

DSG

Generated by Doxygen 1.8.8

Mon Nov 24 2014 22:55:28



# Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Namespace Documentation</b>	<b>11</b>
6.1	DSG Namespace Reference . . . . .	11
6.1.1	Detailed Description . . . . .	13
6.1.2	Typedef Documentation . . . . .	13
6.1.2.1	DSGFrequency . . . . .	13
6.1.2.2	DSGPhase . . . . .	13
6.1.2.3	DSGSample . . . . .	14
6.1.3	Function Documentation . . . . .	14
6.1.3.1	Abs . . . . .	14
6.1.3.2	ArrayToRing . . . . .	14
6.1.3.3	AssertBounds . . . . .	14
6.1.3.4	Cos . . . . .	14
6.1.3.5	Cos . . . . .	14
6.1.3.6	CosineInterpolate . . . . .	15
6.1.3.7	CubicInterpolate . . . . .	15
6.1.3.8	DSF . . . . .	15
6.1.3.9	EnforceBounds . . . . .	15
6.1.3.10	HermitelInterpolate . . . . .	16
6.1.3.11	IsDenormal . . . . .	16

6.1.3.12	LinearInterpolate	16
6.1.3.13	MaxHarms	16
6.1.3.14	Nyquist	17
6.1.3.15	Pow	17
6.1.3.16	RingToArray	17
6.1.3.17	SampleRate	17
6.1.3.18	SampleRate	17
6.1.3.19	Sin	18
6.1.3.20	Sin	18
6.1.3.21	Sinc	18
6.1.3.22	Sleep	18
6.1.3.23	StaticAssertBounds	18
6.2	DSG::Analog Namespace Reference	18
6.2.1	Detailed Description	19
6.3	DSG::BLIT Namespace Reference	19
6.3.1	Detailed Description	19
6.4	DSG::DPW Namespace Reference	19
6.4.1	Detailed Description	20
6.4.2	Function Documentation	20
6.4.2.1	DPW_Polynomial	20
6.4.2.2	DPW_Polynomial< 1 >	20
6.4.2.3	DPW_Polynomial< 2 >	21
6.4.2.4	DPW_Polynomial< 3 >	21
6.4.2.5	DPW_Polynomial< 4 >	21
6.4.2.6	DPW_Polynomial< 5 >	21
6.4.2.7	DPW_Polynomial< 6 >	21
6.5	DSG::EPTR Namespace Reference	22
6.5.1	Detailed Description	22
6.6	DSG::Filter Namespace Reference	22
6.6.1	Detailed Description	22
6.7	DSG::Fourier Namespace Reference	22
6.7.1	Detailed Description	23
6.8	DSG::Noise Namespace Reference	23
6.8.1	Detailed Description	23
6.8.2	Function Documentation	23
6.8.2.1	Gaussian	23
6.8.2.2	Pink	23
6.8.2.3	Random	24
6.8.2.4	White	24
6.9	DSG::Window Namespace Reference	24

6.9.1	Detailed Description	25
6.9.2	Function Documentation	25
6.9.2.1	ApplyWindow	25
6.9.2.2	ApplyWindow	25
6.9.2.3	Blackman	25
<b>7</b>	<b>Class Documentation</b>	<b>27</b>
7.1	DSG::Analog::AnalogSaw Class Reference	27
7.1.1	Detailed Description	27
7.1.2	Constructor & Destructor Documentation	28
7.1.2.1	AnalogSaw	28
7.1.2.2	AnalogSaw	28
7.1.2.3	~AnalogSaw	28
7.1.3	Member Function Documentation	28
7.1.3.1	Perform	28
7.1.3.2	Perform	28
7.1.4	Member Data Documentation	29
7.1.4.1	_stor	29
7.2	DSG::Analog::AnalogSquare Class Reference	29
7.2.1	Detailed Description	29
7.2.2	Constructor & Destructor Documentation	29
7.2.2.1	AnalogSquare	29
7.2.2.2	AnalogSquare	30
7.2.2.3	~AnalogSquare	30
7.2.3	Member Function Documentation	30
7.2.3.1	Perform	30
7.2.3.2	Perform	30
7.3	DSG::Analog::AnalogTriangle Class Reference	30
7.3.1	Detailed Description	31
7.3.2	Constructor & Destructor Documentation	31
7.3.2.1	AnalogTriangle	31
7.3.2.2	AnalogTriangle	31
7.3.2.3	~AnalogTriangle	31
7.3.3	Member Function Documentation	32
7.3.3.1	Perform	32
7.3.3.2	Perform	32
7.3.4	Member Data Documentation	32
7.3.4.1	_stor	32
7.4	DSG::AudioSettings Class Reference	32
7.4.1	Detailed Description	33

7.4.2	Member Function Documentation	33
7.4.2.1	Nyquist	33
7.4.2.2	SampleRate	33
7.4.2.3	SampleRate	33
7.4.3	Member Data Documentation	33
7.4.3.1	_nyquist	33
7.4.3.2	_sampleRate	33
7.5	DSG::BLIT::Blit Class Reference	34
7.5.1	Detailed Description	34
7.5.2	Constructor & Destructor Documentation	34
7.5.2.1	Blit	34
7.5.2.2	Blit	35
7.5.2.3	~Blit	35
7.5.3	Member Function Documentation	35
7.5.3.1	Frequency	35
7.5.3.2	Perform	35
7.5.3.3	Perform	35
7.5.4	Member Data Documentation	36
7.5.4.1	_h	36
7.5.4.2	a_	36
7.5.4.3	denominator	36
7.5.4.4	m_	36
7.5.4.5	p_	36
7.5.4.6	value	36
7.6	DSG::BLIT::BlitSaw Class Reference	36
7.6.1	Detailed Description	37
7.6.2	Constructor & Destructor Documentation	37
7.6.2.1	BlitSaw	37
7.6.2.2	BlitSaw	37
7.6.2.3	~BlitSaw	38
7.6.3	Member Function Documentation	38
7.6.3.1	Frequency	38
7.6.3.2	Perform	38
7.6.3.3	Perform	38
7.6.4	Member Data Documentation	39
7.6.4.1	C2_	39
7.6.4.2	Register_	39
7.7	DSG::BLIT::BlitSquare Class Reference	39
7.7.1	Detailed Description	39
7.8	DSG::BLIT::BlitTriangle Class Reference	39

7.8.1	Detailed Description	40
7.9	DSG::Buffer Class Reference	40
7.9.1	Detailed Description	40
7.9.2	Constructor & Destructor Documentation	40
7.9.2.1	Buffer	40
7.9.2.2	Buffer	41
7.9.2.3	Buffer	41
7.9.2.4	~Buffer	41
7.9.3	Member Function Documentation	41
7.9.3.1	operator=	41
7.9.3.2	operator[]	41
7.9.3.3	Size	41
7.9.4	Member Data Documentation	42
7.9.4.1	_buffer	42
7.9.4.2	_size	42
7.10	DSG::Filter::DCBlocker Class Reference	42
7.10.1	Detailed Description	42
7.10.2	Constructor & Destructor Documentation	43
7.10.2.1	DCBlocker	43
7.10.2.2	~DCBlocker	43
7.10.3	Member Function Documentation	43
7.10.3.1	Perform	43
7.10.3.2	Perform	43
7.10.4	Member Data Documentation	43
7.10.4.1	_a	43
7.10.4.2	_temp	43
7.10.4.3	count	44
7.10.4.4	x	44
7.10.4.5	xm1	44
7.10.4.6	ym1	44
7.11	DSG::Delay< maxLength > Class Template Reference	44
7.11.1	Detailed Description	45
7.11.2	Constructor & Destructor Documentation	45
7.11.2.1	Delay	45
7.11.2.2	Delay	45
7.11.2.3	~Delay	45
7.11.3	Member Function Documentation	46
7.11.3.1	increment	46
7.11.3.2	Length	46
7.11.3.3	Length	46

7.11.3.4	Perform	46
7.11.3.5	Perform	46
7.11.4	Member Data Documentation	47
7.11.4.1	_buffer	47
7.11.4.2	_delay	47
7.11.4.3	_index	47
7.11.4.4	_max	47
7.11.4.5	_swap	47
7.11.4.6	_temp	47
7.11.4.7	count	47
7.12	DSG::DPW::DPW_Differentiator< order > Class Template Reference	47
7.12.1	Detailed Description	48
7.12.2	Constructor & Destructor Documentation	48
7.12.2.1	DPW_Differentiator	48
7.13	DSG::DPW::DPW_Differentiator< 1 > Class Template Reference	48
7.13.1	Detailed Description	48
7.13.2	Member Function Documentation	48
7.13.2.1	operator()	48
7.14	DSG::DPW::DPW_Differentiator< 2 > Class Template Reference	49
7.14.1	Detailed Description	49
7.14.2	Member Function Documentation	49
7.14.2.1	operator()	49
7.14.3	Member Data Documentation	49
7.14.3.1	_delay	49
7.14.3.2	output	49
7.15	DSG::DPW::DPW_Differentiator< 3 > Class Template Reference	50
7.15.1	Detailed Description	50
7.15.2	Member Function Documentation	50
7.15.2.1	operator()	50
7.15.3	Member Data Documentation	50
7.15.3.1	_delay	50
7.15.3.2	output	50
7.16	DSG::DPW::DPW_Differentiator< 4 > Class Template Reference	51
7.16.1	Detailed Description	51
7.16.2	Member Function Documentation	51
7.16.2.1	operator()	51
7.16.3	Member Data Documentation	51
7.16.3.1	_delay	51
7.16.3.2	output	51
7.17	DSG::DPW::DPW_Differentiator< 5 > Class Template Reference	52



7.17.1 Detailed Description . . . . .	52
7.17.2 Member Function Documentation . . . . .	52
7.17.2.1 operator() . . . . .	52
7.17.3 Member Data Documentation . . . . .	52
7.17.3.1 _delay . . . . .	52
7.17.3.2 output . . . . .	52
7.18 DSG::DPW::DPW_Differentiator< 6 > Class Template Reference . . . . .	53
7.18.1 Detailed Description . . . . .	53
7.18.2 Member Function Documentation . . . . .	53
7.18.2.1 operator() . . . . .	53
7.18.3 Member Data Documentation . . . . .	53
7.18.3.1 _delay . . . . .	53
7.18.3.2 output . . . . .	53
7.19 DSG::DPW::DPWSaw< order > Class Template Reference . . . . .	54
7.19.1 Detailed Description . . . . .	54
7.19.2 Constructor & Destructor Documentation . . . . .	54
7.19.2.1 DPWSaw . . . . .	54
7.19.2.2 DPWSaw . . . . .	55
7.19.2.3 ~DPWSaw . . . . .	55
7.19.3 Member Function Documentation . . . . .	55
7.19.3.1 Perform . . . . .	55
7.19.3.2 Perform . . . . .	55
7.19.4 Member Data Documentation . . . . .	55
7.19.4.1 _diff . . . . .	55
7.19.4.2 _register . . . . .	56
7.20 DSG::EPTR::EPTRsaw Class Reference . . . . .	56
7.20.1 Detailed Description . . . . .	56
7.20.2 Constructor & Destructor Documentation . . . . .	56
7.20.2.1 EPTRsaw . . . . .	56
7.20.2.2 EPTRsaw . . . . .	57
7.20.2.3 ~EPTRsaw . . . . .	57
7.20.3 Member Function Documentation . . . . .	57
7.20.3.1 Perform . . . . .	57
7.20.3.2 Perform . . . . .	57
7.20.4 Member Data Documentation . . . . .	57
7.20.4.1 _register . . . . .	57
7.21 DSG::Factorial< N > Struct Template Reference . . . . .	58
7.21.1 Detailed Description . . . . .	58
7.21.2 Member Enumeration Documentation . . . . .	58
7.21.2.1 anonymous enum . . . . .	58

7.22 DSG::Factorial< 0 > Struct Template Reference . . . . .	58
7.22.1 Detailed Description . . . . .	58
7.22.2 Member Enumeration Documentation . . . . .	59
7.22.2.1 anonymous enum . . . . .	59
7.23 DSG::Filter::FilterBase Class Reference . . . . .	59
7.23.1 Detailed Description . . . . .	59
7.23.2 Constructor & Destructor Documentation . . . . .	60
7.23.2.1 FilterBase . . . . .	60
7.23.2.2 ~FilterBase . . . . .	60
7.23.3 Member Function Documentation . . . . .	60
7.23.3.1 Cutoff . . . . .	60
7.23.3.2 Perform . . . . .	60
7.23.3.3 Perform . . . . .	60
7.23.4 Member Data Documentation . . . . .	61
7.23.4.1 _temp . . . . .	61
7.23.4.2 count . . . . .	61
7.24 DSG::Fourier::FourierSaw Class Reference . . . . .	61
7.24.1 Detailed Description . . . . .	61
7.24.2 Constructor & Destructor Documentation . . . . .	62
7.24.2.1 FourierSaw . . . . .	62
7.24.2.2 FourierSaw . . . . .	62
7.24.2.3 ~FourierSaw . . . . .	62
7.24.3 Member Function Documentation . . . . .	62
7.24.3.1 Frequency . . . . .	62
7.24.3.2 Perform . . . . .	62
7.24.3.3 Perform . . . . .	63
7.24.4 Member Data Documentation . . . . .	63
7.24.4.1 _a . . . . .	63
7.24.4.2 _h . . . . .	63
7.24.4.3 i . . . . .	63
7.24.4.4 phs . . . . .	63
7.24.4.5 value . . . . .	63
7.25 DSG::Fourier::FourierSeriesGenerator Class Reference . . . . .	63
7.25.1 Detailed Description . . . . .	64
7.25.2 Member Typedef Documentation . . . . .	64
7.25.2.1 FourierSeries . . . . .	64
7.25.3 Constructor & Destructor Documentation . . . . .	64
7.25.3.1 FourierSeriesGenerator . . . . .	64
7.25.3.2 FourierSeriesGenerator . . . . .	65
7.25.3.3 ~FourierSeriesGenerator . . . . .	65

7.25.4	Member Function Documentation	65
7.25.4.1	Perform	65
7.25.4.2	Perform	65
7.25.4.3	Series	65
7.25.4.4	Series	66
7.25.5	Member Data Documentation	66
7.25.5.1	_series	66
7.25.5.2	value	66
7.26	DSG::Fourier::FourierSquare Class Reference	66
7.26.1	Detailed Description	67
7.26.2	Constructor & Destructor Documentation	67
7.26.2.1	FourierSquare	67
7.26.2.2	FourierSquare	67
7.26.2.3	~FourierSquare	67
7.26.3	Member Function Documentation	67
7.26.3.1	Frequency	67
7.26.3.2	Perform	68
7.26.3.3	Perform	68
7.26.4	Member Data Documentation	68
7.26.4.1	_a	68
7.26.4.2	_h	68
7.26.4.3	i	68
7.26.4.4	phs	68
7.26.4.5	value	68
7.27	DSG::Fourier::FourierTriangle Class Reference	69
7.27.1	Detailed Description	69
7.27.2	Constructor & Destructor Documentation	69
7.27.2.1	FourierTriangle	69
7.27.2.2	FourierTriangle	70
7.27.2.3	~FourierTriangle	70
7.27.3	Member Function Documentation	70
7.27.3.1	Frequency	70
7.27.3.2	Perform	70
7.27.3.3	Perform	70
7.27.4	Member Data Documentation	71
7.27.4.1	_a	71
7.27.4.2	_h	71
7.27.4.3	i	71
7.27.4.4	phs	71
7.27.4.5	value	71

