LabVIEW™ Core 2 Participant Guide

Course Software Version 2014
November 2014 Edition
Part Number 326293A-01

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at ni.com/trademarks for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and vernier.com are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix®, Tek, and Tektronix, Enabling Technology are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or the *National Instruments Patent Notice* at ni.com/patents.

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code feedback.

Table of Contents

Student Guide

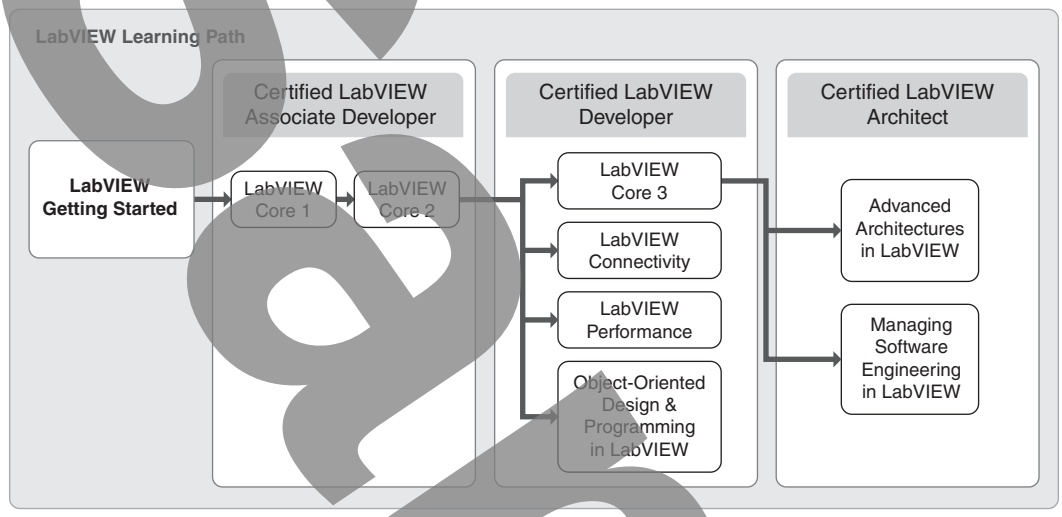
In this section you will learn about the LabVIEW Learning Path, the course description, and the items you need to get started in the LabVIEW Core 2 course.

Topics

+ NI Certification
+ Course Description
+ What You Need to Get Started
+ Installing the Course Software
+ Course Goals

A. NI Certification

The LabVIEW Core 2 course is part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for the NI Certified LabVIEW Associate Developer exam (CLAD). The following illustration shows the courses that are part of the LabVIEW training series. Refer to ni.com/training for more information about NI Certification.



B. Course Description

The LabVIEW Core 2 course teaches you programming concepts, techniques, features, VIs, and functions you can use to create test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications. This course assumes that you are familiar with Windows and that you have experience writing algorithms in the form of flowcharts or block diagrams.

The Participant Guide is divided into lessons. Each lesson contains the following:

- An introduction with the lesson objective and a list of topics and exercises.
- Slide images with additional descriptions of topics, activities, demonstrations, and multimedia segments.
- A set of exercises to reinforce topics. Some lessons include optional and challenge exercises.
- A lesson review that tests and reinforces important concepts and skills taught in the lesson.

Note For course and exercise manual updates and corrections, refer to ni.com/info and enter the Info Code core2.

Several exercises use a plug-in multifunction data acquisition (DAQ) device connected to a DAQ Signal Accessory or BNC-2120 containing a temperature sensor, function generator, and LEDs. If you do not have this hardware, you still can complete the exercises. Alternate instructions are provided for completing the exercises without hardware. You also can substitute other hardware for those previously mentioned. For example, you can use another National Instruments DAQ device connected to a signal source, such as a function generator.

C. What You Need to Get Started

Before you use this course manual, make sure you have all of the following items:

- ☐ Computer running Windows 7/Vista/XP
- ☐ Multifunction DAQ device configured as Dev1 using Measurement & Automation Explorer (MAX)
- ☐ DAQ Signal Accessory or BNC-2120, wires, and cable
- ☐ LabVIEW Professional Development System 2014 or later
- ☐ DAQmx 14.0 or later
- ☐ *LabVIEW Core 2* course CD, from which you install the following folders:

Directory	Description
`Exercises`	Contains VIs used in the course
`Solutions`	Contains completed course exercises

D. Installing the Course Software

Complete the following steps to install the course software.

