Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing

By:
Ed Doering

Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing

By:
Ed Doering

Online:
< http://cnx.org/content/col10440/1.1/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: July 18, 2007
PDF generated: February 3, 2011
For copyright and attribution information for the modules contained in this collection, see p. 32.

Table of Contents

Chapter 1

Programming Tutorials

Chapter 2

Getting Started with LabVIEW[1]

2.1 Overview

Welcome to the exciting world of **LabVIEW** for audio and signal processing applications! This module contains five **screencast** videos, meaning that the videos were captured directly from my computer screen. As I operate the LabVIEW software, I explain each step and discuss what is going on. After you watch each of the videos, you will possess a good idea of some fundamental LabVIEW concepts, including:

- Front panel and block diagram programming paradigm
- Dataflow paradigm
- Data types
- Broken wires
- Debugging techniques

2.2 A Bit of History

In 1986 National Instruments Corporation released the first version of LabVIEW (**Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench), which was designed to help engineers use a computer (the Apple Macintosh) to control and gather data from electronic instrumentation (voltmeters, oscilloscopes, and the like) all interconnected by the standard General Purpose Instrumentation Bus, or GP-IB. From its inception, LabVIEW programming was **graphical** in nature. Instead of writing a text file and compiling it to an executable, you connect various elements such as **controls**, **indicators**, **nodes** and **subVIs** together with **wires**, and in this way create a **blockdiagram**. The controls and indicators reside on the **front panel**, which looks just a traditional electronic instrument, i.e., it can have knobs, sliders, buttons, and display panels. The complete application is called a **virtual instrument**, or **VI** for short.

LabVIEW has since evolved into a complete programming environment; anything that you can imagine can probably be implemented in LabVIEW. Recent versions of LabVIEW have added a full suite of tools for doing signal processing, and since soundcard operations are provided, it becomes natural to develop audio signal processing applications in LabVIEW.

LabVIEW's interactive front panel offers a unique opportunity to explore signal processing concepts in real time. As you work your way through other modules in this series, you will learn how to implement your own applications whose user interface consists of knobs, sliders and switches that can adjust processing parameters **while you listen** to the results.

[1] This content is available online at <http://cnx.org/content/m14764/1.4/>.

2.3 Your First Virtual Instrument (VI)

Watch the following screencast video to learn how to connect front panel **controls** and **indicators** together. You will also learn how to use the **While Loop** structure to make your VI operate continuously until you press a "STOP" button on the front panel.

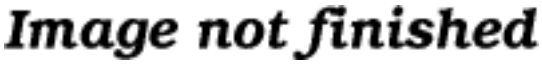

Figure 2.1: [video] Creating your first "VI" (Virtual Instrument)

2.4 The Dataflow Concept

The notion of LabVIEW's **dataflow** programming paradigm must be grasped **immediately** in order for you to make forward progress learning about LabVIEW to create your own applications. Dataflow programming means that valid data must be present at **all** of the input **terminals** on a node (or subVI) before that node (or subVI) will produce valid data on its output terminals. Moreover, the node (or subVI) does **not** continually process its inputs data unless it is embedded in some sort of looping structure.

Click on the following animation of the dataflow concept to watch a screencast video that explains and further explores the dataflow programming concept.

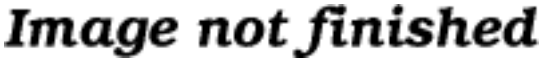

Figure 2.2: [video] Understanding the LabVIEW "dataflow" paradigm

2.5 Data Types

LabVIEW supports a broad range of **data types**, including **numeric**, **Boolean**, and **string**. The following screencast video will acquaint you with the floating point and integer styles of numeric data type, as well as the Boolean data type. The screencast explains the significance of the **coercion indicator** – the red dot that flags a mismatch on data types applied to the input of a node or subVI – as well as **data type conversion** nodes that you can use to intentionally convert a value from one data type to another.