7.28 DSG::GenericGenerator Class Reference . . . . .	71
7.28.1 Detailed Description . . . . .	72
7.28.2 Constructor & Destructor Documentation . . . . .	72
7.28.2.1 GenericGenerator . . . . .	72
7.28.2.2 GenericGenerator . . . . .	72
7.28.2.3 ~GenericGenerator . . . . .	72
7.28.3 Member Function Documentation . . . . .	72
7.28.3.1 Perform . . . . .	72
7.28.3.2 Perform . . . . .	73
7.28.4 Member Data Documentation . . . . .	73
7.28.4.1 _callback . . . . .	73
7.29 DSG::Fourier::Harmonic Class Reference . . . . .	73
7.29.1 Detailed Description . . . . .	73
7.29.2 Constructor & Destructor Documentation . . . . .	74
7.29.2.1 Harmonic . . . . .	74
7.29.2.2 Harmonic . . . . .	74
7.29.2.3 ~Harmonic . . . . .	74
7.29.3 Member Function Documentation . . . . .	74
7.29.3.1 Amplitude . . . . .	74
7.29.3.2 Amplitude . . . . .	74
7.29.3.3 Ratio . . . . .	74
7.29.3.4 Ratio . . . . .	74
7.29.4 Member Data Documentation . . . . .	75
7.29.4.1 _amplitude . . . . .	75
7.29.4.2 _ratio . . . . .	75
7.30 DSG::Filter::LeakyIntegrator Class Reference . . . . .	75
7.30.1 Detailed Description . . . . .	75
7.30.2 Constructor & Destructor Documentation . . . . .	76
7.30.2.1 LeakyIntegrator . . . . .	76
7.30.2.2 LeakyIntegrator . . . . .	76
7.30.2.3 ~LeakyIntegrator . . . . .	76
7.30.3 Member Function Documentation . . . . .	76
7.30.3.1 Cutoff . . . . .	76
7.30.3.2 Perform . . . . .	76
7.30.3.3 Perform . . . . .	77
7.30.4 Member Data Documentation . . . . .	77
7.30.4.1 a . . . . .	77
7.30.4.2 b . . . . .	77
7.30.4.3 x1 . . . . .	77
7.30.4.4 y . . . . .	77

7.30.4.5	y1	77
7.31	DSG::LUT< element, size > Class Template Reference	77
7.31.1	Detailed Description	78
7.31.2	Member Typedef Documentation	78
7.31.2.1	FillFunction	78
7.31.2.2	FillFunctionConstRef	78
7.31.3	Constructor & Destructor Documentation	78
7.31.3.1	LUT	78
7.31.3.2	LUT	79
7.31.3.3	LUT	79
7.31.3.4	~LUT	79
7.31.4	Member Function Documentation	79
7.31.4.1	operator()	79
7.31.4.2	operator[]	79
7.31.4.3	operator[]	80
7.31.4.4	Size	80
7.31.5	Member Data Documentation	80
7.31.5.1	_size	80
7.31.5.2	_table	80
7.31.5.3	phs	80
7.32	DSG::NoiseGenerator Class Reference	80
7.32.1	Detailed Description	81
7.32.2	Constructor & Destructor Documentation	81
7.32.2.1	NoiseGenerator	81
7.32.2.2	~NoiseGenerator	81
7.32.3	Member Function Documentation	81
7.32.3.1	Perform	81
7.32.3.2	Perform	82
7.32.4	Member Data Documentation	82
7.32.4.1	_function	82
7.32.4.2	_storage	82
7.33	DSG::RingBuffer Class Reference	82
7.33.1	Detailed Description	83
7.33.2	Constructor & Destructor Documentation	83
7.33.2.1	RingBuffer	83
7.33.2.2	RingBuffer	83
7.33.2.3	RingBuffer	83
7.33.2.4	~RingBuffer	84
7.33.3	Member Function Documentation	84
7.33.3.1	Count	84

7.33.3.2	Empty	84
7.33.3.3	Flush	84
7.33.3.4	Full	84
7.33.3.5	make_pow_2	84
7.33.3.6	next	84
7.33.3.7	operator=	85
7.33.3.8	Read	85
7.33.3.9	Write	85
7.33.4	Friends And Related Function Documentation	85
7.33.4.1	operator<<	85
7.33.4.2	operator>>	85
7.33.5	Member Data Documentation	85
7.33.5.1	_count	85
7.33.5.2	_read	86
7.33.5.3	_write	86
7.33.5.4	MASK	86
7.33.5.5	read	86
7.33.5.6	write	86
7.34	DSG::SignalGenerator Class Reference	86
7.34.1	Detailed Description	88
7.34.2	Constructor & Destructor Documentation	88
7.34.2.1	SignalGenerator	88
7.34.2.2	SignalGenerator	88
7.34.2.3	~SignalGenerator	88
7.34.3	Member Function Documentation	88
7.34.3.1	Frequency	88
7.34.3.2	Frequency	88
7.34.3.3	Perform	89
7.34.3.4	Perform	89
7.34.3.5	Phase	89
7.34.3.6	Phase	89
7.34.3.7	step	89
7.34.3.8	sync	90
7.34.4	Member Data Documentation	90
7.34.4.1	_dt	90
7.34.4.2	_frequency	90
7.34.4.3	_offset	90
7.34.4.4	_phasor	90
7.34.4.5	_storage	90
7.35	DSG::SignalProcess Class Reference	90

7.35.1 Detailed Description . . . . .	91
7.35.2 Constructor & Destructor Documentation . . . . .	91
7.35.2.1 SignalProcess . . . . .	91
7.35.2.2 ~SignalProcess . . . . .	91
7.35.3 Member Function Documentation . . . . .	91
7.35.3.1 Perform . . . . .	91
7.35.3.2 Perform . . . . .	92
<b>8 File Documentation</b> . . . . .	<b>93</b>
8.1 AnalogSaw.cpp File Reference . . . . .	93
8.2 AnalogSaw.cpp . . . . .	93
8.3 AnalogSaw.h File Reference . . . . .	93
8.4 AnalogSaw.h . . . . .	94
8.5 AnalogSquare.cpp File Reference . . . . .	95
8.6 AnalogSquare.cpp . . . . .	95
8.7 AnalogSquare.h File Reference . . . . .	95
8.8 AnalogSquare.h . . . . .	95
8.9 AnalogTriangle.cpp File Reference . . . . .	96
8.10 AnalogTriangle.cpp . . . . .	96
8.11 AnalogTriangle.h File Reference . . . . .	97
8.12 AnalogTriangle.h . . . . .	97
8.13 AudioSettings.cpp File Reference . . . . .	98
8.14 AudioSettings.cpp . . . . .	98
8.15 AudioSettings.h File Reference . . . . .	99
8.16 AudioSettings.h . . . . .	99
8.17 Blackman.h File Reference . . . . .	100
8.18 Blackman.h . . . . .	100
8.19 BLIT.cpp File Reference . . . . .	101
8.20 BLIT.cpp . . . . .	101
8.21 BLIT.h File Reference . . . . .	102
8.22 BLIT.h . . . . .	102
8.23 BLITSaw.cpp File Reference . . . . .	103
8.24 BLITSaw.cpp . . . . .	103
8.25 BLITSaw.h File Reference . . . . .	104
8.26 BLITSaw.h . . . . .	104
8.27 BLITSquare.cpp File Reference . . . . .	105
8.28 BLITSquare.cpp . . . . .	105
8.29 BLITSquare.h File Reference . . . . .	106
8.30 BLITSquare.h . . . . .	106
8.31 BLITTriangle.cpp File Reference . . . . .	107

8.32	BLITTriangle.cpp	107
8.33	BLITTriangle.h File Reference	107
8.34	BLITTriangle.h	107
8.35	Bounds.h File Reference	108
8.36	Bounds.h	108
8.37	Buffer.cpp File Reference	109
8.38	Buffer.cpp	109
8.39	Buffer.h File Reference	110
8.40	Buffer.h	110
8.41	BufferConversion.h File Reference	111
8.42	BufferConversion.h	111
8.43	DCBlocker.cpp File Reference	112
8.44	DCBlocker.cpp	112
8.45	DCBlocker.h File Reference	112
8.46	DCBlocker.h	113
8.47	Delay.h File Reference	114
8.48	Delay.h	114
8.49	Denormal.h File Reference	115
8.50	Denormal.h	116
8.51	DPW.h File Reference	116
8.52	DPW.h	117
8.53	DPWSaw.h File Reference	120
8.54	DPWSaw.h	120
8.55	Driver.cpp File Reference	121
8.55.1	Macro Definition Documentation	121
8.55.1.1	BufferSize	121
8.55.2	Function Documentation	121
8.55.2.1	Callback	121
8.55.2.2	DriverExit	122
8.55.2.3	DriverInit	122
8.55.3	Variable Documentation	123
8.55.3.1	_buffer	123
8.55.3.2	stream	123
8.56	Driver.cpp	123
8.57	Driver.h File Reference	124
8.57.1	Function Documentation	124
8.57.1.1	Callback	124
8.57.1.2	DriverExit	124
8.57.1.3	DriverInit	125
8.58	Driver.h	125



8.59 DSF.h File Reference . . . . .	125
8.60 DSF.h . . . . .	126
8.61 DSG.h File Reference . . . . .	127
8.61.1 Macro Definition Documentation . . . . .	127
8.61.1.1 DSG_Short_Names . . . . .	127
8.62 DSG.h . . . . .	128
8.63 DSGMath.h File Reference . . . . .	129
8.64 DSGMath.h . . . . .	129
8.65 DSGTypes.h File Reference . . . . .	130
8.66 DSGTypes.h . . . . .	130
8.67 EPTRSaw.cpp File Reference . . . . .	131
8.68 EPTRSaw.cpp . . . . .	131
8.69 EPTRSaw.h File Reference . . . . .	131
8.70 EPTRSaw.h . . . . .	132
8.71 Filter.cpp File Reference . . . . .	133
8.72 Filter.cpp . . . . .	133
8.73 Filter.h File Reference . . . . .	133
8.74 Filter.h . . . . .	133
8.75 FourierSaw.cpp File Reference . . . . .	134
8.76 FourierSaw.cpp . . . . .	134
8.77 FourierSaw.h File Reference . . . . .	135
8.78 FourierSaw.h . . . . .	135
8.79 FourierSeries.cpp File Reference . . . . .	136
8.80 FourierSeries.cpp . . . . .	136
8.81 FourierSeries.h File Reference . . . . .	137
8.82 FourierSeries.h . . . . .	138
8.83 FourierSquare.cpp File Reference . . . . .	139
8.84 FourierSquare.cpp . . . . .	139
8.85 FourierSquare.h File Reference . . . . .	139
8.86 FourierSquare.h . . . . .	140
8.87 FourierTriangle.cpp File Reference . . . . .	141
8.88 FourierTriangle.cpp . . . . .	141
8.89 FourierTriangle.h File Reference . . . . .	141
8.90 FourierTriangle.h . . . . .	142
8.91 Gaussian.h File Reference . . . . .	143
8.92 Gaussian.h . . . . .	143
8.93 GenericGenerator.cpp File Reference . . . . .	144
8.94 GenericGenerator.cpp . . . . .	144
8.95 GenericGenerator.h File Reference . . . . .	144
8.96 GenericGenerator.h . . . . .	145

8.97 Interpolate.h File Reference . . . . .	145
8.98 Interpolate.h . . . . .	146
8.99 Leaky.cpp File Reference . . . . .	147
8.100Leaky.cpp . . . . .	147
8.101Leaky.h File Reference . . . . .	148
8.102Leaky.h . . . . .	148
8.103LUT.h File Reference . . . . .	149
8.104LUT.h . . . . .	150
8.105Noise.h File Reference . . . . .	151
8.106Noise.h . . . . .	151
8.107NoiseGenerator.cpp File Reference . . . . .	151
8.108NoiseGenerator.cpp . . . . .	151
8.109NoiseGenerator.h File Reference . . . . .	152
8.110NoiseGenerator.h . . . . .	152
8.111PI.h File Reference . . . . .	153
8.111.1 Macro Definition Documentation . . . . .	153
8.111.1.1 PI . . . . .	153
8.111.1.2 TWOPI . . . . .	153
8.112PI.h . . . . .	153
8.113Pink.h File Reference . . . . .	154
8.114Pink.h . . . . .	154
8.115Random.h File Reference . . . . .	155
8.115.1 Variable Documentation . . . . .	155
8.115.1.1 max . . . . .	155
8.116Random.h . . . . .	155
8.117RingBuffer.cpp File Reference . . . . .	156
8.118RingBuffer.cpp . . . . .	156
8.119RingBuffer.h File Reference . . . . .	157
8.120RingBuffer.h . . . . .	157
8.121SignalGenerator.cpp File Reference . . . . .	159
8.122SignalGenerator.cpp . . . . .	159
8.123SignalGenerator.h File Reference . . . . .	159
8.124SignalGenerator.h . . . . .	160
8.125SignalProcess.cpp File Reference . . . . .	161
8.126SignalProcess.cpp . . . . .	161
8.127SignalProcess.h File Reference . . . . .	162
8.128SignalProcess.h . . . . .	162
8.129Sinc.h File Reference . . . . .	162
8.130Sinc.h . . . . .	163
8.131Sine.h File Reference . . . . .	163

---

8.131.1 Macro Definition Documentation . . . . .	164
8.131.1.1 LUT_SIZE . . . . .	164
8.132Sine.h . . . . .	164
8.133Sleep.h File Reference . . . . .	165
8.134Sleep.h . . . . .	166
8.135White.h File Reference . . . . .	166
8.136White.h . . . . .	166
8.137Window.h File Reference . . . . .	167
8.138Window.h . . . . .	167
<b>Index</b>	<b>169</b>



