Audio System Toolbox™

User's Guide

MATLAB® & SIMULINK®

R2016a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Audio System Toolbox™ User's Guide

Revision History

March 2016	Online only	New for Version 1.0 (Release 2016a)

Contents

1 Musical Instrument Digital Interface

2 Use the Audio Test Bench

3 Audio Plugin Example Gallery

1

Musical Instrument Digital Interface

Musical Instrument Digital Interface (MIDI)

In this section...

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™ product enables you to use MIDI control surfaces to control MATLAB® programs and Simulink® 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:

```
>> [control, device] = midiid
Move the control you wish to identify; type ^C to abort.
Waiting for control message... done
control =
        1082
device =
BCF2000
>> setpref('midi', 'DefaultDevice', device)
>>
```

This preference persists across MATLAB sessions, so you only have to set it once, unless you want to change devices.

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

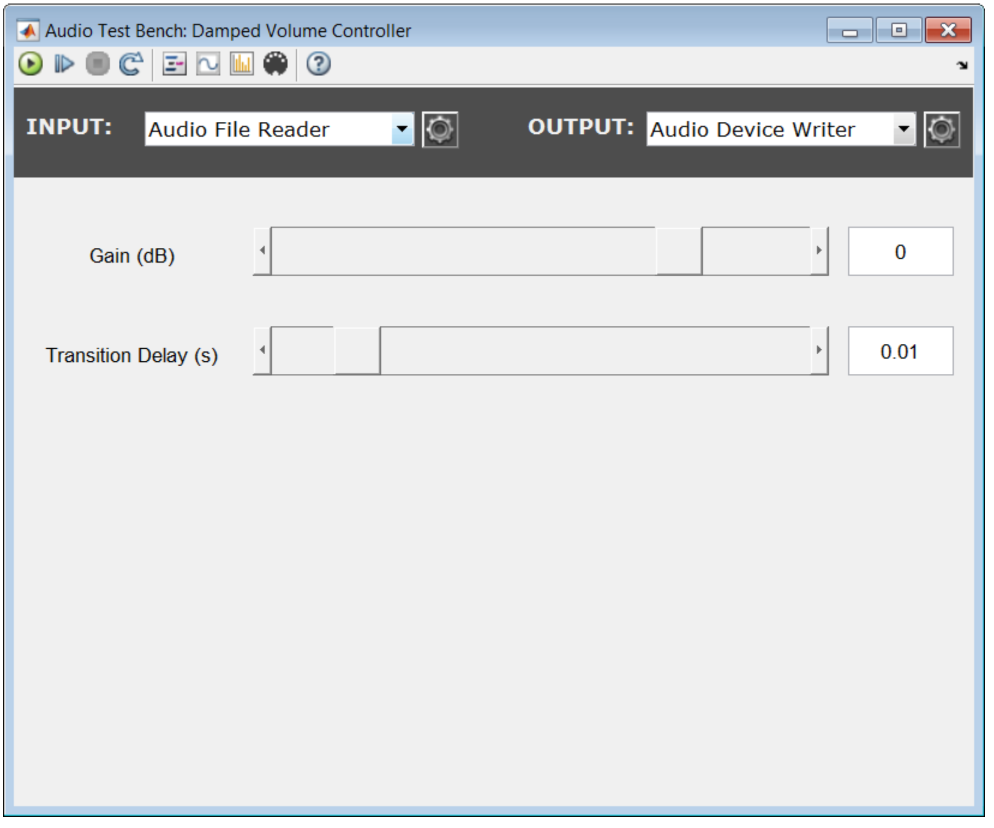
Use the Audio Test Bench

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
```



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.

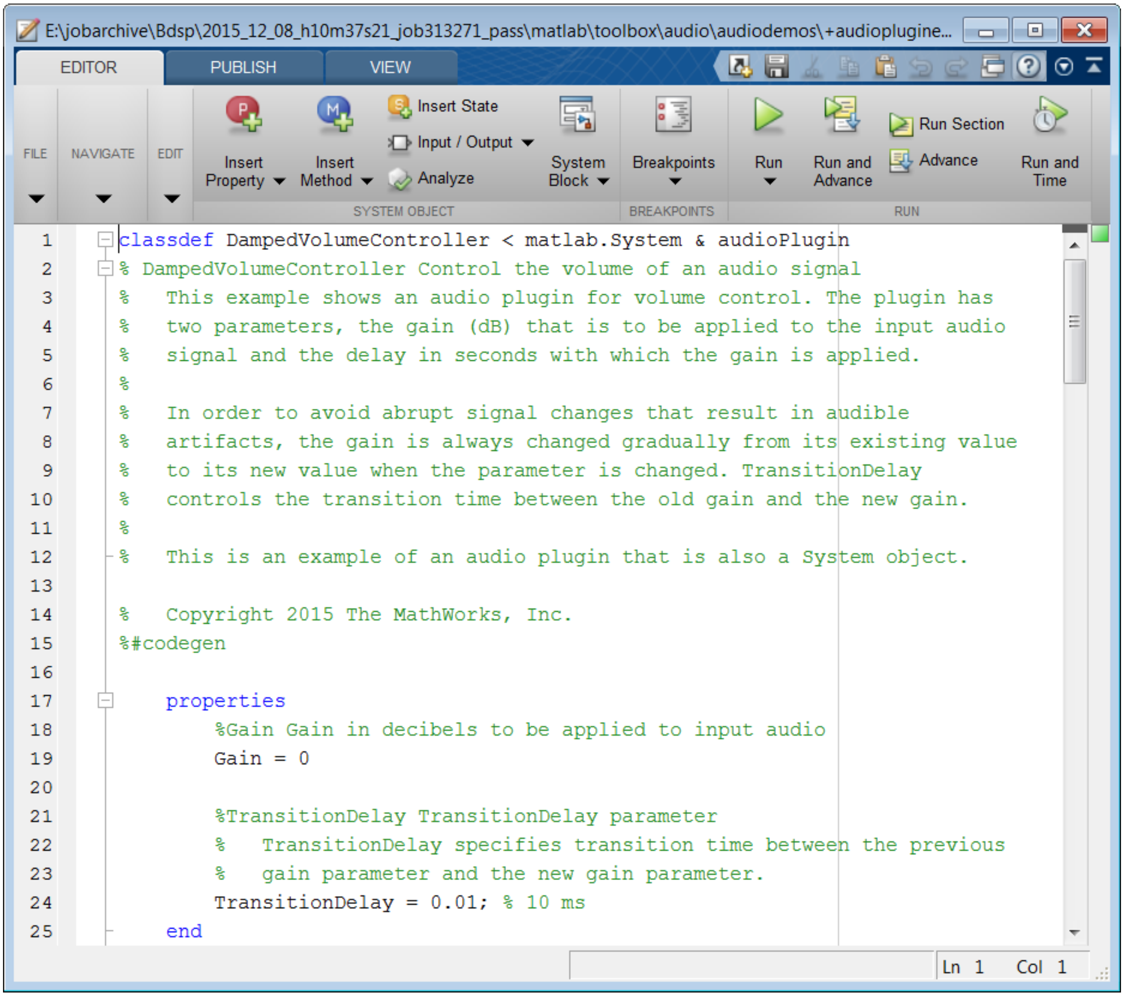
To reset internal states of your audio plugin and return the sliders to their initial positions, click .

Click to run the audio test bench again.

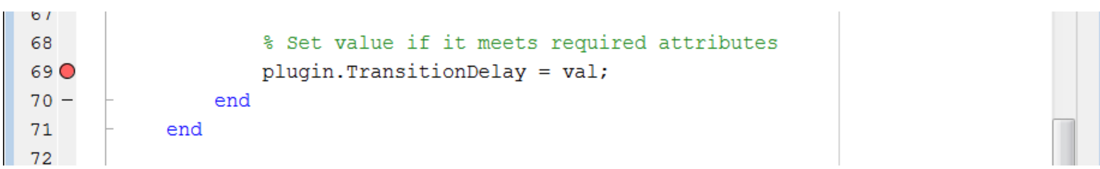
Debug Source Code of Audio Plugin

To pause the audio test bench, click .

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



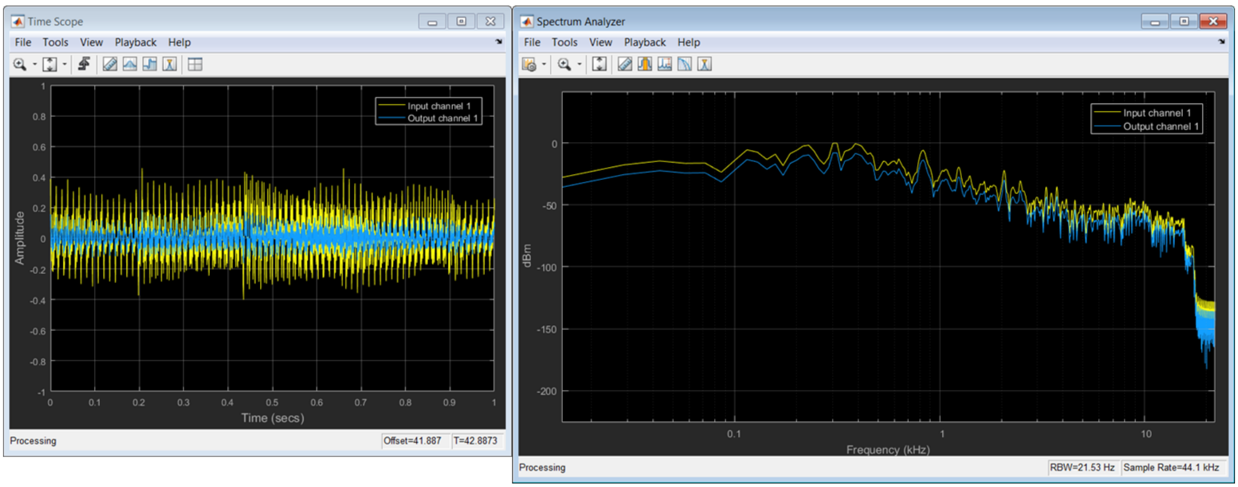
You can inspect the source code of your audio plugin, set breakpoints on it, and modify the code. Set a breakpoint at line 69, and then click the Run button on your audio test bench.



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

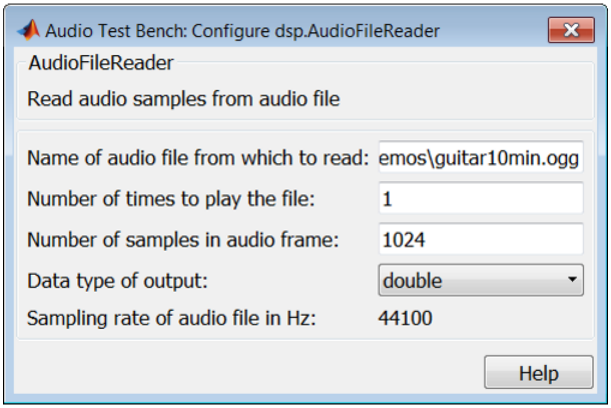
To open the Time Scope to visualize the time-domain input and output for your audio plugin, click . To open the spectrum analyzer to visualize the frequency-domain input and output, click .



