User Manual

The Synthesis ToolKit in C++

by Perry R. Cook and Gary P. Scavone

©1995–2002

Contents

1 STK Hierarchical Index **1**

1.1 STK Class Hierarchy 1

2 STK Compound Index **5**

2.1 STK Compound List 5

3 STK Page Index **9**

3.1 STK Related Pages 9

4 STK Class Documentation **11**

4.1 ADSR Class Reference 11

4.2 BandedWG Class Reference 14

4.3 BeeThree Class Reference 16

4.4 BiQuad Class Reference 18

4.5 BlowBotl Class Reference 21

4.6 BlowHole Class Reference 23

4.7 Bowed Class Reference 25

4.8 BowTabl Class Reference 27

4.9 Brass Class Reference 29

4.10 Chorus Class Reference 31

4.11 Clarinet Class Reference 33

4.12 Delay Class Reference 35

4.13 DelayA Class Reference 37

4.14 DelayL Class Reference 39

4.15 Drummer Class Reference 41
4.16 Echo Class Reference 43
4.17 Envelope Class Reference 45
4.18 Filter Class Reference 47
4.19 Flute Class Reference 50
4.20 FM Class Reference 52
4.21 FMVoices Class Reference 55
4.22 FormSwep Class Reference 57
4.23 HevyMetl Class Reference 60
4.24 Instrmnt Class Reference 62
4.25 JCRev Class Reference 64
4.26 JetTabl Class Reference 65
4.27 Mandolin Class Reference 67
4.28 Mesh2D Class Reference 69
4.29 Messager Class Reference 71
4.30 Modal Class Reference 73
4.31 ModalBar Class Reference 75
4.32 Modulate Class Reference 77
4.33 Moog Class Reference 79
4.34 Noise Class Reference 81
4.35 NRev Class Reference 83
4.36 OnePole Class Reference 84
4.37 OneZero Class Reference 86
4.38 PercFlut Class Reference 88
4.39 PitShift Class Reference 90
4.40 Plucked Class Reference 92
4.41 PluckTwo Class Reference 94
4.42 PoleZero Class Reference 96
4.43 PRCRev Class Reference 99
4.44 ReedTabl Class Reference 100
4.45 Resonate Class Reference 102

4.46 Reverb Class Reference 104
4.47 Rhodey Class Reference 106
4.48 RtDuplex Class Reference 108
4.49 RtMidi Class Reference 111
4.50 RtWvIn Class Reference 113
4.51 RtWvOut Class Reference 116
4.52 Sampler Class Reference 119
4.53 Saxofony Class Reference 121
4.54 Shakers Class Reference 123
4.55 Simple Class Reference 126
4.56 Sitar Class Reference 128
4.57 SKINI Class Reference 130
4.58 Socket Class Reference 133
4.59 StifKarp Class Reference 137
4.60 Stk Class Reference 139
4.61 StkError Class Reference 143
4.62 SubNoise Class Reference 144
4.63 Table Class Reference 146
4.64 TcpWvIn Class Reference 148
4.65 TcpWvOut Class Reference 151
4.66 Thread Class Reference 154
4.67 TubeBell Class Reference 156
4.68 TwoPole Class Reference 158
4.69 TwoZero Class Reference 161
4.70 WaveLoop Class Reference 164
4.71 Wurley Class Reference 166
4.72 WvIn Class Reference 168
4.73 WvOut Class Reference 173

5 STK Page Documentation 177

5.1 General Information 177

5.2 Class Documentation 180
5.3 Download and Release Notes 181
5.4 Release Notes: 181
5.5 Usage Documentation 185
5.6 Directory Structure: 185
5.7 Compiling: 186
5.8 Control Data: 187
5.9 Demo: STK Instruments 187
5.10 Demo: Non-Realtime Use 188
5.11 Demo: Realtime Use 189
5.12 Realtime Control Input using Tcl/Tk Graphical User Interfaces: 189
5.13 Realtime MIDI Control Input: 190
5.14 The Mail List 191
5.15 System Requirements 192
5.16 Tutorial 193
5.17 Introduction 193
5.18 Getting Started 193
5.19 Compiling 196
5.20 "Realtime" vs. "Non-Realtime" 197
5.21 To Be Continued 197
5.22 Synthesis toolKit Instrument Network Interface (SKINI) 198
5.23 MIDI Compatibility 198
5.24 Why SKINI? 199
5.25 SKINI Messages 199
5.26 C Files Used To Implement SKINI 200
5.27 SKINI Messages and the SKINI Parser: 200
5.28 A Short SKINI File: 201
5.29 The SKINI.tbl File and Message Parsing: 202
5.30 Using SKINI: 204

Chapter 1

STK Hierarchical Index

1.1 STK Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Stk 139
  - BowTabl 27
  - Chorus 31
  - Echo 43
  - Envelope 45
    - ADSR 11
  - Filter 47
    - BiQuad 18
      - FormSwep 57
    - Delay 35
      - DelayA 37
      - DelayL 39
    - OnePole 84
    - OneZero 86
    - PoleZero 96
    - TwoPole 158
    - TwoZero 161
  - Instrmnt 62
    - BandedWG 14
    - BlowBotl 21
    - BlowHole 23
    - Bowed 25
    - Brass 29
    - Clarinet 33
    - Drummer 41

Flute . . . 50
FM . . . 52
BeeThree . . . 16
FMVoices . . . 55
HevyMetl . . . 60
PercFlut . . . 88
Rhodey . . . 106
TubeBell . . . 156
Wurley . . . 166
Mesh2D . . . 69
Modal . . . 73
ModalBar . . . 75
Plucked . . . 92
PluckTwo . . . 94
Mandolin . . . 67
Resonate . . . 102
Sampler . . . 119
Moog . . . 79
Saxofony . . . 121
Shakers . . . 123
Simple . . . 126
Sitar . . . 128
StifKarp . . . 137
JetTabl . . . 65
Messager . . . 71
Modulate . . . 77
Noise . . . 81
SubNoise . . . 144
PitShift . . . 90
ReedTabl . . . 100
Reverb . . . 104
JCRev . . . 64
NRev . . . 83
PRCRev . . . 99
RtDuplex . . . 108
RtMidi . . . 111
SKINI . . . 130
Socket . . . 133
Table . . . 146
Thread . . . 154
WvIn . . . 168
RtWvIn . . . 113
TcpWvIn . . . 148
WaveLoop . . . 164
WvOut . . . 173

- RtWvOut 116
- TcpWvOut 151

StkError 143

Chapter 2

STK Compound Index

2.1 STK Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

ADSR (STK ADSR envelope class) 11
BandedWG (Banded waveguide modeling class) 14
BeeThree (STK Hammond-oid organ FM synthesis instrument) . . . 16
BiQuad (STK biquad (two-pole, two-zero) filter class) 18
BlowBotl (STK blown bottle instrument class) 21
BlowHole (STK clarinet physical model with one register hole and one tonehole) 23
Bowed (STK bowed string instrument class) 25
BowTabl (STK bowed string table class) 27
Brass (STK simple brass instrument class) 29
Chorus (STK chorus effect class) 31
Clarinet (STK clarinet physical model class) 33
Delay (STK non-interpolating delay line class) 35
DelayA (STK allpass interpolating delay line class) 37
DelayL (STK linear interpolating delay line class) 39
Drummer (STK drum sample player class) 41
Echo (STK echo effect class) 43
Envelope (STK envelope base class) 45
Filter (STK filter class) 47
Flute (STK flute physical model class) 50
FM (STK abstract FM synthesis base class) 52
FMVoices (STK singing FM synthesis instrument) 55
FormSwep (STK sweepable formant filter class) 57
HevyMetl (STK heavy metal FM synthesis instrument) 60
Instrmnt (STK instrument abstract base class) 62

JCRev (John Chowning's reverberator class) 64
JetTabl (STK jet table class) 65
Mandolin (STK mandolin instrument model class) 67
Mesh2D (Two-dimensional rectilinear waveguide mesh class) 69
Messager (STK input control message parser) 71
Modal (STK resonance model instrument) 73
ModalBar (STK resonant bar instrument class) 75
Modulate (STK periodic/random modulator) 77
Moog (STK moog-like swept filter sampling synthesis class) 79
Noise (STK noise generator) 81
NRev (CCRMA's NRev reverberator class) 83
OnePole (STK one-pole filter class) 84
OneZero (STK one-zero filter class) 86
PercFlut (STK percussive flute FM synthesis instrument) 88
PitShift (STK simple pitch shifter effect class) 90
Plucked (STK plucked string model class) 92
PluckTwo (STK enhanced plucked string model class) 94
PoleZero (STK one-pole, one-zero filter class) 96
PRCRev (Perry's simple reverberator class) 99
ReedTabl (STK reed table class) 100
Resonate (STK noise driven formant filter) 102
Reverb (STK abstract reverberator parent class) 104
Rhodey (STK Fender Rhodes electric piano FM synthesis instrument) 106
RtDuplex (STK realtime audio input/output class) 108
RtMidi (STK realtime MIDI class) 111
RtWvIn (STK realtime audio input class) 113
RtWvOut (STK realtime audio output class) 116
Sampler (STK sampling synthesis abstract base class) 119
Saxofony (STK faux conical bore reed instrument class) 121
Shakers (PhISEM and PhOLIES class) 123
Simple (STK wavetable/noise instrument) 126
Sitar (STK sitar string model class) 128
SKINI (STK SKINI parsing class) 130
Socket (STK TCP socket client/server class) 133
StifKarp (STK plucked stiff string instrument) 137
Stk (STK base class) 139
StkError (STK error handling class) 143
SubNoise (STK sub-sampled noise generator) 144
Table (STK table lookup class) 146
TcpWvIn (STK internet streaming input class) 148
TcpWvOut (STK internet streaming output class) 151
Thread (STK thread class) 154
TubeBell (STK tubular bell (orchestral chime) FM synthesis instrument) 156
TwoPole (STK two-pole filter class) 158
TwoZero (STK two-zero filter class) 161

WaveLoop (STK waveform oscillator class) 164
Wurley (STK Wurlitzer electric piano FM synthesis instrument) . . . 166
WvIn (STK audio data input base class) 168
WvOut (STK audio data output base class) 173

Chapter 3

STK Page Index

3.1 STK Related Pages

Here is a list of all related documentation pages:

General Information 177
Class Documentation 180
Download and Release Notes 181
Usage Documentation 185
The Mail List 191
System Requirements 192
Tutorial 193
Synthesis toolKit Instrument Network Interface (SKINI) 198

Chapter 4

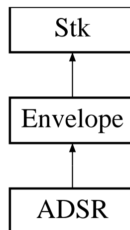
STK Class Documentation

4.1 ADSR Class Reference

STK ADSR envelope class.

`#include <ADSR.h>`

Inheritance diagram for ADSR::



Public Types

- enum { **ATTACK**, **DECAY**, **SUSTAIN**, **RELEASE**, **DONE** }
  Envelope states.

Public Methods

- ADSR (void)
  Default constructor.

- ~ADSR (void)

  Class destructor.

- void keyOn (void)

  Set target = 1, state = ADSR::ATTACK.

- void keyOff (void)

  Set target = 0, state = ADSR::RELEASE.

- void setAttackRate (MY_FLOAT aRate)

  Set the attack rate.

- void setDecayRate (MY_FLOAT aRate)

  Set the decay rate.

- void setSustainLevel (MY_FLOAT aLevel)

  Set the sustain level.

- void setReleaseRate (MY_FLOAT aRate)

  Set the release rate.

- void setAttackTime (MY_FLOAT aTime)

  Set the attack rate based on a time duration.

- void setDecayTime (MY_FLOAT aTime)

  Set the decay rate based on a time duration.

- void setReleaseTime (MY_FLOAT aTime)

  Set the release rate based on a time duration.

- void setAllTimes (MY_FLOAT aTime, MY_FLOAT dTime, MY_FLOAT sLevel, MY_FLOAT rTime)

  Set sustain level and attack, decay, and release state rates based on time durations.

- void setTarget (MY_FLOAT aTarget)

  Set the target value.

- int getState (void) const

  Return the current envelope state *(ATTACK, DECAY, SUSTAIN, RELEASE, DONE).*

- void setValue (MY_FLOAT aValue)

Set to state = ADSR::SUSTAIN with current and target values of aValue*.*

- MY_FLOAT tick (void)

  Return one envelope output value.

- MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Return vectorSize *envelope outputs in* vector*.*

4.1.1 Detailed Description

STK ADSR envelope class.

This Envelope subclass implements a traditional ADSR (Attack, Decay, Sustain, Release) envelope. It responds to simple keyOn and keyOff messages, keeping track of its state. The *state* = ADSR::DONE after the envelope value reaches 0.0 in the ADSR::RELEASE state.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

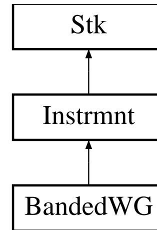
- ADSR.h

4.2 BandedWG Class Reference

Banded waveguide modeling class.

`#include <BandedWG.h>`

Inheritance diagram for BandedWG::



Public Methods

- BandedWG ()

  Class constructor.

- ~BandedWG ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setStrikePosition (MY_FLOAT position)

  Set strike position (0.0 - 1.0).

- void setPreset (int preset)

  Select a preset.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void startBowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply bow velocity/pressure to instrument with given amplitude and rate of increase.

- void stopBowing (MY_FLOAT rate)

  Decrease bow velocity/breath pressure with given rate of decrease.

- void pluck (MY_FLOAT amp)
  Pluck the instrument with given amplitude.
- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.
- void noteOff (MY_FLOAT amplitude)
  Stop a note with the given amplitude (speed of decay).
- MY_FLOAT tick ()
  Compute one output sample.
- void controlChange (int number, MY_FLOAT value)
  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.2.1 Detailed Description

Banded waveguide modeling class.

This class uses banded waveguide techniques to model a variety of sounds, including bowed bars, glasses, and bowls. For more information, see Essl, G. and Cook, P. "Banded Waveguides: Towards Physical Modelling of Bar Percussion Instruments", Proceedings of the 1999 International Computer Music Conference.

Control Change Numbers:

- Bow Pressure = 2
- Bow Motion = 4
- Strike Position = 8 (not implemented)
- Vibrato Frequency = 11
- Gain = 1
- Bow Velocity = 128
- Set Striking = 64
- Instrument Presets = 16
  - Uniform Bar = 0
  - Tuned Bar = 1
  - Glass Harmonica = 2
  - Tibetan Bowl = 3

by Georg Essl, 1999 - 2002. Modified for Stk 4.0 by Gary Scavone.

The documentation for this class was generated from the following file:

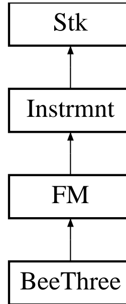
- BandedWG.h

4.3 BeeThree Class Reference

STK Hammond-oid organ FM synthesis instrument.

`#include <BeeThree.h>`

Inheritance diagram for BeeThree::



Public Methods

- BeeThree ()

  Class constructor.

- ~BeeThree ()

  Class destructor.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- MY_FLOAT tick ()

  Compute one output sample.

4.3.1 Detailed Description

STK Hammond-oid organ FM synthesis instrument.

This class implements a simple 4 operator topology, also referred to as algorithm 8 of the TX81Z.

```
Algorithm 8 is :
                 1 --.
                 2 -\|
                     +-> Out
```

```
3 -/|
4 --
```

Control Change Numbers:

- Operator 4 (feedback) Gain = 2
- Operator 3 Gain = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

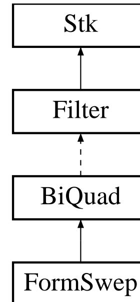
- BeeThree.h

4.4 BiQuad Class Reference

STK biquad (two-pole, two-zero) filter class.

`#include <BiQuad.h>`

Inheritance diagram for BiQuad::



Public Methods

- BiQuad ()

  Default constructor creates a second-order pass-through filter.

- virtual ~BiQuad ()

  Class destructor.

- void clear (void)

  Clears all internal states of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setB1 (MY_FLOAT b1)

  Set the b[1] coefficient value.

- void setB2 (MY_FLOAT b2)

  Set the b[2] coefficient value.

- void setA1 (MY_FLOAT a1)

  Set the a[1] coefficient value.

- void setA2 (MY_FLOAT a2)

Set the a[2] coefficient value.

- void setResonance (MY_FLOAT frequency, MY_FLOAT radius, bool normalize=FALSE)

  Sets the filter coefficients for a resonance at frequency *(in Hz).*

- void setNotch (MY_FLOAT frequency, MY_FLOAT radius)

  Set the filter coefficients for a notch at frequency *(in Hz).*

- void setEqualGainZeroes ()

  Sets the filter zeroes for equal resonance gain.

- void setGain (MY_FLOAT theGain)

  Set the filter gain.

- MY_FLOAT getGain (void) const

  Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.4.1 Detailed Description

STK biquad (two-pole, two-zero) filter class.

This protected Filter subclass implements a two-pole, two-zero digital filter. A method is provided for creating a resonance in the frequency response while maintaining a constant filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.4.2 Member Function Documentation

4.4.2.1 void BiQuad::setResonance (MY_FLOAT *frequency*, MY_FLOAT *radius*, bool *normalize* = FALSE)

Sets the filter coefficients for a resonance at *frequency* (in Hz).

This method determines the filter coefficients corresponding to two complex-conjugate poles with the given *frequency* (in Hz) and *radius* from the z-plane origin. If *normalize* is true, the filter zeros are placed at z = 1, z = -1, and the coefficients are then normalized to produce a constant unity peak gain (independent of the filter *gain* parameter). The resulting filter frequency response has a resonance at the given *frequency*. The closer the poles are to the unit-circle (*radius* close to one), the narrower the resulting resonance width.

4.4.2.2 void BiQuad::setNotch (MY_FLOAT *frequency*, MY_FLOAT *radius*)

Set the filter coefficients for a notch at *frequency* (in Hz).

This method determines the filter coefficients corresponding to two complex-conjugate zeros with the given *frequency* (in Hz) and *radius* from the z-plane origin. No filter normalization is attempted.

4.4.2.3 void BiQuad::setEqualGainZeroes ()

Sets the filter zeroes for equal resonance gain.

When using the filter as a resonator, zeroes places at z = 1, z = -1 will result in a constant gain at resonance of 1 / (1 - R), where R is the pole radius setting.

4.4.2.4 void BiQuad::setGain (MY_FLOAT *theGain*) `[virtual]`

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

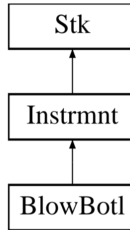
- BiQuad.h

4.5 BlowBotl Class Reference

STK blown bottle instrument class.

`#include <BlowBotl.h>`

Inheritance diagram for BlowBotl::



Public Methods

- BlowBotl ()
  Class constructor.
- ~BlowBotl ()
  Class destructor.
- void clear ()
  Reset and clear all internal state.
- void setFrequency (MY_FLOAT frequency)
  Set instrument parameters for a particular frequency.
- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)
  Apply breath velocity to instrument with given amplitude and rate of increase.
- void stopBlowing (MY_FLOAT rate)
  Decrease breath velocity with given rate of decrease.
- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.
- void noteOff (MY_FLOAT amplitude)
  Stop a note with the given amplitude (speed of decay).
- MY_FLOAT tick ()

Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.5.1 Detailed Description

STK blown bottle instrument class.

This class implements a helmholtz resonator (biquad filter) with a polynomial jet excitation (a la Cook).

Control Change Numbers:

- Noise Gain = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Volume = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

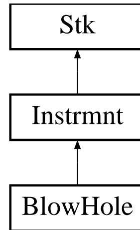
- BlowBotl.h

4.6 BlowHole Class Reference

STK clarinet physical model with one register hole and one tonehole.

`#include <BlowHole.h>`

Inheritance diagram for BlowHole::



Public Methods

- BlowHole (MY_FLOAT lowestFrequency)

  Class constructor.

- ~BlowHole ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setTonehole (MY_FLOAT newValue)

  Set the tonehole state (0.0 = closed, 1.0 = fully open).

- void setVent (MY_FLOAT newValue)

  Set the register hole state (0.0 = closed, 1.0 = fully open).

- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply breath pressure to instrument with given amplitude and rate of increase.

- void stopBlowing (MY_FLOAT rate)

  Decrease breath pressure with given rate of decrease.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.6.1 Detailed Description

STK clarinet physical model with one register hole and one tonehole.

This class is based on the clarinet model, with the addition of a two-port register hole and a three-port dynamic tonehole implementation, as discussed by Scavone and Cook (1998).

In this implementation, the distances between the reed/register hole and tonehole/bell are fixed. As a result, both the tonehole and register hole will have variable influence on the playing frequency, which is dependent on the length of the air column. In addition, the highest playing freqeuency is limited by these fixed lengths.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Reed Stiffness = 2
- Noise Gain = 4
- Tonehole State = 11
- Register State = 1
- Breath Pressure = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

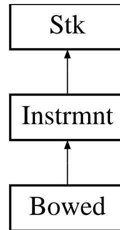
- BlowHole.h

4.7 Bowed Class Reference

STK bowed string instrument class.

`#include <Bowed.h>`

Inheritance diagram for Bowed::



Public Methods

- Bowed (MY_FLOAT lowestFrequency)
  Class constructor, taking the lowest desired playing frequency.
- ∼Bowed ()
  Class destructor.
- void clear ()
  Reset and clear all internal state.
- void setFrequency (MY_FLOAT frequency)
  Set instrument parameters for a particular frequency.
- void setVibrato (MY_FLOAT gain)
  Set vibrato gain.
- void startBowing (MY_FLOAT amplitude, MY_FLOAT rate)
  Apply breath pressure to instrument with given amplitude and rate of increase.
- void stopBowing (MY_FLOAT rate)
  Decrease breath pressure with given rate of decrease.
- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.7.1 Detailed Description

STK bowed string instrument class.

This class implements a bowed string model, a la Smith (1986), after McIntyre, Schumacher, Woodhouse (1983).

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Bow Pressure = 2
- Bow Position = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Volume = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

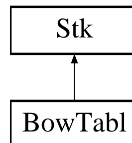
- Bowed.h

4.8 BowTabl Class Reference

STK bowed string table class.

`#include <BowTabl.h>`

Inheritance diagram for BowTabl::



Public Methods

- BowTabl ()

  Default constructor.

- ~BowTabl ()

  Class destructor.

- void setOffset (MY_FLOAT aValue)

  Set the table offset value.

- void setSlope (MY_FLOAT aValue)

  Set the table slope value.

- MY_FLOAT lastOut (void) const

  Return the last output value.

- MY_FLOAT tick (const MY_FLOAT input)

  Return the function value for input.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Take vectorSize *inputs and return the corresponding function values in* vector.

4.8.1 Detailed Description

STK bowed string table class.

This class implements a simple bowed string non-linear function, as described by Smith (1986).

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.8.2 Member Function Documentation

4.8.2.1 void BowTabl::setOffset (MY_FLOAT *aValue*)

Set the table offset value.

The table offset is a bias which controls the symmetry of the friction. If you want the friction to vary with direction, use a non-zero value for the offset. The default value is zero.

4.8.2.2 void BowTabl::setSlope (MY_FLOAT *aValue*)

Set the table slope value.

The table slope controls the width of the friction pulse, which is related to bow force.

4.8.2.3 MY_FLOAT BowTabl::tick (const MY_FLOAT *input*)

Return the function value for *input*.

The function input represents differential string-to-bow velocity.

The documentation for this class was generated from the following file:

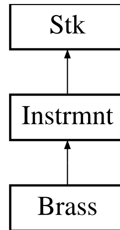
- BowTabl.h

4.9 Brass Class Reference

STK simple brass instrument class.

`#include <Brass.h>`

Inheritance diagram for Brass::



Public Methods

- Brass (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~Brass ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setLip (MY_FLOAT frequency)

  Set the lips frequency.

- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply breath pressure to instrument with given amplitude and rate of increase.

- void stopBlowing (MY_FLOAT rate)

  Decrease breath pressure with given rate of decrease.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.9.1 Detailed Description

STK simple brass instrument class.

This class implements a simple brass instrument waveguide model, a la Cook (TBone, HosePlayer).

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Lip Tension = 2
- Slide Length = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Volume = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

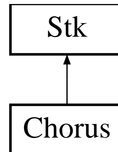
- Brass.h

4.10 Chorus Class Reference

STK chorus effect class.

`#include <Chorus.h>`

Inheritance diagram for Chorus::



Public Methods

- Chorus (MY_FLOAT baseDelay)

  Class constructor, taking the longest desired delay length.

- ~Chorus ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setModDepth (MY_FLOAT depth)

  Set modulation depth.

- void setModFrequency (MY_FLOAT frequency)

  Set modulation frequency.

- void setEffectMix (MY_FLOAT mix)

  Set the mixture of input and processed levels in the output (0.0 = input only, 1.0 = processed only).

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT lastOutLeft () const

  Return the last left output value.

- MY_FLOAT lastOutRight () const

  Return the last right output value.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Take vectorSize *inputs, compute the same number of outputs and return them in* vector*.*

4.10.1 Detailed Description

STK chorus effect class.

This class implements a chorus effect.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

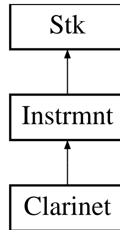
- Chorus.h

4.11 Clarinet Class Reference

STK clarinet physical model class.

`#include <Clarinet.h>`

Inheritance diagram for Clarinet::



Public Methods

- Clarinet (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ∼Clarinet ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply breath pressure to instrument with given amplitude and rate of increase.

- void stopBlowing (MY_FLOAT rate)

  Decrease breath pressure with given rate of decrease.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.11.1 Detailed Description

STK clarinet physical model class.

This class implements a simple clarinet physical model, as discussed by Smith (1986), McIntyre, Schumacher, Woodhouse (1983), and others.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Reed Stiffness = 2
- Noise Gain = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Breath Pressure = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

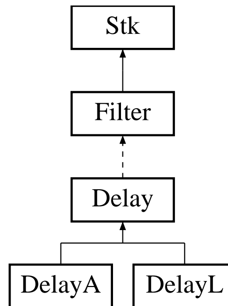
- Clarinet.h

4.12 Delay Class Reference

STK non-interpolating delay line class.