1. Insert the course CD in your computer. The **LabVIEW Core 2 Course Setup** dialog box appears.
2. Click **Install the course materials**.
3. Follow the onscreen instructions to complete installation and setup.

Exercise files are located in the `<Exercises>\LabVIEW Core 2\` folder.

Note Folder names in angle brackets, such as `<Exercises>`, refer to folders on the root directory of your computer.

E. Course Goals

This course prepares you to do the following:

- Apply common design patterns that use queues and events
- Use event programming effectively
- Programmatically control user interface objects
- Evaluate file I/O formats and use them in applications
- Modify existing code for improved usability
- Prepare, build, debug, and deploy stand-alone applications

This course does *not* describe any of the following:

- LabVIEW programming methods covered in the *LabVIEW Core 1* course
- Every built-in VI, function, or object. Refer to the *LabVIEW Help* for more information about LabVIEW features not described in this course.
- Developing a complete application for any student in the class. Refer to the NI Example Finder, available by selecting **Help»Find Examples**, for example VIs you can use and incorporate into VIs you create.

6 Improving an Existing VI

In this lesson you will learn methods to refactor inherited code and experiment with typical issues that appear in inherited code.

Topics

+ Refactoring Inherited Code
+ Typical Refactoring Issues

Exercises

Exercise 6-1 Refactoring a VI

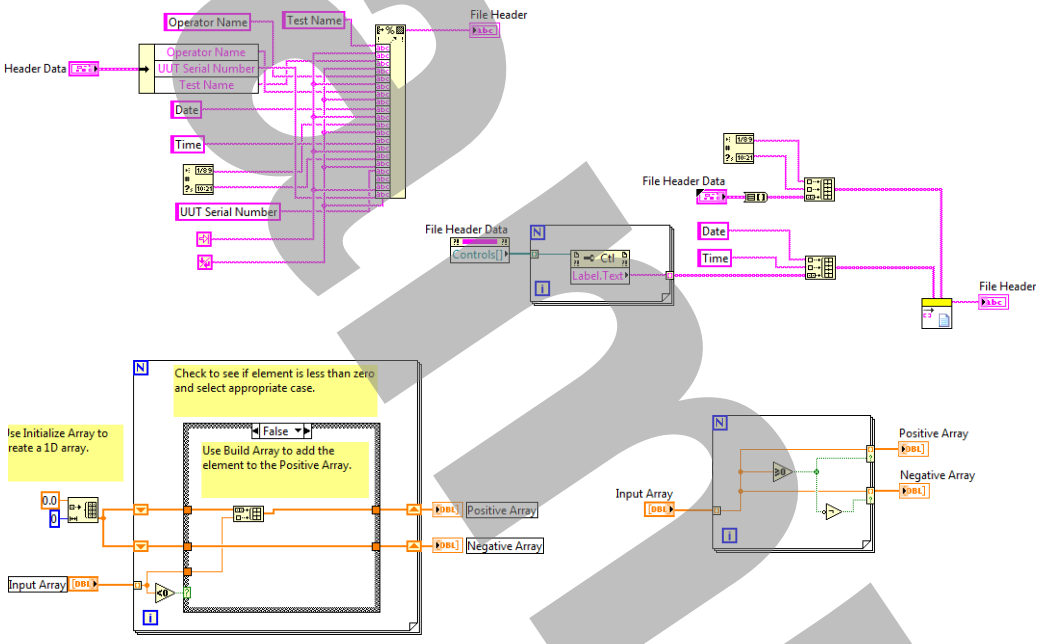
A. Refactoring Inherited Code

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

Definition

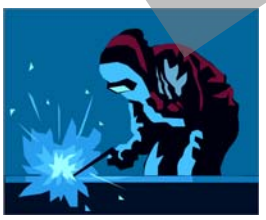
Refactoring The process of redesigning software to make it more readable and maintainable so that the cost of change does not increase over time.

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.

