## **Audio System Toolbox™** User's Guide

# MATLAB&SIMULINK®



**R**2016a

## How to Contact MathWorks



The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098

#### Audio System Toolbox™ User's Guide

© COPYRIGHT 2016 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

#### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

#### Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

#### **Revision History**

March 2016 Online only

New for Version 1.0 (Release 2016a)





#### **Musical Instrument Digital Interface**

Musical Instrument Digital Interface (MIDI)	1-2
About MIDI	1-2
MIDI Control Surfaces	1-2
Using MIDI Control Surfaces with MATLAB and Simulink	1-3

#### Use the Audio Test Bench

Jse the Audio Test Bench
Open Audio Test Bench
Run Audio Test Bench
Debug Source Code of Audio Plugin
Open Scopes
Configure Input to Audio Test Bench
Configure Output from Audio Test Bench
Synchronize Plugin Property with MIDI Control
Play the Audio and Save the Output File

## Audio Plugin Example Gallery

## 3

1

2

Audio Plugin Example Gallery	3-2
Audio Plugin Examples	3-2
Sample Audio Files	3-13

## **Musical Instrument Digital Interface**

## **Musical Instrument Digital Interface (MIDI)**

#### In this section ...

"About MIDI" on page 1-2

"MIDI Control Surfaces" on page 1-2

"Using MIDI Control Surfaces with MATLAB and Simulink" on page 1-3

#### About MIDI

The Musical Instrument Digital Interface (MIDI) was originally developed to interconnect electronic musical instruments. This interface is very flexible and has many uses in many applications far beyond musical instruments. Its simple unidirectional messaging protocol supports many different kinds of messaging.

Windows, Macintosh, and Linux platforms all have native support for MIDI, so software on any of these platforms can send and receive MIDI messages. See http://www.midi.org for more information about MIDI.

## **MIDI Control Surfaces**

One kind of MIDI message is the Control Change message, used to communicate changes in controls, such as knobs, sliders, and buttons. A MIDI Control Surface is a device with controls that sends MIDI Control Change messages when you turn a knob, move a slider, or push a button on a MIDI control surface. This Control Change message indicates which control changed and what its new position is. MIDI control surfaces are quite generic because the interpretation of the Control Change message is entirely up to the message recipient. Even though some control surfaces are tailored for particular applications, the messages they send can be used to control anything.

Hardware MIDI control surfaces are widely available in a range of configurations and prices. MIDI control apps can turn a smartphone or tablet into a virtual MIDI control surface. For custom applications, MIDI control surfaces are not difficult to build using, for example, Arduino boards.

Because the MIDI messaging protocol is unidirectional, determining a particular control's position requires that the receiver listen for Control Change messages that control sends. The protocol does not support querying the control for its position.

The simplest MIDI control surfaces are unidirectional; they end MIDI Control Change messages, but do not receive them. More sophisticated control surfaces are bidirectional: They can both send and receive Control Change messages. These control surfaces have knobs or sliders that can be operated automatically. For example, a control surface can have sliders or knobs that are motorized. When it receives a Control Change message, the appropriate control is moved to the position in the message. You can use this feature to synchronize software GUI with MIDI control surface. For example, moving a slider on the MIDI control surface sends a Control Change message to a GUI slider, which then moves to match the control surface. Similarly, moving the GUI slider sends a Control Change message to the MIDI control surface, which then moves to match the GUI slider.

## Using MIDI Control Surfaces with MATLAB and Simulink

The Audio System Toolbox<sup>™</sup> product enables you to use MIDI control surfaces to control MATLAB<sup>®</sup> programs and Simulink<sup>®</sup> models by providing the capability to listen to Control Change messages. The toolbox also provides a limited capability to send Control Change messages to support synchronizing MIDI controls. The Audio System Toolbox interface to MIDI control surfaces includes five functions and one block:

- midiid function
- midicontrols function
- midiread function
- midisync function
- midicallback function
- MIDI Controls block

#### **Initial Setup**

Your MIDI control surface should be connected to your computer, and turned on, before starting MATLAB. Instructions for connecting your MIDI device to your computer vary from device to device. See the instructions that came with your particular device. If you start MATLAB before connecting your device, MATLAB may not recognize your device when you connect it. To correct the problem, restart MATLAB with the device already connected.

