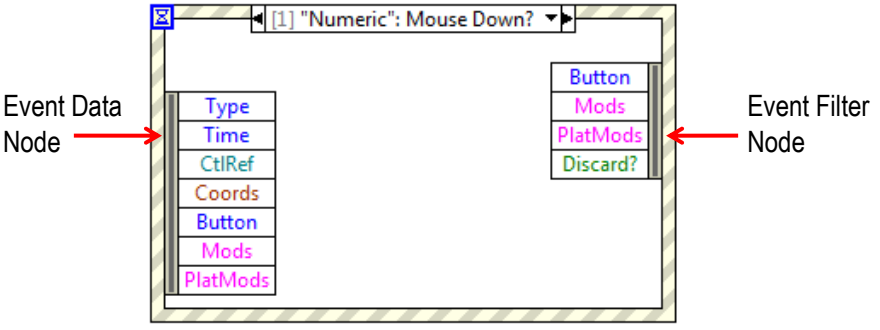
- Event Selector Label—Identifies the event case viewed.
- Timeout—Specifies time in ms to wait for events. Default value is −1 (indefinite).



Parts of an Event Structure (continued)

•Event Data Node—Identifies the data LabVIEW provides when the event occurs; similar to the Unbundle By Name function.

•Event Filter Node—Identifies the subset of data available in the Event Data node that the event case can modify.

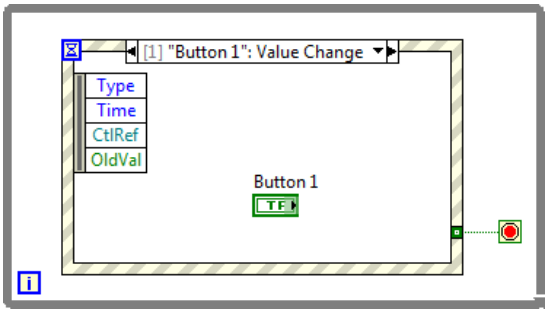


Using an Event Structure

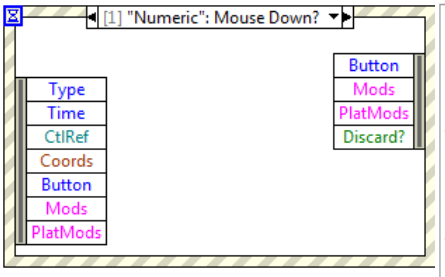
In general, place Event structures inside While Loops.

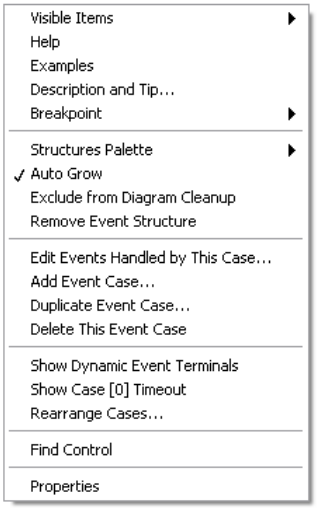
•Event structures handle exactly one event per iteration of the While Loop.

•Event structures sleep when no events occur.



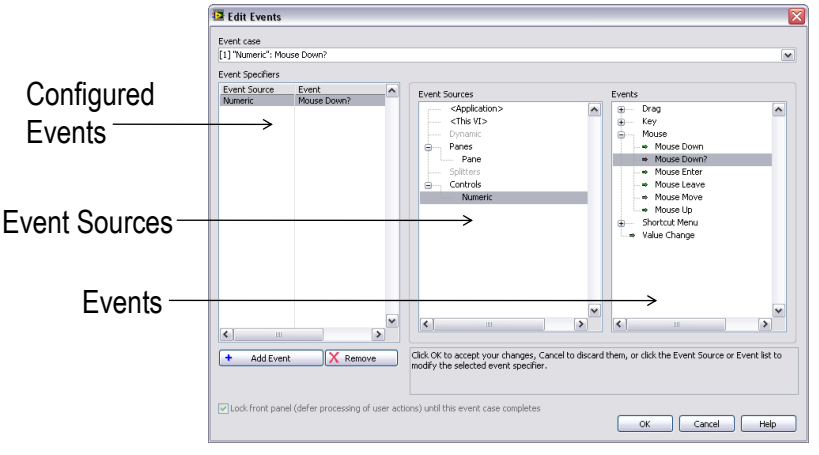
Configuring the Event Structure



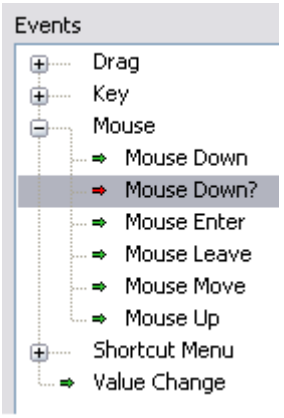


Use a dialog box to configure each event by right-clicking the Event structure border and selecting **Edit Events Handled by This Case** from the shortcut menu.

Edit Events Dialog Box



Notify and Filter Events



Notify Events (green arrow)

User action has already occurred and LabVIEW has processed the event.

Filter Events (red arrow)

User action has already occurred and LabVIEW has NOT processed the event.

Filter events allow you to override default behavior for event.

Caveats and Recommendations

- Avoid using an Event structure outside of a loop.
- Place only one Event structure in a loop.
- Avoid configuring two Event structures for the same event.
- Use a Value Change event to detect value changes.
- Keep event handling code short and quick.
- Place Boolean control terminals inside an event case for latched operations to work properly.

Summary—Quiz

1.Which of the following buffer data?

a)Queues

b)Events

c)Local Variables

Summary—Quiz Answer

1.Which of the following buffer data?

a)Queues

b)Events

c)Local Variables

Summary—Match the Following

1.Obtain Queue 

2.Get Queue Status 

3.Release Queue 

4.Enqueue Element 

a.Destroys the queue reference

b.Assigns the data type of the queue

c.Adds an element to the back of a queue

d.Determines the number of elements currently in the queue

Summary—Match the Following Answer

1.Obtain Queue 

2.Get Queue Status 

3.Release Queue 

4.Enqueue Element 

a.Destroys the queue reference

b.Assigns the data type of the queue

c.Adds an element to the back of a queue

d.Determines the number of elements currently in the queue

Summary—Quiz

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

a)String

b)Numeric

c)Enum

d)Array of Booleans

e)Cluster of a String and a Numeric

Summary—Quiz Answer

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

a)String

b)Numeric

c)Enum

d)Array of Booleans

e)Cluster of a String and a Numeric

Summary—Quiz

4. The Event structure handles only one event each time it executes.

a)True

b)False

Summary—Quiz Answer

4. The Event structure handles only one event each time it executes.

a)True

b)False

Lesson 2
Implementing Design Patterns

TOPICS

A. Design Patterns
B. Simple Design Patterns
C. Multiple Loop Design Patterns
D. Error Handlers
E. Generating Error Codes and Messages
F. Timing a Design Pattern
G. Functional Global Variable Design Pattern

A. Design Patterns

Why Use Design Patterns?

- They have proven themselves useful for developing software.
- You don't have to start a program from scratch.
- They make it easier for others to read and modify your code.

Design Patterns – Code implementations and techniques that are solutions to specific problems in software design

Design patterns typically evolve through the efforts of many developers and are fine-tuned for simplicity, maintainability, and readability.

B. Simple Design Patterns

Simple VI

General VI

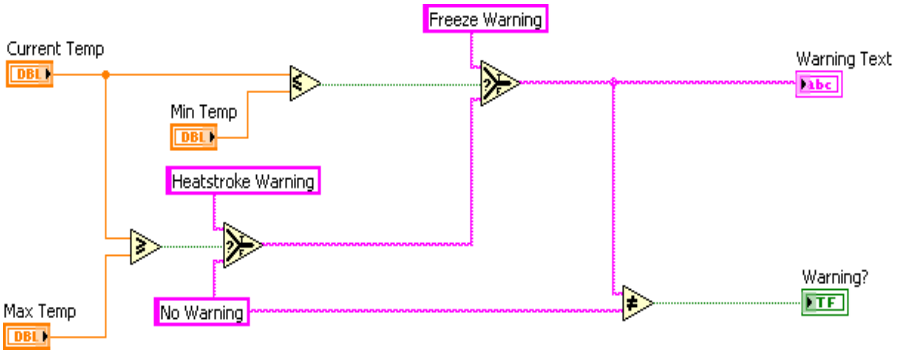
State Machine

Event-Based State Machine

Simple State Machine

Simple VI Pattern

- Single VI that takes a measurement, performs calculations, and either displays the results or records them to disk.
- Usually does not require a specific start or stop action from the user.

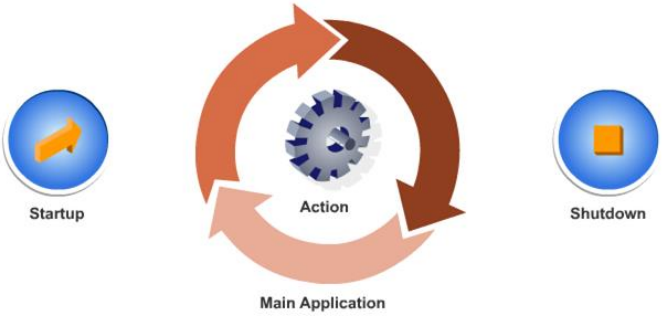


50

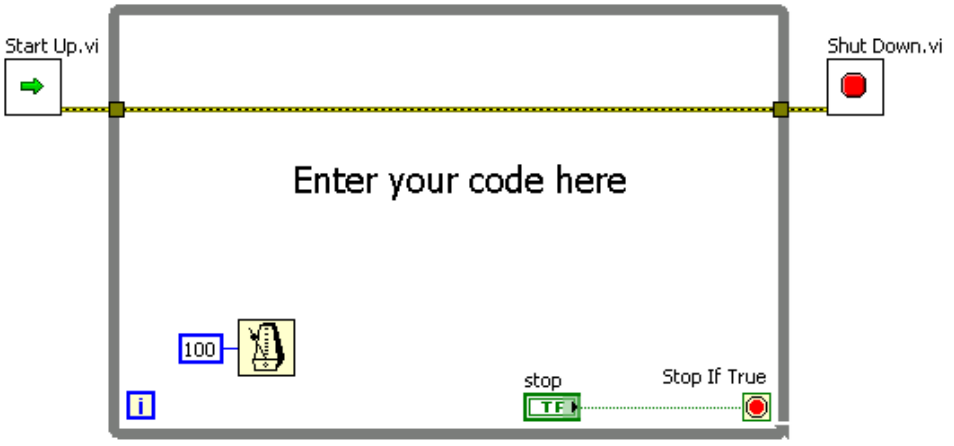
General VI Pattern

This pattern has three phases:

- Start-up
- Main application
- Shut-dow

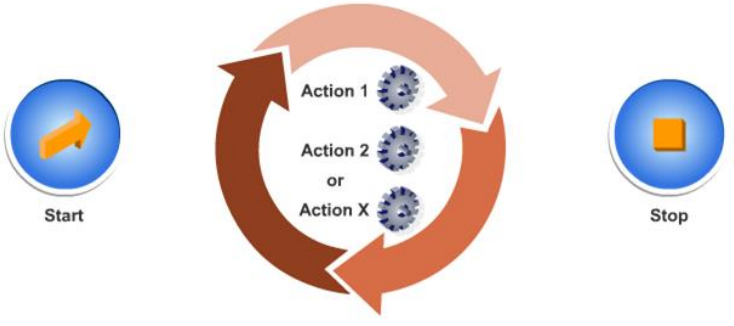


General VI Framework

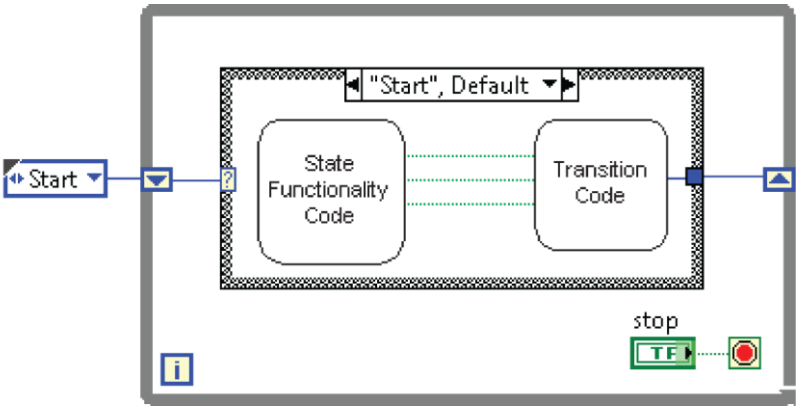


State Machine Pattern

Usually has a start-up and shut-down state, but also contains other states.