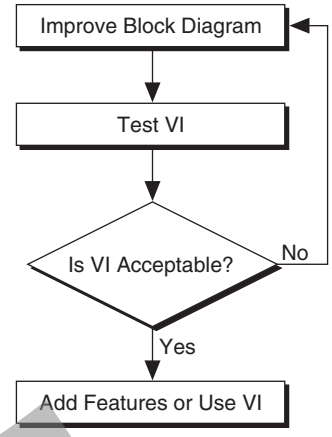


Good candidates for complete rewrites include the following types of VIs:

- 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.

Poor

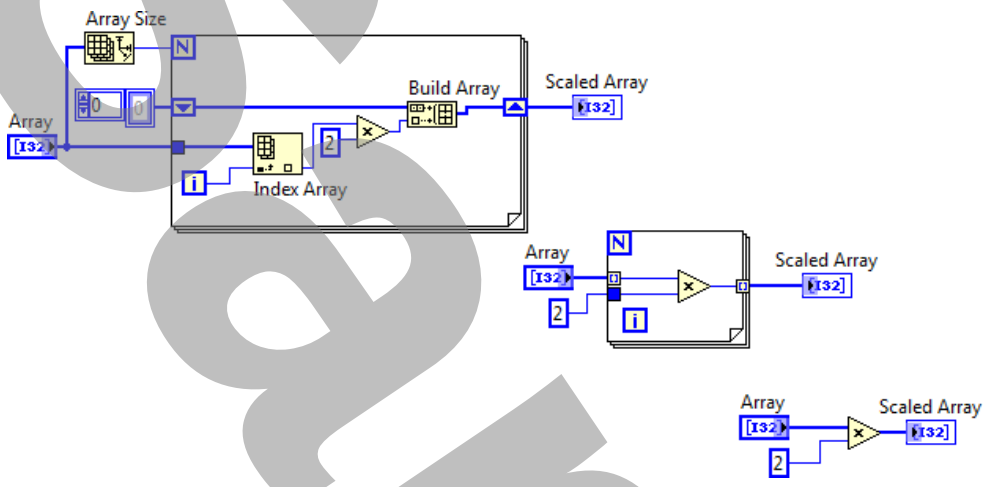
Better

Best

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

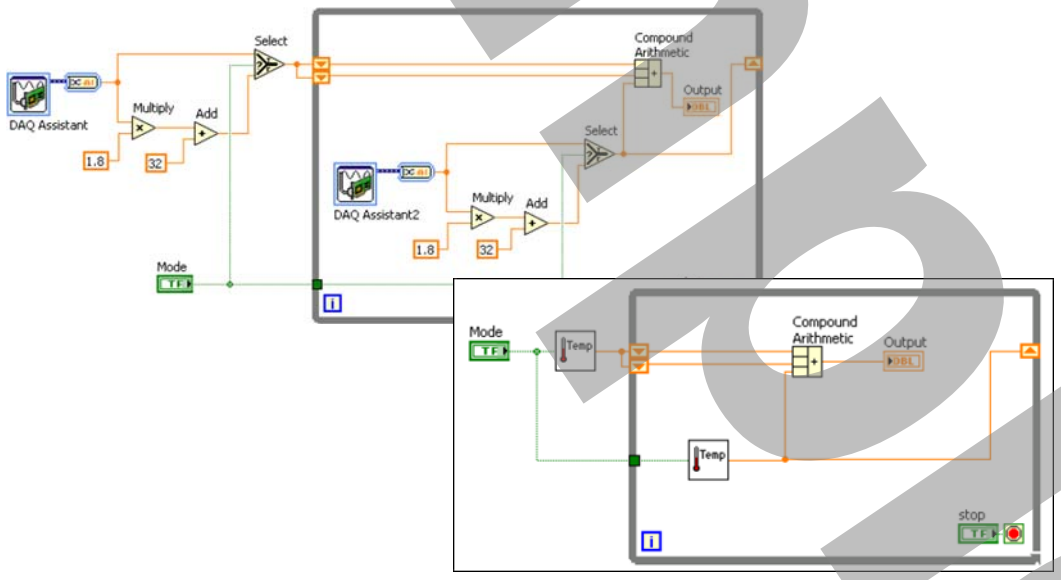
Overly Complicated

The following figure contains three versions of the same task. The top block diagram is overly complicated and contains unnecessary logic. The other two block diagrams simplify the task. Each section of code performs the same functionality. However, the code at the lower right is much easier to read and maintain.