`#include <Delay.h>`

Inheritance diagram for Delay::



Public Methods

- Delay ()

  Default constructor creates a delay-line with maximum length of 4095 samples and zero delay.

- Delay (long theDelay, long maxDelay)

  Overloaded constructor which specifies the current and maximum delay-line lengths.

- virtual ~Delay ()

  Class destructor.

- void clear ()

  Clears the internal state of the delay line.

- void setDelay (long theDelay)

  Set the delay-line length.

- long getDelay (void) const

  Return the current delay-line length.

- MY_FLOAT energy (void) const

  Calculate and return the signal energy in the delay-line.

- MY_FLOAT contentsAt (long tapDelay) const

  Return the value at tapDelay *samples from the delay-line input.*

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- virtual MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the delay-line and return one output.

- virtual MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Input vectorSize *samples to the delay-line and return an equal number of outputs in* vector*.*

4.12.1 Detailed Description

STK non-interpolating delay line class.

This protected Filter subclass implements a non-interpolating digital delay-line. A fixed maximum length of 4095 and a delay of zero is set using the default constructor. Alternatively, the delay and maximum length can be set during instantiation with an overloaded constructor.

A non-interpolating delay line is typically used in fixed delay-length applications, such as for reverberation.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.12.2 Member Function Documentation

4.12.2.1 void Delay::setDelay (long *theDelay*)

Set the delay-line length.

The valid range for *theDelay* is from 0 to the maximum delay-line length.

4.12.2.2 MY_FLOAT Delay::contentsAt (long *tapDelay*) const

Return the value at *tapDelay* samples from the delay-line input.

The valid range for *tapDelay* is 1 to the delay-line length.

The documentation for this class was generated from the following file:

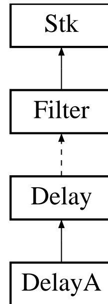
- Delay.h

4.13 DelayA Class Reference

STK allpass interpolating delay line class.

`#include <DelayA.h>`

Inheritance diagram for DelayA::



Public Methods

- DelayA ()

  Default constructor creates a delay-line with maximum length of 4095 samples and zero delay.

- DelayA (MY_FLOAT theDelay, long maxDelay)

  Overloaded constructor which specifies the current and maximum delay-line lengths.

- ~DelayA ()

  Class destructor.

- void clear ()

  Clears the internal state of the delay line.

- void setDelay (MY_FLOAT theDelay)

  Set the delay-line length.

- MY_FLOAT getDelay (void)

  Return the current delay-line length.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the delay-line and return one output.

4.13.1 Detailed Description

STK allpass interpolating delay line class.

This Delay subclass implements a fractional- length digital delay-line using a first-order allpass filter. A fixed maximum length of 4095 and a delay of 0.5 is set using the default constructor. Alternatively, the delay and maximum length can be set during instantiation with an overloaded constructor.

An allpass filter has unity magnitude gain but variable phase delay properties, making it useful in achieving fractional delays without affecting a signal's frequency magnitude response. In order to achieve a maximally flat phase delay response, the minimum delay possible in this implementation is limited to a value of 0.5.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.13.2 Member Function Documentation

4.13.2.1 void DelayA::setDelay (MY_FLOAT *theDelay*)

Set the delay-line length.

The valid range for *theDelay* is from 0.5 to the maximum delay-line length.

The documentation for this class was generated from the following file:

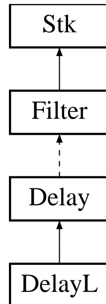
- DelayA.h

4.14 DelayL Class Reference

STK linear interpolating delay line class.

`#include <DelayL.h>`

Inheritance diagram for DelayL::



Public Methods

- DelayL ()

  Default constructor creates a delay-line with maximum length of 4095 samples and zero delay.

- DelayL (MY_FLOAT theDelay, long maxDelay)

  Overloaded constructor which specifies the current and maximum delay-line lengths.

- ~DelayL ()

  Class destructor.

- void setDelay (MY_FLOAT theDelay)

  Set the delay-line length.

- MY_FLOAT getDelay (void) const

  Return the current delay-line length.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the delay-line and return one output.

4.14.1 Detailed Description

STK linear interpolating delay line class.

This Delay subclass implements a fractional- length digital delay-line using first-order linear interpolation. A fixed maximum length of 4095 and a delay of zero is set using the default constructor. Alternatively, the delay and maximum length can be set during instantiation with an overloaded constructor.

Linear interpolation is an efficient technique for achieving fractional delay lengths, though it does introduce high-frequency signal attenuation to varying degrees depending on the fractional delay setting. The use of higher order Lagrange interpolators can typically improve (minimize) this attenuation characteristic.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.14.2 Member Function Documentation

4.14.2.1 void DelayL::setDelay (MY_FLOAT *theDelay*)

Set the delay-line length.

The valid range for *theDelay* is from 0 to the maximum delay-line length.

The documentation for this class was generated from the following file:

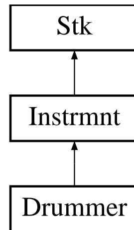
- DelayL.h

4.15 Drummer Class Reference

STK drum sample player class.

`#include <Drummer.h>`

Inheritance diagram for Drummer::



Public Methods

- Drummer ()
  Class constructor.
- ~Drummer ()
  Class destructor.
- void noteOn (MY_FLOAT instrument, MY_FLOAT amplitude)
  Start a note with the given drum type and amplitude.
- void noteOff (MY_FLOAT amplitude)
  Stop a note with the given amplitude (speed of decay).
- MY_FLOAT tick ()
  Compute one output sample.

4.15.1 Detailed Description

STK drum sample player class.

This class implements a drum sampling synthesizer using WvIn objects and one-pole filters. The drum rawwave files are sampled at 22050 Hz, but will be appropriately interpolated for other sample rates. You can specify the maximum polyphony (maximum number of simultaneous voices) via a define in the Drummer.h.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.15.2 Member Function Documentation

4.15.2.1 void Drummer::noteOn (MY_FLOAT *instrument*, MY_FLOAT *amplitude*) [virtual]

Start a note with the given drum type and amplitude.

Use general MIDI drum instrument numbers, converted to frequency values as if MIDI note numbers, to select a particular instrument.

Reimplemented from Instrmnt.

The documentation for this class was generated from the following file:

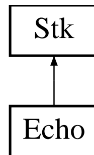
- Drummer.h

4.16 Echo Class Reference

STK echo effect class.

`#include <Echo.h>`

Inheritance diagram for Echo::



Public Methods

- Echo (MY_FLOAT longestDelay)

  Class constructor, taking the longest desired delay length.

- ~Echo ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setDelay (MY_FLOAT delay)

  Set the delay line length in samples.

- void setEffectMix (MY_FLOAT mix)

  Set the mixture of input and processed levels in the output (0.0 = input only, 1.0 = processed only).

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

- MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.16.1 Detailed Description

STK echo effect class.

This class implements a echo effect.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

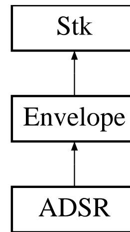
- Echo.h

4.17 Envelope Class Reference

STK envelope base class.

`#include <Envelope.h>`

Inheritance diagram for Envelope::



Public Methods

- Envelope (void)

  Default constructor.

- virtual ~Envelope (void)

  Class destructor.

- virtual void keyOn (void)

  Set target = 1.

- virtual void keyOff (void)

  Set target = 0.

- void setRate (MY_FLOAT aRate)

  Set the rate.

- void setTime (MY_FLOAT aTime)

  Set the rate *based on a time duration.*

- virtual void setTarget (MY_FLOAT aTarget)

  Set the target value.

- virtual void setValue (MY_FLOAT aValue)

  Set current and target values to aValue.

- virtual int getState (void) const

Return the current envelope state *(0 = at target, 1 otherwise).*

- virtual MY_FLOAT tick (void)

  Return one envelope output value.

- virtual MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Return vectorSize *envelope outputs in* vector*.*

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

4.17.1 Detailed Description

STK envelope base class.

This class implements a simple envelope generator which is capable of ramping to a target value by a specified *rate*. It also responds to simple *keyOn* and *keyOff* messages, ramping to 1.0 on keyOn and to 0.0 on keyOff.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

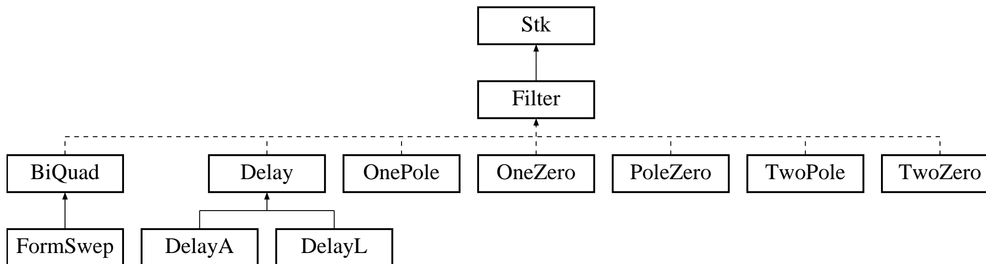
- Envelope.h

4.18 Filter Class Reference

STK filter class.

`#include <Filter.h>`

Inheritance diagram for Filter::



Public Methods

- Filter (void)

  Default constructor creates a zero-order pass-through "filter".

- Filter (int nb, MY_FLOAT *bCoefficients, int na, MY_FLOAT *aCoefficients)

  Overloaded constructor which takes filter coefficients.

- virtual ~Filter (void)

  Class destructor.

- void clear (void)

  Clears all internal states of the filter.

- void setCoefficients (int nb, MY_FLOAT *bCoefficients, int na, MY_FLOAT *aCoefficients)

  Set filter coefficients.

- void setNumerator (int nb, MY_FLOAT *bCoefficients)

  Set numerator coefficients.

- void setDenominator (int na, MY_FLOAT *aCoefficients)

  Set denominator coefficients.

- virtual void setGain (MY_FLOAT theGain)

Set the filter gain.

- virtual MY_FLOAT getGain (void) const

  Return the current filter gain.

- virtual MY_FLOAT lastOut (void) const

  Return the last computed output value.

- virtual MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- virtual MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector*.*

4.18.1 Detailed Description

STK filter class.

This class implements a generic structure which can be used to create a wide range of filters. It can function independently or be subclassed to provide more specific controls based on a particular filter type.

In particular, this class implements the standard difference equation:

a[0]*y[n] = b[0]*x[n] + ... + b[nb]*x[n-nb] - a[1]*y[n-1] - ... - a[na]*y[n-na]

If a[0] is not equal to 1, the filter coeffcients are normalized by a[0].

The *gain* parameter is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0. This structure results in one extra multiply per computed sample, but allows easy control of the overall filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Filter::Filter (int *nb*, MY_FLOAT * *bCoefficients*, int *na*, MY_FLOAT * *aCoefficients*)

Overloaded constructor which takes filter coefficients.

An StkError can be thrown if either *nb* or *na* is less than one, or if the a[0] coefficient is equal to zero.

4.18.3 Member Function Documentation

4.18.3.1 void Filter::setCoefficients (int *nb*, MY_FLOAT * *bCoefficients*, int *na*, MY_FLOAT * *aCoefficients*)

Set filter coefficients.

An StkError can be thrown if either *nb* or *na* is less than one, or if the a[0] coefficient is equal to zero. If a[0] is not equal to 1, the filter coeffcients are normalized by a[0].

4.18.3.2 void Filter::setNumerator (int *nb*, MY_FLOAT * *bCoefficients*)

Set numerator coefficients.

An StkError can be thrown if *nb* is less than one. Any previously set denominator coefficients are left unaffected. Note that the default constructor sets the single denominator coefficient a[0] to 1.0.

4.18.3.3 void Filter::setDenominator (int *na*, MY_FLOAT * *aCoefficients*)

Set denominator coefficients.

An StkError can be thrown if *na* is less than one or if the a[0] coefficient is equal to zero. Previously set numerator coefficients are unaffected unless a[0] is not equal to 1, in which case all coeffcients are normalized by a[0]. Note that the default constructor sets the single numerator coefficient b[0] to 1.0.

4.18.3.4 void Filter::setGain (MY_FLOAT *theGain*) `[virtual]`

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented in BiQuad, OnePole, OneZero, PoleZero, TwoPole, and Two-Zero.

The documentation for this class was generated from the following file:

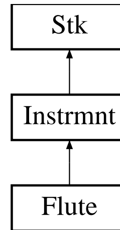
- Filter.h

4.19 Flute Class Reference

STK flute physical model class.

`#include <Flute.h>`

Inheritance diagram for Flute::



Public Methods

- Flute (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~Flute ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setJetReflection (MY_FLOAT coefficient)

  Set the reflection coefficient for the jet delay (-1.0 - 1.0).

- void setEndReflection (MY_FLOAT coefficient)

  Set the reflection coefficient for the air column delay (-1.0 - 1.0).

- void setJetDelay (MY_FLOAT aRatio)

  Set the length of the jet delay in terms of a ratio of jet delay to air column delay lengths.

- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply breath velocity to instrument with given amplitude and rate of increase.

- void stopBlowing (MY_FLOAT rate)

  Decrease breath velocity with given rate of decrease.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.19.1 Detailed Description

STK flute physical model class.

This class implements a simple flute physical model, as discussed by Karjalainen, Smith, Waryznyk, etc. The jet model uses a polynomial, a la Cook.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Jet Delay = 2
- Noise Gain = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Breath Pressure = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

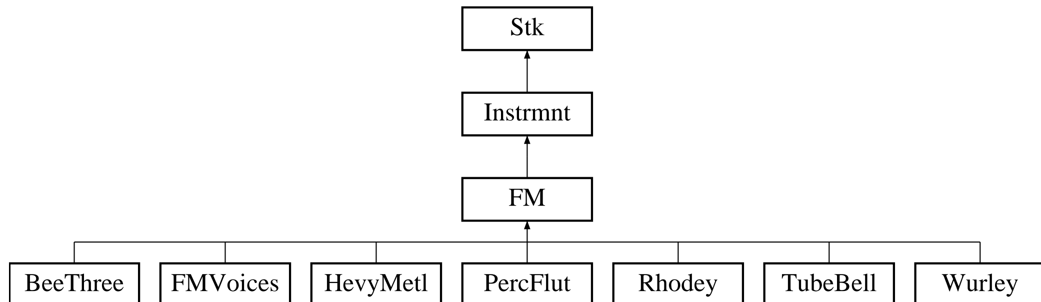
- Flute.h

4.20 FM Class Reference

STK abstract FM synthesis base class.

`#include <FM.h>`

Inheritance diagram for FM::



Public Methods

- FM (int operators=4)

  Class constructor, taking the number of wave/envelope operators to control.

- virtual ~FM ()

  Class destructor.

- void clear ()

  Reset and clear all wave and envelope states.

- void loadWaves (const char **filenames)

  Load the rawwave filenames in waves.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setRatio (int waveIndex, MY_FLOAT ratio)

  Set the frequency ratio for the specified wave.

- void setGain (int waveIndex, MY_FLOAT gain)

  Set the gain for the specified wave.

- void setModulationSpeed (MY_FLOAT mSpeed)

Set the modulation speed in Hz.

- void setModulationDepth (MY_FLOAT mDepth)

  Set the modulation depth.

- void setControl1 (MY_FLOAT cVal)

  Set the value of control1.

- void setControl2 (MY_FLOAT cVal)

  Set the value of control1.

- void keyOn ()

  Start envelopes toward "on" targets.

- void keyOff ()

  Start envelopes toward "off" targets.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()=0

  Pure virtual function ... must be defined in subclasses.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.20.1 Detailed Description

STK abstract FM synthesis base class.

This class controls an arbitrary number of waves and envelopes, determined via a constructor argument.

Control Change Numbers:

- Control One = 2
- Control Two = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

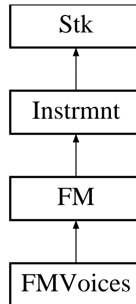
- FM.h

4.21 FMVoices Class Reference

STK singing FM synthesis instrument.

`#include <FMVoices.h>`

Inheritance diagram for FMVoices::



Public Methods

- FMVoices ()

  Class constructor.

- ~FMVoices ()

  Class destructor.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- MY_FLOAT tick ()

  Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.21.1 Detailed Description

STK singing FM synthesis instrument.

This class implements 3 carriers and a common modulator, also referred to as algorithm 6 of the TX81Z.

```
Algorithm 6 is :
                    /->1 -\
                 4-|-->2 - +-> Out
                    \->3 -/
```

Control Change Numbers:

- Vowel = 2
- Spectral Tilt = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

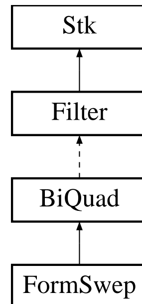
- FMVoices.h

4.22 FormSwep Class Reference

STK sweepable formant filter class.

`#include <FormSwep.h>`

Inheritance diagram for FormSwep::



Public Methods

- FormSwep ()

  Default constructor creates a second-order pass-through filter.

- ~FormSwep ()

  Class destructor.

- void setResonance (MY_FLOAT aFrequency, MY_FLOAT aRadius)

  Sets the filter coefficients for a resonance at frequency *(in Hz).*

- void setStates (MY_FLOAT aFrequency, MY_FLOAT aRadius, MY_FLOAT aGain=1.0)

  Set both the current and target resonance parameters.

- void setTargets (MY_FLOAT aFrequency, MY_FLOAT aRadius, MY_FLOAT aGain=1.0)

  Set target resonance parameters.

- void setSweepRate (MY_FLOAT aRate)

  Set the sweep rate (between 0.0 - 1.0).

- void setSweepTime (MY_FLOAT aTime)

  Set the sweep rate in terms of a time value in seconds.

- MY_FLOAT tick (MY_FLOAT sample)
  Input one sample to the filter and return one output.
- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)
  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.22.1 Detailed Description

STK sweepable formant filter class.

This public BiQuad filter subclass implements a formant (resonance) which can be "swept" over time from one frequency setting to another. It provides methods for controlling the sweep rate and target frequency.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.22.2 Member Function Documentation

4.22.2.1 void FormSwep::setResonance (MY_FLOAT *aFrequency*, MY_FLOAT *aRadius*)

Sets the filter coefficients for a resonance at *frequency* (in Hz).

This method determines the filter coefficients corresponding to two complex-conjugate poles with the given *frequency* (in Hz) and *radius* from the z-plane origin. The filter zeros are placed at z = 1, z = -1, and the coefficients are then normalized to produce a constant unity gain (independent of the filter *gain* parameter). The resulting filter frequency response has a resonance at the given *frequency*. The closer the poles are to the unit-circle (*radius* close to one), the narrower the resulting resonance width.

4.22.2.2 void FormSwep::setSweepRate (MY_FLOAT *aRate*)

Set the sweep rate (between 0.0 - 1.0).

The formant parameters are varied in increments of the sweep rate between their current and target values. A sweep rate of 1.0 will produce an immediate change in resonance parameters from their current values to the target values. A sweep rate of 0.0 will produce no change in resonance parameters.

4.22.2.3 void FormSwep::setSweepTime (MY_FLOAT *aTime*)

Set the sweep rate in terms of a time value in seconds.

This method adjusts the sweep rate based on a given time for the formant parameters to reach their target values.

The documentation for this class was generated from the following file:

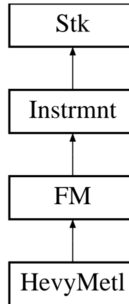
- FormSwep.h

4.23 HevyMetl Class Reference

STK heavy metal FM synthesis instrument.

`#include <HevyMetl.h>`

Inheritance diagram for HevyMetl::



Public Methods

- HevyMetl ()

  Class constructor.

- ~HevyMetl ()

  Class destructor.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- MY_FLOAT tick ()

  Compute one output sample.

4.23.1 Detailed Description

STK heavy metal FM synthesis instrument.

This class implements 3 cascade operators with feedback modulation, also referred to as algorithm 3 of the TX81Z.

```
Algorithm 3 is :    4--\
                3-->2-- + -->1-->Out
```

Control Change Numbers:

- Total Modulator Index = 2
- Modulator Crossfade = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

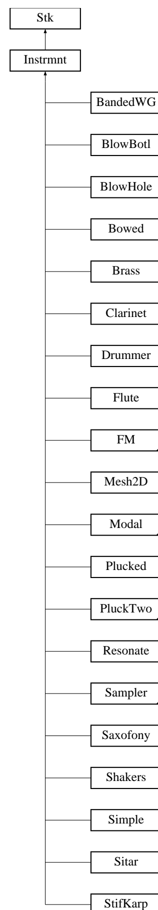
- HevyMetl.h

4.24 Instrmnt Class Reference

STK instrument abstract base class.

`#include <Instrmnt.h>`

Inheritance diagram for Instrmnt::



Public Methods

- Instrmnt ()

  Default constructor.

- virtual ~Instrmnt ()

  Class destructor.

- virtual void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)=0
  Start a note with the given frequency and amplitude.
- virtual void noteOff (MY_FLOAT amplitude)=0
  Stop a note with the given amplitude (speed of decay).
- virtual void setFrequency (MY_FLOAT frequency)
  Set instrument parameters for a particular frequency.
- MY_FLOAT lastOut () const
  Return the last output value.
- virtual MY_FLOAT tick ()=0
  Compute one output sample.
- virtual MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)
  Computer vectorSize *outputs and return them in* vector*.*
- virtual void controlChange (int number, MY_FLOAT value)
  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.24.1 Detailed Description

STK instrument abstract base class.

This class provides a common interface for all STK instruments.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

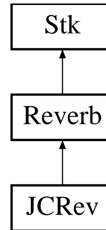
- Instrmnt.h

4.25 JCRev Class Reference

John Chowning's reverberator class.

`#include <JCRev.h>`

Inheritance diagram for JCRev::



Public Methods

- void clear ()

  Reset and clear all internal state.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

4.25.1 Detailed Description

John Chowning's reverberator class.

This class is derived from the CLM JCRev function, which is based on the use of networks of simple allpass and comb delay filters. This class implements three series allpass units, followed by four parallel comb filters, and two decorrelation delay lines in parallel at the output.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

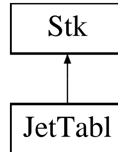
- JCRev.h

4.26 JetTabl Class Reference

STK jet table class.

`#include <JetTabl.h>`

Inheritance diagram for JetTabl::



Public Methods

- JetTabl ()
  Default constructor.
- ~JetTabl ()
  Class destructor.
- MY_FLOAT lastOut () const
  Return the last output value.
- MY_FLOAT tick (MY_FLOAT input)
  Return the function value for input*.*
- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)
  Take vectorSize *inputs and return the corresponding function values in* vector*.*

4.26.1 Detailed Description

STK jet table class.

This class implements a flue jet non-linear function, computed by a polynomial calculation. Contrary to the name, this is not a "table".

Consult Fletcher and Rossing, Karjalainen, Cook, and others for more information.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

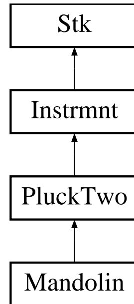
- JetTabl.h

4.27 Mandolin Class Reference

STK mandolin instrument model class.

`#include <Mandolin.h>`

Inheritance diagram for Mandolin::



Public Methods

- Mandolin (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- virtual ~Mandolin ()

  Class destructor.

- void pluck (MY_FLOAT amplitude)

  Pluck the strings with the given amplitude (0.0 - 1.0) using the current frequency.

- void pluck (MY_FLOAT amplitude, MY_FLOAT position)

  Pluck the strings with the given amplitude (0.0 - 1.0) and position (0.0 - 1.0).

- virtual void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude (0.0 - 1.0).

- void setBodySize (MY_FLOAT size)

  Set the body size (a value of 1.0 produces the "default" size).

- virtual MY_FLOAT tick ()

  Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.27.1 Detailed Description

STK mandolin instrument model class.

This class inherits from PluckTwo and uses "commuted synthesis" techniques to model a mandolin instrument.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others. Commuted Synthesis, in particular, is covered by patents, granted, pending, and/or applied-for. All are assigned to the Board of Trustees, Stanford University. For information, contact the Office of Technology Licensing, Stanford University.

Control Change Numbers:

- Body Size = 2
- Pluck Position = 4
- String Sustain = 11
- String Detuning = 1
- Microphone Position = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

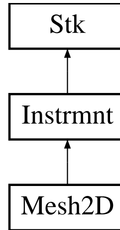
- Mandolin.h

4.28 Mesh2D Class Reference

Two-dimensional rectilinear waveguide mesh class.

`#include <Mesh2D.h>`

Inheritance diagram for Mesh2D::



Public Methods

- Mesh2D (short nX, short nY)

  Class constructor, taking the x and y dimensions in samples.

- ~Mesh2D ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setNX (short lenX)

  Set the x dimension size in samples.

- void setNY (short lenY)

  Set the y dimension size in samples.

- void setInputPosition (MY_FLOAT xFactor, MY_FLOAT yFactor)

  Set the x, y input position on a 0.0 - 1.0 scale.

- void setDecay (MY_FLOAT decayFactor)

  Set the loss filters gains (0.0 - 1.0).

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Impulse the mesh with the given amplitude (frequency ignored).

- void noteOff (MY_FLOAT amplitude)

Stop a note with the given amplitude (speed of decay) ... currently ignored.

- MY_FLOAT energy ()

  Calculate and return the signal energy stored in the mesh.

- MY_FLOAT tick ()

  Compute one output sample, without adding energy to the mesh.

- MY_FLOAT tick (MY_FLOAT input)

  Input a sample to the mesh and compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.28.1 Detailed Description

Two-dimensional rectilinear waveguide mesh class.

This class implements a rectilinear, two-dimensional digital waveguide mesh structure. For details, see Van Duyne and Smith, "Physical Modeling with the 2-D Digital Waveguide Mesh", Proceedings of the 1993 International Computer Music Conference.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- X Dimension = 2
- Y Dimension = 4
- Mesh Decay = 11
- X-Y Input Position = 1

by Julius Smith, 2000 - 2002. Revised by Gary Scavone for STK, 2002.

The documentation for this class was generated from the following file:

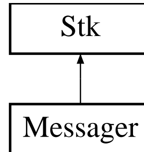
- Mesh2D.h

4.29 Messager Class Reference

STK input control message parser.

