```
//
// Programmer:    Craig Stuart Sapp <craig@ccrma.stanford.edu>
// Creation Date: Fri May 12 09:00:58 PDT 2006
// Last Modified: Wed Jun 21 08:30:52 PDT 2006 (subclassed to MazurkaPlugin)
// Filename:      MzSpectrogramHost.cpp
// URL:           http://sv.mazurka.org.uk/src/MzSpectrogramHost.cpp
// Documentation: http://sv.mazurka.org.uk/MzSpectrogramHost
// Syntax:        ANSI99 C++; vamp 0.9 plugin
//
// Description:   Demonstration of how to process spectral data supplied
//                by the host application.
//

#include "MzSpectrogramHost.h"

#include <math.h>


///////////////////////////////////////////////////////////////////////////
//
// Vamp Interface Functions
//

///////////////////////////////
//
// MzSpectrogramHost::MzSpectrogramHost -- class constructor.
//

MzSpectrogramHost::MzSpectrogramHost(float samplerate) :
      MazurkaPlugin(samplerate) {
   mz_minbin = 0;
   mz_maxbin = 0;
}



///////////////////////////////
//
// MzSpectrogramHost::~MzSpectrogramHost -- class destructor.
//

MzSpectrogramHost::~MzSpectrogramHost() {
   // do nothing
}


////////////////////////////////////////////////////////////
//
// required polymorphic functions inherited from PluginBase:
//

std::string MzSpectrogramHost::getName(void) const
   { return "mzspectrogramhost"; }

std::string MzSpectrogramHost::getMaker(void) const
   { return "The Mazurka Project"; }

std::string MzSpectrogramHost::getCopyright(void) const
   { return "2006 Craig Stuart Sapp"; }

std::string MzSpectrogramHost::getDescription(void) const
   { return "Host Spectrogram"; }

int MzSpectrogramHost::getPluginVersion(void) const {
   #define P_VER    "200606260"
   #define P_NAME   "MzSpectrogramHost"

   const char *v = "@@VampPluginID@" P_NAME "@" P_VER "@" __DATE__ "@@";
   if (v[0] != ’@’) { std::cerr << v << std::endl; return 0; }
   return atol(P_VER);
}


////////////////////////////////////////////////////////////
//
// optional polymorphic parameter functions inherited from PluginBase:
//
// Note that the getParameter() and setParameter() polymorphic functions
// are handled in the MazurkaPlugin class.
//

//////////////////////////////
//
// MzSpectrogramHost::getParameterDescriptors -- return a list of
//      the parameters which can control the plugin.
//

MzSpectrogramHost::ParameterList
MzSpectrogramHost::getParameterDescriptors(void) const {

   ParameterList       pdlist;
   ParameterDescriptor pd;

   // first parameter: The minimum spectral bin to display
   pd.name         = "minbin";
   pd.description  = "Minimum\nfrequency\nbin";
   pd.unit         = "";
   pd.minValue     = 0.0;
   pd.maxValue     = 50000.0;
   pd.defaultValue = 0.0;
   pd.isQuantized  = 1;
   pd.quantizeStep = 1.0;
   pdlist.push_back(pd);

   // second parameter: The maximum spectral bin to display
   pd.name         = "maxbin";
   pd.description  = "Maximum\nfrequency\nbin";
   pd.unit         = "";
   pd.minValue     = -1.0;
   pd.maxValue     = 50000.0;
   pd.defaultValue = -1.0;
   pd.isQuantized  = 1;
   pd.quantizeStep = 1.0;
   pdlist.push_back(pd);

   return pdlist;
}



////////////////////////////////////////////////////////////
//
// required polymorphic functions inherited from Plugin:
//

//////////////////////////////
//
// MzSpectrogramHost::getInputDomain -- the host application needs
//    to know if it should send either:
//
```

```
// TimeDomain      == Time samples from the audio waveform.
// FrequencyDomain == Spectral frequency frames which will arrive
//                    in an array of interleaved real, imaginary
//                    values for the complex spectrum (both positive
//                    and negative frequencies). Zero Hz being the
//                    first frequency sample and negative frequencies
//                    at the far end of the array as is usually done.
//                    Note that frequency data is transmitted from
//                    the host application as floats.  The data will
//                    be transmitted via the process() function which
//                    is defined further below.
//

MzSpectrogramHost::InputDomain MzSpectrogramHost::getInputDomain(void) const {
   return FrequencyDomain;
}



//////////////////////////////
//
// MzSpectrogramHost::getOutputDescriptors -- return a list describing
//    each of the available outputs for the object.  OutputList
//    is defined in the file vamp-sdk/Plugin.h:
//
// .name             == short name of output for computer use.  Must not
//                      contain spaces or punctuation.
// .description      == long name of output for human use.
// .unit             == the units or basic meaning of the data in the
//                      specified output.
// .hasFixedBinCount == true if each output feature (sample) has the
//                      same dimension.
// .binCount         == when hasFixedBinCount is true, then this is the
//                      number of values in each output feature.
//                      binCount=0 if timestamps are the only features,
//                      and they have no labels.
// .binNames         == optional description of each bin in a feature.
// .hasKnownExtent   == true if there is a fixed minimum and maximum
//                      value for the range of the output.
// .minValue         == range minimum if hasKnownExtent is true.
// .maxValue         == range maximum if hasKnownExtent is true.
// .isQuantized      == true if the data values are quantized.  Ignored
//                      if binCount is set to zero.
// .quantizeStep     == if isQuantized, then the size of the quantization,
//                      such as 1.0 for integers.