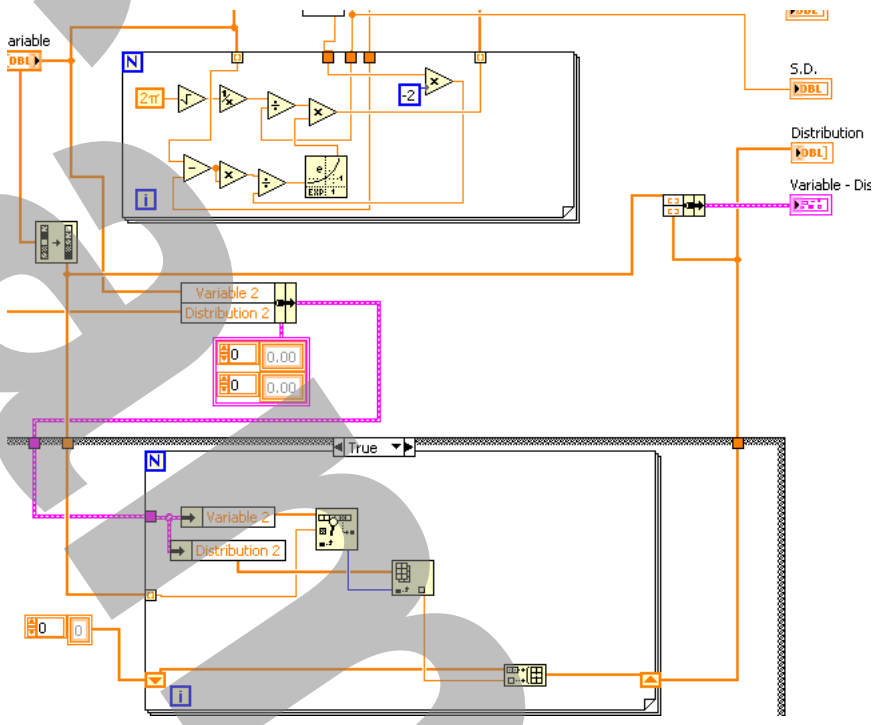
Duplicate Logic

If a VI contains duplicated logic, you always should refactor the VI by creating a subVI for the duplicated logic. This can improve the readability and testability of the VI. The example in the following figure shows how you can create a subVI from sections of code that are reused in two places on the block diagram.



Disorganization

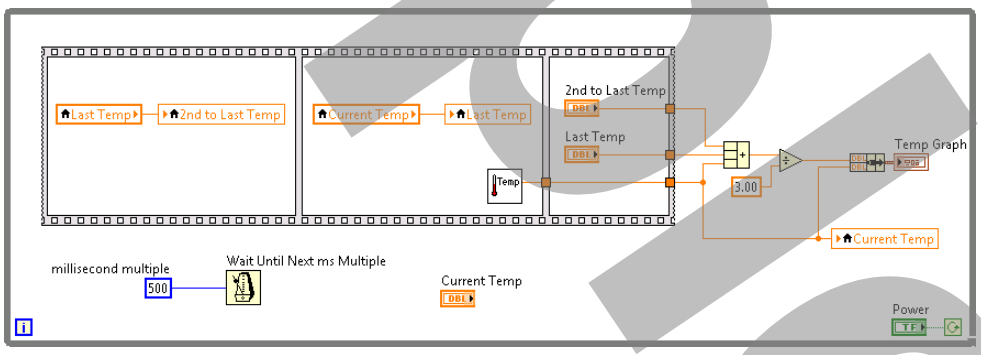
Diagrams that require scrolling in both directions are often too large and can benefit from refactoring. Improve the readability of a disorganized VI by relocating objects within the block diagram. You also can create subVIs for sections of the VI that are disorganized. Place comments on areas of a VI that are disorganized to improve the readability of the VI.



Tip You can use the Clean Up Diagram tool to help organize the block diagram, but it will not address all issues. For example, the Clean Up Diagram tool will not help remove deeply nested structures.

Breaks Dataflow

The block diagram does not use dataflow programming.