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`.

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.

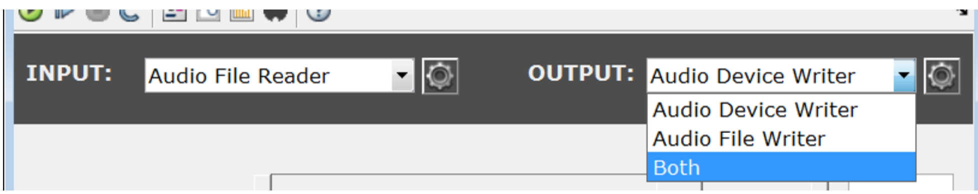
In the **Name of audio file from which to read** box, enter: `RockDrums-44p1-stereo-11secs.mp3`

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 .

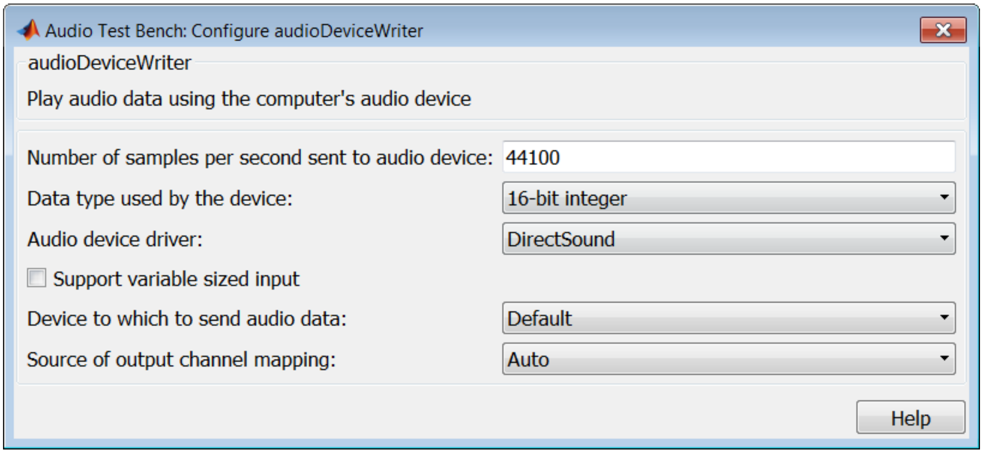
See audioDeviceReader and dsp.AudioFileReader for information about modifying parameters of your audio reader.

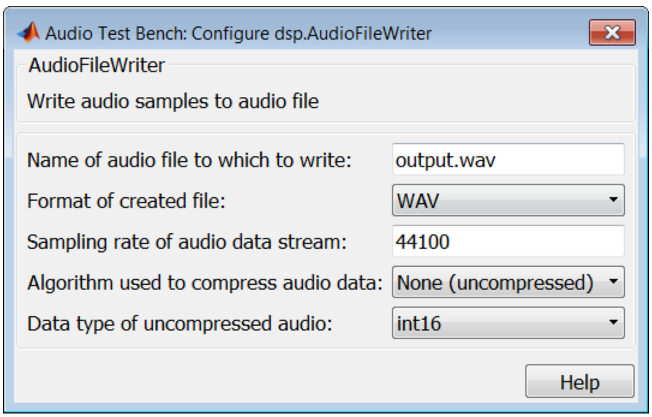
Configure Output from Audio Test Bench

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 .





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 . 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

audioPlugin | Audio Test Bench | `generateAudioPlugin` | `validateAudioPlugin`

More About

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

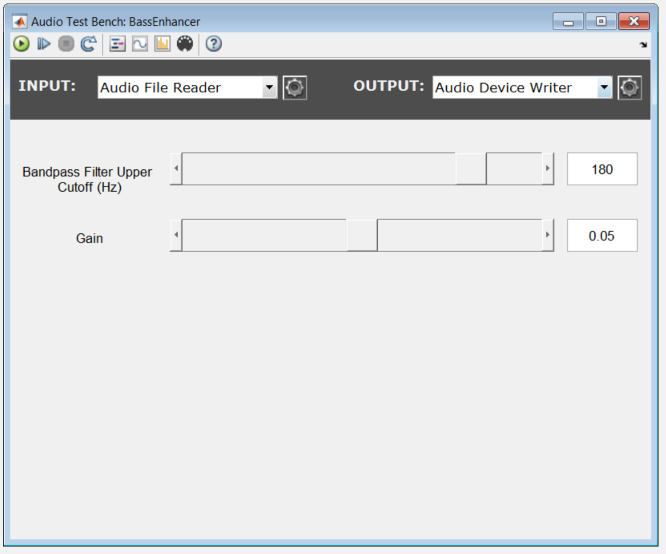
3

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



Name: `audiopluginexample.BassEnhancer`