Next, set the MATLAB preference, specifying the name of the default MIDI device. Use midiid to determine the name of the device, and then use setpref to set the preference:

If you do not set this preference, MATLAB and the host operating system choose a device for you. However, such autoselection can cause unpredictable results because many computers have "virtual" (software) MIDI devices installed that you may not be aware of. For predictable behavior, you should set the preference.

You can always override this default and explicitly specify a device name. Thus, you can use multiple MIDI devices simultaneously.

#### **Identifying Controls**

Before you can connect a MIDI control with MATLAB or Simulink, you must know the identifiers for that particular control:

- Control number
- Device name

The control number is a fixed integer assigned by the device manufacturer. Some devices may change the assigned number based on various modes, or you can reprogram the number. The device name is determined by the manufacturer and the host operating system. You use midiid to determine both.

You do not usually have to use midiid repeatedly. If you use a single device in most cases, then specify that device as the default hardware. You can save the control numbers in a function, a .mat file, or whatever form you find convenient. This example shows a function returning a struct with all the control numbers for a Behringer BCF2000:

```
function ctls = BCF2000
% BCF2000 return MIDI control number assignments
% for Behringer BCF2000 MIDI control surface
```

```
ctls.knobs = 1001:1008;
ctls.buttons = [1065:1072;1073:1080];
ctls.sliders = 1081:1088;
end
```

#### **MATLAB** Interface

To use the MATLAB interface functions, first call midicontrols to specify any devices or controls to listen to. midicontrols returns an object, which you pass to the other functions for subsequent operations. You can now read the values of the specified MIDI controls by calling midiread with that object. MATLAB can respond to changes in MIDI controls by periodically calling midiread.

You can also set a callback on the specified MIDI controls by calling midicallback with that object and a function handle. The next time the MIDI controls change value, the function handle is invoked and passed to the object. The callback function typically calls midiread to determine the new value of the MIDI controls. You can use this callback when you want a MIDI control to trigger an action (such as update a GUI). Using this approach prevents having a continuously running MATLAB program in the command window.

#### Synchronization

If midiread is called before the MIDI control sends a Control Change message, the midicontrols object has no information about the actual state of the MIDI control. During this time, the midicontrols object and the actual MIDI control are out of sync with each other. Thus, calling midiread returns the initial value that was specified in the call to midicontrols (0 by default). You can synchronize the object with the control by moving the MIDI control. The MIDI control responds by sending a Control Change message causing the midicontrols object to sync to the MIDI control. If your MIDI control surface is bidirectional, you can sync in the other direction by calling midisync to send the midicontrols object's initial value to the actual MIDI control. The MIDI control responds by moving into sync with the midicontrols object.

It is generally harmless to call midisync even if the MIDI control surface is not bidirectional, so it is usually good practice to call midisync immediately after calling midicontrols.

Synchronization is also useful to link a MIDI control with a GUI control (a uicontrol slider, for example), so that when one control is changed, the other control tracks it. Typically, you implement such tracking by setting callback functions on both the MIDI control (using midicallback) and the GUI control. The MIDI control callback sends its

new value to the GUI control and the GUI control sends its value to the MIDI control, using midisync.

#### Simulink Interface

The MIDI Controls block provides the Simulink interface. See the block reference page MIDI Controls for more details.

## Use the Audio Test Bench

## Use the Audio Test Bench

In this tutorial, you open the Audio Test Bench and explore some of its key functionality.

## **Open Audio Test Bench**

To open an audio plugin test bench user interface (UI) for an instance of DampedVolumeController, at the MATLAB command prompt, enter:

audioTestBench audiopluginexample.DampedVolumeController

承 Audio Test Bench: Dam		
	L 🏟 🕐	لا
INPUT: Audio Fi	le Reader 🔹 🔯	OUTPUT: Audio Device Writer 💌 💽
Gain (dB)	4	• 0
Transition Delay (s)	4	• 0.01

## **Run Audio Test Bench**

To run the audio test bench for your plugin with default settings, click (). Move the sliders to modify the **Gain (dB)** and **Transition Delay (s)** parameters while streaming.