`#include <Messager.h>`

Inheritance diagram for Messager::



Public Methods

- Messager (int inputMask=0)

  Constructor performs initialization based on an input mask.

- ~Messager ()

  Class destructor.

- long nextMessage (void)

  Check for a new input message and return the message type.

- void setRtDelta (long nSamples)

  Set the delta time (in samples) returned between valid realtime messages. This setting has no affect for scorefile messages.

- long getDelta (void) const

  Return the current message "delta time" in samples.

- long getType () const

  Return the current message type.

- MY_FLOAT getByteTwo () const

  Return the byte two value for the current message.

- MY_FLOAT getByteThree () const

  Return the byte three value for the current message.

- long getChannel () const

  Return the channel number for the current message.

4.29.1 Detailed Description

STK input control message parser.

This class reads and parses control messages from a variety of sources, such as a MIDI port, scorefile, socket connection, or pipe. MIDI messages are retrieved using the RtMidi class. All other input sources (scorefile, socket, or pipe) are assumed to provide SKINI formatted messages.

For each call to nextMessage(), the active input sources are queried to see if a new control message is available.

This class is primarily for use in STK main() event loops.

One of the original goals in creating this class was to simplify the message acquisition process by removing all threads. If the windoze select() function behaved just like the unix one, that would have been possible. Since it does not (it can't be used to poll STDIN), I am using a thread to acquire messages from STDIN, which sends these messages via a socket connection to the message socket server. Perhaps in the future, it will be possible to simplify things.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 Messager::Messager (int *inputMask* = 0)

Constructor performs initialization based on an input mask.

The default constructor is set to read input from a SKINI scorefile. The flags STK_MIDI, STK_PIPE, and STK_SOCKET can be OR'ed together in any combination for multiple "realtime" input source parsing. For realtime input types, an StkError can be thrown during instantiation.

4.29.3 Member Function Documentation

4.29.3.1 long Messager::nextMessage (void)

Check for a new input message and return the message type.

Return type values greater than zero represent valid messages. If an input scorefile has been completely read or all realtime input sources have closed, a negative value is returned. If the return type is zero, no valid messages are present.

The documentation for this class was generated from the following file:

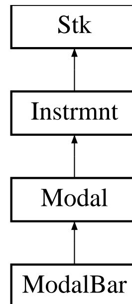
- Messager.h

4.30 Modal Class Reference

STK resonance model instrument.

`#include <Modal.h>`

Inheritance diagram for Modal::



Public Methods

- Modal (int modes=4)

  Class constructor, taking the desired number of modes to create.

- virtual ~Modal ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setRatioAndRadius (int modeIndex, MY_FLOAT ratio, MY_FLOAT radius)

  Set the ratio and radius for a specified mode filter.

- void setMasterGain (MY_FLOAT aGain)

  Set the master gain.

- void setDirectGain (MY_FLOAT aGain)

  Set the direct gain.

- void setModeGain (int modeIndex, MY_FLOAT gain)

Set the gain for a specified mode filter.

- virtual void strike (MY_FLOAT amplitude)

  Initiate a strike with the given amplitude (0.0 - 1.0).

- void damp (MY_FLOAT amplitude)

  Damp modes with a given decay factor (0.0 - 1.0).

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()

  Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)=0

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.30.1 Detailed Description

STK resonance model instrument.

This class contains an excitation wavetable, an envelope, an oscillator, and N resonances (non-sweeping BiQuad filters), where N is set during instantiation.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

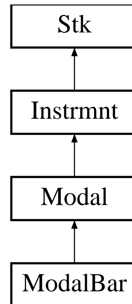
- Modal.h

4.31 ModalBar Class Reference

STK resonant bar instrument class.

`#include <ModalBar.h>`

Inheritance diagram for ModalBar::



Public Methods

- ModalBar ()

  Class constructor.

- ~ModalBar ()

  Class destructor.

- void setStickHardness (MY_FLOAT hardness)

  Set stick hardness (0.0 - 1.0).

- void setStrikePosition (MY_FLOAT position)

  Set stick position (0.0 - 1.0).

- void setPreset (int preset)

  Select a bar preset (currently modulo 9).

- void setModulationDepth (MY_FLOAT mDepth)

  Set the modulation (vibrato) depth.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.31.1 Detailed Description

STK resonant bar instrument class.

This class implements a number of different struck bar instruments. It inherits from the Modal class.

Control Change Numbers:

- Stick Hardness = 2
- Stick Position = 4
- Vibrato Gain = 11
- Vibrato Frequency = 7
- Volume = 128
- Modal Presets = 16
  - Marimba = 0
  - Vibraphone = 1
  - Agogo = 2
  - Wood1 = 3
  - Reso = 4
  - Wood2 = 5
  - Beats = 6
  - Two Fixed = 7
  - Clump = 8

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

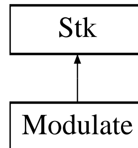
- ModalBar.h

4.32 Modulate Class Reference

STK periodic/random modulator.

`#include <Modulate.h>`

Inheritance diagram for Modulate::



Public Methods

- Modulate ()

  Class constructor.

- ~Modulate ()

  Class destructor.

- void reset ()

  Reset internal state.

- void setVibratoRate (MY_FLOAT aRate)

  Set the periodic (vibrato) rate or frequency in Hz.

- void setVibratoGain (MY_FLOAT aGain)

  Set the periodic (vibrato) gain.

- void setRandomGain (MY_FLOAT aGain)

  Set the random modulation gain.

- MY_FLOAT tick ()

  Compute one output sample.

- virtual MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Return vectorSize *outputs in* vector.

- MY_FLOAT lastOut () const

  Return the last computed output value.

4.32.1 Detailed Description

STK periodic/random modulator.

This class combines random and periodic modulations to give a nice, natural human modulation function.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

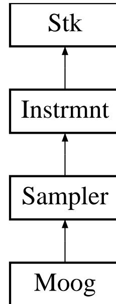
- Modulate.h

4.33 Moog Class Reference

STK moog-like swept filter sampling synthesis class.

`#include <Moog.h>`

Inheritance diagram for Moog::



Public Methods

- Moog ()
  Class constructor.
- ~Moog ()
  Class destructor.
- virtual void setFrequency (MY_FLOAT frequency)
  Set instrument parameters for a particular frequency.
- virtual void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.
- void setModulationSpeed (MY_FLOAT mSpeed)
  Set the modulation (vibrato) speed in Hz.
- void setModulationDepth (MY_FLOAT mDepth)
  Set the modulation (vibrato) depth.
- virtual MY_FLOAT tick ()
  Compute one output sample.
- virtual void controlChange (int number, MY_FLOAT value)
  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.33.1 Detailed Description

STK moog-like swept filter sampling synthesis class.

This instrument uses one attack wave, one looped wave, and an ADSR envelope (inherited from the Sampler class) and adds two sweepable formant (FormSwep) filters.

Control Change Numbers:

- Filter Q = 2
- Filter Sweep Rate = 4
- Vibrato Frequency = 11
- Vibrato Gain = 1
- Gain = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

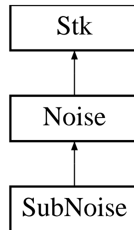
- Moog.h

4.34 Noise Class Reference

STK noise generator.

`#include <Noise.h>`

Inheritance diagram for Noise::



Public Methods

- Noise ()

  Default constructor.

- virtual ~Noise ()

  Class destructor.

- virtual MY_FLOAT tick ()

  Return a random number between -1.0 and 1.0 using rand().

- virtual MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Return vectorSize *random numbers between -1.0 and 1.0 in* vector*.*

- MY_FLOAT lastOut () const

  Return the last computed value.

4.34.1 Detailed Description

STK noise generator.

Generic random number generation using the C rand() function. The quality of the rand() function varies from one OS to another.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

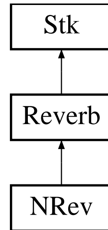
- Noise.h

4.35 NRev Class Reference

CCRMA's NRev reverberator class.

`#include <NRev.h>`

Inheritance diagram for NRev::



Public Methods

- void clear ()

  Reset and clear all internal state.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

4.35.1 Detailed Description

CCRMA's NRev reverberator class.

This class is derived from the CLM NRev function, which is based on the use of networks of simple allpass and comb delay filters. This particular arrangement consists of 6 comb filters in parallel, followed by 3 allpass filters, a lowpass filter, and another allpass in series, followed by two allpass filters in parallel with corresponding right and left outputs.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

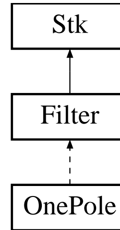
- NRev.h

4.36 OnePole Class Reference

STK one-pole filter class.

`#include <OnePole.h>`

Inheritance diagram for OnePole::



Public Methods

- OnePole ()

  Default constructor creates a first-order low-pass filter.

- OnePole (MY_FLOAT thePole)

  Overloaded constructor which sets the pole position during instantiation.

- ∼OnePole ()

  Class destructor.

- void clear (void)

  Clears the internal state of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setA1 (MY_FLOAT a1)

  Set the a[1] coefficient value.

- void setPole (MY_FLOAT thePole)

  Set the pole position in the z-plane.

- void setGain (MY_FLOAT theGain)

  Set the filter gain.

- MY_FLOAT getGain (void) const

Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector*.*

4.36.1 Detailed Description

STK one-pole filter class.

This protected Filter subclass implements a one-pole digital filter. A method is provided for setting the pole position along the real axis of the z-plane while maintaining a constant peak filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.36.2 Member Function Documentation

4.36.2.1 void OnePole::setPole (MY_FLOAT *thePole*)

Set the pole position in the z-plane.

This method sets the pole position along the real-axis of the z-plane and normalizes the coefficients for a maximum gain of one. A positive pole value produces a low-pass filter, while a negative pole value produces a high-pass filter. This method does not affect the filter *gain* value.

4.36.2.2 void OnePole::setGain (MY_FLOAT *theGain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

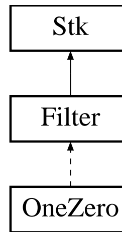
- OnePole.h

4.37 OneZero Class Reference

STK one-zero filter class.

`#include <OneZero.h>`

Inheritance diagram for OneZero::



Public Methods

- OneZero ()

  Default constructor creates a first-order low-pass filter.

- OneZero (MY_FLOAT theZero)

  Overloaded constructor which sets the zero position during instantiation.

- ~OneZero ()

  Class destructor.

- void clear (void)

  Clears the internal state of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setB1 (MY_FLOAT b1)

  Set the b[1] coefficient value.

- void setZero (MY_FLOAT theZero)

  Set the zero position in the z-plane.

- void setGain (MY_FLOAT theGain)

  Set the filter gain.

- MY_FLOAT getGain (void) const

Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.37.1 Detailed Description

STK one-zero filter class.

This protected Filter subclass implements a one-zero digital filter. A method is provided for setting the zero position along the real axis of the z-plane while maintaining a constant filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.37.2 Member Function Documentation

4.37.2.1 void OneZero::setZero (MY_FLOAT *theZero*)

Set the zero position in the z-plane.

This method sets the zero position along the real-axis of the z-plane and normalizes the coefficients for a maximum gain of one. A positive zero value produces a high-pass filter, while a negative zero value produces a low-pass filter. This method does not affect the filter *gain* value.

4.37.2.2 void OneZero::setGain (MY_FLOAT *theGain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

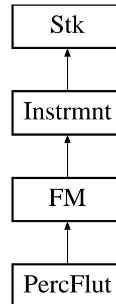
- OneZero.h

4.38 PercFlut Class Reference

STK percussive flute FM synthesis instrument.

`#include <PercFlut.h>`

Inheritance diagram for PercFlut::



Public Methods

- PercFlut ()
  Class constructor.
- ~PercFlut ()
  Class destructor.
- void setFrequency (MY_FLOAT frequency)
  Set instrument parameters for a particular frequency.
- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.
- MY_FLOAT tick ()
  Compute one output sample.

4.38.1 Detailed Description

STK percussive flute FM synthesis instrument.

This class implements algorithm 4 of the TX81Z.

```
Algorithm 4 is :   4->3--\
                      2-- + -->1-->Out
```

Control Change Numbers:

- Total Modulator Index = 2
- Modulator Crossfade = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

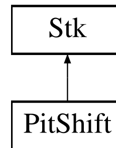
- PercFlut.h

4.39 PitShift Class Reference

STK simple pitch shifter effect class.

`#include <PitShift.h>`

Inheritance diagram for PitShift::



Public Methods

- PitShift ()

  Class constructor.

- ~PitShift ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setShift (MY_FLOAT shift)

  Set the pitch shift factor (1.0 produces no shift).

- void setEffectMix (MY_FLOAT mix)

  Set the mixture of input and processed levels in the output (0.0 = input only, 1.0 = processed only).

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector*.*

4.39.1 Detailed Description

STK simple pitch shifter effect class.

This class implements a simple pitch shifter using delay lines.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

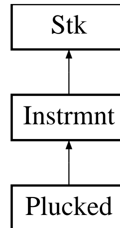
- PitShift.h

4.40 Plucked Class Reference

STK plucked string model class.

`#include <Plucked.h>`

Inheritance diagram for Plucked::



Public Methods

- Plucked (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~Plucked ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void pluck (MY_FLOAT amplitude)

  Pluck the string with the given amplitude using the current frequency.

- virtual void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- virtual void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()

  Compute one output sample.

4.40.1 Detailed Description

STK plucked string model class.

This class implements a simple plucked string physical model based on the Karplus-Strong algorithm.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others. There exist at least two patents, assigned to Stanford, bearing the names of Karplus and/or Strong.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

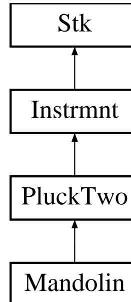
- Plucked.h

4.41 PluckTwo Class Reference

STK enhanced plucked string model class.

#include <PluckTwo.h>

Inheritance diagram for PluckTwo::



Public Methods

- PluckTwo (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- virtual ~PluckTwo ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setDetune (MY_FLOAT detune)

  Detune the two strings by the given factor. A value of 1.0 produces unison strings.

- void setFreqAndDetune (MY_FLOAT frequency, MY_FLOAT detune)

  Efficient combined setting of frequency and detuning.

- void setPluckPosition (MY_FLOAT position)

  Set the pluck or "excitation" position along the string (0.0 - 1.0).

- void setBaseLoopGain (MY_FLOAT aGain)

Set the base loop gain.

- virtual void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()=0

  Virtual (abstract) tick function is implemented by subclasses.

4.41.1 Detailed Description

STK enhanced plucked string model class.

This class implements an enhanced two-string, plucked physical model, a la Jaffe-Smith, Smith, and others.

PluckTwo is an abstract class, with no excitation specified. Therefore, it can't be directly instantiated.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.41.2 Member Function Documentation

4.41.2.1 void PluckTwo::setBaseLoopGain (MY_FLOAT *aGain*)

Set the base loop gain.

The actual loop gain is set according to the frequency. Because of high-frequency loop filter roll-off, higher frequency settings have greater loop gains.

The documentation for this class was generated from the following file:

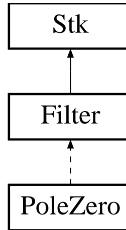
- PluckTwo.h

4.42 PoleZero Class Reference

STK one-pole, one-zero filter class.

`#include <PoleZero.h>`

Inheritance diagram for PoleZero::



Public Methods

- PoleZero ()

  Default constructor creates a first-order pass-through filter.

- ~PoleZero ()

  Class destructor.

- void clear (void)

  Clears the internal states of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setB1 (MY_FLOAT b1)

  Set the b[1] coefficient value.

- void setA1 (MY_FLOAT a1)

  Set the a[1] coefficient value.

- void setAllpass (MY_FLOAT coefficient)

  Set the filter for allpass behavior using coefficient*.*

- void setBlockZero (MY_FLOAT thePole=0.99)

  Create a DC blocking filter with the given pole position in the z-plane.

- void setGain (MY_FLOAT theGain)

Set the filter gain.

- MY_FLOAT getGain (void) const

  Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.42.1 Detailed Description

STK one-pole, one-zero filter class.

This protected Filter subclass implements a one-pole, one-zero digital filter. A method is provided for creating an allpass filter with a given coefficient. Another method is provided to create a DC blocking filter.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.42.2 Member Function Documentation

4.42.2.1 void PoleZero::setAllpass (MY_FLOAT *coefficient*)

Set the filter for allpass behavior using *coefficient*.

This method uses *coefficient* to create an allpass filter, which has unity gain at all frequencies. Note that the *coefficient* magnitude must be less than one to maintain stability.

4.42.2.2 void PoleZero::setBlockZero (MY_FLOAT *thePole* = 0.99)

Create a DC blocking filter with the given pole position in the z-plane.

This method sets the given pole position, together with a zero at z=1, to create a DC blocking filter. *thePole* should be close to one to minimize low-frequency attenuation.

4.42.2.3 void PoleZero::setGain (MY_FLOAT *theGain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

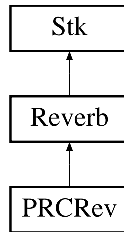
- PoleZero.h

4.43 PRCRev Class Reference

Perry's simple reverberator class.

`#include <PRCRev.h>`

Inheritance diagram for PRCRev::



Public Methods

- void clear ()

  Reset and clear all internal state.

- MY_FLOAT tick (MY_FLOAT input)

  Compute one output sample.

4.43.1 Detailed Description

Perry's simple reverberator class.

This class is based on some of the famous Stanford/CCRMA reverbs (NRev, KipRev), which were based on the Chowning/Moorer/Schroeder reverberators using networks of simple allpass and comb delay filters. This class implements two series allpass units and two parallel comb filters.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

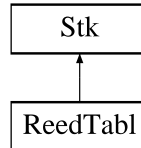
- PRCRev.h

4.44 ReedTabl Class Reference

STK reed table class.

`#include <ReedTabl.h>`

Inheritance diagram for ReedTabl::



Public Methods

- ReedTabl ()

  Default constructor.

- ~ReedTabl ()

  Class destructor.

- void setOffset (MY_FLOAT aValue)

  Set the table offset value.

- void setSlope (MY_FLOAT aValue)

  Set the table slope value.

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT tick (MY_FLOAT input)

  Return the function value for input.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Take vectorSize *inputs and return the corresponding function values in* vector.

4.44.1 Detailed Description

STK reed table class.

This class implements a simple one breakpoint, non-linear reed function, as described by Smith (1986). This function is based on a memoryless non-linear spring model of the reed (the reed mass is ignored) which saturates when the reed collides with the mouthpiece facing.

See McIntyre, Schumacher, & Woodhouse (1983), Smith (1986), Hirschman, Cook, Scavone, and others for more information.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.44.2 Member Function Documentation

4.44.2.1 void ReedTabl::setOffset (MY_FLOAT *aValue*)

Set the table offset value.

The table offset roughly corresponds to the size of the initial reed tip opening (a greater offset represents a smaller opening).

4.44.2.2 void ReedTabl::setSlope (MY_FLOAT *aValue*)

Set the table slope value.

The table slope roughly corresponds to the reed stiffness (a greater slope represents a harder reed).

4.44.2.3 MY_FLOAT ReedTabl::tick (MY_FLOAT *input*)

Return the function value for *input*.

The function input represents the differential pressure across the reeds.

The documentation for this class was generated from the following file:

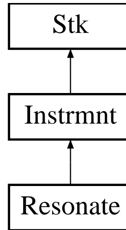
- ReedTabl.h

4.45 Resonate Class Reference

STK noise driven formant filter.

`#include <Resonate.h>`

Inheritance diagram for Resonate::



Public Methods

- Resonate ()

  Class constructor.

- ~Resonate ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setResonance (MY_FLOAT frequency, MY_FLOAT radius)

  Set the filter for a resonance at the given frequency (Hz) and radius.

- void setNotch (MY_FLOAT frequency, MY_FLOAT radius)

  Set the filter for a notch at the given frequency (Hz) and radius.

- void setEqualGainZeroes ()

  Set the filter zero coefficients for contant resonance gain.

- void keyOn ()

  Initiate the envelope with a key-on event.

- void keyOff ()

  Signal a key-off event to the envelope.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.45.1 Detailed Description

STK noise driven formant filter.

This instrument contains a noise source, which excites a biquad resonance filter, with volume controlled by an ADSR.

Control Change Numbers:

- Resonance Frequency (0-Nyquist) = 2
- Pole Radii = 4
- Notch Frequency (0-Nyquist) = 11
- Zero Radii = 1
- Envelope Gain = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

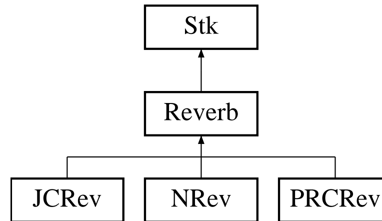
- Resonate.h

4.46 Reverb Class Reference

STK abstract reverberator parent class.

`#include <Reverb.h>`

Inheritance diagram for Reverb::



Public Methods

- Reverb ()

  Class constructor.

- virtual ~Reverb ()

  Class destructor.

- virtual void clear ()=0

  Reset and clear all internal state.

- void setEffectMix (MY_FLOAT mix)

  Set the mixture of input and "reverberated" levels in the output (0.0 = input only, 1.0 = reverb only).

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT lastOutLeft () const

  Return the last left output value.

- MY_FLOAT lastOutRight () const

  Return the last right output value.

- virtual MY_FLOAT tick (MY_FLOAT input)=0

  Abstract tick function ... must be implemented in subclasses.

- virtual MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Take vectorSize *inputs, compute the same number of outputs and return them in* vector.

4.46.1 Detailed Description

STK abstract reverberator parent class.

This class provides common functionality for STK reverberator subclasses.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

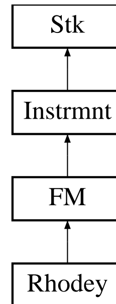
- Reverb.h

4.47 Rhodey Class Reference

STK Fender Rhodes electric piano FM synthesis instrument.

`#include <Rhodey.h>`

Inheritance diagram for Rhodey::



Public Methods

- Rhodey ()

  Class constructor.

- ~Rhodey ()

  Class destructor.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- MY_FLOAT tick ()

  Compute one output sample.

4.47.1 Detailed Description

STK Fender Rhodes electric piano FM synthesis instrument.

This class implements two simple FM Pairs summed together, also referred to as algorithm 5 of the TX81Z.

```
Algorithm 5 is :  4->3--\
                           + --> Out
                  2->1--/
```

Control Change Numbers:

- Modulator Index One = 2
- Crossfade of Outputs = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

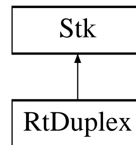
- Rhodey.h

4.48 RtDuplex Class Reference

STK realtime audio input/output class.

`#include <RtDuplex.h>`

Inheritance diagram for RtDuplex::



Public Methods

- RtDuplex (int nChannels=1, MY_FLOAT sampleRate=Stk::sampleRate(), int device=0, int bufferFrames=RT_BUFFER_SIZE, int nBuffers=2)

  Default constructor.

- ~RtDuplex ()

  Class destructor.

- void start (void)

  Start the audio input/output stream.

- void stop (void)

  Stop the audio input/output stream.

- MY_FLOAT lastOut (void) const

  Return the average across the last output sample frame.

- MY_FLOAT tick (const MY_FLOAT sample)

  Output a single sample to all channels in a sample frame and return the average across one new input sample frame of data.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Output each sample in \vector to all channels per frame and return averaged input sample frames of new data in vector.

- const MY_FLOAT* lastFrame (void) const

  Return a pointer to the last output sample frame.

- MY_FLOAT* tickFrame (MY_FLOAT *frameVector, unsigned int frames=1)

  Output sample frames *from* frameVector *and return new input frames in* frameVector*.*

4.48.1 Detailed Description

STK realtime audio input/output class.

This class provides a simplified interface to RtAudio for realtime audio input/output. It is also possible to achieve duplex operation using separate RtWvIn and RtWvOut classes, but this class ensures better input/output syncronization.

RtDuplex supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which output single samples to all channels in a sample frame and return samples produced by averaging across sample frames, from the tickFrame() methods, which take/return pointers to multi-channel sample frames.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 RtDuplex::RtDuplex (int *nChannels* = 1, MY_FLOAT *sampleRate* = Stk::sampleRate(), int *device* = 0, int *bufferFrames* = RT_BUFFER_SIZE, int *nBuffers* = 2)

Default constructor.

The *device* argument is passed to RtAudio during instantiation. The default value (zero) will select the default device on your system or the first device found meeting the specified parameters. On systems with multiple soundcards/devices, values greater than zero can be specified in accordance with the order that the devices are enumerated by the underlying audio API. The default buffer size of RT_BUFFER_SIZE is defined in Stk.h. An StkError will be thrown if an error occurs duing instantiation.

4.48.3 Member Function Documentation

4.48.3.1 void RtDuplex::start (void)

Start the audio input/output stream.

The stream is started automatically, if necessary, when a tick() or tickFrame method is called.

4.48.3.2 void RtDuplex::stop (void)

Stop the audio input/output stream.

It may be necessary to use this method to avoid audio overflow/underflow problems if you wish to temporarily stop the audio stream.

4.48.3.3 MY_FLOAT RtDuplex::tick (const MY_FLOAT *sample*)

Output a single sample to all channels in a sample frame and return the average across one new input sample frame of data.

An StkError will be thrown if an error occurs during input/output.

4.48.3.4 MY_FLOAT * RtDuplex::tick (MY_FLOAT * *vector*, unsigned int *vectorSize*)

Output each sample in \vector to all channels per frame and return averaged input sample frames of new data in *vector*.

An StkError will be thrown if an error occurs during input/output.

4.48.3.5 MY_FLOAT * RtDuplex::tickFrame (MY_FLOAT * *frameVector*, unsigned int *frames* = 1)

Output sample *frames* from *frameVector* and return new input frames in *frameVector*.

An StkError will be thrown if an error occurs during input/output.

The documentation for this class was generated from the following file:

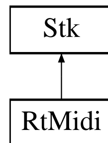
- RtDuplex.h

4.49 RtMidi Class Reference

STK realtime MIDI class.

`#include <RtMidi.h>`

Inheritance diagram for RtMidi::



Public Methods

- RtMidi (int device=0)

  Default constructor with optional device argument.

- ~RtMidi ()

  Class destructor.