Type: System object™ 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

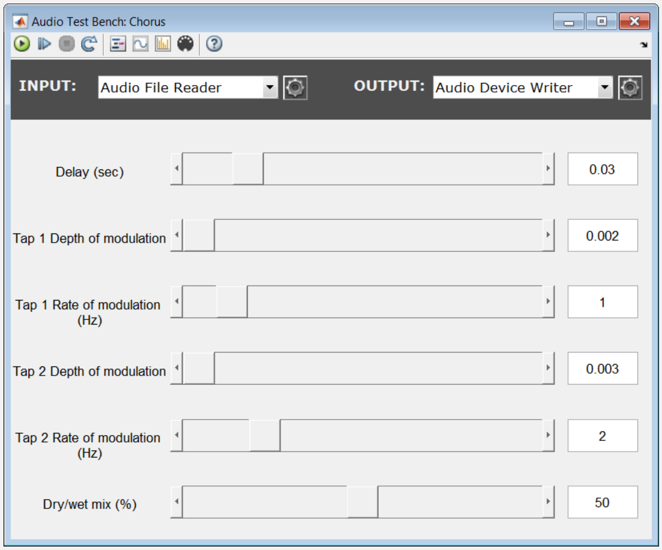
```
edit audiopluginexample.BassEnhancer
```

Run Plugin

`audioTestBench audiopluginexample.BassEnhancer`

Generate Plugin

`generateAudioPlugin audiopluginexample.BassEnhancer`



Name: `audiopluginexample.Chorus`

Type: Basic plugin

Description: Adds an audio chorus effect. The chorus effect is implemented by modulating two delay lines.

Inspect Code

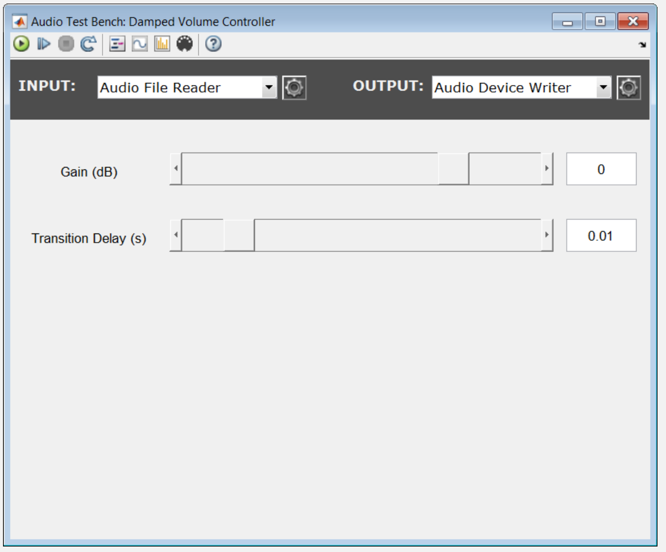
`edit audiopluginexample.Chorus`

Run Plugin

`audioTestBench audiopluginexample.Chorus`

Generate Plugin

`generateAudioPlugin audiopluginexample.Chorus`



Name: `audiopluginexample.DampedVolumeController`

Type: System object plugin

Description: Damps the volume control of an audio signal. The plugin has two parameters: the gain that is applied to the input audio signal, and the transition delay for gain application in seconds.

Inspect Code

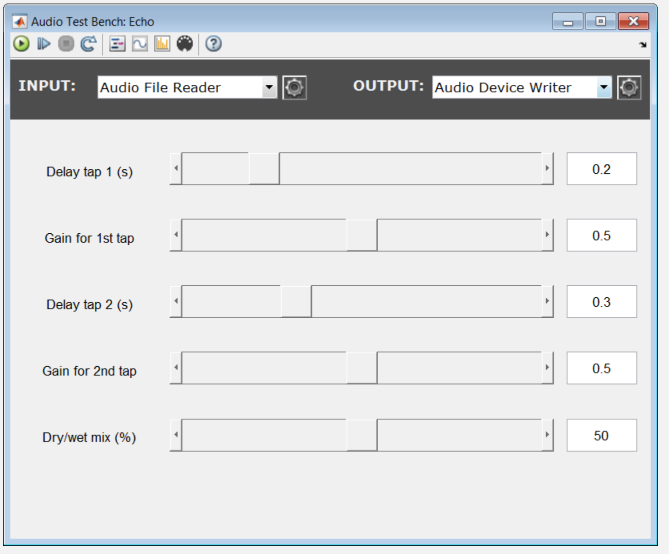
```
edit audiopluginexample.DampedVolumeController
```

Run Plugin