# Chapter 1

## Todo List

**Class** `DSG::BLIT::Blit`

Re-write `DSG::BLIT::Blit` algorithm

**Class** `DSG::BLIT::BlitSaw`

Re-write `DSG::BLIT::BlitSaw` algorithm

**Class** `DSG::BLIT::BlitSquare`

Write `DSG::BLIT::BlitSquare` algorithm

**Class** `DSG::BLIT::BlitTriangle`

Write `DSG::BLIT::BlitTriangle` algorithm

**Class** `DSG::DPW::DPW_Differentiator< order >`

Fix `DSG::DPW::DPW_Differentiator` algorithms for orders 3-6

**Class** `DSG::EPTR::EPTRSaw`

Test and Possibly Re-Write `DSG::EPTR::EPTRSaw` algorithm



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">DSG</a>	
<a href="#">DSG</a> - A Collection of tools for Digital Signal Generation . . . . .	11
<a href="#">DSG::Analog</a>	
<a href="#">DSG::Analog</a> - Namespace Containing <a href="#">Analog</a> Style Oscillators . . . . .	18
<a href="#">DSG::BLIT</a>	
<a href="#">DSG::BLIT</a> - Namespace Containing <a href="#">BLIT</a> Based Oscillators . . . . .	19
<a href="#">DSG::DPW</a>	
<a href="#">DSG::DPW</a> - Generators using the <a href="#">DPW</a> method . . . . .	19
<a href="#">DSG::EPTR</a>	
<a href="#">DSG::EPTR</a> - Generators Based On The Efficient Polynomial Transfer Region Algorithm . . . . .	22
<a href="#">DSG::Filter</a>	
<a href="#">DSG::Filter</a> - Filters . . . . .	22
<a href="#">DSG::Fourier</a>	
<a href="#">DSG::Fourier</a> - Namespace Containing <a href="#">Fourier</a> Series Based Oscillators . . . . .	22
<a href="#">DSG::Noise</a>	
<a href="#">DSG::Noise</a> - <a href="#">Noise</a> Generators . . . . .	23
<a href="#">DSG::Window</a>	
<a href="#">DSG::Window</a> - <a href="#">Window</a> functions and utilities . . . . .	24





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

DSG::AudioSettings . . . . .	32
DSG::Buffer . . . . .	40
DSG::RingBuffer . . . . .	82
DSG::DPW::DPW_Differentiator< order > . . . . .	47
DSG::DPW::DPW_Differentiator< 1 > . . . . .	48
DSG::DPW::DPW_Differentiator< 2 > . . . . .	49
DSG::DPW::DPW_Differentiator< 3 > . . . . .	50
DSG::DPW::DPW_Differentiator< 4 > . . . . .	51
DSG::DPW::DPW_Differentiator< 5 > . . . . .	52
DSG::DPW::DPW_Differentiator< 6 > . . . . .	53
DSG::Factorial< N > . . . . .	58
DSG::Factorial< 0 > . . . . .	58
DSG::Fourier::Harmonic . . . . .	73
DSG::LUT< element, size > . . . . .	77
DSG::SignalProcess . . . . .	90
DSG::Delay< maxLength > . . . . .	44
DSG::Filter::FilterBase . . . . .	59
DSG::Filter::DCBlocker . . . . .	42
DSG::Filter::LeakyIntegrator . . . . .	75
DSG::NoiseGenerator . . . . .	80
DSG::SignalGenerator . . . . .	86
DSG::Analog::AnalogSaw . . . . .	27
DSG::Analog::AnalogSquare . . . . .	29
DSG::Analog::AnalogTriangle . . . . .	30
DSG::BLIT::Blit . . . . .	34
DSG::BLIT::BlitSaw . . . . .	36
DSG::BLIT::BlitSquare . . . . .	39
DSG::BLIT::BlitTriangle . . . . .	39
DSG::DPW::DPWSaw< order > . . . . .	54
DSG::EPTR::EPTRSaw . . . . .	56
DSG::Fourier::FourierSaw . . . . .	61
DSG::Fourier::FourierSeriesGenerator . . . . .	63
DSG::Fourier::FourierSquare . . . . .	66
DSG::Fourier::FourierTriangle . . . . .	69
DSG::GenericGenerator . . . . .	71



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">DSG::Analog::AnalogSaw</a>	
<a href="#">DSG::Analog::AnalogSaw</a> - Analog Syle Saw Wave Generator . . . . .	27
<a href="#">DSG::Analog::AnalogSquare</a>	
<a href="#">DSG::Analog::AnalogSquare</a> - Analog Syle Square Wave Generator . . . . .	29
<a href="#">DSG::Analog::AnalogTriangle</a>	
<a href="#">DSG::Analog::AnalogTriangle</a> - Analog Syle Triangle Wave Generator . . . . .	30
<a href="#">DSG::AudioSettings</a>	
<a href="#">DSG::AudioSettings</a> - Global Storage For Audio Settings Such As Sample Rate . . . . .	32
<a href="#">DSG::BLIT::Blit</a>	
<a href="#">DSG::BLIT::Blit</a> - Band-Limited Impulse Train Generator . . . . .	34
<a href="#">DSG::BLIT::BlitSaw</a>	
<a href="#">DSG::BLIT::BlitSaw</a> - Saw Wave Generator Based on <a href="#">BLIT</a> Algorithm . . . . .	36
<a href="#">DSG::BLIT::BlitSquare</a> . . . . .	39
<a href="#">DSG::BLIT::BlitTriangle</a> . . . . .	39
<a href="#">DSG::Buffer</a>	
<a href="#">DSG::Buffer</a> - Base Class For <a href="#">DSG::RingBuffer</a> . Not For Direct Use . . . . .	40
<a href="#">DSG::Filter::DCBlocker</a>	
<a href="#">DSG::Filter::DCBlocker</a> - DC blocking filter . . . . .	42
<a href="#">DSG::Delay&lt; maxLength &gt;</a>	
<a href="#">DSG::Delay</a> - General purpose delay line . . . . .	44
<a href="#">DSG::DPW::DPW_Differentiator&lt; order &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the <a href="#">DPW</a> Algorithm . . .	47
<a href="#">DSG::DPW::DPW_Differentiator&lt; 1 &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the 1st order <a href="#">DPW</a> Algo- rithm . . . . .	48
<a href="#">DSG::DPW::DPW_Differentiator&lt; 2 &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the 2nd order <a href="#">DPW</a> Algo- rithm . . . . .	49
<a href="#">DSG::DPW::DPW_Differentiator&lt; 3 &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the 3rd order <a href="#">DPW</a> Algo- rithm . . . . .	50
<a href="#">DSG::DPW::DPW_Differentiator&lt; 4 &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the 4th order <a href="#">DPW</a> Algo- rithm . . . . .	51
<a href="#">DSG::DPW::DPW_Differentiator&lt; 5 &gt;</a>	
<a href="#">DSG::DPW::DPW_Differentiator</a> - Class Performing Differentiation for the 5th order <a href="#">DPW</a> Algo- rithm . . . . .	52

DSG::DPW::DPW_Differentiator< 6 >	
DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order <a href="#">DPW</a> Algorithm	53
DSG::DPW::DPWSaw< order >	
DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order <a href="#">DPW</a> algorithm	54
DSG::EPTR::EPTRSaw	
DSG::EPTR::EPTRSaw - Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm	56
DSG::Factorial< N >	
DSG::Factorial - Compute integer factorial	58
DSG::Factorial< 0 >	
DSG::Factorial - Compute integer factorial	58
DSG::Filter::FilterBase	
DSG::Filter::FilterBase - <a href="#">Filter</a> Base Class, implements interface for cutoff frequency	59
DSG::Fourier::FourierSaw	
DSG::Fourier::FourierSaw - <a href="#">Fourier</a> Series Sawtooth Wave Generator	61
DSG::Fourier::FourierSeriesGenerator	
DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified <a href="#">Fourier</a> Series	63
DSG::Fourier::FourierSquare	
DSG::Fourier::FourierSquare - <a href="#">Fourier</a> Series Square Wave Generator	66
DSG::Fourier::FourierTriangle	
DSG::Fourier::FourierTriangle - <a href="#">Fourier</a> Series Triangle Wave Generator	69
DSG::GenericGenerator	
DSG::GenericGenerator - Generator designed to use a stateless generator function such as <a href="#">DSG::Sin()</a>	71
DSG::Fourier::Harmonic	
DSG::Fourier::Harmonic - Represents a single harmonic in a <a href="#">Fourier</a> Series	73
DSG::Filter::LeakyIntegrator	
DSG::Filter::LeakyIntegrator - Leaky integrator	75
DSG::LUT< element, size >	
DSG::LUT - Look Up Table	77
DSG::NoiseGenerator	
DSG::NoiseGenerator - Generator that uses noise functions such as <a href="#">DSG::White()</a> to generate signal	80
DSG::RingBuffer	
DSG::RingBuffer - Circular <a href="#">Buffer</a> of Audio	82
DSG::SignalGenerator	
DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation	86
DSG::SignalProcess	
DSG::SignalProcess - Defines Base Interface For Audio Processing	90

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

AnalogSaw.cpp	93
AnalogSaw.h	93
AnalogSquare.cpp	95
AnalogSquare.h	95
AnalogTriangle.cpp	96
AnalogTriangle.h	97
AudioSettings.cpp	98
AudioSettings.h	99
Blackman.h	100
BLIT.cpp	101
BLIT.h	102
BLITSaw.cpp	103
BLITSaw.h	104
BLITSquare.cpp	105
BLITSquare.h	106
BLITTriangle.cpp	107
BLITTriangle.h	107
Bounds.h	108
Buffer.cpp	109
Buffer.h	110
BufferConversion.h	111
DCBlocker.cpp	112
DCBlocker.h	112
Delay.h	114
Denormal.h	115
DPW.h	116
DPWSaw.h	120
Driver.cpp	121
Driver.h	124
DSF.h	125
DSG.h	127
DSGMath.h	129
DSGTypes.h	130
EPTRSaw.cpp	131
EPTRSaw.h	131
Filter.cpp	133
Filter.h	133
FourierSaw.cpp	134

FourierSaw.h	135
FourierSeries.cpp	136
FourierSeries.h	137
FourierSquare.cpp	139
FourierSquare.h	139
FourierTriangle.cpp	141
FourierTriangle.h	141
Gaussian.h	143
GenericGenerator.cpp	144
GenericGenerator.h	144
Interpolate.h	145
Leaky.cpp	147
Leaky.h	148
LUT.h	149
Noise.h	151
NoiseGenerator.cpp	151
NoiseGenerator.h	152
PI.h	153
Pink.h	154
Random.h	155
RingBuffer.cpp	156
RingBuffer.h	157
SignalGenerator.cpp	159
SignalGenerator.h	159
SignalProcess.cpp	161
SignalProcess.h	162
Sinc.h	162
Sine.h	163
Sleep.h	165
White.h	166
Window.h	167

## Chapter 6

# Namespace Documentation

### 6.1 DSG Namespace Reference

**DSG** - A Collection of tools for Digital Signal Generation.

#### Namespaces

- **Analog**  
*DSG::Analog* - Namespace Containing **Analog** Style Oscillators.
- **BLIT**  
*DSG::BLIT* - Namespace Containing **BLIT** Based Oscillators.
- **DPW**  
*DSG::DPW* - Generators using the **DPW** method.
- **EPTR**  
*DSG::EPTR* - Generators Based On The Efficient Polynomial Transfer Region Algorithm.
- **Filter**  
*DSG::Filter* - Filters.
- **Fourier**  
*DSG::Fourier* - Namespace Containing **Fourier** Series Based Oscillators.
- **Noise**  
*DSG::Noise* - **Noise** Generators.
- **Window**  
*DSG::Window* - **Window** functions and utilities.

#### Classes

- class **AudioSettings**  
*DSG::AudioSettings* - Global Storage For Audio Settings Such As Sample Rate.
- class **Buffer**  
*DSG::Buffer* - Base Class For *DSG::RingBuffer*. Not For Direct Use.
- class **Delay**  
*DSG::Delay* - General purpose delay line.
- struct **Factorial**  
*DSG::Factorial* - Compute integer factorial.
- struct **Factorial< 0 >**  
*DSG::Factorial* - Compute integer factorial.
- class **GenericGenerator**

- [\*DSG::GenericGenerator\*](#) - Generator designed to use a stateless generator function such as [\*DSG::Sin\(\)\*](#)
- class [LUT](#)  
[\*DSG::LUT\*](#) - Look Up Table.
- class [NoiseGenerator](#)  
[\*DSG::NoiseGenerator\*](#) - Generator that uses noise functions such as [\*DSG::White\(\)\*](#) to generate signal.
- class [RingBuffer](#)  
[\*DSG::RingBuffer\*](#) - Circular [Buffer](#) of Audio.
- class [SignalGenerator](#)  
[\*DSG::SignalGenerator\*](#) - Extends [\*DSG::Signal Process With Tools For Signal Generation\*](#).
- class [SignalProcess](#)  
[\*DSG::SignalProcess\*](#) - Defines Base Interface For Audio Processing.

## Typedefs

- typedef float [DSGFrequency](#)  
[\*DSG::DSGFrequency\*](#) - Type for representing a frequency value.
- typedef float [DSGPhase](#)  
[\*DSG::DSGPhase\*](#) - Type for representing a phase value.
- typedef float [DSGSample](#)  
[\*DSG::DSGSample\*](#) - Type for representing an audio sample.

## Functions

- [DSG::DSGFrequency](#) const & [SampleRate](#) ()  
[\*DSG::SampleRate\*](#) - Get Global Sample Rate.
- [DSG::DSGFrequency](#) const & [SampleRate](#) ([DSG::DSGFrequency](#) const &value)  
[\*DSG::SampleRate\*](#) - Set Global Sample Rate.
- [DSG::DSGFrequency](#) Nyquist ()  
[\*DSG::Nyquist\(\)\*](#) - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.
- template<int lower, int upper, typename decimal >  
decimal [EnforceBounds](#) (decimal const &value)  
[\*DSG::EnforceBounds\*](#) - Clip value to set bounds.
- template<int lower, int upper, int value>  
void [StaticAssertBounds](#) ()  
[\*DSG::StaticAssertBounds\*](#) - Fails on compile time if value is not within bounds.
- template<int lower, int upper, typename T >  
void [AssertBounds](#) (T const &value)  
[\*DSG::AssertBounds\*](#) - Fails on runtime if value is not within bounds.
- bool [RingToArray](#) ([DSG::RingBuffer](#) &ring, [DSG::DSGSample](#) \*array, unsigned long length)  
[\*DSG::RingToArray\*](#) - Move Ring [Buffer](#) data to an array.
- bool [ArrayToRing](#) ([DSG::RingBuffer](#) &ring, [DSG::DSGSample](#) \*array, unsigned long length)  
[\*DSG::ArrayToRing\*](#) - Move array data to a Ring [Buffer](#).
- template<typename T >  
bool [IsDenormal](#) (T const &value)  
[\*DSG::IsDenormal\*](#) - Returns True if number is Denormal.
- template<typename decimal = [DSG::DSGSample](#)>  
decimal [DSF](#) (decimal const &beta, decimal const &theta, decimal const &N, decimal const &a)
- template<typename T >  
T [Abs](#) (T const &value)  
[\*DSG::Abs\*](#) - Calculate absolute value.