- void printMessage (void) const

  Print out the current message values.

- int nextMessage (void)

  Check for and parse a new MIDI message in the queue, returning its type.

- int getType () const

  Return the current message type.

- int getChannel () const

  Return the current message channel value.

- MY_FLOAT getByteTwo () const

  Return the current message byte two value.

- MY_FLOAT getByteThree () const

  Return the current message byte three value.

- MY_FLOAT getDeltaTime () const

  Return the current message delta time value in seconds.

4.49.1 Detailed Description

STK realtime MIDI class.

At the moment, this object only handles MIDI input, though MIDI output code can go here when someone decides they need it (and writes it).

This object opens a MIDI input device and parses MIDI messages into a MIDI buffer. Time stamp info is converted to a delta-time value. MIDI data is stored as MY_FLOAT to conform with SKINI. System exclusive messages are currently ignored.

An optional argument to the constructor can be used to specify a device or card. When no argument is given, a default device is opened. If a device argument fails, a list of available devices is printed to allow selection by the user.

This code is based in part on work of Perry Cook (SGI), Paul Leonard (Linux), the RoseGarden team (Linux), and Bill Putnam (Windows).

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.49.2 Member Function Documentation

4.49.2.1 int RtMidi::nextMessage (void)

Check for and parse a new MIDI message in the queue, returning its type.

If a new message is found, the return value is greater than zero.

The documentation for this class was generated from the following file:

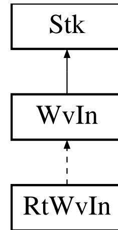
- RtMidi.h

4.50 RtWvIn Class Reference

STK realtime audio input class.

`#include <RtWvIn.h>`

Inheritance diagram for RtWvIn::



Public Methods

- RtWvIn (int nChannels=1, MY_FLOAT sampleRate=Stk::sampleRate(), int device=0, int bufferFrames=RT_BUFFER_SIZE, int nBuffers=2)

  Default constructor.

- ~RtWvIn ()

  Class destructor.

- void start (void)

  Start the audio input stream.

- void stop (void)

  Stop the audio input stream.

- MY_FLOAT lastOut (void) const

  Return the average across the last output sample frame.

- MY_FLOAT tick (void)

  Read out the average across one sample frame of data.

- MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Read out vectorSize averaged sample frames of data in vector.

- const MY_FLOAT∗ lastFrame (void) const

  Return a pointer to the last output sample frame.

- const MY_FLOAT* tickFrame (void)

  Return a pointer to the next sample frame of data.

- MY_FLOAT* tickFrame (MY_FLOAT *frameVector, unsigned int frames)

  Read out sample frames *of data to* frameVector*.*

4.50.1 Detailed Description

STK realtime audio input class.

This class provides a simplified interface to RtAudio for realtime audio input. It is a protected subclass of WvIn.

RtWvIn supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which return samples produced by averaging across sample frames, from the tickFrame() methods, which return pointers to multi-channel sample frames. For single-channel data, these methods return equivalent values.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 RtWvIn::RtWvIn (int *nChannels* = 1, MY_FLOAT *sampleRate* = Stk::sampleRate(), int *device* = 0, int *bufferFrames* = RT_BUFFER_SIZE, int *nBuffers* = 2)

Default constructor.

The *device* argument is passed to RtAudio during instantiation. The default value (zero) will select the default device on your system or the first device found meeting the specified parameters. On systems with multiple soundcards/devices, values greater than zero can be specified in accordance with the order that the devices are enumerated by the underlying audio API. The default buffer size of RT_BUFFER_SIZE is defined in Stk.h. An StkError will be thrown if an error occurs duing instantiation.

4.50.3 Member Function Documentation

4.50.3.1 void RtWvIn::start (void)

Start the audio input stream.

The stream is started automatically, if necessary, when a tick() or tickFrame method is called.

4.50.3.2 void RtWvIn::stop (void)

Stop the audio input stream.

It may be necessary to use this method to avoid audio underflow problems if you wish to temporarily stop audio input.

4.50.3.3 MY_FLOAT RtWvIn::tick (void) `[virtual]`

Read out the average across one sample frame of data.

An StkError will be thrown if an error occurs during input.

Reimplemented from WvIn.

4.50.3.4 MY_FLOAT * RtWvIn::tick (MY_FLOAT * *vector*, unsigned int *vectorSize*) `[virtual]`

Read out vectorSize averaged sample frames of data in *vector*.

An StkError will be thrown if an error occurs during input.

Reimplemented from WvIn.

4.50.3.5 const MY_FLOAT * RtWvIn::tickFrame (void) `[virtual]`

Return a pointer to the next sample frame of data.

An StkError will be thrown if an error occurs during input.

Reimplemented from WvIn.

4.50.3.6 MY_FLOAT * RtWvIn::tickFrame (MY_FLOAT * *frameVector*, unsigned int *frames*) `[virtual]`

Read out sample *frames* of data to *frameVector*.

An StkError will be thrown if an error occurs during input.

Reimplemented from WvIn.

The documentation for this class was generated from the following file:

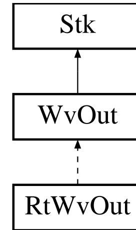
- RtWvIn.h

4.51 RtWvOut Class Reference

STK realtime audio output class.

`#include <RtWvOut.h>`

Inheritance diagram for RtWvOut::



Public Methods

- RtWvOut (unsigned int nChannels=1, MY_FLOAT sampleRate=Stk::sampleRate(), int device=0, int bufferFrames=RT_BUFFER_SIZE, int nBuffers=4)

  Default constructor.

- ~RtWvOut ()

  Class destructor.

- void start (void)

  Start the audio output stream.

- void stop (void)

  Stop the audio output stream.

- unsigned long getFrames (void) const

  Return the number of sample frames output.

- MY_FLOAT getTime (void) const

  Return the number of seconds of data output.

- void tick (const MY_FLOAT sample)

  Output a single sample to all channels in a sample frame.

- void tick (const MY_FLOAT ∗vector, unsigned int vectorSize)

  Output each sample in vector *to all channels in* vectorSize *sample frames.*

- void tickFrame (const MY_FLOAT *frameVector, unsigned int frames=1)

  Output the frameVector *of sample frames of the given length.*

4.51.1 Detailed Description

STK realtime audio output class.

This class provides a simplified interface to RtAudio for realtime audio output. It is a protected subclass of WvOut.

RtWvOut supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which output single samples to all channels in a sample frame, from the tickFrame() method, which takes a pointer to multi-channel sample frame data.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.51.2 Constructor & Destructor Documentation

4.51.2.1 RtWvOut::RtWvOut (unsigned int *nChannels* = 1, MY_FLOAT *sampleRate* = Stk::sampleRate(), int *device* = 0, int *bufferFrames* = RT_BUFFER_SIZE, int *nBuffers* = 4)

Default constructor.

The *device* argument is passed to RtAudio during instantiation. The default value (zero) will select the default device on your system or the first device found meeting the specified parameters. On systems with multiple soundcards/devices, values greater than zero can be specified in accordance with the order that the devices are enumerated by the underlying audio API. The default buffer size of RT_BUFFER_SIZE is defined in Stk.h. An StkError will be thrown if an error occurs during instantiation.

4.51.3 Member Function Documentation

4.51.3.1 void RtWvOut::start (void)

Start the audio output stream.

The stream is started automatically, if necessary, when a tick() or tickFrame method is called.

4.51.3.2 void RtWvOut::stop (void)

Stop the audio output stream.

It may be necessary to use this method to avoid undesireable audio buffer cycling if you wish to temporarily stop audio output.

4.51.3.3 void RtWvOut::tick (const MY_FLOAT *sample*) `[virtual]`

Output a single sample to all channels in a sample frame.

An StkError will be thrown if an error occurs during output.

Reimplemented from WvOut.

4.51.3.4 void RtWvOut::tick (const MY_FLOAT * *vector*, unsigned int *vectorSize*) `[virtual]`

Output each sample in *vector* to all channels in *vectorSize* sample frames.

An StkError will be thrown if an error occurs during output.

Reimplemented from WvOut.

4.51.3.5 void RtWvOut::tickFrame (const MY_FLOAT * *frameVector*, unsigned int *frames* = 1) `[virtual]`

Output the *frameVector* of sample frames of the given length.

An StkError will be thrown if an error occurs during output.

Reimplemented from WvOut.

The documentation for this class was generated from the following file:

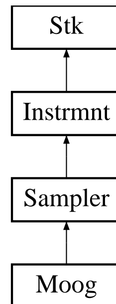
- RtWvOut.h

4.52 Sampler Class Reference

STK sampling synthesis abstract base class.

`#include <Sampler.h>`

Inheritance diagram for Sampler::



Public Methods

- Sampler ()

  Default constructor.

- virtual ~Sampler ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- virtual void setFrequency (MY_FLOAT frequency)=0

  Set instrument parameters for a particular frequency.

- void keyOn ()

  Initiate the envelopes with a key-on event and reset the attack waves.

- void keyOff ()

  Signal a key-off event to the envelopes.

- virtual void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()

Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)=0

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.52.1 Detailed Description

STK sampling synthesis abstract base class.

This instrument contains up to 5 attack waves, 5 looped waves, and an ADSR envelope.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

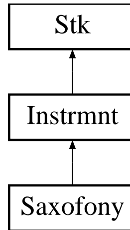
- Sampler.h

4.53 Saxofony Class Reference

STK faux conical bore reed instrument class.

`#include <Saxofony.h>`

Inheritance diagram for Saxofony::



Public Methods

- Saxofony (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~Saxofony ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setBlowPosition (MY_FLOAT aPosition)

  Set the "blowing" position between the air column terminations (0.0 - 1.0).

- void startBlowing (MY_FLOAT amplitude, MY_FLOAT rate)

  Apply breath pressure to instrument with given amplitude and rate of increase.

- void stopBlowing (MY_FLOAT rate)

  Decrease breath pressure with given rate of decrease.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.53.1 Detailed Description

STK faux conical bore reed instrument class.

This class implements a "hybrid" digital waveguide instrument that can generate a variety of wind-like sounds. It has also been referred to as the "blowed string" model. The waveguide section is essentially that of a string, with one rigid and one lossy termination. The non-linear function is a reed table. The string can be "blown" at any point between the terminations, though just as with strings, it is impossible to excite the system at either end. If the excitation is placed at the string mid-point, the sound is that of a clarinet. At points closer to the "bridge", the sound is closer to that of a saxophone. See Scavone (2002) for more details.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Reed Stiffness = 2
- Reed Aperture = 26
- Noise Gain = 4
- Blow Position = 11
- Vibrato Frequency = 29
- Vibrato Gain = 1
- Breath Pressure = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

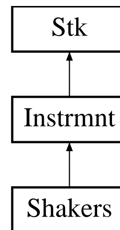
- Saxofony.h

4.54 Shakers Class Reference

PhISEM and PhOLIES class.

`#include <Shakers.h>`

Inheritance diagram for Shakers::



Public Methods

- Shakers ()

  Class constructor.

- ~Shakers ()

  Class destructor.

- virtual void noteOn (MY_FLOAT instrument, MY_FLOAT amplitude)

  Start a note with the given instrument and amplitude.

- virtual void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.54.1 Detailed Description

PhISEM and PhOLIES class.

PhISEM (Physically Informed Stochastic Event Modeling) is an algorithmic approach for simulating collisions of multiple independent sound producing objects. This class is a meta-model that can simulate a Maraca, Sekere, Cabasa, Bamboo Wind Chimes, Water Drops, Tambourine, Sleighbells, and a Guiro.

PhOLIES (Physically-Oriented Library of Imitated Environmental Sounds) is a similar approach for the synthesis of environmental sounds. This class implements simulations of breaking sticks, crunchy snow (or not), a wrench, sandpaper, and more.

Control Change Numbers:

- Shake Energy = 2
- System Decay = 4
- Number Of Objects = 11
- Resonance Frequency = 1
- Shake Energy = 128
- Instrument Selection = 1071
  - Maraca = 0
  - Cabasa = 1
  - Sekere = 2
  - Guiro = 3
  - Water Drops = 4
  - Bamboo Chimes = 5
  - Tambourine = 6
  - Sleigh Bells = 7
  - Sticks = 8
  - Crunch = 9
  - Wrench = 10
  - Sand Paper = 11
  - Coke Can = 12
  - Next Mug = 13
  - Penny + Mug = 14
  - Nickle + Mug = 15
  - Dime + Mug = 16
  - Quarter + Mug = 17
  - Franc + Mug = 18
  - Peso + Mug = 19
  - Big Rocks = 20
  - Little Rocks = 21
  - Tuned Bamboo Chimes = 22

by Perry R. Cook, 1996 - 1999.

4.54.2 Member Function Documentation

4.54.2.1 void Shakers::noteOn (MY_FLOAT *instrument*, MY_FLOAT *amplitude*) [virtual]

Start a note with the given instrument and amplitude.

Use the instrument numbers above, converted to frequency values as if MIDI note numbers, to select a particular instrument.

Reimplemented from Instrmnt.

The documentation for this class was generated from the following file:

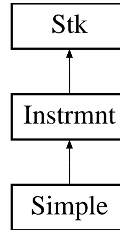
- Shakers.h

4.55 Simple Class Reference

STK wavetable/noise instrument.

`#include <Simple.h>`

Inheritance diagram for Simple::



Public Methods

- Simple ()

  Class constructor.

- virtual ~Simple ()

  Class destructor.

- void clear ()

  Clear internal states.

- virtual void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void keyOn ()

  Start envelope toward "on" target.

- void keyOff ()

  Start envelope toward "off" target.

- virtual void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- virtual void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- virtual MY_FLOAT tick ()

Compute one output sample.

- virtual void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.55.1 Detailed Description

STK wavetable/noise instrument.

This class combines a looped wave, a noise source, a biquad resonance filter, a one-pole filter, and an ADSR envelope to create some interesting sounds.

Control Change Numbers:

- Filter Pole Position = 2
- Noise/Pitched Cross-Fade = 4
- Envelope Rate = 11
- Gain = 128

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

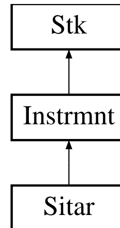
- Simple.h

4.56 Sitar Class Reference

STK sitar string model class.

`#include <Sitar.h>`

Inheritance diagram for Sitar::



Public Methods

- Sitar (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~Sitar ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void pluck (MY_FLOAT amplitude)

  Pluck the string with the given amplitude using the current frequency.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

4.56.1 Detailed Description

STK sitar string model class.

This class implements a sitar plucked string physical model based on the Karplus-Strong algorithm.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others. There exist at least two patents, assigned to Stanford, bearing the names of Karplus and/or Strong.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

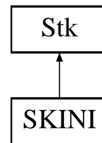
- Sitar.h

4.57 SKINI Class Reference

STK SKINI parsing class.

`#include <SKINI.h>`

Inheritance diagram for SKINI::



Public Methods

- SKINI ()

  Default constructor used for parsing messages received externally.

- SKINI (char *fileName)

  Overloaded constructor taking a SKINI formatted scorefile.

- ~SKINI ()

  Class destructor.

- long parseThis (char *aString)

  Attempt to parse the given string, returning the message type.

- long nextMessage ()

  Parse the next message (if a file is loaded) and return the message type.

- long getType () const

  Return the current message type.

- long getChannel () const

  Return the current message channel value.

- MY_FLOAT getDelta () const

  Return the current message delta time value (in seconds).

- MY_FLOAT getByteTwo () const

  Return the current message byte two value.

- MY_FLOAT getByteThree () const

  Return the current message byte three value.

- long getByteTwoInt () const

  Return the current message byte two value (integer).

- long getByteThreeInt () const

  Return the current message byte three value (integer).

- const char* getRemainderString ()

  Return remainder string after parsing.

- const char* getMessageTypeString ()

  Return the message type as a string.

- const char* whatsThisType (long type)

  Return the SKINI type string for the given type value.

- const char* whatsThisController (long number)

  Return the SKINI controller string for the given controller number.

4.57.1 Detailed Description

STK SKINI parsing class.

This class parses SKINI formatted text messages. It can be used to parse individual messages or it can be passed an entire file. The file specification is Perry's and his alone, but it's all text so it shouldn't be to hard to figure out.

SKINI (Synthesis toolKit Instrument Network Interface) is like MIDI, but allows for floating-point control changes, note numbers, etc. The following example causes a sharp middle C to be played with a velocity of 111.132:

```
noteOn  60.01  111.13
```

See also:
Synthesis toolKit Instrument Network Interface (SKINI)

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.57.2 Member Function Documentation

4.57.2.1 long SKINI::parseThis (char * *aString*)

Attempt to parse the given string, returning the message type.

A type value equal to zero indicates an invalid message.

4.57.2.2 long SKINI::nextMessage (void)

Parse the next message (if a file is loaded) and return the message type.

A negative value is returned when the file end is reached.

The documentation for this class was generated from the following file:

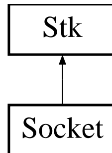
- SKINI.h

4.58 Socket Class Reference

STK TCP socket client/server class.

`#include <Socket.h>`

Inheritance diagram for Socket::



Public Methods

- Socket (int port=2006)

  Default constructor which creates a local socket server on port 2006 (or the specified port number).

- Socket (int port, const char *hostname)

  Class constructor which creates a socket client connection to the specified host and port.

- ~Socket ()

  The class destructor closes the socket instance, breaking any existing connections.

- int connect (int port, const char *hostname="localhost")

  Connect a socket client to the specified host and port and returns the resulting socket descriptor.

- void close (void)

  Close this socket.

- int socket (void) const

  Return the server/client socket descriptor.

- int port (void) const

  Return the server/client port number.

- int accept (void)

If this is a socket server, extract the first pending connection request from the queue and create a new connection, returning the descriptor for the accepted socket.

- int writeBuffer (const void *buffer, long bufferSize, int flags=0)

  Write a buffer over the socket connection. Returns the number of bytes written or -1 if an error occurs.

- int readBuffer (void *buffer, long bufferSize, int flags=0)

  Read a buffer from the socket connection, up to length bufferSize. *Returns the number of bytes read or -1 if an error occurs.*

Static Public Methods

- void setBlocking (int socket, bool enable)

  If enable = false, the socket is set to non-blocking mode. When first created, sockets are by default in blocking mode.

- void close (int socket)

  Close the socket with the given descriptor.

- bool isValid (int socket)

  Returns TRUE is the socket descriptor is valid.

- int writeBuffer (int socket, const void *buffer, long bufferSize, int flags)

  Write a buffer via the specified socket. Returns the number of bytes written or -1 if an error occurs.

- int readBuffer (int socket, void *buffer, long bufferSize, int flags)

  Read a buffer via the specified socket. Returns the number of bytes read or -1 if an error occurs.

4.58.1 Detailed Description

STK TCP socket client/server class.

This class provides a uniform cross-platform TCP socket client or socket server interface. Methods are provided for reading or writing data buffers to/from connections. This class also provides a number of static functions for use with external socket descriptors.

The user is responsible for checking the values returned by the read/write methods. Values less than or equal to zero indicate a closed or lost connection or the occurence of an error.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.58.2 Constructor & Destructor Documentation

4.58.2.1 Socket::Socket (int *port* = 2006)

Default constructor which creates a local socket server on port 2006 (or the specified port number).

An StkError will be thrown if a socket error occurs during instantiation.

4.58.2.2 Socket::Socket (int *port*, const char * *hostname*)

Class constructor which creates a socket client connection to the specified host and port.

An StkError will be thrown if a socket error occurs during instantiation.

4.58.3 Member Function Documentation

4.58.3.1 int Socket::connect (int *port*, const char * *hostname* = "localhost")

Connect a socket client to the specified host and port and returns the resulting socket descriptor.

This method is valid for socket clients only. If it is called for a socket server, -1 is returned. If the socket client is already connected, that connection is terminated and a new connection is attempted. Server connections are made using the accept() method. An StkError will be thrown if a socket error occurs during instantiation.

See also:
accept

4.58.3.2 int Socket::accept (void)

If this is a socket server, extract the first pending connection request from the queue and create a new connection, returning the descriptor for the accepted socket.

If no connection requests are pending and the socket has not been set non-blocking, this function will block until a connection is present. If an error occurs or this is a socket client, -1 is returned.

The documentation for this class was generated from the following file:

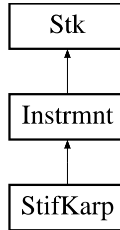
- Socket.h

4.59 StifKarp Class Reference

STK plucked stiff string instrument.

`#include <StifKarp.h>`

Inheritance diagram for StifKarp::



Public Methods

- StifKarp (MY_FLOAT lowestFrequency)

  Class constructor, taking the lowest desired playing frequency.

- ~StifKarp ()

  Class destructor.

- void clear ()

  Reset and clear all internal state.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void setStretch (MY_FLOAT stretch)

  Set the stretch "factor" of the string (0.0 - 1.0).

- void setPickupPosition (MY_FLOAT position)

  Set the pluck or "excitation" position along the string (0.0 - 1.0).

- void setBaseLoopGain (MY_FLOAT aGain)

  Set the base loop gain.

- void pluck (MY_FLOAT amplitude)

  Pluck the string with the given amplitude using the current frequency.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

Start a note with the given frequency and amplitude.

- void noteOff (MY_FLOAT amplitude)

  Stop a note with the given amplitude (speed of decay).

- MY_FLOAT tick ()

  Compute one output sample.

- void controlChange (int number, MY_FLOAT value)

  Perform the control change specified by number *and* value *(0.0 - 128.0).*

4.59.1 Detailed Description

STK plucked stiff string instrument.

This class implements a simple plucked string algorithm (Karplus Strong) with enhancements (Jaffe-Smith, Smith, and others), including string stiffness and pluck position controls. The stiffness is modeled with allpass filters.

This is a digital waveguide model, making its use possibly subject to patents held by Stanford University, Yamaha, and others.

Control Change Numbers:

- Pickup Position = 4
- String Sustain = 11
- String Stretch = 1

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.59.2 Member Function Documentation

4.59.2.1 void StifKarp::setBaseLoopGain (MY_FLOAT *aGain*)

Set the base loop gain.

The actual loop gain is set according to the frequency. Because of high-frequency loop filter roll-off, higher frequency settings have greater loop gains.

The documentation for this class was generated from the following file:

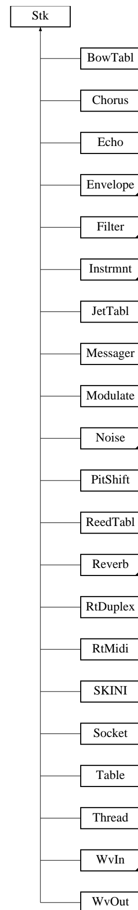
- StifKarp.h

4.60 Stk Class Reference

STK base class.

`#include <Stk.h>`

Inheritance diagram for Stk::



Static Public Methods

- MY_FLOAT sampleRate (void)

  Static method which returns the current STK sample rate.

- void setSampleRate (MY_FLOAT newRate)

  Static method which sets the STK sample rate.

- void swap16 (unsigned char *ptr)

  Static method which byte-swaps a 16-bit data type.

- void swap32 (unsigned char *ptr)

  Static method which byte-swaps a 32-bit data type.

- void swap64 (unsigned char *ptr)

  Static method which byte-swaps a 64-bit data type.

- void sleep (unsigned long milliseconds)

  Static cross-platform method to sleep for a number of milliseconds.

Static Public Attributes

- const STK_FORMAT STK_SINT8
- const STK_FORMAT STK_SINT16
- const STK_FORMAT STK_SINT32
- const STK_FORMAT STK_FLOAT32
- const STK_FORMAT STK_FLOAT64

Protected Methods

- Stk (void)

  Default constructor.

- virtual ~Stk (void)

  Class destructor.

Static Protected Methods

- void handleError (const char *message, StkError::TYPE type)

  Function for error reporting and handling.

4.60.1 Detailed Description

STK base class.

Nearly all STK classes inherit from this class. The global sample rate can be queried and modified via Stk. In addition, this class provides error handling and byte-swapping functions.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.60.2 Member Function Documentation

4.60.2.1 void Stk::setSampleRate (MY_FLOAT *newRate*) `[static]`

Static method which sets the STK sample rate.

The sample rate set using this method is queried by all STK classes which depend on its value. It is initialized to the default SRATE set in Stk.h. Many STK classes use the sample rate during instantiation. Therefore, if you wish to use a rate which is different from the default rate, it is imperative that it be set *BEFORE* STK objects are instantiated.

4.60.3 Member Data Documentation

4.60.3.1 const STK_FORMAT Stk::STK_SINT8 `[static]`

-128 to +127

4.60.3.2 const STK_FORMAT Stk::STK_SINT16 `[static]`

-32768 to +32767

4.60.3.3 const STK_FORMAT Stk::STK_SINT32 `[static]`

-2147483648 to +2147483647.

4.60.3.4 const STK_FORMAT Stk::STK_FLOAT32 `[static]`

Normalized between plus/minus 1.0.

4.60.3.5 const STK_FORMAT Stk::STK_FLOAT64 `[static]`

Normalized between plus/minus 1.0.

The documentation for this class was generated from the following file:

- Stk.h

4.61 StkError Class Reference

STK error handling class.

`#include <Stk.h>`

Public Methods

- StkError (const char *p, TYPE tipe=StkError::UNSPECIFIED)

  The constructor.

- virtual ~StkError (void)

  The destructor.

- virtual void printMessage (void)

  Prints "thrown" error message to stdout.

- virtual const TYPE& getType (void)

  Returns the "thrown" error message TYPE.

- virtual const char* getMessage (void) const

  Returns the "thrown" error message string.

4.61.1 Detailed Description

STK error handling class.

This is a fairly abstract exception handling class. There could be sub-classes to take care of more specific error conditions ... or not.

The documentation for this class was generated from the following file:

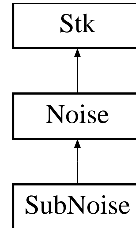
- Stk.h

4.62 SubNoise Class Reference

STK sub-sampled noise generator.

`#include <SubNoise.h>`

Inheritance diagram for SubNoise::



Public Methods

- SubNoise (int subRate=16)

  Default constructor sets sub-sample rate to 16.

- ~SubNoise ()

  Class destructor.