```
audioTestBench audiopluginexample.DampedVolumeController
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.DampedVolumeController
```



Name: `audiopluginexample.Echo`

Type: Basic plugin

Description: Implements an audio echo effect using two delay lines. The plugin user tunes the delay taps in seconds, the gain of the delay taps, and the output dry/wet mix.

Inspect Code

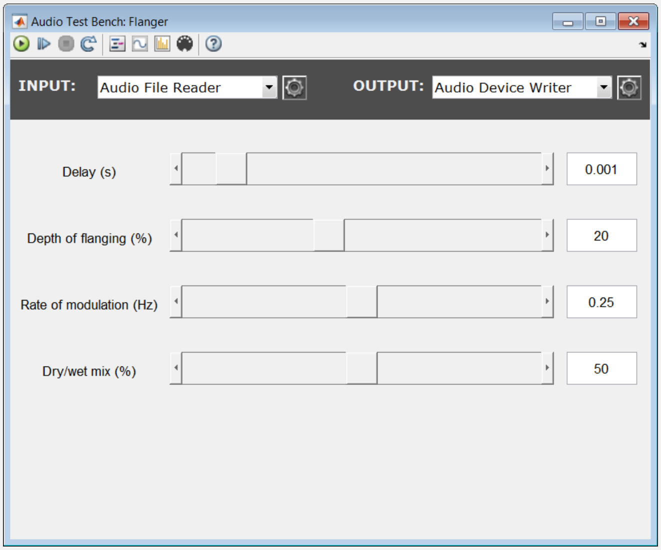
```
edit audiopluginexample.Echo
```

Run Plugin

