

DaisySP

Generated by Doxygen 1.9.1



<b>1 Main Page</b>	<b>1</b>
1.0.1 Applications	1
1.0.2 Features	1
1.0.3 Code Example	2
1.0.4 Getting Started	2
1.0.5 Community	2
1.0.6 Contributing	2
1.0.7 License	2
<b>2 Todo List</b>	<b>3</b>
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Class Documentation</b>	<b>11</b>
5.1 daisysp::AdEnv Class Reference	11
5.1.1 Detailed Description	11
5.1.2 Member Function Documentation	11
5.1.2.1 GetCurrentSegment()	12
5.1.2.2 GetValue()	12
5.1.2.3 Init()	12
5.1.2.4 IsRunning()	12
5.1.2.5 Process()	12
5.1.2.6 SetCurve()	13
5.1.2.7 SetMax()	13
5.1.2.8 SetMin()	13
5.1.2.9 SetTime()	13
5.1.2.10 Trigger()	13
5.2 daisysp::Adsr Class Reference	13
5.2.1 Detailed Description	14
5.2.2 Member Function Documentation	14
5.2.2.1 GetCurrentSegment()	14
5.2.2.2 Init()	14
5.2.2.3 IsRunning()	15
5.2.2.4 Process()	15
5.2.2.5 SetSustainLevel()	15
5.2.2.6 SetTime()	15
5.3 daisysp::Allpass Class Reference	16
5.3.1 Detailed Description	16
5.3.2 Member Function Documentation	16
5.3.2.1 Init()	16

---

5.3.2.2 Process()	17
5.3.2.3 SetFreq()	17
5.3.2.4 SetRevTime()	17
5.4 daisysp::AnalogBassDrum Class Reference	17
5.4.1 Detailed Description	18
5.4.2 Member Function Documentation	18
5.4.2.1 Init()	18
5.4.2.2 Process()	18
5.4.2.3 SetAccent()	19
5.4.2.4 SetAttackFmAmount()	19
5.4.2.5 SetDecay()	19
5.4.2.6 SetFreq()	20
5.4.2.7 SetSelfFmAmount()	20
5.4.2.8 SetSustain()	20
5.4.2.9 SetTone()	20
5.4.2.10 Trig()	21
5.5 daisysp::AnalogSnareDrum Class Reference	21
5.5.1 Detailed Description	21
5.5.2 Member Function Documentation	22
5.5.2.1 Init()	22
5.5.2.2 Process()	22
5.5.2.3 SetAccent()	22
5.5.2.4 SetDecay()	22
5.5.2.5 SetFreq()	24
5.5.2.6 SetSnappy()	24
5.5.2.7 SetSustain()	24
5.5.2.8 SetTone()	25
5.5.2.9 Trig()	25
5.6 daisysp::ATone Class Reference	25
5.6.1 Detailed Description	25
5.6.2 Member Function Documentation	26
5.6.2.1 GetFreq()	26
5.6.2.2 Init()	26
5.6.2.3 Process()	26
5.6.2.4 SetFreq()	26
5.7 daisysp::Autowah Class Reference	27
5.7.1 Detailed Description	27
5.7.2 Member Function Documentation	27
5.7.2.1 Init()	27
5.7.2.2 Process()	28
5.7.2.3 SetDryWet()	28
5.7.2.4 SetLevel()	28

---

5.7.2.5 SetWah()	28
5.8 daisysp::Balance Class Reference	29
5.8.1 Detailed Description	29
5.8.2 Member Function Documentation	29
5.8.2.1 Init()	29
5.8.2.2 Process()	30
5.8.2.3 SetCutoff()	30
5.9 daisysp::Biquad Class Reference	30
5.9.1 Detailed Description	30
5.9.2 Member Function Documentation	31
5.9.2.1 Init()	31
5.9.2.2 Process()	31
5.9.2.3 SetCutoff()	31
5.9.2.4 SetRes()	31
5.10 daisysp::Bitcrush Class Reference	33
5.10.1 Detailed Description	33
5.10.2 Member Function Documentation	33
5.10.2.1 Init()	33
5.10.2.2 Process()	34
5.10.2.3 SetBitDepth()	34
5.10.2.4 SetCrushRate()	34
5.11 daisysp::BIOsc Class Reference	34
5.11.1 Detailed Description	35
5.11.2 Member Enumeration Documentation	35
5.11.2.1 Waveforms	35
5.11.3 Member Function Documentation	35
5.11.3.1 Init()	35
5.11.3.2 Process()	35
5.11.3.3 Reset()	36
5.11.3.4 SetAmp()	36
5.11.3.5 SetFreq()	36
5.11.3.6 SetPw()	36
5.11.3.7 SetWaveform()	36
5.12 daisysp::Chorus Class Reference	37
5.12.1 Detailed Description	37
5.12.2 Member Function Documentation	37
5.12.2.1 GetLeft()	37
5.12.2.2 GetRight()	38
5.12.2.3 Init()	38
5.12.2.4 Process()	38
5.12.2.5 SetDelay() [1/2]	38
5.12.2.6 SetDelay() [2/2]	38

---

5.12.2.7 SetDelayMs() [1/2]	39
5.12.2.8 SetDelayMs() [2/2]	39
5.12.2.9 SetFeedback() [1/2]	39
5.12.2.10 SetFeedback() [2/2]	40
5.12.2.11 SetLfoDepth() [1/2]	40
5.12.2.12 SetLfoDepth() [2/2]	40
5.12.2.13 SetLfoFreq() [1/2]	41
5.12.2.14 SetLfoFreq() [2/2]	41
5.12.2.15 SetPan() [1/2]	41
5.12.2.16 SetPan() [2/2]	41
5.13 daisysp::ChorusEngine Class Reference	42
5.13.1 Detailed Description	42
5.13.2 Member Function Documentation	42
5.13.2.1 Init()	42
5.13.2.2 Process()	43
5.13.2.3 SetDelay()	43
5.13.2.4 SetDelayMs()	43
5.13.2.5 SetFeedback()	44
5.13.2.6 SetLfoDepth()	44
5.13.2.7 SetLfoFreq()	44
5.14 daisysp::ClockedNoise Class Reference	44
5.14.1 Detailed Description	45
5.14.2 Member Function Documentation	45
5.14.2.1 Init()	45
5.14.2.2 Process()	45
5.14.2.3 SetFreq()	46
5.14.2.4 Sync()	46
5.15 daisysp::Comb Class Reference	46
5.15.1 Detailed Description	46
5.15.2 Member Function Documentation	46
5.15.2.1 Init()	46
5.15.2.2 Process()	47
5.15.2.3 SetFreq()	47
5.15.2.4 SetPeriod()	47
5.15.2.5 SetRevTime()	47
5.16 daisysp::Compressor Class Reference	47
5.16.1 Detailed Description	48
5.16.2 Member Function Documentation	48
5.16.2.1 Apply()	48
5.16.2.2 AutoMakeup()	49
5.16.2.3 GetAttack()	49
5.16.2.4 GetGain()	49

5.16.2.5 GetMakeup()	49
5.16.2.6 GetRatio()	49
5.16.2.7 GetRelease()	50
5.16.2.8 GetThreshold()	50
5.16.2.9 Init()	50
5.16.2.10 Process() [1/2]	50
5.16.2.11 Process() [2/2]	50
5.16.2.12 ProcessBlock() [1/3]	51
5.16.2.13 ProcessBlock() [2/3]	51
5.16.2.14 ProcessBlock() [3/3]	52
5.16.2.15 SetAttack()	52
5.16.2.16 SetMakeup()	52
5.16.2.17 SetRatio()	52
5.16.2.18 SetRelease()	53
5.16.2.19 SetThreshold()	53
5.17 daisysp::CrossFade Class Reference	53
5.17.1 Detailed Description	54
5.17.2 Member Function Documentation	54
5.17.2.1 GetCurve()	54
5.17.2.2 GetPos()	54
5.17.2.3 Init() [1/2]	54
5.17.2.4 Init() [2/2]	54
5.17.2.5 Process()	55
5.17.2.6 SetCurve()	55
5.17.2.7 SetPos()	55
5.18 daisysp::DcBlock Class Reference	55
5.18.1 Detailed Description	55
5.18.2 Member Function Documentation	55
5.18.2.1 Init()	56
5.18.2.2 Process()	56
5.19 daisysp::Decimator Class Reference	56
5.19.1 Detailed Description	56
5.19.2 Member Function Documentation	56
5.19.2.1 GetBitcrushFactor()	56
5.19.2.2 GetDownsampleFactor()	57
5.19.2.3 Init()	57
5.19.2.4 Process()	57
5.19.2.5 SetBitcrushFactor()	57
5.19.2.6 SetBitsToCrush()	57
5.19.2.7 SetDownsampleFactor()	57
5.20 daisysp::DelayLine< T, max_size > Class Template Reference	58
5.20.1 Detailed Description	58

---

5.20.2 Member Function Documentation	58
5.20.2.1 Init()	58
5.20.2.2 Read() [1/2]	58
5.20.2.3 Read() [2/2]	59
5.20.2.4 Reset()	59
5.20.2.5 SetDelay() [1/2]	59
5.20.2.6 SetDelay() [2/2]	59
5.20.2.7 Write()	59
5.21 daisysp::Drip Class Reference	59
5.21.1 Detailed Description	60
5.21.2 Member Function Documentation	60
5.21.2.1 Init()	60
5.21.2.2 Process()	60
5.22 daisysp::Dust Class Reference	61
5.22.1 Detailed Description	61
5.23 daisysp::Flanger Class Reference	62
5.23.1 Detailed Description	62
5.23.2 Member Function Documentation	62
5.23.2.1 Init()	62
5.23.2.2 Process()	62
5.23.2.3 SetDelay()	63
5.23.2.4 SetDelayMs()	63
5.23.2.5 SetFeedback()	63
5.23.2.6 SetLfoDepth()	64
5.23.2.7 SetLfoFreq()	64
5.24 daisysp::Fm2 Class Reference	64
5.24.1 Detailed Description	64
5.24.2 Member Function Documentation	65
5.24.2.1 GetIndex()	65
5.24.2.2 Init()	65
5.24.2.3 Process()	65
5.24.2.4 Reset()	65
5.24.2.5 SetFrequency()	65
5.24.2.6 SetIndex()	66
5.24.2.7 SetRatio()	66
5.25 daisysp::Fold Class Reference	66
5.25.1 Detailed Description	66
5.25.2 Member Function Documentation	67
5.25.2.1 Init()	67
5.25.2.2 Process()	67
5.25.2.3 SetIncrement()	67
5.26 daisysp::FormantOscillator Class Reference	67



---

5.26.1 Detailed Description	68
5.26.2 Member Function Documentation	68
5.26.2.1 Init()	68
5.26.2.2 Process()	68
5.26.2.3 SetCarrierFreq()	68
5.26.2.4 SetFormantFreq()	69
5.26.2.5 SetPhaseShift()	69
5.27 daisysp::FractalRandomGenerator< T, order > Class Template Reference	69
5.27.1 Detailed Description	70
5.27.2 Member Function Documentation	70
5.27.2.1 Init()	70
5.27.2.2 Process()	70
5.27.2.3 SetColor()	71
5.27.2.4 SetFreq()	71
5.28 daisysp::GrainletOscillator Class Reference	71
5.28.1 Detailed Description	71
5.28.2 Member Function Documentation	72
5.28.2.1 Init()	72
5.28.2.2 Process()	72
5.28.2.3 SetBleed()	72
5.28.2.4 SetFormantFreq()	73
5.28.2.5 SetFreq()	73
5.28.2.6 SetShape()	73
5.29 daisysp::HarmonicOscillator< num_harmonics > Class Template Reference	73
5.29.1 Detailed Description	74
5.29.2 Member Function Documentation	74
5.29.2.1 Init()	74
5.29.2.2 Process()	75
5.29.2.3 SetAmplitudes()	75
5.29.2.4 SetFirstHarmlIdx()	75
5.29.2.5 SetFreq()	75
5.29.2.6 SetSingleAmp()	76
5.30 daisysp::HiHat< MetallicNoiseSource, VCA, resonance > Class Template Reference	76
5.30.1 Detailed Description	76
5.30.2 Member Function Documentation	77
5.30.2.1 Init()	77
5.30.2.2 Process()	77
5.30.2.3 SetAccent()	78
5.30.2.4 SetDecay()	78
5.30.2.5 SetFreq()	78
5.30.2.6 SetNoisiness()	78
5.30.2.7 SetSustain()	79

---

5.30.2.8 SetTone()	79
5.30.2.9 Trig()	79
5.31 daisysp::Jitter Class Reference	80
5.31.1 Detailed Description	80
5.31.2 Member Function Documentation	80
5.31.2.1 Init()	80
5.31.2.2 Process()	80
5.31.2.3 SetAmp()	81
5.31.2.4 SetCpsMax()	81
5.31.2.5 SetCpsMin()	81
5.32 daisysp::Limiter Class Reference	81
5.32.1 Detailed Description	82
5.32.2 Member Function Documentation	82
5.32.2.1 Init()	82
5.32.2.2 ProcessBlock()	82
5.33 daisysp::Line Class Reference	82
5.33.1 Detailed Description	83
5.33.2 Member Function Documentation	83
5.33.2.1 Init()	83
5.33.2.2 Process()	83
5.33.2.3 Start()	83
5.34 daisysp::LinearVCA Class Reference	84
5.34.1 Detailed Description	84
5.35 daisysp::Maytrig Class Reference	84
5.35.1 Detailed Description	84
5.35.2 Member Function Documentation	85
5.35.2.1 Process()	85
5.36 daisysp::Metro Class Reference	86
5.36.1 Detailed Description	86
5.36.2 Member Function Documentation	86
5.36.2.1 GetFreq()	86
5.36.2.2 Init()	87
5.36.2.3 Process()	87
5.36.2.4 Reset()	87
5.36.2.5 SetFreq()	87
5.37 daisysp::ModalVoice Class Reference	87
5.37.1 Detailed Description	88
5.37.2 Member Function Documentation	88
5.37.2.1 GetAux()	88
5.37.2.2 Init()	88
5.37.2.3 Process()	89
5.37.2.4 SetAccent()	89

---

5.37.2.5 SetBrightness()	89
5.37.2.6 SetDamping()	90
5.37.2.7 SetFreq()	90
5.37.2.8 SetStructure()	90
5.37.2.9 SetSustain()	90
5.37.2.10 Trig()	91
5.38 daisysp::Mode Class Reference	91
5.38.1 Detailed Description	91
5.38.2 Member Function Documentation	91
5.38.2.1 Clear()	92
5.38.2.2 Init()	92
5.38.2.3 Process()	92
5.38.2.4 SetFreq()	92
5.38.2.5 SetQ()	92
5.39 daisysp::MoogLadder Class Reference	92
5.39.1 Detailed Description	93
5.39.2 Member Function Documentation	93
5.39.2.1 Init()	93
5.39.2.2 Process()	93
5.39.2.3 SetFreq()	93
5.39.2.4 SetRes()	94
5.40 daisysp::NIFilt Class Reference	94
5.40.1 Detailed Description	94
5.40.2 Member Function Documentation	95
5.40.2.1 Init()	95
5.40.2.2 ProcessBlock()	95
5.40.2.3 SetA()	95
5.40.2.4 SetB()	95
5.40.2.5 SetC()	95
5.40.2.6 SetCoefficients()	95
5.40.2.7 SetD()	96
5.40.2.8 SetL()	96
5.41 daisysp::Oscillator Class Reference	96
5.41.1 Detailed Description	96
5.41.2 Member Enumeration Documentation	97
5.41.2.1 anonymous enum	97
5.41.3 Member Function Documentation	97
5.41.3.1 Init()	97
5.41.3.2 IsEOC()	97
5.41.3.3 IsEOR()	97
5.41.3.4 IsFalling()	98
5.41.3.5 IsRising()	98

---

5.41.3.6 PhaseAdd()	98
5.41.3.7 Process()	98
5.41.3.8 Reset()	98
5.41.3.9 SetAmp()	98
5.41.3.10 SetFreq()	98
5.41.3.11 SetWaveform()	99
5.42 daisysp::OscillatorBank Class Reference	99
5.42.1 Detailed Description	99
5.42.2 Member Function Documentation	99
5.42.2.1 Init()	99
5.42.2.2 Process()	100
5.42.2.3 SetAmplitudes()	100
5.42.2.4 SetFreq()	100
5.42.2.5 SetGain()	100
5.42.2.6 SetSingleAmp()	101
5.43 daisysp::Overdrive Class Reference	101
5.43.1 Detailed Description	101
5.43.2 Member Function Documentation	102
5.43.2.1 Init()	102
5.43.2.2 Process()	102
5.43.2.3 SetDrive()	102
5.44 daisysp::Particle Class Reference	102
5.44.1 Detailed Description	103
5.44.2 Member Function Documentation	103
5.44.2.1 GetNoise()	103
5.44.2.2 Init()	103
5.44.2.3 Process()	104
5.44.2.4 SetDensity()	104
5.44.2.5 SetFreq()	104
5.44.2.6 SetGain()	104
5.44.2.7 SetRandomFreq()	105
5.44.2.8 SetResonance()	105
5.44.2.9 SetSpread()	105
5.44.2.10 SetSync()	106
5.45 daisysp::Phasor Class Reference	106
5.45.1 Detailed Description	106
5.45.2 Member Function Documentation	106
5.45.2.1 GetFreq()	106
5.45.2.2 Init() [1/3]	107
5.45.2.3 Init() [2/3]	107
5.45.2.4 Init() [3/3]	107
5.45.2.5 Process()	107

---

5.45.2.6 SetFreq()	107
5.46 daisysp::PitchShifter Class Reference	108
5.46.1 Detailed Description	108
5.46.2 Member Function Documentation	108
5.46.2.1 Init()	108
5.46.2.2 Process()	108
5.46.2.3 SetDelSize()	109
5.46.2.4 SetFun()	109
5.46.2.5 SetTransposition()	109
5.47 daisysp::Pluck Class Reference	109
5.47.1 Detailed Description	110
5.47.2 Member Function Documentation	110
5.47.2.1 GetAmp()	110
5.47.2.2 GetDamp()	110
5.47.2.3 GetDecay()	110
5.47.2.4 GetFreq()	110
5.47.2.5 GetMode()	110
5.47.2.6 Init()	111
5.47.2.7 Process()	111
5.47.2.8 SetAmp()	111
5.47.2.9 SetDamp()	111
5.47.2.10 SetDecay()	111
5.47.2.11 SetFreq()	111
5.47.2.12 SetMode()	112
5.48 daisysp::PolyPluck< num_voices > Class Template Reference	112
5.48.1 Detailed Description	112
5.48.2 Member Function Documentation	112
5.48.2.1 Init()	112
5.48.2.2 Process()	113
5.48.2.3 SetDecay()	113
5.49 daisysp::Port Class Reference	113
5.49.1 Detailed Description	114
5.49.2 Member Function Documentation	114
5.49.2.1 GetHtime()	114
5.49.2.2 Init()	114
5.49.2.3 Process()	114
5.49.2.4 SetHtime()	115
5.50 daisysp::Resonator Class Reference	115
5.50.1 Detailed Description	115
5.50.2 Member Function Documentation	116
5.50.2.1 Init()	116
5.50.2.2 Process()	117

---

5.50.2.3 SetBrightness()	117
5.50.2.4 SetDamping()	117
5.50.2.5 SetFreq()	118
5.50.2.6 SetStructure()	118
5.51 daisysp::ResonatorSvf< batch_size > Class Template Reference	118
5.51.1 Detailed Description	119
5.52 daisysp::ReverbSc Class Reference	119
5.52.1 Detailed Description	119
5.52.2 Member Function Documentation	120
5.52.2.1 Init()	120
5.52.2.2 Process()	120
5.52.2.3 SetFeedback()	120
5.52.2.4 SetLpFreq()	120
5.53 daisysp::ReverbScDI Struct Reference	121
5.53.1 Detailed Description	121
5.53.2 Member Data Documentation	121
5.53.2.1 buf	121
5.53.2.2 buffer_size	121
5.53.2.3 dummy	121
5.53.2.4 filter_state	122
5.53.2.5 rand_line_cnt	122
5.53.2.6 read_pos	122
5.53.2.7 read_pos_frac	122
5.53.2.8 read_pos_frac_inc	122
5.53.2.9 seed_val	122
5.53.2.10 write_pos	122
5.54 daisysp::RingModNoise Class Reference	123
5.54.1 Detailed Description	123
5.55 daisysp::SampleHold Class Reference	123
5.55.1 Detailed Description	124
5.55.2 Member Function Documentation	124
5.55.2.1 Process()	124
5.56 daisysp::SampleRateReducer Class Reference	124
5.56.1 Detailed Description	125
5.56.2 Member Function Documentation	125
5.56.2.1 Init()	125
5.56.2.2 Process()	125
5.56.2.3 SetFreq()	125
5.57 daisysp::SmoothRandomGenerator Class Reference	126
5.57.1 Detailed Description	126
5.57.2 Member Function Documentation	126
5.57.2.1 Init()	126

---

5.57.2.2 Process()	127
5.57.2.3 SetFreq()	127
5.58 daisysp::SquareNoise Class Reference	127
5.58.1 Detailed Description	127
5.59 daisysp::String Class Reference	128
5.59.1 Detailed Description	128
5.59.2 Member Function Documentation	128
5.59.2.1 Init()	128
5.59.2.2 Process()	129
5.59.2.3 Reset()	129
5.59.2.4 SetBrightness()	129
5.59.2.5 SetDamping()	129
5.59.2.6 SetFreq()	130
5.59.2.7 SetNonLinearity()	130
5.60 daisysp::StringVoice Class Reference	130
5.60.1 Detailed Description	131
5.60.2 Member Function Documentation	131
5.60.2.1 GetAux()	131
5.60.2.2 Init()	131
5.60.2.3 Process()	132
5.60.2.4 Reset()	132
5.60.2.5 SetAccent()	132
5.60.2.6 SetBrightness()	132
5.60.2.7 SetDamping()	133
5.60.2.8 SetFreq()	133
5.60.2.9 SetStructure()	133
5.60.2.10 SetSustain()	134
5.60.2.11 Trig()	134
5.61 daisysp::Svf Class Reference	134
5.61.1 Detailed Description	134
5.61.2 Member Function Documentation	135
5.61.2.1 Band()	135
5.61.2.2 High()	135
5.61.2.3 Init()	135
5.61.2.4 Low()	135
5.61.2.5 Notch()	136
5.61.2.6 Peak()	136
5.61.2.7 Process()	136
5.61.2.8 SetDrive()	136
5.61.2.9 SetFreq()	136
5.61.2.10 SetRes()	137
5.62 daisysp::SwingVCA Class Reference	137

---

5.62.1 Detailed Description	137
5.63 daisysp::SyntheticBassDrum Class Reference	137
5.63.1 Detailed Description	138
5.63.2 Member Function Documentation	138
5.63.2.1 DistortedSine()	138
5.63.2.2 Init()	138
5.63.2.3 Process()	139
5.63.2.4 SetAccent()	139
5.63.2.5 SetDecay()	139
5.63.2.6 SetDirtiness()	140
5.63.2.7 SetFmEnvelopeAmount()	140
5.63.2.8 SetFmEnvelopeDecay()	140
5.63.2.9 SetFreq()	140
5.63.2.10 SetSustain()	141
5.63.2.11 SetTone()	141
5.63.2.12 TransistorVCA()	141
5.63.2.13 Trig()	142
5.64 daisysp::SyntheticBassDrumAttackNoise Class Reference	142
5.64.1 Detailed Description	142
5.64.2 Member Function Documentation	142
5.64.2.1 Init()	142
5.64.2.2 Process()	143
5.65 daisysp::SyntheticBassDrumClick Class Reference	143
5.65.1 Detailed Description	143
5.65.2 Member Function Documentation	143
5.65.2.1 Init()	143
5.65.2.2 Process()	144
5.66 daisysp::SyntheticSnareDrum Class Reference	144
5.66.1 Detailed Description	145
5.66.2 Member Function Documentation	145
5.66.2.1 Init()	145
5.66.2.2 Process()	145
5.66.2.3 SetAccent()	146
5.66.2.4 SetDecay()	146
5.66.2.5 SetFmAmount()	146
5.66.2.6 SetFreq()	147
5.66.2.7 SetSnappy()	147
5.66.2.8 SetSustain()	147
5.66.2.9 Trig()	147
5.67 daisysp::Tone Class Reference	148
5.67.1 Detailed Description	148
5.67.2 Member Function Documentation	148



---

5.67.2.1	GetFreq()	148
5.67.2.2	Init()	148
5.67.2.3	Process()	148
5.67.2.4	SetFreq()	148
5.68	daisysp::Tremolo Class Reference	149
5.68.1	Detailed Description	149
5.68.2	Member Function Documentation	149
5.68.2.1	Init()	149
5.68.2.2	Process()	150
5.68.2.3	SetDepth()	150
5.68.2.4	SetFreq()	150
5.68.2.5	SetWaveform()	151
5.69	daisysp::VariableSawOscillator Class Reference	151
5.69.1	Detailed Description	151
5.69.2	Member Function Documentation	152
5.69.2.1	Process()	152
5.69.2.2	SetFreq()	152
5.69.2.3	SetPW()	152
5.69.2.4	SetWaveshape()	152
5.70	daisysp::VariableShapeOscillator Class Reference	153
5.70.1	Detailed Description	153
5.70.2	Member Function Documentation	153
5.70.2.1	Init()	153
5.70.2.2	Process()	154
5.70.2.3	SetFreq()	154
5.70.2.4	SetPW()	154
5.70.2.5	SetSync()	154
5.70.2.6	SetSyncFreq()	155
5.70.2.7	SetWaveshape()	155
5.71	daisysp::VosimOscillator Class Reference	155
5.71.1	Detailed Description	156
5.71.2	Member Function Documentation	156
5.71.2.1	Init()	156
5.71.2.2	Process()	156
5.71.2.3	SetForm1Freq()	156
5.71.2.4	SetForm2Freq()	157
5.71.2.5	SetFreq()	157
5.71.2.6	SetShape()	157
5.72	daisysp::WhiteNoise Class Reference	158
5.72.1	Detailed Description	158
5.72.2	Member Function Documentation	158
5.72.2.1	Init()	158

5.72.2.2 Process()	158
5.72.2.3 SetAmp()	158
5.73 daisysp::ZOscillator Class Reference	159
5.73.1 Detailed Description	159
5.73.2 Member Function Documentation	159
5.73.2.1 Init()	159
5.73.2.2 Process()	160
5.73.2.3 SetFormantFreq()	160
5.73.2.4 SetFreq()	160
5.73.2.5 SetMode()	160
5.73.2.6 SetShape()	161
<b>6 File Documentation</b>	<b>163</b>
6.1 Source/Drums/analogbassdrum.h File Reference	163
6.2 Source/Drums/analogsnaredrum.h File Reference	163
6.3 Source/Drums/hihat.h File Reference	163
6.4 Source/Drums/synthbassdrum.h File Reference	164
6.5 Source/Drums/synthsnaresdrum.h File Reference	164
6.6 Source/Effects/chorus.h File Reference	164
6.7 Source/Effects/flanger.h File Reference	165
6.8 Source/Effects/overdrive.h File Reference	165
6.9 Source/Effects/sampleratereducer.h File Reference	165
6.10 Source/Effects/tremolo.h File Reference	165
6.11 Source/Filters/allpass.h File Reference	166
6.12 Source/Noise/clockednoise.h File Reference	166
6.13 Source/Noise/dust.h File Reference	166
6.14 Source/Noise/fractal_noise.h File Reference	166
6.15 Source/Noise/grainlet.h File Reference	167
6.16 Source/Noise/particle.h File Reference	167
6.17 Source/PhysicalModeling/drip.h File Reference	167
6.18 Source/PhysicalModeling/KarplusString.h File Reference	167
6.19 Source/PhysicalModeling/modalvoice.h File Reference	168
6.20 Source/PhysicalModeling/resonator.h File Reference	168
6.21 Source/PhysicalModeling/stringvoice.h File Reference	168
6.22 Source/Synthesis/formantosc.h File Reference	169
6.23 Source/Synthesis/harmonic_osc.h File Reference	169
6.24 Source/Synthesis/oscillatorbank.h File Reference	169
6.25 Source/Synthesis/variablesawosc.h File Reference	169
6.26 Source/Synthesis/variableshapeosc.h File Reference	170
6.27 Source/Synthesis/vosim.h File Reference	170
6.28 Source/Synthesis/zoscillator.h File Reference	170
6.29 Source/Utility/smooth_random.h File Reference	170





# Chapter 1

## Main Page

### DaisySP A Powerful, Open Source DSP Library in C++

[Applications](#) • [Features](#) • [Examples](#) • [Getting Started](#) • [Community](#) • [Contributing](#)  
• [License](#)

#### 1.0.1 Applications

- Embedded hardware using the [Daisy Audio Platform](#)
- Audio plug-ins (VST, AU, [JUCE](#))
- Mobile apps (iOS, Android)
- [VCV Rack](#) modules

#### 1.0.2 Features

- **Synthesis Methods:** Subtractive, Physical Modeling, FM
- **Filters:** Biquad, State-Variable, Modal, Comb
- **Effects Processors:** Reverb, Delay, Decimate, Compressor
- **Utilities:** Math Functions, Signal Conditioning, Aleatoric Generators

### 1.0.3 Code Example

### 1.0.4 Getting Started

- Get the source: `git clone https://github.com/electro-smith/DaisySP`
- Navigate to the DaisySP repo: `cd DaisySP`
- Build the library: `make`
- Make some noise with the `example programs!`

### 1.0.5 Community

Connect with other users and developers:

- Join the [Daisy Forum](#)
- Chat on the [Daisy Slack Workspace](#)

### 1.0.6 Contributing

Here are some ways that you can get involved:

- Proof read the [documentation](#) and suggest improvements
- Test existing functionality and make [issues](#)
- Make new DSP modules. See issues labeled "feature"
- Port existing DSP modules from other open source projects (MIT). See issues labeled "port"
- Fix problems with existing modules. See issues labeled "bug" and/or "polish"

Before working on code, please check out our [Contribution Guidelines](#) and [Style Guide](#).