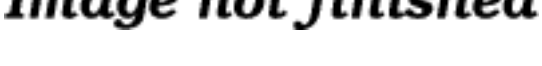

Figure 2.3: [video] Datatypes: Numeric and Boolean

2.6 Broken Wires

Broken wires indicate an error that must be corrected before your VI will run. Broken wires result from a number of causes, and it is important to understand why the wire is broken and how to correct the situation. The following screencast describes broken wires in detail.

Image not finished

Figure 2.4: [video] Understanding and correcting broken wires

---

2.7 Debugging Techniques

As you begin learning LabVIEW so that you can develop your own VIs, you will find the debugging techniques described by the next screencast video helpful. Topics include adding additional indicators, using the **Highlight Execution** feature, using the **Retain Wire Values** feature, **single-stepping**, viewing wire values using **probes**, and creating **breakpoints** to pause execution when new data is available on a wire.

Image not finished

Figure 2.5: [video] Basic debugging techniques

2.8 For Further Study

If you are new to LabVIEW, I recommend the excellent text by Robert H. Bishop, *Learning with LabVIEW 8 (Pearson Prentice Hall, 2007, ISBN 0-13-239025-6)*. With this text you can learn basic LabVIEW programming techniques and get a better idea of everything that LabVIEW has to offer.

Once you have developed some skill with LabVIEW, consider Peter A. Blume's text, *The LabVIEW Style Book (Prentice Hall 2007, ISBN 0-13-145835-3)*. This text covers a wide variety of techniques that will help you to develop robust and well-designed LabVIEW applications.

Don't forget to check out the on-line documentation that is part of your LabVIEW product installation.

Visit the National Instruments[2] website, including their Academic[3] page and the NI Developer Zone[4] .

[2] http://www.ni.com/
[3] http://www.ni.com/academic/
[4] http://zone.ni.com/devzone/cda/main

Chapter 3

Editing Tips for LabVIEW[1]

3.1 Overview

LabVIEW offers an extensive range of techniques to create and edit block diagrams and front panels. This screencast video reviews some of the commonly-used editing techniques, including:

- adding comments and documentation with **text labels**
- placing front panel terminals as **icons**
- **replacing** front panel controls and indicators with alternate forms
- creating **controls**, **constants**, and **indicators** directly at the **terminal** (using context menu)
- cleaning up wire routing (**Clean Up Wire** context menu)
- **moving** objects using **arrow keys** (fine positioning) and Shift+arrow keys (larger steps)
- **aligning** multiple objects
- **replacing** nodes and subVIs with similar forms
- **removing** all **broken wires** (**Ctrl+B**)
- **distributing** objects
- converting front panel **controls** into **constants**
- obtaining help: **Context Help** and detailed **help page**
- adding **decorations** (arrows, lines, text labels)
- viewing the **Navigation Window**
- **replicating** existing structures

Figure 3.1: [video] Editing tips for front panels and block diagrams

[1] This content is available online at <http://cnx.org/content/m14765/1.3/>.

Chapter 4

Essential Programming Structures in LabVIEW[1]

4.1 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:

[1] This content is available online at <http://cnx.org/content/m14766/1.6/>.