State Machine Framework

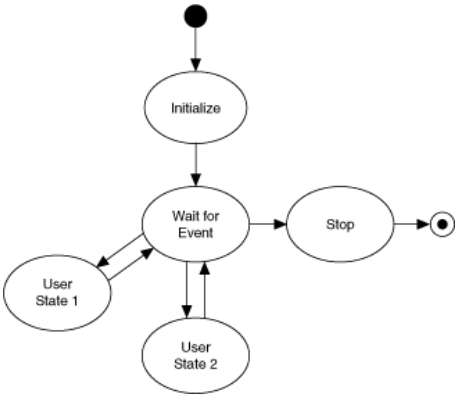


Event-Based State Machine

- Combines event programming with a State Machine design.
- Includes a "Wait on Event" case to processes user-interface events.

Simple State Machine Template

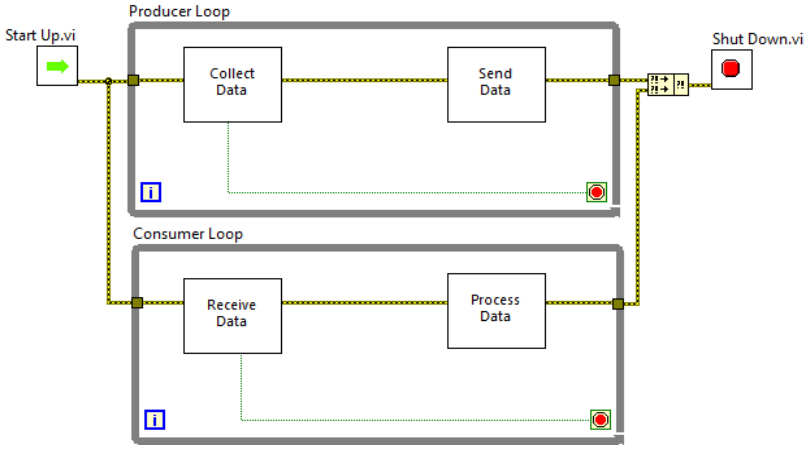
Use the **Create Project** dialog box to expedite implementing an event-based state machine application.



C. Multiple Loop Design Patterns

Producer/Consumer (Events)

Producer/Consumer Design Patterns



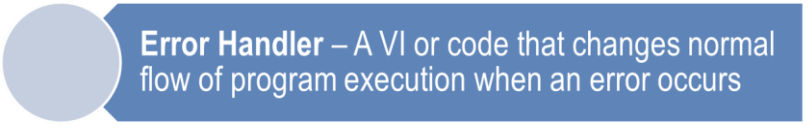
D. Error Handlers

Examples

Producer/Consumer Error Handler

Examples of Error Handlers

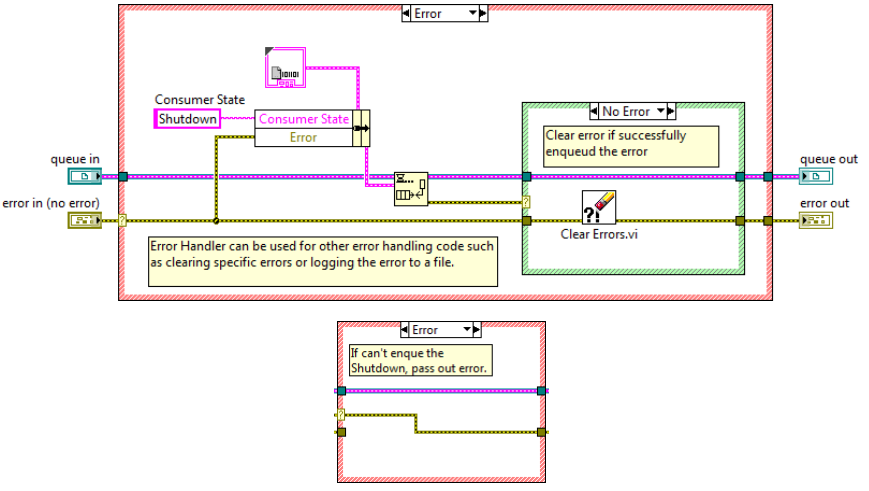
- Simple Error Handler VI
- State machine error handler
- I/O error handler



Producer/Consumer Error Handler

- Both producer and consumer loops gracefully stop when an error occurs.
  - Producer loops pass error information to consumer loops.
  - Consumer loops send stop information to producer loops.
- Transition to a Shutdown case in the consumer loop to execute shutdown code prior to stopping the VI.
- Report error information to the user.

Error Handler VI



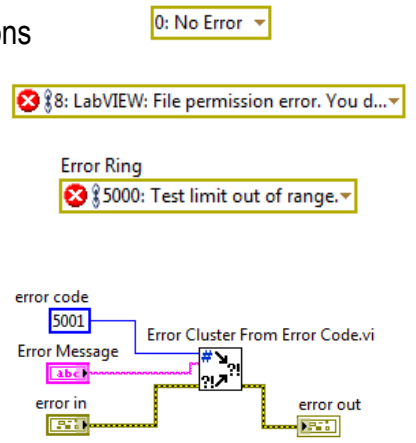
E. Generating Error Codes and Messages

Error Reporting Options

Error Ring

Error Reporting Options

- Use existing error reporting mechanisms to report error conditions detected with your code. Errors to include:
  - Invalid inputs to subVIs
  - File and resource errors
  - LabVIEW-generated messages
- Options for error reporting:
  - Pre-defined errors
  - User-defined errors
  - Overriding LabVIEW-generated messages



F. Timing a Design Pattern

Execution Timing

Software Control Timing

- Synchronized Timeout
- Get Data/Time in Seconds

Timing a Design Pattern

Execution Timing

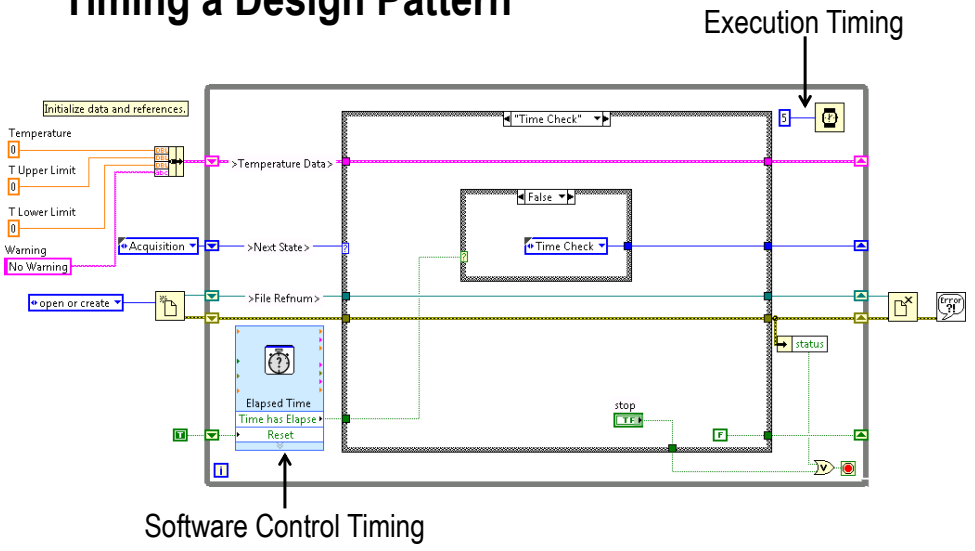
•Provides the design pattern with a function that specifically allows the processor time to complete other tasks.

Software Control Timing

•Times a real-world operation to perform within a set time period.

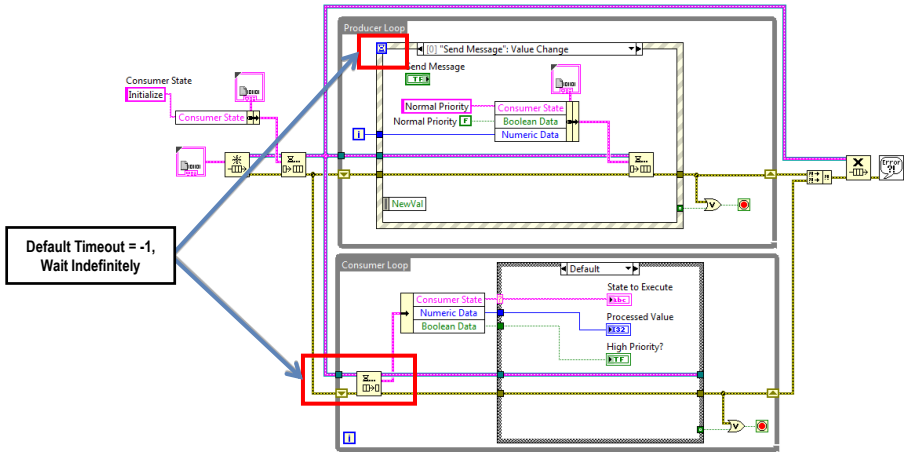
•Controls the frequency at which a loop executes.

Timing a Design Pattern



Execution Timing

Design patterns where the timing is based on the occurrence of events do not need execution timing.



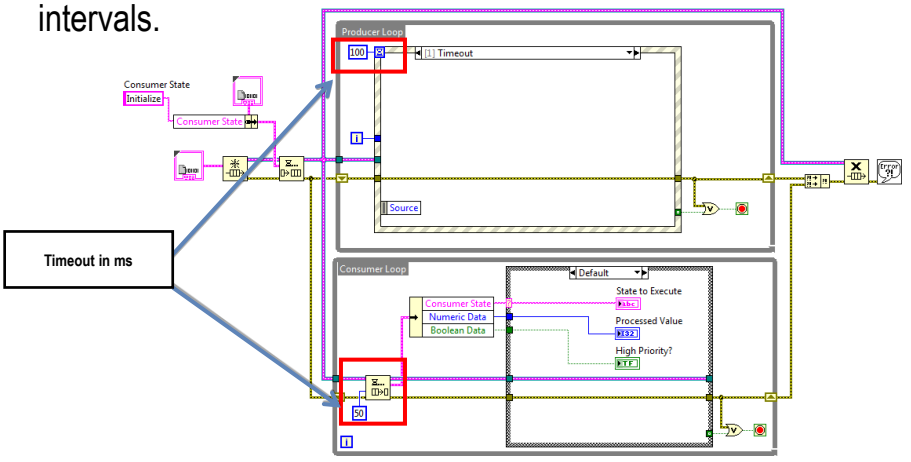
75

Software Control Timing

- Software control timing must allow the design pattern to run continuously without stopping.
- The following functions are better for execution timing rather than software control timing.
  - Wait (ms)
  - Wait Until Next ms Multiple

Synchronized Timeout

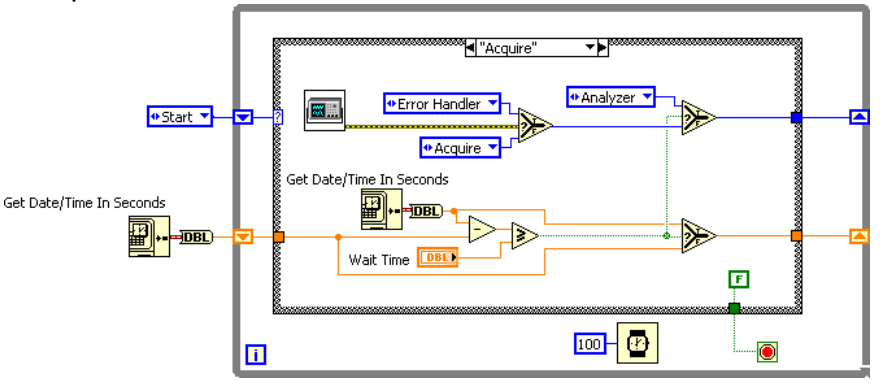
Even if no other events occur or the queue is empty, both the producer and consumer loops continue to execute at regular intervals.