- int subRate (void) const

  Return the current sub-sampling rate.

- void setRate (int subRate)

  Set the sub-sampling rate.

- MY_FLOAT tick ()

  Return a sub-sampled random number between -1.0 and 1.0.

4.62.1 Detailed Description

STK sub-sampled noise generator.

Generates a new random number every "rate" ticks using the C rand() function. The quality of the rand() function varies from one OS to another.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

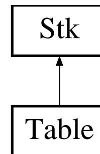
- SubNoise.h

4.63 Table Class Reference

STK table lookup class.

`#include <Table.h>`

Inheritance diagram for Table::



Public Methods

- Table (char *fileName)

  Constructor loads the data from fileName*.*

- ~Table ()

  Class destructor.

- long getLength () const

  Return the number of elements in the table.

- MY_FLOAT lastOut () const

  Return the last output value.

- MY_FLOAT tick (MY_FLOAT index)

  Return the table value at position index*.*

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Take vectorSize *index positions and return the corresponding table values in* vector*.*

4.63.1 Detailed Description

STK table lookup class.

This class loads a table of floating-point doubles, which are assumed to be in big-endian format. Linear interpolation is performed for fractional lookup indexes.

An StkError will be thrown if the table file is not found.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.63.2 Member Function Documentation

4.63.2.1 MY_FLOAT Table::tick (MY_FLOAT *index*)

Return the table value at position *index*.

Linear interpolation is performed if *index* is fractional.

The documentation for this class was generated from the following file:

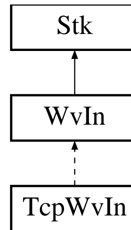
- Table.h

4.64 TcpWvIn Class Reference

STK internet streaming input class.

`#include <TcpWvIn.h>`

Inheritance diagram for TcpWvIn::



Public Methods

- TcpWvIn (int port=2006)

  Default constructor starts a socket server. If not specified, the server is associated with port 2006.

- ~TcpWvIn ()

  Class destructor.

- void listen (unsigned int nChannels=1, Stk::STK_FORMAT format=STK_SINT16)

  Listen for a (new) connection with specified data channels and format.

- bool isConnected (void)

  Returns TRUE is an input connection exists or input data remains in the queue.

- MY_FLOAT lastOut (void) const

  Return the average across the last output sample frame.

- MY_FLOAT tick (void)

  Read out the average across one sample frame of data.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Read out vectorSize averaged sample frames of data in vector.

- const MY_FLOAT* lastFrame (void) const

Return a pointer to the last output sample frame.

- const MY_FLOAT* tickFrame (void)

  Return a pointer to the next sample frame of data.

- MY_FLOAT* tickFrame (MY_FLOAT *frameVector, unsigned int frames)

  Read out sample frames *of data to* frameVector.

4.64.1 Detailed Description

STK internet streaming input class.

This protected Wvin subclass can read streamed data over a network via a TCP socket connection. The data is assumed in big-endian, or network, byte order.

TcpWvIn supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which return samples produced by averaging across sample frames, from the tickFrame() methods, which return pointers to multi-channel sample frames. For single-channel data, these methods return equivalent values.

This class starts a socket server, which waits for a single remote connection. The default data type for the incoming stream is signed 16-bit integers, though any of the defined STK_FORMATs are permissible.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.64.2 Constructor & Destructor Documentation

4.64.2.1 TcpWvIn::TcpWvIn (int *port* = 2006)

Default constructor starts a socket server. If not specified, the server is associated with port 2006.

An StkError will be thrown if an error occurs while initializing the input thread or starting the socket server.

4.64.3 Member Function Documentation

4.64.3.1 void TcpWvIn::listen (unsigned int *nChannels* = 1, Stk::STK_FORMAT *format* = STK_SINT16)

Listen for a (new) connection with specified data channels and format.

An StkError will be thrown a socket error or an invalid function argument.

4.64.3.2 bool TcpWvIn::isConnected (void)

Returns TRUE is an input connection exists or input data remains in the queue.

This method will not return FALSE after an input connection has been closed until all buffered input data has been read out.

The documentation for this class was generated from the following file:

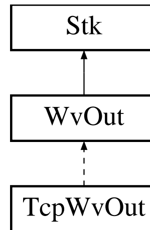
- TcpWvIn.h

4.65 TcpWvOut Class Reference

STK internet streaming output class.

`#include <TcpWvOut.h>`

Inheritance diagram for TcpWvOut::



Public Methods

- TcpWvOut ()

  Default constructor ... the socket is not instantiated.

- TcpWvOut (int port, const char *hostname="localhost", unsigned int nChannels=1, Stk::STK_FORMAT format=STK_SINT16)

  Overloaded constructor which opens a network connection during instantiation.

- ~TcpWvOut ()

  Class destructor.

- void connect (int port, const char *hostname="localhost", unsigned int nChannels=1, Stk::STK_FORMAT format=STK_SINT16)

  Connect to the specified host and port and prepare to stream nChannels *of data in the given data format.*

- void disconnect (void)

  If a connection is open, write out remaining samples in the queue and then disconnect.

- unsigned long getFrames (void) const

  Return the number of sample frames output.

- MY_FLOAT getTime (void) const

  Return the number of seconds of data output.

- void tick (MY_FLOAT sample)

  Output a single sample to all channels in a sample frame.

- void tick (const MY_FLOAT ∗vector, unsigned int vectorSize)

  Output each sample in vector *to all channels in* vectorSize *sample frames.*

- void tickFrame (const MY_FLOAT ∗frameVector, unsigned int frames=1)

  Output the frameVector *of sample frames of the given length.*

4.65.1 Detailed Description

STK internet streaming output class.

This protected WvOut subclass can stream data over a network via a TCP socket connection. The data is converted to big-endian byte order, if necessary, before being transmitted.

TcpWvOut supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which output single samples to all channels in a sample frame, from the tickFrame() method, which takes a pointer to multi-channel sample frame data.

This class connects to a socket server, the port and IP address of which must be specified as constructor arguments. The default data type is signed 16-bit integers but any of the defined STK_FORMATs are permissible.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.65.2 Constructor & Destructor Documentation

4.65.2.1 TcpWvOut::TcpWvOut (int *port*, const char ∗ *hostname* = "localhost", unsigned int *nChannels* = 1, Stk::STK_FORMAT *format* = STK_SINT16)

Overloaded constructor which opens a network connection during instantiation.

An StkError is thrown if a socket error occurs or an invalid argument is specified.

4.65.3 Member Function Documentation

4.65.3.1 void TcpWvOut::connect (int *port*, const char * *hostname* = "localhost", unsigned int *nChannels* = 1, Stk::STK_FORMAT *format* = STK_SINT16)

Connect to the specified host and port and prepare to stream *nChannels* of data in the given data format.

An StkError is thrown if a socket error occurs or an invalid argument is specified.

4.65.3.2 void TcpWvOut::tick (MY_FLOAT *sample*) [virtual]

Output a single sample to all channels in a sample frame.

An StkError is thrown if a socket write error occurs.

Reimplemented from WvOut.

4.65.3.3 void TcpWvOut::tick (const MY_FLOAT * *vector*, unsigned int *vectorSize*) [virtual]

Output each sample in *vector* to all channels in *vectorSize* sample frames.

An StkError is thrown if a socket write error occurs.

Reimplemented from WvOut.

4.65.3.4 void TcpWvOut::tickFrame (const MY_FLOAT * *frameVector*, unsigned int *frames* = 1) [virtual]

Output the *frameVector* of sample frames of the given length.

An StkError is thrown if a socket write error occurs.

Reimplemented from WvOut.

The documentation for this class was generated from the following file:

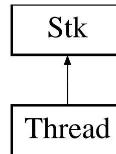
- TcpWvOut.h

4.66 Thread Class Reference

STK thread class.

`#include <Thread.h>`

Inheritance diagram for Thread::



Public Methods

- Thread ()

  Default constructor.

- ~Thread ()

  The class destructor waits indefinitely for the thread to end before returning.

- bool start (THREAD_FUNCTION routine, void *ptr=NULL)

  Begin execution of the thread routine. *Upon success, TRUE is returned.*

- bool wait (long milliseconds=-1)

  Wait the specified number of milliseconds for the thread to terminate. Return TRUE on success.

Static Public Methods

- void test (void)

  Test for a thread cancellation request.

4.66.1 Detailed Description

STK thread class.

This class provides a uniform interface for cross-platform threads. On unix systems, the pthread library is used. Under Windows, the C runtime threadex functions are used.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.66.2 Member Function Documentation

4.66.2.1 bool Thread::start (THREAD_FUNCTION *routine*, void * *ptr* = NULL)

Begin execution of the thread *routine*. Upon success, TRUE is returned.

The thread routine can be passed an argument via *ptr*. If the thread cannot be created, the return value is FALSE.

4.66.2.2 bool Thread::wait (long *milliseconds* = -1)

Wait the specified number of milliseconds for the thread to terminate. Return TRUE on success.

If the specified time value is negative, the function will block indefinitely. Otherwise, the function will block up to a maximum of the specified time. A return value of FALSE indicates the thread did not terminate within the specified time limit.

The documentation for this class was generated from the following file:

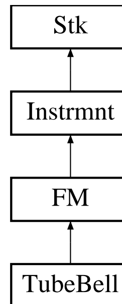
- Thread.h

4.67 TubeBell Class Reference

STK tubular bell (orchestral chime) FM synthesis instrument.

`#include <TubeBell.h>`

Inheritance diagram for TubeBell::



Public Methods

- TubeBell ()
  Class constructor.
- ~TubeBell ()
  Class destructor.
- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)
  Start a note with the given frequency and amplitude.
- MY_FLOAT tick ()
  Compute one output sample.

4.67.1 Detailed Description

STK tubular bell (orchestral chime) FM synthesis instrument.

This class implements two simple FM Pairs summed together, also referred to as algorithm 5 of the TX81Z.

```
Algorithm 5 is :  4->3--\
                         + --> Out
                  2->1--/
```

Control Change Numbers:

- Modulator Index One = 2
- Crossfade of Outputs = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

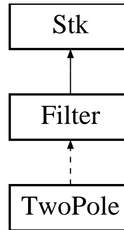
- TubeBell.h

4.68 TwoPole Class Reference

STK two-pole filter class.

`#include <TwoPole.h>`

Inheritance diagram for TwoPole::



Public Methods

- TwoPole ()

  Default constructor creates a second-order pass-through filter.

- ~TwoPole ()

  Class destructor.

- void clear (void)

  Clears the internal states of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setA1 (MY_FLOAT a1)

  Set the a[1] coefficient value.

- void setA2 (MY_FLOAT a2)

  Set the a[2] coefficient value.

- void setResonance (MY_FLOAT frequency, MY_FLOAT radius, bool normalize=FALSE)

  Sets the filter coefficients for a resonance at frequency *(in Hz).*

- void setGain (MY_FLOAT theGain)

  Set the filter gain.

- MY_FLOAT getGain (void) const

  Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT* tick (MY_FLOAT *vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector.

4.68.1 Detailed Description

STK two-pole filter class.

This protected Filter subclass implements a two-pole digital filter. A method is provided for creating a resonance in the frequency response while maintaining a nearly constant filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.68.2 Member Function Documentation

4.68.2.1 void TwoPole::setResonance (MY_FLOAT *frequency*, MY_FLOAT *radius*, bool *normalize* = FALSE)

Sets the filter coefficients for a resonance at *frequency* (in Hz).

This method determines the filter coefficients corresponding to two complex-conjugate poles with the given *frequency* (in Hz) and *radius* from the z-plane origin. If *normalize* is true, the coefficients are then normalized to produce unity gain at *frequency* (the actual maximum filter gain tends to be slightly greater than unity when *radius* is not close to one). The resulting filter frequency response has a resonance at the given *frequency.* The closer the poles are to the unit-circle (*radius* close to one), the narrower the resulting resonance width. An unstable filter will result for *radius* >= 1.0. For a better resonance filter, use a BiQuad filter.

See also:
BiQuad filter class

4.68.2.2 void TwoPole::setGain (MY_FLOAT *theGain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

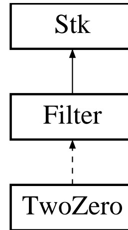
- TwoPole.h

4.69 TwoZero Class Reference

STK two-zero filter class.

`#include <TwoZero.h>`

Inheritance diagram for TwoZero::



Public Methods

- TwoZero ()

  Default constructor creates a second-order pass-through filter.

- ~TwoZero ()

  Class destructor.

- void clear (void)

  Clears the internal states of the filter.

- void setB0 (MY_FLOAT b0)

  Set the b[0] coefficient value.

- void setB1 (MY_FLOAT b1)

  Set the b[1] coefficient value.

- void setB2 (MY_FLOAT b2)

  Set the b[2] coefficient value.

- void setNotch (MY_FLOAT frequency, MY_FLOAT radius)

  Sets the filter coefficients for a "notch" at frequency *(in Hz).*

- void setGain (MY_FLOAT theGain)

  Set the filter gain.

- MY_FLOAT getGain (void) const

Return the current filter gain.

- MY_FLOAT lastOut (void) const

  Return the last computed output value.

- MY_FLOAT tick (MY_FLOAT sample)

  Input one sample to the filter and return one output.

- MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Input vectorSize *samples to the filter and return an equal number of outputs in* vector*.*

4.69.1 Detailed Description

STK two-zero filter class.

This protected Filter subclass implements a two-zero digital filter. A method is provided for creating a "notch" in the frequency response while maintaining a constant filter gain.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.69.2 Member Function Documentation

4.69.2.1 void TwoZero::setNotch (MY_FLOAT *frequency*, MY_FLOAT *radius*)

Sets the filter coefficients for a "notch" at *frequency* (in Hz).

This method determines the filter coefficients corresponding to two complex-conjugate zeros with the given *frequency* (in Hz) and *radius* from the z-plane origin. The coefficients are then normalized to produce a maximum filter gain of one (independent of the filter *gain* parameter). The resulting filter frequency response has a "notch" or anti-resonance at the given *frequency*. The closer the zeros are to the unit-circle (*radius* close to or equal to one), the narrower the resulting notch width.

4.69.2.2 void TwoZero::setGain (MY_FLOAT *theGain*) [virtual]

Set the filter gain.

The gain is applied at the filter input and does not affect the coefficient values. The default gain value is 1.0.

Reimplemented from Filter.

The documentation for this class was generated from the following file:

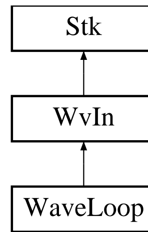
- TwoZero.h

4.70 WaveLoop Class Reference

STK waveform oscillator class.

`#include <WaveLoop.h>`

Inheritance diagram for WaveLoop::



Public Methods

- WaveLoop (const char *fileName, bool raw=FALSE)

  Class constructor.

- virtual ~WaveLoop ()

  Class destructor.

- void setFrequency (MY_FLOAT aFrequency)

  Set the data interpolation rate based on a looping frequency.

- void addTime (MY_FLOAT aTime)

  Increment the read pointer by aTime *samples, modulo file size.*

- void addPhase (MY_FLOAT anAngle)

  Increment current read pointer by anAngle*, relative to a looping frequency.*

- void addPhaseOffset (MY_FLOAT anAngle)

  Add a phase offset to the current read pointer.

- const MY_FLOAT* tickFrame (void)

  Return a pointer to the next sample frame of data.

4.70.1 Detailed Description

STK waveform oscillator class.

This class inherits from WvIn and provides audio file looping functionality.

WaveLoop supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which return samples produced by averaging across sample frames, from the tickFrame() methods, which return pointers to multi-channel sample frames. For single-channel data, these methods return equivalent values.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.70.2 Member Function Documentation

4.70.2.1 void WaveLoop::setFrequency (MY_FLOAT *aFrequency*)

Set the data interpolation rate based on a looping frequency.

This function determines the interpolation rate based on the file size and the current Stk::sampleRate. The *aFrequency* value corresponds to file cycles per second. The frequency can be negative, in which case the loop is read in reverse order.

4.70.2.2 void WaveLoop::addPhase (MY_FLOAT *anAngle*)

Increment current read pointer by *anAngle*, relative to a looping frequency.

This function increments the read pointer based on the file size and the current Stk::sampleRate. The *anAngle* value is a multiple of file size.

4.70.2.3 void WaveLoop::addPhaseOffset (MY_FLOAT *anAngle*)

Add a phase offset to the current read pointer.

This function determines a time offset based on the file size and the current Stk::sampleRate. The *anAngle* value is a multiple of file size.

The documentation for this class was generated from the following file:

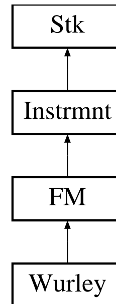
- WaveLoop.h

4.71 Wurley Class Reference

STK Wurlitzer electric piano FM synthesis instrument.

`#include <Wurley.h>`

Inheritance diagram for Wurley::



Public Methods

- Wurley ()

  Class constructor.

- ~Wurley ()

  Class destructor.

- void setFrequency (MY_FLOAT frequency)

  Set instrument parameters for a particular frequency.

- void noteOn (MY_FLOAT frequency, MY_FLOAT amplitude)

  Start a note with the given frequency and amplitude.

- MY_FLOAT tick ()

  Compute one output sample.

4.71.1 Detailed Description

STK Wurlitzer electric piano FM synthesis instrument.

This class implements two simple FM Pairs summed together, also referred to as algorithm 5 of the TX81Z.

```
Algorithm 5 is :  4->3--\
                         + --> Out
                  2->1--/
```

Control Change Numbers:

- Modulator Index One = 2
- Crossfade of Outputs = 4
- LFO Speed = 11
- LFO Depth = 1
- ADSR 2 & 4 Target = 128

The basic Chowning/Stanford FM patent expired in 1995, but there exist follow-on patents, mostly assigned to Yamaha. If you are of the type who should worry about this (making money) worry away.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

The documentation for this class was generated from the following file:

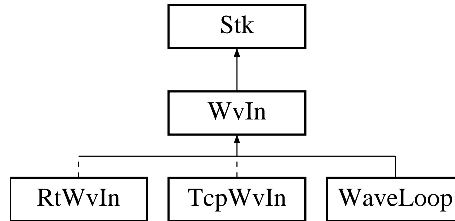
- Wurley.h

4.72 WvIn Class Reference

STK audio data input base class.

`#include <WvIn.h>`

Inheritance diagram for WvIn::



Public Methods

- WvIn ()

  Default constructor.

- WvIn (const char *fileName, bool raw=FALSE)

  Overloaded constructor for file input.

- virtual ∼WvIn ()

  Class destructor.

- void openFile (const char *fileName, bool raw=FALSE)

  Open the specified file and load its data.

- void closeFile (void)

  If a file is open, close it.

- void reset (void)

  Clear outputs and reset time (file pointer) to zero.

- void normalize (void)

  Normalize data to a maximum of +-1.0.

- void normalize (MY_FLOAT peak)

  Normalize data to a maximum of +-peak.

- unsigned long getSize (void) const

Return the file size in sample frames.

- unsigned int getChannels (void) const

  Return the number of audio channels in the file.

- MY_FLOAT getFileRate (void) const

  Return the input file sample rate in Hz (not the data read rate).

- bool isFinished (void) const

  Query whether reading is complete.

- void setRate (MY_FLOAT aRate)

  Set the data read rate in samples. The rate can be negative.

- virtual void addTime (MY_FLOAT aTime)

  Increment the read pointer by aTime *samples.*

- void setInterpolate (bool doInterpolate)

  Turn linear interpolation on/off.

- virtual MY_FLOAT lastOut (void) const

  Return the average across the last output sample frame.

- virtual MY_FLOAT tick (void)

  Read out the average across one sample frame of data.

- virtual MY_FLOAT∗ tick (MY_FLOAT ∗vector, unsigned int vectorSize)

  Read out vectorSize averaged sample frames of data in vector*.*

- virtual const MY_FLOAT∗ lastFrame (void) const

  Return a pointer to the last output sample frame.

- virtual const MY_FLOAT∗ tickFrame (void)

  Return a pointer to the next sample frame of data.

- virtual MY_FLOAT∗ tickFrame (MY_FLOAT ∗frameVector, unsigned int frames)

  Read out sample frames *of data to* frameVector*.*

4.72.1 Detailed Description

STK audio data input base class.

This class provides input support for various audio file formats. It also serves as a base class for "realtime" streaming subclasses.

WvIn loads the contents of an audio file for subsequent output. Linear interpolation is used for fractional "read rates".

WvIn supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which return samples produced by averaging across sample frames, from the tickFrame() methods, which return pointers to multi-channel sample frames. For single-channel data, these methods return equivalent values.

Small files are completely read into local memory during instantiation. Large files are read incrementally from disk. The file size threshold and the increment size values are defined in WvIn.h.

WvIn currently supports WAV, AIFF, SND (AU), MAT-file (Matlab), and STK RAW file formats. Signed integer (8-, 16-, and 32-bit) and floating- point (32- and 64-bit) data types are supported. Uncompressed data types are not supported. If using MAT-files, data should be saved in an array with each data channel filling a matrix row.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.72.2 Constructor & Destructor Documentation

4.72.2.1 WvIn::WvIn (const char * *fileName*, bool *raw* = FALSE)

Overloaded constructor for file input.

An StkError will be thrown if the file is not found, its format is unknown, or a read error occurs.

4.72.3 Member Function Documentation

4.72.3.1 void WvIn::openFile (const char * *fileName*, bool *raw* = FALSE)

Open the specified file and load its data.

An StkError will be thrown if the file is not found, its format is unknown, or a read error occurs.

4.72.3.2 void WvIn::normalize (void)

Normalize data to a maximum of +-1.0.

For large, incrementally loaded files with integer data types, normalization is computed relative to the data type maximum. No normalization is performed for incrementally loaded files with floating-point data types.

4.72.3.3 void WvIn::normalize (MY_FLOAT *peak*)

Normalize data to a maximum of *+-peak*.

For large, incrementally loaded files with integer data types, normalization is computed relative to the data type maximum (*peak/maximum)*. For incrementally loaded files with floating- point data types, direct scaling by *peak* is performed.

4.72.3.4 MY_FLOAT WvIn::getFileRate (void) const

Return the input file sample rate in Hz (not the data read rate).

WAV, SND, and AIF formatted files specify a sample rate in their headers. STK RAW files have a sample rate of 22050 Hz by definition. MAT-files are assumed to have a rate of 44100 Hz.

4.72.3.5 void WvIn::setRate (MY_FLOAT *aRate*)

Set the data read rate in samples. The rate can be negative.

If the rate value is negative, the data is read in reverse order.

4.72.3.6 void WvIn::setInterpolate (bool *doInterpolate*)

Turn linear interpolation on/off.

Interpolation is automatically off when the read rate is an integer value. If interpolation is turned off for a fractional rate, the time index is truncated to an integer value.

4.72.3.7 MY_FLOAT WvIn::tick (void) `[virtual]`

Read out the average across one sample frame of data.

An StkError will be thrown if a file is read incrementally and a read error occurs.

Reimplemented in RtWvIn, and TcpWvIn.

4.72.3.8 MY_FLOAT * WvIn::tick (MY_FLOAT * *vector*, unsigned int *vectorSize*) [virtual]

Read out vectorSize averaged sample frames of data in *vector*.

An StkError will be thrown if a file is read incrementally and a read error occurs.

Reimplemented in RtWvIn, and TcpWvIn.

4.72.3.9 const MY_FLOAT * WvIn::tickFrame (void) [virtual]

Return a pointer to the next sample frame of data.

An StkError will be thrown if a file is read incrementally and a read error occurs.

Reimplemented in RtWvIn, TcpWvIn, and WaveLoop.

4.72.3.10 MY_FLOAT * WvIn::tickFrame (MY_FLOAT * *frameVector*, unsigned int *frames*) [virtual]

Read out sample *frames* of data to *frameVector*.

An StkError will be thrown if a file is read incrementally and a read error occurs.

Reimplemented in RtWvIn, and TcpWvIn.

The documentation for this class was generated from the following file:

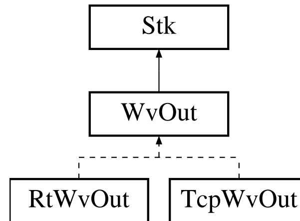
- WvIn.h

4.73 WvOut Class Reference

STK audio data output base class.

`#include <WvOut.h>`

Inheritance diagram for WvOut::



Public Methods

- WvOut ()

  Default constructor.

- WvOut (const char *fileName, unsigned int nChannels=1, FILE_TYPE type=WVOUT_WAV, Stk::STK_FORMAT format=STK_SINT16)

  Overloaded constructor used to specify a file name, type, and data format with this object.

- virtual ~WvOut ()

  Class destructor.

- void openFile (const char *fileName, unsigned int nChannels=1, WvOut::FILE_TYPE type=WVOUT_WAV, Stk::STK_FORMAT format=STK_SINT16)

  Create a file of the specified type and name and output samples to it in the given data format.

- void closeFile (void)

  If a file is open, write out samples in the queue and then close it.

- unsigned long getFrames (void) const

  Return the number of sample frames output.

- MY_FLOAT getTime (void) const

  Return the number of seconds of data output.

- virtual void tick (const MY_FLOAT sample)

    Output a single sample to all channels in a sample frame.

- virtual void tick (const MY_FLOAT *vector, unsigned int vectorSize)

    Output each sample in vector *to all channels in* vectorSize *sample frames.*

- virtual void tickFrame (const MY_FLOAT *frameVector, unsigned int frames=1)

    Output the frameVector *of sample frames of the given length.*

Static Public Attributes

- const FILE_TYPE WVOUT_RAW
- const FILE_TYPE WVOUT_WAV
- const FILE_TYPE WVOUT_SND
- const FILE_TYPE WVOUT_AIF
- const FILE_TYPE WVOUT_MAT

4.73.1 Detailed Description

STK audio data output base class.

This class provides output support for various audio file formats. It also serves as a base class for "realtime" streaming subclasses.

WvOut writes samples to an audio file. It supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which output single samples to all channels in a sample frame, from the tickFrame() method, which takes a pointer to multi-channel sample frame data.