// .sampleType       == Enumeration with three possibilities:
//   OD::OneSamplePerStep    -- output feature will be aligned with
//                              the beginning time of the input block data.
//   OD::FixedSampleRate     -- results are evenly spaced according to
//                              .sampleRate (see below).
//   OD::VariableSampleRate  -- output features have individual timestamps.
// .sampleRate       == samples per second spacing of output features when
//                      sampleType is set toFixedSampleRate.
//                      Ignored if sampleType is set to OneSamplePerStep
//                      since the start time of the input block will be used.
//                      Usually set the sampleRate to 0.0 if VariableSampleRate
//                      is used; otherwise, see vamp-sdk/Plugin.h for what
//                      positive sampleRates would mean.
//

MzSpectrogramHost::OutputList
MzSpectrogramHost::getOutputDescriptors(void) const {

   OutputList       list;
   OutputDescriptor od;
```

```

   // First and only output channel:
   od.name             = "magnitude";
   od.description      = "Magnitude Spectrum";
   od.unit             = "decibels";
   od.hasFixedBinCount = true;
   od.binCount         = mz_maxbin - mz_minbin + 1;
   od.hasKnownExtents  = false;
   // od.minValue      = 0.0;
   // od.maxValue      = 0.0;
   od.isQuantized      = false;
   // od.quantizeStep  = 1.0;
   od.sampleType       = OutputDescriptor::OneSamplePerStep;
   // od.sampleRate    = 0.0;
   list.push_back(od);

   return list;
}



//////////////////////////////
//
// MzSpectrogramHost::initialise -- this function is called once
//     before the first call to process().
//

bool MzSpectrogramHost::initialise(size_t channels, size_t stepsize,
      size_t blocksize) {

   if (channels < getMinChannelCount() || channels > getMaxChannelCount()) {
      return false;
   }

   // step size and block size should never be zero
   if (stepsize <= 0 || blocksize <= 0) {
      return false;
   }

   setBlockSize(blocksize);
   setStepSize(stepsize);
   setChannelCount(channels);

   mz_minbin = getParameterInt("minbin");
   mz_maxbin = getParameterInt("maxbin");

   if (mz_minbin >= getBlockSize()/4) { mz_minbin = getBlockSize()/4-1; }
   if (mz_maxbin >= getBlockSize()/4) { mz_maxbin = getBlockSize()/4-1; }
   if (mz_maxbin <  0)                { mz_maxbin = getBlockSize()/4-1; }
   if (mz_maxbin >  mz_minbin)        { std::swap(mz_minbin, mz_maxbin); }

   return true;
}



//////////////////////////////
//
// MzSpectrogramHost::process -- This function is called sequentially on the
//    input data, block by block.  After the sequence of blocks has been
//    processed with process(), the function getRemainingFeatures() will
//    be called.
//
// Here is a reference chart for the Feature struct:
//
```

```
// .hasTimestamp   == If the OutputDescriptor.sampleType is set to
//                    VariableSampleRate, then this should be "true".
// .timestamp      == The time at which the feature occurs in the time stream.
// .values         == The float values for the feature.  Should match
//                    OD::binCount.
// .label          == Text associated with the feature (for time instants).
//

#define ZEROLOG         -120.0

MzSpectrogramHost::FeatureSet
MzSpectrogramHost::process(float **inputbufs, Vamp::RealTime timestamp) {

   if (getChannelCount() <= 0) {
      std::cerr << "ERROR: MzSpectrogramHost::process: "
                << "MzSpectrogramHost has not been initialized"
                << std::endl;
      return FeatureSet();
   }

   FeatureSet returnFeatures;
   Feature    feature;

   feature.hasTimestamp = false;  // constant sampling rate, so don’t need.

   float real;       // real part of frequency spectrum
   float imag;       // imaginary part of frequency spectrum
   float magnitude;  // temporary holding space for magnitude value

   for (int i=mz_minbin; i<=mz_maxbin; i++) {
      real  = inputbufs[0][2*i];
      imag  = inputbufs[0][2*i + 1];
      magnitude = real * real + imag * imag;

      // convert to decibels:
      if (magnitude <= 0) { magnitude = ZEROLOG; }
      else                { magnitude = 10.0 * log10(magnitude); }

      feature.values.push_back(magnitude);
   }

   // Append new frame of data onto the output channel
   // specified in the function getOutputDescriptors():
   returnFeatures[0].push_back(feature);

   return returnFeatures;
}



//////////////////////////////
//
// MzSpectrogramHost::getRemainingFeatures -- This function is called
//    after the last call to process() on the input data stream has
//    been completed.  Features which are non-causal can be calculated
//    at this point.  See the comment above the process() function
//    for the format of output Features.
//

MzSpectrogramHost::FeatureSet MzSpectrogramHost::getRemainingFeatures(void) {
   // no remaining features, so return a dummy feature
   return FeatureSet();
}
```

```
//////////////////////////////
//
// MzSpectrogramHost::reset -- This function may be called after data processing
//    has been started with the process() function.  It will be called when
//    processing has been interrupted for some reason and the processing
//    sequence needs to be restarted (and current analysis output thrown out).
//    After this function is called, process() will start at the beginning
//    of the input selection as if initialise() had just been called.
//    Note, however, that initialise() will NOT be called before processing
//    is restarted after a reset().
//

void MzSpectrogramHost::reset(void) {
   // no actions necessary to reset this plugin
}


///////////////////////////////////////////////////////////////////////////
//
// Non-Interface Functions
//


// no non-interface functions
```