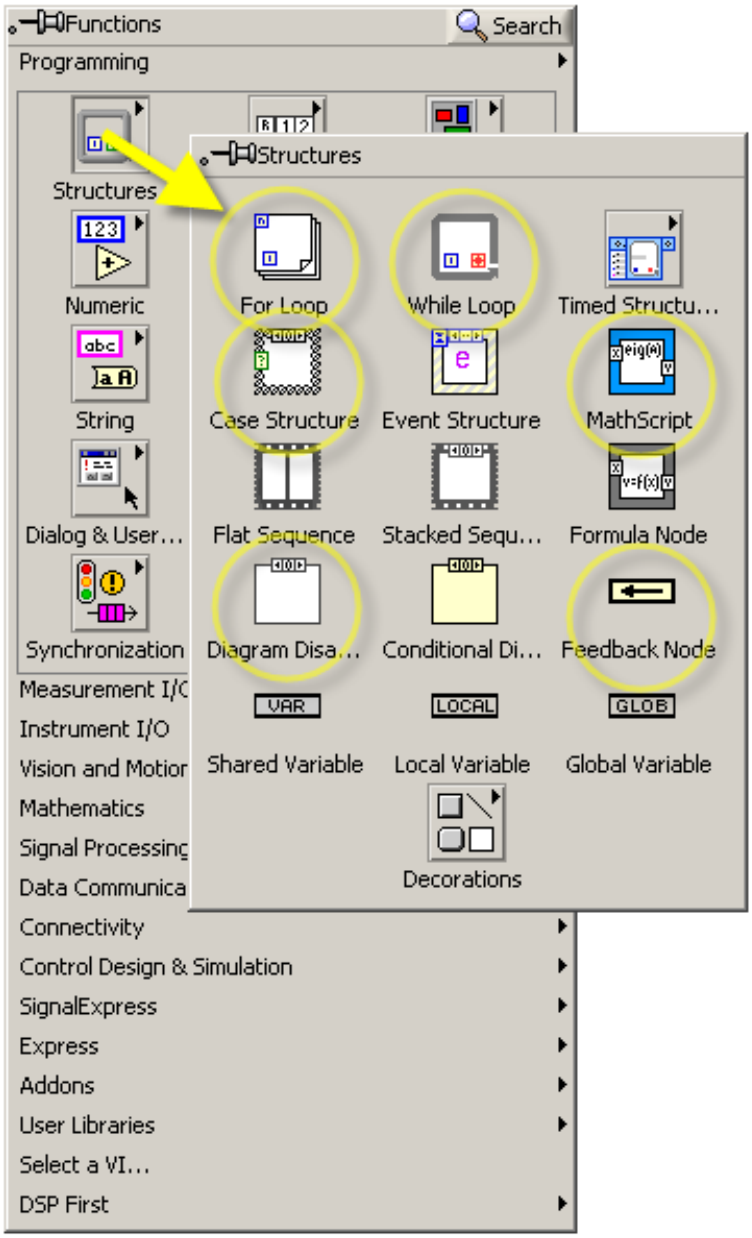

Figure 4.1: Structures on the "Programming | Structures" palette described in this module

4.2 For-Loop Structure

4.2.1 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 4.2: [video] LabVIEW Techniques: For-Loop structure

4.2.2 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 4.3: [video] LabVIEW Techniques: For-Loop structure with arrays as input

---

4.3 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 4.4: [video] LabVIEW Techniques: While-Loop structure

4.4 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.

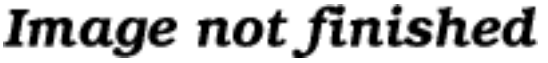

Figure 4.5: [video] LabVIEW Techniques: Case Structure with "Boolean" and "integer" data types at the selector terminal

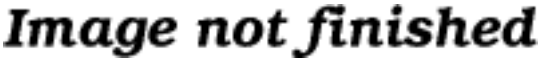

Figure 4.6: [video] LabVIEW Techniques: Case Structure with "string" and "enumerated" data types at the selector terminal