To stop the audio stream loop, click •. The MATLAB command line and objects used by the test bench are now released.

To reset internal states of your audio plugin and return the sliders to their initial positions, click  $\mathbb{G}$ .

Click 🕑 to run the audio test bench again.

## Debug Source Code of Audio Plugin

To pause the audio test bench, click.

To open the source file of your audio plugin, click  $\blacksquare$ .

	<u> </u>	\Bdsp\2015_12_08_h10m37s21_job313271_pass\matlab\toolbox\audio\audiodemos\+audioplugine
	EDITOR	PUBLISH VIEW
FILE	NAVIGATE	EDIT Insert Insert Property  Method  Advance Analyze Block  Method  Advance Advance Run and Advance Run and Time
•	•	SYSTEM OBJECT BREAKPOINTS RUN
1	- Cla	assdef DampedVolumeController < matlab.System & audioPlugin
2		DampedVolumeController Control the volume of an audio signal
3	8	This example shows an audio plugin for volume control. The plugin has
4	8	two parameters, the gain (dB) that is to be applied to the input audio
5	8	signal and the delay in seconds with which the gain is applied.
6	용	
7	8	In order to avoid abrupt signal changes that result in audible
8	8	artifacts, the gain is always changed gradually from its existing value
9	8	to its new value when the parameter is changed. TransitionDelay
10	용	controls the transition time between the old gain and the new gain.
11	용	
12	- %	This is an example of an audio plugin that is also a System object.
13		
14	8	Copyright 2015 The MathWorks, Inc.
15	8#0	codegen
16		
17	Ē	properties
18		%Gain Gain in decibels to be applied to input audio
19		Gain = 0
20		
21		%TransitionDelay TransitionDelay parameter
22		% TransitionDelay specifies transition time between the previous
23		% gain parameter and the new gain parameter.
24		TransitionDelay = 0.01; % 10 ms
25	Γ	end 🔽
		Ln 1 Col 1 ":

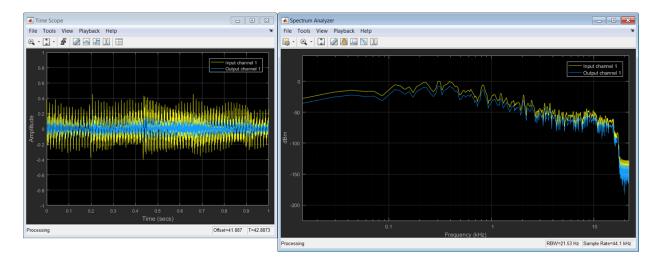
You can inspect the source code of your audio plugin, set breakpoints on it, and modify the code. Set a breakpoint at line 69, and then click  $\bigcirc$  on your audio test bench.

07	
68	% Set value if it meets required attributes
69 🔴	<pre>plugin.TransitionDelay = val;</pre>
70 -	- end
71	- end
72	

The audio test bench runs your plugin until it reaches the breakpoint. To reach the breakpoint, move the **Transition Delay (s)** slider on your audio test bench. To quit debugging, remove the breakpoint. In the MATLAB editor, click **Quit Debugging**.

#### **Open Scopes**

To open the Time Scope to visualize the time-domain input and output for your audio plugin, click  $\square$ . To open the spectrum analyzer to visualize the frequency-domain input and output, click  $\square$ .

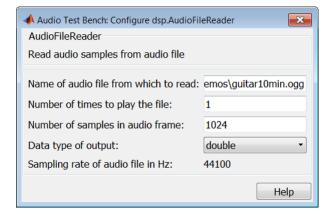


## Configure Input to Audio Test Bench

To release objects and stop the audio stream loop, click •. The input to the audio test bench uses the functionality of audioDeviceReader and dsp.AudioFileReader. You can input from device or file by selecting from the **Input** menu. Select Audio File Reader.