- `template<unsigned exponent, class T >`  
`T constexpr Pow (T const base)`  
*[DSG::Pow](#) - Any type to an integer power, i.e.  $N^I$ .*
- `template<typename decimal >`  
`decimal LinearInterpolate (decimal const &y1, decimal const &y2, decimal const &mu)`  
*[DSG::LinearInterpolate](#) - Linear Interpolation.*
- `template<typename decimal >`  
`decimal CosineInterpolate (decimal y1, decimal y2, decimal mu)`  
*[DSG::CosineInterpolate](#) - Cosine Interpolation.*
- `template<typename decimal >`  
`decimal CubicInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu)`  
*[DSG::CubicInterpolate](#) - Cubic Interpolation.*
- `template<typename decimal >`  
`decimal HermiteInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu, decimal const &tension, decimal const &bias)`  
*[DSG::HermiteInterpolate](#) - Hermite Interpolation.*
- `unsigned long MaxHarms (DSG::DSGFrequency const &frequency)`
- `template<typename decimal >`  
`decimal Sinc (decimal const &x)`  
*[DSG::Sinc](#) - Implements the [Sinc\(\)](#) function ( $\sin(\pi \cdot x) / \pi \cdot x$ )*
- `double Sin (double const &x)`  
*[DSG::Sin\(\)](#) - General Purpose Sin Function, double precision.*
- `float Sin (float const &x)`  
*[DSG::Sin\(\)](#) - General Purpose Sin Function, single precision.*
- `double Cos (double const &x)`  
*[DSG::Cos\(\)](#) - General Purpose Cos Function, double precision.*
- `float Cos (float const &x)`  
*[DSG::Cos\(\)](#) - General Purpose Cos Function, single precision.*
- `template<typename integer >`  
`void Sleep (integer const &milliseconds)`  
*[DSG::Sleep](#) - Millisecond Sleep Function.*

### 6.1.1 Detailed Description

[DSG](#) - A Collection of tools for Digital Signal Generation.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 `typedef float DSG::DSGFrequency`

[DSG::DSGFrequency](#) - Type for representing a frequency value.

Definition at line 28 of file [DSGTypes.h](#).

#### 6.1.2.2 `typedef float DSG::DSGPhase`

[DSG::DSGPhase](#) - Type for representing a phase value.

Definition at line 30 of file [DSGTypes.h](#).

### 6.1.2.3 typedef float DSG::DSGSample

[DSG::DSGSample](#) - Type for representing an audio sample.

Definition at line 32 of file [DSGTypes.h](#).

## 6.1.3 Function Documentation

### 6.1.3.1 template<typename T > T DSG::Abs ( T const & *value* ) [inline]

[DSG::Abs](#) - Calculate absolute value.

Definition at line 31 of file [DSGMath.h](#).

```
00031         {
00032         return value < 0.0 ? -1.0 * value : value;
00033     }
```

### 6.1.3.2 bool DSG::ArrayToRing ( DSG::RingBuffer & *ring*, DSG::DSGSample \* *array*, unsigned long *length* ) [inline]

[DSG::ArrayToRing](#) - Move array data to a Ring Buffer.

Definition at line 37 of file [BufferConversion.h](#).

```
00037                                     {
00038         int i=0;
00039         ring.Flush();
00040         while (!ring.Full()) {
00041             ring.Write(array[i]);
00042             ++i;
00043         }return true;
00044     }
```

### 6.1.3.3 template<int lower, int upper, typename T > void DSG::AssertBounds ( T const & *value* )

[DSG::AssertBounds](#) - Fails on runtime if value is not within bounds.

Definition at line 44 of file [Bounds.h](#).

```
00044         {
00045         assert(value>=lower && value<=upper);
00046     }
```

### 6.1.3.4 double DSG::Cos ( double const & *x* ) [inline]

[DSG::Cos\(\)](#) - General Purpose Cos Function, double precision.

Definition at line 76 of file [Sine.h](#).

```
00076         {
00077         return static_cast<double>(Cos<Sine_Default>(x)); //wrap default implementation as non template
00078     }
```

### 6.1.3.5 float DSG::Cos ( float const & *x* ) [inline]

[DSG::Cos\(\)](#) - General Purpose Cos Function, single precision.

Definition at line 81 of file [Sine.h](#).

```

00081         {
00082         return static_cast<float>(Cos<Sine_Default>(x));
00083     }

```

#### 6.1.3.6 template<typename decimal > decimal DSG::CosineInterpolate ( decimal *y1*, decimal *y2*, decimal *mu* )

[DSG::CosineInterpolate](#) - Cosine Interpolation.

Definition at line 39 of file [Interpolate.h](#).

```

00042     {
00043         decimal mu2;
00044         mu2 = (1-cos(mu*PI))/2.0;
00045         return (y1*(1-mu2)+y2*mu2);
00046     }

```

#### 6.1.3.7 template<typename decimal > decimal DSG::CubicInterpolate ( decimal const & *y0*, decimal const & *y1*, decimal const & *y2*, decimal const & *y3*, decimal const & *mu* )

[DSG::CubicInterpolate](#) - Cubic Interpolation.

Definition at line 49 of file [Interpolate.h](#).

```

00052     {
00053         decimal a0,a1,a2,a3,mu2;
00054         mu2 = mu*mu;
00055         a0 = y3 - y2 - y0 + y1;
00056         a1 = y0 - y1 - a0;
00057         a2 = y2 - y0;
00058         a3 = y1;
00059         return (a0*mu*mu2+a1*mu2+a2*mu+a3);
00060     }

```

#### 6.1.3.8 template<typename decimal = DSG::DSGSample> decimal DSG::DSF ( decimal const & *beta*, decimal const & *theta*, decimal const & *N*, decimal const & *a* )

Definition at line 30 of file [DSF.h](#).

```

00030                                     {
00031     #ifdef __APPLE__
00032     #warning Untested DSG::DSF()
00033     #endif
00034         decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
00035         decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a *
sin(theta + (N*beta)));
00036         return numerator/denominator;
00037     }

```

#### 6.1.3.9 template<int lower, int upper, typename decimal > decimal DSG::EnforceBounds ( decimal const & *value* )

[DSG::EnforceBounds](#) - Clip value to set bounds.

Definition at line 30 of file [Bounds.h](#).

```

00030                                     {
00031         if (value<lower) {
00032             return lower;
00033         }else if(value> upper){
00034             return upper;
00035         }else return value;
00036     }

```

**6.1.3.10** `template<typename decimal > decimal DSG::HermiteInterpolate ( decimal const & y0, decimal const & y1, decimal const & y2, decimal const & y3, decimal const & mu, decimal const & tension, decimal const & bias )`

[DSG::HermiteInterpolate](#) - Hermite Interpolation.

Definition at line 63 of file [Interpolate.h](#).

```
00068     {
00069         /*
00070          Tension: 1 is high, 0 normal, -1 is low
00071          Bias: 0 is even,
00072          positive is towards first segment,
00073          negative towards the other
00074          */
00075         decimal m0,m1,mu2,mu3;
00076         decimal a0,a1,a2,a3;
00077         mu2 = mu * mu;
00078         mu3 = mu2 * mu;
00079         m0 = (y1-y0)*(1+bias)*(1-tension)/2.0;
00080         m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
00081         m1 = (y2-y1)*(1+bias)*(1-tension)/2.0;
00082         m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
00083         a0 = 2*mu3 - 3*mu2 + 1;
00084         a1 = mu3 - 2*mu2 + mu;
00085         a2 = mu3 - mu2;
00086         a3 = -2*mu3 + 3*mu2;
00087         return (a0*y1+a1*m0+a2*m1+a3*y2);
00088     }
```

**6.1.3.11** `template<typename T > bool DSG::IsDenormal ( T const & value ) [inline]`

[DSG::IsDenormal](#) - Returns True if number is Denormal.

Definition at line 31 of file [Denormal.h](#).

```
00031     {
00032         return DSG::Abs (value) <= std::numeric_limits<T>::epsilon(); //return true if number is
00033         denormal
00034     }
```

**6.1.3.12** `template<typename decimal > decimal DSG::LinearInterpolate ( decimal const & y1, decimal const & y2, decimal const & mu )`

[DSG::LinearInterpolate](#) - Linear Interpolation.

Definition at line 34 of file [Interpolate.h](#).

```
00034     {
00035         return (y1*(1-mu)+y2*mu);
00036     }
```

**6.1.3.13** `unsigned long DSG::MaxHarms ( DSG::DSGFrequency const & frequency ) [inline]`

Definition at line 56 of file [SignalGenerator.h](#).

```
00056     {
00057         double _s = DSG::SampleRate() * 20000.0/DSG::SampleRate();
00058         _s/=frequency;
00059         return _s;
00060     }
```

**6.1.3.14 DSG::DSGFrequency DSG::Nyquist ( ) [inline]**

**DSG::Nyquist()** - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.

Definition at line 48 of file [AudioSettings.h](#).

```
00048                                     {
00049     return DSG::AudioSettings::Nyquist();
00050 }
```

**6.1.3.15 template<unsigned exponent, class T > T constexpr DSG::Pow ( T const base )**

**DSG::Pow** - Any type to an integer power, i.e.  $N^I$ .

Definition at line 60 of file [DSGMath.h](#).

```
00060                                     {
00061     return power<T, exponent>::value(base);
00062 }
```

**6.1.3.16 bool DSG::RingToArray ( DSG::RingBuffer & ring, DSG::DSGSample \* array, unsigned long length ) [inline]**

**DSG::RingToArray** - Move Ring Buffer data to an array.

Definition at line 29 of file [BufferConversion.h](#).

```
00029                                     {
00030     for (int i=0; i<length; ++i) {
00031         if (!ring.Empty()) {
00032             ring.Read(array[i]);
00033         }
00034     }return true;
00035 }
```

**6.1.3.17 DSG::DSGFrequency const& DSG::SampleRate ( ) [inline]**

**DSG::SampleRate** - Get Global Sample Rate.

Definition at line 40 of file [AudioSettings.h](#).

```
00040                                     {
00041     return DSG::AudioSettings::SampleRate();
00042 }
```

**6.1.3.18 DSG::DSGFrequency const& DSG::SampleRate ( DSG::DSGFrequency const & value ) [inline]**

**DSG::SampleRate** - Set Global Sample Rate.

Definition at line 44 of file [AudioSettings.h](#).

```
00044                                     {
00045     return DSG::AudioSettings::SampleRate(value);
00046 }
```

**6.1.3.19 double DSG::Sin ( double const & x ) [inline]**

[DSG::Sin\(\)](#) - General Purpose Sin Function, double precision.

Definition at line 66 of file [Sine.h](#).

```
00066      {
00067      return static_cast<double>(Sin<Sine_Default>(x)); //wrap default implementation as non template
00068      }
```

**6.1.3.20 float DSG::Sin ( float const & x ) [inline]**

[DSG::Sin\(\)](#) - General Purpose Sin Function, single precision.

Definition at line 71 of file [Sine.h](#).

```
00071      {
00072      return static_cast<float>(Sin<Sine_Default>(x));
00073      }
```

**6.1.3.21 template<typename decimal > decimal DSG::Sinc ( decimal const & x ) [inline]**

[DSG::Sinc](#) - Implements the [Sinc\(\)](#) function ( $\sin(\pi x)/\pi x$ )

Definition at line 34 of file [Sinc.h](#).

```
00034      {
00035      static_assert(std::is_floating_point<decimal>::value==true, "DSG::Sinc Function Requires Floating
Point Type");
00036      decimal pix;
00037      if (DSG::IsDenormal(x)) {
00038          return 1.0;
00039      }else{
00040          pix = PI*x;
00041          return DSG::Sin(pix)/pix;
00042      }
00043      }
```

**6.1.3.22 template<typename integer > void DSG::Sleep ( integer const & milliseconds )**

[DSG::Sleep](#) - Millisecond Sleep Function.

Definition at line 31 of file [Sleep.h](#).

```
00031      {
00032      std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
00033      }
```

**6.1.3.23 template<int lower, int upper, int value> void DSG::StaticAssertBounds ( )**

[DSG::StaticAssertBounds](#) - Fails on compile time if value is not within bounds.

Definition at line 39 of file [Bounds.h](#).

```
00039      {
00040      static_assert(value>=lower && value<=upper, "Failed Static Bounds Assert");
00041      }
```

**6.2 DSG::Analog Namespace Reference**

[DSG::Analog](#) - Namespace Containing [Analog](#) Style Oscillators.

## Classes

- class [AnalogSaw](#)  
*DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.*
- class [AnalogSquare](#)  
*DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator.*
- class [AnalogTriangle](#)  
*DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.*

### 6.2.1 Detailed Description

[DSG::Analog](#) - Namespace Containing [Analog](#) Style Oscillators.

## 6.3 DSG::BLIT Namespace Reference

[DSG::BLIT](#) - Namespace Containing [BLIT](#) Based Oscillators.

## Classes

- class [Blit](#)  
*DSG::BLIT::Blit - Band-Limited Impulse Train Generator.*
- class [BlitSaw](#)  
*DSG::BLIT::BlitSaw - Saw Wave Generator Based on [BLIT](#) Algorithm.*
- class [BlitSquare](#)
- class [BlitTriangle](#)

### 6.3.1 Detailed Description

[DSG::BLIT](#) - Namespace Containing [BLIT](#) Based Oscillators.

## 6.4 DSG::DPW Namespace Reference

[DSG::DPW](#) - Generators using the [DPW](#) method.

## Classes

- class [DPW\\_Differentiator](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the [DPW](#) Algorithm.*
- class [DPW\\_Differentiator< 1 >](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the 1st order [DPW](#) Algorithm.*
- class [DPW\\_Differentiator< 2 >](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the 2nd order [DPW](#) Algorithm.*
- class [DPW\\_Differentiator< 3 >](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the 3rd order [DPW](#) Algorithm.*
- class [DPW\\_Differentiator< 4 >](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the 4th order [DPW](#) Algorithm.*
- class [DPW\\_Differentiator< 5 >](#)  
*DSG::DPW::DPW\_Differentiator - Class Performing Differentiation for the 5th order [DPW](#) Algorithm.*

- class [DPW\\_Differentiator< 6 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 6th order [DPW](#) Algorithm.
- class [DPWSaw](#)  
[DSG::DPW::DPWSaw](#) - Sawtooth Generator using the Nth Order [DPW](#) algorithm.

## Functions

- `template<unsigned order>`  
[DSG::DSGSample DPW\\_Polynomial](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 1 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 1st Order Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 2 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 2nd order Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 3 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 3rd order Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 4 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 4th order Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 5 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 5th order Polynomial used in [DPW](#) Algorithm.
- `template<>`  
[DSG::DSGSample DPW\\_Polynomial< 6 >](#) ([DSG::DSGSample](#) const &value)  
[DSG::DPW::DPW\\_Polynomial](#) - 6th order Polynomial used in [DPW](#) Algorithm.

### 6.4.1 Detailed Description

[DSG::DPW](#) - Generators using the [DPW](#) method.

### 6.4.2 Function Documentation

**6.4.2.1** `template<unsigned order> DSG::DSGSample DSG::DPW::DPW_Polynomial ( DSG::DSGSample const & value ) [inline]`

[DSG::DPW::DPW\\_Polynomial](#) - Polynomial used in [DPW](#) Algorithm.

Definition at line 38 of file [DPW.h](#).

```
00038                                     {
00039         DSG::StaticAssertBounds<1,6,order>(); //must be 1-6 order
00040         return value;
00041     }
```

**6.4.2.2** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 1 > ( DSG::DSGSample const & value ) [inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 1st Order Polynomial used in [DPW](#) Algorithm.

