# LabVIEW™ Core 2 Exercises

Course Software Version 2012
August 2012 Edition
Part Number 325293D-01

## Worldwide Technical Support and Product Information

ni.com

## Worldwide Offices

Visit ni.com/niglobal to access the branch office Web sites, 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

To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.

# Contents

# 5

# Improving an Existing VI

## Exercise 5-1 Concept: Refactoring Inherited Code

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

Select from the following options to practice refactoring LabVIEW code:

- SubVIs to For Loops
- Array Manipulation VI
- 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 would 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.

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

## 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.
- Cleanup a 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 VI

## 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.

## Implementation

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

2. Test the VI.

   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.

## Hints

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

## 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:

1. Acquire a waveform as a Time Series.
2. Calculate the FFT of the waveform (that is, generate the Spectrum).
3. 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 it grow.

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

### Hints

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

# 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.

## Implementation

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

   - ☐ 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

If an additional parameter needs to be added to the Result string later, and array of parameter values makes it easy to add the new name.

## 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 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.

Hints:

- Create a subVI which formats each name-value pair. Separate the name and value using a Tab constant and terminate with the End of Line constant.
- Then process a 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.

## End of Exercise 5-1