Solution

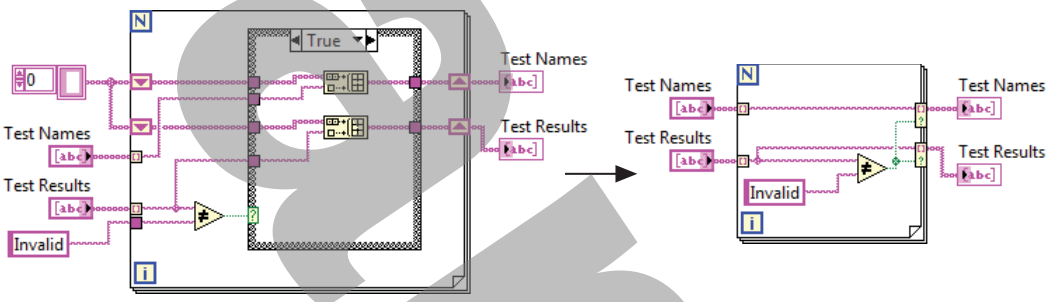
- Replace Sequence structures with state machines.
- Delete local variables and wire directly to controls or indicators.

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.



Exercise 6-1 Refactoring a VI

This exercise consists of five VIs that you can evaluate for ways to improve. Look over each option and choose one or two to complete during the time allotted in class. The options are listed from easiest to hardest in terms of refactoring.

Select from the following options to practice refactoring LabVIEW code:

- SubVIs to For Loops
- Array Manipulation
- Polling to Events
- Format Into String
- String Formatting

SubVIs to For Loops

Goal

To take an existing VI and make it more readable, scalable, and maintainable.

Description

In the course of the development of a LabVIEW application, there are times when VIs or sections of VIs end up being written “badly”.

Scenario

Your customer is a research facility that is doing experiments on superconducting material. The researchers must perform experiments at very low temperatures. The materials are tested in a chamber that contains four temperature sensors spread throughout the chamber. The sensors return temperatures in °C. Due to the low temperatures involved, the temperatures in °C are less readable than K. For this reason the customer’s application already includes a VI that converts the temperatures from °C to K.

The customer has recently decided to monitor more than four temperatures. He is worried that every time he increases the number of temperatures he will have to update the VI that does the conversion. In this exercise you will refactor the conversion VI to make it more scalable. You also will make the VI more readable and maintainable.

Note The Kelvin scale defines Absolute Zero as the lowest temperature possible. No temperature below Absolute Zero is allowed. Absolute Zero is approximately equal to -273 °C. You should build your refactored application to generate errors if the user tries to convert invalid temperatures, for example, temperatures less than -273 °C.

Implementation

Open the Convert Temperatures VI located in the `<Exercises>\LabVIEW Core 2\Refactoring\Use subVIs_ForLoop` and refactor it.

Hints

- Find repeated code and replace it with subVIs.
- Find code that works on a limited number of elements of an array and scale it to work on an unlimited number of elements.
- Clean up the block diagram of the VI to make it readable.
- Organize subVIs and related files in a project.

Test

Test your refactored code to ensure that it works as the original application did. Also ensure that the refactored application generates errors if the user tries to convert invalid temperatures.

Array Manipulation

Goal

Refactor a VI that uses an outdated technique for conditionally separating an array into multiple arrays.

Description

Each release of LabVIEW introduces new features that improve coding efficiencies. Therefore, you might refactor code you inherited from someone who developed the code in an earlier version of LabVIEW.

Scenario

You inherit an old LabVIEW application which uses outdated programming techniques.

Implementation

Open `Separate Array Values.vi` from the Array Manipulation project located in the `<Exercises>\LabVIEW Core 2\Refactoring\Array Manipulation` directory and refactor it.

Hints

Conditional auto-indexing allows you to conditionally build an array within a For Loop.

Test

Test your refactored code to ensure that it works as the original application did. Notice that the input array contains a mix of positive and negative values. After running, the Positive Array contains positive values while the Negative Array contains negative values.

Polling to Events

Goal

To take an existing VI that uses outdated techniques and refactor it to be more readable, scalable, and maintainable.

Description

A lot of existing LabVIEW code was written using practices which were standard and accepted in the past but which were discovered to be less than ideal in terms of readability, scalability, and maintainability.