### 1.0.7 License

DaisySP uses the MIT license.

It can be used in both closed source and commercial projects, and does not provide a warranty of any kind.

For the full license, read the [LICENSE](#) file in the root directory.

## Chapter 2

### Todo List

**Class `daisysp::AdEnv`**

- Add Cycling
- Implement Curve (its only linear for now).
- Maybe make this an `ADsr_` that has `AD/AR/Asr_` modes.

**Class `daisysp::Compressor`**

Add soft/hard knee settings

**Class `daisysp::NIFilt`**

make this work on a single sample instead of just on blocks at a time.

**Class `daisysp::Phasor`**

Selecting which channels should be initialized/included in the sequence conversion.

Setup a similar start function for an external mux, but that seems outside the scope of this file.

**Class `daisysp::PitchShifter`**

- move `hash_xs32` and `myrand` to `dsp.h` and give appropriate names





# Chapter 3

## Class Index

### 3.1 Class List

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

<a href="#">daisysp::AdEnv</a>	11
<a href="#">daisysp::Adsr</a>	13
<a href="#">daisysp::Allpass</a>	16
<a href="#">daisysp::AnalogBassDrum</a>	
808 bass drum model, revisited	17
<a href="#">daisysp::AnalogSnareDrum</a>	
808 snare drum model, revisited	21
<a href="#">daisysp::ATone</a>	25
<a href="#">daisysp::Autowah</a>	27
<a href="#">daisysp::Balance</a>	29
<a href="#">daisysp::Biquad</a>	30
<a href="#">daisysp::Bitcrush</a>	33
<a href="#">daisysp::BIOsc</a>	34
<a href="#">daisysp::Chorus</a>	
Chorus Effect	37
<a href="#">daisysp::ChorusEngine</a>	
Single Chorus engine. Used in <a href="#">Chorus</a>	42
<a href="#">daisysp::ClockedNoise</a>	44
<a href="#">daisysp::Comb</a>	46
<a href="#">daisysp::Compressor</a>	47
<a href="#">daisysp::CrossFade</a>	53
<a href="#">daisysp::DcBlock</a>	55
<a href="#">daisysp::Decimator</a>	56
<a href="#">daisysp::DelayLine&lt; T, max_size &gt;</a>	58
<a href="#">daisysp::Drip</a>	59
<a href="#">daisysp::Dust</a>	
Dust Module	61
<a href="#">daisysp::Flanger</a>	
Flanging Audio Effect	62
<a href="#">daisysp::Fm2</a>	64
<a href="#">daisysp::Fold</a>	66
<a href="#">daisysp::FormantOscillator</a>	
Formant <a href="#">Oscillator</a> Module	67
<a href="#">daisysp::FractalRandomGenerator&lt; T, order &gt;</a>	
Fractal Noise, stacks octaves of a noise source	69

<a href="#">daisysp::GrainletOscillator</a>	
Granular <a href="#">Oscillator</a> Module	71
<a href="#">daisysp::HarmonicOscillator&lt; num_harmonics &gt;</a>	
Harmonic <a href="#">Oscillator</a> Module based on Chebyshev polynomials	73
<a href="#">daisysp::HiHat&lt; MetallicNoiseSource, VCA, resonance &gt;</a>	
808 HH, with a few extra parameters to push things to the CY territory.	76
<a href="#">daisysp::Jitter</a>	80
<a href="#">daisysp::Limiter</a>	81
<a href="#">daisysp::Line</a>	82
<a href="#">daisysp::LinearVCA</a>	
Linear type VCA	84
<a href="#">daisysp::Maytrig</a>	84
<a href="#">daisysp::Metro</a>	86
<a href="#">daisysp::ModalVoice</a>	
Simple modal synthesis voice with a mallet exciter: click -> LPF -> resonator	87
<a href="#">daisysp::Mode</a>	91
<a href="#">daisysp::MoogLadder</a>	92
<a href="#">daisysp::NIFilt</a>	94
<a href="#">daisysp::Oscillator</a>	96
<a href="#">daisysp::OscillatorBank</a>	
<a href="#">Oscillator</a> Bank module	99
<a href="#">daisysp::Overdrive</a>	
Distortion / <a href="#">Overdrive</a> Module	101
<a href="#">daisysp::Particle</a>	
Random impulse train processed by a resonant filter	102
<a href="#">daisysp::Phasor</a>	106
<a href="#">daisysp::PitchShifter</a>	108
<a href="#">daisysp::Pluck</a>	109
<a href="#">daisysp::PolyPluck&lt; num_voices &gt;</a>	112
<a href="#">daisysp::Port</a>	113
<a href="#">daisysp::Resonator</a>	
Resonant Body Simulation	115
<a href="#">daisysp::ResonatorSvf&lt; batch_size &gt;</a>	
SVF for use in the <a href="#">Resonator</a> Class	118
<a href="#">daisysp::ReverbSc</a>	119
<a href="#">daisysp::ReverbScDI</a>	121
<a href="#">daisysp::RingModNoise</a>	
Ring mod style metallic noise generator	123
<a href="#">daisysp::SampleHold</a>	123
<a href="#">daisysp::SampleRateReducer</a>	
Sample rate reducer	124
<a href="#">daisysp::SmoothRandomGenerator</a>	
Smooth random generator for internal modulation.	126
<a href="#">daisysp::SquareNoise</a>	
808 style "metallic noise" with 6 square oscillators	127
<a href="#">daisysp::String</a>	
Comb filter / KS string	128
<a href="#">daisysp::StringVoice</a>	
Extended Karplus-Strong, with all the niceties from Rings	130
<a href="#">daisysp::Svf</a>	134
<a href="#">daisysp::SwingVCA</a>	
Swing type VCA	137
<a href="#">daisysp::SyntheticBassDrum</a>	
Naive bass drum model (modulated oscillator with FM + envelope)	137
<a href="#">daisysp::SyntheticBassDrumAttackNoise</a>	
Attack Noise generator for <a href="#">SyntheticBassDrum</a>	142

---

<a href="#">daisysp::SyntheticBassDrumClick</a>	
Click noise for <a href="#">SyntheticBassDrum</a> . . . . .	143
<a href="#">daisysp::SyntheticSnareDrum</a>	
Naive snare drum model (two modulated oscillators + filtered noise) . . . . .	144
<a href="#">daisysp::Tone</a> . . . . .	148
<a href="#">daisysp::Tremolo</a>	
Tremolo effect . . . . .	149
<a href="#">daisysp::VariableSawOscillator</a>	
Variable Saw <a href="#">Oscillator</a> . . . . .	151
<a href="#">daisysp::VariableShapeOscillator</a>	
Variable Waveshape <a href="#">Oscillator</a> . . . . .	153
<a href="#">daisysp::VosimOscillator</a>	
Vosim <a href="#">Oscillator</a> Module	
155	
<a href="#">daisysp::WhiteNoise</a> . . . . .	158
<a href="#">daisysp::ZOscillator</a>	
<a href="#">ZOscillator</a> Module	
159	



# Chapter 4

## File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

Source/ <b>daisysp.h</b>	??
Source/Control/ <b>adenv.h</b>	??
Source/Control/ <b>adsr.h</b>	??
Source/Control/ <b>line.h</b>	??
Source/Control/ <b>phasor.h</b>	??
Source/Drums/ <b>analogbassdrum.h</b>	163
Source/Drums/ <b>analogsnaredrum.h</b>	163
Source/Drums/ <b>hihat.h</b>	163
Source/Drums/ <b>synthbassdrum.h</b>	164
Source/Drums/ <b>synthsnaredrum.h</b>	164
Source/Dynamics/ <b>balance.h</b>	??
Source/Dynamics/ <b>compressor.h</b>	??
Source/Dynamics/ <b>crossfade.h</b>	??
Source/Dynamics/ <b>limiter.h</b>	??
Source/Effects/ <b>autowah.h</b>	??
Source/Effects/ <b>bitcrush.h</b>	??
Source/Effects/ <b>chorus.h</b>	164
Source/Effects/ <b>decimator.h</b>	??
Source/Effects/ <b>flanger.h</b>	165
Source/Effects/ <b>fold.h</b>	??
Source/Effects/ <b>overdrive.h</b>	165
Source/Effects/ <b>pitchshifter.h</b>	??
Source/Effects/ <b>reverbosc.h</b>	??
Source/Effects/ <b>sampleratereducer.h</b>	165
Source/Effects/ <b>tremolo.h</b>	165
Source/Filters/ <b>allpass.h</b>	166
Source/Filters/ <b>atone.h</b>	??
Source/Filters/ <b>biquad.h</b>	??
Source/Filters/ <b>comb.h</b>	??
Source/Filters/ <b>mode.h</b>	??
Source/Filters/ <b>moogladder.h</b>	??
Source/Filters/ <b>nlfilt.h</b>	??
Source/Filters/ <b>svf.h</b>	??
Source/Filters/ <b>tone.h</b>	??
Source/Noise/ <b>clockednoise.h</b>	166

Source/Noise/ <a href="#">dust.h</a>	166
Source/Noise/ <a href="#">fractal_noise.h</a>	166
Source/Noise/ <a href="#">grainlet.h</a>	167
Source/Noise/ <a href="#">particle.h</a>	167
Source/Noise/ <b>whitenoise.h</b>	??
Source/PhysicalModeling/ <a href="#">drip.h</a>	167
Source/PhysicalModeling/ <a href="#">KarplusString.h</a>	167
Source/PhysicalModeling/ <a href="#">modalvoice.h</a>	168
Source/PhysicalModeling/ <b>pluck.h</b>	??
Source/PhysicalModeling/ <b>PolyPluck.h</b>	??
Source/PhysicalModeling/ <a href="#">resonator.h</a>	168
Source/PhysicalModeling/ <a href="#">stringvoice.h</a>	168
Source/Synthesis/ <b>blosc.h</b>	??
Source/Synthesis/ <b>fm2.h</b>	??
Source/Synthesis/ <a href="#">formantosc.h</a>	169
Source/Synthesis/ <a href="#">harmonic_osc.h</a>	169
Source/Synthesis/ <b>oscillator.h</b>	??
Source/Synthesis/ <a href="#">oscillatorbank.h</a>	169
Source/Synthesis/ <a href="#">variablesawosc.h</a>	169
Source/Synthesis/ <a href="#">variableshapeosc.h</a>	170
Source/Synthesis/ <a href="#">vosim.h</a>	170
Source/Synthesis/ <a href="#">zoscillator.h</a>	170
Source/Utility/ <b>dcblock.h</b>	??
Source/Utility/ <b>delayline.h</b>	??
Source/Utility/ <b>dsp.h</b>	??
Source/Utility/ <b>jitter.h</b>	??
Source/Utility/ <b>maytrig.h</b>	??
Source/Utility/ <b>metro.h</b>	??
Source/Utility/ <b>port.h</b>	??
Source/Utility/ <b>samplehold.h</b>	??
Source/Utility/ <a href="#">smooth_random.h</a>	170

## Chapter 5

# Class Documentation

### 5.1 daisysp::AdEnv Class Reference

```
#include <adenv.h>
```

#### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [Trigger](#) ()
- void [SetTime](#) (uint8\_t seg, float time)
- void [SetCurve](#) (float scalar)
- void [SetMin](#) (float min)
- void [SetMax](#) (float max)
- float [GetValue](#) () const
- uint8\_t [GetCurrentSegment](#) ()
- bool [IsRunning](#) () const

#### 5.1.1 Detailed Description

Trigger-able envelope with adjustable min/max, and independent per-segment time control.

##### Author

shensley

- Todo**
- Add Cycling
  - Implement Curve (its only linear for now).
  - Maybe make this an ADsr\_ that has AD/AR/Asr\_ modes.

#### 5.1.2 Member Function Documentation

### 5.1.2.1 GetCurrentSegment()

```
uint8_t daisysp::AdEnv::GetCurrentSegment ( ) [inline]
```

Returns the segment of the envelope that the phase is currently located in.

### 5.1.2.2 GetValue()

```
float daisysp::AdEnv::GetValue ( ) const [inline]
```

Returns the current output value without processing the next sample

### 5.1.2.3 Init()

```
void AdEnv::Init (
    float sample_rate )
```

Initializes the ad envelope.

Defaults:

- current segment = idle
- curve = linear
- phase = 0
- min = 0
- max = 1

#### Parameters

<i>sample_rate</i>	sample rate of the audio engine being run
--------------------	---

### 5.1.2.4 IsRunning()

```
bool daisysp::AdEnv::IsRunning ( ) const [inline]
```

Returns true if the envelope is currently in any stage apart from idle.

### 5.1.2.5 Process()

```
float AdEnv::Process ( )
```

Processes the current sample of the envelope. This should be called once per sample period.

#### Returns

the current envelope value.



### 5.1.2.6 SetCurve()

```
void daisysp::AdEnv::SetCurve (
    float scalar ) [inline]
```

Sets the amount of curve applied. A positive value will create a log curve. Input range: -100 to 100. (or more)

### 5.1.2.7 SetMax()

```
void daisysp::AdEnv::SetMax (
    float max ) [inline]
```

Sets the maximum value of the envelope output. Input range: -FLTmax\_ to FLTmax\_

### 5.1.2.8 SetMin()

```
void daisysp::AdEnv::SetMin (
    float min ) [inline]
```

Sets the minimum value of the envelope output. Input range: -FLTmax\_ to FLTmax\_

### 5.1.2.9 SetTime()

```
void daisysp::AdEnv::SetTime (
    uint8_t seg,
    float time ) [inline]
```

Sets the length of time (in seconds) for a specific segment.

### 5.1.2.10 Trigger()

```
void daisysp::AdEnv::Trigger ( ) [inline]
```

Starts or retriggers the envelope.

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

- Source/Control/adenv.h
- Source/Control/adenv.cpp

## 5.2 daisysp::Adsr Class Reference

```
#include <adsr.h>
```

## Public Member Functions

- void [Init](#) (float *sample\_rate*)
- float [Process](#) (bool *gate*)
- void [SetTime](#) (int *seg*, float *time*)
- void [SetSustainLevel](#) (float *sus\_level*)
- uint8\_t [GetCurrentSegment](#) ()
- bool [IsRunning](#) () const

### 5.2.1 Detailed Description

adsr envelope module

Original author(s) : Paul Batchelor

Ported from Soundpipe by Ben Sergentanis, May 2020

### 5.2.2 Member Function Documentation

#### 5.2.2.1 GetCurrentSegment()

```
uint8_t daisysp::Adsr::GetCurrentSegment ( ) [inline]
```

get the current envelope segment

#### Returns

the segment of the envelope that the phase is currently located in.

#### 5.2.2.2 Init()

```
void Adsr::Init (
    float sample_rate )
```

Initializes the [Adsr](#) module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.2.2.3 IsRunning()

```
bool daisysp::Adsr::IsRunning ( ) const [inline]
```

Tells whether envelope is active

#### Returns

true if the envelope is currently in any stage apart from idle.

### 5.2.2.4 Process()

```
float Adsr::Process (
    bool gate )
```

Processes one sample through the filter and returns one sample.

#### Parameters

<i>gate</i>	- trigger the envelope, hold it to sustain
-------------	--

### 5.2.2.5 SetSustainLevel()

```
void daisysp::Adsr::SetSustainLevel (
    float sus_level ) [inline]
```

Sustain level

#### Parameters

<i>sus_level</i>	- sets sustain level
------------------	----------------------

### 5.2.2.6 SetTime()

```
void daisysp::Adsr::SetTime (
    int seg,
    float time ) [inline]
```

Sets time Set time per segment in seconds

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

- Source/Control/adsr.h
- Source/Control/adsr.cpp

## 5.3 daisyp::Allpass Class Reference

```
#include <allpass.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate, float \*buff, size\_t size)
- float [Process](#) (float in)
- void [SetFreq](#) (float looptime)
- void [SetRevTime](#) (float revtime)

### 5.3.1 Detailed Description

[Allpass](#) filter module

Passes all frequencies at their original levels, with a phase shift.  
Ported from soundpipe by Ben Sergentanis, May 2020

#### Author

Barry Vercoe, John ffitch

#### Date

1991

### 5.3.2 Member Function Documentation

#### 5.3.2.1 Init()

```
void Allpass::Init (
    float sample_rate,
    float * buff,
    size_t size )
```

Initializes the allpass module.  
\param sample\_rate The sample rate of the audio engine being run.

#### Parameters

<i>buff</i>	Buffer for allpass to use.
<i>size</i>	Size of buff.

### 5.3.2.2 Process()

```
float Allpass::Process (
    float in )
```

#### Parameters

<i>in</i>	Input sample.
-----------	---------------

#### Returns

Next floating point sample.

### 5.3.2.3 SetFreq()

```
void Allpass::SetFreq (
    float looptime )
```

Sets the filter frequency (Implemented by delay time).

#### Parameters

<i>looptime</i>	Filter looptime in seconds.
-----------------	-----------------------------

### 5.3.2.4 SetRevTime()

```
void daisysp::Allpass::SetRevTime (
    float revtime ) [inline]
```

#### Parameters

<i>revtime</i>	Reverb time in seconds.
----------------	-------------------------

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

- Source/Filters/[allpass.h](#)
- Source/Filters/allpass.cpp

## 5.4 daisysp::AnalogBassDrum Class Reference

808 bass drum model, revisited.

```
#include <analogbassdrum.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (bool trigger=false)
- void [Trig](#) ()
- void [SetSustain](#) (bool sustain)
- void [SetAccent](#) (float accent)
- void [SetFreq](#) (float f0)
- void [SetTone](#) (float tone)
- void [SetDecay](#) (float decay)
- void [SetAttackFmAmount](#) (float attack\_fm\_amount)
- void [SetSelfFmAmount](#) (float self\_fm\_amount)

### 5.4.1 Detailed Description

808 bass drum model, revisited.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/analog\_bass\_drum.h to an independent module.  
Original code written by Emilie Gillet in 2016.

### 5.4.2 Member Function Documentation

#### 5.4.2.1 Init()

```
void AnalogBassDrum::Init (
    float sample_rate )
```

Initialize the module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

#### 5.4.2.2 Process()

```
float AnalogBassDrum::Process (
    bool trigger = false )
```

Get the next sample.

#### Parameters

<i>trigger</i>	True strikes the drum. Defaults to false.
----------------	---

### 5.4.2.3 SetAccent()

```
void AnalogBassDrum::SetAccent (
    float accent )
```

Set a small accent.

#### Parameters

<i>accent</i>	Works 0-1
---------------	-----------

### 5.4.2.4 SetAttackFmAmount()

```
void AnalogBassDrum::SetAttackFmAmount (
    float attack_fm_amount )
```

Set the amount of fm attack. Works together with self fm.

#### Parameters

<i>attack_fm_amount</i>	Works best 0-1.
-------------------------	-----------------

### 5.4.2.5 SetDecay()

```
void AnalogBassDrum::SetDecay (
    float decay )
```

Set the decay length of the drum.

#### Parameters

<i>decay</i>	Works best 0-1.
--------------	-----------------

#### 5.4.2.6 SetFreq()

```
void AnalogBassDrum::SetFreq (
    float f0 )
```

Set the drum's root frequency

##### Parameters

<i>f0</i>	Frequency in Hz
-----------	-----------------

#### 5.4.2.7 SetSelfFmAmount()

```
void AnalogBassDrum::SetSelfFmAmount (
    float self_fm_amount )
```

Set the amount of self fm. Also affects fm attack, and volume decay.

##### Parameters

<i>self_fm_amount</i>	Works best 0-1.
-----------------------	-----------------

#### 5.4.2.8 SetSustain()

```
void AnalogBassDrum::SetSustain (
    bool sustain )
```

Set the bassdrum to play infinitely

##### Parameters

<i>sustain</i>	True = infinite length
----------------	------------------------

#### 5.4.2.9 SetTone()

```
void AnalogBassDrum::SetTone (
    float tone )
```

Set the amount of click.



## Parameters

<i>tone</i>	Works 0-1.
-------------	------------

#### 5.4.2.10 Trig()

```
void AnalogBassDrum::Trig ( )
```

Strikes the drum.

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

- Source/Drums/[analogbassdrum.h](#)
- Source/Drums/[analogbassdrum.cpp](#)

## 5.5 daisysp::AnalogSnareDrum Class Reference

808 snare drum model, revisited.

```
#include <analogsnaredrum.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (bool trigger=false)
- void [Trig](#) ()
- void [SetSustain](#) (bool sustain)
- void [SetAccent](#) (float accent)
- void [SetFreq](#) (float f0)
- void [SetTone](#) (float tone)
- void [SetDecay](#) (float decay)
- void [SetSnappy](#) (float snappy)

### Static Public Attributes

- static const int **kNumModes** = 5

### 5.5.1 Detailed Description

808 snare drum model, revisited.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/analog\_snare\_drum.h to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.5.2 Member Function Documentation

### 5.5.2.1 Init()

```
void AnalogSnareDrum::Init (
    float sample_rate )
```

Init the module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

### 5.5.2.2 Process()

```
float AnalogSnareDrum::Process (
    bool trigger = false )
```

Get the next sample

#### Parameters

<i>trigger</i>	Hit the drum with true. Defaults to false.
----------------	--

### 5.5.2.3 SetAccent()

```
void AnalogSnareDrum::SetAccent (
    float accent )
```

Set how much accent to use

#### Parameters

<i>accent</i>	Works 0-1.
---------------	------------

### 5.5.2.4 SetDecay()

```
void AnalogSnareDrum::SetDecay (
    float decay )
```

Set the length of the drum decay

## Parameters

<i>decay</i>	Works with positive numbers
--------------	-----------------------------

**5.5.2.5 SetFreq()**

```
void AnalogSnareDrum::SetFreq (
    float f0 )
```

Set the drum's root frequency

## Parameters

<i>f0</i>	Freq in Hz
-----------	------------

**5.5.2.6 SetSnappy()**

```
void AnalogSnareDrum::SetSnappy (
    float snappy )
```

Sets the mix between snare and drum.

## Parameters

<i>snappy</i>	1 = just snare. 0 = just drum.
---------------	--------------------------------

**5.5.2.7 SetSustain()**

```
void AnalogSnareDrum::SetSustain (
    bool sustain )
```

Init the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

### 5.5.2.8 SetTone()

```
void AnalogSnareDrum::SetTone (
    float tone )
```

Set the brightness of the drum tone.

#### Parameters

<i>tone</i>	Works 0-1. 1 = bright, 0 = dark.
-------------	----------------------------------

### 5.5.2.9 Trig()

```
void AnalogSnareDrum::Trig ( )
```

Trigger the drum

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

- Source/Drums/[analogsnaredrum.h](#)
- Source/Drums/[analogsnaredrum.cpp](#)

## 5.6 daisysp::ATone Class Reference

```
#include <atone.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float &in)
- void [SetFreq](#) (float &freq)
- float [GetFreq](#) ()

### 5.6.1 Detailed Description

A first-order recursive high-pass filter with variable frequency response. Original Author(s): Barry Vercoe, John Ffitch, Gabriel Maldonado

Year: 1991

Original Location: Csound – OOps/ugens5.c

Ported from soundpipe by Ben Sergentanis, May 2020

## 5.6.2 Member Function Documentation

### 5.6.2.1 GetFreq()

```
float daisysp::ATone::GetFreq ( ) [inline]
```

get current frequency

#### Returns

the current value for the cutoff frequency or half-way point of the filter.

### 5.6.2.2 Init()

```
void ATone::Init (
    float sample_rate )
```

Initializes the [ATone](#) module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.6.2.3 Process()

```
float ATone::Process (
    float & in )
```

Processes one sample through the filter and returns one sample.

#### Parameters

<i>in</i>	- input signal
-----------	----------------

### 5.6.2.4 SetFreq()

```
void daisysp::ATone::SetFreq (
    float & freq ) [inline]
```

Sets the cutoff frequency or half-way point of the filter.

