Real Time Audio Processor

Tucker McClure ([Tucker.McClure@MathWorks.com](mailto:Tucker.McClure@MathWorks.com))

The RealTimeAudioProcessor makes it easy to stream or generate audio in real time in MATLAB. It is a class, which means any application could derive from a RealTimeAudioProcessor to become a real-time application. It uses several relatively new features of MATLAB to do this, discussed below.

The RealTimeAudioProcessor class is a System Object that interfaces with the DSP System Toolbox’s dsp.AudioPlayer and dsp.AudioRecord objects to implement low-latency operation of an algorithm on an audio stream. It is most effective with, but does not require, an ASIO audio driver (more in below).

# Requirements

* Windows
* MATLAB r2012a or greater
* Signal Processing Toolbox
* DSP System Toolbox
* Soundcard with ASIO driver (recommended)
* MIDI input device (recommended only for MidiDemo)

# Setup

## Windows

First, for usably-low latency, appropriate drivers are necessary for your soundcard. On Windows, a great standard for this interface is the ASIO standard developed by Steinberg GmbH. If a custom ASIO driver exists for your soundcard, this should be installed. Otherwise, ASIO4All can be used for most soundcards. This can be downloaded and installed for free. Note that some generic hardware, regardless of the driver used, will be unable to support simultaneous inputs and outputs.

Next, MATLAB must be configured to use the ASIO standard

* Start MATLAB.
* Select File::Preferences::DSP System Toolbox, select ASIO, and then click Ok.

The ASIO driver is now ready to use.

## Mac and Linux

These operating systems are currently unsupported.

# Execution of the Real Time Audio Processor

The RealTimeAudioProcessor configures a dsp.AudioPlayer and dsp.AudioRecorder with input sample rates, buffer sizes, and devices and seeds the output buffer with one frame of silence (necessary to configure the buffer). It then waits for a Play command. Upon receiving this command, it immediately enters a real-time-locked loop where it collects the inputs and passes these inputs and corresponding time data to the step method. This method constructs the appropriate output frame, which by default is just silence. The loop exits when an end time is reached or the UI figure is closed.

There are two ways to use the RealTimeAudioProcessor. First, a subclass can derive from RealTimeAudioProcessor and overload the stepImpl method. This method is most useful for complex algorithms. Second, the derived subclass HandlePlayer can be used. This object takes a handle to a function which will create a single frame of outputs from the given inputs, time data, and current state and will update the output state. This algorithm is most useful for simple algorithms with small states.

## Custom Object Mode

An example of a custom object is provided in QwertyDemo.m. The script qwerty\_driver.m executes this demo. This demo allows the user to trigger sound from keyboard input. When latency values are set very low, there is no noticeable lag between pressing the key and hearing the sound. This provides tactile meaning for “low latency”.

A more advanced example of a custom object is MidiDemo.m, executed by midi\_driver.m. This opens a Musical Instrument Digital Interface (MIDI) input on the system, collects the input over time, and maps these inputs to musical notes, allowing the user to play a MATLAB object as a live synthesizer with any standard MIDI keyboard. That this can be played “live” shows fairly complex behavior within MATLAB during a real-time loop. It’s also way fun. This only works on Windows, as the MIDI input MEX function (midi\_interface.cpp) uses the Windows API.

## Handle Mode

An example of using the HandlePlayer object is provided in handle\_driver.m. The HandlePlayer takes a function handle on creation and passes the inputs through the specified function as its stepImpl. This effectively turns any algorithm into a streaming, real-time algorithm (if it can run quickly enough).

Copyright 2012 The MathWorks, Inc.