WvOut currently supports WAV, AIFF, AIFC, SND (AU), MAT-file (Matlab), and STK RAW file formats. Signed integer (8-, 16-, and 32-bit) and floating-point (32- and 64-bit) data types are supported. STK RAW files use 16-bit integers by definition. MAT-files will always be written as 64-bit floats. If a data type specification does not match the specified file type, the data type will automatically be modified. Uncompressed data types are not supported.

Currently, WvOut is non-interpolating and the output rate is always Stk::sampleRate().

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

4.73.2 Constructor & Destructor Documentation

4.73.2.1 WvOut::WvOut (const char * *fileName*, unsigned int *nChannels* = 1, FILE_TYPE *type* = WVOUT_WAV, Stk::STK_FORMAT *format* = STK_SINT16)

Overloaded constructor used to specify a file name, type, and data format with this object.

An StkError is thrown for invalid argument values or if an error occurs when initializing the output file.

4.73.3 Member Function Documentation

4.73.3.1 void WvOut::openFile (const char * *fileName*, unsigned int *nChannels* = 1, WvOut::FILE_TYPE *type* = WVOUT_WAV, Stk::STK_FORMAT *format* = STK_SINT16)

Create a file of the specified type and name and output samples to it in the given data format.

An StkError is thrown for invalid argument values or if an error occurs when initializing the output file.

4.73.3.2 void WvOut::tick (const MY_FLOAT *sample*) `[virtual]`

Output a single sample to all channels in a sample frame.

An StkError is thrown if a file read error occurs.

Reimplemented in RtWvOut, and TcpWvOut.

4.73.3.3 void WvOut::tick (const MY_FLOAT * *vector*, unsigned int *vectorSize*) `[virtual]`

Output each sample in *vector* to all channels in *vectorSize* sample frames.

An StkError is thrown if a file read error occurs.

Reimplemented in RtWvOut, and TcpWvOut.

4.73.3.4 void WvOut::tickFrame (const MY_FLOAT * *frameVector*, unsigned int *frames* = 1) `[virtual]`

Output the *frameVector* of sample frames of the given length.

An StkError is thrown if a file read error occurs.

Reimplemented in RtWvOut, and TcpWvOut.

4.73.4 Member Data Documentation

4.73.4.1 const FILE_TYPE WvOut::WVOUT_RAW [static]

STK RAW file type.

4.73.4.2 const FILE_TYPE WvOut::WVOUT_WAV [static]

WAV file type.

4.73.4.3 const FILE_TYPE WvOut::WVOUT_SND [static]

SND (AU) file type.

4.73.4.4 const FILE_TYPE WvOut::WVOUT_AIF [static]

AIFF file type.

4.73.4.5 const FILE_TYPE WvOut::WVOUT_MAT [static]

Matlab MAT-file type.

The documentation for this class was generated from the following file:

- WvOut.h

Chapter 5

STK Page Documentation

5.1 General Information

References

- `ICMC99 Paper`

  A somewhat recent paper by Perry and Gary about the Synthesis ToolKit in C++.
- `SIGGRAPH96 Paper`

  A not-so-recent paper by Perry about the Synthesis ToolKit in C++.
- `Perry's STK Web Page`

  This is a link to Perry Cook's STK Web page. He has information about the Synthesis toolKit Instrument Network Interface (SKINI), the protocol used to control STK instruments, as well as a lot of other cool stuff.

What is the *Synthesis ToolKit*?

The Synthesis ToolKit in C++ (STK) is a set of audio signal processing and synthesis classes and algorithms written in C++. You can use these classes to create programs that make sounds with a variety of synthesis techniques. This is not a terribly novel concept, except that the Synthesis ToolKit is extremely portable (it's mostly platform-independent C and C++ code), and it's completely user-extensible (no libraries, no hidden drivers, and all source code is included). We like to think that this increases the chances that our programs will still work in another 5-10 years. In fact, the ToolKit has been working continuously for nearly 8 years now. STK currently runs with "realtime" support (audio and MIDI) on SGI (Irix), Linux, and Windows computer platforms.

Generic, non-realtime support has been tested under NeXTStep, Sun, and other platforms and should work with any standard C++ compiler.

The Synthesis ToolKit is free for non-commercial use. The only parts of the Synthesis ToolKit that are platform-dependent concern real-time audio and MIDI input and output, and that is taken care of with a few special classes. The interface for MIDI input and the simple `Tcl/Tk` graphical user interfaces (GUIs) provided is the same, so it's easy to experiment in real time using either the GUIs or MIDI. The Synthesis ToolKit can generate simultaneous SND (AU), WAV, AIFF, and MAT-file output soundfile formats (as well as realtime sound output), so you can view your results using one of a large variety of sound/signal analysis tools already available (e.g. `Snd`, Cool Edit, Matlab).

What the *Synthesis ToolKit* is not.

The Synthesis Toolkit is not one particular program. Rather, it is a set of C++ classes that you can use to create your own programs. A few example applications are provided to demonstrate some of the ways to use the classes. If you have specific needs, you will probably have to either modify the example programs or write a new program altogether. Further, the example programs don't have a fancy GUI wrapper. If you feel the need to have a "drag and drop" graphical patching GUI, you probably don't want to use the ToolKit. Spending hundreds of hours making platform-dependent graphics code would go against one of the fundamental design goals of the ToolKit - platform independence.

For those instances where a simple GUI with sliders and buttons is helpful, we use `Tcl/Tk` (which is freely distributed for all the supported ToolKit platforms). A number of Tcl/Tk GUI scripts are distributed with the ToolKit release. For control, the Synthesis Toolkit uses raw MIDI (on supported platforms), and SKINI (Synthesis ToolKit Instrument Network Interface, a MIDI-like text message synthesis control format).

A brief history of the *Synthesis ToolKit in C++*.

Perry Cook began developing a pre-cursor to the Synthesis ToolKit (also called STK) under NeXTStep at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University in the early-1990s. With his move to Princeton University in 1996, he ported everything to C++ on SGI hardware, added real-time capabilities, and greatly expanded the synthesis techniques available. With the help of Bill Putnam, Perry also made a port of STK to Windows95. Gary Scavone began using STK extensively in the summer of 1997 and completed a full port of STK to Linux early in 1998. He finished the fully compatable Windows port (using Direct Sound API) in June 1998. Numerous improvements and extensions have been made since then.

The Toolkit has been distributed continuously since 1996 via the `Princeton`

`Sound Kitchen`, `Perry Cook's home page` at Princeton, `Gary Scavone's home page` at Stanford's Center for Computer Research in Music and Acoustics (CCRMA), and the `Synthesis ToolKit home page`. The ToolKit has been in included in various collections of software. Much of it has also been ported to MAX/MSP on Macintosh computers by Dan Trueman and Luke Dubois of Columbia University, and is distributed as `PeRColate`. Help on real-time sound and MIDI has been provided by Tim Stilson, Bill Putnam, and Gabriel Maldonado.

Legal and Ethical Notes

This software was designed and created to be made publicly available for free, primarily for academic purposes, so if you use it, pass it on with this documentation, and for free. If you make a million dollars with it, give us some. If you make compositions with it, put us in the program notes.

Some of the concepts are covered by various patents, some known to us and likely others which are unknown. Many of the ones known to us are administered by the Stanford Office of Technology and Licensing. The good news is that large hunks of the techniques used here are public domain. To avoid subtle legal issues, we will not state what's freely useable here, but we will try to note within the various classes where certain things are likely to be protected by patents.

Disclaimer

STK is free and we do not guarantee anything. We've been hacking on this code for a while now and most of it seems to work pretty well. But, there surely are some bugs floating around. Sometimes things work fine on one computer platform but not so fine on another. FPU overflows and underflows cause *very* weird behavior which also depends on the particular CPU and OS. Let us know about bugs you find and we'll do our best to correct them.

5.2 Class Documentation

- Class Hierarchy
- Class/Enum List
- File List
- Compound Members

5.3 Download and Release Notes

Version 4.0, 30 April 2002

`STK Version 4.0: Source distribution` (1.64 MB tar/gzipped)

`STK Version 4.0: Source distribution with precompiled windows binaries` (2.26 MB tar/gzipped)

5.4 Release Notes:

5.4.1 Version 4.0

- New documentation and tutorial.
- Several new instruments, including Saxofony, BlowBotl, and StifKarp.
- New Stk base class, replacing Object class.
- New Filter class structure and methods.
- Extensive modifications to WvIn and WvOut class structures and methods.
- Looping functionality moved to WaveLoop (subclass of WvIn).
- Automatic file type detection in WvIn ... hosed WavWvIn, AifWvIn, RawWavIn, SndWavIn, and MatWvIn subclasses.
- New file type specifier argument in WvOut ... hosed WavWvOut, AifWvOut, RawWavOut, SndWavOut, and MatWvOut subclasses.
- Some simplifications of Messager class (was Controller).
- New independent RtAudio class.
- Extensive revisions in code and a significant number of API changes.

5.4.2 Version 3.2

- New input control handling class (Controller)
- Added AIFF file input/output support.
- New C++ error handling capabilities.
- New input/output internet streaming support (StrmWvIn/StrmWvOut).
- Added native ALSA support for linux.
- Added optional "device" argument to all "Rt" classes (audio and MIDI) and printout of devices when argument is invalid.
- WvIn classes rewritten to support very big files (incremental load from disk).
- Changed WvIn/WvOut classes to work with sample frame buffers.
- Fixed looping and negative rate calculations in WvIn classes.
- Fixed interpolation bug in RtWvIn.

- Windoze RtAudio code rewritten (thank Dave!).
- Simplified byte-swapping functions (in-place swapping).
- "Stereo-ized" RagaMatic.
- Miscellaneous renamings.
- Probably a bunch more fixes that I've long since forgotten about.

5.4.3 Version 3.1

- New RagaMatic project ... very cool!!!
- Less clipping in the Shakers class.
- Added "microphone position" to Mandolin in STKdemo.
- Fixed MIDI system message exclusion under Irix.
- Added a few bitmaps for the Shaker instruments.
- Made destructors virtual for Reverb.h, WvIn.h and Simple.h.
- Fixed bug setting delay length in DLineA when value too big.
- Fixed bug in WinMM realtime code (RTSoundIO).
- Added tick() method to BowTabl, JetTabl, and ReedTabl (same as lookup).
- Switched to pthread API on SGI platforms.
- Added some defines to Object.h for random number generation, FPU overflow checking, etc....
- A few minor changes, some bug fixes ... can't remember all of them.

5.4.4 Version 3.0

- New define flags for OS and realtime dependencies (this will probably cause problems for old personal STK code, but it was necessary to make future ports easier).
- Expanded and cleaned the Shakers class.
- New BowedBar algorithm/class.
- Fixed Linux MIDI input bug.
- Fixed MIDI status masking problem in Windows.
- OS type defines now in Makefile.
- New RAWWAVE_PATH define in Object.h.
- Syntmono project pulled out to separate directory and cleaned up.
- Socketing capabilities under Unix, as well as Windoze.
- Multiple simultaneous socket client connections to STK servers now possible.
- MD2SKINI now can merge MIDI and piped messages under Irix and Linux (for TCL->MD2SKINI->syntmono control).
- Defined INT16 and INT32 types and fixed various WvIn and WvOut classes.

- Updated MatWvIn and MatWvOut for new MAT-file documentation from Matlab.
- New demo Tcl/Tk GUI (TclDemo.tcl).
- Minor fixes to FM behavior.
- Added record/duplex capabilities to RTSoundIO (Linux, SGI, and Windoze).
- Fixed bugs in WavWvOut and MatWvOut header specifications.
- Added RawWvOut class.
- New WvIn class with RawWvIn, SndWvIn, WavWvIn, MatWvIn, and RTWvIn subclasses.
- Removed RawWave, RawShot, RawInterp, and RawLoop classes (supplanted by RawWvIn).
- Multi-channel data support in WvIn and WvOut classes using MY_MULTI data type (pointer to MY_FLOAT) and the methods mtick() and mlastOutput().
- Now writing to primary buffer under Windoze when allowed by hardware.
- Cleaned up Object.h a bit.
- Pulled various utility and thread functions out of syntmono.cpp (to aid readability of the code).

5.4.5 Version 2.02

- Created RawWave abstract class, with subclasses of RawLoop (looping rawwave oscillator), RawShot (non-looping, non-interpolating rawwave player ... used to be RawWvIn), and RawInterp (looping or non-looping, interpolating rawwave player ... used to be RawWave).
- Modified DrumSynt to correctly handle sample rates different than 22050 Hz.
- Modified syntmono parsing vs. tick routine so that some ticking occurs between each message. When multiple messages are waiting to be processed, the time between message updates is inversely proportional to the number of messages in the buffer.
- Fixed DirectSound playback bug in WinXX distribution. Sound was being played at 8-bit, 22 kHz in all cases. Playback is now 16-bit and dependent on SRATE.
- Fixed bug in MD2SKINI which prevented some NoteOff statements from being output.
- This distribution includes an example STK project, mus151, which demonstrates a means for keeping a user's personal projects separate from the main distribution. This is highly recommended, in order to simplify upgrades to future STK releases.

5.4.6 Version 2

- Unification of the capabilities of STK across the various platforms. All of the previous SGI functionality has been ported to Linux and Windows, including realtime sound output and MIDI input.
- MIDI input (with optional time-stamping) supported on SGI, Linux (OSS device drivers only), and Windows operating systems. Time stamping under IRIX and Windows is quantized to milliseconds and under Linux to hundredths of a second.
- Various Sound Output Options - .wav, .snd, and .mat (Matlab MAT-file) soundfile outputs are supported on all operating systems. I hacked out the MAT-file structure, so you don't have to include any platform-specific libraries. Realtime sound output is provided as well, except under NeXTStep.
- Multiple Reverberator Implementations - Reverb subclasses of JCRev and NRev (popular reverberator implementations from CCRMA) have been written. Perry's original reverb implementation still exists as PRCRev. All reverberators now take a T60 initializer argument.
- MD2SKINI - A program which parses a MIDI input stream and spits out SKINI code. The output of MD2SKINI is typically piped into an STK instrument executable (eg. <FONT FACE="Geneva">MD2SKINI | syntmono Clarinet -r -i</FONT>). In addition, you can supply a filename argument to MD2SKINI and have it simultaneously record a SKINI score file for future reuse.
- Modifications to *Object.h* for OS_TYPE compilation dependencies. *Makefile* automatically determines OS_TYPE when invoked (if you have the GNU makefile utilities installed on your system).
- A single distribution for all platforms. The Unix and Windows versions have been merged into a single set of classes. Makefiles and Visual C++ workspace/project files are provided for compiling.

5.5 Usage Documentation

- Directory Structure:
- Compiling:
- Control Data:
- Demo: STK Instruments
- Demo: Non-Realtime Use
- Demo: Realtime Use
- Realtime Control Input using Tcl/Tk Graphical User Interfaces:
- Realtime MIDI Control Input:

5.6 Directory Structure:

The top level distribution contains the following directories:

- The **src** directory contains the source .cpp files for almost all the STK unit generator and algorithm classes.
- The **include** directory contains the header files for almost all the STK unit generator and algorithm classes.
- The **rawwaves** directory contains various raw, monophonic, 16-bit, big-endian soundfiles used with the STK classes.
- The **doc** directory contains documentation about STK.
- The **projects** directory contains various demo and example STK programs.

This release of STK comes with four separate "project" directories:

1. The **demo** project is used to demonstrate nearly all of the STK instruments. The **demo** program has been written to allow a variety of control input and sound data output options. Simple graphical user interfaces (GUIs) are also provided.

2. The **effects** project demonstrates realtime duplex mode (simultaneous audio input and output) operation, when available, as well as various delay-line based effects algorithms.

3. The **ragamatic** project is just cool. Fire it up and be enlightened.

4. The **examples** project contains several simple programs which demonstrate audio input/output, as well as the use of the audio internet streaming classes.

5.7 Compiling:

- **Generic (non-realtime):** Most STK classes are operating system *independent* and can be compiled using any current C++ compiler. STK assumes big-endian host byte order by default, so if your system is little-endian (i.e. Intel processor), you must provide the `__LITTLE_ENDIAN__` preprocessor definition to your compiler. The **demo** project will compile without realtime support, allowing the use of SKINI scorefiles for input control and output to a variety of soundfile formats. The following classes *cannot* be used without realtime support: RtAudio, RtWvIn, RtWvOut, RtDuplex, RtMidi, Socket, Thread, TcpWvIn, TcpWvOut. Because of this, it is not possible to compile the **effects**, **ragamatic**, and most of the **examples** projects for non-realtime use.
- **Linux:** Realtime support is enabled with either the `__LINUX_OSS__` or `__LINUX_ALSA__` preprocessor definitions, which are used to select the underlying audio/MIDI system API. Realtime programs must also link with the `pthread` library. When using the ALSA API, it is also necessary to link with the `asound` library. In addition, the `__LITTLE_ENDIAN__` preprocessor definition is necessary if compiling on a little-endian system. Assuming your system has the `GNU` Makefile utilities installed, typing `make` within a particular project directory will initiate the compilation process. The `Makefile` will have to be modified to change the default audio/MIDI system API and for big-endian processors. Special support exists under Linux for the MIDIator serial MIDI device, enabled using the __MIDIATOR__ preprocessor definition (together with either the __LINUX_ALSA__ or __LINUX_OSS__ definitions). See the README-Linux file for further system configuration information.
- **SGI:** Realtime support is enabled with the `__IRIX_AL__` preprocessor definition and linkage with the `audio`, `md`, and `pthread` libraries. If your system has the `GNU` Makefile utilities installed, typing `make` (or `gmake`) within a particular project directory will initiate the compilation process. If your system does not have the GNU Makefile utilities, you should first try to download and install them. If this is not possible, a generic Makefile is provided with the **demo** project (Makefile.sgi). It can be invoked by typing `make -f Makefile.sgi` within that project directory. STK 4.0 is confirmed to compile using CC version 7.30. There may be problems with old compiler versions.
- **Windows95/98/2000/XP:** Realtime support is enabled with the `__WINDOWS_DS__` preprocessor definition and linkage with the `dsound.lib`, `winmm.lib`, and `Wsock32.lib` libraries. In addition, the `__LITTLE_ENDIAN__` preprocessor definition is necessary for all Windows systems. A distribution of the release is available with precompiled binaries for all the projects. In order for these binaries to function properly, your system must have the DirectX 5.0 (or higher) runtime libraries installed (available from `Microsoft`). Further, the **effects** project requires that your sound-

card and drivers provide full duplex mode capabilities. Visual C++ 6.0 project file are provided in each project directory as well should you wish to compile your own binaries. It is important to link with the non-debug libraries when compiling "release" program versions and debug libraries when compiling "debug" program versions.

- **WindowsNT:** I've given up trying to make things work under NT. You'll have to switch to Windows 2000 (which does seem to work).

5.8 Control Data:

All STK programs in this distribution take input control data in the form of `SKINI` or MIDI messages only. The Messager class unifies the various means of acquiring control data under a single, easy to use set of functions. The way that SKINI messages can be sent to the programs is dependent upon the operating system in use, as well as whether the program is running in realtime or not. In general, it is possible to:

1. Redirect or pipe SKINI scorefiles to an executable.
2. Pipe realtime SKINI input messages to an executable (not possible under Windows95/98).
3. Socket realtime SKINI input messages to an executable.
4. Acquire realtime MIDI messages from a MIDI port on your computer.

`Tcl/Tk` graphical user interfaces (GUI) are provided with this distribution which can generate realtime SKINI messages. Note that the Messager class allows multiple simultaneous socket client connections, together with MIDI and/or piped input. The **Md2Skini** program (in the **demo** directory) is mostly obsolete but can be used to create SKINI scorefiles from realtime MIDI input.

5.9 Demo: STK Instruments

The **demo** project demonstrates the behavior of all the distributed STK instruments. The instruments available with this release include:

- Clarinet: Pretty good physical model of the clarinet
- BlowHole: A clarinet physical model with one tonehole and one register vent
- Saxofony: A psuedo-conical bore reed instrument which sometimes sounds like a saxophone
- Flute: Pretty good physical model of the flute

- Brass: Not so bad physical model of a brass instrument
- BlowBotl: A basic helmholtz resonator and air jet model
- Bowed: Not hideous physical model of a bowed string instrument
- Plucked: Yer basic plucked string physical model
- StifKarp: A simple plucked, stiff string physical model
- Sitar: A simple sitar/plucked string physical model
- Mandolin: Two-string mandolin physical model
- Rhodey: Rhodes-like electric piano FM synthesis model
- Wurley: Wurlitzer-like electric piano FM synthesis model
- TubeBell: FM synthesis model
- HevyMetl: Distorted synthesizer FM synthesis model
- PercFlut: Percussive flute-like FM synthesis model
- BeeThree: Cheezy organ FM synthesis model
- Moog: Swept filter sampler
- FMVoices: Three-formant FM voice synthesis
- Resonate: Noise through a BiQuad filter
- Drummer: Sampling synthesis
- BandedWG: Banded waveguide meta-object for bowed bars, tibetan bowls, etc.
- Shakers: Various stochastic event models of shaker instruments
- ModalBar: Various four-resonance presets (marimba, vibraphone, etc...)
- Mesh2D: Two-dimensional, rectilinear digital waveguide mesh

5.10 Demo: Non-Realtime Use

See the information above with respect to compiling STK for non-realtime use.

In non-realtime mode, it is assumed that input control messages are provided from a SKINI scorefile and that audio output is written to a soundfile (.snd, .wav, .aif, .mat, .raw). A number of SKINI scorefiles are provided in the *scores* directory of the **demo** project. Assuming a successful compilation of the **demo** program, typing:

```
cat scores/bookert.ski | demo BeeThree -w myfile.wav
```

or (on WindowsXX and/or Unix)

```
demo BeeThree -w myfile.wav < scores\bookert.ski
```

from the **demo** directory will play the scorefile *bookert.ski* using the STK BeeThree instrument and write the resulting audio data to a WAV formatted soundfile called "myfile.wav". Typing `demo` without any arguments will provide a full program usage description.

5.11 Demo: Realtime Use

STK realtime audio and MIDI input/output and realtime SKINI control input via socketing support is provided for Linux, SGI, and Windows95/98/2000/XP operating systems. STK realtime SKINI control input via piping is possible under Linux, SGI, and Windows2000/XP only.

Control input and audio output options are typically specified as command-line arguments to STK programs. For example, the **demo** program is invoked as:

```
demo instrument flags
```

where instruments include those described above and flags can be any or all of:

- *-or* for realtime audio output,
- *-ow* <*file name*> for WAV soundfile output,
- *-os* <*file name*> for SND (AU) soundfile output,
- *-om* <*file name*> for MAT-file output,
- *-ip* or *-is* for realtime SKINI control input via piping or socketing, respectively,
- *-im* <*file name*> for MIDI control input

The <-ip> and <-is> flags must be used when piping or socketing realtime SKINI control data to an STK program. The <-im> flag must be used to read MIDI control input from your MIDI port. Note that you can use all three input types simultaneously.

Assuming a successful compilation of the **demo** program, typing:

```
cat scores/bookert.ski | demo BeeThree -or
```

or (on WindowsXX and/or Unix)

```
demo BeeThree -or < scores\bookert.ski
```

from the **demo** directory will play the scorefile *bookert.ski* using the STK BeeThree instrument and stream the resulting audio data in realtime to the audio output channel of your computer. Typing `demo` without any arguments will provide a full program usage description.

5.12 Realtime Control Input using Tcl/Tk Graphical User Interfaces:

There are a number of `Tcl/Tk` GUIs supplied with the STK projects. These scripts require Tcl/Tk version 8.0 or later, which can be downloaded for free

over the WWW. On Unix and Windows2000/XP platforms, you can run the various executable scripts (e.g. StkDemo.bat) provided with each project to start everything up (you may need to symbolically link the wish80 executable to the name *wish*). The PhysicalDemo script just implements the following command-line sequence:

```
wish < tcl/Physical.tcl | demo Clarinet -or -ip
```

On WindowsXX and Unix platforms, the following operations are necessary to establish a socket connection between the Tcl/Tk GUI and the STK program:

1. Open a DOS shell and start the STK program with the *-is* flag (ex. **demo Clarinet -or -is**).
2. Open the Tcl/Tk GUI (e.g. tcl/Physical.tcl) by double-clicking on it, or type `wish < tcl/Physical.tcl` in another DOS shell.
3. Establish the socket connection by selecting *Socket* under the Communications menu item in the Tcl/Tk GUI.

Note that it is possible to specify a hostname when establishing the socket connection from the socket client. Thus, the STK socket server program and the Tcl/Tk GUI need not necessarily reside on the same computer.

5.13 Realtime MIDI Control Input:

On all supported realtime platforms, you can direct realtime MIDI input to the STK Clarinet by typing:

```
demo Clarinet -or -im
```

5.14 The Mail List

An STK mailing list has been set up to facilitate communication among STK users. Subscribing to this list is your best way of keeping on top of new releases, bug fixes, and various user developments.

To join send a message to <stk-request@ccrma.stanford.edu> with the contents: `subscribe`

To be removed from the list send a message to <stk-request@ccrma.stanford.edu> with the contents: `unsubscribe`

5.15 System Requirements

General

- A MIDI interface to use MIDI input controls. (NOTE: This may be built into the soundcard on your computer.)
- `Tcl/Tk` version 8.0 or higher to use the simple Tcl/Tk GUIs provided with the STK distribution (available free over the WWW for all supported realtime platforms).

Linux (specific)

- A soundcard to use realtime audio input/output capabilities. In order to use the **effects** project, the soundcard and drivers must support full duplex mode.
- `OSS` or `ALSA` device drivers for realtime sound output and MIDI input.

Windows95/98/2000/XP (specific)

- A soundcard to use realtime audio input/output capabilities. In order to use the **effects** project, the soundcard and drivers must support full duplex mode.
- `DirectX` 5.0 (or higher) runtime libraries to use the precompiled binaries.
- Visual C++ 6.0 for compiling (though a precompiled distribution is available).
- For compiling the source (if not already in your system):
  - `dsound.h` header file (DirectX 6.1) - put somewhere in your header search path
  - `dsound.lib` library file (DirectX 6.1) - put somewhere in your library search path

WindowsNT (specific)

- STK is no longer supported under WindowsNT because DirectX support for NT is minimal. Unless DirectX 5.0 or higher becomes available for NT, STK won't work.