#### Parameters

<i>freq</i>	- frequency value in Hz. Range: Any positive value.
-------------	---

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

- Source/Filters/atone.h
- Source/Filters/atone.cpp

## 5.7 daisysp::Autowah Class Reference

```
#include <autowah.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetWah](#) (float wah)
- void [SetDryWet](#) (float drywet)
- void [SetLevel](#) (float level)

### 5.7.1 Detailed Description

[Autowah](#) module

Original author(s) :

Ported from soundpipe by Ben Sergentanis, May 2020

### 5.7.2 Member Function Documentation

#### 5.7.2.1 Init()

```
void Autowah::Init (  
    float sample_rate )
```

Initializes the [Autowah](#) module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.7.2.2 Process()

```
float Autowah::Process (  
    float in )
```

Initializes the [Autowah](#) module.

#### Parameters

<i>in</i>	- input signal to be wah'd
-----------	----------------------------

### 5.7.2.3 SetDryWet()

```
void daisysp::Autowah::SetDryWet (  
    float drywet ) [inline]
```

sets mix amount

#### Parameters

<i>drywet</i>	: set effect dry/wet
---------------	----------------------

### 5.7.2.4 SetLevel()

```
void daisysp::Autowah::SetLevel (  
    float level ) [inline]
```

sets wah level

#### Parameters

<i>level</i>	: set wah level
--------------	-----------------

### 5.7.2.5 SetWah()

```
void daisysp::Autowah::SetWah (  
    float wah ) [inline]
```

sets wah



#### Parameters

<i>wah</i>	: set wah amount
------------	------------------

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

- Source/Effects/autowah.h
- Source/Effects/autowah.cpp

## 5.8 daisysp::Balance Class Reference

```
#include <balance.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float sig, float comp)
- void [SetCutoff](#) (float cutoff)

### 5.8.1 Detailed Description

Balances two sound sources. Sig is boosted to the level of comp.

Original author(s) : Barry Vercoe, john ffitch, Gabriel Maldonado

Year: 1991

Ported from soundpipe by Ben Sergentanis, May 2020

### 5.8.2 Member Function Documentation

#### 5.8.2.1 Init()

```
void Balance::Init (
    float sample_rate )
```

Initializes the balance module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.8.2.2 Process()

```
float Balance::Process (
    float sig,
    float comp )
```

adjust sig level to level of comp

### 5.8.2.3 SetCutoff()

```
void daisysp::Balance::SetCutoff (
    float cutoff ) [inline]
```

adjusts the rate at which level compensation happens

#### Parameters

<i>cutoff</i>	: Sets half power point of special internal cutoff filter.
---------------	--

defaults to 10

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

- Source/Dynamics/balance.h
- Source/Dynamics/balance.cpp

## 5.9 daisysp::Biquad Class Reference

```
#include <biquad.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetRes](#) (float res)
- void [SetCutoff](#) (float cutoff)

### 5.9.1 Detailed Description

Two pole recursive filter

Original author(s) : Hans Mikelson

Year: 1998

Ported from soundpipe by Ben Sergentanis, May 2020

## 5.9.2 Member Function Documentation

### 5.9.2.1 Init()

```
void Biquad::Init (
    float sample_rate )
```

Initializes the biquad module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.9.2.2 Process()

```
float Biquad::Process (
    float in )
```

Filters the input signal

#### Returns

filtered output

### 5.9.2.3 SetCutoff()

```
void daisysp::Biquad::SetCutoff (
    float cutoff ) [inline]
```

Sets filter cutoff in Hz

#### Parameters

<i>cutoff</i>	: Set filter cutoff.
---------------	----------------------

### 5.9.2.4 SetRes()

```
void daisysp::Biquad::SetRes (
    float res ) [inline]
```

Sets resonance amount

**Parameters**

<i>res</i>	: Set filter resonance.
------------	-------------------------

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

- Source/Filters/biquad.h
- Source/Filters/biquad.cpp

## 5.10 daisysp::Bitcrush Class Reference

```
#include <bitcrush.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetBitDepth](#) (int bitdepth)
- void [SetCrushRate](#) (float crushrate)

### 5.10.1 Detailed Description

bitcrush module

Original author(s) : Paul Batchelor,

Ported from soundpipe by Ben Sergentanis, May 2020

### 5.10.2 Member Function Documentation

#### 5.10.2.1 Init()

```
void Bitcrush::Init (  
    float sample_rate )
```

Initializes the bitcrush module.

**Parameters**

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

### 5.10.2.2 Process()

```
float Bitcrush::Process (
    float in )
```

bit crushes and downsamples the input

### 5.10.2.3 SetBitDepth()

```
void daisysp::Bitcrush::SetBitDepth (
    int bitdepth ) [inline]
```

adjusts bitdepth

#### Parameters

<i>bitdepth</i>	: Sets bit depth.
-----------------	-------------------

### 5.10.2.4 SetCrushRate()

```
void daisysp::Bitcrush::SetCrushRate (
    float crushrate ) [inline]
```

adjusts the downsampling frequency

#### Parameters

<i>crushrate</i>	: Sets rate to downsample to.
------------------	-------------------------------

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

- Source/Effects/bitcrush.h
- Source/Effects/bitcrush.cpp

## 5.11 daisysp::BIOsc Class Reference

```
#include <blosc.h>
```

### Public Types

- enum [Waveforms](#) { `WAVE_TRIANGLE`, `WAVE_SAW`, `WAVE_SQUARE`, `WAVE_OFF` }

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetAmp](#) (float amp)
- void [SetPw](#) (float pw)
- void [SetWaveform](#) (uint8\_t waveform)
- void [Reset](#) ()

### 5.11.1 Detailed Description

Band Limited [Oscillator](#)

Based on bltriangle, blsaw, blsquare from soundpipe

Original Author(s): Paul Batchelor, saw2 Faust by Julius Smith

Ported by Ben Sergentanis, May 2020

### 5.11.2 Member Enumeration Documentation

#### 5.11.2.1 Waveforms

```
enum daisysp::BlOsc::Waveforms
```

Bl Waveforms

### 5.11.3 Member Function Documentation

#### 5.11.3.1 Init()

```
void BlOsc::Init (  
    float sample_rate )
```

-Initialize oscillator. -Defaults to: 440Hz, .5 amplitude, .5 pw, Triangle.

#### 5.11.3.2 Process()

```
float BlOsc::Process ( )
```

- Get next floating point oscillator sample.

### 5.11.3.3 Reset()

```
void BlOsc::Reset ( )
```

- reset the phase of the oscillator.

### 5.11.3.4 SetAmp()

```
void daisysp::BlOsc::SetAmp (
    float amp ) [inline]
```

- Float amp: Set oscillator amplitude, 0 to 1.

### 5.11.3.5 SetFreq()

```
void daisysp::BlOsc::SetFreq (
    float freq ) [inline]
```

- Float freq: Set oscillator frequency in Hz.

### 5.11.3.6 SetPw()

```
void daisysp::BlOsc::SetPw (
    float pw ) [inline]
```

- Float pw: Set square osc pulsewidth, 0 to 1. (no thru 0 at the moment)

### 5.11.3.7 SetWaveform()

```
void daisysp::BlOsc::SetWaveform (
    uint8_t waveform ) [inline]
```

- uint8\_t waveform: select between waveforms from enum above.
- i.e. SetWaveform(BL\_WAVEFORM\_SAW); to set waveform to saw

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

- Source/Synthesis/blosc.h
- Source/Synthesis/blosc.cpp



## 5.12 daisysp::Chorus Class Reference

[Chorus](#) Effect.

```
#include <chorus.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- float [GetLeft](#) ()
- float [GetRight](#) ()
- void [SetPan](#) (float panl, float panr)
- void [SetPan](#) (float pan)
- void [SetLfoDepth](#) (float depthl, float depthr)
- void [SetLfoDepth](#) (float depth)
- void [SetLfoFreq](#) (float freql, float freqr)
- void [SetLfoFreq](#) (float freq)
- void [SetDelay](#) (float delayl, float delayr)
- void [SetDelay](#) (float delay)
- void [SetDelayMs](#) (float msl, float msr)
- void [SetDelayMs](#) (float ms)
- void [SetFeedback](#) (float feedbackl, float feedbackr)
- void [SetFeedback](#) (float feedback)

### 5.12.1 Detailed Description

[Chorus](#) Effect.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Based on <https://www.izotope.com/en/learn/understanding-chorus-flangers-and-phasers.html>  
and [https://www.researchgate.net/publication/236629475\\_Implementing\\_Professional\\_Audio\\_Effects\\_with\\_DSPs](https://www.researchgate.net/publication/236629475_Implementing_Professional_Audio_Effects_with_DSPs)

### 5.12.2 Member Function Documentation

#### 5.12.2.1 [GetLeft\(\)](#)

```
float Chorus::GetLeft ( )
```

Get the left channel's last sample

### 5.12.2.2 GetRight()

```
float Chorus::GetRight ( )
```

Get the right channel's last sample

### 5.12.2.3 Init()

```
void Chorus::Init (
    float sample_rate )
```

Initialize the module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

### 5.12.2.4 Process()

```
float Chorus::Process (
    float in )
```

Get the net floating point sample. Defaults to left channel.

#### Parameters

<i>in</i>	Sample to process
-----------	-------------------

### 5.12.2.5 SetDelay() [1/2]

```
void Chorus::SetDelay (
    float delay )
```

Set both channel delay amounts.

#### Parameters

<i>delay</i>	Both channel delay amount. Works 0-1.
--------------	---------------------------------------

### 5.12.2.6 SetDelay() [2/2]

```
void Chorus::SetDelay (
```

```
float delayl,
float delayr )
```

Set both channel delay amounts individually.

#### Parameters

<i>delayl</i>	Left channel delay amount. Works 0-1.
<i>delayr</i>	Right channel delay amount.

#### 5.12.2.7 SetDelayMs() [1/2]

```
void Chorus::SetDelayMs (
    float ms )
```

Set both channel delay in ms.

#### Parameters

<i>ms</i>	Both channel delay amounts in ms.
-----------	-----------------------------------

#### 5.12.2.8 SetDelayMs() [2/2]

```
void Chorus::SetDelayMs (
    float msl,
    float msr )
```

Set both channel delay individually.

#### Parameters

<i>msl</i>	Left channel delay in ms.
<i>msr</i>	Right channel delay in ms.

#### 5.12.2.9 SetFeedback() [1/2]

```
void Chorus::SetFeedback (
    float feedback )
```

Set both channels feedback.

## Parameters

<i>feedback</i>	Both channel feedback. Works 0-1.
-----------------	-----------------------------------

**5.12.2.10 SetFeedback()** [2/2]

```
void Chorus::SetFeedback (
    float feedbackl,
    float feedbackr )
```

Set both channels feedback individually.

## Parameters

<i>feedbackl</i>	Left channel feedback. Works 0-1.
<i>feedbackr</i>	Right channel feedback.

**5.12.2.11 SetLfoDepth()** [1/2]

```
void Chorus::SetLfoDepth (
    float depth )
```

Set both lfo depths.

## Parameters

<i>depth</i>	Both channels lfo depth. Works 0-1.
--------------	-------------------------------------

**5.12.2.12 SetLfoDepth()** [2/2]

```
void Chorus::SetLfoDepth (
    float depthl,
    float depthr )
```

Set both lfo depths individually.

## Parameters

<i>depthl</i>	Left channel lfo depth. Works 0-1.
<i>depthr</i>	Right channel lfo depth.

### 5.12.2.13 SetLfoFreq() [1/2]

```
void Chorus::SetLfoFreq (
    float freq )
```

Set both lfo frequencies.

#### Parameters

<i>depth</i>	Both channel lfo freqs in Hz.
--------------	-------------------------------

### 5.12.2.14 SetLfoFreq() [2/2]

```
void Chorus::SetLfoFreq (
    float freql,
    float freqr )
```

Set both lfo frequencies individually.

#### Parameters

<i>depthl</i>	Left channel lfo freq in Hz.
<i>depthr</i>	Right channel lfo freq in Hz.

### 5.12.2.15 SetPan() [1/2]

```
void Chorus::SetPan (
    float pan )
```

Pan both channels.

#### Parameters

<i>pan</i>	Where to pan both channels to. 0 is left, 1 is right.
------------	---

### 5.12.2.16 SetPan() [2/2]

```
void Chorus::SetPan (
    float panl,
    float panr )
```

Pan both channels individually.

#### Parameters

<i>panl</i>	Pan the left channel. 0 is left, 1 is right.
<i>panr</i>	Pan the right channel.

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

- Source/Effects/[chorus.h](#)
- Source/Effects/chorus.cpp

## 5.13 daisysp::ChorusEngine Class Reference

Single [Chorus](#) engine. Used in [Chorus](#).

```
#include <chorus.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetLfoDepth](#) (float depth)
- void [SetLfoFreq](#) (float freq)
- void [SetDelay](#) (float delay)
- void [SetDelayMs](#) (float ms)
- void [SetFeedback](#) (float feedback)

### 5.13.1 Detailed Description

Single [Chorus](#) engine. Used in [Chorus](#).

#### Author

Ben Sergentanis

### 5.13.2 Member Function Documentation

#### 5.13.2.1 Init()

```
void ChorusEngine::Init (  
    float sample_rate )
```

Initialize the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

**5.13.2.2 Process()**

```
float ChorusEngine::Process (  
    float in )
```

Get the next sample

## Parameters

<i>in</i>	Sample to process
-----------	-------------------

**5.13.2.3 SetDelay()**

```
void ChorusEngine::SetDelay (  
    float delay )
```

Set the internal delay rate.

## Parameters

<i>delay</i>	Tuned for 0-1. Maps to .1 to 50 ms.
--------------	-------------------------------------

**5.13.2.4 SetDelayMs()**

```
void ChorusEngine::SetDelayMs (  
    float ms )
```

Set the delay time in ms.

## Parameters

<i>ms</i>	Delay time in ms.
-----------	-------------------

### 5.13.2.5 SetFeedback()

```
void ChorusEngine::SetFeedback (
    float feedback )
```

Set the feedback amount.

#### Parameters

<i>feedback</i>	Amount from 0-1.
-----------------	------------------

### 5.13.2.6 SetLfoDepth()

```
void ChorusEngine::SetLfoDepth (
    float depth )
```

How much to modulate the delay by.

#### Parameters

<i>depth</i>	Works 0-1.
--------------	------------

### 5.13.2.7 SetLfoFreq()

```
void ChorusEngine::SetLfoFreq (
    float freq )
```

Set lfo frequency.

#### Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

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

- Source/Effects/[chorus.h](#)
- Source/Effects/chorus.cpp

## 5.14 daisysp::ClockedNoise Class Reference

```
#include <clockednoise.h>
```



## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [Sync](#) ()

### 5.14.1 Detailed Description

@brief Clocked Noise Module

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 Noise processed by a sample and hold running at a target frequency.

Ported from pichenettes/eurorack/plaits/dsp/noise/clocked\_noise.h to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.14.2 Member Function Documentation

#### 5.14.2.1 Init()

```
void ClockedNoise::Init (  
    float sample_rate )
```

Initialize module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

#### 5.14.2.2 Process()

```
float ClockedNoise::Process ( )
```

Get the next floating point sample

### 5.14.2.3 SetFreq()

```
void ClockedNoise::SetFreq (
    float freq )
```

Set the frequency at which the next sample is generated.

#### Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

### 5.14.2.4 Sync()

```
void ClockedNoise::Sync ( )
```

Calling this forces another random float to be generated

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

- Source/Noise/[clockednoise.h](#)
- Source/Noise/[clockednoise.cpp](#)

## 5.15 daisysp::Comb Class Reference

```
#include <comb.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate, float \*buff, size\_t size)
- float [Process](#) (float in)
- void [SetPeriod](#) (float looptime)
- void [SetFreq](#) (float freq)
- void [SetRevTime](#) (float revtime)

### 5.15.1 Detailed Description

[Comb](#) filter module

Original author(s) :

Ported from soundpipe by Ben Sergentanis, May 2020

### 5.15.2 Member Function Documentation

#### 5.15.2.1 Init()

```
void Comb::Init (
    float sample_rate,
    float * buff,
    size_t size )
```

Initializes the [Comb](#) module.

## Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
<i>buff</i>	- input buffer, kept in either main() or global space
<i>size</i>	- size of buff

**5.15.2.2 Process()**

```
float Comb::Process (
    float in )
```

processes the comb filter

**5.15.2.3 SetFreq()**

```
void daisysp::Comb::SetFreq (
    float freq ) [inline]
```

Sets the frequency of the comb filter in Hz

**5.15.2.4 SetPeriod()**

```
void Comb::SetPeriod (
    float looptime )
```

Sets the period of the comb filter in seconds

**5.15.2.5 SetRevTime()**

```
void daisysp::Comb::SetRevTime (
    float revtime ) [inline]
```

Sets the decay time of the comb filter

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

- Source/Filters/comb.h
- Source/Filters/comb.cpp

**5.16 daisysp::Compressor Class Reference**

```
#include <compressor.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- float [Process](#) (float in, float key)
- float [Apply](#) (float in)
- void [ProcessBlock](#) (float \*in, float \*out, size\_t size)
- void [ProcessBlock](#) (float \*in, float \*out, float \*key, size\_t size)
- void [ProcessBlock](#) (float \*\*in, float \*\*out, float \*key, size\_t channels, size\_t size)
- float [GetRatio](#) ()
- void [SetRatio](#) (float ratio)
- float [GetThreshold](#) ()
- void [SetThreshold](#) (float threshold)
- float [GetAttack](#) ()
- void [SetAttack](#) (float attack)
- float [GetRelease](#) ()
- void [SetRelease](#) (float release)
- float [GetMakeup](#) ()
- void [SetMakeup](#) (float gain)
- void [AutoMakeup](#) (bool enable)
- float [GetGain](#) ()

### 5.16.1 Detailed Description

dynamics compressor

influenced by compressor in soundpipe (from faust).

Modifications made to do:

- Less calculations during each process loop (coefficients recalculated on parameter change).
- C++-ified
- added sidechain support
- pulled gain apart for monitoring and multichannel support
- improved readability
- improved makeup-gain calculations
- changing controls now costs a lot less
- a lot less expensive

by: shensley, improved upon by AvAars

**Todo** Add soft/hard knee settings

### 5.16.2 Member Function Documentation

#### 5.16.2.1 Apply()

```
float daisysp::Compressor::Apply (
    float in ) [inline]
```

Apply compression to the audio signal, based on the previously calculated gain

**Parameters**

<i>in</i>	audio input signal
-----------	--------------------

**5.16.2.2 AutoMakeup()**

```
void daisysp::Compressor::AutoMakeup (
    bool enable ) [inline]
```

Enables or disables the automatic makeup gain. Disabling sets the makeup gain to 0.0

**Parameters**

<i>enable</i>	true to enable, false to disable
---------------	----------------------------------

**5.16.2.3 GetAttack()**

```
float daisysp::Compressor::GetAttack ( ) [inline]
```

Gets the envelope time for onset of compression

**5.16.2.4 GetGain()**

```
float daisysp::Compressor::GetGain ( ) [inline]
```

Gets the gain reduction in dB

**5.16.2.5 GetMakeup()**

```
float daisysp::Compressor::GetMakeup ( ) [inline]
```

Gets the additional gain to make up for the compression

**5.16.2.6 GetRatio()**

```
float daisysp::Compressor::GetRatio ( ) [inline]
```

Gets the amount of gain reduction

### 5.16.2.7 GetRelease()

```
float daisysp::Compressor::GetRelease ( ) [inline]
```

Gets the envelope time for release of compression

### 5.16.2.8 GetThreshold()

```
float daisysp::Compressor::GetThreshold ( ) [inline]
```

Gets the threshold in dB

### 5.16.2.9 Init()

```
void Compressor::Init (
    float sample_rate )
```

Initializes compressor

#### Parameters

<i>sample_rate</i>	rate at which samples will be produced by the audio engine.
--------------------	---

### 5.16.2.10 Process() [1/2]

```
float Compressor::Process (
    float in )
```

Compress the audio input signal, saves the calculated gain

#### Parameters

<i>in</i>	audio input signal
-----------	--------------------

### 5.16.2.11 Process() [2/2]

```
float daisysp::Compressor::Process (
    float in,
    float key ) [inline]
```

Compresses the audio input signal, keyed by a secondary input.

## Parameters

<i>in</i>	audio input signal (to be compressed)
<i>key</i>	audio input that will be used to side-chain the compressor

**5.16.2.12 ProcessBlock() [1/3]**

```
void Compressor::ProcessBlock (
    float ** in,
    float ** out,
    float * key,
    size_t channels,
    size_t size )
```

Compresses a block of multiple channels of audio, keyed by a secondary input

## Parameters

<i>in</i>	audio input signals (to be compressed)
<i>out</i>	audio output signals
<i>key</i>	audio input that will be used to side-chain the compressor
<i>channels</i>	the number of audio channels
<i>size</i>	the size of the block

**5.16.2.13 ProcessBlock() [2/3]**

```
void Compressor::ProcessBlock (
    float * in,
    float * out,
    float * key,
    size_t size )
```

Compresses a block of audio, keyed by a secondary input

## Parameters

<i>in</i>	audio input signal (to be compressed)
<i>out</i>	audio output signal
<i>key</i>	audio input that will be used to side-chain the compressor
<i>size</i>	the size of the block

**5.16.2.14 ProcessBlock()** [3/3]

```
void daisysp::Compressor::ProcessBlock (
    float * in,
    float * out,
    size_t size ) [inline]
```

Compresses a block of audio

**Parameters**

<i>in</i>	audio input signal
<i>out</i>	audio output signal
<i>size</i>	the size of the block

**5.16.2.15 SetAttack()**

```
void daisysp::Compressor::SetAttack (
    float attack ) [inline]
```

Sets the envelope time for onset of compression for signals above the threshold.

**Parameters**

<i>attack</i>	Expects 0.001 -> 10
---------------	---------------------

**5.16.2.16 SetMakeup()**

```
void daisysp::Compressor::SetMakeup (
    float gain ) [inline]
```

Manually sets the additional gain to make up for the compression

**Parameters**

<i>gain</i>	Expects 0.0 -> 80
-------------	-------------------

**5.16.2.17 SetRatio()**

```
void daisysp::Compressor::SetRatio (
    float ratio ) [inline]
```

Sets the amount of gain reduction applied to compressed signals



## Parameters

<i>ratio</i>	Expects 1.0 -> 40. (untested with values < 1.0)
--------------	---

**5.16.2.18 SetRelease()**

```
void daisysp::Compressor::SetRelease (
    float release ) [inline]
```

Sets the envelope time for release of compression as input signal falls below threshold.

## Parameters

<i>release</i>	Expects 0.001 -> 10
----------------	---------------------

**5.16.2.19 SetThreshold()**

```
void daisysp::Compressor::SetThreshold (
    float threshold ) [inline]
```

Sets the threshold in dB at which compression will be applied

## Parameters

<i>threshold</i>	Expects 0.0 -> -80.
------------------	---------------------

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

- Source/Dynamics/compressor.h
- Source/Dynamics/compressor.cpp

**5.17 daisysp::CrossFade Class Reference**

```
#include <crossfade.h>
```

**Public Member Functions**

- void [Init](#) (int curve)
- void [Init](#) ()
- float [Process](#) (float &in1, float &in2)
- void [SetPos](#) (float pos)
- void [SetCurve](#) (uint8\_t curve)
- float [GetPos](#) (float pos)
- uint8\_t [GetCurve](#) (uint8\_t curve)

### 5.17.1 Detailed Description

Performs a [CrossFade](#) between two signals

Original author: Paul Batchelor

Ported from Soundpipe by Andrew Ikenberry

added curve option for constant power, etc.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 GetCurve()

```
uint8_t daisysp::CrossFade::GetCurve (
    uint8_t curve ) [inline]
```

Returns current curve

#### 5.17.2.2 GetPos()

```
float daisysp::CrossFade::GetPos (
    float pos ) [inline]
```

Returns current position

#### 5.17.2.3 Init() [1/2]

```
void daisysp::CrossFade::Init ( ) [inline]
```

Initialize with default linear curve

#### 5.17.2.4 Init() [2/2]

```
void daisysp::CrossFade::Init (
    int curve ) [inline]
```

Initializes [CrossFade](#) module Defaults

- current position = .5
- curve = linear

### 5.17.2.5 Process()

```
float CrossFade::Process (
    float & in1,
    float & in2 )
```

processes [CrossFade](#) and returns single sample

### 5.17.2.6 SetCurve()

```
void daisysp::CrossFade::SetCurve (
    uint8_t curve ) [inline]
```

Sets current curve applied to [CrossFade](#) Expected input: See [Curve Options](#)

### 5.17.2.7 SetPos()

```
void daisysp::CrossFade::SetPos (
    float pos ) [inline]
```

Sets position of [CrossFade](#) between two input signals Input range: 0 to 1

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

- Source/Dynamics/crossfade.h
- Source/Dynamics/crossfade.cpp

## 5.18 daisysp::DcBlock Class Reference

```
#include <dcblock.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)

### 5.18.1 Detailed Description

Removes DC component of a signal

### 5.18.2 Member Function Documentation

### 5.18.2.1 Init()

```
void DcBlock::Init (
    float sample_rate )
```

Initializes [DcBlock](#) module

### 5.18.2.2 Process()

```
float DcBlock::Process (
    float in )
```

performs [DcBlock](#) Process

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

- Source/Utility/dcblock.h
- Source/Utility/dcblock.cpp

## 5.19 daisysp::Decimator Class Reference

```
#include <decimator.h>
```

### Public Member Functions

- void [Init](#) ()
- float [Process](#) (float input)
- void [SetDownsampleFactor](#) (float downsample\_factor)
- void [SetBitcrushFactor](#) (float bitcrush\_factor)
- void [SetBitsToCrush](#) (const uint8\_t &bits)
- float [GetDownsampleFactor](#) ()
- float [GetBitcrushFactor](#) ()

### 5.19.1 Detailed Description

Performs downsampling and bitcrush effects

## 5.19.2 Member Function Documentation

### 5.19.2.1 GetBitcrushFactor()

```
float daisysp::Decimator::GetBitcrushFactor ( ) [inline]
```

Returns current setting of bitcrush

### 5.19.2.2 GetDownsampleFactor()

```
float daisysp::Decimator::GetDownsampleFactor ( ) [inline]
```

Returns current setting of downsample

### 5.19.2.3 Init()

```
void Decimator::Init ( )
```

Initializes downsample module

### 5.19.2.4 Process()

```
float Decimator::Process (
    float input )
```

Applies downsample and bitcrush effects to input signal.

#### Returns

one sample. This should be called once per sample period.

### 5.19.2.5 SetBitcrushFactor()

```
void daisysp::Decimator::SetBitcrushFactor (
    float bitcrush_factor ) [inline]
```

Sets amount of bitcrushing Input range:

### 5.19.2.6 SetBitsToCrush()

```
void daisysp::Decimator::SetBitsToCrush (
    const uint8_t & bits ) [inline]
```

Sets the exact number of bits to crush 0-16 bits

### 5.19.2.7 SetDownsampleFactor()

```
void daisysp::Decimator::SetDownsampleFactor (
    float downsample_factor ) [inline]
```

Sets amount of downsample Input range:

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

- Source/Effects/decimator.h
- Source/Effects/decimator.cpp

## 5.20 daisysp::DelayLine< T, max\_size > Class Template Reference

```
#include <delayline.h>
```

### Public Member Functions

- void [Init](#) ()
- void [Reset](#) ()
- void [SetDelay](#) (size\_t delay)
- void [SetDelay](#) (float delay)
- void [Write](#) (const T sample)
- const T [Read](#) () const
- const T [Read](#) (float delay) const
- const T [ReadHermite](#) (float delay) const
- const T [Allpass](#) (const T sample, size\_t delay, const T coefficient)

### 5.20.1 Detailed Description

```
template<typename T, size_t max_size>
class daisysp::DelayLine< T, max_size >
```

Simple Delay line. November 2019

Converted to Template December 2019

declaration example: (1 second of floats)

```
DelayLine<float, SAMPLE_RATE> del;
```

By: shensley

### 5.20.2 Member Function Documentation

#### 5.20.2.1 Init()

```
template<typename T , size_t max_size>
void daisysp::DelayLine< T, max_size >::Init ( ) [inline]
```

initializes the delay line by clearing the values within, and setting delay to 1 sample.

#### 5.20.2.2 Read() [1/2]

```
template<typename T , size_t max_size>
const T daisysp::DelayLine< T, max_size >::Read ( ) const [inline]
```

returns the next sample of type T in the delay line, interpolated if necessary.

**5.20.2.3 Read()** [2/2]

```
template<typename T , size_t max_size>
const T daisysp::DelayLine< T, max_size >::Read (
    float delay ) const [inline]
```

Read from a set location

**5.20.2.4 Reset()**

```
template<typename T , size_t max_size>
void daisysp::DelayLine< T, max_size >::Reset ( ) [inline]
```

clears buffer, sets write ptr to 0, and delay to 1 sample.

**5.20.2.5 SetDelay()** [1/2]

```
template<typename T , size_t max_size>
void daisysp::DelayLine< T, max_size >::SetDelay (
    float delay ) [inline]
```

sets the delay time in samples If a float is passed in, a fractional component will be calculated for interpolating the delay line.

**5.20.2.6 SetDelay()** [2/2]

```
template<typename T , size_t max_size>
void daisysp::DelayLine< T, max_size >::SetDelay (
    size_t delay ) [inline]
```

sets the delay time in samples If a float is passed in, a fractional component will be calculated for interpolating the delay line.

**5.20.2.7 Write()**

```
template<typename T , size_t max_size>
void daisysp::DelayLine< T, max_size >::Write (
    const T sample ) [inline]
```

writes the sample of type T to the delay line, and advances the write ptr

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

- Source/Utility/delayline.h

**5.21 daisysp::Drip Class Reference**

```
#include <drip.h>
```

## Public Member Functions

- void [Init](#) (float *sample\_rate*, float *dettack*)
- float [Process](#) (bool *trig*)

### 5.21.1 Detailed Description

Imitates the sound of dripping water via Physical Modeling Synthesis.  
Ported from soundpipe by Ben Sergentanis, May 2020

#### Author

Perry Cook

#### Date

2000

### 5.21.2 Member Function Documentation

#### 5.21.2.1 Init()

```
void Drip::Init (  
    float sample_rate,  
    float dettack )
```

Initializes the Drip module.  
\param *sample\_rate* The sample rate of the audio engine being run.

#### Parameters

<i>dettack</i>	The period of time over which all sound is stopped.
----------------	---

#### 5.21.2.2 Process()

```
float Drip::Process (  
    bool trig )
```

Process the next floating point sample.



#### Parameters

<i>trig</i>	If true, begins a new drip.
-------------	-----------------------------

#### Returns

Next sample.

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

- [Source/PhysicalModeling/drip.h](#)
- [Source/PhysicalModeling/drip.cpp](#)

## 5.22 daisyp::Dust Class Reference

[Dust](#) Module.