INPUT:	Audio File Reader	<b>-</b>	OUTPUT: Audio Device Writer	- Ø
	Audio File Reader			
	Audio Device Reader			

Click 🖾 to open a UI for Audio File Reader configuration.



You can enter any file name included on the MATLAB path. If you want to specify a file that is not on that MATLAB path, specify its path completely.

In the Name of audio file from which to read box, enter: RockDrums-44p1stereo-11secs.mp3

Press **Enter** on your keyboard, and then exit the Audio File Reader configuration UI. To run the audio test bench with your new input, click  $\odot$ .

See audioDeviceReader and dsp.AudioFileReader for information about modifying parameters of your audio reader.

#### Configure Output from Audio Test Bench

To release your output object and stop the audio stream loop, click •. The output from the audio test bench uses the functionality of audioDeviceWriter and dsp.AudioFileWriter. Choose to output to device and file by selecting Both from the **Output** menu.



To open a UI for Audio Device Writer and Audio File Writer configuration, click

📣 Audio Test Bench: Configure audioDeviceWriter	×
audioDeviceWriter	
Play audio data using the computer's audio device	
Number of samples per second sent to audio device:	44100
Data type used by the device:	16-bit integer
Audio device driver:	DirectSound
Support variable sized input	
Device to which to send audio data:	Default
Source of output channel mapping:	Auto •
	Help

📣 Audio Test Bench: Configure dsp.AudioFileWriter		
	AudioFileWriter	
	Write audio samples to audio file	
	Name of audio file to which to write:	output.wav
	Format of created file:	WAV
	Sampling rate of audio data stream:	44100
	Algorithm used to compress audio data:	None (uncompressed) •
	Data type of uncompressed audio:	int16 🔹
		Help

See audioDeviceWriter and dsp.AudioFileWriter for information about modifying parameters of your audio reader.

## Synchronize Plugin Property with MIDI Control

If you have a MIDI device connected to your computer, you can synchronize plugin properties with MIDI controls. To open a MIDI configuration UI, click . Synchronize the Gain and TransitionDelay properties with MIDI controls you choose. Click **OK**.

See configureMIDI for more information.

## Play the Audio and Save the Output File

To run your audio plugin, click O. Adjust your plugin properties in real time using your synchronized MIDI controls and UI sliders. Your processed audio file is saved to the current folder.

## See Also

 $audio Plugin \ | \ Audio \ Test \ Bench \ | \ \texttt{generateAudioPlugin} \ | \ \texttt{validateAudioPlugin}$ 

## More About

- "Design an Audio Plugin"
- "Audio Plugin Example Gallery" on page 3-2
- "Export a MATLAB Plugin to a DAW"

## Audio Plugin Example Gallery

## Audio Plugin Example Gallery

Use these Audio System Toolbox plugin examples and audio files to analyze design patterns and practice your workflow.

## **Audio Plugin Examples**

Audio Test Bench: Bass			<mark>- X</mark>
INPUT: Audio Fi	le Reader 🔻 🚺	OUTPUT: Audio Dev	vice Writer 🔻 🔯
Bandpass Filter Upper Cutoff (Hz)	•		180
Gain	4		• 0.05

 $Name: {\tt audiopluginexample.BassEnhancer}$ 

**Type:** System object<sup>™</sup> plugin

**Description:** Implements a psychoacoustic bass enhancement algorithm. The plugin parameters are the upper cutoff frequency of the bandpass filter and the gain applied at the output of the bandpass filter.

Related Example: Psychoacoustic Bass Enhancement for Band-Limited Signals

**Inspect Code** 

edit audiopluginexample.BassEnhancer

#### Run Plugin

audioTestBench audiopluginexample.BassEnhancer				
Generate Plugin				
generateAudioPlugin audioplugin	example.BassEnhancer			
🕢 Audio Test Bench: Chorus				
INPUT: Audio File Reader	OUTPUT: Audio Device Writer			
Delay (sec)	• 0.03			
Tap 1 Depth of modulation	) 0.002			
Tap 1 Rate of modulation (Hz)	· · · · · · · · · · · · · · · · · · ·			
Tap 2 Depth of modulation	) 0.003			
Tap 2 Rate of modulation (Hz)	, 2			
Dry/wet mix (%)	, 50			
Name: audiopluginexample.Chorus				
Type: Basic plugin				
<b>Description:</b> Adds an audio chorus effect. The chorus effect is implemented by modulating two delay lines.				
Inspect Code				
edit audiopluginexample.Chorus				
Run Plugin				
audioTestBench audiopluginexample.Chorus				
Generate Plugin				
generateAudioPlugin audiopluginexample.Chorus				