5.16 Tutorial

- Introduction
- Getting Started
- Compiling
- "Realtime" vs. "Non-Realtime"

5.17 Introduction

First and foremost, the Synthesis ToolKit is a set of C++ classes. That means you need to know some basics about programming in C++ to make use of STK (beyond the example programs we provide). STK's "target audience" is people who:

- already know how to program in C and C++
- want to create audio DSP and/or synthesis programs
- want to save some time by using our unit generators and input/output routines
- know C, but want to learn about synthesis and processing algorithms
- wish to teach real-time synthesis and processing, and wish to use some of our classes and examples

Most ToolKit programmers will likely end up writing a class or two for their own particular needs, but this task is typically simplified by making use of pre-existing STK classes (filters, oscillators, etc.).

5.18 Getting Started

We'll begin our introduction to the Synthesis ToolKit with a simple sine-wave oscillator program. STK doesn't provide a specific oscillator for sine waves. Instead, it provides a generic waveform oscillator class, WaveLoop, which can load a variety of common file types. In this example, we load a sine "table" from an STK RAW file. The class RtWvOut will send "realtime" samples to the audio output hardware on your computer.

```
// sineosc.cpp

#include "WaveLoop.h"
#include "RtWvOut.h"

int main()
{
  // Set the global sample rate before creating class instances.
```

```
  Stk::setSampleRate( 44100.0 );

  // Define and load the sine wave file
  WaveLoop *input = new WaveLoop("sinewave.raw", TRUE);
  input->setFrequency(440.0);

  // Define and open the default realtime output device for one-channel playback
  RtWvOut *output = new RtWvOut(1);

  // Play the oscillator for 40000 samples
  for (int i=0; i<40000; i++) {
    output->tick( input->tick() );
  }

  // Clean up
  delete input;
  delete output;

  return 0;
}
```

WaveLoop is a subclass of WvIn, which supports WAV, SND (AU), AIFF, MAT-file (Matlab), and RAW file formats with 8-, 16-, and 32-bit integer and 32- and 64-bit floating-point data types. WvIn provides interpolating, read once ("oneshot") functionality, as well as methods for setting the read rate and read position.

Nearly all STK classes implement tick() methods which take and/or return sample values. Within the tick() method, the fundamental sample calculations are performed for a given class. Most STK classes consume/generate a single sample per operation and their tick() method takes/returns each sample "by value". In addition, every class implementing a tick() method also provides an overloaded tick() function taking pointer and size arguments which can be used for vectorized computations.

The WvIn and WvOut classes support multi-channel sample frames. To distinguish single-sample frame operations from multi-channel frame operations, these classes also implement tickFrame() functions. When a tick() method is called for multi-channel data, frame averages are returned or the input sample is distributed across all channels of a sample frame.

Nearly all STK classes inherit from the Stk base class. Stk provides a static sample rate which is queried by subclasses as needed. Because many classes use the current sample rate value during instantiation, it is important that the desired value be set at the beginning of a program. The default STK sample rate is 22050 Hz.

Another primary concept that is somewhat obscurred in this example concerns the data format in which sample values are passed and received. Audio and control signals throughout STK use a floating-point data type, the exact precision of which can be controlled via the MY_FLOAT #define statement in Stk.h.

Thus, the ToolKit can use any normalization scheme desired. The base instruments and algorithms are implemented with a general audio sample dynamic maximum of +/-1.0, and the WvIn and WvOut classes and subclasses scale appropriately for DAC or soundfile input and output.

Finally, STK has some basic C++ error handling functionality built in. Classes which access files and/or hardware are most prone to runtime errors. To properly "catch" such errors, the above example should be rewritten as shown below.

```
// sineosc.cpp

#include "WaveLoop.h"
#include "RtWvOut.h"

int main()
{
  // Set the global sample rate before creating class instances.
  Stk::setSampleRate( 44100.0 );

  WaveLoop *input = 0;
  RtWvOut *output = 0;

  try {
    // Define and load the sine wave file
    input = new WaveLoop( "sinewave.raw", TRUE );

    // Define and open the default realtime output device for one-channel playback
    output = new RtWvOut(1);
  }
  catch (StkError &) {
    goto cleanup;
  }

  input->setFrequency(440.0);

  // Play the oscillator for 40000 samples
  for (int i=0; i<40000; i++) {
    try {
      output->tick(input->tick());
    }
    catch (StkError &) {
      goto cleanup;
    }
  }

 cleanup:
  delete input;
  delete output;

  return 0;
}
```

In this particular case, we simply exit the program if an error occurs (an error message is automatically printed to stderr). A more refined program might

attempt to recover from or fix a particular problem and, if successful, continue processing.

5.19 Compiling

5.19.1 Linux

In general, you will probably want to use a `Makefile` for your STK programs and projects. For this particular program, however, the following will suffice (on a linux system):

```
g++ -Wall -D__LINUX_OSS__ -D__LITTLE_ENDIAN__ -o sineosc Stk.cpp WvIn.cpp WaveLoop.cpp WvOut.cpp RtWvOut.cp
```

This assumes you've set up a directory that includes the files `sineosc.cpp`, the rawwave file `sinewave.raw`, and the header and source files for the classes Stk, WvIn, WaveLoop, WvOut, RtWvOut, and RtAudio. There are other, more convenient, means for structuring projects that will be discussed later.

Most linux systems currently come installed with the OSS audio hardware drivers. If your system instead has ALSA audio drivers installed and you wish to make use of native ALSA API calls, a link to the ALSA library must be specified in the above compile statement (`-lasound`) and the preprocessor definition should instead be `__LINUX_ALSA__`.

5.19.2 Irix

The irix (SGI) and linux operating systems are both flavors of unix and thus behave similarly. Making the same assumptions as in the linux case, the following compile statement should work:

```
CC -Wall -D__IRIX_AL__ -o sineosc Stk.cpp WvIn.cpp WaveLoop.cpp WvOut.cpp RtWvOut.cpp RtAudio.cpp sineosc.c
```

5.19.3 Windows

I have personally only worked with Visual C++ when compiling programs under windoze. I'll assume you've become familiar with Visual C+ and don't need a tutorial on its particular idiosyncrasies. In creating the VC++ project, add the Stk, WvIn, WaveLoop, WvOut, RtWvOut, and RtAudio class files, as well as the `sineosc.cpp` and `sinewave.raw` files. You will also need to link to the DirectSound library (`dsound.lib`), select the multithreaded library, and provide the `__WINDOWS_DS__` and `__LITTLE_ENDIAN__` preprocessor definitions.

5.20 "Realtime" vs. "Non-Realtime"

Most of the Synthesis ToolKit classes are platform independent. That means that they should compile on any reasonably current C++ compiler. The functionality needed for realtime audio and MIDI input/output, as well as realtime control message acquistion, is inherently platform and operating-system (OS) *dependent*. STK classes which require specific platform/OS support include RtAudio, RtWvOut, RtWvIn, RtDuplex, RtMidi, TcpWvIn, TcpWvOut, Socket, and Thread. These classes currently can only be compiled on Linux, Irix, and Windows (except Windows NT) systems using the `__LINUX_OSS__`, `__LINUX_ALSA__`, `__IRIX_AL__`, or `__WINDOWS_DS__` preprocessor definitions.

Without the "realtime" classes, it is still possible to read SKINI scorefiles for control input and to read and write to/from a variety of audio file formats (WAV, SND, AIFF, MAT-file, and RAW). If compiling for a "little-endian" host processor, the `__LITTLE_ENDIAN__` preprocessor definition should be provided.

5.21 To Be Continued ...

5.22 Synthesis toolKit Instrument Network Interface (SKINI)

This describes the latest (version 1.1) implementation of SKINI for the Synthesis Toolkit in C++ (STK) by Perry R. Cook.

```
      Too good to be true?
   Have control and read it too?
        A SKINI haiku.
```

Profound thanks to Dan Trueman, Brad Garton, and Gary Scavone for input on this revision. Thanks also to MIDI, the NeXT MusicKit, ZIPI and all the creators and modifiers of these for good bases upon/from which to build and depart.

5.23 MIDI Compatibility

SKINI was designed to be MIDI compatible wherever possible, and extend MIDI in incremental, then maybe profound ways.

Differences from MIDI, and motivations, include:

- Text-based messages are used, with meaningful names wherever possible. This allows any language or system capable of formatted printing to generate SKINI. Similarly, any system capable of reading in a string and turning delimited fields into strings, floats, and ints can consume SKINI for control. More importantly, humans can actually read, and even write if they want, SKINI files and streams. Use an editor and search/replace or macros to change a channel or control number. Load a SKINI score into a spread sheet to apply transformations to time, control parameters, MIDI velocities, etc. Put a monkey on a special typewriter and get your next great work. Life's too short to debug bit/nybble packed variable length mumble messages. Disk space gets cheaper, available bandwidth increases, music takes up so little space and bandwidth compared to video and grapics. Live a little.

- Floating point numbers are used wherever possible. Note Numbers, Velocities, Controller Values, and Delta and Absolute Times are all represented and scanned as ASCII double-precision floats. MIDI byte values are preserved, so that incoming MIDI bytes from an interface can be put directly into SKINI messages. 60.0 or 60 is middle C, 127.0 or 127 is maximum velocity etc. But, unlike MIDI, 60.5 can cause a 50cent sharp middle C to be played. As with MIDI byte values like velocity, use of the integer and SKINI-added fractional parts is up to the implementor of the algorithm

being controlled by SKINI messages. But the extra precision is there to be used or ignored.

5.24 Why SKINI?

SKINI was designed to be extensable and hackable for a number of applications: imbedded synthesis in a game or VR simulation, scoring and mixing tasks, real-time and non-real time applications which could benefit from controllable sound synthesis, JAVA controlled synthesis, or eventually maybe JAVA synthesis, etc. SKINI is not intended to be "the mother of scorefiles," but since the entire system is based on text representations of names, floats, and ints, converters from one scorefile language to SKINI, or back, should be easily created.

I am basically a bottom-up designer with an awareness of top-down design ideas, so SKINI above all reflects the needs of my particular research and creative projects as they have arisen and developed. SKINI 1.1 represents a profound advance beyond versions 0.8 and 0.9 (the first versions), future SKINI's might reflect some changes. Compatibility with prior scorefiles will be attempted, but there aren't that many scorefiles out there yet.

5.25 SKINI Messages

A basic SKINI message is a line of text. There are only three required fields, the message type (an ASCII name), the time (either delta or absolute), and the channel number. Don't freak out and think that this is MIDI channel 0-15 (which is supported), because the channel number is scanned as a long int. Channels could be socket numbers, machine IDs, serial numbers, or even unique tags for each event in a synthesis. Other fields might be used, as specified in the SKINI.tbl file. This is described in more detail later.

Fields in a SKINI line are delimited by spaces, commas, or tabs. The SKINI parser only operates on a line at a time, so a newline means the message is over. Multiple messages are NOT allowed directly on a single line (by use of the ; for example in C). This could be supported, but it isn't in version 1.1.

Message types include standard MIDI types like NoteOn, NoteOff, Control-Change, etc. MIDI extension message types (messages which look better than MIDI but actually get turned into MIDI-like messages) include LipTension, StringDamping, etc. Non-MIDI message types include SetPath (sets a path for file use later), and OpenReadFile (for streaming, mixing, and applying effects to soundfiles along with synthesis, for example). Other non-MIDI message types include Trilling, HammerOn, etc. (these translate to gestures, behaviors, and contexts for use by intellegent players and instruments using SKINI). Where possible I will still use these as MIDI extension messages, so foot switches, etc.

can be used to control them in real time.

All fields other than type, time, and channel are optional, and the types and useage of the additional fields is defined in the file SKINI.tbl.

The other important file used by SKINI is SKINI.msg, which is a set of defines to make C code more readable, and to allow reasonably quick re-mapping of control numbers, etc.. All of these defined symbols are assigned integer values. For Java, the defines could be replaced by declaration and assignment statements, preserving the look and behavior of the rest of the code.

5.26 C Files Used To Implement SKINI

SKINI.cpp is an object which can either open a SKINI file, and successively read and parse lines of text as SKINI strings, or accept strings from another object and parse them. The latter functionality would be used by a socket, pipe, or other connection receiving SKINI messages a line at a time, usually in real time, but not restricted to real time.

SKINI.msg should be included by anything wanting to use the SKINI.cpp object. This is not mandatory, but use of the __SK_blah_ symbols which are defined in the .msg file will help to ensure clarity and consistency when messages are added and changed.

SKINI.tbl is used only by the SKINI parser object (SKINI.cpp). In the file SKINI.tbl, an array of structures is declared and assigned values which instruct the parser as to what the message types are, and what the fields mean for those message types. This table is compiled and linked into applications using SKINI, but could be dynamically loaded and changed in a future version of SKINI.

5.27 SKINI Messages and the SKINI Parser:

The parser isn't all that smart, but neither am I. Here are the basic rules governing a valid SKINI message:

- If the first (non-delimiter ... see below) character in a SKINI string is '/' that line is treated as a comment and echoed to stdout.
- If there are no characters on a line, that line is treated as blank and echoed to stdout. Tabs and spaces are treated as non-characters.
- Spaces, commas, and tabs delimit the fields in a SKINI message line. (We might allow for multiple messages per line later using the semicolon, but probably not. A series of lines with deltaTimes of 0.0 denotes simultaneous

events. For read-ability, multiple messages per line doesn't help much, so it's unlikely to be supported later).

- The first field must be a SKINI message name (like NoteOn). These might become case-insensitive in future versions, so don't plan on exciting clever overloading of names (like noTeOn being different from NoTeON). There can be a number of leading spaces or tabs, but don't exceed 32 or so.

- The second field must be a time specification in seconds. A time field can be either delta-time (most common and the only one supported in version 0.8), or absolute time. Absolute time messages have an '=' appended to the beginning of the floating point number with no space. So 0.10000 means delta time of 100 ms, while =0.10000 means absolute time of 100 ms. Absolute time messages make most sense in score files, but could also be used for (loose) synchronization in a real-time context. Real-time messages should be time-ordered AND time-correct. That is, if you've sent 100 total delta-time messages of 1.0 seconds, and then send an absolute time message of =90.0 seconds, or if you send two absolute time messages of =100.0 and =90.0 in that order, things will get really fouled up. The SKINI parser doesn't know about time, however. The WvOut device is the master time keeper in the Synthesis Toolkit, so it should be queried to see if absolute time messages are making sense. There's an example of how to do that later in this document. Absolute times are returned by the parser as negative numbers (since negative deltaTimes are not allowed).

- The third field must be an integer channel number. Don't go crazy and think that this is just MIDI channel 0-15 (which is supported). The channel number is scanned as a long int. Channels 0-15 are in general to be treated as MIDI channels. After that it's wide open. Channels could be socket numbers, machine IDs, serial numbers, or even unique tags for each event in a synthesis. A -1 channel can be used as don't care, omni, or other functions depending on your needs and taste.

- All remaining fields are specified in the SKINI.tbl file. In general, there are maximum two more fields, which are either SK_INT (long), SK_DBL (double float), or SK_STR (string). The latter is the mechanism by which more arguments can be specified on the line, but the object using SKINI must take that string apart (retrived by using getRemainderString()) and scan it. Any excess fields are stashed in remainderString.

5.28 A Short SKINI File:

```
/*  Howdy!!! Welcome to SKINI, by P. Cook 1999
```

```
NoteOn         0.000082 2 55 82
NoteOff        1.000000 2 55 0
NoteOn         0.000082 2 69 82
StringDetune   0.100000 2    10
StringDetune   0.100000 2    30
StringDetune   0.100000 2    50
NoteOn         0.000000 2 69 82
StringDetune   0.100000 2    40
StringDetune   0.100000 2    22
StringDetune   0.100000 2    12
//
StringDamping  0.000100 2     0.0
NoteOn         0.000082 2 55 82
NoteOn         0.200000 2 62 82
NoteOn         0.100000 2 71 82
NoteOn         0.200000 2 79 82
NoteOff        1.000000 2 55 82
NoteOff        0.000000 2 62 82
NoteOff        0.000000 2 71 82
NoteOff        0.000000 2 79 82
StringDamping =4.000000 2       0.0
NoteOn         0.000082 2 55 82
NoteOn         0.200000 2 62 82
NoteOn         0.100000 2 71 82
NoteOn         0.200000 2 79 82
NoteOff        1.000000 2 55 82
NoteOff        0.000000 2 62 82
NoteOff        0.000000 2 71 82
NoteOff        0.000000 2 79 82
```

5.29 The SKINI.tbl File and Message Parsing:

The SKINI.tbl file contains an array of structures which are accessed by the parser object SKINI.cpp. The struct is:

```
struct SKINISpec {
   char messageString[32];
   long  type;
   long data2;
   long data3;
};
```

so an assignment of one of these structs looks like:

```
MessageStr$      ,type, data2, data3,
```

`type` is the message type sent back from the SKINI line parser.

`data<n>` is either:

- NOPE : field not used, specifically, there aren't going to be any more fields on this line. So if there is is NOPE in data2, data3 won't even be checked.

- SK_INT : byte (actually scanned as 32 bit signed long int). If it's a MIDI data field which is required to be an integer, like a controller number, it's 0-127. Otherwise, get creative with SK_INTs.

- SK_DBL : double precision floating point. SKINI uses these in the MIDI context for note numbers with micro tuning, velocities, controller values, etc.

- SK_STR : only valid in final field. This allows (nearly) arbitrary message types to be supported by simply scanning the string to EndOfLine and then passing it to a more intellegent handler. For example, MIDI SYSEX (system exclusive) messages of up to 256 bytes can be read as space-delimited integers into the 1K SK_STR buffer. Longer bulk dumps, soundfiles, etc. should be handled as a new message type pointing to a FileName, Socket, or something else stored in the SK_STR field, or as a new type of multi-line message.

Here's a couple of lines from the SKINI.tbl file

```
{"NoteOff"          ,        __SK_NoteOff_,               SK_DBL,  SK_DBL},
{"NoteOn"           ,         __SK_NoteOn_,               SK_DBL,  SK_DBL},

{"ControlChange"    ,  __SK_ControlChange_,               SK_INT,  SK_DBL},
{"Volume"           ,  __SK_ControlChange_, __SK_Volume_        ,  SK_DBL},

{"StringDamping"    ,   __SK_ControlChange_, __SK_StringDamping_,  SK_DBL},
{"StringDetune"     ,    __SK_ControlChange_, __SK_StringDetune_,  SK_DBL},
```

The first three are basic MIDI messages. The first two would cause the parser, after recognizing a match of the string "NoteOff" or "NoteOn", to set the message type to 128 or 144 (__SK_NoteOff_ and __SK_NoteOn_ are defined in the file SKINI.msg to be the MIDI byte value, without channel, of the actual MIDI messages for NoteOn and NoteOff). The parser would then set the time or delta time (this is always done and is therefore not described in the SKINI Message Struct). The next two fields would be scanned as double-precision floats and assigned to the byteTwo and byteThree variables of the SKINI parser. The remainder of the line is stashed in the remainderString variable.

The ControlChange spec is basically the same as NoteOn and NoteOff, but the second data byte is set to an integer (for checking later as to what MIDI control is being changed).

The Volume spec is a MIDI Extension message, which behaves like a Control-Change message with the controller number set explicitly to the value for MIDI

Volume (7). Thus the following two lines would accomplish the same changing of MIDI volume on channel 2:

```
ControlChange  0.000000 2 7 64.1
Volume         0.000000 2   64.1
```

I like the 2nd line better, thus my motivation for SKINI in the first place.

The StringDamping and StringDetune messages behave the same as the Volume message, but use Control Numbers which aren't specifically nailed-down in MIDI. Note that these Control Numbers are carried around as long ints, so we're not limited to 0-127. If, however, you want to use a MIDI controller to play an instrument, using controller numbers in the 0-127 range might make sense.

5.30 Using SKINI:

Here's a simple example of code which uses the SKINI object to read a SKINI file and control a single instrument.

```
instrument = new Mandolin(50.0);
score = new SKINI(argv[1]);
while(score->getType() > 0) {
    tempDouble = score->getDelta();
    if (tempDouble < 0)     {
        tempDouble = - tempDouble;
        tempDouble = tempDouble - output.getTime();
        if (tempDouble < 0) {
           printf("Bad News Here!!!  Backward Absolute Time Required.\n");
           tempDouble = 0.0;
        }
    }
    tempLong = (long) (tempDouble * Stk::sampleRate());
    for (i=0;i<tempLong;i++)   {
        output.tick(instrument->tick());
    }
    tempDouble3 = score->getByteThree();
    if (score->getType()== __SK_NoteOn_ )        {
        tempDouble3 *= NORM_MIDI;
        if (score->getByteThree() == 0)  {
            tempDouble3 = 0.5;
            instrument->noteOff(tempDouble3);
        }
        else {
            tempLong = (int) score->getByteTwo();
            tempDouble2 = Midi2Pitch[tempLong];
            instrument->noteOn(tempDouble2,tempDouble3);
        }
    }
    else if (score->getType() == __SK_NoteOff_) {
```

```
            tempDouble3 *= NORM_MIDI;
            instrument->noteOff(tempDouble3);
        }
        else if (score->getType() == __SK_ControlChange_)        {
            tempLong = score->getByteTwoInt();
            instrument->controlChange(tempLong,temp3.0);
        }
        score->nextMessage();
    }
```

When the score (SKINI object) object is created from the filename in argv[1], the first valid command line is read from the file and parsed.

The score->getType() retrieves the messageType. If this is -1, there are no more valid messages in the file and the synthesis loop terminates. Otherwise, the message type is returned.

getDelta() retrieves the deltaTime until the current message should occur. If this is greater than 0, synthesis occurs until the deltaTime has elapsed. If deltaTime is less than zero, the time is interpreted as absolute time and the output device is queried as to what time it is now. That is used to form a deltaTime, and if it's positive we synthesize. If it's negative, we print an error and pretend this never happened and we hang around hoping to eventually catch up.

The rest of the code sorts out message types NoteOn, NoteOff (including NoteOn with velocity 0), and ControlChange. The code implicitly takes into account the integer type of the control number, but all other data is treated as double float.

The last line reads and parses the next message in the file.

Index

~ADSR
ADSR, 12
~BandedWG
BandedWG, 14
~BeeThree
BeeThree, 16
~BiQuad
BiQuad, 18
~BlowBotl
BlowBotl, 21
~BlowHole
BlowHole, 23
~BowTabl
BowTabl, 27
~Bowed
Bowed, 25
~Brass
Brass, 29
~Chorus
Chorus, 31
~Clarinet
Clarinet, 33
~Delay
Delay, 35
~DelayA
DelayA, 37
~DelayL
DelayL, 39
~Drummer
Drummer, 41
~Echo
Echo, 43
~Envelope
Envelope, 45
~FM
FM, 52
~FMVoices
FMVoices, 55
~Filter
Filter, 47
~Flute
Flute, 50
~FormSwep
FormSwep, 57
~HevyMetl
HevyMetl, 60
~Instrmnt
Instrmnt, 62
~JetTabl
JetTabl, 65
~Mandolin
Mandolin, 67
~Mesh2D
Mesh2D, 69
~Messager
Messager, 71
~Modal
Modal, 73
~ModalBar
ModalBar, 75
~Modulate
Modulate, 77
~Moog
Moog, 79
~Noise
Noise, 81
~OnePole
OnePole, 84
~OneZero
OneZero, 86
~PercFlut
PercFlut, 88

~PitShift
    PitShift, 90
~PluckTwo
    PluckTwo, 94
~Plucked
    Plucked, 92
~PoleZero
    PoleZero, 96
~ReedTabl
    ReedTabl, 100
~Resonate
    Resonate, 102
~Reverb
    Reverb, 104
~Rhodey
    Rhodey, 106
~RtDuplex
    RtDuplex, 108
~RtMidi
    RtMidi, 111
~RtWvIn
    RtWvIn, 113
~RtWvOut
    RtWvOut, 116
~SKINI
    SKINI, 130
~Sampler
    Sampler, 119
~Saxofony
    Saxofony, 121
~Shakers
    Shakers, 123
~Simple
    Simple, 126
~Sitar
    Sitar, 128
~Socket
    Socket, 133
~StifKarp
    StifKarp, 137
~Stk
    Stk, 140
~StkError
    StkError, 143
~SubNoise
    SubNoise, 144
~Table
    Table, 146
~TcpWvIn
    TcpWvIn, 148
~TcpWvOut
    TcpWvOut, 151
~Thread
    Thread, 154
~TubeBell
    TubeBell, 156
~TwoPole
    TwoPole, 158
~TwoZero
    TwoZero, 161
~WaveLoop
    WaveLoop, 164
~Wurley
    Wurley, 166
~WvIn
    WvIn, 168
~WvOut
    WvOut, 173

accept
    Socket, 135
addPhase
    WaveLoop, 165
addPhaseOffset
    WaveLoop, 165
addTime
    WaveLoop, 164
    WvIn, 169
ADSR, 11
    ~ADSR, 12
    ADSR, 11
    getState, 12
    keyOff, 12
    keyOn, 12
    setAllTimes, 12
    setAttackRate, 12
    setAttackTime, 12
    setDecayRate, 12
    setDecayTime, 12
    setReleaseRate, 12
    setReleaseTime, 12
    setSustainLevel, 12

- setTarget, 12
- setValue, 12
- tick, 13

BandedWG
- ~BandedWG, 14
- BandedWG, 14
- clear, 14
- controlChange, 15
- noteOff, 15
- noteOn, 15
- pluck, 15
- setFrequency, 14
- setPreset, 14
- setStrikePosition, 14
- startBowing, 14
- stopBowing, 14
- tick, 15

BandedWG, 14
BeeThree
- ~BeeThree, 16
- BeeThree, 16
- noteOn, 16
- tick, 16

BeeThree, 16
BiQuad
- ~BiQuad, 18
- BiQuad, 18
- clear, 18
- getGain, 19
- lastOut, 19
- setA1, 18
- setA2, 18
- setB0, 18
- setB1, 18
- setB2, 18
- tick, 19

BiQuad, 18
- setEqualGainZeroes, 20
- setGain, 20
- setNotch, 20
- setResonance, 19