Scenario

You inherit an old LabVIEW application which performs the following functions:

- Acquire a waveform as a Time Series.
- Calculate the FFT of the waveform (that is, generate the Spectrum).
- Calculate the Max and Min values of the Waveform.

The waveform and spectrum are displayed in separate Waveform Graph indicators as are the Max and Min values.

You are asked to add a feature to calculate the Standard Deviation of the Time Series. You notice that the block diagram of the VI is built in such a way that adding more features makes the block diagram larger and more difficult to read.

Implementation

Open the Waveform Analysis (Polling) VI located in the `<Exercises>\LabVIEW Core 2\Refactoring\Polling to Events` directory and refactor it.

Hints

- Use Events instead of Polling.
- Use Shift Registers instead of Local Variables.
- Use a Project to organize the files.

Test

Test your refactored code to ensure that it works as the original application did.

Format Into String

Goal

Refactor a VI that uses the Format Into String function to make the VI more scalable.

Description

The Format Into String function is very versatile; it converts multiple pieces of data into a string according to a format string. However, if new parameters are introduced, both the Format Into String function and the format string must be modified.

You can add parameters without changing the VI if all the parameters are of the same data type.

Scenario

You inherit an old LabVIEW application which is difficult to scale and maintain. You notice that it uses individual controls to input information into a Format Into String function.

Implementation

Open `Format Gas Params.vi` from the Format Gas Parameters project located in the `<Exercises>\LabVIEW Core 2\Refactoring\Format Into String` directory and refactor it.

☐ Assume you need to add a new DBL parameter (for example, Explosiveness).

☐ Notice that the Format Into String node needs to be expanded.

☐ Also notice that the format string needs to have `\r\nExplosiveness:\s%f` added.

Hints

- Arrays of parameter values are easily expandable to include future values.
- Some string functions can handle arrays.

Test

Test your refactored code to ensure that it works as the original application did.

String Formatting

Goal

Refactor a VI that uses the Format Into String function to make it more scalable.

Description

The Format Into String function is very versatile; it converts multiple pieces of data into a string according to a format string. However, if new parameters are introduced, both the Format Into String function and the format string must be modified.

Scenario

You inherited some code that creates a file header and includes a series of name-value pairs for your test data. Because the file is expected to be loaded into Excel, each name and value is separated by a tab and terminated with an End of Line character. In addition to time and date information, the file header also includes information contained in a cluster. The cluster element names and values are used in the name-value pairs.

Your manager wants to re-order the name-value pairs so that Date and Time appear first. In the future, you may want to expand the number of elements in the File Header Data cluster from 3 elements to 10 elements. You must update the code to change the order and prepare for future scalability of the cluster elements.

Implementation

Open the Generate File Header VI in the Format File Header project located in the `<Exercises>\LabVIEW Core 2\Refactoring\String Formatting` directory and refactor it.

Hints

- Create a subVI which formats each name-value pair. Separate the name and value using a Tab constant and terminate each pair with an End of Line constant.
- Process the list of name-value pairs. The challenge is to create two parallel arrays, one for names and one for values.
- If all cluster elements are of the same data type, you can convert a cluster to an array using the Cluster to Array function. You can then use a For Loop to process each cluster element.
- Use a control property node to get a list of control references to all the cluster elements. You can then get access to the Label names of the cluster elements. Use that to build an array of names.

Test

Test your refactored code to ensure that it works as the original application did.

End of Exercise 6-1

Job Aid—Refactoring Checklist

Use the following refactoring checklist to help determine if you should refactor a VI. If you answer yes to any of the items in the checklist, refer to the guidelines in the *When to Refactor* section of this lesson to refactor the VI.

- ☐ Disorganized block diagram
- ☐ Overly large block diagram
- ☐ Poorly named objects and poorly designed icons
- ☐ Unnecessary logic
- ☐ Duplicated logic
- ☐ Lack of dataflow programming
- ☐ Complicated algorithms

Additional Resources

Learn More About	LabVIEW Help Topic
Replacing polling with events	*Events in LabVIEW*
Reducing duplicate logic	*Creating SubVIs from Selections*
Creating quality code	*LabVIEW Style Checklist*