77

Get Date/Time in Seconds

Use the Get Date/Time In Seconds function for working with elapsed times.



G. Functional Global Variable Design Pattern

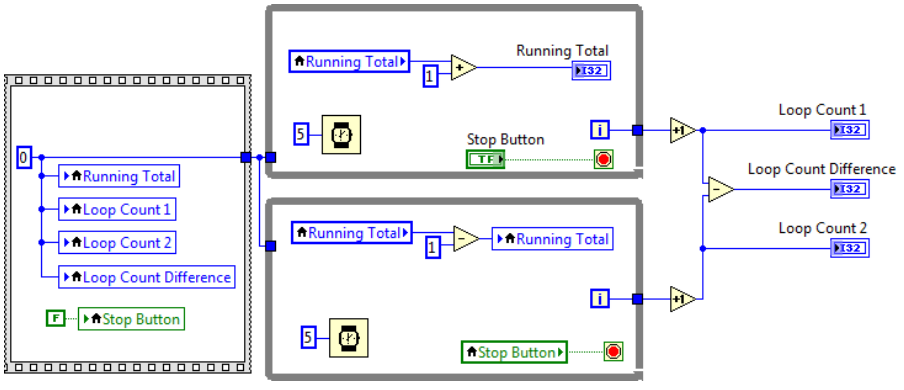
Motivation
- Problem: Read-Modify-Race Conditions
- Solution: Use FGV SubVI

Function Global Variable (FGV)
- Definition
- Basic Framework

Other Use Cases for FGVs

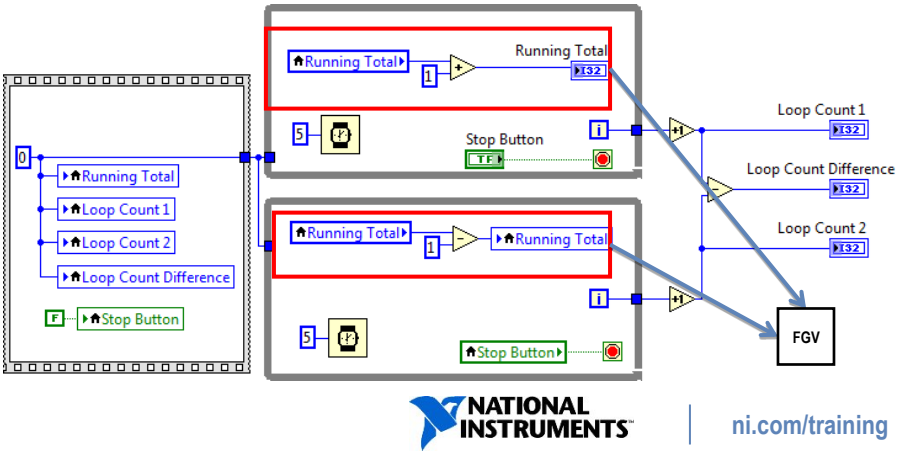
Problem: Read-Modify-Write Race Conditions

- The block diagram below has a "read-modify-write" race condition.
- Simultaneous transactions can generate incorrect data.



Solution: Use a SubVI Based on the Functional Global Variable (FGV) Design Pattern

Wrap the shared global resource and all functions that work on the resource into a FGV subVI.

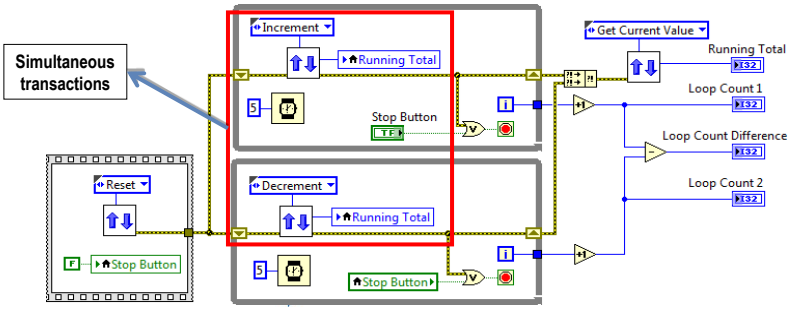


83

The Up Down Counter VI Example

Up Down Counter VI is an FGV VI that:

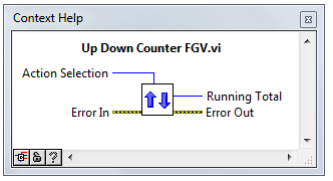
- Eliminates the read-modify-write race condition for simultaneous transactions.
- Encapsulates methods for resetting, incrementing, and decrementing.

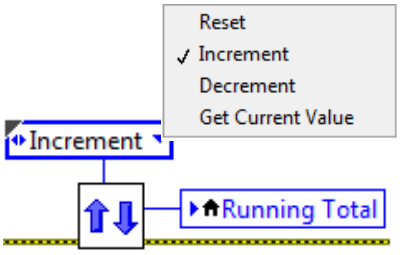


84

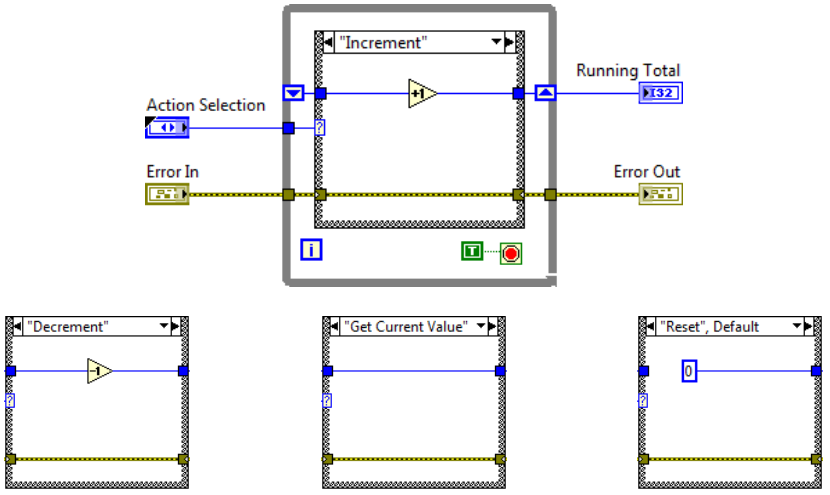
Up Down Counter VI

- **Action Selection—**Uses an Enum to select the counter operation:
  - –Reset
  - –Increment
  - –Decrement
  - –Get Current Value
- **Running Total—**Outputs the current value of the Up Down Counter VI.





Up Down Counter VI Framework



Functional Global Variable

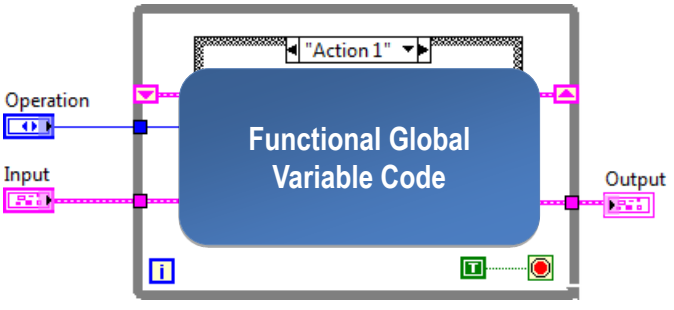
Functional Global Variable – A non-reentrant VI that uses uninitialized shift registers to hold global data. The VI often allows for actions to be performed on the data.

- **Functional**
  - Can perform calculations or manipulate data.
- **Global**
  - Make data globally available within an Application Instance.
- **Variable**
  - Stores data.
  - Can be written to or read from like a variable.

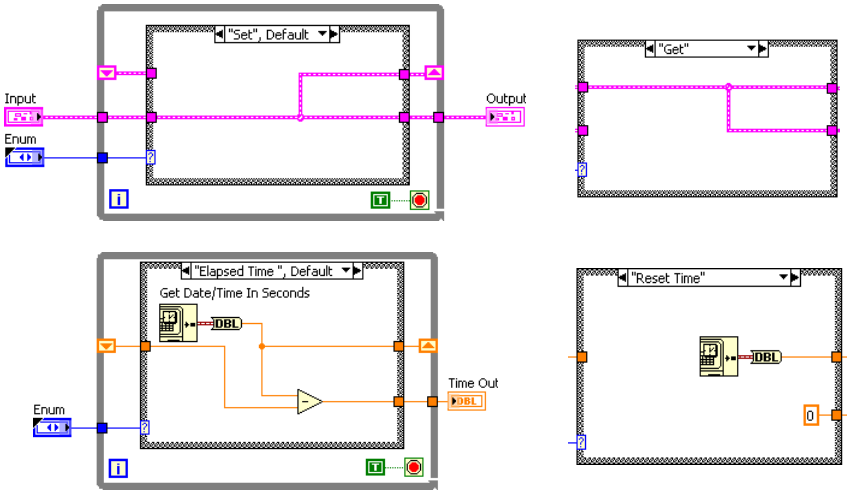
The FGV Basic Framework

The general form of a functional global variable VI includes:

- An uninitialized shift register with a single iteration While Loop.
- A Case structure.



Other Use Cases for FGVs



Summary—Quiz

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

Summary—Quiz Answer

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

Summary—Quiz

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

Summary—Quiz Answer

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

Summary—Quiz

3.What is the default timeout value of an Event structure?

a)0

b)100 ms

c)Never time out

d)The input value of the Wait (ms) function that exists in the same loop as the Event structure

Summary—Quiz Answer

3.What is the default timeout value of an Event structure?

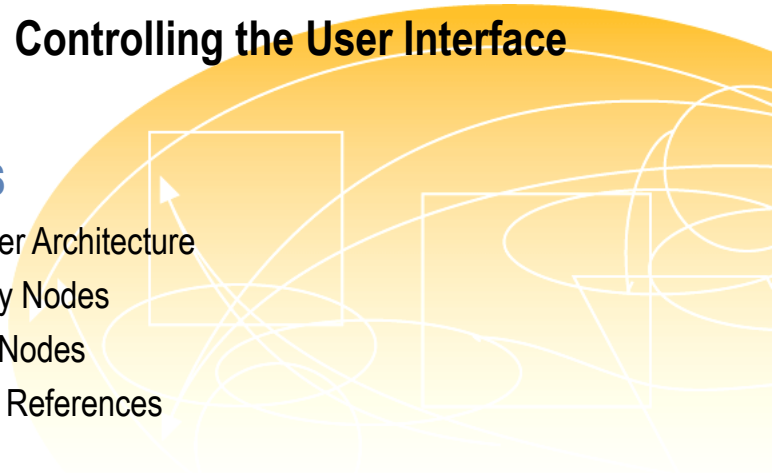
a)0

b)100 ms

c)Never time out

d)The input value of the Wait (ms) function that exists in same loop as the Event structure

Lesson 3
Controlling the User Interface



TOPICS

A.VI Server Architecture

B.Property Nodes

C.Invoke Nodes

D.Control References

A. VI Server Architecture

VI Server Architecture

Properties and Methods

VI Class Hierarchy

VI Server Architecture

The VI Server provides programmatic access to LabVIEW.