Definition at line 44 of file [DPW.h](#).



```

00044                                     {
00045         return value;
00046     }

```

**6.4.2.3** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 2 > ( DSG::DSGSample const & value )`  
`[inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 2nd order Polynoimal used in [DPW](#) Algorithm.

Definition at line 49 of file [DPW.h](#).

```

00049                                     {
00050         return DSG::Pow<2>(value);
00051     }

```

**6.4.2.4** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 3 > ( DSG::DSGSample const & value )`  
`[inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 3rd order Polynoimal used in [DPW](#) Algorithm.

Definition at line 54 of file [DPW.h](#).

```

00054                                     {
00055         return DSG::Pow<3>(value) - value;
00056     }

```

**6.4.2.5** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 4 > ( DSG::DSGSample const & value )`  
`[inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 4th order Polynoimal used in [DPW](#) Algorithm.

Definition at line 59 of file [DPW.h](#).

```

00059                                     {
00060         return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
00061     }

```

**6.4.2.6** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 5 > ( DSG::DSGSample const & value )`  
`[inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 5th order Polynoimal used in [DPW](#) Algorithm.

Definition at line 64 of file [DPW.h](#).

```

00064                                     {
00065         return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
00066     }

```

**6.4.2.7** `template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 6 > ( DSG::DSGSample const & value )`  
`[inline]`

[DSG::DPW::DPW\\_Polynomial](#) - 6th order Polynoimal used in [DPW](#) Algorithm.

Definition at line 69 of file [DPW.h](#).

```

00069                                     {
00070         return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
    DPW_Polynomial<2>(value);
00071     }

```

## 6.5 DSG::EPTR Namespace Reference

[DSG::EPTR](#) - Generators Based On The Efficient Polynomial Transfer Region Algorithm.

### Classes

- class [EPTRSaw](#)  
*[DSG::EPTR::EPTRSaw](#) - Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm.*

### 6.5.1 Detailed Description

[DSG::EPTR](#) - Generators Based On The Efficient Polynomial Transfer Region Algorithm.

## 6.6 DSG::Filter Namespace Reference

[DSG::Filter](#) - Filters.

### Classes

- class [DCBlocker](#)  
*[DSG::Filter::DCBlocker](#) - DC blocking filter.*
- class [FilterBase](#)  
*[DSG::Filter::FilterBase](#) - Filter Base Class, implements interface for cutoff frequency.*
- class [LeakyIntegrator](#)  
*[DSG::Filter::LeakyIntegrator](#) - Leaky integrator.*

### 6.6.1 Detailed Description

[DSG::Filter](#) - Filters.

## 6.7 DSG::Fourier Namespace Reference

[DSG::Fourier](#) - Namespace Containing [Fourier](#) Series Based Oscillators.

### Classes

- class [FourierSaw](#)  
*[DSG::Fourier::FourierSaw](#) - Fourier Series Sawtooth Wave Generator.*
- class [FourierSeriesGenerator](#)  
*[DSG::Fourier::FourierSeriesGenerator](#) - Generates a wave form using a user specified [Fourier](#) Series.*
- class [FourierSquare](#)  
*[DSG::Fourier::FourierSquare](#) - Fourier Series Square Wave Generator.*
- class [FourierTriangle](#)  
*[DSG::Fourier::FourierTriangle](#) - Fourier Series Triangle Wave Generator.*
- class [Harmonic](#)  
*[DSG::Fourier::Harmonic](#) - Represents a single harmonic in a [Fourier](#) Series.*

### 6.7.1 Detailed Description

[DSG::Fourier](#) - Namespace Containing [Fourier](#) Series Based Oscillators.

## 6.8 DSG::Noise Namespace Reference

[DSG::Noise](#) - [Noise](#) Generators.

### Functions

- `template<typename decimal = DSG::DSGSample>`  
`decimal Gaussian (decimal=0.0)`  
*[DSG::Noise::Gaussian](#) - Gaussian [Noise](#) Generator Function.*
- `template<typename decimal = DSG::DSGSample>`  
`decimal Pink (decimal=0.0)`  
*[DSG::Noise::Pink](#) - Pink [Noise](#) Generator Function.*
- `template<typename decimal = DSG::DSGSample>`  
`decimal Random (decimal=0.0)`  
*[DSG::Noise::Random](#) - Random Number Function.*
- `template<typename decimal = DSG::DSGSample>`  
`decimal White (decimal=0.0)`  
*[DSG::Noise::White](#) - White [Noise](#) Generator Function.*

### 6.8.1 Detailed Description

[DSG::Noise](#) - [Noise](#) Generators.

### 6.8.2 Function Documentation

**6.8.2.1** `template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Gaussian ( decimal = 0.0 )`

[DSG::Noise::Gaussian](#) - Gaussian [Noise](#) Generator Function.

Definition at line 35 of file [Gaussian.h](#).

```
00035         {
00036             static decimal normalizer=1; //variable used to actively normalize the output
00037             //to enforce compatability with DSG::LUT a dummy parameter is applied
00038             //this parameter is useless except for compatability reasons
00039             decimal R1 = DSG::Noise::White();
00040             decimal R2 = DSG::Noise::White();
00041             decimal x= (decimal)sqrt(-2.0f * log(R1)) * DSG::Cos(R2);
00042             if (DSG::Abs(x)>normalizer) {
00043                 //store highest output
00044                 normalizer=DSG::Abs(x);
00045             }
00046             x/=normalizer; //normalize
00047             return x;
00048         }
```

**6.8.2.2** `template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Pink ( decimal = 0.0 )`

[DSG::Noise::Pink](#) - Pink [Noise](#) Generator Function.

Definition at line 35 of file [Pink.h](#).

```

00035         {
00036             //routine: Get white or gaussian, filter, return
00037             static decimal b0,b1,b2,b3,b4,b5,b6;
00038             static decimal normalizer=1;//variable used to actively normalize the output
00039             static DSG::DCBlocker _block;
00040             decimal white = DSG::Noise::Gaussian();
00041             decimal pink;
00042             //pinking filter
00043             b0 = 0.99886 * b0 + white * 0.0555179;
00044             b1 = 0.99332 * b1 + white * 0.0750759;
00045             b2 = 0.96900 * b2 + white * 0.1538520;
00046             b3 = 0.86650 * b3 + white * 0.3104856;
00047             b4 = 0.55000 * b4 + white * 0.5329522;
00048             b5 = -0.7616 * b5 - white * 0.0168980;
00049             pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
00050             b6 = white * 0.115926;
00051             if (DSG::Abs(pink)>normalizer) {
00052                 //store highest output
00053                 normalizer=DSG::Abs(pink);
00054             }
00055             pink/=normalizer;
00056             _block.Perform(pink);
00057             return pink;
00058         }

```

**6.8.2.3** `template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Random ( decimal = 0.0 ) [inline]`

[DSG::Noise::Random](#) - Random Number Function.

Definition at line 50 of file [Random.h](#).

```

00050         {
00051             static DSG::Noise::random_helper<decimal> _rand{};
00052             return _rand.next();
00053         }

```

**6.8.2.4** `template<typename decimal = DSG::DSGSample> decimal DSG::Noise::White ( decimal = 0.0 ) [inline]`

[DSG::Noise::White](#) - White Noise Generator Function.

Definition at line 35 of file [White.h](#).

```

00035         {
00036             return DSG::Random<decimal>();
00037         }

```

## 6.9 DSG::Window Namespace Reference

[DSG::Window](#) - Window functions and utilities.

### Functions

- `template<typename decimal >`  
`decimal Blackman (decimal const &x)`  
[DSG::Window::Blackman](#) - Blackman Window Function.
- `template<typename decimal , unsigned long lutsz>`  
`void ApplyWindow (DSG::LUT< decimal, lutsz > &lut, decimal(&>windowFunction)(decimal const &), decimal range=1.0)`  
[DSG::Window::ApplyWindow](#) - Apply a window function to a LUT.
- `template<typename decimal , unsigned long lutsz>`  
`void ApplyWindow (DSG::LUT< decimal, lutsz > &lut, decimal(&>windowFunction)(decimal), decimal range=1.0)`  
[DSG::Window::ApplyWindow](#) - Apply a window function to a LUT.

### 6.9.1 Detailed Description

[DSG::Window](#) - [Window](#) functions and utilities.

### 6.9.2 Function Documentation

**6.9.2.1** `template<typename decimal , unsigned long lutsizes> void DSG::Window::ApplyWindow ( DSG::LUT< decimal, lutsizes > & lut, decimal(&)(decimal const &) windowFunction, decimal range = 1.0 )`

[DSG::Window::ApplyWindow](#) - Apply a window function to a [LUT](#).

Definition at line 35 of file [Window.h](#).

```
00035
00036     {
00037         decimal step = range/(decimal)lut.Size();
00038         decimal phs=0;
00039         for (int i=0; i<lut.Size(); ++i) {
00040             lut[i]*=windowFunction(phs);
00041             phs+=step;
00042     }
```

**6.9.2.2** `template<typename decimal , unsigned long lutsizes> void DSG::Window::ApplyWindow ( DSG::LUT< decimal, lutsizes > & lut, decimal(&)(decimal) windowFunction, decimal range = 1.0 )`

[DSG::Window::ApplyWindow](#) - Apply a window function to a [LUT](#).

Definition at line 45 of file [Window.h](#).

```
00045
00046     {
00047         decimal step = range/(decimal)lut.Size();
00048         decimal phs=0;
00049         for (int i=0; i<lut.Size(); ++i) {
00050             lut[i]*=windowFunction(phs);
00051             phs+=step;
00052     }
```

**6.9.2.3** `template<typename decimal > decimal DSG::Window::Blackman ( decimal const & x ) [inline]`

[DSG::Window::Blackman](#) - Blackman [Window](#) Function.

Definition at line 36 of file [Blackman.h](#).

```
00036
00037     // Generate Blackman Window
00038     /*
00039     Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
00040     */
00041     static_assert(std::is_floating_point<decimal>::value==true, "DSG::Blackman Function Requires
Floating Point Type");
00042     //we will implement the blackman window as a function as if it were sin(x)
00043     //cos input domain 0-1 not 0-2pi
00044     //range checking is handles within DSG::Cos
00045     decimal phs=x;
00046     while (phs>1.0) {
00047         phs-=1.0;
00048     }
00049     return 0.42 - (0.5 * DSG::Cos(phs)) + (0.08 * DSG::Cos(2.0*phs));
00050 }
```



## Chapter 7

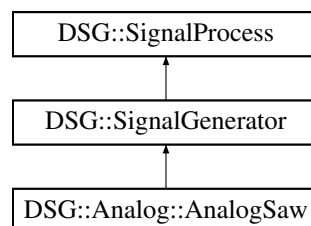
# Class Documentation

### 7.1 DSG::Analog::AnalogSaw Class Reference

[DSG::Analog::AnalogSaw](#) - [Analog](#) Syle Saw Wave Generator.

```
#include <AnalogSaw.h>
```

Inheritance diagram for DSG::Analog::AnalogSaw:



#### Public Member Functions

- [AnalogSaw](#) ()
- [AnalogSaw](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~AnalogSaw](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

#### Protected Attributes

- [DSG::DSGSample \\_stor](#)

#### Additional Inherited Members

##### 7.1.1 Detailed Description

[DSG::Analog::AnalogSaw](#) - [Analog](#) Syle Saw Wave Generator.

Definition at line 34 of file [AnalogSaw.h](#).

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 DSG::Analog::AnalogSaw::AnalogSaw ( )

Definition at line 25 of file [AnalogSaw.cpp](#).

```
00025 :DSG::SignalGenerator() {}
```

### 7.1.2.2 DSG::Analog::AnalogSaw::AnalogSaw ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [AnalogSaw.cpp](#).

```
00026 :DSG::SignalGenerator(frequency,offset) {}
```

### 7.1.2.3 DSG::Analog::AnalogSaw::~~AnalogSaw ( ) [virtual]

Definition at line 27 of file [AnalogSaw.cpp](#).

```
00027 {}
```

## 7.1.3 Member Function Documentation

### 7.1.3.1 bool DSG::Analog::AnalogSaw::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 44 of file [AnalogSaw.h](#).

```
00044                                     {
00045         _stor=_phasor;
00046         _stor+=0.5;
00047         if (_stor>1.0) {
00048             --_stor;
00049         }
00050         _stor-=0.5;
00051         _stor*=2.0;
00052         signal=_stor;
00053         step();
00054         return true;
00055     }
```

### 7.1.3.2 bool DSG::Analog::AnalogSaw::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 56 of file [AnalogSaw.h](#).

```
00056                                     {
00057         signal.Flush();
00058         while (!signal.Full()) {
00059             if (Perform(_storage)) {
00060                 if(signal.Write(_storage)){
00061                     }else return false;
00062                 }else return false;
00063             }return true;
00064         }
```



### 7.1.4 Member Data Documentation

#### 7.1.4.1 DSG::DSGSample DSG::Analog::AnalogSaw::\_stor [protected]

Definition at line 42 of file [AnalogSaw.h](#).

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

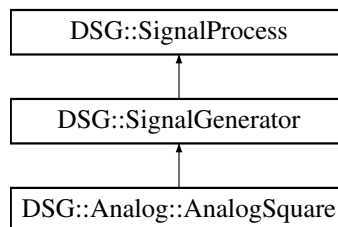
- [AnalogSaw.h](#)
- [AnalogSaw.cpp](#)

## 7.2 DSG::Analog::AnalogSquare Class Reference

[DSG::Analog::AnalogSquare](#) - [Analog](#) Syle Square Wave Generator.

```
#include <AnalogSquare.h>
```

Inheritance diagram for DSG::Analog::AnalogSquare:



### Public Member Functions

- [AnalogSquare](#) ()
- [AnalogSquare](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~AnalogSquare](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

### Additional Inherited Members

#### 7.2.1 Detailed Description

[DSG::Analog::AnalogSquare](#) - [Analog](#) Syle Square Wave Generator.

Definition at line 34 of file [AnalogSquare.h](#).

#### 7.2.2 Constructor & Destructor Documentation

##### 7.2.2.1 DSG::Analog::AnalogSquare::AnalogSquare ( )

Definition at line 25 of file [AnalogSquare.cpp](#).

```
00025 :DSG::SignalGenerator() {}
```

### 7.2.2.2 DSG::Analog::AnalogSquare::AnalogSquare ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [AnalogSquare.cpp](#).

```
00026 :DSG::SignalGenerator(frequency,offset){}
```

### 7.2.2.3 DSG::Analog::AnalogSquare::~~AnalogSquare ( ) [virtual]

Definition at line 27 of file [AnalogSquare.cpp](#).

```
00027 {}
```

## 7.2.3 Member Function Documentation

### 7.2.3.1 bool DSG::Analog::AnalogSquare::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 42 of file [AnalogSquare.h](#).

```
00042                                     {
00043     signal=_phasor < 0.5 ? 1.0:-1.0;
00044     step();
00045     return true;
00046 }
```

### 7.2.3.2 bool DSG::Analog::AnalogSquare::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 47 of file [AnalogSquare.h](#).

```
00047                                     {
00048     signal.Flush();
00049     while (!signal.Full()) {
00050         if (Perform(_storage)) {
00051             if(signal.Write(_storage)){
00052                 }else return false;
00053             }else return false;
00054         }return true;
00055     }
```

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

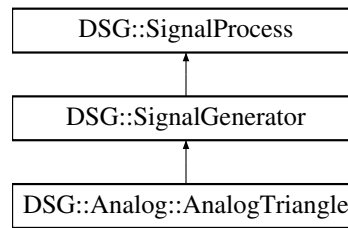
- [AnalogSquare.h](#)
- [AnalogSquare.cpp](#)

## 7.3 DSG::Analog::AnalogTriangle Class Reference

[DSG::Analog::AnalogTriangle](#) - Analog Syle Triangle Wave Generator.

```
#include <AnalogTriangle.h>
```

Inheritance diagram for DSG::Analog::AnalogTriangle:



## Public Member Functions

- [AnalogTriangle](#) ()
- [AnalogTriangle](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~AnalogTriangle](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

## Protected Attributes

- [DSG::DSGSample \\_stor](#)

## Additional Inherited Members

### 7.3.1 Detailed Description

[DSG::Analog::AnalogTriangle](#) - Analog Syle Triangle Wave Generator.

