ESSENTIAL PROGRAMMING STRUCTURES IN LABVIEW*

Ed Doering

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 2.0^{\dagger}

Abstract

Learn how to work with LabVIEW's essential programming structures such as for-loops, while-loops, case structure, MathScript node, and diagram disable.

1 Overview

Signal processing applications developed in LabVIEW make frequent use of basic constructs that are common to all high-level programming languages: for-loops, while-loops, and case structures. A **For Loop** repeats a block of code a fixed number of times, a **While Loop** repeats a block of code as long as a particular condition is true, and a **Case Structure** executes one of several blocks of code depending on some selection criterion. After completing this module you will be able to use these three essential structures in your own LabVIEW VIs.

You will also learn about two additional structures. The **MathScript Node** provides a way for you to develop a customized node whose behavior is defined using the MathScript text-based programming language. MathScript syntax and functions are quite similar to MATLAB, so the MathScript node can help you to leverage any MATLAB programming experience you may have. Lastly, the **Diagram Disable** structure is useful when you need to temporarily "comment out" a portion of your LabVIEW code.

The following diagram highlights the structures on the "Programming | Structures" palette about which you will learn in this module:

^{*}Version 1.6: Jan 5, 2010 4:57 pm -0600

 $^{^{\}rm +}{\rm http://creative commons.org/licenses/by/2.0/}$



Figure 1: Structures on the "Programming | Structures" palette described in this module

2 For-Loop Structure

2.1 Basic concepts

The **For Loop** structure provides a means to repeatedly run a block of code (or **subdiagram**) for a fixed number of times. The following screencast video introduces the **For Loop** structure, including concepts such as **loop count terminal**, **iterator**, **loop tunnel**, **indexing**, **shift register**, and the **feedback node**.

Image not finished

Figure 2: [video] LabVIEW Techniques: For-Loop structure

2.2 Working with arrays as inputs

The **For Loop** structure works efficiently with arrays. The next screencast video describes how to work with arrays that serve as inputs to the for-loop structure. Array elements can be used individually or collectively within the for-loop, depending on whether you have enabled **indexing** on the **loop tunnel**. Indexing on the output side of the for-loop can also be used to store a value (either a scalar or an array) on each iteration of the for-loop, thereby producing an array as output. Arrays can also be **concatenated**, a standard technique for constructing an audio signal from individual segments.

Image not finished

Figure 3: [video] LabVIEW Techniques: For-Loop structure with arrays as input

3 While-Loop Structure

The While Loop structure is similar to the For Loop structure with its ability to repeatedly run a **subdiagram**, but the number of times is not fixed in advance. Instead, the while-loop structure will execute its subdiagram as long as a particular condition is true. The following screencast will show you how to use the While Loop structure.

Image not finished

Figure 4: [video] LabVIEW Techniques: While-Loop structure

4 Case Structure

The **Case Structure** provides a mechanism by which exactly one of several possible subdiagrams will be executed, depending on the value connected to the **selector terminal**. When the selector terminal is a Boolean type (either True or False), the case structure implements the "if-else" construct of text-based languages. When the selector terminal is an integer type, the case structure implements the "case" or "switch" construct of text-based languages.

The following screencast introduces you to the **Case Structure**. You will learn how to add and delete subdiagrams, how to choose the default subdiagram, and how to ensure that valid outputs are generated for all possible cases. The **Boolean** and **integer** data types are covered in this screencast; the next screencast describes how to work with the **string** and **enumerated** data types, which provide a user-friendly way to select cases from the front panel.

Image not finished

Figure 5: [video] LabVIEW Techniques: Case Structure with "Boolean" and "integer" data types at the selector terminal

Image not finished

Figure 6: [video] LabVIEW Techniques: Case Structure with "string" and "enumerated" data types at the selector terminal

5 MathScript Node

The **MathScript Node** offers a convenient way to implement a programming concept that may be otherwise difficult to implement using standard G code (i.e., creating LabVIEW block diagrams by wiring available structures and nodes). **MathScript** is a text-based programming language that uses syntax very similar to **MATLAB**. If you have prior experience with MATLAB, you can easily develop and debug a MathScript-based script in the **MathScript interactive window**, then copy the text into a MathScript node on your block diagram. After you create input and output terminals on the MathScript node, it can be connected to the rest of the block diagram as you would any other structure.

The following screencast video introduces you to the **MathScript Node**. The example walks you through the process to create a specialized node that accepts a scalar N as input and produces a specialized array as output.

Image not finished

Figure 7: [video] LabVIEW Techniques: MathScript node and MathScript Interactive Window

```
% Create a triangle array x = [linspace(0,1,N/2) \ linspace(1,0,N/2)]
The screencast in this section (just above) walks through creation of a LabVIEW VI that includes a Math
```



Figure 8: The screencast in this section (just above) describes how to build a VI that includes a MathScript node.

6 Diagram-Disable Structure

You probably have used the software debugging technique known as "commenting out" a block of code; you do this so that you can effectively remove a portion of code without actually deleting it from the file. Watch the next screencast video to learn how to use the **Diagram Disable** structure to "comment out" a portion of your LabVIEW block diagram.

Image not finished

Figure 9: [video] LabVIEW Techniques: Use Diagram-Disable structure to "comment out" a section of code