Use the VI Server to:

−Programmatically control front panel objects & VIs

−Dynamically load and call VIs

−Run VIs on a computer or remotely across a network

−Programmatically access to the LabVIEW environment and editor (Scripting)

Properties and Methods

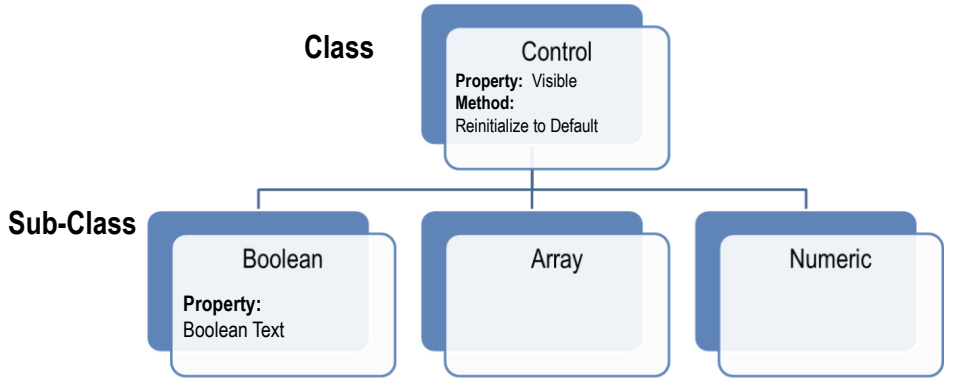
Properties — Single-valued attributes of the object: read/write, read only, write only

Properties include color, position, size, visibility, label text, and label font.

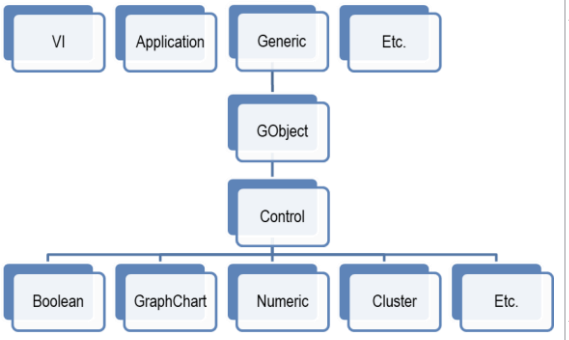
Methods — Functions that operate on the object

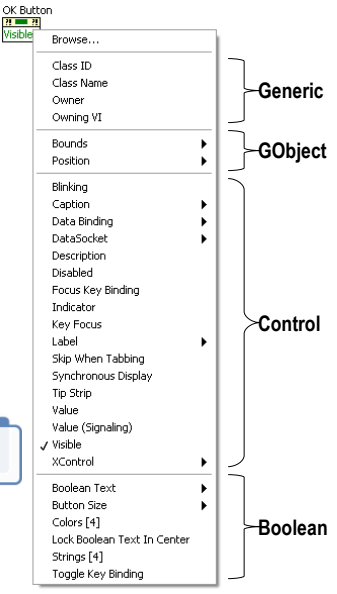
Methods include reinitializing values to default and exporting graph images.

VI Server—Class Hierarchy



VI Server—Class Hierarchy





B. Property Nodes

Definition

Creating Property Nodes

Execution Order

Property Nodes

•Property Nodes read and write the properties of an object. Property Nodes can:

−Change the color of a chart plot.

−Disable and enable controls.

−Get the location of a control or indicator.

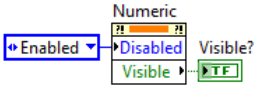
•Property Nodes allow you to make these modifications programmatically.

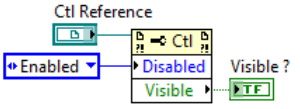
•Use Context Help to get information about properties.

•There are two types of Property Nodes.

−Implicitly linked

−Explicitly linked





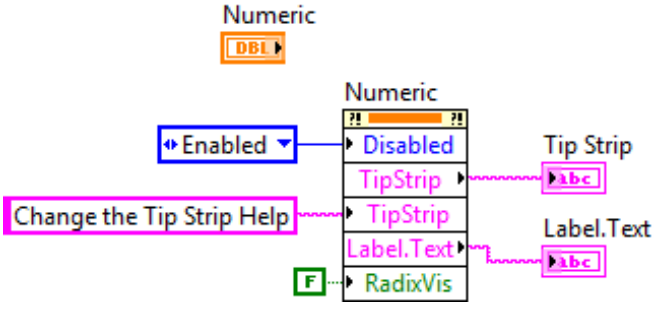
Creating Property Nodes

Create a Property Node for a front panel object.

DEMONSTRATION

Execution Order

- Property Nodes can have multiple properties.
- Properties execute from top to bottom.



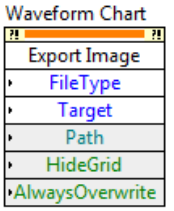
C. Invoke Nodes

Definition

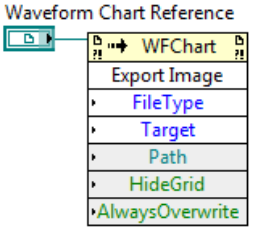
Control Methods

VI Methods

Invoke Nodes



- Invoke Nodes call methods or actions on objects.
  - Get VI Version
  - Print VI panel
  - Reinitialize All to Default
- Invoke Nodes perform actions on referenced items such as VIs and controls.
- Most methods have parameters.
- Use Context Help to get information on methods.
- There are two types of control Invoke Nodes.
  - Implicitly linked
  - Explicitly linked

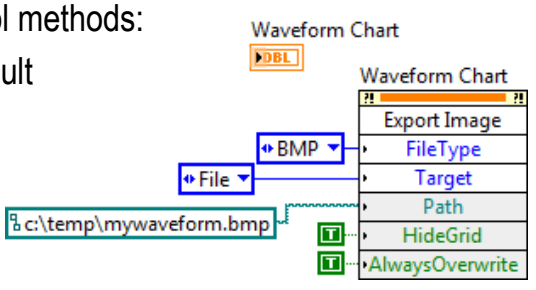


Control Methods

- To create an implicitly linked Invoke Node:

1. Right-click the control terminal on the block diagram and select **Create»Invoke Node**.
2. Select a method from the submenu.

- Examples of control methods:
  - Reinitialize to Default
  - Export Image



VI Methods

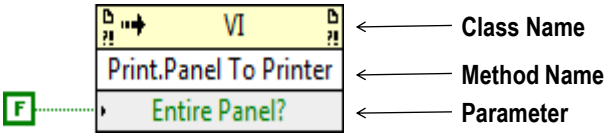
•Use a VI Server Reference to associate an Invoke Node with the current VI.

•To create a VI method:

1.Place an Invoke Node on the block diagram.

2.Right-click and select **Select Class** to choose a class.

3.Right-click again and select **Select Method** to choose a method.



D. Control References

Implicitly and Explicitly Linked Property Nodes

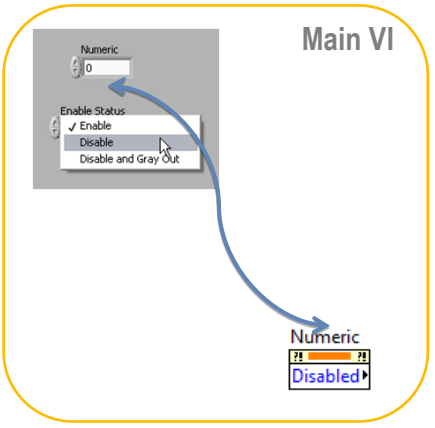
Create SubVIs

- Use the Create SubVI Tool
- Create Manually

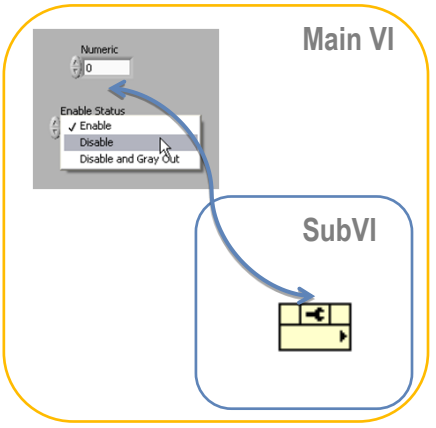
Select a VI Server Class

Control References

Implicitly Linked Property Node

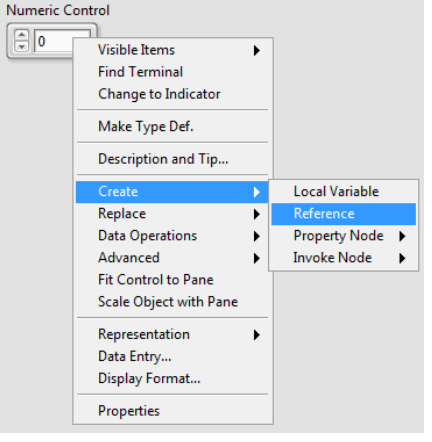


Explicitly Linked Property Node



Control References

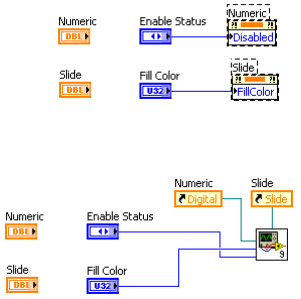
- A control reference is a reference to a front panel object.
- Wire control references to generic Property Nodes.
- Pass control references to subVIs.



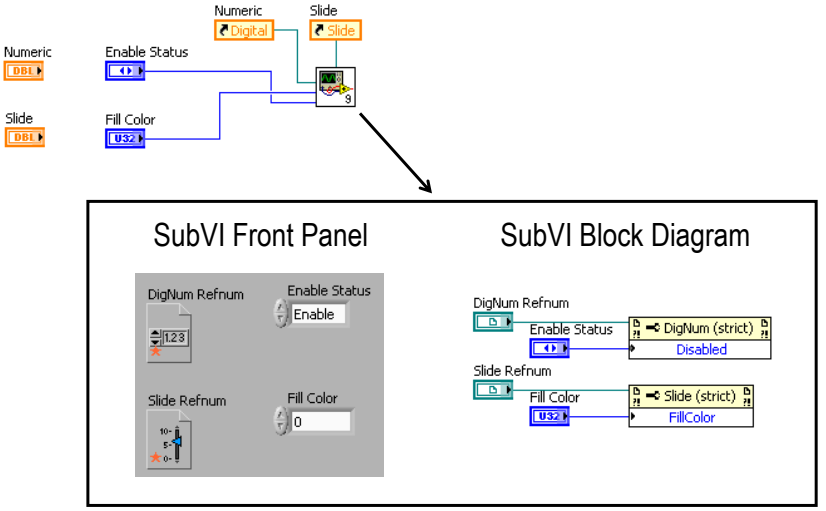
Creating a SubVI

To create explicitly linked Property Nodes in a subVI:

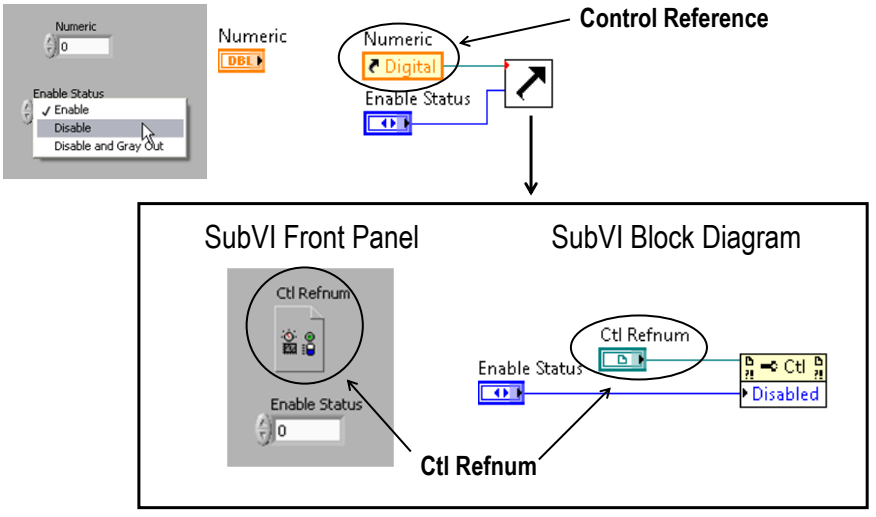
1. Create your VI.
2. Select the portion of the block diagram that will be in the subVI.
3. Select **Edit»Create SubVI**. LabVIEW automatically creates the control references needed for the subVI.
4. Customize and save the subVI.