	Audio Test Bench: Damped Volume Controller				
	INPUT: Audio File Reader • 🐼 OUTPUT: Audio Device Writer • 🐼				
	Gain (dB)				
	Transition Delay (s)				
Name: audionlu	uginexample.DampedVolumeController				
Type: System object plugin					
Type: System obj	ject plugin				
-	amps the volume control of an audio signal. The plugin has gain that is applied to the input audio signal, and the tran ion in seconds.				
Inspect Code					
edit audiopluginexample.DampedVolumeController					
Run Plugin					
<pre>audioTestBench audiopluginexample.DampedVolumeController</pre>					

#### **Generate Plugin**

generateAudioPlugin audiopluginexample.DampedVolumeController

🛋 Audio Test Bench 🕑 🕪 🔘 🖾 🔄					× •		
INPUT: Aud	dio File Reader	• ۞	ουτρυτ: 🗛	udio Device Writer	- 💿		
Delay tap 1 (	s) 4			•	0.2		
Gain for 1st t	ap			Þ	0.5		
Delay tap 2 (	s) 4			ŀ	0.3		
Gain for 2nd t	ap 🖣			Þ	0.5		
Dry/wet mix (	%) 4			Þ	50		
Name: audiopluginexample.Echo							
Type: Basic plugin							
<b>Description:</b> Implements a tunes the delay taps in seco							
Inspect Code							
edit audiopluginexample.Echo							
Run Plugin							
audioTestBench audiopluginexample.Echo							
Generate Plugin							
generateAudioPlugin audiopluginexample.Echo							

	Audio Test Bench: Flanger			- • ×
	INPUT: Audio File	e Reader 🔹 🔯	OUTPUT: Audio Device	Writer 💌 🔯
	Delay (s)			• 0.001
	Depth of flanging (%)	•		> 20
	Rate of modulation (Hz)	4		• 0.25
	Dry/wet mix (%)	4		• 50
Name: audioplu	uginexample	.Flanger		
Type: Basic plug	in			
<b>Description:</b> Im				

plugin uses audioOscillator to create the control signal for modulation. The plugin user tunes the delay tap in seconds, the amplitude and frequency of the delay line modulation, and the output dry/wet mix.

#### **Inspect Code**

edit audiopluginexample.Flanger

#### **Run Plugin**

audioTestBench audiopluginexample.Flanger

#### **Generate Plugin**

generateAudioPlugin audiopluginexample.Flanger

	Audio Test Bench: Highpass IIR Filter     S				
	INPUT: Audio File Reader 💌 💽 OUTPUT: Audio Dev	ice Writer 🔻 💽			
		, 20			
	Cutoff Frequency (Hz)	, 20			
Name: audioplu	uginexample.HighPassIIRFilter				
Type: System obj	ject plugin				
	plements a second-order IIR highpass filt ugin uses dsp.BiquadFilter to implement				
Related Exampl	le: Tunable Filtering and Visualization U	sing Audio Plug-Ins			
Inspect Code					
edit audioplugi	nexample.HighPassIIRFilter				
Run Plugin					
<pre>audioTestBench audiopluginexample.HighPassIIRFilter</pre>					
Generate Plugin					
generateAudioPl	.ugin audiopluginexample.HighPassIIR	Filter			