Definition at line 34 of file [AnalogTriangle.h](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 [DSG::Analog::AnalogTriangle::AnalogTriangle](#) ( )

Definition at line 25 of file [AnalogTriangle.cpp](#).

```
00025 :DSG::SignalGenerator() {}
```

#### 7.3.2.2 [DSG::Analog::AnalogTriangle::AnalogTriangle](#) ( [DSG::DSGFrequency](#) const & *frequency*, [DSG::DSGPhase](#) const & *offset* )

Definition at line 26 of file [AnalogTriangle.cpp](#).

```
00026 :DSG::SignalGenerator(frequency,offset) {}
```

#### 7.3.2.3 [DSG::Analog::AnalogTriangle::~~AnalogTriangle](#) ( ) [virtual]

Definition at line 27 of file [AnalogTriangle.cpp](#).

```
00027 {}
```

### 7.3.3 Member Function Documentation

#### 7.3.3.1 `bool DSG::Analog::AnalogTriangle::Perform ( DSG::DSGSample & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 44 of file [AnalogTriangle.h](#).

```

00044                                     {
00045         _stor = _phasor;
00046         _stor+=0.25;
00047         while (_stor>1.0) {
00048             _stor-=1.0;
00049         }
00050         _stor-=0.5;
00051         if (_stor<0) {
00052             _stor*=-1.0;
00053         }
00054         _stor-=0.25;
00055         _stor*=-4.0;
00056         signal = _stor;
00057         step();//always last
00058         return true;
00059     }

```

#### 7.3.3.2 `bool DSG::Analog::AnalogTriangle::Perform ( DSG::RingBuffer & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 60 of file [AnalogTriangle.h](#).

```

00060                                     {
00061         signal.Flush();
00062         while (!signal.Full()) {
00063             if (Perform(_storage)) {
00064                 if(signal.Write(_storage)){
00065                     }else return false;
00066                 }else return false;
00067             }return true;
00068     }

```

### 7.3.4 Member Data Documentation

#### 7.3.4.1 `DSG::DSGSample DSG::Analog::AnalogTriangle::_stor` `[protected]`

Definition at line 42 of file [AnalogTriangle.h](#).

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

- [AnalogTriangle.h](#)
- [AnalogTriangle.cpp](#)

## 7.4 DSG::AudioSettings Class Reference

[DSG::AudioSettings](#) - Global Storage For Audio Settings Such As Sample Rate.

```
#include <AudioSettings.h>
```

### Static Public Member Functions

- static [DSG::DSGFrequency](#) const & [SampleRate](#) ()
- static [DSG::DSGFrequency](#) const & [SampleRate](#) ([DSG::DSGFrequency](#) const &value)
- static [DSG::DSGFrequency](#) const & [Nyquist](#) ()

## Static Protected Attributes

- static [DSG::DSGFrequency \\_sampleRate](#)
- static [DSG::DSGFrequency \\_nyquist](#)

### 7.4.1 Detailed Description

[DSG::AudioSettings](#) - Global Storage For Audio Settings Such As Sample Rate.

Definition at line 30 of file [AudioSettings.h](#).

### 7.4.2 Member Function Documentation

#### 7.4.2.1 DSG::DSGFrequency const & DSG::AudioSettings::Nyquist ( ) [static]

Definition at line 35 of file [AudioSettings.cpp](#).

```
00035                                     {
00036     return _nyquist;
00037 }
```

#### 7.4.2.2 DSG::DSGFrequency const & DSG::AudioSettings::SampleRate ( ) [static]

Definition at line 27 of file [AudioSettings.cpp](#).

```
00027                                     {
00028     return _sampleRate;
00029 }
```

#### 7.4.2.3 DSG::DSGFrequency const & DSG::AudioSettings::SampleRate ( DSG::DSGFrequency const & value ) [static]

Definition at line 30 of file [AudioSettings.cpp](#).

```
00030                                     {
00031     _sampleRate = value;
00032     _nyquist = _sampleRate*0.5;
00033     return _sampleRate;
00034 }
```

### 7.4.3 Member Data Documentation

#### 7.4.3.1 DSG::DSGFrequency DSG::AudioSettings::\_nyquist [static], [protected]

Definition at line 37 of file [AudioSettings.h](#).

#### 7.4.3.2 DSG::DSGFrequency DSG::AudioSettings::\_sampleRate [static], [protected]

Definition at line 36 of file [AudioSettings.h](#).

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

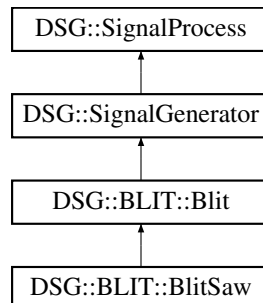
- [AudioSettings.h](#)
- [AudioSettings.cpp](#)

## 7.5 DSG::BLIT::Blit Class Reference

[DSG::BLIT::Blit](#) - Band-Limited Impulse Train Generator.

```
#include <BLIT.h>
```

Inheritance diagram for DSG::BLIT::Blit:



### Public Member Functions

- [Blit](#) ()
- [Blit](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~Blit](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ([DSG::DSGFrequency](#) const &value)

### Protected Attributes

- unsigned long [p\\_](#)
- unsigned long [m\\_](#)
- unsigned long [\\_h](#)
- double [a\\_](#)
- [DSG::DSGSample](#) denominator
- [DSG::DSGSample](#) value

### Additional Inherited Members

#### 7.5.1 Detailed Description

[DSG::BLIT::Blit](#) - Band-Limited Impulse Train Generator.

**Todo** Re-write [DSG::BLIT::Blit](#) algorithm

Definition at line 39 of file [BLIT.h](#).

#### 7.5.2 Constructor & Destructor Documentation

##### 7.5.2.1 DSG::BLIT::Blit::Blit ( )

Definition at line 25 of file [BLIT.cpp](#).

```

00025         :DSG::SignalGenerator() {
00026     Frequency(0);
00027 }
```

### 7.5.2.2 DSG::BLIT::Blit::Blit ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 28 of file [BLIT.cpp](#).

```
00028                                     :
00029     DSG::SignalGenerator (frequency,offset) {
00029         Frequency (frequency);
00030 }
```

### 7.5.2.3 DSG::BLIT::Blit::~Blit ( ) [virtual]

Definition at line 31 of file [BLIT.cpp](#).

```
00031 {}
```

## 7.5.3 Member Function Documentation

### 7.5.3.1 DSG::DSGFrequency const & DSG::BLIT::Blit::Frequency ( DSG::DSGFrequency const & *value* ) [inline], [virtual]

Reimplemented from [DSG::SignalGenerator](#).

Reimplemented in [DSG::BLIT::BlitSaw](#).

Definition at line 78 of file [BLIT.h](#).

```
00078                                     {
00079     this->SignalGenerator::Frequency (value);
00080     p_ = DSG::SampleRate ()/_frequency;
00081     _h = (unsigned) floor (p_*0.5);
00082     m_ = 2 * (_h)+1;
00083     a_ = m_/ (double) p_;
00084     return _frequency;
00085 }
```

### 7.5.3.2 bool DSG::BLIT::Blit::Perform ( DSG::DSGSample & *signal* ) [inline], [virtual]

Reimplemented from [DSG::SignalGenerator](#).

Reimplemented in [DSG::BLIT::BlitSaw](#).

Definition at line 55 of file [BLIT.h](#).

```
00055                                     {
00056     //found better results in this case with built in sine function. not performance wise but
00056     algorithmically
00057     denominator = m_ * sin (PI*_phasor);
00058     if (DSG::IsDenormal (denominator)) {
00059         signal = a_;
00060     }else{
00061         value = sin (PI*_phasor * m_);
00062         value/=denominator;
00063         value*=a_;
00064         signal = value;
00065     }
00066     step();
00067     return true;
00068 }
```

### 7.5.3.3 bool DSG::BLIT::Blit::Perform ( DSG::RingBuffer & *signal* ) [inline], [virtual]

Reimplemented from [DSG::SignalGenerator](#).

Reimplemented in [DSG::BLIT::BlitSaw](#).

Definition at line 69 of file [BLIT.h](#).

```

00069                                     {
00070         signal.Flush();
00071         while (!signal.Full()) {
00072             if (Perform(_storage)) {
00073                 if(signal.Write(_storage)){
00074                     }else return false;
00075                 }else return false;
00076             }return true;
00077         }

```

## 7.5.4 Member Data Documentation

### 7.5.4.1 unsigned long DSG::BLIT::Blit::\_h [protected]

Definition at line 50 of file [BLIT.h](#).

### 7.5.4.2 double DSG::BLIT::Blit::a\_ [protected]

Definition at line 51 of file [BLIT.h](#).

### 7.5.4.3 DSG::DSGSample DSG::BLIT::Blit::denominator [protected]

Definition at line 52 of file [BLIT.h](#).

### 7.5.4.4 unsigned long DSG::BLIT::Blit::m\_ [protected]

Definition at line 49 of file [BLIT.h](#).

### 7.5.4.5 unsigned long DSG::BLIT::Blit::p\_ [protected]

Definition at line 48 of file [BLIT.h](#).

### 7.5.4.6 DSG::DSGSample DSG::BLIT::Blit::value [protected]

Definition at line 53 of file [BLIT.h](#).

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

- [BLIT.h](#)
- [BLIT.cpp](#)

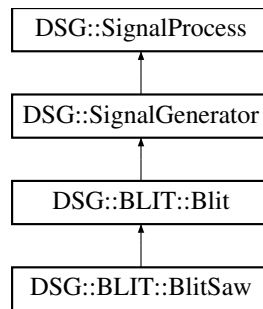
## 7.6 DSG::BLIT::BlitSaw Class Reference

[DSG::BLIT::BlitSaw](#) - Saw Wave Generator Based on [BLIT](#) Algorithm.

```
#include <BLITSaw.h>
```

Inheritance diagram for [DSG::BLIT::BlitSaw](#):





## Public Member Functions

- [BlitSaw](#) ()
- [BlitSaw](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~BlitSaw](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ([DSG::DSGFrequency](#) const &value)

## Protected Attributes

- [DSG::DSGSample](#) [C2\\_](#)
- [DSG::DSGSample](#) [Register\\_](#)

## Additional Inherited Members

### 7.6.1 Detailed Description

[DSG::BLIT::BlitSaw](#) - Saw Wave Generator Based on [BLIT](#) Algorithm.

**Todo** Re-write [DSG::BLIT::BlitSaw](#) algorithm

Definition at line 34 of file [BLITSaw.h](#).

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 [DSG::BLIT::BlitSaw::BlitSaw](#) ( )

Definition at line 25 of file [BLITSaw.cpp](#).

```

00025         :DSG::BLIT::Blit (),Register\_ (0) {
00026     Frequency (0);
00027 }
```

#### 7.6.2.2 [DSG::BLIT::BlitSaw::BlitSaw](#) ( [DSG::DSGFrequency](#) const & frequency, [DSG::DSGPhase](#) const & offset )

Definition at line 28 of file [BLITSaw.cpp](#).

```

00028                                     :
00029     DSG::BLIT::Blit (frequency,offset),Register\_ (0) {
00029     Frequency (frequency);
00030 }
```

### 7.6.2.3 DSG::BLIT::BlitSaw::~~BlitSaw ( ) [virtual]

Definition at line 31 of file [BLITSaw.cpp](#).

```
00031 {}
```

## 7.6.3 Member Function Documentation

### 7.6.3.1 DSG::DSGFrequency const & DSG::BLIT::BlitSaw::Frequency ( DSG::DSGFrequency const & value ) [inline], [virtual]

Reimplemented from [DSG::BLIT::Blit](#).

Definition at line 72 of file [BLITSaw.h](#).

```
00072                                     {
00073         this->SignalGenerator::Frequency(value);
00074         p_ = DSG::SampleRate()/_frequency;
00075         _h = (unsigned) floor(p_*0.5);
00076         m_ = 2 * (_h)+1;
00077         a_ = m_/ (double)p_;
00078         C2_ = 1.0/(double)p_;
00079         return _frequency;
00080     }
```

### 7.6.3.2 bool DSG::BLIT::BlitSaw::Perform ( DSG::DSGSample & signal ) [inline], [virtual]

Reimplemented from [DSG::BLIT::Blit](#).

Definition at line 46 of file [BLITSaw.h](#).

```
00046                                     {
00047         denominator = m_ * sin(PI*_phasor);
00048         if (DSG::IsDenormal(denominator)) {
00049             signal = a_;
00050         }else{
00051             value = sin(PI*_phasor * m_);
00052             value/=denominator;
00053             value*=a_;
00054             signal = value;
00055         }
00056         step();
00057         signal += (Register_ - C2_);
00058         Register_ = signal * 0.995;
00059         C2_+=signal;
00060         C2_*=0.5;
00061         return true;
00062     }
```

### 7.6.3.3 bool DSG::BLIT::BlitSaw::Perform ( DSG::RingBuffer & signal ) [inline], [virtual]

Reimplemented from [DSG::BLIT::Blit](#).

Definition at line 63 of file [BLITSaw.h](#).

```
00063                                     {
00064         signal.Flush();
00065         while (!signal.Full()) {
00066             if (Perform(_storage)) {
00067                 if(signal.Write(_storage)){
00068                     }else return false;
00069                 }else return false;
00070             }return true;
00071         }
```

### 7.6.4 Member Data Documentation

#### 7.6.4.1 DSG::DSGSample DSG::BLIT::BlitSaw::C2\_ [protected]

Definition at line 43 of file [BLITSaw.h](#).

#### 7.6.4.2 DSG::DSGSample DSG::BLIT::BlitSaw::Register\_ [protected]

Definition at line 44 of file [BLITSaw.h](#).

