```
/*****************************************/
/*
 playsaw.cpp
 by Gary P. Scavone, 2006
 Modified for hw822_playsine.cpp
 // REPLACE WITH
 Modified for hw723_playsine_rta.cpp
 John Ellinger CS312
*/
/*****************************************/
#include "RtAudio.h"
// REPLACE WITH
#ifndef RTAUDIOUTILS_H_
#include "rtaudioutils.h"
#endif

// IMPORTANT: if you get an unwanted space in
// #include < iostream>
// fix it

#include <iostream>
#include <cmath> // for M_PI

// EITHER/OR
typedef float MY_TYPE;
#define FORMAT RTAUDIO_FLOAT32
// typedef double MY_TYPE;
// #define FORMAT RTAUDIO_FLOAT64
// EITHER/OR END

const int FS = 44100;      // CD sample rate
const MY_TYPE T = 1.0 / FS; // sample period
const MY_TYPE k2PI = M_PI * 2.0;
const MY_TYPE k2PIT = k2PI * T;

void errorCallback(RtAudioError::Type type, const std::string &errorText)
{
  // This example error handling function does exactly the same thing
  // as the embedded RtAudio::error() function.
  std::cout << "in errorCallback" << std::endl;
  if (type == RtAudioError::WARNING)
    std::cerr << '\n'
          << errorText << "\n\n";
  else if (type != RtAudioError::WARNING)
    throw(RtAudioError(errorText, type));
}

unsigned int channels = 1;
RtAudio::StreamOptions options;
unsigned int frameCounter = 0;
bool checkCount = false;
unsigned int nFrames = 0;
const unsigned int callbackReturnValue = 1;

// One-channel sine wave generator replaces saw callback function
int sine(void *outputBuffer, void *inputBuffer, unsigned int nBufferFrames,
       double streamTime, RtAudioStreamStatus status, void *userData)
{
```

```
  MY_TYPE *buffer = (MY_TYPE *)outputBuffer;
  if (status)
    std::cout << "Stream underflow detected!" << std::endl;
  static MY_TYPE phz = 0;
  // //phase increment formula
  MY_TYPE freq = 440.0;
  MY_TYPE amp = 1.0;

  const MY_TYPE phzinc = k2PIT * freq;

 for (uint32_t i = 0; i < nBufferFrames; i++)
  {
    *buffer++ = amp * sin(phz);
    phz += phzinc;
    if (phz >= k2PI)
      phz -= k2PI;
  }
  frameCounter += nBufferFrames;
  if (checkCount && (frameCounter >= nFrames))
    return callbackReturnValue;
  return 0;
 }

 int main(int argc, char *argv[])
 {
  unsigned int bufferFrames;
  unsigned int fs = FS;
  unsigned int device = 0;
  unsigned int offset = 0;

  RtAudio dac;
  if (dac.getDeviceCount() < 1)
  {
    std::cout << "\nNo audio devices found!\n";
    exit(1);
  }

  MY_TYPE *data = (MY_TYPE *)calloc(channels, sizeof(MY_TYPE));

  // Let RtAudio print messages to stderr.
  dac.showWarnings(true);

  // Set our stream parameters for output only.
  bufferFrames = 512;
  RtAudio::StreamParameters oParams;
  oParams.deviceId = device;
  oParams.nChannels = channels;
  oParams.firstChannel = offset;

  if (device == 0)
    oParams.deviceId = dac.getDefaultOutputDevice();

  options.flags = RTAUDIO_HOG_DEVICE;
  options.flags |= RTAUDIO_SCHEDULE_REALTIME;
  options.flags |= RTAUDIO_NONINTERLEAVED;

 try
  {
```

```
    dac.openStream(&oParams, NULL, FORMAT, fs, &bufferFrames, &sine, (void *)data, &options, &errorCallback);
    dac.startStream();
  }
  catch (RtAudioError &e)
  {
    e.printMessage();
    goto cleanup;
  }

  char input;
  //std::cout << "Stream latency = " << dac.getStreamLatency() << "\n" << std::endl;
  std::cout << "\nPlaying ... press <enter> to quit (buffer size = " << bufferFrames << ").\n";
  std::cin.get(input);

  try
  {
    // Stop the stream
    dac.stopStream();
  }
  catch (RtAudioError &e)
  {
    e.printMessage();
  }

cleanup:
  if (dac.isStreamOpen())
    dac.closeStream();
  free(data);

  return 0;
}
```