🔺 Audio Test Bench: Param 📀 🕪 🗐 🛱 🖃 🖸	-					
INPUT: Audio Fil		OUTPUT: Audio [	Device Writer 🔻 🔯			
Low Peak Gain (dB)	4		► 0			
Low Center Frequency (Hz)	4		▶ 100			
Low Q Factor	4		▶ 2			
Medium Peak Gain (dB)	4		• 0			
Medium Center Frequency (Hz)	4		▶ 1000			
Medium Q Factor	4		▶ 2			
High Peak Gain (dB)	4		<b>)</b> 0			
High Center Frequency (Hz)	4		► 10000			
High Q Factor	4		> 2			
Name: audiopluginexample.ParametricEqualizer						
Type: System object plugin						
<b>Description:</b> Implements a three-band parametric equalizer with tunable center frequencies, Q factors, and gains. The plugin uses designParamEQ to obtain filter coefficients and dsp.BiquadFilter to implement filtering.						
Related Example: Tunable Filtering and Visualization Using Audio Plug-Ins						
Inspect Code						
edit audiopluginexample.ParametricEqualizer						
Run Plugin						
audioTestBench audiopluginexample.ParametricEqualizer						
Generate Plugin						
generateAudioPlugin audiop	luginexample	.Parametric	cEqualizer			

INPUT:       Audio File Reader       OUTPUT:       Audio Device Writer       Image: Control of the state of	Audio Test Bench: Paran					• ×
Low Center Frequency (Hz) 4 9 100 Low Q Factor 4 9 2 Medum Peak Gain (dB) 4 9 0 Medum Q Factor 4 9 2 High Peak Gain (dB) 4 9 0	INPUT: Audio Fi	le Reader	- Q	OUTPUT: Au	dio Device Writer	• ۞
Low Q Factor 4 2 Medum Peak Gain (dB) 4 0 Medum Center Frequency (Hz) 4 1000 Medum Q Factor 4 2 High Peak Gain (dB) 4 0	Low Peak Gain (dB)	4			•	0
Medium Peak Gain (dB)     Image: Control of the control	Low Center Frequency (Hz)	4			•	100
Medum Center Frequency (Hz)         4         1000           Medum D Factor         4         2           High Peak Gain (dB)         4         9         0	Low Q Factor	4				2
Medum Q Factor	Medium Peak Gain (dB)	4				0
High Peak Gain (dB)	Medium Center Frequency (Hz)	4			•	1000
	Medium Q Factor	4				2
High Center Frequency (Hz)	High Peak Gain (dB)	4			•	0
	High Center Frequency (Hz)	4				10000
High Q Factor 2	High Q Factor	4			Þ	2

Name: audiopluginexample.ParametricEqualizerWithUDP

Type: System object plugin

**Description:** Extends audiopluginexample.ParametricEqualizer by adding a UDP sender. Adding a UDP sender enables the generated VST plugin to communicate with MATLAB. The digital audio workstation and MATLAB can then exchange information in real time. This plugin uses UDP to send the equalizer filter coefficients back to MATLAB for visualization purposes. You can alter this plugin to send the input or output audio instead of, or in addition to, the filter coefficients.

Related Example: Communicating Between a DAW and MATLAB via UDP

#### **Inspect Code**

edit audiopluginexample.ParametricEqualizerWithUDP

#### Run Plugin

audioTestBench audiopluginexample.ParametricEqualizerWithUDP

#### **Generate Plugin**

generateAudioPlugin audiopluginexample.ParametricEqualizerWithUDP

	Audio Test Bench: audiopluginexample.PitchShifter					
	INPUT: Audio File Reader 👻 🐼 OUTPUT: Audio Device Writer 👻 🐼					
	Pitch Shift (Semi-Tones)					
	Overlap (0.01					
Name: audiopluginexample.PitchShifter						
<b>Type:</b> System obj	ect plugin					
	plements a pitch-shifting algorithm using cross-fading between two ne-varying delays and gains.					
Related Exampl	e: Delay-based Pitch Shifter					
Inspect Code						
edit audiopluginexample.PitchShifter						
Run Plugin						
audioTestBench audiopluginexample.PitchShifter						
Generate Plugin						
generateAudioPlugin audiopluginexample.PitchShifter						