```
#include <dust.h>
```

### Public Member Functions

- void **Init** ()
- float **Process** ()
- void **SetDensity** (float density)

### 5.22.1 Detailed Description

[Dust](#) Module.

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 Randomly Clocked Samples

Ported from pichenettes/eurorack/plaits/dsp/noise/dust.h  
to an independent module.

Original code written by Emilie Gillet in 2016.

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

- [Source/Noise/dust.h](#)

## 5.23 daisysp::Flanger Class Reference

Flanging Audio Effect.

```
#include <flanger.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetFeedback](#) (float feedback)
- void [SetLfoDepth](#) (float depth)
- void [SetLfoFreq](#) (float freq)
- void [SetDelay](#) (float delay)
- void [SetDelayMs](#) (float ms)

### 5.23.1 Detailed Description

Flanging Audio Effect.

Generates a modulating phase shifted copy of a signal, and recombines with the original to create a 'flanging' sound effect.

### 5.23.2 Member Function Documentation

#### 5.23.2.1 Init()

```
void Flanger::Init (  
    float sample_rate )
```

Initialize the modules

Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

#### 5.23.2.2 Process()

```
float Flanger::Process (  
    float in )
```

Get the next sample

## Parameters

<i>in</i>	Sample to process
-----------	-------------------

**5.23.2.3 SetDelay()**

```
void Flanger::SetDelay (
    float delay )
```

Set the internal delay rate.

## Parameters

<i>delay</i>	Tuned for 0-1. Maps to .1 to 7 ms.
--------------	------------------------------------

**5.23.2.4 SetDelayMs()**

```
void Flanger::SetDelayMs (
    float ms )
```

Set the delay time in ms.

## Parameters

<i>ms</i>	Delay time in ms.
-----------	-------------------

**5.23.2.5 SetFeedback()**

```
void Flanger::SetFeedback (
    float feedback )
```

How much of the signal to feedback into the delay line.

## Parameters

<i>feedback</i>	Works 0-1.
-----------------	------------

### 5.23.2.6 SetLfoDepth()

```
void Flanger::SetLfoDepth (
    float depth )
```

How much to modulate the delay by.

#### Parameters

<i>depth</i>	Works 0-1.
--------------	------------

### 5.23.2.7 SetLfoFreq()

```
void Flanger::SetLfoFreq (
    float freq )
```

Set lfo frequency.

#### Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

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

- Source/Effects/[flanger.h](#)
- Source/Effects/[flanger.cpp](#)

## 5.24 daisysp::Fm2 Class Reference

```
#include <fm2.h>
```

### Public Member Functions

- void [Init](#) (float samplerate)
- float [Process](#) ()
- void [SetFrequency](#) (float freq)
- void [SetRatio](#) (float ratio)
- void [SetIndex](#) (float index)
- float [GetIndex](#) ()
- void [Reset](#) ()

### 5.24.1 Detailed Description

Simple 2 operator FM synth voice.

Date: November, 2020

Author: Ben Sergentanis

## 5.24.2 Member Function Documentation

### 5.24.2.1 GetIndex()

```
float Fm2::GetIndex ( )
```

Returns the current FM index.

### 5.24.2.2 Init()

```
void Fm2::Init (
    float samplerate )
```

Initializes the FM2 module.

#### Parameters

<i>samplerate</i>	- The sample rate of the audio engine being run.
-------------------	--

### 5.24.2.3 Process()

```
float Fm2::Process ( )
```

Returns the next sample

### 5.24.2.4 Reset()

```
void Fm2::Reset ( )
```

Resets both oscillators

### 5.24.2.5 SetFrequency()

```
void Fm2::SetFrequency (
    float freq )
```

Carrier freq. setter

#### Parameters

<i>freq</i>	Carrier frequency in Hz
-------------	-------------------------

### 5.24.2.6 SetIndex()

```
void Fm2::SetIndex (
    float index )
```

Index setter

Parameters

<i>FM</i>	depth, 5 = 2PI rads
-----------	---------------------

### 5.24.2.7 SetRatio()

```
void Fm2::SetRatio (
    float ratio )
```

Set modulator freq. relative to carrier

Parameters

<i>ratio</i>	New modulator freq = carrier freq. * ratio
--------------	--

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

- Source/Synthesis/fm2.h
- Source/Synthesis/fm2.cpp

## 5.25 daisysp::Fold Class Reference

```
#include <fold.h>
```

### Public Member Functions

- void [Init](#) ()
- float [Process](#) (float in)
- void [SetIncrement](#) (float incr)

### 5.25.1 Detailed Description

fold module

Original author(s) : John FFitch, Gabriel Maldonado

Year : 1998

Ported from soundpipe by Ben Sergentanis, May 2020

## 5.25.2 Member Function Documentation

### 5.25.2.1 Init()

```
void Fold::Init ( )
```

Initializes the fold module.

### 5.25.2.2 Process()

```
float Fold::Process (
    float in )
```

applies foldvoer distortion to input

### 5.25.2.3 SetIncrement()

```
void daisysp::Fold::SetIncrement (
    float incr ) [inline]
```

#### Parameters

<i>incr</i>	: set fold increment
-------------	----------------------

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

- Source/Effects/fold.h
- Source/Effects/fold.cpp

## 5.26 daisysp::FormantOscillator Class Reference

Formant [Oscillator](#) Module.

```
#include <formantosc.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFormantFreq](#) (float freq)
- void [SetCarrierFreq](#) (float freq)
- void [SetPhaseShift](#) (float ps)

### 5.26.1 Detailed Description

Formant [Oscillator](#) Module.

#### Author

Ben Sergentanis

#### Date

Dec 2020 Sinewave with aliasing-free phase reset.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/formant\_oscillator.h to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.26.2 Member Function Documentation

#### 5.26.2.1 Init()

```
void FormantOscillator::Init (
    float sample_rate )
```

Initializes the [FormantOscillator](#) module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

#### 5.26.2.2 Process()

```
float FormantOscillator::Process ( )
```

Get the next sample

#### 5.26.2.3 SetCarrierFreq()

```
void FormantOscillator::SetCarrierFreq (
    float freq )
```

Set the carrier frequency. This is the "main" frequency.



## Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

**5.26.2.4 SetFormantFreq()**

```
void FormantOscillator::SetFormantFreq (
    float freq )
```

Set the formant frequency.

## Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

**5.26.2.5 SetPhaseShift()**

```
void FormantOscillator::SetPhaseShift (
    float ps )
```

Set the amount of phase shift

## Parameters

<i>ps</i>	Typically 0-1. Works with other values though, including negative.
-----------	--

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

- Source/Synthesis/[formantosc.h](#)
- Source/Synthesis/formantosc.cpp

**5.27 daisysp::FractalRandomGenerator< T, order > Class Template Reference**

Fractal Noise, stacks octaves of a noise source.

```
#include <fractal_noise.h>
```

**Public Member Functions**

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetColor](#) (float color)

### 5.27.1 Detailed Description

```
template<typename T, int order>
class daisysp::FractalRandomGenerator< T, order >
```

Fractal Noise, stacks octaves of a noise source.

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 T is the noise source to use. T must have [SetFreq\(\)](#) and `Init(sample_rate)` functions. Order is the number of noise sources to stack.

Ported from `pichenettes/eurorack/plaits/dsp/noise/fractal_random_generator.h` to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.27.2 Member Function Documentation

#### 5.27.2.1 Init()

```
template<typename T , int order>
void daisysp::FractalRandomGenerator< T, order >::Init (
    float sample_rate ) [inline]
```

Initialize the module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

#### 5.27.2.2 Process()

```
template<typename T , int order>
float daisysp::FractalRandomGenerator< T, order >::Process ( ) [inline]
```

Get the next sample.

### 5.27.2.3 SetColor()

```
template<typename T , int order>
void daisysp::FractalRandomGenerator< T, order >::SetColor (
    float color ) [inline]
```

Sets the amount of high frequency noise. \*\* Works 0-1. 1 is the brightest, and 0 is the darkest.

### 5.27.2.4 SetFreq()

```
template<typename T , int order>
void daisysp::FractalRandomGenerator< T, order >::SetFreq (
    float freq ) [inline]
```

Set the lowest noise frequency.

#### Parameters

<i>freq</i>	Frequency of the lowest noise source in Hz.
-------------	---

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

- Source/Noise/[fractal\\_noise.h](#)

## 5.28 daisysp::GrainletOscillator Class Reference

Granular [Oscillator](#) Module.

```
#include <grainlet.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetFormantFreq](#) (float freq)
- void [SetShape](#) (float shape)
- void [SetBleed](#) (float bleed)

### 5.28.1 Detailed Description

Granular [Oscillator](#) Module.

#### Author

Ben Sergentanis

**Date**

Dec 2020 A phase-distorted single cycle sine \* another continuously running sine, the whole thing synced to a main oscillator.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/grainlet\_oscillator.h to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.28.2 Member Function Documentation

### 5.28.2.1 Init()

```
void GrainletOscillator::Init (
    float sample_rate )
```

Initialize the oscillator

**Parameters**

<i>sample_rate</i>	Sample rate of audio engine
--------------------	-----------------------------

### 5.28.2.2 Process()

```
float GrainletOscillator::Process ( )
```

Get the next sample

### 5.28.2.3 SetBleed()

```
void GrainletOscillator::SetBleed (
    float bleed )
```

Sets the amount of formant to bleed through

**Parameters**

<i>bleed</i>	Works best 0-1
--------------	----------------

#### 5.28.2.4 SetFormantFreq()

```
void GrainletOscillator::SetFormantFreq (
    float freq )
```

Sets the formant frequency

##### Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

#### 5.28.2.5 SetFreq()

```
void GrainletOscillator::SetFreq (
    float freq )
```

Sets the carrier frequency

##### Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

#### 5.28.2.6 SetShape()

```
void GrainletOscillator::SetShape (
    float shape )
```

Waveshaping

##### Parameters

<i>shape</i>	Shapes differently from 0-1, 1-2, and > 2.
--------------	--

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

- Source/Noise/[grainlet.h](#)
- Source/Noise/[grainlet.cpp](#)

## 5.29 daisysp::HarmonicOscillator< num\_harmonics > Class Template Reference

Harmonic [Oscillator](#) Module based on Chebyshev polynomials.

```
#include <harmonic_osc.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetFirstHarmlIdx](#) (int idx)
- void [SetAmplitudes](#) (const float \*amplitudes)
- void [SetSingleAmp](#) (const float amp, int idx)

### 5.29.1 Detailed Description

```
template<int num_harmonics = 16>
class daisysp::HarmonicOscillator< num_harmonics >
```

Harmonic [Oscillator](#) Module based on Chebyshev polynomials.

#### Author

Ben Sergentanis

#### Date

Dec 2020 Harmonic [Oscillator](#) Module based on Chebyshev polynomials  
Works well for a small number of harmonics. For the higher order harmonics.  
We need to reinitialize the recurrence by computing two high harmonics.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/harmonic\_oscillator.h  
to an independent module.  
Original code written by Emilie Gillet in 2016.

### 5.29.2 Member Function Documentation

#### 5.29.2.1 Init()

```
template<int num_harmonics = 16>
void daisysp::HarmonicOscillator< num_harmonics >::Init (
    float sample_rate ) [inline]
```

Initialize harmonic oscillator

#### Parameters

<i>sample_rate</i>	Audio engine samplerate
--------------------	-------------------------

### 5.29.2.2 Process()

```
template<int num_harmonics = 16>
float daisysp::HarmonicOscillator< num_harmonics >::Process ( ) [inline]
```

Get the next floating point sample

### 5.29.2.3 SetAmplitudes()

```
template<int num_harmonics = 16>
void daisysp::HarmonicOscillator< num_harmonics >::SetAmplitudes (
    const float * amplitudes ) [inline]
```

Set the amplitudes of each harmonic of the root.

#### Parameters

<i>amplitudes</i>	Amplitudes to set. Sum of all amplitudes must be < 1. The array referenced must be at least as large as num_harmonics.
-------------------	--

### 5.29.2.4 SetFirstHarmIdx()

```
template<int num_harmonics = 16>
void daisysp::HarmonicOscillator< num_harmonics >::SetFirstHarmIdx (
    int idx ) [inline]
```

Offset the set of harmonics. Passing in 3 means "harmonic 0" is the 3rd harm., 1 is the 4th, etc.

#### Parameters

<i>idx</i>	Default behavior is 1. Values < 0 default to 1.
------------	---

### 5.29.2.5 SetFreq()

```
template<int num_harmonics = 16>
void daisysp::HarmonicOscillator< num_harmonics >::SetFreq (
    float freq ) [inline]
```

Set the main frequency

#### Parameters

<i>freq</i>	Freq to be set in Hz.
-------------	-----------------------

### 5.29.2.6 SetSingleAmp()

```
template<int num_harmonics = 16>
void daisysp::HarmonicOscillator< num_harmonics >::SetSingleAmp (
    const float amp,
    int idx ) [inline]
```

Sets one amplitude. Does nothing if idx out of range.

#### Parameters

<i>amp</i>	Amplitude to set
<i>idx</i>	Which harmonic to set.

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

- Source/Synthesis/[harmonic\\_osc.h](#)

## 5.30 daisysp::HiHat< MetallicNoiseSource, VCA, resonance > Class Template Reference

808 HH, with a few extra parameters to push things to the CY territory...

```
#include <hihat.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (bool trigger=false)
- void [Trig](#) ()
- void [SetSustain](#) (bool sustain)
- void [SetAccent](#) (float accent)
- void [SetFreq](#) (float f0)
- void [SetTone](#) (float tone)
- void [SetDecay](#) (float decay)
- void [SetNoisiness](#) (float noisiness)

#### 5.30.1 Detailed Description

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance = true>
class daisysp::HiHat< MetallicNoiseSource, VCA, resonance >
```

808 HH, with a few extra parameters to push things to the CY territory...



**Author**

Ben Sergentanis

**Date**

Jan 2021 The template parameter MetallicNoiseSource allows another kind of "metallic \n noise" to be used, for results which are more similar to KR-55 or FM hi-hats.

Ported from pichenettes/eurorack/plaits/dsp/drums/hihat.h to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.30.2 Member Function Documentation

### 5.30.2.1 Init()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance = true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::Init (
    float sample_rate ) [inline]
```

Initialize the module

**Parameters**

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

### 5.30.2.2 Process()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance = true>
float daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::Process (
    bool trigger = false ) [inline]
```

Get the next sample

**Parameters**

<i>trigger</i>	Hit the hihat with true. Defaults to false.
----------------	---

### 5.30.2.3 SetAccent()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetAccent (
    float accent ) [inline]
```

Set how much accent to use

#### Parameters

<i>accent</i>	Works 0-1.
---------------	------------

### 5.30.2.4 SetDecay()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetDecay (
    float decay ) [inline]
```

Set the length of the hihat decay

#### Parameters

<i>decay</i>	Works > 0. Tuned for 0-1.
--------------	---------------------------

### 5.30.2.5 SetFreq()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetFreq (
    float f0 ) [inline]
```

Set the hihat tone's root frequency

#### Parameters

<i>f0</i>	Freq in Hz
-----------	------------

### 5.30.2.6 SetNoisiness()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
```

```
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetNoisiness (
    float noisiness ) [inline]
```

Sets the mix between tone and noise

#### Parameters

<i>snappy</i>	1 = just noise. 0 = just tone.
---------------	--------------------------------

#### 5.30.2.7 SetSustain()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetSustain (
    bool sustain ) [inline]
```

Make the hihat ring out infinitely.

#### Parameters

<i>sustain</i>	True = infinite sustain.
----------------	--------------------------

#### 5.30.2.8 SetTone()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::SetTone (
    float tone ) [inline]
```

Set the overall brightness / darkness of the hihat.

#### Parameters

<i>tone</i>	Works from 0-1.
-------------	-----------------

#### 5.30.2.9 Trig()

```
template<typename MetallicNoiseSource = SquareNoise, typename VCA = LinearVCA, bool resonance
= true>
void daisysp::HiHat< MetallicNoiseSource, VCA, resonance >::Trig ( ) [inline]
```

Trigger the hihat

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

- Source/Drums/[hihat.h](#)

## 5.31 daisyp::Jitter Class Reference

```
#include <jitter.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ( )
- void [SetCpsMin](#) (float cps\_min)
- void [SetCpsMax](#) (float cps\_max)
- void [SetAmp](#) (float amp)

### 5.31.1 Detailed Description

Randomly segmented line generator  
Originally extracted from csound by Paul Batchelor.  
Ported by Ben Sergentanis, June 2020

#### Author

Gabriel Maldonado

@year 1998

Location: Opcodes/uggab.c (csound)

### 5.31.2 Member Function Documentation

#### 5.31.2.1 Init()

```
void Jitter::Init (
    float sample_rate )
```

Initializes [Jitter](#) module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

#### 5.31.2.2 Process()

```
float Jitter::Process ( )
```

Get next floating point jitter sample

### 5.31.2.3 SetAmp()

```
void Jitter::SetAmp (
    float amp )
```

Set the amplitude of the jitter. Jitters fall from -amp to +amp

#### Parameters

<i>amp</i>	Jitter amplitude
------------	------------------

### 5.31.2.4 SetCpsMax()

```
void Jitter::SetCpsMax (
    float cps_max )
```

Set the maximum speed of the jitter engine.

#### Parameters

<i>cps_max</i>	Maximum number of jitters per second.
----------------	---------------------------------------

### 5.31.2.5 SetCpsMin()

```
void Jitter::SetCpsMin (
    float cps_min )
```

Set the minimum speed of the jitter engine.

#### Parameters

<i>cps_min</i>	Number of new jitters per second
----------------	----------------------------------

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

- Source/Utility/jitter.h
- Source/Utility/jitter.cpp

## 5.32 daisysp::Limiter Class Reference

```
#include <limiter.h>
```

## Public Member Functions

- void [Init](#) ()
- void [ProcessBlock](#) (float \*in, size\_t size, float pre\_gain)

### 5.32.1 Detailed Description

Simple Peak [Limiter](#)

This was extracted from pichenettes/stmlib.

Credit to pichenettes/Mutable Instruments

### 5.32.2 Member Function Documentation

#### 5.32.2.1 Init()

```
void daisysp::Limiter::Init ( )
```

Initializes the [Limiter](#) instance.

#### 5.32.2.2 ProcessBlock()

```
void daisysp::Limiter::ProcessBlock (
    float * in,
    size_t size,
    float pre_gain )
```

Processes a block of audio through the limiter.

#### Parameters

<i>in</i>	- pointer to a block of audio samples to be processed. The buffer is operated on directly.
<i>size</i>	- size of the buffer "in"
<i>pre_gain</i>	- amount of pre_gain applied to the signal.

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

- Source/Dynamics/limiter.h
- Source/Dynamics/limiter.cpp

## 5.33 daisysp::Line Class Reference

```
#include <line.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (uint8\_t \*finished)
- void [Start](#) (float start, float end, float dur)

### 5.33.1 Detailed Description

creates a [Line](#) segment signal

### 5.33.2 Member Function Documentation

#### 5.33.2.1 Init()

```
void Line::Init (
    float sample_rate )
```

Initializes [Line](#) module.

#### 5.33.2.2 Process()

```
float Line::Process (
    uint8_t * finished )
```

Processes [Line](#) segment. Returns one sample. value of finished will be updated to a 1, upon completion of the [Line](#)'s trajectory.

#### 5.33.2.3 Start()

```
void Line::Start (
    float start,
    float end,
    float dur )
```

Begin creation of [Line](#).

#### Parameters

<i>start</i>	- beginning value
<i>end</i>	- ending value
<i>dur</i>	- duration in seconds of <a href="#">Line</a> segment

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

- Source/Control/line.h

- [Source/Control/line.cpp](#)

## 5.34 daisysp::LinearVCA Class Reference

Linear type VCA.

```
#include <hihat.h>
```

### Public Member Functions

- float **operator()** (float s, float gain)

#### 5.34.1 Detailed Description

Linear type VCA.

##### Author

Ben Sergentanis

##### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/hihat.h to an independent module.  
Original code written by Emilie Gillet in 2016.

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

- [Source/Drums/hihat.h](#)

## 5.35 daisysp::Maytrig Class Reference

```
#include <maytrig.h>
```

### Public Member Functions

- float **Process** (float prob)

#### 5.35.1 Detailed Description

Probabilistic trigger module

Original author(s) : Paul Batchelor

Ported from soundpipe by Ben Sergentanis, May 2020



## 5.35.2 Member Function Documentation

### 5.35.2.1 Process()

```
float daisysp::Maytrig::Process (  
    float prob ) [inline]
```

probabilistically generates triggers

#### Parameters

<i>prob</i>	(1 always returns true, 0 always false)
-------------	---

#### Returns

given a probability 0 to 1, returns true or false.

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

- Source/Utility/maytrig.h

## 5.36 daisysp::Metro Class Reference

```
#include <metro.h>
```

### Public Member Functions

- void [Init](#) (float freq, float sample\_rate)
- uint8\_t [Process](#) ()
- void [Reset](#) ()
- void [SetFreq](#) (float freq)
- float [GetFreq](#) ()

#### 5.36.1 Detailed Description

Creates a clock signal at a specific frequency.

#### 5.36.2 Member Function Documentation

##### 5.36.2.1 GetFreq()

```
float daisysp::Metro::GetFreq ( ) [inline]
```

Returns current value for frequency.

### 5.36.2.2 Init()

```
void Metro::Init (
    float freq,
    float sample_rate )
```

Initializes [Metro](#) module. Arguments:

- freq: frequency at which new clock signals will be generated Input Range:
- sample\_rate: sample rate of audio engine Input range:

### 5.36.2.3 Process()

```
uint8_t Metro::Process ( )
```

checks current state of [Metro](#) object and updates state if necessary.

### 5.36.2.4 Reset()

```
void daisysp::Metro::Reset ( ) [inline]
```

resets phase to 0

### 5.36.2.5 SetFreq()

```
void Metro::SetFreq (
    float freq )
```

Sets frequency at which [Metro](#) module will run at.

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

- Source/Utility/metro.h
- Source/Utility/metro.cpp

## 5.37 daisysp::ModalVoice Class Reference

Simple modal synthesis voice with a mallet exciter: click -> LPF -> resonator.

```
#include <modalvoice.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (bool trigger=false)
- void [SetSustain](#) (bool sustain)
- void [Trig](#) ()
- void [SetFreq](#) (float freq)
- void [SetAccent](#) (float accent)
- void [SetStructure](#) (float structure)
- void [SetBrightness](#) (float brightness)
- void [SetDamping](#) (float damping)
- float [GetAux](#) ()

### 5.37.1 Detailed Description

Simple modal synthesis voice with a mallet exciter: click -> LPF -> resonator.

#### Author

Ben Sergentanis

#### Date

Jan 2021 The click can be replaced by continuous white noise.

Ported from pichenettes/eurorack/plaits/dsp/physical\_modelling/modal\_voice.h and pichenettes/eurorack/plaits/dsp/physical\_modelling/modal\_voice.cc to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.37.2 Member Function Documentation

#### 5.37.2.1 [GetAux\(\)](#)

```
float ModalVoice::GetAux ( )
```

Get the raw excitation signal. Must call [Process\(\)](#) first.