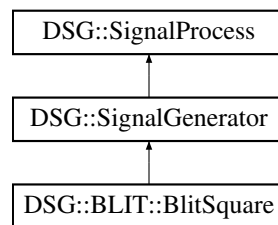
The documentation for this class was generated from the following files:

- [BLITSaw.h](#)
- [BLITSaw.cpp](#)

## 7.7 DSG::BLIT::BlitSquare Class Reference

```
#include <BLITSquare.h>
```

Inheritance diagram for DSG::BLIT::BlitSquare:



### Additional Inherited Members

#### 7.7.1 Detailed Description

**Todo** Write [DSG::BLIT::BlitSquare](#) algorithm

Definition at line 33 of file [BLITSquare.h](#).

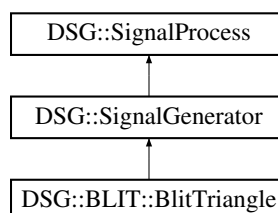
The documentation for this class was generated from the following file:

- [BLITSquare.h](#)

## 7.8 DSG::BLIT::BlitTriangle Class Reference

```
#include <BLITTriangle.h>
```

Inheritance diagram for DSG::BLIT::BlitTriangle:



## Additional Inherited Members

### 7.8.1 Detailed Description

**Todo** Write [DSG::BLIT::BlitTriangle](#) algorithm

Definition at line 34 of file [BLITTriangle.h](#).

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

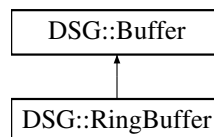
- [BLITTriangle.h](#)

## 7.9 DSG::Buffer Class Reference

[DSG::Buffer](#) - Base Class For [DSG::RingBuffer](#). Not For Direct Use.

```
#include <Buffer.h>
```

Inheritance diagram for [DSG::Buffer](#):



### Public Member Functions

- [Buffer](#) ()
- [Buffer](#) (size\_t size)
- [Buffer](#) ([Buffer](#) const &other)
- [Buffer](#) & [operator=](#) ([Buffer](#) const &other)
- virtual [~Buffer](#) ()
- [DSG::DSGSample](#) & [operator\[\]](#) (size\_t const &index)
- size\_t const & [Size](#) () const

### Protected Attributes

- [DSG::DSGSample](#) \* [\\_buffer](#)
- size\_t [\\_size](#)

### 7.9.1 Detailed Description

[DSG::Buffer](#) - Base Class For [DSG::RingBuffer](#). Not For Direct Use.

Definition at line 34 of file [Buffer.h](#).

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 DSG::Buffer::Buffer ( )

Definition at line 25 of file [Buffer.cpp](#).

```
00025 :_size(0),_buffer(nullptr){}
```

### 7.9.2.2 DSG::Buffer::Buffer ( size\_t size )

Definition at line 26 of file [Buffer.cpp](#).

```
00026 :_size(size),_buffer(new DSG::DSGSample[size]) {}
```

### 7.9.2.3 DSG::Buffer::Buffer ( Buffer const & other )

Definition at line 27 of file [Buffer.cpp](#).

```
00027 {
00028     _buffer = new DSG::DSGSample[_size];
00029     _size = other._size;
00030     *this = other;
00031 }
```

### 7.9.2.4 DSG::Buffer::~Buffer ( ) [virtual]

Definition at line 45 of file [Buffer.cpp](#).

```
00045 {
00046     if (_buffer!=nullptr) {
00047         delete [] _buffer;
00048     }
00049 }
```

## 7.9.3 Member Function Documentation

### 7.9.3.1 DSG::Buffer & DSG::Buffer::operator= ( Buffer const & other )

Definition at line 32 of file [Buffer.cpp](#).

```
00032 {
00033     if (_size!=other._size) {
00034         if (_buffer!=nullptr) {
00035             delete [] _buffer;
00036         }
00037         _size = other._size;
00038         _buffer = new DSG::DSGSample[_size];
00039     }
00040     for (int i=0; i<_size; ++i) {
00041         _buffer[i] = other._buffer[i];
00042     }
00043     return *this;
00044 }
```

### 7.9.3.2 DSG::DSGSample & DSG::Buffer::operator[] ( size\_t const & index )

Definition at line 50 of file [Buffer.cpp](#).

```
00050 {
00051     #ifdef DEBUG
00052         assert(index<_size);
00053     #endif
00054     return _buffer[index];
00055 }
```

### 7.9.3.3 size\_t const & DSG::Buffer::Size ( ) const [inline]

Definition at line 47 of file [Buffer.h](#).

```
00047 {
00048     return _size;
00049 }
```

### 7.9.4 Member Data Documentation

#### 7.9.4.1 `DSG::DSGSample* DSG::Buffer::_buffer` [protected]

Definition at line 44 of file [Buffer.h](#).

#### 7.9.4.2 `size_t DSG::Buffer::_size` [protected]

Definition at line 45 of file [Buffer.h](#).

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

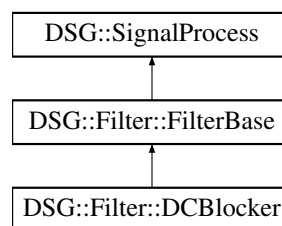
- [Buffer.h](#)
- [Buffer.cpp](#)

## 7.10 DSG::Filter::DCBlocker Class Reference

[DSG::Filter::DCBlocker](#) - DC blocking filter.

```
#include <DCBlocker.h>
```

Inheritance diagram for `DSG::Filter::DCBlocker`:



### Public Member Functions

- [DCBlocker](#) ()
- virtual [~DCBlocker](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

### Protected Attributes

- unsigned long [count](#)
- [DSG::DSGSample \\_temp](#)
- [DSG::DSGSample xm1](#)
- [DSG::DSGSample ym1](#)
- [DSG::DSGSample x](#)
- [DSG::DSGSample \\_a](#)

### 7.10.1 Detailed Description

[DSG::Filter::DCBlocker](#) - DC blocking filter.

Definition at line 33 of file [DCBlocker.h](#).

## 7.10.2 Constructor & Destructor Documentation

### 7.10.2.1 DSG::Filter::DCBlocker::DCBlocker ( )

Definition at line 25 of file [DCBlocker.cpp](#).

```
00025 :DSG::Filter::FilterBase(),_a(0.995),xml(0),yml(0),
      x(0),_temp(0){}
```

### 7.10.2.2 DSG::Filter::DCBlocker::~~DCBlocker ( ) [virtual]

Definition at line 26 of file [DCBlocker.cpp](#).

```
00026 {}
```

## 7.10.3 Member Function Documentation

### 7.10.3.1 bool DSG::Filter::DCBlocker::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::Filter::FilterBase](#).

Definition at line 47 of file [DCBlocker.h](#).

```
00047                                     {
00048         x = signal;
00049         signal= x - xml+ (_a * yml);
00050         xml = x;
00051         yml=signal;
00052         return true;
00053     }
```

### 7.10.3.2 bool DSG::Filter::DCBlocker::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Reimplemented from [DSG::Filter::FilterBase](#).

Definition at line 54 of file [DCBlocker.h](#).

```
00054                                     {
00055         if (!signal.Empty()) {
00056             count = signal.Count();
00057             while (count-- > 0) {
00058                 if(signal.Read(_temp)){
00059                     if (Perform(_temp)) {
00060                         signal.Write(_temp);
00061                     }else return false;
00062                 }else return false;
00063             }return true;
00064         }else return false;
00065     }
```

## 7.10.4 Member Data Documentation

### 7.10.4.1 DSG::DSGSample DSG::Filter::DCBlocker::\_a [protected]

Definition at line 45 of file [DCBlocker.h](#).

### 7.10.4.2 DSG::DSGSample DSG::Filter::DCBlocker::\_temp [protected]

Definition at line 41 of file [DCBlocker.h](#).

#### 7.10.4.3 unsigned long DSG::Filter::DCBlocker::count [protected]

Definition at line 40 of file [DCBlocker.h](#).

#### 7.10.4.4 DSG::DSGSample DSG::Filter::DCBlocker::x [protected]

Definition at line 44 of file [DCBlocker.h](#).

#### 7.10.4.5 DSG::DSGSample DSG::Filter::DCBlocker::xm1 [protected]

Definition at line 42 of file [DCBlocker.h](#).

#### 7.10.4.6 DSG::DSGSample DSG::Filter::DCBlocker::ym1 [protected]

Definition at line 43 of file [DCBlocker.h](#).

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

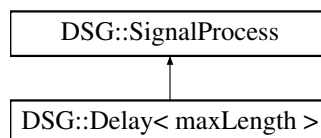
- [DCBlocker.h](#)
- [DCBlocker.cpp](#)

## 7.11 DSG::Delay< maxLength > Class Template Reference

[DSG::Delay](#) - General purpose delay line.

```
#include <Delay.h>
```

Inheritance diagram for DSG::Delay< maxLength >:



### Public Member Functions

- [Delay](#) ()
- [Delay](#) (double const &samples)
- virtual [~Delay](#) ()
- virtual unsigned long const & [Length](#) () const
- virtual unsigned long const & [Length](#) (unsigned long const &samples)
- virtual bool [Perform](#) (DSG::DSGSample &signal)
- virtual bool [Perform](#) (DSG::RingBuffer &signal)

### Protected Member Functions

- virtual void [increment](#) ()



## Protected Attributes

- unsigned long `count`
- unsigned long `_delay`
- unsigned long `_index`
- const unsigned long `_max`
- `DSG::DSGSample _buffer` [maxLength]
- `DSG::DSGSample _swap`
- `DSG::DSGSample _temp`

### 7.11.1 Detailed Description

`template<unsigned long maxLength>class DSG::Delay< maxLength >`

`DSG::Delay` - General purpose delay line.

Definition at line 33 of file `Delay.h`.

### 7.11.2 Constructor & Destructor Documentation

7.11.2.1 `template<unsigned long maxLength> DSG::Delay< maxLength >::Delay ( ) [inline]`

Definition at line 35 of file `Delay.h`.

```
00035         :DSG::SignalProcess(),_max(maxLength),_swap(0),
    _temp(0),count(0),_index(0),_delay(0){
00036         for (int i=0; i<_max; ++i) {
00037             _buffer[i]=0;
00038         }
00039     }
```

7.11.2.2 `template<unsigned long maxLength> DSG::Delay< maxLength >::Delay ( double const & samples ) [inline]`

Definition at line 40 of file `Delay.h`.

```
00040         :DSG::SignalProcess(),_max(maxLength),
    _swap(0),_temp(0),count(0),_index(0),_delay(0){
00041         for (int i=0; i<_max; ++i) {
00042             _buffer[i]=0;
00043         }
00044         if (samples>maxLength) {
00045             _delay = maxLength;
00046         }else{
00047             _delay = samples;
00048         }
00049     }
```

7.11.2.3 `template<unsigned long maxLength> virtual DSG::Delay< maxLength >::~~Delay ( ) [inline], [virtual]`

Definition at line 50 of file `Delay.h`.

```
00050 {}
```

### 7.11.3 Member Function Documentation

**7.11.3.1** `template<unsigned long maxLength> virtual void DSG::Delay< maxLength >::increment ( ) [inline], [protected], [virtual]`

Definition at line 72 of file [Delay.h](#).

```
00072                                     {
00073         ++_index;
00074         if (_index>_delay) {
00075             _index-=_delay;
00076         }
00077     }
```

**7.11.3.2** `template<unsigned long maxLength> virtual unsigned long const& DSG::Delay< maxLength >::Length ( ) const [inline], [virtual]`

Definition at line 51 of file [Delay.h](#).

```
00051                                     {
00052         return _delay;
00053     }
```

**7.11.3.3** `template<unsigned long maxLength> virtual unsigned long const& DSG::Delay< maxLength >::Length ( unsigned long const & samples ) [inline], [virtual]`

Definition at line 54 of file [Delay.h](#).

```
00054                                     {
00055         if (samples>maxLength) {
00056             _delay = maxLength;
00057         }else{
00058             _delay = samples;
00059         }
00060         return _delay;
00061     }
```

**7.11.3.4** `template<unsigned long maxLength> bool DSG::Delay< maxLength >::Perform ( DSG::DSGSample & signal ) [inline], [virtual]`

Implements [DSG::SignalProcess](#).

Definition at line 80 of file [Delay.h](#).

```
00080                                     {
00081         _swap = _buffer[_index-1];
00082         _buffer[_index-1]=signal;
00083         signal = _swap;
00084         increment ();
00085         return true;
00086     }
```

**7.11.3.5** `template<unsigned long maxLength> bool DSG::Delay< maxLength >::Perform ( DSG::RingBuffer & signal ) [inline], [virtual]`

Implements [DSG::SignalProcess](#).

Definition at line 88 of file [Delay.h](#).

```

00088                                     {
00089         if (!signal.Empty()) {
00090             count = signal.Count();
00091             while (count-- > 0) {
00092                 if (signal.Read(_temp)) {
00093                     if (Perform(_temp)) {
00094                         signal.Write(_temp);
00095                     } else return false;
00096                 } else return false;
00097             } return true;
00098         } else return false;
00099     }

```

### 7.11.4 Member Data Documentation

7.11.4.1 `template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_buffer[maxLength]`  
[protected]

Definition at line 69 of file [Delay.h](#).

7.11.4.2 `template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::_delay` [protected]

Definition at line 66 of file [Delay.h](#).

7.11.4.3 `template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::_index` [protected]

Definition at line 67 of file [Delay.h](#).

7.11.4.4 `template<unsigned long maxLength> const unsigned long DSG::Delay< maxLength >::_max` [protected]

Definition at line 68 of file [Delay.h](#).

7.11.4.5 `template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_swap`  
[protected]

Definition at line 70 of file [Delay.h](#).

7.11.4.6 `template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_temp`  
[protected]

Definition at line 71 of file [Delay.h](#).

7.11.4.7 `template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::count` [protected]

Definition at line 65 of file [Delay.h](#).

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

- [Delay.h](#)

## 7.12 DSG::DPW::DPW\_Differentiator< order > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the [DPW](#) Algorithm.

```
#include <DPW.h>
```

## Public Member Functions

- [DPW\\_Differentiator](#) ()

### 7.12.1 Detailed Description

template<unsigned order>class DSG::DPW::DPW\_Differentiator< order >

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the [DPW](#) Algorithm.

**Todo** Fix [DSG::DPW::DPW\\_Differentiator](#) algorithms for orders 3-6

Definition at line 79 of file [DPW.h](#).

### 7.12.2 Constructor & Destructor Documentation

7.12.2.1 template<unsigned order> DSG::DPW::DPW\_Differentiator< order >::DPW\_Differentiator ( )  
[inline]

Definition at line 81 of file [DPW.h](#).

```
00081             {
00082             DSG::StaticAssertBounds<1, 6,order>();//order must be 1-6
00083             }
```

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

- [DPW.h](#)

## 7.13 DSG::DPW::DPW\_Differentiator< 1 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 1st order [DPW](#) Algorithm.

#include <DPW.h>

## Public Member Functions

- [DSG::DSGSample operator\(\)](#) (DSG::DSGSample const &signal, [DSG::DSGSample](#) const &dt)

### 7.13.1 Detailed Description

template<>class DSG::DPW::DPW\_Differentiator< 1 >

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 1st order [DPW](#) Algorithm.

Definition at line 87 of file [DPW.h](#).