		Bench: Shelving Equalize				• × •	
	INPUT:	Audio File Reader	• 🗘	ουτρυτ: Αι	udio Device Writer	• Ø	
	Low Shelf	Cutoff (Hz)			Þ	200	
	Low Sh	elf Gain			•	0	
	Low Sh	elf Slope			·	1.5	
	High Shelf	Cutoff (Hz)			•	200	
	High St	elf Gain			•	0	
	High Sh	elf Slope			•	1.5	
Name: audiopluginexample.ShelvingEqualizer							
<b>Type:</b> System object plugin							
<b>Description:</b> Implements a shelving equalizer with tunable cutoffs, gains, and slopes. The plugin uses designShelvingEQ to obtain filter coefficients and dsp.BiquadFilter to implement filtering.							
Related Example: Tunable Filtering and Visualization Using Audio Plug-Ins							
Inspect Code							
edit audiopluginexample.ShelvingEqualizer							
Run Plugin							
audioTestBench audiopluginexample.ShelvingEqualizer							
Generate Plugin	1						
generateAudioPl	ugin <mark>a</mark>	udioplugi	nexample	.Shelvin	gEqualize	r	

	Audio Test Bench: Variab				
	INPUT: Audio File	e Reader 🔻 🔘	OUTPUT: Audio Device	e Writer 🔻 👰	
	Low Cutoff (Hz)			> 20	
	High Cutoff (Hz)	4		18000	
	Low Slope (dB/octave)	12	•		
	High Slope (dB/octave)	30	•		
Name: audiopluginexample.VarSlopeBandpassFilter					
Type: System object plugin					
<b>Description:</b> Implements a variable slope IIR bandpass filter with tunable cutoff frequencies and slopes. The plugin uses designVarSlopeFilter to obtain filter coefficients and dsp.BiquadFilter to implement filtering.					
Related Example: Tunable Filtering and Visualization Using Audio Plug-Ins					
Inspect Code					
edit audiopluginexample.VarSlopeBandpassFilter					
Run Plugin					
audioTestBench audiopluginexample.VarSlopeBandpassFilter					
Generate Plugin					
generateAudioPlugin audiopluginexample.VarSlopeBandpassFilter					

## Sample Audio Files

File Name	Audio Information
Ambiance-16-44p1-mono-12secs.wav	NumChannels: 1
<b>Description:</b> Footfalls in a noisy office	SampleRate: 44,100 Hz
hallway	Duration: 12.2369 seconds
	BitsPerSample: 16
AudioArray-16-16-4channels-20secs.	NumChannels: 4
<b>Description:</b> Moving noise source	SampleRate: 16,000 Hz
	Duration: 20.0320 seconds
	BitsPerSample: 16
Counting-16-44p1-mono-15secs.wav	NumChannels: 1
<b>Description:</b> Male voice counts to 10	SampleRate: 44,100 Hz
	Duration: 15.5341 seconds
	BitsPerSample: 16
Engine-16-44p1-stereo-20sec.wav	NumChannels: 2
<b>Description:</b> Running motor engine	SampleRate: 44,100 Hz
	Duration: 20.0156 seconds
	BitsPerSample: 16
FunkyDrums-44p1-stereo-25secs.mp3	NumChannels: 2
<b>Description:</b> Funky synthetic drum beat	SampleRate: 44,100 Hz
	Duration: 25.3127 seconds
	BitRate: 320
FunkyDrums-48-stereo-25secs.mp3	NumChannels: 1

File Name	Audio Information
<b>Description:</b> Funky synthetic drum beat	SampleRate: 48,000 Hz
	Duration: 25.3127 seconds
	BitRate: 320
JetAirplane-16-11p025-	NumChannels: 1
mono-16secs.wav	SampleRate: 11,025 Hz
<b>Description:</b> Jet airplane	Duration: 16.3468 seconds
	BitsPerSample: 16
MainStreetOne-24-96-	NumChannels: 2
stereo-63secs.wav	SampleRate: 96,000 Hz
<b>Description:</b> Ambient sounds of a busy street (bird chirps, cars, mumbling)	Duration: 63.2967 seconds
	BitsPerSample: 24
RandomOscThree-24-96-	NumChannels: 2
stereo-13secs.aif	SampleRate: 96,000 Hz
<b>Description:</b> Synthetic percussive tone scale	Duration: 13.1868 seconds
	BitsPerSample: 24
RockDrums-44p1-stereo-11secs.mp3	NumChannels: 2
Description: Rock drums	SampleRate: 44,100 Hz
	Duration: 11.4678 seconds
	BitRate: 320