#### 5.37.2.2 [Init\(\)](#)

```
void ModalVoice::Init (
    float sample_rate )
```

Initialize the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

**5.37.2.3 Process()**

```
float ModalVoice::Process (
    bool trigger = false )
```

Get the next sample

## Parameters

<i>trigger</i>	Strike the resonator. Defaults to false.
----------------	--

**5.37.2.4 SetAccent()**

```
void ModalVoice::SetAccent (
    float accent )
```

Hit the resonator a bit harder.

## Parameters

<i>accent</i>	Works 0-1.
---------------	------------

**5.37.2.5 SetBrightness()**

```
void ModalVoice::SetBrightness (
    float brightness )
```

Set the brightness of the resonator, and the noise density.

## Parameters

<i>brightness</i>	Works best 0-1
-------------------	----------------

### 5.37.2.6 SetDamping()

```
void ModalVoice::SetDamping (
    float damping )
```

How long the resonant body takes to decay.

#### Parameters

<i>damping</i>	Works best 0-1
----------------	----------------

### 5.37.2.7 SetFreq()

```
void ModalVoice::SetFreq (
    float freq )
```

Set the resonator root frequency.

#### Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

### 5.37.2.8 SetStructure()

```
void ModalVoice::SetStructure (
    float structure )
```

Changes the general character of the resonator (stiffness, brightness)

#### Parameters

<i>structure</i>	Works best from 0-1
------------------	---------------------

### 5.37.2.9 SetSustain()

```
void ModalVoice::SetSustain (
    bool sustain )
```

Continually excite the resonator with noise.

#### Parameters

<i>sustain</i>	True turns on the noise.
----------------	--------------------------

#### 5.37.2.10 Trig()

```
void ModalVoice::Trig ( )
```

Strike the resonator.

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

- Source/PhysicalModeling/[modalvoice.h](#)
- Source/PhysicalModeling/[modalvoice.cpp](#)

## 5.38 daisysp::Mode Class Reference

```
#include <mode.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [Clear](#) ()
- void [SetFreq](#) (float freq)
- void [SetQ](#) (float q)

### 5.38.1 Detailed Description

Resonant Modal Filter

Extracted from soundpipe to work as a Daisy Module,  
originally extracted from csound by Paul Batchelor.

Original Author(s): Francois Blanc, Steven Yi

Year: 2001

Location: Opcodes/biquad.c (csound)

### 5.38.2 Member Function Documentation

### 5.38.2.1 Clear()

```
void Mode::Clear ( )
```

Clears the filter, returning the output to 0.0

### 5.38.2.2 Init()

```
void Mode::Init (
    float sample_rate )
```

Initializes the instance of the module. `sample_rate`: frequency of the audio engine in Hz

### 5.38.2.3 Process()

```
float Mode::Process (
    float in )
```

Processes one input sample through the filter, and returns the output.

### 5.38.2.4 SetFreq()

```
void daisysp::Mode::SetFreq (
    float freq ) [inline]
```

Sets the resonant frequency of the modal filter. Range: Any frequency such that  $\text{sample\_rate} / \text{freq} < \text{PI}$  (about 15.2kHz at 48kHz)

### 5.38.2.5 SetQ()

```
void daisysp::Mode::SetQ (
    float q ) [inline]
```

Sets the quality factor of the filter. Range: Positive Numbers (Good values range from 70 to 1400)

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

- Source/Filters/mode.h
- Source/Filters/mode.cpp

## 5.39 daisysp::MoogLadder Class Reference

```
#include <moogladder.h>
```



## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)
- void [SetFreq](#) (float freq)
- void [SetRes](#) (float res)

### 5.39.1 Detailed Description

Moog ladder filter module

Ported from soundpipe

Original author(s) : Victor Lazzarini, John ffitch (fast tanh), Bob Moog

### 5.39.2 Member Function Documentation

#### 5.39.2.1 Init()

```
void MoogLadder::Init (  
    float sample_rate )
```

Initializes the [MoogLadder](#) module. sample\_rate - The sample rate of the audio engine being run.

#### 5.39.2.2 Process()

```
float MoogLadder::Process (  
    float in )
```

Processes the lowpass filter

#### 5.39.2.3 SetFreq()

```
void daisysp::MoogLadder::SetFreq (  
    float freq ) [inline]
```

Sets the cutoff frequency or half-way point of the filter. Arguments

- freq - frequency value in Hz. Range: Any positive value.

### 5.39.2.4 SetRes()

```
void daisysp::MoogLadder::SetRes (
    float res ) [inline]
```

Sets the resonance of the filter.

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

- Source/Filters/moogladder.h
- Source/Filters/moogladder.cpp

## 5.40 daisysp::NIFilt Class Reference

```
#include <nlfilt.h>
```

### Public Member Functions

- void [Init](#) ()
- void [ProcessBlock](#) (float \*in, float \*out, size\_t size)
- void [SetCoefficients](#) (float a, float b, float d, float C, float L)
- void [SetA](#) (float a)
- void [SetB](#) (float b)
- void [SetD](#) (float d)
- void [SetC](#) (float C)
- void [SetL](#) (float L)

### 5.40.1 Detailed Description

Non-linear filter

port by: Stephen Hensley, December 2019

The four 5-coefficients: a, b, d, C, and L are used to configure different filter types.

Structure for Dobson/Fitch nonlinear filter

Revised Formula from Risto Holopainen 12 Mar 2004

$$Y\{n\} = \tanh(a Y\{n-1\} + b Y\{n-2\} + d Y^2\{n-L\} + X\{n\} - C)$$

Though traditional filter types can be made, the effect will always respond differently to different input.

This Source is a heavily modified version of the original source from Csound.

**Todo** make this work on a single sample instead of just on blocks at a time.

## 5.40.2 Member Function Documentation

### 5.40.2.1 Init()

```
void NlFilt::Init ( )
```

Initializes the [NIFilt](#) object.

### 5.40.2.2 ProcessBlock()

```
void NlFilt::ProcessBlock (
    float * in,
    float * out,
    size_t size )
```

Process the array pointed to by \*in and updates the output to \*out; This works on a block of audio at once, the size of which is set with the size.

### 5.40.2.3 SetA()

```
void daisysp::NlFilt::SetA (
    float a ) [inline]
```

Set Coefficient a

### 5.40.2.4 SetB()

```
void daisysp::NlFilt::SetB (
    float b ) [inline]
```

Set Coefficient b

### 5.40.2.5 SetC()

```
void daisysp::NlFilt::SetC (
    float C ) [inline]
```

Set Coefficient C

### 5.40.2.6 SetCoefficients()

```
void daisysp::NlFilt::SetCoefficients (
    float a,
    float b,
    float d,
    float C,
    float L ) [inline]
```

inputs these are the five coefficients for the filter.

### 5.40.2.7 SetD()

```
void daisysp::NlFilt::SetD (  
    float d ) [inline]
```

Set Coefficient d

### 5.40.2.8 SetL()

```
void daisysp::NlFilt::SetL (  
    float L ) [inline]
```

Set Coefficient L

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

- Source/Filters/nfilt.h
- Source/Filters/nfilt.cpp

## 5.41 daisysp::Oscillator Class Reference

```
#include <oscillator.h>
```

### Public Types

- enum {  
    WAVE\_SIN , WAVE\_TRI , WAVE\_SAW , WAVE\_RAMP ,  
    WAVE\_SQUARE , WAVE\_POLYBLEP\_TRI , WAVE\_POLYBLEP\_SAW , WAVE\_POLYBLEP\_SQUARE ,  
    WAVE\_LAST }

### Public Member Functions

- void [Init](#) (float sample\_rate)
- void [SetFreq](#) (const float f)
- void [SetAmp](#) (const float a)
- void [SetWaveform](#) (const uint8\_t wf)
- bool [IsEOR](#) ()
- bool [IsEOC](#) ()
- bool [IsRising](#) ()
- bool [IsFalling](#) ()
- float [Process](#) ()
- void [PhaseAdd](#) (float \_phase)
- void [Reset](#) (float \_phase=0.0f)

#### 5.41.1 Detailed Description

Synthesis of several waveforms, including polyBLEP bandlimited waveforms.

## 5.41.2 Member Enumeration Documentation

### 5.41.2.1 anonymous enum

anonymous enum

Choices for output waveforms, POLYBLEP are appropriately labeled. Others are naive forms.

## 5.41.3 Member Function Documentation

### 5.41.3.1 Init()

```
void daisysp::Oscillator::Init (
    float sample_rate ) [inline]
```

Initializes the [Oscillator](#)

#### Parameters

<i>sample_rate</i>	- sample rate of the audio engine being run, and the frequency that the Process function will be called.
--------------------	--

Defaults:

- `freq_ = 100 Hz`
- `amp_ = 0.5`
- `waveform_ = sine wave.`

### 5.41.3.2 IsEOC()

```
bool daisysp::Oscillator::IsEOC ( ) [inline]
```

Returns true if cycle is at end of cycle. Set during call to Process.

### 5.41.3.3 IsEOR()

```
bool daisysp::Oscillator::IsEOR ( ) [inline]
```

Returns true if cycle is at end of rise. Set during call to Process.

#### 5.41.3.4 IsFalling()

```
bool daisysp::Oscillator::IsFalling ( ) [inline]
```

Returns true if cycle falling.

#### 5.41.3.5 IsRising()

```
bool daisysp::Oscillator::IsRising ( ) [inline]
```

Returns true if cycle rising.

#### 5.41.3.6 PhaseAdd()

```
void daisysp::Oscillator::PhaseAdd (
    float _phase ) [inline]
```

Adds a value 0.0-1.0 (mapped to 0.0-TWO\_PI) to the current phase. Useful for PM and "FM" synthesis.

#### 5.41.3.7 Process()

```
float Oscillator::Process ( )
```

Processes the waveform to be generated, returning one sample. This should be called once per sample period.

#### 5.41.3.8 Reset()

```
void daisysp::Oscillator::Reset (
    float _phase = 0.0f ) [inline]
```

Resets the phase to the input argument. If no argument is present, it will reset phase to 0.0;

#### 5.41.3.9 SetAmp()

```
void daisysp::Oscillator::SetAmp (
    const float a ) [inline]
```

Sets the amplitude of the waveform.

#### 5.41.3.10 SetFreq()

```
void daisysp::Oscillator::SetFreq (
    const float f ) [inline]
```

Changes the frequency of the [Oscillator](#), and recalculates phase increment.

### 5.41.3.11 SetWaveform()

```
void daisysp::Oscillator::SetWaveform (
    const uint8_t wf ) [inline]
```

Sets the waveform to be synthesized by the [Process\(\)](#) function.

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

- Source/Synthesis/oscillator.h
- Source/Synthesis/oscillator.cpp

## 5.42 daisysp::OscillatorBank Class Reference

[Oscillator](#) Bank module.

```
#include <oscillatorbank.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetAmplitudes](#) (const float \*amplitudes)
- void [SetSingleAmp](#) (float amp, int idx)
- void [SetGain](#) (float gain)

### 5.42.1 Detailed Description

[Oscillator](#) Bank module.

#### Author

Ben Sergentanis

#### Date

Dec 2020 A mixture of 7 sawtooth and square waveforms in the style of divide-down organs

Ported from pichenettes/eurorack/plaits/dsp/oscillator/string\_synth\_oscillator.h

to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.42.2 Member Function Documentation

#### 5.42.2.1 Init()

```
void OscillatorBank::Init (
    float sample_rate )
```

Init string synth module

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

**5.42.2.2 Process()**

```
float OscillatorBank::Process ( )
```

Get next floating point sample

**5.42.2.3 SetAmplitudes()**

```
void OscillatorBank::SetAmplitudes (
    const float * amplitudes )
```

Set amplitudes of 7 oscillators. 0-6 are Saw 8', Square 8', Saw 4', Square 4', Saw 2', Square 2', Saw 1'

## Parameters

<i>amplitudes</i>	array of 7 floating point amplitudes. Must sum to 1.
-------------------	--

**5.42.2.4 SetFreq()**

```
void OscillatorBank::SetFreq (
    float freq )
```

Set oscillator frequency (8' oscillator)

## Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

**5.42.2.5 SetGain()**

```
void OscillatorBank::SetGain (
    float gain )
```

Set overall gain.



## Parameters

<i>gain</i>	Gain to set. 0-1.
-------------	-------------------

**5.42.2.6 SetSingleAmp()**

```
void OscillatorBank::SetSingleAmp (
    float amp,
    int idx )
```

Set a single amplitude

## Parameters

<i>amp</i>	Amplitude to set.
<i>idx</i>	Which wave's amp to set

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

- Source/Synthesis/[oscillatorbank.h](#)
- Source/Synthesis/[oscillatorbank.cpp](#)

**5.43 daisysp::Overdrive Class Reference**

Distortion / [Overdrive](#) Module.

```
#include <overdrive.h>
```

**Public Member Functions**

- void [Init](#) ()
- float [Process](#) (float in)
- void [SetDrive](#) (float drive)

**5.43.1 Detailed Description**

Distortion / [Overdrive](#) Module.

**Author**

Ported by Ben Sergentanis

**Date**

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/fx/overdrive.h to an independent module.

Original code written by Emilie Gillet in 2014.

## 5.43.2 Member Function Documentation

### 5.43.2.1 Init()

```
void daisysp::Overdrive::Init ( )
```

Initializes the module with 0 gain

### 5.43.2.2 Process()

```
float daisysp::Overdrive::Process (
    float in )
```

Get the next sample

#### Parameters

<i>in</i>	Input to be overdriven
-----------	------------------------

### 5.43.2.3 SetDrive()

```
void daisysp::Overdrive::SetDrive (
    float drive )
```

Set the amount of drive

#### Parameters

<i>drive</i>	Works from 0-1
--------------	----------------

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

- [Source/Effects/overdrive.h](#)
- [Source/Effects/overdrive.cpp](#)

## 5.44 daisysp::Particle Class Reference

Random impulse train processed by a resonant filter.

```
#include <particle.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- float [GetNoise](#) ()
- void [SetFreq](#) (float frequency)
- void [SetResonance](#) (float resonance)
- void [SetRandomFreq](#) (float freq)
- void [SetDensity](#) (float density)
- void [SetGain](#) (float gain)
- void [SetSpread](#) (float spread)
- void [SetSync](#) (bool sync)

### 5.44.1 Detailed Description

Random impulse train processed by a resonant filter.

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 Noise processed by a sample and hold running at a target frequency.

Ported from pichenettes/eurorack/plaits/dsp/noise/particle.h to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.44.2 Member Function Documentation

#### 5.44.2.1 [GetNoise\(\)](#)

```
float Particle::GetNoise ( )
```

Get the raw noise output. Must call [Process\(\)](#) first.

#### 5.44.2.2 [Init\(\)](#)

```
void Particle::Init (
    float sample_rate )
```

Initialize the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

**5.44.2.3 Process()**

```
float Particle::Process ( )
```

Get the next sample

**5.44.2.4 SetDensity()**

```
void Particle::SetDensity (
    float density )
```

Noise density

## Parameters

<i>Works</i>	0-1.
--------------	------

**5.44.2.5 SetFreq()**

```
void Particle::SetFreq (
    float frequency )
```

Set the resonant filter frequency

## Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

**5.44.2.6 SetGain()**

```
void Particle::SetGain (
    float gain )
```

Overall module gain

## Parameters

<i>Works</i>	0-1.
--------------	------

**5.44.2.7 SetRandomFreq()**

```
void Particle::SetRandomFreq (
    float freq )
```

How often to randomize filter frequency

## Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

**5.44.2.8 SetResonance()**

```
void Particle::SetResonance (
    float resonance )
```

Set the filter resonance

## Parameters

<i>resonance</i>	Works 0-1
------------------	-----------

**5.44.2.9 SetSpread()**

```
void Particle::SetSpread (
    float spread )
```

How much to randomize the set filter frequency.

## Parameters

<i>spread</i>	Works over positive numbers.
---------------	------------------------------

### 5.44.2.10 SetSync()

```
void Particle::SetSync (
    bool sync )
```

Force randomize the frequency.

#### Parameters

<i>sync</i>	True to randomize freq.
-------------	-------------------------

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

- Source/Noise/[particle.h](#)
- Source/Noise/[particle.cpp](#)

## 5.45 daisysp::Phasor Class Reference

```
#include <phasor.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate, float freq, float initial\_phase)
- void [Init](#) (float sample\_rate, float freq)
- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- float [GetFreq](#) ()

### 5.45.1 Detailed Description

Generates a normalized signal moving from 0-1 at the specified frequency.

**Todo** Selecting which channels should be initialized/included in the sequence conversion.

Setup a similar start function for an external mux, but that seems outside the scope of this file.

### 5.45.2 Member Function Documentation

#### 5.45.2.1 GetFreq()

```
float daisysp::Phasor::GetFreq ( ) [inline]
```

Returns current frequency value in Hz

### 5.45.2.2 Init() [1/3]

```
void daisysp::Phasor::Init (
    float sample_rate ) [inline]
```

Initialize phasor with samplerate

### 5.45.2.3 Init() [2/3]

```
void daisysp::Phasor::Init (
    float sample_rate,
    float freq ) [inline]
```

Initialize phasor with samplerate and freq

### 5.45.2.4 Init() [3/3]

```
void daisysp::Phasor::Init (
    float sample_rate,
    float freq,
    float initial_phase ) [inline]
```

Initializes the [Phasor](#) module sample rate, and freq are in Hz initial phase is in radians Additional Init functions have defaults when arg is not specified:

- `phs = 0.0f`
- `freq = 1.0f`

### 5.45.2.5 Process()

```
float Phasor::Process ( )
```

processes [Phasor](#) and returns current value

### 5.45.2.6 SetFreq()

```
void Phasor::SetFreq (
    float freq )
```

Sets frequency of the [Phasor](#) in Hz

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

- Source/Control/phasor.h
- Source/Control/phasor.cpp

## 5.46 daisysp::PitchShifter Class Reference

```
#include <pitchshifter.h>
```

### Public Member Functions

- void [Init](#) (float sr)
- float [Process](#) (float &in)
- void [SetTransposition](#) (const float &transpose)
- void [SetDelSize](#) (uint32\_t size)
- void [SetFun](#) (float f)

### 5.46.1 Detailed Description

time-domain pitchshifter

Author: shensley

Based on "Pitch Shifting" from ucsd.edu

$$t = 1 - ((s * f) / R)$$

where: s is the size of the delay f is the frequency of the lfo r is the sample\_rate

solving for t = 12.0 f = (12 - 1) \* 48000 / SHIFT\_BUFFER\_SIZE;

**Todo** • move hash\_xs32 and myrand to [dsp.h](#) and give appropriate names

### 5.46.2 Member Function Documentation

#### 5.46.2.1 Init()

```
void daisysp::PitchShifter::Init (
    float sr ) [inline]
```

Initialize pitch shifter

#### 5.46.2.2 Process()

```
float daisysp::PitchShifter::Process (
    float & in ) [inline]
```

process pitch shifter



### 5.46.2.3 SetDelSize()

```
void daisysp::PitchShifter::SetDelSize (
    uint32_t size ) [inline]
```

sets delay size changing the timbre of the pitchshifting

### 5.46.2.4 SetFun()

```
void daisysp::PitchShifter::SetFun (
    float f ) [inline]
```

sets an amount of internal random modulation, kind of sounds like tape-flutter

### 5.46.2.5 SetTransposition()

```
void daisysp::PitchShifter::SetTransposition (
    const float & transpose ) [inline]
```

sets transposition in semitones

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

- Source/Effects/pitchshifter.h

## 5.47 daisysp::Pluck Class Reference

```
#include <pluck.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate, float \*buf, int32\_t npt, int32\_t mode)
- float [Process](#) (float &trig)
- void [SetAmp](#) (float amp)
- void [SetFreq](#) (float freq)
- void [SetDecay](#) (float decay)
- void [SetDamp](#) (float damp)
- void [SetMode](#) (int32\_t mode)
- float [GetAmp](#) ()
- float [GetFreq](#) ()
- float [GetDecay](#) ()
- float [GetDamp](#) ()
- int32\_t [GetMode](#) ()

### 5.47.1 Detailed Description

Produces a naturally decaying plucked string or drum sound based on the Karplus-Strong algorithms.

Ported from soundpipe to DaisySP

This code was originally extracted from the Csound opcode "pluck"

Original Author(s): Barry Vercoe, John fitch Year: 1991

Location: OOps/ugens4.c

### 5.47.2 Member Function Documentation

#### 5.47.2.1 GetAmp()

```
float daisysp::Pluck::GetAmp ( ) [inline]
```

Returns the current value for amp.

#### 5.47.2.2 GetDamp()

```
float daisysp::Pluck::GetDamp ( ) [inline]
```

Returns the current value for damp.

#### 5.47.2.3 GetDecay()

```
float daisysp::Pluck::GetDecay ( ) [inline]
```

Returns the current value for decay.

#### 5.47.2.4 GetFreq()

```
float daisysp::Pluck::GetFreq ( ) [inline]
```

Returns the current value for freq.

#### 5.47.2.5 GetMode()

```
int32_t daisysp::Pluck::GetMode ( ) [inline]
```

Returns the current value for mode.

### 5.47.2.6 Init()

```
void Pluck::Init (
    float sample_rate,
    float * buf,
    int32_t npt,
    int32_t mode )
```

Initializes the [Pluck](#) module.

```
\param sample_rate: Sample rate of the audio engine being run.
\param buf: buffer used as an impulse when triggering the Pluck algorithm
\param npt: number of elementes in buf.
\param mode: Sets the mode of the algorithm.
```

### 5.47.2.7 Process()

```
float Pluck::Process (
    float & trig )
```

Processes the waveform to be generated, returning one sample. This should be called once per sample period.

### 5.47.2.8 SetAmp()

```
void daisysp::Pluck::SetAmp (
    float amp ) [inline]
```

Sets the amplitude of the output signal. Input range: 0-1?

### 5.47.2.9 SetDamp()

```
void daisysp::Pluck::SetDamp (
    float damp ) [inline]
```

Sets the dampening factor applied by the filter (based on `PLUCK_MODE`) Input range: 0-1

### 5.47.2.10 SetDecay()

```
void daisysp::Pluck::SetDecay (
    float decay ) [inline]
```

Sets the time it takes for a triggered note to end in seconds. Input range: 0-1

### 5.47.2.11 SetFreq()

```
void daisysp::Pluck::SetFreq (
    float freq ) [inline]
```

Sets the frequency of the output signal in Hz. Input range: Any positive value

### 5.47.2.12 SetMode()

```
void daisysp::Pluck::SetMode (
    int32_t mode ) [inline]
```

Sets the mode of the algorithm.

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

- Source/PhysicalModeling/pluck.h
- Source/PhysicalModeling/pluck.cpp

## 5.48 daisysp::PolyPluck< num\_voices > Class Template Reference

```
#include <PolyPluck.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float &trig, float note)
- void [SetDecay](#) (float p)

### 5.48.1 Detailed Description

```
template<size_t num_voices>
class daisysp::PolyPluck< num_voices >
```

Simplified Pseudo-Polyphonic [Pluck](#) Voice

Template Based [Pluck](#) Voice, with configurable number of voices and simple pseudo-polyphony.

DC Blocking included to prevent biases from causing unwanted saturation distortion.

Author\*\*: shensley

Date Added\*\*: March 2020

### 5.48.2 Member Function Documentation

#### 5.48.2.1 Init()

```
template<size_t num_voices>
void daisysp::PolyPluck< num_voices >::Init (
    float sample_rate ) [inline]
```

Initializes the [PolyPluck](#) instance.

## Parameters

<i>sample_rate</i>	rate in Hz that the <a href="#">Process()</a> function will be called.
--------------------	--

**5.48.2.2 Process()**

```
template<size_t num_voices>
float daisysp::PolyPluck< num_voices >::Process (
    float & trig,
    float note ) [inline]
```

Process function, synthesizes and sums the output of all voices, triggering a new voice with frequency of MIDI note number when trig > 0.

## Parameters

<i>trig</i>	value by reference of trig. When trig > 0 a the next voice will be triggered, and trig will be set to 0.
<i>note</i>	MIDI note number for the active_voice.

**5.48.2.3 SetDecay()**

```
template<size_t num_voices>
void daisysp::PolyPluck< num_voices >::SetDecay (
    float p ) [inline]
```

Sets the decay coefficients of the pluck voices.

## Parameters

<i>p</i>	expects 0.0-1.0 input.
----------	------------------------

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

- Source/PhysicalModeling/PolyPluck.h

**5.49 daisysp::Port Class Reference**

```
#include <port.h>
```

**Public Member Functions**

- void [Init](#) (float sample\_rate, float htime)
- float [Process](#) (float in)
- void [SetHtime](#) (float htime)
- float [GetHtime](#) ()

### 5.49.1 Detailed Description

Applies portamento to an input signal.

At each new step value, the input is low-pass filtered to move towards that value at a rate determined by `ihitim`. `ihitim` is the half-time of the function (in seconds), during which the curve will traverse half the distance towards the new value, then half as much again, etc., theoretically never reaching its asymptote.

This code has been ported from Soundpipe to DaisySP by Paul Batchelor.

The Soundpipe module was extracted from the Csound opcode "portk".

Original Author(s): Robbin Whittle, John ffitch

Year: 1995, 1998

Location: Opcodes/biquad.c

### 5.49.2 Member Function Documentation

#### 5.49.2.1 GetHtime()

```
float daisysp::Port::GetHtime ( ) [inline]
```

returns current value of `htime`

#### 5.49.2.2 Init()

```
void Port::Init (
    float sample_rate,
    float htime )
```

Initializes `Port` module

##### Parameters

<i>sample_rate</i>	sample rate of audio engine
<i>htime</i>	half-time of the function, in seconds.

#### 5.49.2.3 Process()

```
float Port::Process (
    float in )
```

Applies portamento to input signal and returns processed signal.

**Returns**

slewed output signal

**5.49.2.4 SetHtime()**

```
void daisysp::Port::SetHtime (
    float htime ) [inline]
```

Sets htime

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

- Source/Utility/port.h
- Source/Utility/port.cpp

## 5.50 daisysp::Resonator Class Reference

Resonant Body Simulation.

```
#include <resonator.h>
```

### Public Member Functions

- void [Init](#) (float position, int resolution, float sample\_rate)
- float [Process](#) (const float in)
- void [SetFreq](#) (float freq)
- void [SetStructure](#) (float structure)
- void [SetBrightness](#) (float brightness)
- void [SetDamping](#) (float damping)

### 5.50.1 Detailed Description

Resonant Body Simulation.

**Author**

Ported by Ben Sergentanis

**Date**

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/physical\_modelling/resonator.h to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.50.2 Member Function Documentation

### 5.50.2.1 Init()

```
void Resonator::Init (  
    float position,  
    int resolution,  
    float sample_rate )
```

Initialize the module



## Parameters

<i>position</i>	Offset the phase of the amplitudes. 0-1
<i>resolution</i>	Quality vs speed scalar
<i>sample_rate</i>	Samplerate of the audio engine being run.