4.5 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 4.7: [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 MathS
```

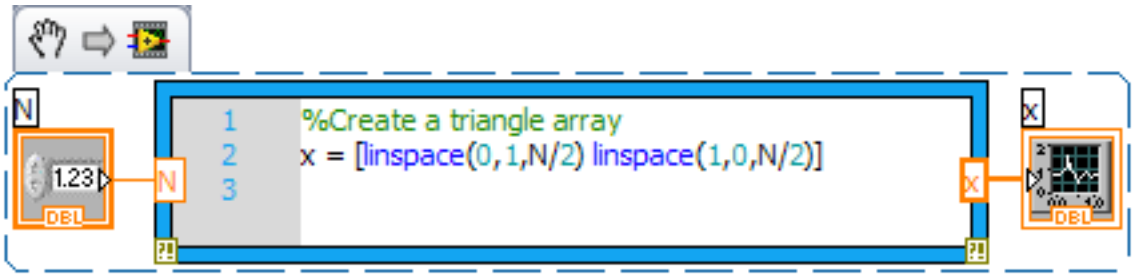

Figure 4.8: The screencast in this section (just above) describes how to build a VI that includes a MathScript node.

4.6 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 4.9: [video] LabVIEW Techniques: Use Diagram-Disable structure to "comment out" a section of code

Chapter 5

Create a SubVI in LabVIEW[1]

A **subVI** is the LabVIEW equivalent to "functions," "subroutines," and "methods" in other programming languages. You can easily create a subVI that can be used reused many times in other VIs; editing the subVI one time updates its behavior for all VIs that use the subVI, which simplifies program design and maintenance. Moreover, the subVI forms the basic method by which you can create a hierarchical program.

The screencast video below details the necessary steps to convert a conventional block diagram into a subVI by adding a **connector pane** and an **icon**. These two elements enable the block diagram to appear as an element on another block diagram with a connection pattern specific to the requirements of the task performed by the subVI. In addition, you can set the **VI Description** (part of the **VI Properties**) of your subVI so that hovering your cursor over your subVI reveals a helpful description of its behavior.

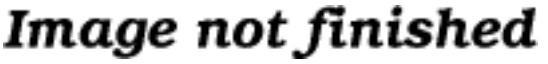

Figure 5.1: [video] LabVIEW Techniques: Create a subVI from a block diagram

[1] This content is available online at <http://cnx.org/content/m14767/1.4/>.

Chapter 6

Arrays in LabVIEW[1]

6.1 Overview

Arrays are a fundamental data type for signal processing. LabVIEW offers a complete set of techniques to create and manipulate arrays, and to perform mathematical and signal processing operations on arrays. This module will acquaint you with the basic techniques for working with arrays.

6.2 Creating Arrays

The following video screencast describes how to create arrays on the front panel and on the block diagram. The elements of an array can be edited and modified directly, a suitable technique when the array is relatively small.

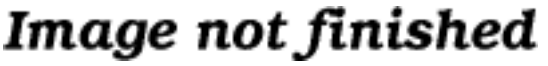

Figure 6.1: [video] Creating arrays

6.3 Manipulating Arrays

The next video screencast explains essential array manipulations for signal processing tasks, including: determining the **dimensions** of an array, retrieving individual elements, rows, columns, or other **subarrays**, appending (concatenating) arrays, and **reshaping** arrays.

[1] This content is available online at <http://cnx.org/content/m14768/1.4/>.

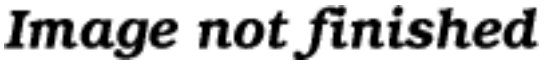

Figure 6.2: [video] Manipulating arrays

6.4 Mathematical Operations with Arrays

Signal processing operations commonly operate on all of the array elements at once. For example, adding a scalar constant to an array implies a loop operation in which the constant is added to each element of the array.

The following screencast video describes several techniques for performing mathematical operations on arrays. Important side effects (such as what happens when two arrays of different lengths are added together) are discussed. The **Ramp Pattern** subVI is also described as a method to create a time basis for mathematical functions such as the **exponential**.

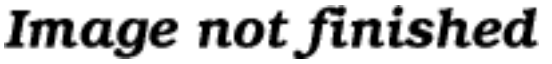

Figure 6.3: [video] Performing mathematical operations with arrays

6.5 Arrays and Spreadsheets

Spreadsheets are often used as a data storage mechanism by other applications. The next screencast video shows you how to retrieve an array from a comma-separated-values (CSV-format) spreadsheet, and how to create your own CSV-format spreadsheet from an existing array within LabVIEW.

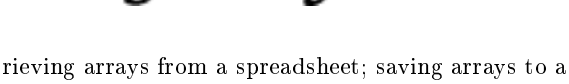

Figure 6.4: [video] Retrieving arrays from a spreadsheet; saving arrays to a spreadsheet

Chapter 7

Audio Input and Output

Chapter 8