Creating a SubVI



Create Control References Manually

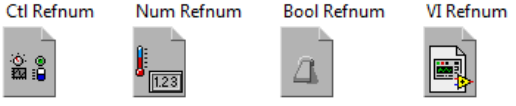


NATIONAL INSTRUMENTS | ni.com/training

122

Selecting the VI Server Class

- After you place a Control Refnum on the front panel of a subVI, specify the VI Server class of the control.
  - Right-click and select **VI Server Class** from the shortcut menu.
  - Alternatively, drag a control into a control refnum to specify the type.
- The class specifies the type of control references that the subVI accepts.



123

Summary—Quiz

1.For each of the following items, determine whether they operate on a VI class or a Control class.

a.Format and Precision

b.Visible

c.Reinitialize to Default Value

d.Show Tool Bar

Summary—Quiz Answer

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**

Summary—Quiz

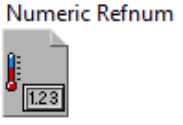
2.You have a Numeric control refnum in a subVI. Which control references could you wire to the control refnum terminal of the subVI?

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



Summary—Quiz Answer

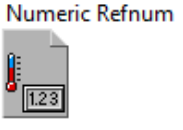
2.You have a Numeric control refnum in a subVI. Which control references could you wire to the control refnum terminal of the subVI?

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



Lesson 4
File I/O Techniques

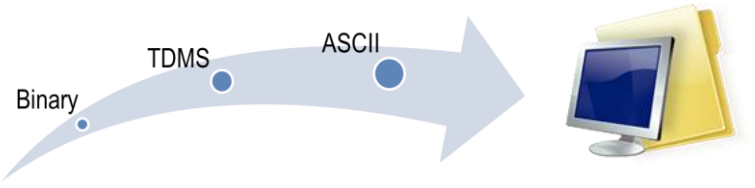
TOPICS

A. Compare File Formats
B. Create File and Folder Paths
C. Write and Read Binary Files
D. Work with Multichannel Text Files with Headers
E. Access TDMS Files in LabVIEW and Excel

A. Compare File Formats

Compare File Formats

At their lowest level, all files written to your computer's hard drive are a series of bits.



Compare File Formats

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 .

B. Create File and Folder Paths

Methods of Creating

Creating Relative Paths and Folders

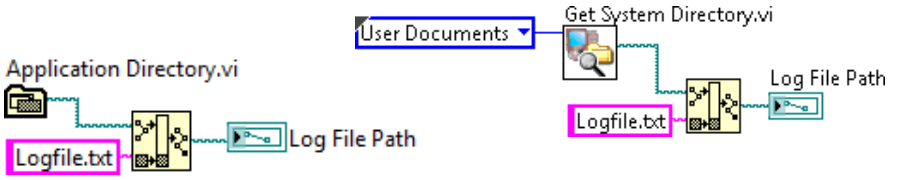
Dynamically Creating Filenames

Methods of Creating File and Folder Paths

- Hard-coded paths
  - Useful for quick prototypes.
  - Not recommended for applications.
- File Dialog
  - Allow user to specify the path to a file or directory.
  - Customize dialog options to limit options (*.txt).
- Programmatic creation
  - Create consistent filenames and extensions.
    Example: testdata_001.txt, testdata_002.txt, etc.
  - Specify a consistent location.

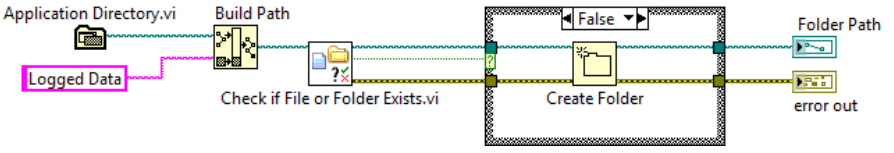
Creating Relative Paths

- Relative paths set paths relative to the application or system directory.
- Use the Application Directory VI to get project directory paths.
- Use the Get System Directory VI to get system directory paths.
- Paths differ based on the operating system and user.

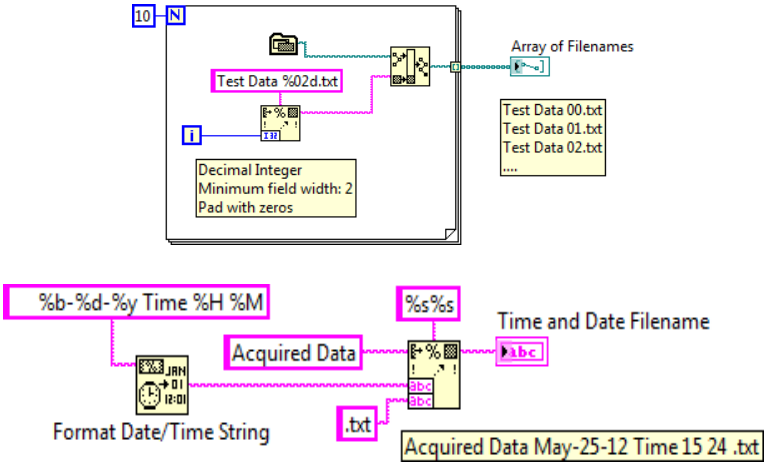


Creating Folders

Check if folder already exists before creating. Otherwise, attempting to create an existing folder will result in an error.



Dynamically Creating Filenames



C. Write and Read Binary Files

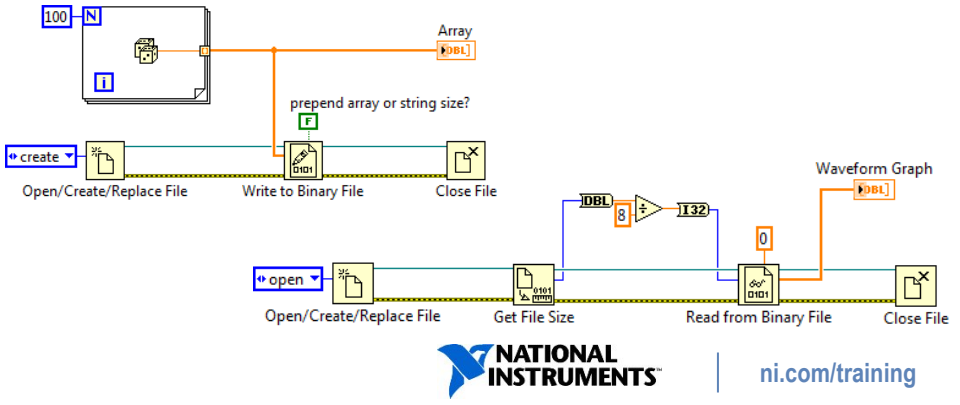
Using Binary File Functions

Demonstration

Using Binary File Functions

Use Binary File functions to read and write binary files.

- You can create custom file types.
- You must know the required file formats for your system.



145

Demonstrate Writing a Bitmap File

Use Binary File I/O to write a bitmap format file.

DEMONSTRATION

146

D. Work with Multichannel Text Files with Headers

Review of Text Files

Add Headers to the File

Write Multichannel Data

Read Data and Extract Information

Review of Text Files

- Use ASCII characters to store information in text files.
  - Each character (number, letter, punctuation) takes 1 byte.
- Many applications can open text files, including Excel.
- Files are typically larger and slower to write/read than binary.
- You cannot randomly access text files.

Creating Text Files with Headers

No Header Data

24.45	34.54
23.41	35.32
22.97	35.98
21.56	36.76

Tab

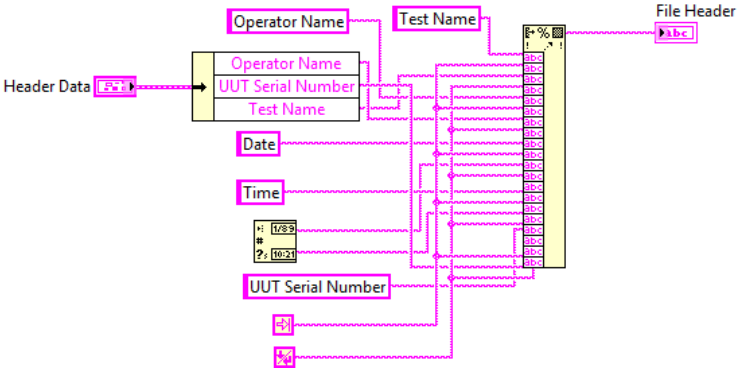
CR/LF

Header Data

Operator Name	David		
UUT S/N	A1234		
Test Name	Pressure		
Channel Name	Temperature	Pressure	
Units	Kelvin	PSI	
Max. Value	24.45	36.76	
	24.45	34.54	
	23.41	35.32	
	22.97	35.98	
	21.56	36.76	

149

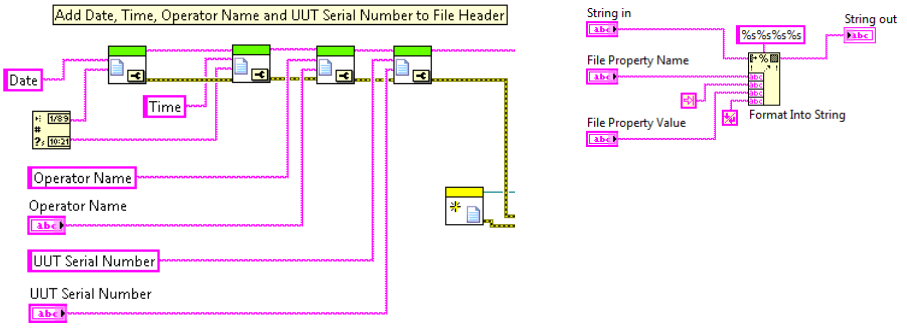
Creating Text Files with Headers—Hard Coding



Hard coding header information becomes difficult to maintain.

150

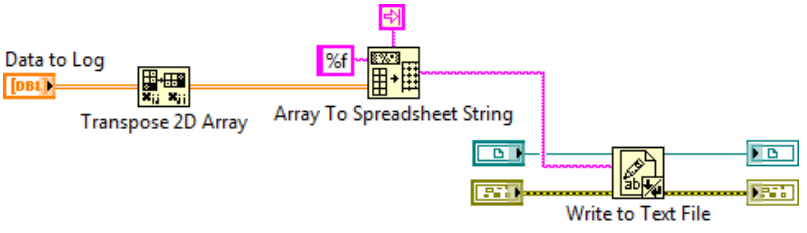
Creating Text Files with Headers—SubVIs



- Using subVIs to build strings is a scalable solution.
- With subVIs you can add and remove header data easily.
- The block diagram is more readable.

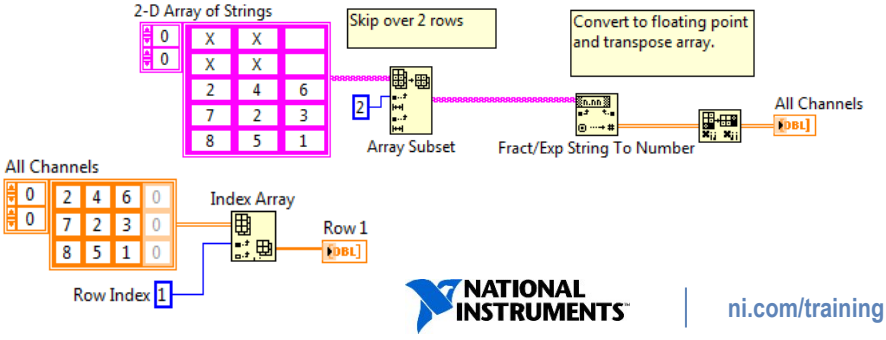
Writing Multiple Channels

- LabVIEW stores multi-dimensional arrays in row-major order.
  - Rows are identified by the first index of a 2-D array.
  - Columns are identified by the second index.
- Transpose data before writing to file to view channel data in column format.



Reading Channel Data

- Use Array Subset to skip over header information.
- Transpose data after reading to convert data back to row-major order.
- Use Index Array to extract one column or row of data.