**5.50.2.2 Process()**

```
float Resonator::Process (  
    const float in )
```

Get the next *sample\_rate*

## Parameters

<i>in</i>	The signal to excited the resonant body
-----------	---

**5.50.2.3 SetBrightness()**

```
void Resonator::SetBrightness (  
    float brightness )
```

Set the brightness of the resonator

## Parameters

<i>brightness</i>	Works best 0-1
-------------------	----------------

**5.50.2.4 SetDamping()**

```
void Resonator::SetDamping (  
    float damping )
```

How long the resonant body takes to decay.

## Parameters

<i>damping</i>	Works best 0-1
----------------	----------------

### 5.50.2.5 SetFreq()

```
void Resonator::SetFreq (
    float freq )
```

[Resonator](#) frequency.

#### Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

### 5.50.2.6 SetStructure()

```
void Resonator::SetStructure (
    float structure )
```

Changes the general charater of the resonator (stiffness, brightness)

#### Parameters

<i>structure</i>	Works best from 0-1
------------------	---------------------

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

- Source/PhysicalModeling/[resonator.h](#)
- Source/PhysicalModeling/resonator.cpp

## 5.51 daisysp::ResonatorSvf< batch\_size > Class Template Reference

SVF for use in the [Resonator](#) Class

.

```
#include <resonator.h>
```

### Public Types

- enum **FilterMode** { **LOW\_PASS** , **BAND\_PASS** , **BAND\_PASS\_NORMALIZED** , **HIGH\_PASS** }

### Public Member Functions

- void **Init** ()
- template<FilterMode mode, bool add>  
void **Process** (const float \*f, const float \*q, const float \*gain, const float in, float \*out)

### 5.51.1 Detailed Description

```
template<int batch_size>
class daisysp::ResonatorSvf< batch_size >
```

SVF for use in the [Resonator](#) Class

.

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/physical\_modelling/resonator.h to an independent module.

Original code written by Emilie Gillet in 2016.

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

- [Source/PhysicalModeling/resonator.h](#)

## 5.52 daisysp::ReverbSc Class Reference

```
#include <reverbsc.h>
```

### Public Member Functions

- int [Init](#) (float sample\_rate)
- int [Process](#) (const float &in1, const float &in2, float \*out1, float \*out2)
- void [SetFeedback](#) (const float &fb)
- void [SetLpFreq](#) (const float &freq)

### 5.52.1 Detailed Description

Stereo Reverb

Reverb SC: Ported from csound/soundpipe

Original author(s): Sean Costello, Istvan Varga

Year: 1999, 2005

Ported to soundpipe by: Paul Batchelor

Ported by: Stephen Hensley

## 5.52.2 Member Function Documentation

### 5.52.2.1 Init()

```
int ReverbSc::Init (
    float sample_rate )
```

Initializes the reverb module, and sets the `sample_rate` at which the `Process` function will be called. Returns 0 if all good, or 1 if it runs out of delay times exceed maximum allowed.

### 5.52.2.2 Process()

```
int ReverbSc::Process (
    const float & in1,
    const float & in2,
    float * out1,
    float * out2 )
```

Process the input through the reverb, and updates values of `out1`, and `out2` with the new processed signal.

### 5.52.2.3 SetFeedback()

```
void daisysp::ReverbSc::SetFeedback (
    const float & fb ) [inline]
```

controls the reverb time. reverb tail becomes infinite when set to 1.0

#### Parameters

<i>fb</i>	- sets reverb time. range: 0.0 to 1.0
-----------	---------------------------------------

### 5.52.2.4 SetLpFreq()

```
void daisysp::ReverbSc::SetLpFreq (
    const float & freq ) [inline]
```

controls the internal dampening filter's cutoff frequency.

#### Parameters

<i>freq</i>	- low pass frequency. range: 0.0 to <code>sample_rate / 2</code>
-------------	--

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

- Source/Effects/reverb.h
- Source/Effects/reverb.cpp

## 5.53 daisysp::ReverbScDI Struct Reference

```
#include <reverb.h>
```

### Public Attributes

- int [write\\_pos](#)
- int [buffer\\_size](#)
- int [read\\_pos](#)
- int [read\\_pos\\_frac](#)
- int [read\\_pos\\_frac\\_inc](#)
- int [dummy](#)
- int [seed\\_val](#)
- int [rand\\_line\\_cnt](#)
- float [filter\\_state](#)
- float \* [buf](#)

### 5.53.1 Detailed Description

Delay line for internal reverb use

### 5.53.2 Member Data Documentation

#### 5.53.2.1 buf

```
float* daisysp::ReverbScDI::buf
```

buffer ptr

#### 5.53.2.2 buffer\_size

```
int daisysp::ReverbScDI::buffer_size
```

buffer size

#### 5.53.2.3 dummy

```
int daisysp::ReverbScDI::dummy
```

dummy var

#### 5.53.2.4 filter\_state

```
float daisysp::ReverbScDl::filter_state
```

state of filter

#### 5.53.2.5 rand\_line\_cnt

```
int daisysp::ReverbScDl::rand_line_cnt
```

number of random lines

#### 5.53.2.6 read\_pos

```
int daisysp::ReverbScDl::read_pos
```

read position

#### 5.53.2.7 read\_pos\_frac

```
int daisysp::ReverbScDl::read_pos_frac
```

fractional component of read pos

#### 5.53.2.8 read\_pos\_frac\_inc

```
int daisysp::ReverbScDl::read_pos_frac_inc
```

increment for fractional

#### 5.53.2.9 seed\_val

```
int daisysp::ReverbScDl::seed_val
```

randseed

#### 5.53.2.10 write\_pos

```
int daisysp::ReverbScDl::write_pos
```

write position

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

- Source/Effects/reverbSc.h

## 5.54 daisysp::RingModNoise Class Reference

Ring mod style metallic noise generator.

```
#include <hihat.h>
```

### Public Member Functions

- void **Init** (float sample\_rate)
- float **Process** (float f0)

#### 5.54.1 Detailed Description

Ring mod style metallic noise generator.

##### Author

Ben Sergentanis

##### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/hihat.h to an independent module.  
Original code written by Emilie Gillet in 2016.

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

- Source/Drums/[hihat.h](#)
- Source/Drums/hihat.cpp

## 5.55 daisysp::SampleHold Class Reference

```
#include <samplehold.h>
```

### Public Types

- enum **Mode** { **MODE\_SAMPLE\_HOLD** , **MODE\_TRACK\_HOLD** , **MODE\_LAST** }

### Public Member Functions

- float **Process** (bool trigger, float input, Mode mode=MODE\_SAMPLE\_HOLD)

### 5.55.1 Detailed Description

Dual track and hold / Sample and hold module.  
Ported from soundpipe by Ben Sergentanis, June 2020.

#### Author

Paul Batchelor

#### Date

2015

### 5.55.2 Member Function Documentation

#### 5.55.2.1 Process()

```
float daisysp::SampleHold::Process (
    bool trigger,
    float input,
    Mode mode = MODE_SAMPLE_HOLD ) [inline]
```

Process the next sample. Both sample and track and hold are run in parallel

#### Parameters

<i>trigger</i>	Trigger the sample/track and hold
<i>input</i>	Signal to be sampled/tracked and held
<i>mode</i>	Whether to output the tracked or sampled values.

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

- Source/Utility/samplehold.h

## 5.56 daisysp::SampleRateReducer Class Reference

Sample rate reducer.

```
#include <sampleratereducer.h>
```

### Public Member Functions

- void [Init](#) ()
- float [Process](#) (float in)
- void [SetFreq](#) (float frequency)



## 5.56.1 Detailed Description

Sample rate reducer.

### Author

Ben Sergentanis

### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/fx/sample\_rate\_reducer.h to an independent module.

Original code written by Emilie Gillet in 2014.

## 5.56.2 Member Function Documentation

### 5.56.2.1 Init()

```
void SampleRateReducer::Init ( )
```

Initialize the module

### 5.56.2.2 Process()

```
float SampleRateReducer::Process (
    float in )
```

Get the next floating point sample

#### Parameters

<i>in</i>	Sample to be processed.
-----------	-------------------------

### 5.56.2.3 SetFreq()

```
void SampleRateReducer::SetFreq (
    float frequency )
```

Set the new sample rate.

### Parameters

<i>Works</i>	over 0-1. 1 is full quality, .5 is half sample rate, etc.
--------------	---

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

- Source/Effects/[sampleratereducer.h](#)
- Source/Effects/[sampleratereducer.cpp](#)

## 5.57 daisysp::SmoothRandomGenerator Class Reference

Smooth random generator for internal modulation.

```
#include <smooth_random.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)

#### 5.57.1 Detailed Description

Smooth random generator for internal modulation.

#### Author

Ported by Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/noise/smooth\_random\_generator.h to an independent module.  
Original code written by Emilie Gillet in 2016.

#### 5.57.2 Member Function Documentation

##### 5.57.2.1 Init()

```
void daisysp::SmoothRandomGenerator::Init (  
    float sample_rate ) [inline]
```

Initialize the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

### 5.57.2.2 Process()

```
float daisysp::SmoothRandomGenerator::Process ( ) [inline]
```

Get the next float. Ranges from -1 to 1.

### 5.57.2.3 SetFreq()

```
void daisysp::SmoothRandomGenerator::SetFreq (
    float freq ) [inline]
```

How often to slew to a new random value

## Parameters

<i>freq</i>	Rate in Hz
-------------	------------

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

- [Source/Utility/smooth\\_random.h](#)

## 5.58 daisysp::SquareNoise Class Reference

808 style "metallic noise" with 6 square oscillators.

```
#include <hihat.h>
```

### Public Member Functions

- void **Init** (float sample\_rate)
- float **Process** (float f0)

### 5.58.1 Detailed Description

808 style "metallic noise" with 6 square oscillators.

## Author

Ben Sergentanis

**Date**

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/hihat.h to an independent module.  
Original code written by Emilie Gillet in 2016.

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

- Source/Drums/hihat.h
- Source/Drums/hihat.cpp

## 5.59 daisysp::String Class Reference

[Comb](#) filter / KS string.

```
#include <KarplusString.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- void [Reset](#) ()
- float [Process](#) (const float in)
- void [SetFreq](#) (float freq)
- void [SetNonLinearity](#) (float non\_linearity\_amount)
- void [SetBrightness](#) (float brightness)
- void [SetDamping](#) (float damping)

### 5.59.1 Detailed Description

[Comb](#) filter / KS string.

**Author**

Ben Sergentanis

**Date**

Jan 2021 "Lite" version of the implementation used in Rings

Ported from pichenettes/eurorack/plaits/dsp/oscillator/formant\_oscillator.h to an independent module.  
Original code written by Emilie Gillet in 2016.

### 5.59.2 Member Function Documentation

#### 5.59.2.1 Init()

```
void String::Init (  
    float sample_rate )
```

Initialize the module.

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

**5.59.2.2 Process()**

```
float String::Process (
    const float in )
```

Get the next floating point sample

## Parameters

<i>in</i>	Signal to excite the string.
-----------	------------------------------

**5.59.2.3 Reset()**

```
void String::Reset ( )
```

Clear the delay line

**5.59.2.4 SetBrightness()**

```
void String::SetBrightness (
    float brightness )
```

Set the string's overall brightness

## Parameters

<i>Works</i>	0-1.
--------------	------

**5.59.2.5 SetDamping()**

```
void String::SetDamping (
    float damping )
```

Set the string's decay time.

## Parameters

<i>damping</i>	Works 0-1.
----------------	------------

### 5.59.2.6 SetFreq()

```
void String::SetFreq (
    float freq )
```

Set the string frequency.

## Parameters

<i>freq</i>	Frequency in Hz
-------------	-----------------

### 5.59.2.7 SetNonLinearity()

```
void String::SetNonLinearity (
    float non_linearity_amount )
```

Set the string's behavior.

## Parameters

<i>-1</i>	to 0 is curved bridge, 0 to 1 is dispersion.
-----------	--

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

- [Source/PhysicalModeling/KarplusString.h](#)
- [Source/PhysicalModeling/KarplusString.cpp](#)

## 5.60 daisysp::StringVoice Class Reference

Extended Karplus-Strong, with all the niceties from Rings.

```
#include <stringvoice.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- void [Reset](#) ()
- float [Process](#) (bool trigger=false)
- void [SetSustain](#) (bool sustain)
- void [Trig](#) ()
- void [SetFreq](#) (float freq)
- void [SetAccent](#) (float accent)
- void [SetStructure](#) (float structure)
- void [SetBrightness](#) (float brightness)
- void [SetDamping](#) (float damping)
- float [GetAux](#) ()

### 5.60.1 Detailed Description

Extended Karplus-Strong, with all the niceties from Rings.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/physical\_modelling/string\_voice.h and pichenettes/eurorack/plaits/dsp/physical\_modelling/string\_voice.cc to an independent module.  
Original code written by Emilie Gillet in 2016.

### 5.60.2 Member Function Documentation

#### 5.60.2.1 GetAux()

```
float StringVoice::GetAux ( )
```

Get the raw excitation signal. Must call [Process\(\)](#) first.

#### 5.60.2.2 Init()

```
void StringVoice::Init (
    float sample_rate )
```

Initialize the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

**5.60.2.3 Process()**

```
float StringVoice::Process (
    bool trigger = false )
```

Get the next sample

## Parameters

<i>trigger</i>	Strike the string. Defaults to false.
----------------	---------------------------------------

**5.60.2.4 Reset()**

```
void StringVoice::Reset ( )
```

Reset the string oscillator

**5.60.2.5 SetAccent()**

```
void StringVoice::SetAccent (
    float accent )
```

Hit the string a bit harder. Influences brightness and decay.

## Parameters

<i>accent</i>	Works 0-1.
---------------	------------

**5.60.2.6 SetBrightness()**

```
void StringVoice::SetBrightness (
    float brightness )
```

Set the brightness of the string, and the noise density.



## Parameters

<i>brightness</i>	Works best 0-1
-------------------	----------------

**5.60.2.7 SetDamping()**

```
void StringVoice::SetDamping (
    float damping )
```

How long the resonant body takes to decay relative to the accent level.

## Parameters

<i>damping</i>	Works best 0-1. Full damp is only achieved with full accent.
----------------	--

**5.60.2.8 SetFreq()**

```
void StringVoice::SetFreq (
    float freq )
```

Set the string root frequency.

## Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

**5.60.2.9 SetStructure()**

```
void StringVoice::SetStructure (
    float structure )
```

Changes the string's nonlinearity (string type).

## Parameters

<i>structure</i>	Works 0-1. 0-.26 is curved bridge, .26-1 is dispersion.
------------------	---

### 5.60.2.10 SetSustain()

```
void StringVoice::SetSustain (
    bool sustain )
```

Continually excite the string with noise.

#### Parameters

<i>sustain</i>	True turns on the noise.
----------------	--------------------------

### 5.60.2.11 Trig()

```
void StringVoice::Trig ( )
```

Strike the string.

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

- [Source/PhysicalModeling/stringvoice.h](#)
- [Source/PhysicalModeling/stringvoice.cpp](#)

## 5.61 daisyp::Svf Class Reference

```
#include <svf.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- void [Process](#) (float in)
- void [SetFreq](#) (float f)
- void [SetRes](#) (float r)
- void [SetDrive](#) (float d)
- float [Low](#) ()
- float [High](#) ()
- float [Band](#) ()
- float [Notch](#) ()
- float [Peak](#) ()

### 5.61.1 Detailed Description

Double Sampled, Stable State Variable Filter

Credit to Andrew Simper from musicdsp.org

This is his "State Variable Filter (Double Sampled, Stable)"

Additional thanks to Laurent de Soras for stability limit, and Stefan Diedrichsen for the correct notch output

Ported by: Stephen Hensley

## 5.61.2 Member Function Documentation

### 5.61.2.1 Band()

```
float daisysp::Svf::Band ( ) [inline]
```

bandpass output

#### Returns

band pass output of the filter

### 5.61.2.2 High()

```
float daisysp::Svf::High ( ) [inline]
```

highpass output

#### Returns

high pass output of the filter

### 5.61.2.3 Init()

```
void Svf::Init (
    float sample_rate )
```

Initializes the filter float *sample\_rate* - sample rate of the audio engine being run, and the frequency that the Process function will be called.

### 5.61.2.4 Low()

```
float daisysp::Svf::Low ( ) [inline]
```

lowpass output

#### Returns

low pass output of the filter

### 5.61.2.5 Notch()

```
float daisyssp::Svf::Notch ( ) [inline]
```

notchpass output

#### Returns

notch pass output of the filter

### 5.61.2.6 Peak()

```
float daisyssp::Svf::Peak ( ) [inline]
```

peak output

#### Returns

peak output of the filter

### 5.61.2.7 Process()

```
void Svf::Process (
    float in )
```

Process the input signal, updating all of the outputs.

### 5.61.2.8 SetDrive()

```
void Svf::SetDrive (
    float d )
```

sets the drive of the filter affects the response of the resonance of the filter

### 5.61.2.9 SetFreq()

```
void Svf::SetFreq (
    float f )
```

sets the frequency of the cutoff frequency.  $f$  must be between 0.0 and  $\text{sample\_rate} / 3$

### 5.61.2.10 SetRes()

```
void SvF::SetRes (
    float r )
```

sets the resonance of the filter. Must be between 0.0 and 1.0 to ensure stability.

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

- Source/Filters/svf.h
- Source/Filters/svf.cpp

## 5.62 daisysp::SwingVCA Class Reference

Swing type VCA.

```
#include <hihat.h>
```

### Public Member Functions

- float **operator()** (float s, float gain)

### 5.62.1 Detailed Description

Swing type VCA.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/hihat.h to an independent module.

Original code written by Emilie Gillet in 2016.

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

- Source/Drums/[hihat.h](#)

## 5.63 daisysp::SyntheticBassDrum Class Reference

Naive bass drum model (modulated oscillator with FM + envelope).

```
#include <synthbassdrum.h>
```

## Public Member Functions

- void [Init](#) (float sample\_rate)
- float [DistortedSine](#) (float phase, float phase\_noise, float dirtiness)
- float [TransistorVCA](#) (float s, float gain)
- float [Process](#) (bool trigger=false)
- void [Trig](#) ()
- void [SetSustain](#) (bool sustain)
- void [SetAccent](#) (float accent)
- void [SetFreq](#) (float freq)
- void [SetTone](#) (float tone)
- void [SetDecay](#) (float decay)
- void [SetDirtiness](#) (float dirtiness)
- void [SetFmEnvelopeAmount](#) (float fm\_envelope\_amount)
- void [SetFmEnvelopeDecay](#) (float fm\_envelope\_decay)

### 5.63.1 Detailed Description

Naive bass drum model (modulated oscillator with FM + envelope).

#### Author

Ben Sergentanis

#### Date

Jan 2021 Inadvertently 909-ish.

Ported from pichenettes/eurorack/plaits/dsp/drums/synthetic\_bass\_drum.h to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.63.2 Member Function Documentation

#### 5.63.2.1 DistortedSine()

```
float SyntheticBassDrum::DistortedSine (
    float phase,
    float phase_noise,
    float dirtiness ) [inline]
```

Generates a distorted sine wave

#### 5.63.2.2 Init()

```
void SyntheticBassDrum::Init (
    float sample_rate )
```

Init the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

**5.63.2.3 Process()**

```
float SyntheticBassDrum::Process (  
    bool trigger = false )
```

Get the next sample.

## Parameters

<i>trigger</i>	True triggers the BD. This is optional.
----------------	---

**5.63.2.4 SetAccent()**

```
void SyntheticBassDrum::SetAccent (  
    float accent )
```

Sets the amount of accent.

## Parameters

<i>accent</i>	Works 0-1.
---------------	------------

**5.63.2.5 SetDecay()**

```
void SyntheticBassDrum::SetDecay (  
    float decay )
```

Sets how long the drum's volume takes to decay.

## Parameters

<i>Works</i>	0-1.
--------------	------

### 5.63.2.6 SetDirtiness()

```
void SyntheticBassDrum::SetDirtiness (
    float dirtiness )
```

Makes things grimy

#### Parameters

<i>dirtiness</i>	Works 0-1.
------------------	------------

### 5.63.2.7 SetFmEnvelopeAmount()

```
void SyntheticBassDrum::SetFmEnvelopeAmount (
    float fm_envelope_amount )
```

Sets how much of a pitch sweep the drum experiences when triggered.

#### Parameters

<i>fm_envelope_amount</i>	Works 0-1.
---------------------------	------------

### 5.63.2.8 SetFmEnvelopeDecay()

```
void SyntheticBassDrum::SetFmEnvelopeDecay (
    float fm_envelope_decay )
```

Sets how long the initial pitch sweep takes.

#### Parameters

<i>fm_envelope_decay</i>	Works 0-1.
--------------------------	------------

### 5.63.2.9 SetFreq()

```
void SyntheticBassDrum::SetFreq (
    float freq )
```

Set the bass drum's root frequency.



## Parameters

<i>Frequency</i>	in Hz.
------------------	--------

**5.63.2.10 SetSustain()**

```
void SyntheticBassDrum::SetSustain (
    bool sustain )
```

Allows the drum to play continuously

## Parameters

<i>sustain</i>	True sets the drum on infinite sustain.
----------------	---

**5.63.2.11 SetTone()**

```
void SyntheticBassDrum::SetTone (
    float tone )
```

Sets the overall bright / darkness of the drum.

## Parameters

<i>tone</i>	Works 0-1.
-------------	------------

**5.63.2.12 TransistorVCA()**

```
float SyntheticBassDrum::TransistorVCA (
    float s,
    float gain ) [inline]
```

Transistor VCA simulation.

## Parameters

<i>s</i>	Input sample.
<i>gain</i>	VCA gain.

### 5.63.2.13 Trig()

```
void SyntheticBassDrum::Trig ( )
```

Trigger the drum

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

- Source/Drums/[synthbassdrum.h](#)
- Source/Drums/[synthbassdrum.cpp](#)

## 5.64 daisyp::SyntheticBassDrumAttackNoise Class Reference

Attack Noise generator for [SyntheticBassDrum](#).

```
#include <synthbassdrum.h>
```

### Public Member Functions

- void [Init](#) ()
- float [Process](#) ()

#### 5.64.1 Detailed Description

Attack Noise generator for [SyntheticBassDrum](#).

##### Author

Ben Sergentanis

##### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/synthetic\_bass\_drum.h to an independent module.  
Original code written by Emilie Gillet in 2016.

#### 5.64.2 Member Function Documentation

##### 5.64.2.1 Init()

```
void SyntheticBassDrumAttackNoise::Init ( )
```

Init the module

### 5.64.2.2 Process()

```
float SyntheticBassDrumAttackNoise::Process ( )
```

Get the next sample.

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

- Source/Drums/[synthbassdrum.h](#)
- Source/Drums/[synthbassdrum.cpp](#)

## 5.65 daisysp::SyntheticBassDrumClick Class Reference

Click noise for [SyntheticBassDrum](#).

```
#include <synthbassdrum.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float in)

### 5.65.1 Detailed Description

Click noise for [SyntheticBassDrum](#).

#### Author

Ben Sergentanis

#### Date

Jan 2021 Ported from pichenettes/eurorack/plaits/dsp/drums/synthetic\_bass\_drum.h to an independent module.  
Original code written by Emilie Gillet in 2016.

### 5.65.2 Member Function Documentation

#### 5.65.2.1 Init()

```
void SyntheticBassDrumClick::Init (
    float sample_rate )
```

Init the module

## Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

**5.65.2.2 Process()**

```
float SyntheticBassDrumClick::Process (
    float in )
```

Get the next sample.

## Parameters

<i>in</i>	Trigger the click.
-----------	--------------------

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

- Source/Drums/[synthbassdrum.h](#)
- Source/Drums/synthbassdrum.cpp

**5.66 daisysp::SyntheticSnareDrum Class Reference**

Naive snare drum model (two modulated oscillators + filtered noise).

```
#include <synthsnaredrum.h>
```

**Public Member Functions**

- void [Init](#) (float sample\_rate)
- float [Process](#) (bool trigger=false)
- void [Trig](#) ()
- void [SetSustain](#) (bool sustain)
- void [SetAccent](#) (float accent)
- void [SetFreq](#) (float f0)
- void [SetFmAmount](#) (float fm\_amount)
- void [SetDecay](#) (float decay)
- void [SetSnappy](#) (float snappy)

## 5.66.1 Detailed Description

Naive snare drum model (two modulated oscillators + filtered noise).

### Author

Ben Sergentanis

### Date

Jan 2021 Uses a few magic numbers taken from the 909 schematics:

- Ratio between the two modes of the drum set to 1.47.
- Funky coupling between the two modes.
- Noise coloration filters and envelope shapes for the snare.

Ported from pichenettes/eurorack/plaits/dsp/drums/synthetic\_snare\_drum.h to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.66.2 Member Function Documentation

### 5.66.2.1 Init()

```
void SyntheticSnareDrum::Init (  
    float sample_rate )
```

Init the module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

### 5.66.2.2 Process()

```
float SyntheticSnareDrum::Process (  
    bool trigger = false )
```

Get the next sample.

## Parameters

<i>trigger</i>	True = hit the drum. This argument is optional.
----------------	---

**5.66.2.3 SetAccent()**

```
void SyntheticSnareDrum::SetAccent (
    float accent )
```

Set how much accent to use

## Parameters

<i>accent</i>	Works 0-1.
---------------	------------

**5.66.2.4 SetDecay()**

```
void SyntheticSnareDrum::SetDecay (
    float decay )
```

Set the length of the drum decay

## Parameters

<i>decay</i>	Works with positive numbers
--------------	-----------------------------

**5.66.2.5 SetFmAmount()**

```
void SyntheticSnareDrum::SetFmAmount (
    float fm_amount )
```

Set the amount of fm sweep.

## Parameters

<i>fm_amount</i>	Works from 0 - 1.
------------------	-------------------

### 5.66.2.6 SetFreq()

```
void SyntheticSnareDrum::SetFreq (
    float f0 )
```

Set the drum's root frequency

#### Parameters

<i>f0</i>	Freq in Hz
-----------	------------

### 5.66.2.7 SetSnappy()

```
void SyntheticSnareDrum::SetSnappy (
    float snappy )
```

Sets the mix between snare and drum.

#### Parameters

<i>snappy</i>	1 = just snare. 0 = just drum.
---------------	--------------------------------

### 5.66.2.8 SetSustain()

```
void SyntheticSnareDrum::SetSustain (
    bool sustain )
```

Make the drum ring out infinitely.

#### Parameters

<i>sustain</i>	True = infinite sustain.
----------------	--------------------------

### 5.66.2.9 Trig()

```
void SyntheticSnareDrum::Trig ( )
```

Trigger the drum

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

- Source/Drums/[synthsnaredrum.h](#)
- Source/Drums/[synthsnaredrum.cpp](#)

## 5.67 daisysp::Tone Class Reference

```
#include <tone.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) (float &in)
- void [SetFreq](#) (float &freq)
- float [GetFreq](#) ()

### 5.67.1 Detailed Description

A first-order recursive low-pass filter with variable frequency response.

### 5.67.2 Member Function Documentation

#### 5.67.2.1 GetFreq()

```
float daisysp::Tone::GetFreq ( ) [inline]
```

#### Returns

the current value for the cutoff frequency or half-way point of the filter.