```
audioTestBench audiopluginexample.Echo
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.Echo
```



Name: `audiopluginexample.Flanger`

Type: Basic plugin

Description: Implements an audio flanging effect using a modulated delay line. The 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

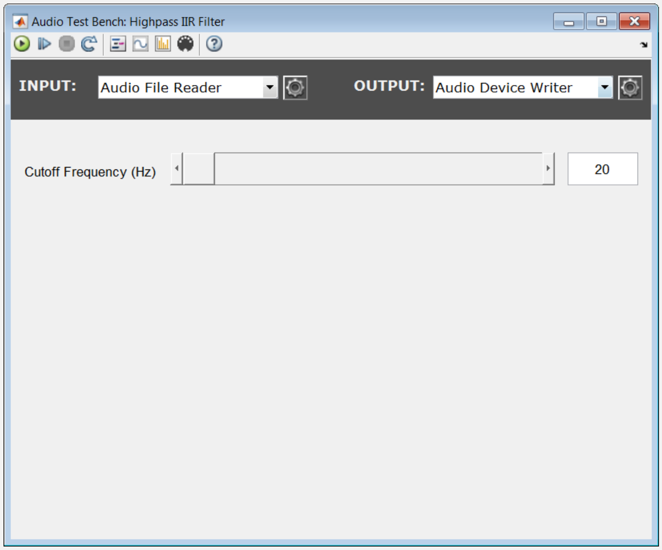
```
edit audiopluginexample.Flanger
```

Run Plugin