File Name	Audio Information
RockDrums-48-stereo-11secs.mp3	NumChannels: 2
Description: Rock drums	SampleRate: 48,000 Hz
	Duration: 11.4678 seconds
	BitRate: 320
RockGuitar-16-44p1-	NumChannels: 2
stereo-72secs.wav	SampleRate: 44,100 Hz
<b>Description:</b> Rock guitar with distortion	Duration: 72.4695 seconds
	BitsPerSample: 16
RockGuitar-16-96-	NumChannels: 2
stereo-72secs.flac	SampleRate: 96,000 Hz
<b>Description:</b> Rock guitar with distortion	Duration: 72.500 seconds
	BitsPerSample: 16
SoftGuitar-44p1_mono-10mins.ogg	NumChannels: 1
<b>Description:</b> Solo acoustic folk guitar	SampleRate: 44,100 Hz
	Duration: 596.3719 seconds
SpeechDFT-16-8-mono-5secs.wav	NumChannels: 1
<b>Description:</b> Male voice speaking	SampleRate: 8,000 Hz
	Duration: 4.9902 seconds
	BitsPerSample: 16

File Name	Audio Information
TrainWhistle-16-44p1- mono-9secs.wav <b>Description:</b> Train whistle	NumChannels: 1 SampleRate: 44,100 Hz Duration: 9.3344 seconds BitsPerSample: 16
Turbine-16-44p1-mono-22secs.wav <b>Description:</b> Turbine	NumChannels: 1 SampleRate: 44,100 Hz Duration: 22.4305 seconds BitsPerSample: 16
WashingMachine-16-44p1- stereo-10secs.wav <b>Description:</b> Washing machine	NumChannels: 2 SampleRate: 44,100 Hz Duration: 18.0651 seconds BitsPerSample: 16
WaveGuideLoopOne-24-96- stereo-10secs.aif <b>Description:</b> Synthetic percussive tone	NumChannels: 2 SampleRate: 96,000 Hz Duration: 10.5495 seconds BitsPerSample: 24
guitar10min.ogg <b>Description:</b> Solo acoustic folk guitar	NumChannels: 2 SampleRate: 44,100 Hz Duration: 595.2392 seconds
handel.ogg <b>Description:</b> Hallelujah chorus	NumChannels: 1 SampleRate: 44,100 Hz Duration: 8.9249 seconds

File Name	Audio Information
audio48kHz.wav	NumChannels: 1
<b>Description:</b> Rock with vocals, drums, and guitar	SampleRate: 48,000 Hz
	Duration: 8.9634 seconds
	BitsPerSample: 16
dspafsx_mono.wav	NumChannels: 1
<b>Description:</b> Electric guitar solo	SampleRate: 16,000 Hz
	Duration: 3.7500 seconds
	BitsPerSample: 8
dspafxf.wav	NumChannels: 1
<b>Description:</b> Drum beat with trap and bass	SampleRate: 22,050 Hz
	Duration: 3.9343 seconds
	BitsPerSample: 16
speech_dft_8kHz.wav	NumChannels: 1
<b>Description:</b> Male voice speaking	SampleRate: 8000 Hz
	Duration: 4.9902 seconds
	BitsPerSample: 16
Swept_int.wav	NumChannels: 1
<b>Description:</b> Tone sweep	SampleRate: 96,000 Hz
	Duration: 8 seconds
	BitsPerSample: 32

File Name	Audio Information
speech_dft.mp3	NumChannels: 1
Description: Male voice speaking	SampleRate: 22,500 Hz
	Duration: 5.1199 seconds
	BitRate: 64

#### See Also

| | Audio Test Bench | audioPluginInterface | audioPluginParameter

## More About

• "Use the Audio Test Bench" on page 2-2