BlowBotl
- ~BlowBotl, 21
- BlowBotl, 21
- clear, 21
- controlChange, 22
- noteOff, 21
- noteOn, 21
- setFrequency, 21
- startBlowing, 21
- stopBlowing, 21
- tick, 21

BlowBotl, 21
BlowHole
- ~BlowHole, 23
- BlowHole, 23
- clear, 23
- controlChange, 24
- noteOff, 24
- noteOn, 24
- setFrequency, 23
- setTonehole, 23
- setVent, 23
- startBlowing, 23
- stopBlowing, 23
- tick, 24

BlowHole, 23
Bowed, 25
- ~Bowed, 25
- Bowed, 25
- clear, 25
- controlChange, 26
- noteOff, 26
- noteOn, 25
- setFrequency, 25
- setVibrato, 25
- startBowing, 25
- stopBowing, 25
- tick, 26

BowTabl
- ~BowTabl, 27
- BowTabl, 27
- lastOut, 27
- tick, 27

BowTabl, 27
- setOffset, 28
- setSlope, 28
- tick, 28

Brass, 29
- ~Brass, 29
- Brass, 29

clear, 29
controlChange, 30
noteOff, 30
noteOn, 29
setFrequency, 29
setLip, 29
startBlowing, 29
stopBlowing, 29
tick, 30

Chorus, 31
~Chorus, 31
Chorus, 31
clear, 31
lastOut, 31
lastOutLeft, 31
lastOutRight, 31
setEffectMix, 31
setModDepth, 31
setModFrequency, 31
tick, 32
Clarinet, 33
~Clarinet, 33
Clarinet, 33
clear, 33
controlChange, 34
noteOff, 33
noteOn, 33
setFrequency, 33
startBlowing, 33
stopBlowing, 33
tick, 34
clear
BandedWG, 14
BiQuad, 18
BlowBotl, 21
BlowHole, 23
Bowed, 25
Brass, 29
Chorus, 31
Clarinet, 33
Delay, 35
DelayA, 37
Echo, 43
Filter, 47
Flute, 50
FM, 52
JCRev, 64
Mesh2D, 69
Modal, 73
NRev, 83
OnePole, 84
OneZero, 86
PitShift, 90
Plucked, 92
PluckTwo, 94
PoleZero, 96
PRCRev, 99
Resonate, 102
Reverb, 104
Sampler, 119
Saxofony, 121
Simple, 126
Sitar, 128
StifKarp, 137
TwoPole, 158
TwoZero, 161
close
Socket, 133, 134
closeFile
WvIn, 168
WvOut, 173
connect
Socket, 135
TcpWvOut, 153
contentsAt
Delay, 36
controlChange
BandedWG, 15
BlowBotl, 22
BlowHole, 24
Bowed, 26
Brass, 30
Clarinet, 34
Flute, 51
FM, 53
FMVoices, 55
Instrmnt, 63
Mandolin, 68
Mesh2D, 70
Modal, 74
ModalBar, 75

Moog, 79
Resonate, 103
Sampler, 120
Saxofony, 122
Shakers, 123
Simple, 127
StifKarp, 138

damp
Modal, 74
Delay, 35
~Delay, 35
clear, 35
contentsAt, 36
Delay, 35
energy, 35
getDelay, 35
lastOut, 36
setDelay, 36
tick, 36
DelayA
~DelayA, 37
clear, 37
DelayA, 37
getDelay, 37
tick, 37
DelayA, 37
setDelay, 38
DelayL
~DelayL, 39
DelayL, 39
getDelay, 39
tick, 39
DelayL, 39
setDelay, 40
disconnect
TcpWvOut, 151
Drummer, 41
~Drummer, 41
Drummer, 41
noteOff, 41
noteOn, 42
tick, 41

Echo, 43
~Echo, 43
clear, 43
Echo, 43
lastOut, 43
setDelay, 43
setEffectMix, 43
tick, 43
energy
Delay, 35
Mesh2D, 70
Envelope, 45
~Envelope, 45
Envelope, 45
getState, 45
keyOff, 45
keyOn, 45
lastOut, 46
setRate, 45
setTarget, 45
setTime, 45
setValue, 45
tick, 46

Filter, 47
~Filter, 47
clear, 47
Filter, 47, 48
getGain, 48
lastOut, 48
setCoefficients, 49
setDenominator, 49
setGain, 49
setNumerator, 49
tick, 48
Flute, 50
~Flute, 50
clear, 50
controlChange, 51
Flute, 50
noteOff, 51
noteOn, 51
setEndReflection, 50
setFrequency, 50
setJetDelay, 50
setJetReflection, 50
startBlowing, 50
stopBlowing, 51

tick, 51
FM, 52
~FM, 52
clear, 52
controlChange, 53
FM, 52
keyOff, 53
keyOn, 53
loadWaves, 52
noteOff, 53
setControl1, 53
setControl2, 53
setFrequency, 52
setGain, 52
setModulationDepth, 53
setModulationSpeed, 52
setRatio, 52
tick, 53
FMVoices, 55
~FMVoices, 55
controlChange, 55
FMVoices, 55
noteOn, 55
setFrequency, 55
tick, 55
FormSwep
~FormSwep, 57
FormSwep, 57
setStates, 57
setTargets, 57
tick, 58
FormSwep, 57
setResonance, 58
setSweepRate, 58
setSweepTime, 58

getByteThree
Messager, 71
RtMidi, 111
SKINI, 131
getByteThreeInt
SKINI, 131
getByteTwo
Messager, 71
RtMidi, 111
SKINI, 130
getByteTwoInt
SKINI, 131
getChannel
Messager, 71
RtMidi, 111
SKINI, 130
getChannels
WvIn, 169
getDelay
Delay, 35
DelayA, 37
DelayL, 39
getDelta
Messager, 71
SKINI, 130
getDeltaTime
RtMidi, 111
getFileRate
WvIn, 171
getFrames
RtWvOut, 116
TcpWvOut, 151
WvOut, 173
getGain
BiQuad, 19
Filter, 48
OnePole, 84
OneZero, 86
PoleZero, 97
TwoPole, 159
TwoZero, 161
getLength
Table, 146
getMessage
StkError, 143
getMessageTypeString
SKINI, 131
getRemainderString
SKINI, 131
getSize
WvIn, 168
getState
ADSR, 12
Envelope, 45
getTime
RtWvOut, 116

TcpWvOut, 151
WvOut, 173
getType
Messager, 71
RtMidi, 111
SKINI, 130
StkError, 143

handleError
Stk, 140
HevyMetl
~HevyMetl, 60
HevyMetl, 60
noteOn, 60
tick, 60
HevyMetl, 60

Instrmnt, 62
~Instrmnt, 62
controlChange, 63
Instrmnt, 62
lastOut, 63
noteOff, 63
noteOn, 63
setFrequency, 63
tick, 63
isConnected
TcpWvIn, 150
isFinished
WvIn, 169
isValid
Socket, 134

JCRev, 64
clear, 64
tick, 64
JetTabl
~JetTabl, 65
JetTabl, 65
lastOut, 65
tick, 65
JetTabl, 65

keyOff
ADSR, 12
Envelope, 45
FM, 53
Resonate, 102
Sampler, 119
Simple, 126
keyOn
ADSR, 12
Envelope, 45
FM, 53
Resonate, 102
Sampler, 119
Simple, 126

lastFrame
RtDuplex, 108
RtWvIn, 113
TcpWvIn, 148
WvIn, 169
lastOut
BiQuad, 19
BowTabl, 27
Chorus, 31
Delay, 36
Echo, 43
Envelope, 46
Filter, 48
Instrmnt, 63
JetTabl, 65
Modulate, 77
Noise, 81
OnePole, 85
OneZero, 87
PitShift, 90
PoleZero, 97
ReedTabl, 100
Reverb, 104
RtDuplex, 108
RtWvIn, 113
Table, 146
TcpWvIn, 148
TwoPole, 159
TwoZero, 162
WvIn, 169
lastOutLeft
Chorus, 31
Reverb, 104
lastOutRight

Chorus, 31
Reverb, 104
listen
TcpWvIn, 149
loadWaves
FM, 52

Mandolin, 67
~Mandolin, 67
controlChange, 68
Mandolin, 67
noteOn, 67
pluck, 67
setBodySize, 67
tick, 67
Mesh2D, 69
~Mesh2D, 69
clear, 69
controlChange, 70
energy, 70
Mesh2D, 69
noteOff, 69
noteOn, 69
setDecay, 69
setInputPosition, 69
setNX, 69
setNY, 69
tick, 70
Messager, 71
~Messager, 71
getByteThree, 71
getByteTwo, 71
getChannel, 71
getDelta, 71
getType, 71
Messager, 72
nextMessage, 72
setRtDelta, 71
Modal, 73
~Modal, 73
clear, 73
controlChange, 74
damp, 74
Modal, 73
noteOff, 74
noteOn, 74
setDirectGain, 73
setFrequency, 73
setMasterGain, 73
setModeGain, 73
setRatioAndRadius, 73
strike, 74
tick, 74
ModalBar
~ModalBar, 75
controlChange, 75
ModalBar, 75
setModulationDepth, 75
setPreset, 75
setStickHardness, 75
setStrikePosition, 75
ModalBar, 75
Modulate, 77
~Modulate, 77
lastOut, 77
Modulate, 77
reset, 77
setRandomGain, 77
setVibratoGain, 77
setVibratoRate, 77
tick, 77
Moog, 79
~Moog, 79
controlChange, 79
Moog, 79
noteOn, 79
setFrequency, 79
setModulationDepth, 79
setModulationSpeed, 79
tick, 79

nextMessage
Messager, 72
RtMidi, 112
SKINI, 132
Noise, 81
~Noise, 81
lastOut, 81
Noise, 81
tick, 81
normalize
WvIn, 170, 171

noteOff
- BandedWG, 15
- BlowBotl, 21
- BlowHole, 24
- Bowed, 26
- Brass, 30
- Clarinet, 33
- Drummer, 41
- Flute, 51
- FM, 53
- Instrmnt, 63
- Mesh2D, 69
- Modal, 74
- Plucked, 92
- PluckTwo, 95
- Resonate, 103
- Sampler, 119
- Saxofony, 122
- Shakers, 123
- Simple, 126
- Sitar, 128
- StifKarp, 138

noteOn
- BandedWG, 15
- BeeThree, 16
- BlowBotl, 21
- BlowHole, 24
- Bowed, 25
- Brass, 29
- Clarinet, 33
- Drummer, 42
- Flute, 51
- FMVoices, 55
- HevyMetl, 60
- Instrmnt, 63
- Mandolin, 67
- Mesh2D, 69
- Modal, 74
- Moog, 79
- PercFlut, 88
- Plucked, 92
- Resonate, 102
- Rhodey, 106
- Saxofony, 121
- Shakers, 125
- Simple, 126
- Sitar, 128
- StifKarp, 137
- TubeBell, 156
- Wurley, 166

NRev, 83
- clear, 83
- tick, 83

OnePole
- ~OnePole, 84
- clear, 84
- getGain, 84
- lastOut, 85
- OnePole, 84
- setA1, 84
- setB0, 84
- tick, 85

OnePole, 84
- setGain, 85
- setPole, 85

OneZero
- ~OneZero, 86
- clear, 86
- getGain, 86
- lastOut, 87
- OneZero, 86
- setB0, 86
- setB1, 86
- tick, 87

OneZero, 86
- setGain, 87
- setZero, 87

openFile
- WvIn, 170
- WvOut, 175

parseThis
- SKINI, 132

PercFlut
- ~PercFlut, 88
- noteOn, 88
- PercFlut, 88
- setFrequency, 88
- tick, 88

PercFlut, 88

PitShift

~PitShift, 90
clear, 90
lastOut, 90
PitShift, 90
setEffectMix, 90
setShift, 90
tick, 90
PitShift, 90
pluck
BandedWG, 15
Mandolin, 67
Plucked, 92
Sitar, 128
StifKarp, 137
Plucked, 92
~Plucked, 92
clear, 92
noteOff, 92
noteOn, 92
pluck, 92
Plucked, 92
setFrequency, 92
tick, 92
PluckTwo
~PluckTwo, 94
clear, 94
noteOff, 95
PluckTwo, 94
setDetune, 94
setFreqAndDetune, 94
setFrequency, 94
setPluckPosition, 94
tick, 95
PluckTwo, 94
setBaseLoopGain, 95
PoleZero
~PoleZero, 96
clear, 96
getGain, 97
lastOut, 97
PoleZero, 96
setA1, 96
setB0, 96
setB1, 96
tick, 97
PoleZero, 96
setAllpass, 97
setBlockZero, 97
setGain, 97
port
Socket, 133
PRCRev, 99
clear, 99
tick, 99
printMessage
RtMidi, 111
StkError, 143

readBuffer
Socket, 134
ReedTabl
~ReedTabl, 100
lastOut, 100
ReedTabl, 100
tick, 100
ReedTabl, 100
setOffset, 101
setSlope, 101
tick, 101
reset
Modulate, 77
WvIn, 168
Resonate, 102
~Resonate, 102
clear, 102
controlChange, 103
keyOff, 102
keyOn, 102
noteOff, 103
noteOn, 102
Resonate, 102
setEqualGainZeroes, 102
setNotch, 102
setResonance, 102
tick, 103
Reverb, 104
~Reverb, 104
clear, 104
lastOut, 104
lastOutLeft, 104
lastOutRight, 104
Reverb, 104

setEffectMix, 104
tick, 104, 105
Rhodey, 106
~Rhodey, 106
noteOn, 106
Rhodey, 106
setFrequency, 106
tick, 106
RtDuplex
~RtDuplex, 108
lastFrame, 108
lastOut, 108
RtDuplex, 109
RtDuplex, 108
RtDuplex, 109
start, 109
stop, 110
tick, 110
tickFrame, 110
RtMidi
~RtMidi, 111
getByteThree, 111
getByteTwo, 111
getChannel, 111
getDeltaTime, 111
getType, 111
printMessage, 111
RtMidi, 111
RtMidi, 111
nextMessage, 112
RtWvIn
~RtWvIn, 113
lastFrame, 113
lastOut, 113
RtWvIn, 114
RtWvIn, 113
RtWvIn, 114
start, 114
stop, 115
tick, 115
tickFrame, 115
RtWvOut
~RtWvOut, 116
getFrames, 116
getTime, 116
RtWvOut, 117
RtWvOut, 116
RtWvOut, 117
start, 117
stop, 117
tick, 118
tickFrame, 118

Sampler, 119
~Sampler, 119
clear, 119
controlChange, 120
keyOff, 119
keyOn, 119
noteOff, 119
Sampler, 119
setFrequency, 119
tick, 119
sampleRate
Stk, 139
Saxofony, 121
~Saxofony, 121
clear, 121
controlChange, 122
noteOff, 122
noteOn, 121
Saxofony, 121
setBlowPosition, 121
setFrequency, 121
startBlowing, 121
stopBlowing, 121
tick, 122
setA1
BiQuad, 18
OnePole, 84
PoleZero, 96
TwoPole, 158
setA2
BiQuad, 18
TwoPole, 158
setAllpass
PoleZero, 97
setAllTimes
ADSR, 12
setAttackRate
ADSR, 12
setAttackTime

ADSR, 12
setB0
BiQuad, 18
OnePole, 84
OneZero, 86
PoleZero, 96
TwoPole, 158
TwoZero, 161
setB1
BiQuad, 18
OneZero, 86
PoleZero, 96
TwoZero, 161
setB2
BiQuad, 18
TwoZero, 161
setBaseLoopGain
PluckTwo, 95
StifKarp, 138
setBlocking
Socket, 134
setBlockZero
PoleZero, 97
setBlowPosition
Saxofony, 121
setBodySize
Mandolin, 67
setCoefficients
Filter, 49
setControl1
FM, 53
setControl2
FM, 53
setDecay
Mesh2D, 69
setDecayRate
ADSR, 12
setDecayTime
ADSR, 12
setDelay
Delay, 36
DelayA, 38
DelayL, 40
Echo, 43
setDenominator
Filter, 49
setDetune
PluckTwo, 94
setDirectGain
Modal, 73
setEffectMix
Chorus, 31
Echo, 43
PitShift, 90
Reverb, 104
setEndReflection
Flute, 50
setEqualGainZeroes
BiQuad, 20
Resonate, 102
setFreqAndDetune
PluckTwo, 94
setFrequency
BandedWG, 14
BlowBotl, 21
BlowHole, 23
Bowed, 25
Brass, 29
Clarinet, 33
Flute, 50
FM, 52
FMVoices, 55
Instrmnt, 63
Modal, 73
Moog, 79
PercFlut, 88
Plucked, 92
PluckTwo, 94
Rhodey, 106
Sampler, 119
Saxofony, 121
Simple, 126
Sitar, 128
StifKarp, 137
WaveLoop, 165
Wurley, 166
setGain
BiQuad, 20
Filter, 49
FM, 52
OnePole, 85
OneZero, 87

PoleZero, 97
TwoPole, 159
TwoZero, 162
setInputPosition
Mesh2D, 69
setInterpolate
WvIn, 171
setJetDelay
Flute, 50
setJetReflection
Flute, 50
setLip
Brass, 29
setMasterGain
Modal, 73
setModDepth
Chorus, 31
setModeGain
Modal, 73
setModFrequency
Chorus, 31
setModulationDepth
FM, 53
ModalBar, 75
Moog, 79
setModulationSpeed
FM, 52
Moog, 79
setNotch
BiQuad, 20
Resonate, 102
TwoZero, 162
setNumerator
Filter, 49
setNX
Mesh2D, 69
setNY
Mesh2D, 69
setOffset
BowTabl, 28
ReedTabl, 101
setPickupPosition
StifKarp, 137
setPluckPosition
PluckTwo, 94
setPole
OnePole, 85
setPreset
BandedWG, 14
ModalBar, 75
setRandomGain
Modulate, 77
setRate
Envelope, 45
SubNoise, 144
WvIn, 171
setRatio
FM, 52
setRatioAndRadius
Modal, 73
setReleaseRate
ADSR, 12
setReleaseTime
ADSR, 12
setResonance
BiQuad, 19
FormSwep, 58
Resonate, 102
TwoPole, 159
setRtDelta
Messager, 71
setSampleRate
Stk, 141
setShift
PitShift, 90
setSlope
BowTabl, 28
ReedTabl, 101
setStates
FormSwep, 57
setStickHardness
ModalBar, 75
setStretch
StifKarp, 137
setStrikePosition
BandedWG, 14
ModalBar, 75
setSustainLevel
ADSR, 12
setSweepRate
FormSwep, 58
setSweepTime

FormSwep, 58
setTarget
ADSR, 12
Envelope, 45
setTargets
FormSwep, 57
setTime
Envelope, 45
setTonehole
BlowHole, 23
setValue
ADSR, 12
Envelope, 45
setVent
BlowHole, 23
setVibrato
Bowed, 25
setVibratoGain
Modulate, 77
setVibratoRate
Modulate, 77
setZero
OneZero, 87
Shakers, 123
~Shakers, 123
controlChange, 123
noteOff, 123
noteOn, 125
Shakers, 123
tick, 123
Simple, 126
~Simple, 126
clear, 126
controlChange, 127
keyOff, 126
keyOn, 126
noteOff, 126
noteOn, 126
setFrequency, 126
Simple, 126
tick, 126
Sitar, 128
~Sitar, 128
clear, 128
noteOff, 128
noteOn, 128
pluck, 128
setFrequency, 128
Sitar, 128
tick, 128
SKINI, 130
~SKINI, 130
getByteThree, 131
getByteThreeInt, 131
getByteTwo, 130
getByteTwoInt, 131
getChannel, 130
getDelta, 130
getMessageTypeString, 131
getRemainderString, 131
getType, 130
nextMessage, 132
parseThis, 132
SKINI, 130
whatsThisController, 131
whatsThisType, 131
sleep
Stk, 140
Socket, 133
~Socket, 133
accept, 135
close, 133, 134
connect, 135
isValid, 134
port, 133
readBuffer, 134
setBlocking, 134
Socket, 135
socket, 133
writeBuffer, 134
socket
Socket, 133
start
RtDuplex, 109
RtWvIn, 114
RtWvOut, 117
Thread, 155
startBlowing
BlowBotl, 21
BlowHole, 23
Brass, 29
Clarinet, 33

Flute, 50
Saxofony, 121
startBowing
BandedWG, 14
Bowed, 25
StifKarp
~StifKarp, 137
clear, 137
controlChange, 138
noteOff, 138
noteOn, 137
pluck, 137
setFrequency, 137
setPickupPosition, 137
setStretch, 137
StifKarp, 137
tick, 138
StifKarp, 137
setBaseLoopGain, 138
Stk, 139
~Stk, 140
handleError, 140
sampleRate, 139
setSampleRate, 141
sleep, 140
Stk, 140
STK_FLOAT32, 141
STK_FLOAT64, 141
STK_SINT16, 141
STK_SINT32, 141
STK_SINT8, 141
swap16, 140
swap32, 140
swap64, 140
STK_FLOAT32
Stk, 141
STK_FLOAT64
Stk, 141
STK_SINT16
Stk, 141
STK_SINT32
Stk, 141
STK_SINT8
Stk, 141
StkError
~StkError, 143
getMessage, 143
getType, 143
printMessage, 143
StkError, 143
StkError, 143
stop
RtDuplex, 110
RtWvIn, 115
RtWvOut, 117
stopBlowing
BlowBotl, 21
BlowHole, 23
Brass, 29
Clarinet, 33
Flute, 51
Saxofony, 121
stopBowing
BandedWG, 14
Bowed, 25
strike
Modal, 74
SubNoise
~SubNoise, 144
setRate, 144
SubNoise, 144
subRate, 144
tick, 144
SubNoise, 144
subRate
SubNoise, 144
swap16
Stk, 140
swap32
Stk, 140
swap64
Stk, 140

Table, 146
~Table, 146
getLength, 146
lastOut, 146
Table, 146
tick, 146, 147
TcpWvIn
~TcpWvIn, 148
lastFrame, 148

lastOut, 148
TcpWvIn, 149
tick, 148
tickFrame, 149
TcpWvIn, 148
isConnected, 150
listen, 149
TcpWvIn, 149
TcpWvOut
~TcpWvOut, 151
disconnect, 151
getFrames, 151
getTime, 151
TcpWvOut, 151, 152
TcpWvOut, 151
connect, 153
TcpWvOut, 152
tick, 153
tickFrame, 153
test
Thread, 154
Thread, 154
~Thread, 154
start, 155
test, 154
Thread, 154
wait, 155
tick
ADSR, 13
BandedWG, 15
BeeThree, 16
BiQuad, 19
BlowBotl, 21
BlowHole, 24
Bowed, 26
BowTabl, 27, 28
Brass, 30
Chorus, 32
Clarinet, 34
Delay, 36
DelayA, 37
DelayL, 39
Drummer, 41
Echo, 43
Envelope, 46
Filter, 48
Flute, 51
FM, 53
FMVoices, 55
FormSwep, 58
HevyMetl, 60
Instrmnt, 63
JCRev, 64
JetTabl, 65
Mandolin, 67
Mesh2D, 70
Modal, 74
Modulate, 77
Moog, 79
Noise, 81
NRev, 83
OnePole, 85
OneZero, 87
PercFlut, 88
PitShift, 90
Plucked, 92
PluckTwo, 95
PoleZero, 97
PRCRev, 99
ReedTabl, 100, 101
Resonate, 103
Reverb, 104, 105
Rhodey, 106
RtDuplex, 110
RtWvIn, 115
RtWvOut, 118
Sampler, 119
Saxofony, 122
Shakers, 123
Simple, 126
Sitar, 128
StifKarp, 138
SubNoise, 144
Table, 146, 147
TcpWvIn, 148
TcpWvOut, 153
TubeBell, 156
TwoPole, 159
TwoZero, 162
Wurley, 166
WvIn, 171
WvOut, 175

tickFrame
  RtDuplex, 110
  RtWvIn, 115
  RtWvOut, 118
  TcpWvIn, 149
  TcpWvOut, 153
  WaveLoop, 164
  WvIn, 172
  WvOut, 175
TubeBell
  ~TubeBell, 156
  noteOn, 156
  tick, 156
  TubeBell, 156
TubeBell, 156
TwoPole
  ~TwoPole, 158
  clear, 158
  getGain, 159
  lastOut, 159
  setA1, 158
  setA2, 158
  setB0, 158
  tick, 159
  TwoPole, 158
TwoPole, 158
  setGain, 159
  setResonance, 159
TwoZero
  ~TwoZero, 161
  clear, 161
  getGain, 161
  lastOut, 162
  setB0, 161
  setB1, 161
  setB2, 161
  tick, 162
  TwoZero, 161
TwoZero, 161
  setGain, 162
  setNotch, 162

wait
  Thread, 155
WaveLoop
  ~WaveLoop, 164
  addTime, 164
  tickFrame, 164
  WaveLoop, 164
WaveLoop, 164
  addPhase, 165
  addPhaseOffset, 165
  setFrequency, 165
whatsThisController
  SKINI, 131
whatsThisType
  SKINI, 131
writeBuffer
  Socket, 134
Wurley, 166
  ~Wurley, 166
  noteOn, 166
  setFrequency, 166
  tick, 166
  Wurley, 166
WvIn
  ~WvIn, 168
  addTime, 169
  closeFile, 168
  getChannels, 169
  getSize, 168
  isFinished, 169
  lastFrame, 169
  lastOut, 169
  reset, 168
  WvIn, 168, 170
WvIn, 168
  getFileRate, 171
  normalize, 170, 171
  openFile, 170
  setInterpolate, 171
  setRate, 171
  tick, 171
  tickFrame, 172
  WvIn, 170
WvOut
  ~WvOut, 173
  closeFile, 173
  getFrames, 173
  getTime, 173
  WvOut, 173, 175
WvOut, 173

openFile, 175
tick, 175
tickFrame, 175
WvOut, 175
WVOUT_AIF, 176
WVOUT_MAT, 176
WVOUT_RAW, 176
WVOUT_SND, 176
WVOUT_WAV, 176
WVOUT_AIF
WvOut, 176
WVOUT_MAT
WvOut, 176
WVOUT_RAW
WvOut, 176
WVOUT_SND
WvOut, 176
WVOUT_WAV
WvOut, 176