#### 5.67.2.2 Init()

```
void Tone::Init (
    float sample_rate )
```

Initializes the [Tone](#) module. `sample_rate` - The sample rate of the audio engine being run.

#### 5.67.2.3 Process()

```
float Tone::Process (
    float & in )
```

Processes one sample through the filter and returns one sample. `in` - input signal

#### 5.67.2.4 SetFreq()

```
void daisysp::Tone::SetFreq (
    float & freq ) [inline]
```

Sets the cutoff frequency or half-way point of the filter.



## Parameters

<i>freq</i>	- frequency value in Hz. Range: Any positive value.
-------------	---

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

- Source/Filters/tone.h
- Source/Filters/tone.cpp

## 5.68 daisysp::Tremolo Class Reference

**Tremolo** effect.

```
#include <tremolo.h>
```

### Public Member Functions

- void **Init** (float *sample\_rate*)
- float **Process** (float *in*)
- void **SetFreq** (float *freq*)
- void **SetWaveform** (int *waveform*)
- void **SetDepth** (float *depth*)

### 5.68.1 Detailed Description

**Tremolo** effect.

#### Author

Ben Sergentanis

#### Date

Jan 2021 Based on <https://christianfloisand.wordpress.com/2012/04/18/coding-some-tremolo>

### 5.68.2 Member Function Documentation

#### 5.68.2.1 Init()

```
void Tremolo::Init (
    float sample_rate )
```

Initializes the module

**Parameters**

<i>sample_rate</i>	The sample rate of the audio engine being run.
--------------------	--

**5.68.2.2 Process()**

```
float Tremolo::Process (  
    float in )
```

**Parameters**

<i>in</i>	Input sample.
-----------	---------------

**Returns**

Next floating point sample.

**5.68.2.3 SetDepth()**

```
void Tremolo::SetDepth (  
    float depth )
```

How much to modulate your volume.

**Parameters**

<i>depth</i>	Works 0-1.
--------------	------------

**5.68.2.4 SetFreq()**

```
void Tremolo::SetFreq (  
    float freq )
```

Sets the tremolo rate.

**Parameters**

<i>freq</i>	Tremolo freq in Hz.
-------------	---------------------

### 5.68.2.5 SetWaveform()

```
void Tremolo::SetWaveform (
    int waveform )
```

Shape of the modulating lfo

#### Parameters

<i>waveform</i>	<a href="#">Oscillator</a> waveform. Use <code>Oscillator::WAVE_SIN</code> for example.
-----------------	---

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

- Source/Effects/[tremolo.h](#)
- Source/Effects/tremolo.cpp

## 5.69 daisysp::VariableSawOscillator Class Reference

Variable Saw [Oscillator](#).

```
#include <variablesawosc.h>
```

### Public Member Functions

- void **Init** (float sample\_rate)
- float **Process** ()
- void **SetFreq** (float frequency)
- void **SetPW** (float pw)
- void **SetWaveshape** (float waveshape)

### 5.69.1 Detailed Description

Variable Saw [Oscillator](#).

#### Author

Ben Sergentanis

#### Date

Dec 2020 Saw with variable slope or notch.

Ported from [pichenettes/eurorack/plaits/dsp/oscillator/variable\\_saw\\_oscillator.h](#)

to an independent module.

Original code written by Emilie Gillet in 2016.

## 5.69.2 Member Function Documentation

### 5.69.2.1 Process()

```
float VariableSawOscillator::Process ( )
```

Get the next sample

### 5.69.2.2 SetFreq()

```
void VariableSawOscillator::SetFreq (
    float frequency )
```

Set master freq.

#### Parameters

<i>frequency</i>	Freq in Hz.
------------------	-------------

### 5.69.2.3 SetPW()

```
void VariableSawOscillator::SetPW (
    float pw )
```

Adjust the wave depending on the shape

#### Parameters

<i>pw</i>	Notch or slope. Works best -1 to 1.
-----------	-------------------------------------

### 5.69.2.4 SetWaveshape()

```
void VariableSawOscillator::SetWaveshape (
    float waveshape )
```

Slope or notch

#### Parameters

<i>waveshape</i>	0 = notch, 1 = slope
------------------	----------------------

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

- Source/Synthesis/[variablesawosc.h](#)
- Source/Synthesis/[variablesawosc.cpp](#)

## 5.70 daisysp::VariableShapeOscillator Class Reference

Variable Waveshape [Oscillator](#).

```
#include <variableshapeosc.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float frequency)
- void [SetPW](#) (float pw)
- void [SetWaveshape](#) (float waveshape)
- void [SetSync](#) (bool enable\_sync)
- void [SetSyncFreq](#) (float frequency)

### 5.70.1 Detailed Description

Variable Waveshape [Oscillator](#).

#### Author

Ben Sergentanis

#### Date

Dec 2020 Continuously variable waveform.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/variable\_shape\_oscillator.h

to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.70.2 Member Function Documentation

#### 5.70.2.1 Init()

```
void VariableShapeOscillator::Init (  
    float sample_rate )
```

Initialize the oscillator

## Parameters

<i>sample_rate</i>	Audio engine sample rate
--------------------	--------------------------

**5.70.2.2 Process()**

```
float VariableShapeOscillator::Process ( )
```

Get next sample

**5.70.2.3 SetFreq()**

```
void VariableShapeOscillator::SetFreq (
    float frequency )
```

Set master freq.

## Parameters

<i>frequency</i>	Freq in Hz.
------------------	-------------

**5.70.2.4 SetPW()**

```
void VariableShapeOscillator::SetPW (
    float pw )
```

Set pulse width / saw, ramp, tri.

## Parameters

<i>pw</i>	PW when shape is square. Saw, ramp, tri otherwise.
-----------	--

**5.70.2.5 SetSync()**

```
void VariableShapeOscillator::SetSync (
    bool enable_sync )
```

Whether or not to sync to the sync oscillator

## Parameters

<i>enable_sync</i>	True to turn sync on.
--------------------	-----------------------

**5.70.2.6 SetSyncFreq()**

```
void VariableShapeOscillator::SetSyncFreq (
    float frequency )
```

Set sync oscillator freq.

## Parameters

<i>frequency</i>	Freq in Hz.
------------------	-------------

**5.70.2.7 SetWaveshape()**

```
void VariableShapeOscillator::SetWaveshape (
    float waveshape )
```

Switch from saw/ramp/tri to square.

## Parameters

<i>waveshape</i>	0 is saw/ramp/tri, 1 is square.
------------------	---------------------------------

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

- Source/Synthesis/[variableshapeosc.h](#)
- Source/Synthesis/variableshapeosc.cpp

**5.71 daisysp::VosimOscillator Class Reference**

Vosim [Oscillator](#) Module

.

```
#include <vosim.h>
```

**Public Member Functions**

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetForm1Freq](#) (float freq)
- void [SetForm2Freq](#) (float freq)
- void [SetShape](#) (float shape)

### 5.71.1 Detailed Description

Vosim [Oscillator](#) Module

.

#### Author

Ben Sergentanis

#### Date

Dec 2020 Two sinewaves multiplied by and sync'ed to a carrier.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/vosim\_oscillator.h

to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.71.2 Member Function Documentation

#### 5.71.2.1 Init()

```
void VosimOscillator::Init (
    float sample_rate )
```

Initializes the [FormantOscillator](#) module.

#### Parameters

<i>sample_rate</i>	- The sample rate of the audio engine being run.
--------------------	--

#### 5.71.2.2 Process()

```
float VosimOscillator::Process ( )
```

Get the next sample

#### 5.71.2.3 SetForm1Freq()

```
void VosimOscillator::SetForm1Freq (
    float freq )
```

Set formant 1 frequency.



## Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

**5.71.2.4 SetForm2Freq()**

```
void VosimOscillator::SetForm2Freq (
    float freq )
```

Set formant 2 frequency.

## Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

**5.71.2.5 SetFreq()**

```
void VosimOscillator::SetFreq (
    float freq )
```

Set carrier frequency.

## Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

**5.71.2.6 SetShape()**

```
void VosimOscillator::SetShape (
    float shape )
```

Waveshaping

## Parameters

<i>shape</i>	Shape to set. Works -1 to 1
--------------	-----------------------------

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

- Source/Synthesis/[vosim.h](#)
- Source/Synthesis/[vosim.cpp](#)

## 5.72 daisysp::WhiteNoise Class Reference

```
#include <whitenoise.h>
```

### Public Member Functions

- void [Init](#) ()
- void [SetAmp](#) (float a)
- float [Process](#) ()

### 5.72.1 Detailed Description

fast white noise generator

I think this came from musicdsp.org at some point

### 5.72.2 Member Function Documentation

#### 5.72.2.1 Init()

```
void daisysp::WhiteNoise::Init ( ) [inline]
```

Initializes the [WhiteNoise](#) object

#### 5.72.2.2 Process()

```
float daisysp::WhiteNoise::Process ( ) [inline]
```

returns a new sample of noise in the range of -amp\_ to amp\_

#### 5.72.2.3 SetAmp()

```
void daisysp::WhiteNoise::SetAmp (
    float a ) [inline]
```

sets the amplitude of the noise output

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

- Source/Noise/whitenoise.h

## 5.73 daisysp::ZOscillator Class Reference

[ZOscillator](#) Module

.

```
#include <zoscillator.h>
```

### Public Member Functions

- void [Init](#) (float sample\_rate)
- float [Process](#) ()
- void [SetFreq](#) (float freq)
- void [SetFormantFreq](#) (float freq)
- void [SetShape](#) (float shape)
- void [SetMode](#) (float mode)

### 5.73.1 Detailed Description

[ZOscillator](#) Module

.

#### Author

Ben Sergentanis

#### Date

Dec 2020 Sinewave multiplied by and sync'ed to a carrier.

Ported from pichenettes/eurorack/plaits/dsp/oscillator/z\_oscillator.h

to an independent module.

Original code written by Emilie Gillet in 2016.

### 5.73.2 Member Function Documentation

#### 5.73.2.1 Init()

```
void ZOscillator::Init (  
    float sample_rate )
```

Init [ZOscillator](#) module

#### Parameters

<i>sample_rate</i>	Audio engine sample rate.
--------------------	---------------------------

### 5.73.2.2 Process()

```
float ZOscillator::Process ( )
```

Get next sample

### 5.73.2.3 SetFormantFreq()

```
void ZOscillator::SetFormantFreq (
    float freq )
```

Set the formant osc. freq

#### Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

### 5.73.2.4 SetFreq()

```
void ZOscillator::SetFreq (
    float freq )
```

Set the carrier frequency

#### Parameters

<i>freq</i>	Frequency in Hz.
-------------	------------------

### 5.73.2.5 SetMode()

```
void ZOscillator::SetMode (
    float mode )
```

Set the offset amount and phase shift.

< 1/3 is just phase shift, > 2/3 is just offset, and between them is both.

#### Parameters

<i>mode</i>	Mode to set. Works best -1 to 1
-------------	---------------------------------

### 5.73.2.6 SetShape()

```
void ZOscillator::SetShape (  
    float shape )
```

Adjust the contour of the waveform.

#### Parameters

<i>shape</i>	Waveshape to set. Works best 0-1.
--------------	-----------------------------------

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

- Source/Synthesis/[zoscillator.h](#)
- Source/Synthesis/zoscillator.cpp



## Chapter 6

# File Documentation

### 6.1 Source/Drums/analogbassdrum.h File Reference

```
#include <stdint.h>
#include "Synthesis/oscillator.h"
#include "Filters/svf.h"
```

#### Classes

- class [daisysp::AnalogBassDrum](#)  
*808 bass drum model, revisited.*

### 6.2 Source/Drums/analogsnaredrum.h File Reference

```
#include "Filters/svf.h"
#include <stdint.h>
```

#### Classes

- class [daisysp::AnalogSnareDrum](#)  
*808 snare drum model, revisited.*

### 6.3 Source/Drums/hihat.h File Reference

```
#include "Filters/svf.h"
#include "Synthesis/oscillator.h"
#include <stdint.h>
#include <stdlib.h>
```

## Classes

- class [daisysp::SquareNoise](#)  
*808 style "metallic noise" with 6 square oscillators.*
- class [daisysp::RingModNoise](#)  
*Ring mod style metallic noise generator.*
- class [daisysp::SwingVCA](#)  
*Swing type VCA.*
- class [daisysp::LinearVCA](#)  
*Linear type VCA.*
- class [daisysp::HiHat](#) < [MetallicNoiseSource](#), [VCA](#), [resonance](#) >  
*808 HH, with a few extra parameters to push things to the CY territory...*

## 6.4 Source/Drums/synthbassdrum.h File Reference

```
#include "Filters/svf.h"
#include "Utility/dsp.h"
#include <stdint.h>
```

### Classes

- class [daisysp::SyntheticBassDrumClick](#)  
*Click noise for [SyntheticBassDrum](#).*
- class [daisysp::SyntheticBassDrumAttackNoise](#)  
*Attack Noise generator for [SyntheticBassDrum](#).*
- class [daisysp::SyntheticBassDrum](#)  
*Naive bass drum model (modulated oscillator with FM + envelope).*

## 6.5 Source/Drums/synthsnaresdrum.h File Reference

```
#include "Filters/svf.h"
#include <stdint.h>
```

### Classes

- class [daisysp::SyntheticSnareDrum](#)  
*Naive snare drum model (two modulated oscillators + filtered noise).*

## 6.6 Source/Effects/chorus.h File Reference

```
#include <stdint.h>
#include "Utility/delayline.h"
```



## Classes

- class [daisysp::ChorusEngine](#)  
*Single Chorus engine. Used in Chorus.*
- class [daisysp::Chorus](#)  
*Chorus Effect.*

## 6.7 Source/Effects/flanger.h File Reference

```
#include <stdint.h>
#include "Utility/delayline.h"
```

## Classes

- class [daisysp::Flanger](#)  
*Flanging Audio Effect.*

## 6.8 Source/Effects/overdrive.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::Overdrive](#)  
*Distortion / Overdrive Module.*

## 6.9 Source/Effects/sampleratereducer.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::SampleRateReducer](#)  
*Sample rate reducer.*

## 6.10 Source/Effects/tremolo.h File Reference

```
#include <stdint.h>
#include <math.h>
#include "Synthesis/oscillator.h"
```

## Classes

- class [daisysp::Tremolo](#)  
*Tremolo effect.*

## 6.11 Source/Filters/allpass.h File Reference

```
#include <stdint.h>
#include <math.h>
```

## Classes

- class [daisysp::Allpass](#)

## 6.12 Source/Noise/clockednoise.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::ClockedNoise](#)

## 6.13 Source/Noise/dust.h File Reference

```
#include <cstdlib>
#include <random>
#include "Utility/dsp.h"
```

## Classes

- class [daisysp::Dust](#)  
*Dust Module.*

## 6.14 Source/Noise/fractal\_noise.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::FractalRandomGenerator< T, order >](#)  
*Fractal Noise, stacks octaves of a noise source.*

## 6.15 Source/Noise/grainlet.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::GrainletOscillator](#)  
*Granular Oscillator Module.*

## 6.16 Source/Noise/particle.h File Reference

```
#include "Filters/svf.h"  
#include <stdint.h>  
#include <cstdlib>
```

## Classes

- class [daisysp::Particle](#)  
*Random impulse train processed by a resonant filter.*

## 6.17 Source/PhysicalModeling/drip.h File Reference

```
#include <stdint.h>
```

## Classes

- class [daisysp::Drip](#)

## 6.18 Source/PhysicalModeling/KarplusString.h File Reference

```
#include <stdint.h>  
#include "Dynamics/crossfade.h"  
#include "Utility/dcblock.h"  
#include "Utility/delayline.h"  
#include "Filters/svf.h"  
#include "Filters/tone.h"
```

## Classes

- class [daisysp::String](#)  
*Comb filter / KS string.*

## 6.19 Source/PhysicalModeling/modalvoice.h File Reference

```
#include <stdint.h>
#include "Filters/svf.h"
#include "PhysicalModeling/resonator.h"
#include "Noise/dust.h"
```

## Classes

- class [daisysp::ModalVoice](#)  
*Simple modal synthesis voice with a mallet exciter: click -> LPF -> resonator.*

## 6.20 Source/PhysicalModeling/resonator.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include "Utility/dsp.h"
```

## Classes

- class [daisysp::ResonatorSvf< batch\\_size >](#)  
*SVF for use in the [Resonator](#) Class*
- class [daisysp::Resonator](#)  
*Resonant Body Simulation.*

## 6.21 Source/PhysicalModeling/stringvoice.h File Reference

```
#include "Filters/svf.h"
#include "PhysicalModeling/KarplusString.h"
#include "Noise/dust.h"
#include <stdint.h>
```

## Classes

- class [daisysp::StringVoice](#)  
*Extended Karplus-Strong, with all the niceties from Rings.*

## 6.22 Source/Synthesis/formantosc.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::FormantOscillator](#)  
*Formant [Oscillator](#) Module.*

## 6.23 Source/Synthesis/harmonic\_osc.h File Reference

```
#include <stdint.h>  
#include "Utility/dsp.h"
```

### Classes

- class [daisysp::HarmonicOscillator< num\\_harmonics >](#)  
*Harmonic [Oscillator](#) Module based on Chebyshev polynomials.*

## 6.24 Source/Synthesis/oscillatorbank.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::OscillatorBank](#)  
*[Oscillator](#) Bank module.*

## 6.25 Source/Synthesis/variablesawosc.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::VariableSawOscillator](#)  
*Variable Saw [Oscillator](#).*

## 6.26 Source/Synthesis/variableshapeosc.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::VariableShapeOscillator](#)  
*Variable Waveshape Oscillator.*

## 6.27 Source/Synthesis/vosim.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::VosimOscillator](#)  
*Vosim Oscillator Module*

## 6.28 Source/Synthesis/zoscillator.h File Reference

```
#include <stdint.h>
```

### Classes

- class [daisysp::ZOscillator](#)  
*ZOscillator Module*

## 6.29 Source/Utility/smooth\_random.h File Reference

```
#include "dsp.h"  
#include <stdint.h>  
#include <stdlib.h>
```

### Classes

- class [daisysp::SmoothRandomGenerator](#)  
*Smooth random generator for internal modulation.*

# Index

- Apply
  - daisysp::Compressor, 48
- AutoMakeup
  - daisysp::Compressor, 49
- Band
  - daisysp::Svf, 135
- buf
  - daisysp::ReverbScDI, 121
- buffer\_size
  - daisysp::ReverbScDI, 121
- Clear
  - daisysp::Mode, 91
- daisysp::AdEnv, 11
  - GetCurrentSegment, 11
  - GetValue, 12
  - Init, 12
  - IsRunning, 12
  - Process, 12
  - SetCurve, 12
  - SetMax, 13
  - SetMin, 13
  - SetTime, 13
  - Trigger, 13
- daisysp::Adsr, 13
  - GetCurrentSegment, 14
  - Init, 14
  - IsRunning, 14
  - Process, 15
  - SetSustainLevel, 15
  - SetTime, 15
- daisysp::Allpass, 16
  - Init, 16
  - Process, 16
  - SetFreq, 17
  - SetRevTime, 17
- daisysp::AnalogBassDrum, 17
  - Init, 18
  - Process, 18
  - SetAccent, 19
  - SetAttackFmAmount, 19
  - SetDecay, 19
  - SetFreq, 19
  - SetSelfFmAmount, 20
  - SetSustain, 20
  - SetTone, 20
  - Trig, 21
- daisysp::AnalogSnareDrum, 21
  - Init, 22
  - Process, 22
  - SetAccent, 22
  - SetDecay, 22
  - SetFreq, 24
  - SetSnappy, 24
  - SetSustain, 24
  - SetTone, 24
  - Trig, 25
- daisysp::ATone, 25
  - GetFreq, 26
  - Init, 26
  - Process, 26
  - SetFreq, 26
- daisysp::Autowah, 27
  - Init, 27
  - Process, 28
  - SetDryWet, 28
  - SetLevel, 28
  - SetWah, 28
- daisysp::Balance, 29
  - Init, 29
  - Process, 30
  - SetCutoff, 30
- daisysp::Biquad, 30
  - Init, 31
  - Process, 31
  - SetCutoff, 31
  - SetRes, 31
- daisysp::Bitcrush, 33
  - Init, 33
  - Process, 33
  - SetBitDepth, 34
  - SetCrushRate, 34
- daisysp::BIOsc, 34
  - Init, 35
  - Process, 35
  - Reset, 35
  - SetAmp, 36
  - SetFreq, 36
  - SetPw, 36
  - SetWaveform, 36
  - Waveforms, 35
- daisysp::Chorus, 37
  - GetLeft, 37
  - GetRight, 37
  - Init, 38
  - Process, 38
  - SetDelay, 38

- SetDelayMs, 39
- SetFeedback, 39, 40
- SetLfoDepth, 40
- SetLfoFreq, 41
- SetPan, 41
- daisysp::ChorusEngine, 42
  - Init, 42
  - Process, 43
  - SetDelay, 43
  - SetDelayMs, 43
  - SetFeedback, 43
  - SetLfoDepth, 44
  - SetLfoFreq, 44
- daisysp::ClockedNoise, 44
  - Init, 45
  - Process, 45
  - SetFreq, 45
  - Sync, 46
- daisysp::Comb, 46
  - Init, 46
  - Process, 47
  - SetFreq, 47
  - SetPeriod, 47
  - SetRevTime, 47
- daisysp::Compressor, 47
  - Apply, 48
  - AutoMakeup, 49
  - GetAttack, 49
  - GetGain, 49
  - GetMakeup, 49
  - GetRatio, 49
  - GetRelease, 49
  - GetThreshold, 50
  - Init, 50
  - Process, 50
  - ProcessBlock, 51
  - SetAttack, 52
  - SetMakeup, 52
  - SetRatio, 52
  - SetRelease, 53
  - SetThreshold, 53
- daisysp::CrossFade, 53
  - GetCurve, 54
  - GetPos, 54
  - Init, 54
  - Process, 54
  - SetCurve, 55
  - SetPos, 55
- daisysp::DcBlock, 55
  - Init, 55
  - Process, 56
- daisysp::Decimator, 56
  - GetBitcrushFactor, 56
  - GetDownsampleFactor, 56
  - Init, 57
  - Process, 57
  - SetBitcrushFactor, 57
  - SetBitsToCrush, 57
  - SetDownsampleFactor, 57
- daisysp::DelayLine< T, max\_size >, 58
  - Init, 58
  - Read, 58
  - Reset, 59
  - SetDelay, 59
  - Write, 59
- daisysp::Drip, 59
  - Init, 60
  - Process, 60
- daisysp::Dust, 61
- daisysp::Flanger, 62
  - Init, 62
  - Process, 62
  - SetDelay, 63
  - SetDelayMs, 63
  - SetFeedback, 63
  - SetLfoDepth, 63
  - SetLfoFreq, 64
- daisysp::Fm2, 64
  - GetIndex, 65
  - Init, 65
  - Process, 65
  - Reset, 65
  - SetFrequency, 65
  - SetIndex, 66
  - SetRatio, 66
- daisysp::Fold, 66
  - Init, 67
  - Process, 67
  - SetIncrement, 67
- daisysp::FormantOscillator, 67
  - Init, 68
  - Process, 68
  - SetCarrierFreq, 68
  - SetFormantFreq, 69
  - SetPhaseShift, 69
- daisysp::FractalRandomGenerator< T, order >, 69
  - Init, 70
  - Process, 70
  - SetColor, 70
  - SetFreq, 71
- daisysp::GrainletOscillator, 71
  - Init, 72
  - Process, 72
  - SetBleed, 72
  - SetFormantFreq, 72
  - SetFreq, 73
  - SetShape, 73
- daisysp::HarmonicOscillator< num\_harmonics >, 73
  - Init, 74
  - Process, 74
  - SetAmplitudes, 75
  - SetFirstHarmIdx, 75
  - SetFreq, 75
  - SetSingleAmp, 76
- daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 76



- Init, [77](#)
- Process, [77](#)
- SetAccent, [77](#)
- SetDecay, [78](#)
- SetFreq, [78](#)
- SetNoisiness, [78](#)
- SetSustain, [79](#)
- SetTone, [79](#)
- Trig, [79](#)
- daisysp::Jitter, [80](#)
  - Init, [80](#)
  - Process, [80](#)
  - SetAmp, [80](#)
  - SetCpsMax, [81](#)
  - SetCpsMin, [81](#)
- daisysp::Limiter, [81](#)
  - Init, [82](#)
  - ProcessBlock, [82](#)
- daisysp::Line, [82](#)
  - Init, [83](#)
  - Process, [83](#)
  - Start, [83](#)
- daisysp::LinearVCA, [84](#)
- daisysp::Maytrig, [84](#)
  - Process, [85](#)
- daisysp::Metro, [86](#)
  - GetFreq, [86](#)
  - Init, [86](#)
  - Process, [87](#)
  - Reset, [87](#)
  - SetFreq, [87](#)
- daisysp::ModalVoice, [87](#)
  - GetAux, [88](#)
  - Init, [88](#)
  - Process, [89](#)
  - SetAccent, [89](#)
  - SetBrightness, [89](#)
  - SetDamping, [89](#)
  - SetFreq, [90](#)
  - SetStructure, [90](#)
  - SetSustain, [90](#)
  - Trig, [91](#)
- daisysp::Mode, [91](#)
  - Clear, [91](#)
  - Init, [92](#)
  - Process, [92](#)
  - SetFreq, [92](#)
  - SetQ, [92](#)
- daisysp::MoogLadder, [92](#)
  - Init, [93](#)
  - Process, [93](#)
  - SetFreq, [93](#)
  - SetRes, [93](#)
- daisysp::NIFilt, [94](#)
  - Init, [95](#)
  - ProcessBlock, [95](#)
  - SetA, [95](#)
  - SetB, [95](#)
  - SetC, [95](#)
  - SetCoefficients, [95](#)
  - SetD, [95](#)
  - SetL, [96](#)
- daisysp::Oscillator, [96](#)
  - Init, [97](#)
  - IsEOC, [97](#)
  - IsEOR, [97](#)
  - IsFalling, [97](#)
  - IsRising, [98](#)
  - PhaseAdd, [98](#)
  - Process, [98](#)
  - Reset, [98](#)
  - SetAmp, [98](#)
  - SetFreq, [98](#)
  - SetWaveform, [98](#)
- daisysp::OscillatorBank, [99](#)
  - Init, [99](#)
  - Process, [100](#)
  - SetAmplitudes, [100](#)
  - SetFreq, [100](#)
  - SetGain, [100](#)
  - SetSingleAmp, [101](#)
- daisysp::Overdrive, [101](#)
  - Init, [102](#)
  - Process, [102](#)
  - SetDrive, [102](#)
- daisysp::Particle, [102](#)
  - GetNoise, [103](#)
  - Init, [103](#)
  - Process, [104](#)
  - SetDensity, [104](#)
  - SetFreq, [104](#)
  - SetGain, [104](#)
  - SetRandomFreq, [105](#)
  - SetResonance, [105](#)
  - SetSpread, [105](#)
  - SetSync, [105](#)
- daisysp::Phasor, [106](#)
  - GetFreq, [106](#)
  - Init, [106](#), [107](#)
  - Process, [107](#)
  - SetFreq, [107](#)
- daisysp::PitchShifter, [108](#)
  - Init, [108](#)
  - Process, [108](#)
  - SetDelSize, [108](#)
  - SetFun, [109](#)
  - SetTransposition, [109](#)
- daisysp::Pluck, [109](#)
  - GetAmp, [110](#)
  - GetDamp, [110](#)
  - GetDecay, [110](#)
  - GetFreq, [110](#)
  - GetMode, [110](#)
  - Init, [110](#)
  - Process, [111](#)
  - SetAmp, [111](#)