Audio Output Using LabVIEW's "Play Waveform" Express VI[1]

8.1 Overview

A 1-D array representing an audio signal can be played directly by the computer's soundcard using the **Play Waveform** Express VI. The screencast video shows how to listen to an audio signal source (a sinusoid in this example). The 1-D array must first be converted to the **waveform** data type using the **Build Waveform** node.

Figure 8.1: [video] Listen to an audio signal (1-D array) on the soundcard

[1] This content is available online at <http://cnx.org/content/m14769/1.3/>.

Chapter 9

Audio Sources in LabVIEW[1]

9.1 Overview

The **Sine Wave** subVI in the Signal Processing palette is a useful way to produce a sinusoidal audio signal. The screenscast video will acquaint you with the Sine Wave subVI, especially its requirement for **normalized frequency**. This subVI is one of several **re-entrant** signal generators, meaning that they have the ability to "pick up where they left off;" that is, they can maintain continuity in their output across execution runs and loop iterations.

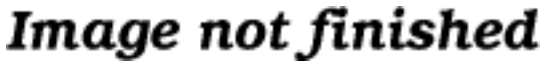

Figure 9.1: [video] Sinewave source for audio applications

---

[1] This content is available online at <http://cnx.org/content/m14770/1.3/>.

Chapter 10

Reading and Writing Audio Files in LabVIEW[1]

10.1 Overview

LabVIEW offers a variety of ways to read and write audio files in WAV format. After completing this module you will be able to use the **Simple Read** and **Simple Write** subVIs located in the "Programming | Graphics & Sound | Sound | Files" palette to retrieve an audio signal as a 1-D array from a .wav file, and also to save a 1-D array that represents an audio signal to a .wav file. Additional points covered include scaling your audio signal to have a maximum absolute value of one before saving as an audio file, and how to create a two-channel (stereo) audio file.

10.2 Retrieve an Audio Signal from a .wav File

The **Sound File Simple Read** subVI accepts a filename for an audio file in .wav format and returns a waveform data type. You can read mono or stereo files, and also determine information such as the audio signal's sampling frequency and its total number of samples.

The following video screencast shows how to use **Simple Read** to retrieve the audio signal as an array data type that can be used as a signal input for your own VIs. You may want to start LabVIEW now, then follow along to create your own version of the VI pictured below. If so, you will need the two audio files referenced in the screencast: tone-noise.wav[2] and left-right.wav[3].

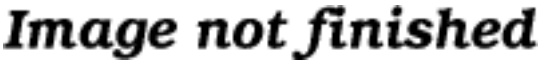

Figure 10.1: [video] LabVIEW Techniques: Retrieve an audio signal from a .wav file

[1] This content is available online at <http://cnx.org/content/m14771/1.6/>.
[2] See the file at <http://cnx.org/content/m14771/latest/tone-noise.wav>
[3] See the file at <http://cnx.org/content/m14771/latest/left-right.wav>

10.3 Save an Audio Signal to a .wav File

The **Sound File Simple Write** subVI accepts a signal in waveform data type and a filename and stores the signal as a .wav file. You can set the number of bits per sample (16 is recommended for best fidelity). You must ensure that your signal values lie in the range -1 to +1, otherwise other applications may not be able to read your .wav file properly.

The next video screencast shows how to use **Simple Write** to save an existing 1-D array that represents an audio signal to a .wav file. The screencast also covers some not-so-obvious data-type transformations required to successfully create your .wav file. You may want to start LabVIEW now, then follow along to create your own version of the VI pictured below. If so, you will need the audio file referenced in the screencast: tone-noise.wav[4].

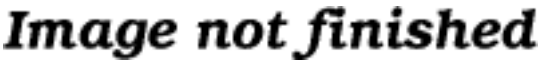

Figure 10.2: [video] LabVIEW Techniques: Save an audio signal to a .wav file

10.4 Scale Your Audio Before Saving to a .wav File