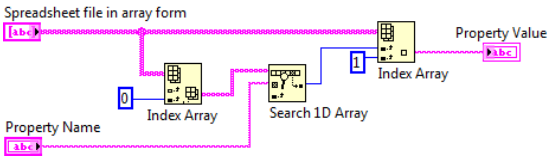
Read Data and Extract Information

- After reading a spreadsheet file, how do you extract information?
- How do you find a property value?
- How do you read the UUT Seria Number value?
- How do you extract channel dat for column 1?

Operator Name	Fred	
UUT Serial Number	A001	
Test Name	Stress Response	
Channel Data		
0	0.000000	0.000000
1	0.049068	0.309017
2	0.098017	0.587785
3	0.146730	0.809017
4	0.195090	0.951057
5	0.242980	1.000000

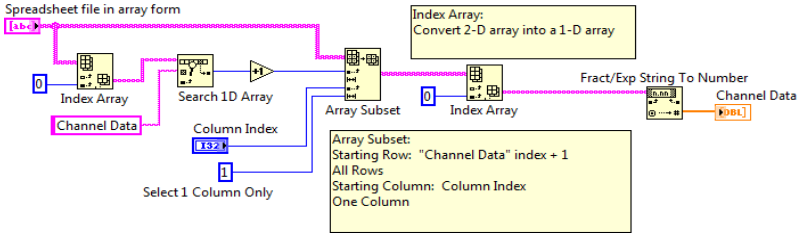
Given a Property Name, Find the Value

- Use Index Array to put column 0 in a 1-D array.
- Use Search 1D Array to find Property Name:
  - Return row index if found.
  - Return -1 if not found.
- Use Index Array to get element at row index and column 1.
- What should you do if Property Name is not found?



Extracting a Data Channel

- Identify row index of first data value.
- Use Array Subset to extract Data Channel.
  - Output is one channel of data in a 2-D string array.
- Convert to 1-D numeric array.



E. Access TDMS Files in LabVIEW and Excel

TDMS File Format

Data Hierarchy and Properties

TDMS Functions

File Viewer

TDMS File Format

The Technical Data Management Streaming (TDMS) file format contains two types of data:

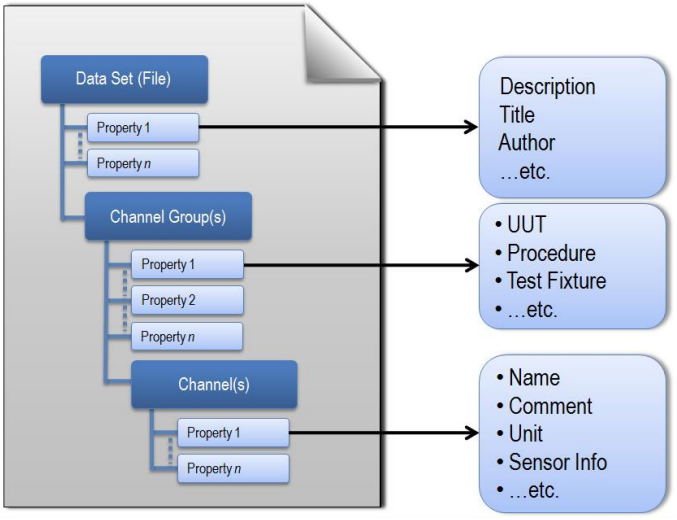
- Meta data – Names and properties
- Raw data – Measurement data in binary format

Use TDMS files for the following purposes:

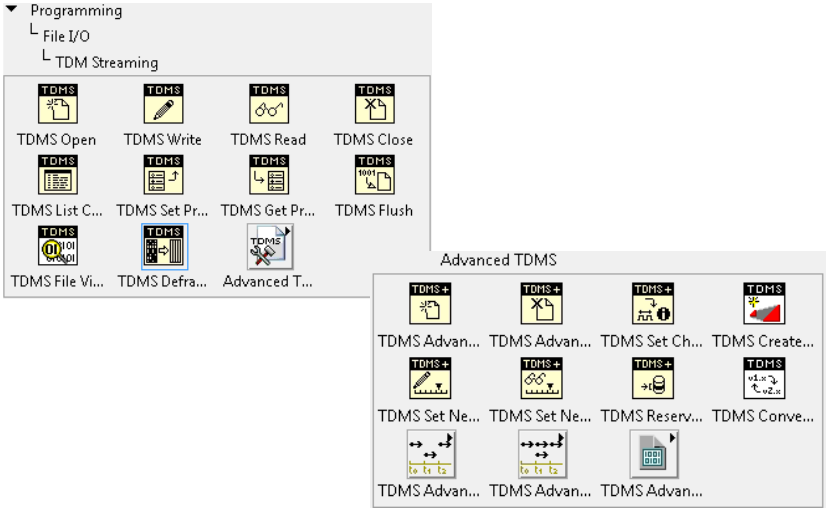
- To store test or measurement data.
- To create structures for grouping your data.
- To store headers/properties about your data.
- To read and write data at high speeds.

A variety of applications, including Microsoft Excel, can access TDMS files.

TDMS Files—Data Hierarchy and Properties

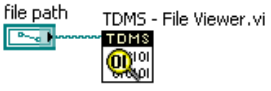


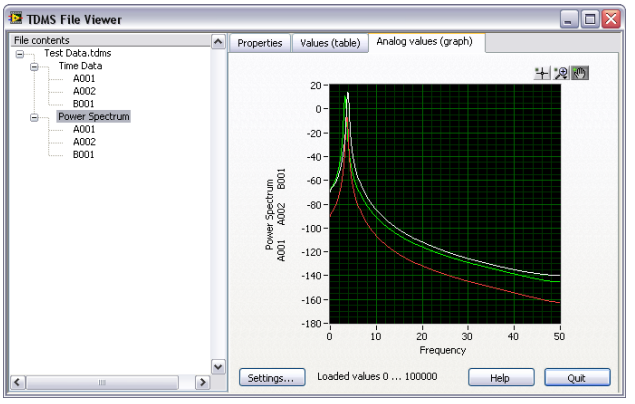
TDMS Functions



TDMS Files—File Viewer

Open TDMS files and present the file data in the TDMS File Viewer dialog box.





Exercise 4-3
Write and Read TDMS Files

Learn how to read data from a TDMS file.

GOAL

Summary—Quiz

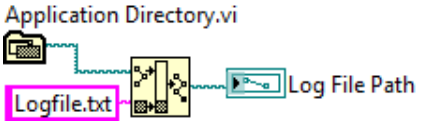
1.Consider the following code. The resulting Log File Path contains a text file path in which folder?

a)Same folder as VI that executed the code.

b)Same folder as the LabVIEW project.

c)Current user's AppData directory.

d)Same folder as the Application Directory VI.



Summary—Quiz Answer

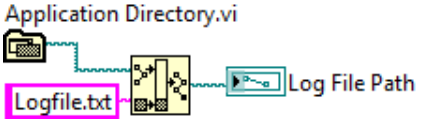
1.Consider the following code. The resulting Log File Path contains a text file path in which folder?

a)Same folder as VI that executed the code

b)Same folder as the LabVIEW project

c)Current user's AppData directory

d)Same folder as the Application Directory VI



Summary—Quiz

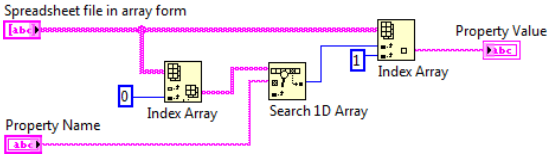
2.What index value is returned from Search 1D Array function if Property Name is not found in the input array?

a)NaN (Not a Number)

b)0

c)-1

d)Negative Infinity



Summary—Quiz Answer

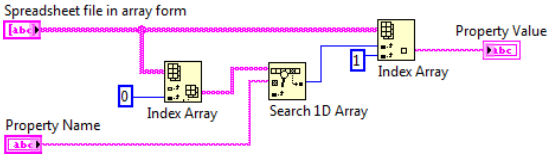
2.What index value is returned from Search 1D Array function if Property Name is not found in the input array?

a)NaN (Not a Number)

b)0

c)-1

d)Negative Infinity



Summary—Quiz

3.You need to store data that 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

Summary—Quiz Answer

3.You need to store data that 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

Summary—Quiz

4.TDMS files store properties at which of the following levels?

a)File

b)Channel Group

c)Channel

d)Value

Summary—Quiz Answer

4.TDMS File store properties at which of the following levels?

a)File

b)Channel Group

c)Channel

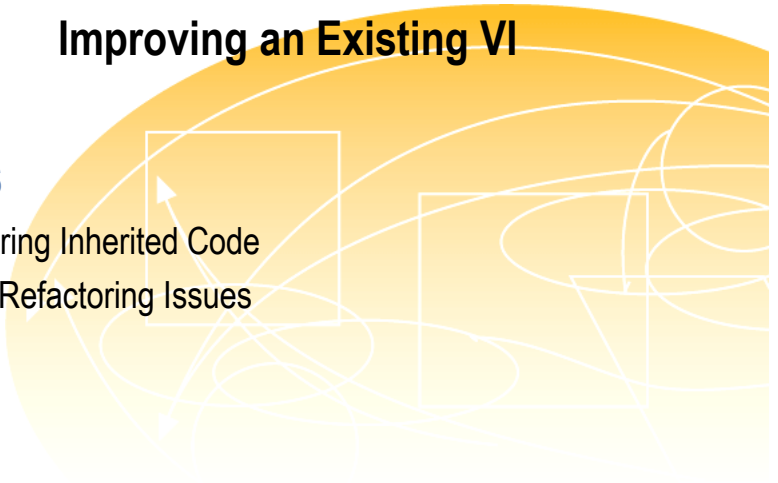
d)Value

Lesson 5
Improving an Existing VI

TOPICS

A. Refactoring Inherited Code

B. Typical Refactoring Issues



A. Refactoring Inherited Code

Definition

When To Refactor

The Refactoring Process

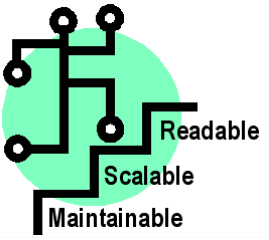
Refactoring Inherited Code

Inherited VIs may be poorly designed, making it difficult to add features later in the life of the VI.

Refactoring:

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

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



When to Refactor

•When you are adding a feature to a VI or debugging it.

•Good candidates for complete rewrites:

−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.

B. Typical Refactoring Issues

Disorganization
Poor Naming
Overly Complicated
Duplicated Logic
Breaks Dataflow
Outdated Practices

Disorganization

The block diagram is too disorganized, too big, or includes too many nested structures

Solution:

- Move objects within the block diagram.
- Create subVIs to make it smaller and more organized.
- Place comments to improve readability.

Poor Naming

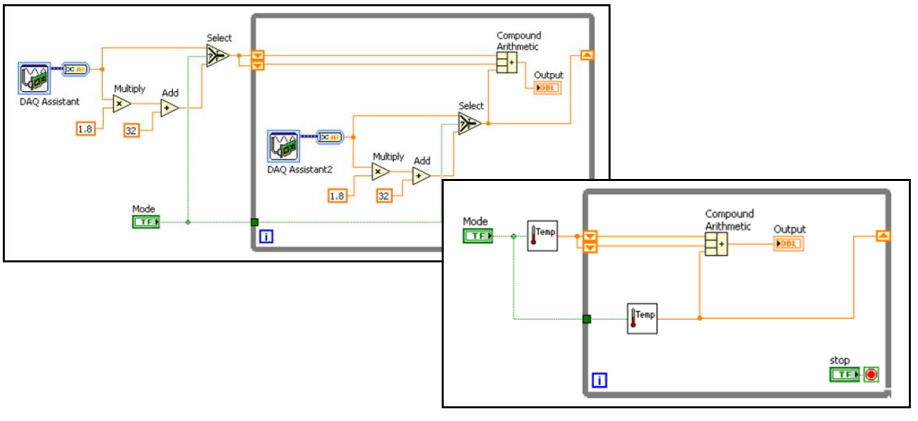
The block diagram uses incorrect object names and poor icons.

Duplicate Logic

The block diagram uses duplicate logic.

Solution:

Refactor the VI by creating a subVI for the duplicated logic.