```
audioTestBench audiopluginexample.Flanger
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.Flanger
```



Name: `audiopluginexample.HighPassIIRFilter`

Type: System object plugin

Description: Implements a second-order IIR highpass filter with tunable cutoff frequency. The plugin uses dsp.BiquadFilter to implement filtering.

Related Example: Tunable Filtering and Visualization Using Audio Plug-Ins

Inspect Code

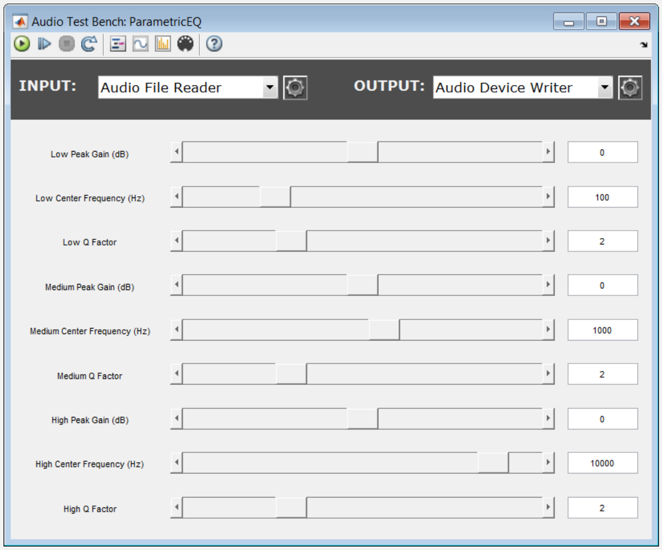
```
edit audiopluginexample.HighPassIIRFilter
```

Run Plugin

```
audioTestBench audiopluginexample.HighPassIIRFilter
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.HighPassIIRFilter
```



Name: `audiopluginexample.ParametricEqualizer`

Type: System object plugin

Description: 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

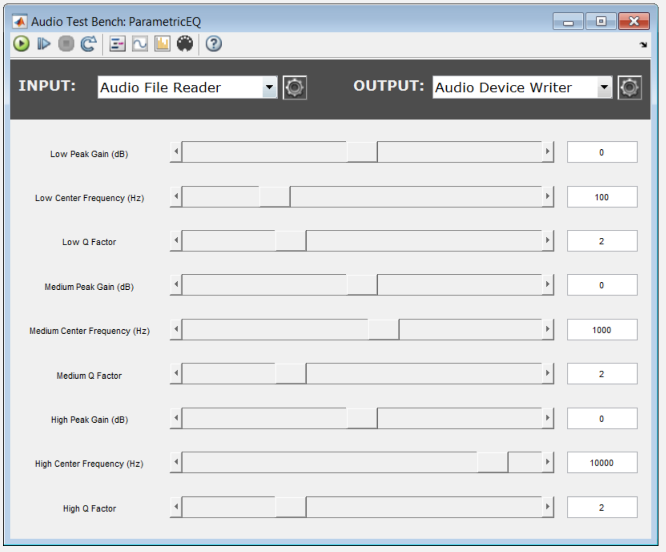
```
edit audiopluginexample.ParametricEqualizer
```

Run Plugin

```
audioTestBench audiopluginexample.ParametricEqualizer
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.ParametricEqualizer
```



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

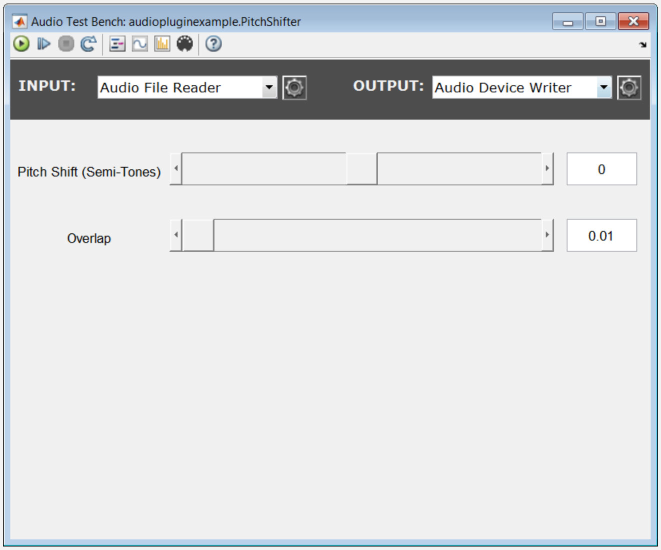
```
edit audiopluginexample.ParametricEqualizerWithUDP
```

Run Plugin