As mentioned earlier, you must ensure that all of the samples in your signal lie in the range -1 to +1. It is easy to create signals that exceed this range, especially when you add multiple signals together.

The next screencast illustrates how **Quick Scale** is an easy way to scale your 1-D array such that the maximum absolute value is always one; **Quick Scale** is located in the "Signal Processing | Sig Operation" palette, and should be used as the last step before converting to the waveform data type.

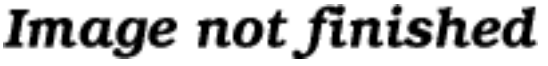

Figure 10.3: [video] LabVIEW Techniques: Scale audio signal to +/- 1 range before saving to a .wav file

10.5 Create a Two-Channel (Stereo) .wav File

Creating a stereo .wav file requires that you assemble an array of waveforms, one for the left channel and the other for the right channel. Watch the next screencast video to learn how.

[4]See the file at <http://cnx.org/content/m14771/latest/tone-noise.wav>

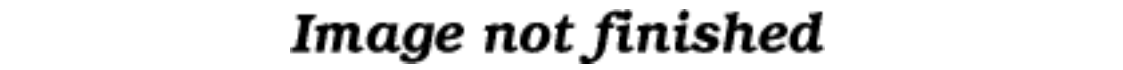

Figure 10.4: [video] LabVIEW Techniques: Save a stereo (two-channel) audio signal to a .wav file

Chapter 11

Real-Time Audio Output in LabVIEW[1]

11.1 Overview

The interactive front panel is a hallmark of the LabVIEW programming paradigm. In this screencast video you will learn how to use the low-level **Sound Output** subVIs called **Configure**, **Write**, and **Clear** to build a general-purpose framework that can continually produce audio output, all the while responding to parameter changes on the front panel in real time. The example uses a sinusoidal source, but you can easily adapt the VI for your own signal sources; the finished VI from this example is available: lvt_audio_realtime-out.vi[2] (right-click and choose "Save As")

Figure 11.1: [video] Real-time audio output with interactive controls

[1] This content is available online at <http://cnx.org/content/m14772/1.5/>.
[2] http://cnx.org/content/m14772/latest/lvt_audio_realtime-out.vi

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing*
Edited by: Ed Doering
URL: http://cnx.org/content/col10440/1.1/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Getting Started with LabVIEW"
By: Ed Doering
URL: http://cnx.org/content/m14764/1.4/
Pages: 3-5
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Editing Tips for LabVIEW"
By: Ed Doering
URL: http://cnx.org/content/m14765/1.3/
Page: 7
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Essential Programming Structures in LabVIEW"
By: Ed Doering
URL: http://cnx.org/content/m14766/1.6/
Pages: 9-13
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Create a SubVI in LabVIEW"
By: Ed Doering
URL: http://cnx.org/content/m14767/1.4/
Page: 15
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Arrays in LabVIEW"
By: Ed Doering
URL: http://cnx.org/content/m14768/1.4/
Pages: 17-18
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Module: "Audio Output Using LabVIEW's "Play Waveform" Express VI"
By: Ed Doering
URL: http://cnx.org/content/m14769/1.3/
Page: 21
Copyright: Ed Doering
License: http://creativecommons.org/licenses/by/2.0/

Musical Signal Processing with LabVIEW – Programming Techniques for Audio Signal Processing

After completing this multi-media course you will be well-equipped to start creating your own audio and signal processing applications within the LabVIEW development environment. The modules in this course make extensive use of "screencasts" – videos captured directly from the computer screen that show the LabVIEW tool in operation, with audio narration to explain each step. The course includes a "Getting Started" tutorial, editing tips, essential programming structures, subVIs, arrays, audio sources, audio output to the soundcard, reading and writing audio files, and real-time audio output with interactive parameter control. This course is part of the series "Musical Signal Processing with LabVIEW".

About Connexions

Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.