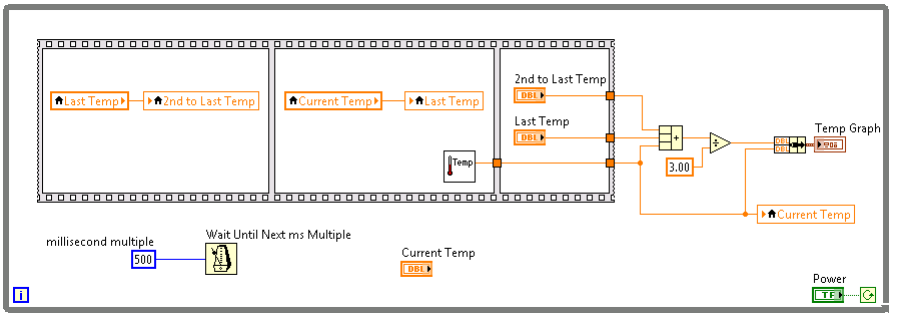
Breaks Dataflow

The block diagram does not use dataflow programming.

Solution:

•Replace Sequence structures with state machines.

•Delete local variables and wire directly to controls or indicators.



184

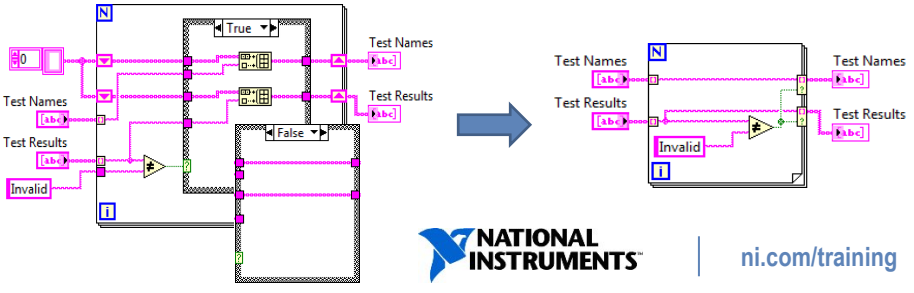
Outdated Practices

The VI was created in an earlier version of LabVIEW and has outdated practices.

Solutions:

•Replace polling-based design with event-based design.

•Use new features that simplify code.



NATIONAL INSTRUMENTS | ni.com/training

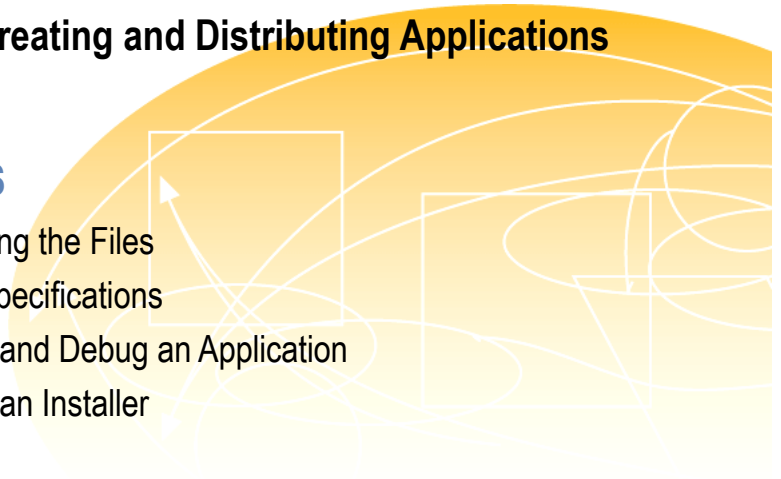
185

Summary—Refactoring Checklist

Use the following refactoring checklist to help determine if you should refactor a VI:

- ❑VI is too disorganized, too big, or includes too many nested structures.
- ❑VI uses incorrect object names and poor icons.
- ❑VI uses unnecessary logic or has complicated algorithms.
- ❑VI has duplicated logic.
- ❑VI does not use dataflow programming.
- ❑VI uses outdated development practices.

Lesson 6
Creating and Distributing Applications



TOPICS

A. Preparing the Files
B. Build Specifications
C. Create and Debug an Application
D. Create an Installer

A. Preparing the Files

Building Applications Checklist

VI Properties

Paths and System Paths

Quit Application

Preparing the Files

To create a professional, stand-alone application for your VIs, you must first prepare your files:

- Recompile and save changes to VIs.
- Verify desired VI Properties settings.
- Ensure paths generate correctly.
- Conditionally call the Quit LabVIEW function.

Building Applications Checklist

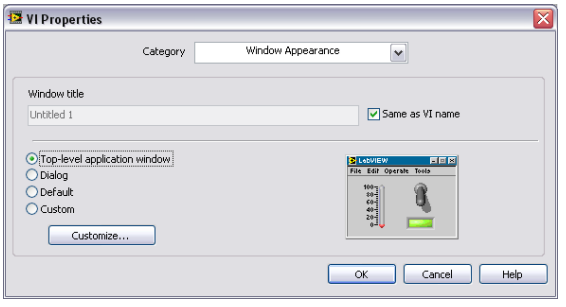
Refer to the *Building Applications Checklist* topic of the *LabVIEW Help* for more information about:

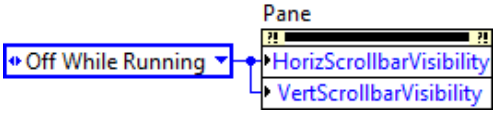
- Preparing files.
- Configuring build specifications.
- Distributing builds.

Preparing the Files – VI Properties

To set VI Properties settings:

- Manually edit the **VI Properties** dialog box.
- Programmatically change VI Properties using VI Server.

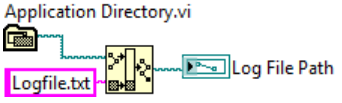




Preparing the Files – Paths

Set paths relative to Application Directory VI path.

From a stand-alone application	The path to the folder containing the application executable
From a LabVIEW project (.lvproj) in the LabVIEW Development Environment	The path to the project folder



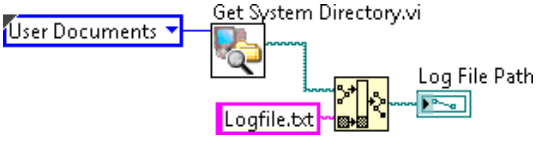
Preparing the Files – System Paths

- Use the Get System Directory VI to get system directory paths.
- Paths differ based on the operating system and user.
- Windows XP User Documents:

```
-C:\Documents and Settings\<user>\My Documents
```

- Windows 7 User Documents:

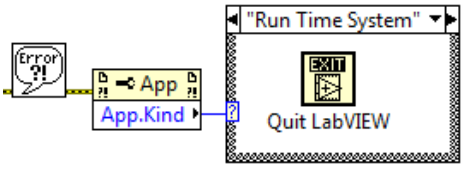
```
-C:\Users\<user>\Documents
```



Preparing the Files – Quit Application

Programmatically quit your application using the Quit LabVIEW Function.

To conditionally call the Quit LabVIEW function, use the App.Kind property.



Exercise 6-1A
Preparing Files for Distribution

- Review the Building Applications Checklist.
- Prepare VIs to build a standalone application.

GOAL

Exercise 6-1A
Preparing Files for Distribution

What are some of the changes you observed when you ran your VI after changing the Window Appearance properties?

DISCUSSION

198

B. Build Specifications

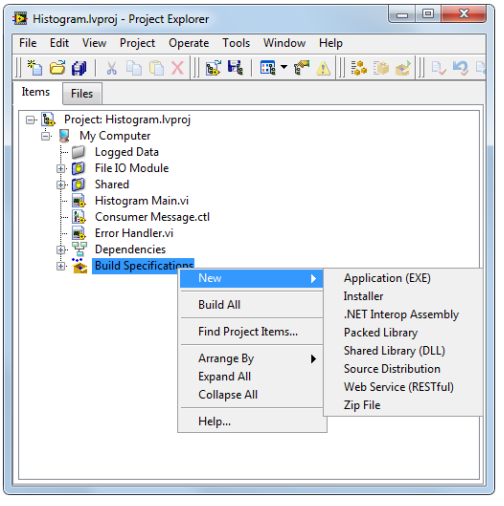
Build Specifications

A build specification contains all the settings for the build, such as files to include, directories to create, and settings for VIs.

Why Use Build Specifications?

Use build specifications to build the following:

- Stand-alone applications
- Installers
- Source distributions
- Zip files
- Shared libraries
- Packed Project libraries
- .NET Interop Assemblies
- Web services



C. Create and Debug an Application

System Requirements for Applications

Configure the EXE Build Specification

Build and Run the EXE

Debug the EXE

System Requirements for Applications

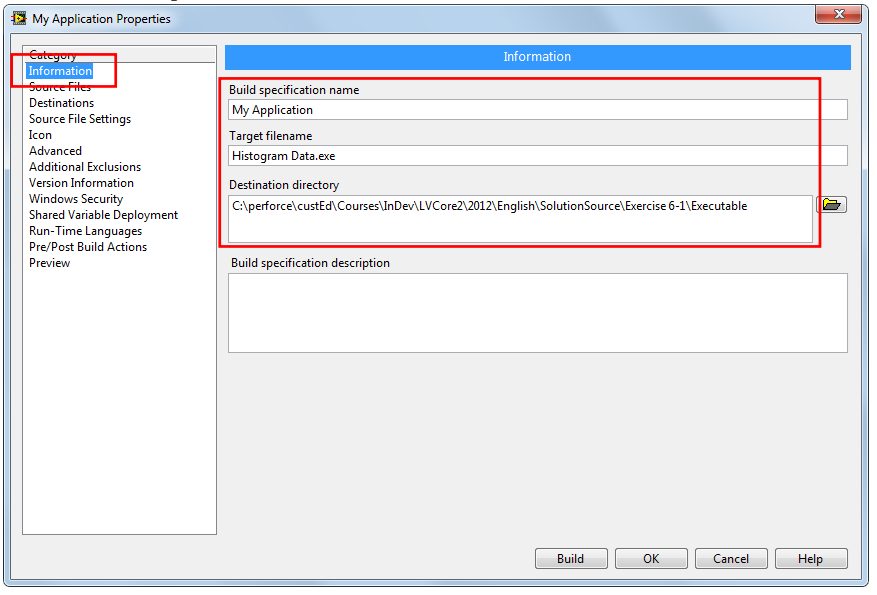
- Applications that you create with Build Specifications generally have the same system requirements as the LabVIEW development system used to create the VI or application.
- Memory requirements vary depending on the size of the application created.

Configure the EXE Build Specification

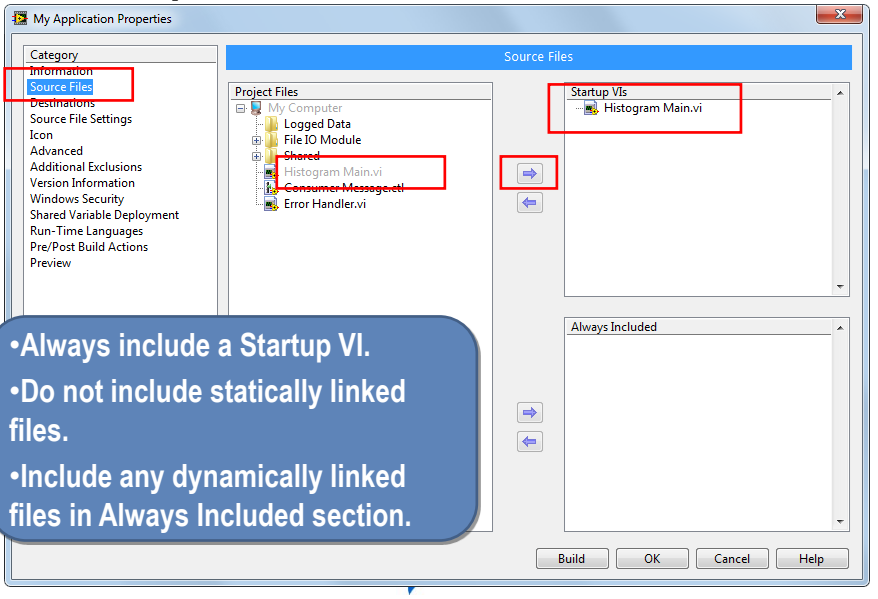
In the **Application Properties** dialog box:

- Specify name of executable.
- Specify known destination for generated executable files
- Identify a startup VI and include any dynamically linked files.
- Enable debugging, if desired.
- Preview the build.