```
audioTestBench audiopluginexample.ParametricEqualizerWithUDP
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.ParametricEqualizerWithUDP
```



Name: `audiopluginexample.PitchShifter`

Type: System object plugin

Description: Implements a pitch-shifting algorithm using cross-fading between two channels with time-varying delays and gains.

Related Example: Delay-based Pitch Shifter

Inspect Code

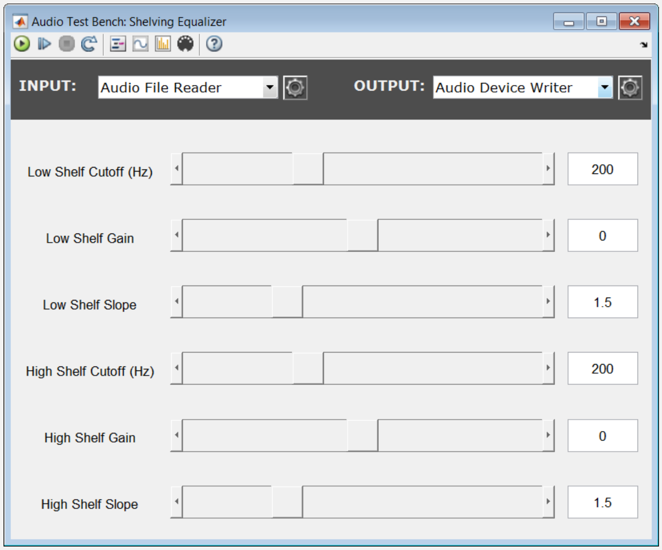
```
edit audiopluginexample.PitchShifter
```

Run Plugin

```
audioTestBench audiopluginexample.PitchShifter
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.PitchShifter
```



Name: `audiopluginexample.ShelvingEqualizer`

Type: System object plugin

Description: 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

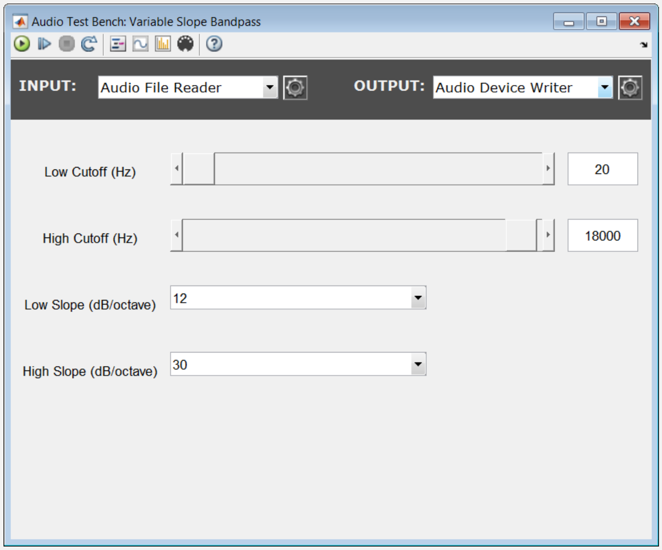
```
edit audiopluginexample.ShelvingEqualizer
```

Run Plugin

```
audioTestBench audiopluginexample.ShelvingEqualizer
```

Generate Plugin

```
generateAudioPlugin audiopluginexample.ShelvingEqualizer
```



Name: `audiopluginexample.VarSlopeBandpassFilter`

Type: System object plugin

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

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

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

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

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

File Name	Audio Information
`speech_dft.mp3` **Description:** Male voice speaking	**NumChannels:** 1 **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