- SetDamp, [111](#)
- SetDecay, [111](#)
- SetFreq, [111](#)
- SetMode, [111](#)
- daisysp::PolyPluck< num\_voices >, [112](#)
  - Init, [112](#)
  - Process, [113](#)
  - SetDecay, [113](#)
- daisysp::Port, [113](#)
  - GetHtime, [114](#)
  - Init, [114](#)
  - Process, [114](#)
  - SetHtime, [115](#)
- daisysp::Resonator, [115](#)
  - Init, [116](#)
  - Process, [117](#)
  - SetBrightness, [117](#)
  - SetDamping, [117](#)
  - SetFreq, [117](#)
  - SetStructure, [118](#)
- daisysp::ResonatorSvf< batch\_size >, [118](#)
- daisysp::ReverbSc, [119](#)
  - Init, [120](#)
  - Process, [120](#)
  - SetFeedback, [120](#)
  - SetLpFreq, [120](#)
- daisysp::ReverbScDI, [121](#)
  - buf, [121](#)
  - buffer\_size, [121](#)
  - dummy, [121](#)
  - filter\_state, [121](#)
  - rand\_line\_cnt, [122](#)
  - read\_pos, [122](#)
  - read\_pos\_frac, [122](#)
  - read\_pos\_frac\_inc, [122](#)
  - seed\_val, [122](#)
  - write\_pos, [122](#)
- daisysp::RingModNoise, [123](#)
- daisysp::SampleHold, [123](#)
  - Process, [124](#)
- daisysp::SampleRateReducer, [124](#)
  - Init, [125](#)
  - Process, [125](#)
  - SetFreq, [125](#)
- daisysp::SmoothRandomGenerator, [126](#)
  - Init, [126](#)
  - Process, [127](#)
  - SetFreq, [127](#)
- daisysp::SquareNoise, [127](#)
- daisysp::String, [128](#)
  - Init, [128](#)
  - Process, [129](#)
  - Reset, [129](#)
  - SetBrightness, [129](#)
  - SetDamping, [129](#)
  - SetFreq, [130](#)
  - SetNonLinearity, [130](#)
- daisysp::StringVoice, [130](#)
  - GetAux, [131](#)
  - Init, [131](#)
  - Process, [132](#)
  - Reset, [132](#)
  - SetAccent, [132](#)
  - SetBrightness, [132](#)
  - SetDamping, [133](#)
  - SetFreq, [133](#)
  - SetStructure, [133](#)
  - SetSustain, [133](#)
  - Trig, [134](#)
- daisysp::Svf, [134](#)
  - Band, [135](#)
  - High, [135](#)
  - Init, [135](#)
  - Low, [135](#)
  - Notch, [135](#)
  - Peak, [136](#)
  - Process, [136](#)
  - SetDrive, [136](#)
  - SetFreq, [136](#)
  - SetRes, [136](#)
- daisysp::SwingVCA, [137](#)
- daisysp::SyntheticBassDrum, [137](#)
  - DistortedSine, [138](#)
  - Init, [138](#)
  - Process, [139](#)
  - SetAccent, [139](#)
  - SetDecay, [139](#)
  - SetDirtiness, [139](#)
  - SetFmEnvelopeAmount, [140](#)
  - SetFmEnvelopeDecay, [140](#)
  - SetFreq, [140](#)
  - SetSustain, [141](#)
  - SetTone, [141](#)
  - TransistorVCA, [141](#)
  - Trig, [141](#)
- daisysp::SyntheticBassDrumAttackNoise, [142](#)
  - Init, [142](#)
  - Process, [142](#)
- daisysp::SyntheticBassDrumClick, [143](#)
  - Init, [143](#)
  - Process, [144](#)
- daisysp::SyntheticSnareDrum, [144](#)
  - Init, [145](#)
  - Process, [145](#)
  - SetAccent, [146](#)
  - SetDecay, [146](#)
  - SetFmAmount, [146](#)
  - SetFreq, [146](#)
  - SetSnappy, [147](#)
  - SetSustain, [147](#)
  - Trig, [147](#)
- daisysp::Tone, [148](#)
  - GetFreq, [148](#)
  - Init, [148](#)
  - Process, [148](#)
  - SetFreq, [148](#)

- daisysp::Tremolo, [149](#)
  - Init, [149](#)
  - Process, [150](#)
  - SetDepth, [150](#)
  - SetFreq, [150](#)
  - SetWaveform, [150](#)
- daisysp::VariableSawOscillator, [151](#)
  - Process, [152](#)
  - SetFreq, [152](#)
  - SetPW, [152](#)
  - SetWaveshape, [152](#)
- daisysp::VariableShapeOscillator, [153](#)
  - Init, [153](#)
  - Process, [154](#)
  - SetFreq, [154](#)
  - SetPW, [154](#)
  - SetSync, [154](#)
  - SetSyncFreq, [155](#)
  - SetWaveshape, [155](#)
- daisysp::VosimOscillator, [155](#)
  - Init, [156](#)
  - Process, [156](#)
  - SetForm1Freq, [156](#)
  - SetForm2Freq, [157](#)
  - SetFreq, [157](#)
  - SetShape, [157](#)
- daisysp::WhiteNoise, [158](#)
  - Init, [158](#)
  - Process, [158](#)
  - SetAmp, [158](#)
- daisysp::ZOscillator, [159](#)
  - Init, [159](#)
  - Process, [160](#)
  - SetFormantFreq, [160](#)
  - SetFreq, [160](#)
  - SetMode, [160](#)
  - SetShape, [161](#)
- DistortedSine
  - daisysp::SyntheticBassDrum, [138](#)
- dummy
  - daisysp::ReverbScDI, [121](#)
- filter\_state
  - daisysp::ReverbScDI, [121](#)
- GetAmp
  - daisysp::Pluck, [110](#)
- GetAttack
  - daisysp::Compressor, [49](#)
- GetAux
  - daisysp::ModalVoice, [88](#)
  - daisysp::StringVoice, [131](#)
- GetBitcrushFactor
  - daisysp::Decimator, [56](#)
- GetCurrentSegment
  - daisysp::AdEnv, [11](#)
  - daisysp::Adsr, [14](#)
- GetCurve
  - daisysp::CrossFade, [54](#)
- GetDamp
  - daisysp::Pluck, [110](#)
- GetDecay
  - daisysp::Pluck, [110](#)
- GetDownsampleFactor
  - daisysp::Decimator, [56](#)
- GetFreq
  - daisysp::ATone, [26](#)
  - daisysp::Metro, [86](#)
  - daisysp::Phasor, [106](#)
  - daisysp::Pluck, [110](#)
  - daisysp::Tone, [148](#)
- GetGain
  - daisysp::Compressor, [49](#)
- GetHtime
  - daisysp::Port, [114](#)
- GetIndex
  - daisysp::Fm2, [65](#)
- GetLeft
  - daisysp::Chorus, [37](#)
- GetMakeup
  - daisysp::Compressor, [49](#)
- GetMode
  - daisysp::Pluck, [110](#)
- GetNoise
  - daisysp::Particle, [103](#)
- GetPos
  - daisysp::CrossFade, [54](#)
- GetRatio
  - daisysp::Compressor, [49](#)
- GetRelease
  - daisysp::Compressor, [49](#)
- GetRight
  - daisysp::Chorus, [37](#)
- GetThreshold
  - daisysp::Compressor, [50](#)
- GetValue
  - daisysp::AdEnv, [12](#)
- High
  - daisysp::Svf, [135](#)
- Init
  - daisysp::AdEnv, [12](#)
  - daisysp::Adsr, [14](#)
  - daisysp::Allpass, [16](#)
  - daisysp::AnalogBassDrum, [18](#)
  - daisysp::AnalogSnareDrum, [22](#)
  - daisysp::ATone, [26](#)
  - daisysp::Autowah, [27](#)
  - daisysp::Balance, [29](#)
  - daisysp::Biquad, [31](#)
  - daisysp::Bitcrush, [33](#)
  - daisysp::BIOsc, [35](#)
  - daisysp::Chorus, [38](#)
  - daisysp::ChorusEngine, [42](#)
  - daisysp::ClockedNoise, [45](#)
  - daisysp::Comb, [46](#)
  - daisysp::Compressor, [50](#)

- daisysp::CrossFade, 54
- daisysp::DcBlock, 55
- daisysp::Decimator, 57
- daisysp::DelayLine< T, max\_size >, 58
- daisysp::Drip, 60
- daisysp::Flanger, 62
- daisysp::Fm2, 65
- daisysp::Fold, 67
- daisysp::FormantOscillator, 68
- daisysp::FractalRandomGenerator< T, order >, 70
- daisysp::GrainletOscillator, 72
- daisysp::HarmonicOscillator< num\_harmonics >, 74
- daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 77
- daisysp::Jitter, 80
- daisysp::Limiter, 82
- daisysp::Line, 83
- daisysp::Metro, 86
- daisysp::ModalVoice, 88
- daisysp::Mode, 92
- daisysp::MoogLadder, 93
- daisysp::NIFilt, 95
- daisysp::Oscillator, 97
- daisysp::OscillatorBank, 99
- daisysp::Overdrive, 102
- daisysp::Particle, 103
- daisysp::Phasor, 106, 107
- daisysp::PitchShifter, 108
- daisysp::Pluck, 110
- daisysp::PolyPluck< num\_voices >, 112
- daisysp::Port, 114
- daisysp::Resonator, 116
- daisysp::ReverbSc, 120
- daisysp::SampleRateReducer, 125
- daisysp::SmoothRandomGenerator, 126
- daisysp::String, 128
- daisysp::StringVoice, 131
- daisysp::Svf, 135
- daisysp::SyntheticBassDrum, 138
- daisysp::SyntheticBassDrumAttackNoise, 142
- daisysp::SyntheticBassDrumClick, 143
- daisysp::SyntheticSnareDrum, 145
- daisysp::Tone, 148
- daisysp::Tremolo, 149
- daisysp::VariableShapeOscillator, 153
- daisysp::VosimOscillator, 156
- daisysp::WhiteNoise, 158
- daisysp::ZOscillator, 159
- IsEOC
  - daisysp::Oscillator, 97
- IsEOR
  - daisysp::Oscillator, 97
- IsFalling
  - daisysp::Oscillator, 97
- IsRising
  - daisysp::Oscillator, 98
- IsRunning
  - daisysp::AdEnv, 12
  - daisysp::Adsr, 14
- Low
  - daisysp::Svf, 135
- Notch
  - daisysp::Svf, 135
- Peak
  - daisysp::Svf, 136
- PhaseAdd
  - daisysp::Oscillator, 98
- Process
  - daisysp::AdEnv, 12
  - daisysp::Adsr, 15
  - daisysp::Allpass, 16
  - daisysp::AnalogBassDrum, 18
  - daisysp::AnalogSnareDrum, 22
  - daisysp::ATone, 26
  - daisysp::Autowah, 28
  - daisysp::Balance, 30
  - daisysp::Biquad, 31
  - daisysp::Bitcrush, 33
  - daisysp::BIOsc, 35
  - daisysp::Chorus, 38
  - daisysp::ChorusEngine, 43
  - daisysp::ClockedNoise, 45
  - daisysp::Comb, 47
  - daisysp::Compressor, 50
  - daisysp::CrossFade, 54
  - daisysp::DcBlock, 56
  - daisysp::Decimator, 57
  - daisysp::Drip, 60
  - daisysp::Flanger, 62
  - daisysp::Fm2, 65
  - daisysp::Fold, 67
  - daisysp::FormantOscillator, 68
  - daisysp::FractalRandomGenerator< T, order >, 70
  - daisysp::GrainletOscillator, 72
  - daisysp::HarmonicOscillator< num\_harmonics >, 74
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 77
  - daisysp::Jitter, 80
  - daisysp::Line, 83
  - daisysp::Maytrig, 85
  - daisysp::Metro, 87
  - daisysp::ModalVoice, 89
  - daisysp::Mode, 92
  - daisysp::MoogLadder, 93
  - daisysp::Oscillator, 98
  - daisysp::OscillatorBank, 100
  - daisysp::Overdrive, 102
  - daisysp::Particle, 104
  - daisysp::Phasor, 107
  - daisysp::PitchShifter, 108
  - daisysp::Pluck, 111
  - daisysp::PolyPluck< num\_voices >, 113

- daisysp::Port, 114
- daisysp::Resonator, 117
- daisysp::ReverbSc, 120
- daisysp::SampleHold, 124
- daisysp::SampleRateReducer, 125
- daisysp::SmoothRandomGenerator, 127
- daisysp::String, 129
- daisysp::StringVoice, 132
- daisysp::Svf, 136
- daisysp::SyntheticBassDrum, 139
- daisysp::SyntheticBassDrumAttackNoise, 142
- daisysp::SyntheticBassDrumClick, 144
- daisysp::SyntheticSnareDrum, 145
- daisysp::Tone, 148
- daisysp::Tremolo, 150
- daisysp::VariableSawOscillator, 152
- daisysp::VariableShapeOscillator, 154
- daisysp::VosimOscillator, 156
- daisysp::WhiteNoise, 158
- daisysp::ZOscillator, 160
- ProcessBlock
  - daisysp::Compressor, 51
  - daisysp::Limiter, 82
  - daisysp::NIFilt, 95
- rand\_line\_cnt
  - daisysp::ReverbScDI, 122
- Read
  - daisysp::DelayLine< T, max\_size >, 58
- read\_pos
  - daisysp::ReverbScDI, 122
- read\_pos\_frac
  - daisysp::ReverbScDI, 122
- read\_pos\_frac\_inc
  - daisysp::ReverbScDI, 122
- Reset
  - daisysp::BIOsc, 35
  - daisysp::DelayLine< T, max\_size >, 59
  - daisysp::Fm2, 65
  - daisysp::Metro, 87
  - daisysp::Oscillator, 98
  - daisysp::String, 129
  - daisysp::StringVoice, 132
- seed\_val
  - daisysp::ReverbScDI, 122
- SetA
  - daisysp::NIFilt, 95
- SetAccent
  - daisysp::AnalogBassDrum, 19
  - daisysp::AnalogSnareDrum, 22
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 77
  - daisysp::ModalVoice, 89
  - daisysp::StringVoice, 132
  - daisysp::SyntheticBassDrum, 139
  - daisysp::SyntheticSnareDrum, 146
- SetAmp
  - daisysp::BIOsc, 36
- daisysp::Jitter, 80
- daisysp::Oscillator, 98
- daisysp::Pluck, 111
- daisysp::WhiteNoise, 158
- SetAmplitudes
  - daisysp::HarmonicOscillator< num\_harmonics >, 75
  - daisysp::OscillatorBank, 100
- SetAttack
  - daisysp::Compressor, 52
- SetAttackFmAmount
  - daisysp::AnalogBassDrum, 19
- SetB
  - daisysp::NIFilt, 95
- SetBitcrushFactor
  - daisysp::Decimator, 57
- SetBitDepth
  - daisysp::Bitcrush, 34
- SetBitsToCrush
  - daisysp::Decimator, 57
- SetBleed
  - daisysp::GrainletOscillator, 72
- SetBrightness
  - daisysp::ModalVoice, 89
  - daisysp::Resonator, 117
  - daisysp::String, 129
  - daisysp::StringVoice, 132
- SetC
  - daisysp::NIFilt, 95
- SetCarrierFreq
  - daisysp::FormantOscillator, 68
- SetCoefficients
  - daisysp::NIFilt, 95
- SetColor
  - daisysp::FractalRandomGenerator< T, order >, 70
- SetCpsMax
  - daisysp::Jitter, 81
- SetCpsMin
  - daisysp::Jitter, 81
- SetCrushRate
  - daisysp::Bitcrush, 34
- SetCurve
  - daisysp::AdEnv, 12
  - daisysp::CrossFade, 55
- SetCutoff
  - daisysp::Balance, 30
  - daisysp::Biquad, 31
- SetD
  - daisysp::NIFilt, 95
- SetDamp
  - daisysp::Pluck, 111
- SetDamping
  - daisysp::ModalVoice, 89
  - daisysp::Resonator, 117
  - daisysp::String, 129
  - daisysp::StringVoice, 133
- SetDecay
  - daisysp::AnalogBassDrum, 19

- daisysp::AnalogSnareDrum, [22](#)
- daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, [78](#)
- daisysp::Pluck, [111](#)
- daisysp::PolyPluck< num\_voices >, [113](#)
- daisysp::SyntheticBassDrum, [139](#)
- daisysp::SyntheticSnareDrum, [146](#)
- SetDelay
  - daisysp::Chorus, [38](#)
  - daisysp::ChorusEngine, [43](#)
  - daisysp::DelayLine< T, max\_size >, [59](#)
  - daisysp::Flanger, [63](#)
- SetDelayMs
  - daisysp::Chorus, [39](#)
  - daisysp::ChorusEngine, [43](#)
  - daisysp::Flanger, [63](#)
- SetDelSize
  - daisysp::PitchShifter, [108](#)
- SetDensity
  - daisysp::Particle, [104](#)
- SetDepth
  - daisysp::Tremolo, [150](#)
- SetDirtiness
  - daisysp::SyntheticBassDrum, [139](#)
- SetDownsampleFactor
  - daisysp::Decimator, [57](#)
- SetDrive
  - daisysp::Overdrive, [102](#)
  - daisysp::Svf, [136](#)
- SetDryWet
  - daisysp::Autowah, [28](#)
- SetFeedback
  - daisysp::Chorus, [39](#), [40](#)
  - daisysp::ChorusEngine, [43](#)
  - daisysp::Flanger, [63](#)
  - daisysp::ReverbSc, [120](#)
- SetFirstHarmlDx
  - daisysp::HarmonicOscillator< num\_harmonics >, [75](#)
- SetFmAmount
  - daisysp::SyntheticSnareDrum, [146](#)
- SetFmEnvelopeAmount
  - daisysp::SyntheticBassDrum, [140](#)
- SetFmEnvelopeDecay
  - daisysp::SyntheticBassDrum, [140](#)
- SetForm1Freq
  - daisysp::VosimOscillator, [156](#)
- SetForm2Freq
  - daisysp::VosimOscillator, [157](#)
- SetFormantFreq
  - daisysp::FormantOscillator, [69](#)
  - daisysp::GrainletOscillator, [72](#)
  - daisysp::ZOscillator, [160](#)
- SetFreq
  - daisysp::Allpass, [17](#)
  - daisysp::AnalogBassDrum, [19](#)
  - daisysp::AnalogSnareDrum, [24](#)
  - daisysp::ATone, [26](#)
  - daisysp::BIOsc, [36](#)
  - daisysp::ClockedNoise, [45](#)
  - daisysp::Comb, [47](#)
  - daisysp::FractalRandomGenerator< T, order >, [71](#)
  - daisysp::GrainletOscillator, [73](#)
  - daisysp::HarmonicOscillator< num\_harmonics >, [75](#)
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, [78](#)
  - daisysp::Metro, [87](#)
  - daisysp::ModalVoice, [90](#)
  - daisysp::Mode, [92](#)
  - daisysp::MoogLadder, [93](#)
  - daisysp::Oscillator, [98](#)
  - daisysp::OscillatorBank, [100](#)
  - daisysp::Particle, [104](#)
  - daisysp::Phasor, [107](#)
  - daisysp::Pluck, [111](#)
  - daisysp::Resonator, [117](#)
  - daisysp::SampleRateReducer, [125](#)
  - daisysp::SmoothRandomGenerator, [127](#)
  - daisysp::String, [130](#)
  - daisysp::StringVoice, [133](#)
  - daisysp::Svf, [136](#)
  - daisysp::SyntheticBassDrum, [140](#)
  - daisysp::SyntheticSnareDrum, [146](#)
  - daisysp::Tone, [148](#)
  - daisysp::Tremolo, [150](#)
  - daisysp::VariableSawOscillator, [152](#)
  - daisysp::VariableShapeOscillator, [154](#)
  - daisysp::VosimOscillator, [157](#)
  - daisysp::ZOscillator, [160](#)
- SetFrequency
  - daisysp::Fm2, [65](#)
- SetFun
  - daisysp::PitchShifter, [109](#)
- SetGain
  - daisysp::OscillatorBank, [100](#)
  - daisysp::Particle, [104](#)
- SetHtime
  - daisysp::Port, [115](#)
- SetIncrement
  - daisysp::Fold, [67](#)
- SetIndex
  - daisysp::Fm2, [66](#)
- SetL
  - daisysp::NIFilt, [96](#)
- SetLevel
  - daisysp::Autowah, [28](#)
- SetLfoDepth
  - daisysp::Chorus, [40](#)
  - daisysp::ChorusEngine, [44](#)
  - daisysp::Flanger, [63](#)
- SetLfoFreq
  - daisysp::Chorus, [41](#)
  - daisysp::ChorusEngine, [44](#)
  - daisysp::Flanger, [64](#)
- SetLpFreq

- daisysp::ReverbSc, 120
- SetMakeup
  - daisysp::Compressor, 52
- SetMax
  - daisysp::AdEnv, 13
- SetMin
  - daisysp::AdEnv, 13
- SetMode
  - daisysp::Pluck, 111
  - daisysp::ZOscillator, 160
- SetNoisiness
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 78
- SetNonLinearity
  - daisysp::String, 130
- SetPan
  - daisysp::Chorus, 41
- SetPeriod
  - daisysp::Comb, 47
- SetPhaseShift
  - daisysp::FormantOscillator, 69
- SetPos
  - daisysp::CrossFade, 55
- SetPW
  - daisysp::VariableSawOscillator, 152
  - daisysp::VariableShapeOscillator, 154
- SetPw
  - daisysp::BIOsc, 36
- SetQ
  - daisysp::Mode, 92
- SetRandomFreq
  - daisysp::Particle, 105
- SetRatio
  - daisysp::Compressor, 52
  - daisysp::Fm2, 66
- SetRelease
  - daisysp::Compressor, 53
- SetRes
  - daisysp::Biquad, 31
  - daisysp::MoogLadder, 93
  - daisysp::Svf, 136
- SetResonance
  - daisysp::Particle, 105
- SetRevTime
  - daisysp::Allpass, 17
  - daisysp::Comb, 47
- SetSelfFmAmount
  - daisysp::AnalogBassDrum, 20
- SetShape
  - daisysp::GrainletOscillator, 73
  - daisysp::VosimOscillator, 157
  - daisysp::ZOscillator, 161
- SetSingleAmp
  - daisysp::HarmonicOscillator< num\_harmonics >, 76
  - daisysp::OscillatorBank, 101
- SetSnappy
  - daisysp::AnalogSnareDrum, 24
  - daisysp::SyntheticSnareDrum, 147
- SetSpread
  - daisysp::Particle, 105
- SetStructure
  - daisysp::ModalVoice, 90
  - daisysp::Resonator, 118
  - daisysp::StringVoice, 133
- SetSustain
  - daisysp::AnalogBassDrum, 20
  - daisysp::AnalogSnareDrum, 24
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 79
  - daisysp::ModalVoice, 90
  - daisysp::StringVoice, 133
  - daisysp::SyntheticBassDrum, 141
  - daisysp::SyntheticSnareDrum, 147
- SetSustainLevel
  - daisysp::Adsr, 15
- SetSync
  - daisysp::Particle, 105
  - daisysp::VariableShapeOscillator, 154
- SetSyncFreq
  - daisysp::VariableShapeOscillator, 155
- SetThreshold
  - daisysp::Compressor, 53
- SetTime
  - daisysp::AdEnv, 13
  - daisysp::Adsr, 15
- SetTone
  - daisysp::AnalogBassDrum, 20
  - daisysp::AnalogSnareDrum, 24
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, 79
  - daisysp::SyntheticBassDrum, 141
- SetTransposition
  - daisysp::PitchShifter, 109
- SetWah
  - daisysp::Autowah, 28
- SetWaveform
  - daisysp::BIOsc, 36
  - daisysp::Oscillator, 98
  - daisysp::Tremolo, 150
- SetWaveshape
  - daisysp::VariableSawOscillator, 152
  - daisysp::VariableShapeOscillator, 155
- Source/Drums/analogbassdrum.h, 163
- Source/Drums/analogsnaredrum.h, 163
- Source/Drums/hihat.h, 163
- Source/Drums/synthbassdrum.h, 164
- Source/Drums/synthsnaresdrum.h, 164
- Source/Effects/chorus.h, 164
- Source/Effects/flanger.h, 165
- Source/Effects/overdrive.h, 165
- Source/Effects/sampleratereducer.h, 165
- Source/Effects/tremolo.h, 165
- Source/Filters/allpass.h, 166
- Source/Noise/clockednoise.h, 166
- Source/Noise/dust.h, 166

- Source/Noise/fractal\_noise.h, [166](#)
- Source/Noise/grainlet.h, [167](#)
- Source/Noise/particle.h, [167](#)
- Source/PhysicalModeling/drip.h, [167](#)
- Source/PhysicalModeling/KarplusString.h, [167](#)
- Source/PhysicalModeling/modalvoice.h, [168](#)
- Source/PhysicalModeling/resonator.h, [168](#)
- Source/PhysicalModeling/stringvoice.h, [168](#)
- Source/Synthesis/formantosc.h, [169](#)
- Source/Synthesis/harmonic\_osc.h, [169](#)
- Source/Synthesis/oscillatorbank.h, [169](#)
- Source/Synthesis/variablesawosc.h, [169](#)
- Source/Synthesis/variableshapeosc.h, [170](#)
- Source/Synthesis/vosim.h, [170](#)
- Source/Synthesis/zoscillator.h, [170](#)
- Source/Utility/smooth\_random.h, [170](#)
- Start
  - daisysp::Line, [83](#)
- Sync
  - daisysp::ClockedNoise, [46](#)
- TransistorVCA
  - daisysp::SyntheticBassDrum, [141](#)
- Trig
  - daisysp::AnalogBassDrum, [21](#)
  - daisysp::AnalogSnareDrum, [25](#)
  - daisysp::HiHat< MetallicNoiseSource, VCA, resonance >, [79](#)
  - daisysp::ModalVoice, [91](#)
  - daisysp::StringVoice, [134](#)
  - daisysp::SyntheticBassDrum, [141](#)
  - daisysp::SyntheticSnareDrum, [147](#)
- Trigger
  - daisysp::AdEnv, [13](#)
- Waveforms
  - daisysp::BIOsc, [35](#)
- Write
  - daisysp::DelayLine< T, max\_size >, [59](#)
- write\_pos
  - daisysp::ReverbScDI, [122](#)