EXE Properties – Information



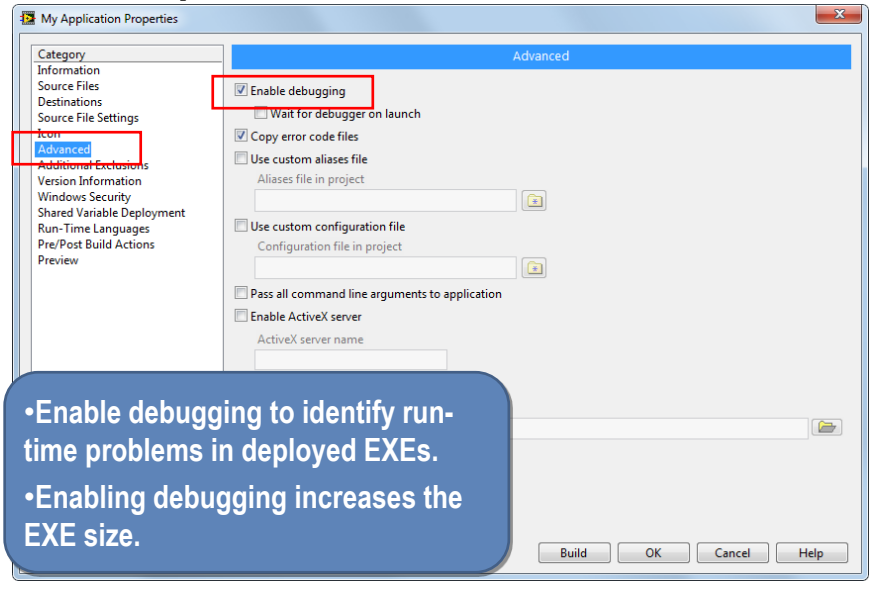
EXE Properties – Source Files



- Always include a Startup VI.
- Do not include statically linked files.
- Include any dynamically linked files in Always Included section.

206

EXE Properties – Advanced



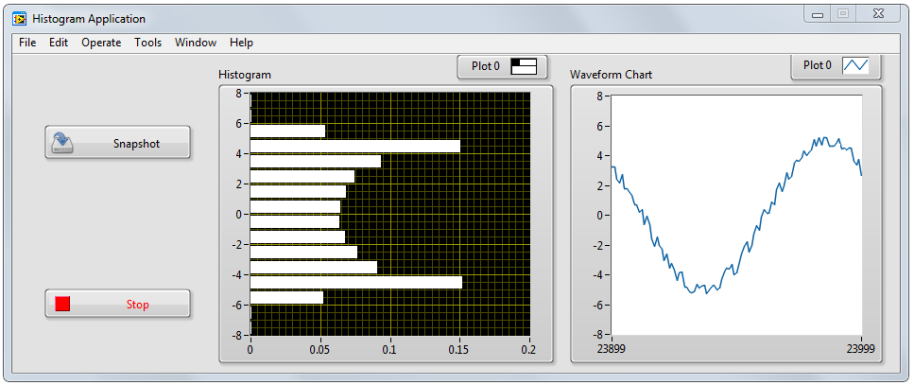
- Enable debugging to identify run-time problems in deployed EXEs.
- Enabling debugging increases the EXE size.

207

208

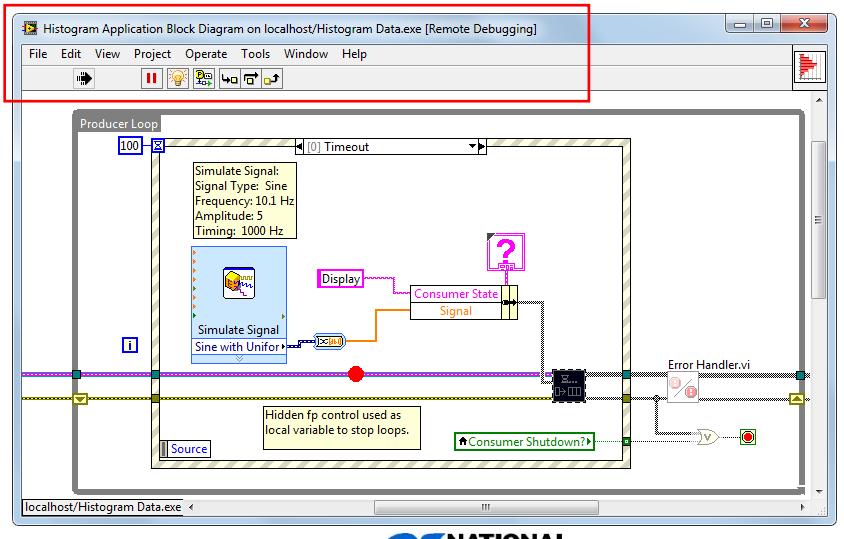
209

Run EXE and Verify Execution



212

Debug the EXE from LabVIEW



213

Why Executables Might Behave Differently

- File paths may change, which can lead to errors.
- Drivers or support files are missing.
- System resources like memory or CPU speed may differ, which can lead to timing changes.
- The application INI file differs from the LabVIEW INI file.
- Not all features are supported in the Run-Time Engine.

D. Create an Installer

Why Create an Installer?

Configure the Installer Build Specifications

Deploy the Application to Another Machine

Debug the Executable

Why Create an Installer?

- Executables need the LabVIEW Run-Time Engine (RTE) to execute on a target system.
- If an application relies on drivers, those drivers need to be installed on the target system.
- An installer ensures files are copied to the right places.
- Professional applications use installers.

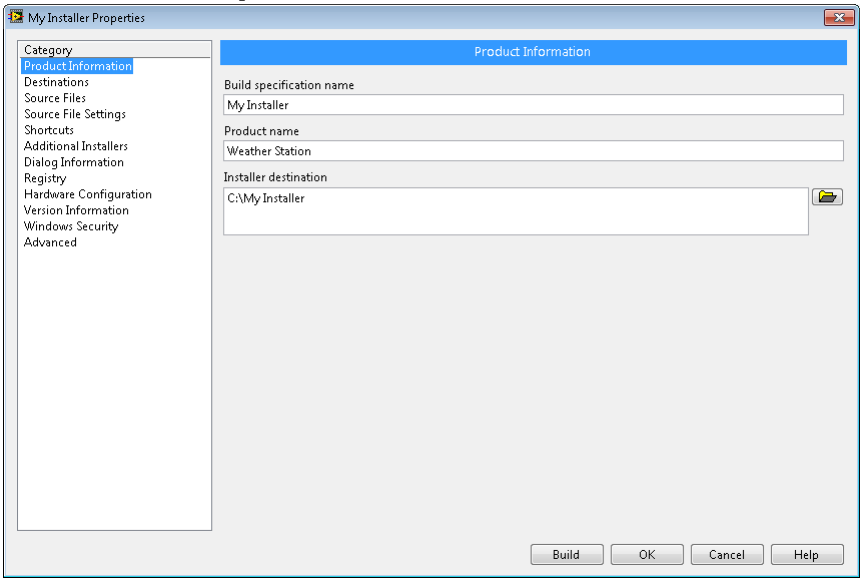
Configuring Installer Build Specifications

In the **Installer Properties** dialog box:

- Include the LabVIEW Run-Time Engine.
  - From the Additional Installers page, select the **NI LabVIEW Run-Time Engine.**
- Include drivers used in the application.
  - For example, if you use DAQmx VIs in your application, then include the NI-DAQmx driver.

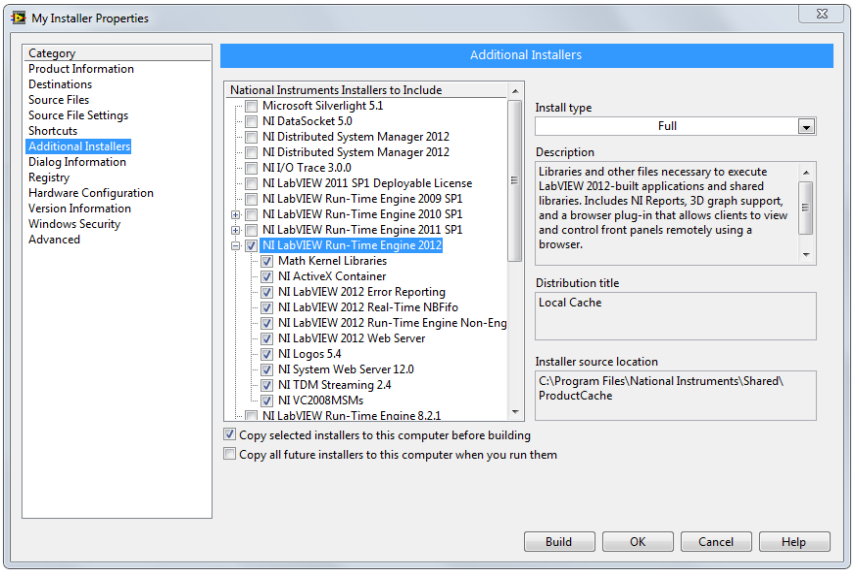
Refer to the *Caveats and Recommendations for Building Installers* topic of the *LabVIEW Help* for more information.

Installer Properties – Product Information



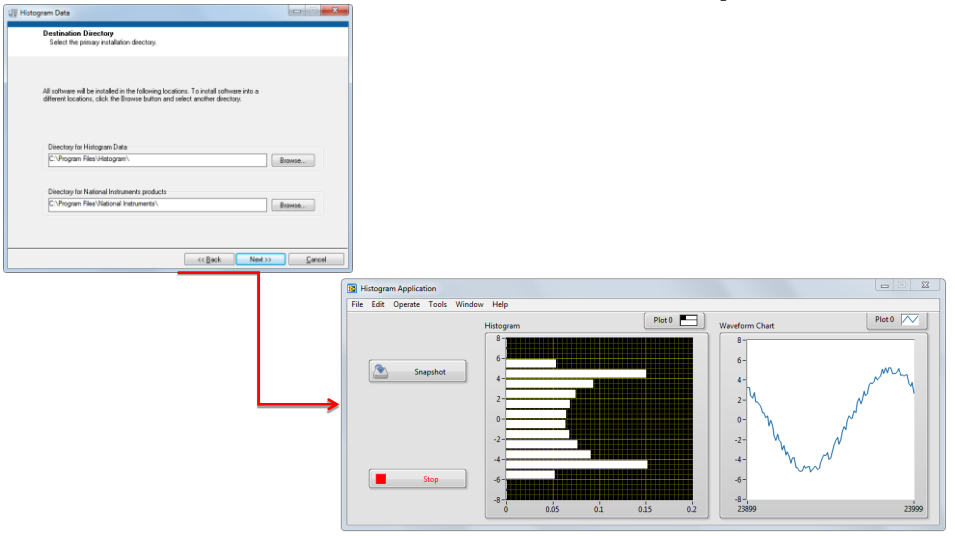
220

Installer Properties – Additional Installers



221

Install and Run on Destination Computer



Debug Executable on Destination Computer

1.Find IP address of destination computer.

–Type "`ipconfig`" in command prompt window of the destination computer to discover IP address.

2.Launch LabVIEW on development machine.

3.Enter the IP address or machine name in **Debug Application or Shared Library** dialog box.

4.Click the **Connect** button in the **Debug Application or Shared Library** dialog box.

Control is transferred to development machine.

5.Debug the executable using LabVIEW debugging tools.

Summary

- The Application Builder enables you to create stand-alone applications and installers.
- To create a professional, stand-alone application you must understand:
  - The architecture of your application.
  - The programming issues particular to your application.
  - The application building process.
  - The installer building process.
- Test your application and installer regularly as you develop.

Thank you!