### 7.13.2 Member Function Documentation

7.13.2.1 [DSG::DSGSample](#) DSG::DPW::DPW\_Differentiator< 1 >::operator() ( [DSG::DSGSample](#) const & *signal*,  
[DSG::DSGSample](#) const & *dt* ) [inline]

Definition at line 89 of file [DPW.h](#).

```

00089                                     {
00090                                     return signal;
00091                                     }

```

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

- [DPW.h](#)

## 7.14 DSG::DPW::DPW\_Differentiator< 2 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 2nd order [DPW](#) Algorithm.

```
#include <DPW.h>
```

### Public Member Functions

- [DSG::DSGSample operator\(\)](#) ([DSG::DSGSample](#) const &signal, [DSG::DSGSample](#) const &dt)

### Protected Attributes

- [DSG::DSGSample output](#)
- [DSG::DSGSample \\_delay](#)

#### 7.14.1 Detailed Description

```
template<>class DSG::DPW::DPW_Differentiator< 2 >
```

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 2nd order [DPW](#) Algorithm.

Definition at line 95 of file [DPW.h](#).

#### 7.14.2 Member Function Documentation

**7.14.2.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 2 >::operator\(\)](#) ([DSG::DSGSample](#) const & *signal*, [DSG::DSGSample](#) const & *dt*) [[inline](#)]

Definition at line 97 of file [DPW.h](#).

```

00097                                     {
00098                                     output = (signal - _delay)/(4.0 * dt);
00099                                     _delay = signal;
00100                                     return output;
00101                                     }

```

#### 7.14.3 Member Data Documentation

**7.14.3.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 2 >::\\_delay](#) [[protected](#)]

Definition at line 104 of file [DPW.h](#).

**7.14.3.2** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 2 >::output](#) [[protected](#)]

Definition at line 103 of file [DPW.h](#).

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

- [DPW.h](#)

## 7.15 DSG::DPW::DPW\_Differentiator< 3 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 3rd order [DPW](#) Algorithm.

```
#include <DPW.h>
```

### Public Member Functions

- [DSG::DSGSample operator\(\)](#) ([DSG::DSGSample](#) const &signal, [DSG::DSGSample](#) const &dt)

### Protected Attributes

- [DSG::DSGSample output](#)
- [DSG::DSGSample \\_delay](#) [2]

#### 7.15.1 Detailed Description

```
template<>class DSG::DPW::DPW_Differentiator< 3 >
```

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 3rd order [DPW](#) Algorithm.

Definition at line 108 of file [DPW.h](#).

#### 7.15.2 Member Function Documentation

**7.15.2.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 3 >::operator\(\)](#) ([DSG::DSGSample](#) const & *signal*, [DSG::DSGSample](#) const & *dt*) [[inline](#)]

Definition at line 110 of file [DPW.h](#).

```
00110                                     {
00111         output = (signal - _delay[0]);
00112         output -= (_delay[0] - _delay[1]);
00113         output /= (24.*DSG::Pow<2>(dt));
00114         _delay[1]=_delay[0];
00115         _delay[0]=signal;
00116         return output;
00117     }
```

#### 7.15.3 Member Data Documentation

**7.15.3.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 3 >::\\_delay](#)[2] [[protected](#)]

Definition at line 120 of file [DPW.h](#).

**7.15.3.2** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 3 >::output](#) [[protected](#)]

Definition at line 119 of file [DPW.h](#).

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

- [DPW.h](#)

## 7.16 DSG::DPW::DPW\_Differentiator< 4 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 4th order [DPW](#) Algorithm.

```
#include <DPW.h>
```

### Public Member Functions

- [DSG::DSGSample operator\(\)](#) ([DSG::DSGSample](#) const &signal, [DSG::DSGSample](#) const &dt)

### Protected Attributes

- [DSG::DSGSample output](#)
- [DSG::DSGSample \\_delay](#) [3]

#### 7.16.1 Detailed Description

```
template<>class DSG::DPW::DPW_Differentiator< 4 >
```

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 4th order [DPW](#) Algorithm.

Definition at line [124](#) of file [DPW.h](#).

#### 7.16.2 Member Function Documentation

**7.16.2.1 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 4 >::operator() ( DSG::DSGSample const & signal, DSG::DSGSample const & dt ) [inline]**

Definition at line [126](#) of file [DPW.h](#).

```
00126                                     {
00127         output = (signal - _delay[0]);
00128         output -= (_delay[0] - _delay[1]);
00129         output -= (_delay[1] - _delay[2]);
00130         output /= 144*DSG::Pow<3>(dt);
00131         _delay[2]=_delay[1];
00132         _delay[1]=_delay[0];
00133         _delay[0]=signal;
00134         return output;
00135     }
```

#### 7.16.3 Member Data Documentation

**7.16.3.1 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 4 >::\_delay[3] [protected]**

Definition at line [138](#) of file [DPW.h](#).

**7.16.3.2 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 4 >::output [protected]**

Definition at line [137](#) of file [DPW.h](#).

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

- [DPW.h](#)

## 7.17 DSG::DPW::DPW\_Differentiator< 5 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 5th order [DPW](#) Algorithm.

```
#include <DPW.h>
```

### Public Member Functions

- [DSG::DSGSample operator\(\)](#) ([DSG::DSGSample](#) const &signal, [DSG::DSGSample](#) const &dt)

### Protected Attributes

- [DSG::DSGSample output](#)
- [DSG::DSGSample \\_delay](#) [4]

#### 7.17.1 Detailed Description

```
template<>class DSG::DPW::DPW_Differentiator< 5 >
```

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 5th order [DPW](#) Algorithm.

Definition at line 142 of file [DPW.h](#).

#### 7.17.2 Member Function Documentation

**7.17.2.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 5 >::operator\(\)](#) ([DSG::DSGSample](#) const & *signal*, [DSG::DSGSample](#) const & *dt*) [[inline](#)]

Definition at line 144 of file [DPW.h](#).

```
00144                                     {
00145         output = (signal - _delay[0]);
00146         output -= (_delay[0] - _delay[1]);
00147         output -= (_delay[1] - _delay[2]);
00148         output -= (_delay[2] - _delay[3]);
00149         output /= 960*DSG::Pow<4>(dt);
00150         _delay[3]=_delay[2];
00151         _delay[2]=_delay[1];
00152         _delay[1]=_delay[0];
00153         _delay[0]=signal;
00154         return output;
00155     }
```

#### 7.17.3 Member Data Documentation

**7.17.3.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 5 >::\\_delay](#)[4] [[protected](#)]

Definition at line 158 of file [DPW.h](#).

**7.17.3.2** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 5 >::output](#) [[protected](#)]

Definition at line 157 of file [DPW.h](#).

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

- [DPW.h](#)



## 7.18 DSG::DPW::DPW\_Differentiator< 6 > Class Template Reference

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 6th order [DPW](#) Algorithm.

```
#include <DPW.h>
```

### Public Member Functions

- [DSG::DSGSample operator\(\)](#) ([DSG::DSGSample](#) const &signal, [DSG::DSGSample](#) const &dt)

### Protected Attributes

- [DSG::DSGSample output](#)
- [DSG::DSGSample \\_delay](#) [5]

#### 7.18.1 Detailed Description

```
template<>class DSG::DPW::DPW_Differentiator< 6 >
```

[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 6th order [DPW](#) Algorithm.

Definition at line 162 of file [DPW.h](#).

#### 7.18.2 Member Function Documentation

**7.18.2.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 6 >::operator\(\)](#) ([DSG::DSGSample](#) const & *signal*, [DSG::DSGSample](#) const & *dt* ) [\[inline\]](#)

Definition at line 164 of file [DPW.h](#).

```
00164                                     {
00165         output = (signal - _delay[0]);
00166         output -= (_delay[0] - _delay[1]);
00167         output -= (_delay[1] - _delay[2]);
00168         output -= (_delay[2] - _delay[3]);
00169         output -= (_delay[3] - _delay[4]);
00170         output /= 7200*DSG::Pow<5>(dt);
00171         _delay[4]=_delay[3];
00172         _delay[3]=_delay[2];
00173         _delay[2]=_delay[1];
00174         _delay[1]=_delay[0];
00175         _delay[0]=signal;
00176         return output;
00177     }
```

#### 7.18.3 Member Data Documentation

**7.18.3.1** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 6 >::\\_delay](#)[5] [\[protected\]](#)

Definition at line 180 of file [DPW.h](#).

**7.18.3.2** [DSG::DSGSample DSG::DPW::DPW\\_Differentiator< 6 >::output](#) [\[protected\]](#)

Definition at line 179 of file [DPW.h](#).

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

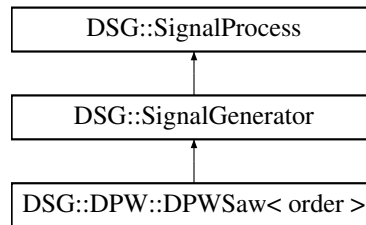
- [DPW.h](#)

## 7.19 DSG::DPW::DPWSaw< order > Class Template Reference

[DSG::DPW::DPWSaw](#) - Sawtooth Generator using the Nth Order [DPW](#) algorithm.

```
#include <DPWSaw.h>
```

Inheritance diagram for [DSG::DPW::DPWSaw< order >](#):



### Public Member Functions

- [DPWSaw](#) ()
- [DPWSaw](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~DPWSaw](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

### Protected Attributes

- [DSG::DSGSample \\_register](#)
- [DSG::DPW::DPW\\_Differentiator](#)  
< order > [\\_diff](#)

### Additional Inherited Members

#### 7.19.1 Detailed Description

```
template<unsigned order>class DSG::DPW::DPWSaw< order >
```

[DSG::DPW::DPWSaw](#) - Sawtooth Generator using the Nth Order [DPW](#) algorithm.

Definition at line 34 of file [DPWSaw.h](#).

#### 7.19.2 Constructor & Destructor Documentation

7.19.2.1 `template<unsigned order> DSG::DPW::DPWSaw< order >::DPWSaw ( ) [inline]`

Definition at line 36 of file [DPWSaw.h](#).

```

00036         :DSG::SignalGenerator(),_register(0){
00037             DSG::StaticAssertBounds<1, 6,order>();
00038     }
```

**7.19.2.2** `template<unsigned order> DSG::DPW::DPWSaw< order >::DPWSaw ( DSG::DSGFrequency const & frequency, DSG::DSGPhase const & offset ) [inline]`

Definition at line 39 of file [DPWSaw.h](#).

```
00039 :DSG::SignalGenerator(frequency,offset),_register(0){
      DSG::StaticAssertBounds<1, 6,order>();}
```

**7.19.2.3** `template<unsigned order> virtual DSG::DPW::DPWSaw< order >::~~DPWSaw ( ) [inline], [virtual]`

Definition at line 40 of file [DPWSaw.h](#).

```
00040 {}
```

## 7.19.3 Member Function Documentation

**7.19.3.1** `template<unsigned order> virtual bool DSG::DPW::DPWSaw< order >::Perform ( DSG::DSGSample & signal ) [inline],[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 41 of file [DPWSaw.h](#).

```
00041                                     {
00042                                     //trivial saw ramping from -1 to 1
00043                                     _register = _phasor;
00044                                     _register-=0.5;
00045                                     _register*=2.0;
00046                                     /*-----*/
00047                                     //DPW algorithm
00048                                     //polynomial shaping
00049                                     _register=DSG::DPW::DPW_Polynomial<order>(_register);
00050                                     //differentiating
00051                                     signal = _diff(_register,_dt);
00052                                     /*-----*/
00053                                     //signal = DSG::EnforceBounds<-1, 1>(signal);
00054                                     //advance phase
00055                                     step();
00056                                     return true;
00057                                     }
```

**7.19.3.2** `template<unsigned order> virtual bool DSG::DPW::DPWSaw< order >::Perform ( DSG::RingBuffer & signal ) [inline],[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 58 of file [DPWSaw.h](#).

```
00058                                     {
00059                                     signal.Flush();
00060                                     while (!signal.Full()) {
00061                                         if (Perform(_storage)) {
00062                                             if(signal.Write(_storage)){
00063                                                 }else return false;
00064                                             }else return false;
00065                                         }return true;
00066                                     }
```

## 7.19.4 Member Data Documentation

**7.19.4.1** `template<unsigned order> DSG::DPW::DPW_Differentiator<order> DSG::DPW::DPWSaw< order >::_diff [protected]`

Definition at line 69 of file [DPWSaw.h](#).

7.19.4.2 `template<unsigned order> DSG::DSGSample DSG::DPW::DPWSaw< order >::_register`  
`[protected]`

Definition at line 68 of file [DPWSaw.h](#).

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

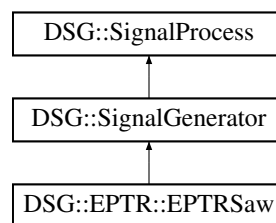
- [DPWSaw.h](#)

## 7.20 DSG::EPTR::EPTR Saw Class Reference

[DSG::EPTR::EPTR Saw](#)-Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm.

```
#include <EPTR Saw.h>
```

Inheritance diagram for DSG::EPTR::EPTR Saw:



### Public Member Functions

- [EPTR Saw](#) ()
- [EPTR Saw](#) (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual [~EPTR Saw](#) ()
- virtual bool [Perform](#) (DSG::DSGSample &signal)
- virtual bool [Perform](#) (DSG::RingBuffer &signal)

### Protected Attributes

- [DSG::DSGSample \\_register](#)

### Additional Inherited Members

#### 7.20.1 Detailed Description

[DSG::EPTR::EPTR Saw](#)-Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm.

**Todo** Test and Possibly Re-Write [DSG::EPTR::EPTR Saw](#) algorithm

Definition at line 35 of file [EPTR Saw.h](#).

#### 7.20.2 Constructor & Destructor Documentation

##### 7.20.2.1 DSG::EPTR::EPTR Saw::EPTR Saw ( )

Definition at line 25 of file [EPTR Saw.cpp](#).

```
00025 :DSG::SignalGenerator() {}
```

### 7.20.2.2 DSG::EPTR::EPTRsaw::EPTRsaw ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [EPTRsaw.cpp](#).

```
00026 :DSG::SignalGenerator(frequency,offset){}
```

### 7.20.2.3 DSG::EPTR::EPTRsaw::~~EPTRsaw ( ) [virtual]

Definition at line 27 of file [EPTRsaw.cpp](#).

```
00027 {}
```

## 7.20.3 Member Function Documentation

### 7.20.3.1 bool DSG::EPTR::EPTRsaw::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 45 of file [EPTRsaw.h](#).

```
00045                                     {
00046 #ifdef __APPLE__
00047 #warning Untested For Aliasing DSG::EPTR::EPTRsaw::Perform()
00048 #endif
00049     //generate trivial saw
00050     _register = _phasor;
00051     _register+=0.5;
00052     if (_register>1.0) {
00053         --_register;
00054     }
00055     _register-=0.5;
00056     _register*=2.0;
00057     if (_register > 1.0-_dt) {
00058         //transition region detected
00059         //apply eptr correction
00060         signal = _register - (_register/_dt) + (1.0/
00061 _dt) -1;
00062     }else{
00063         signal = _register;
00064     }
00065     step();//avance phase
00066     return true;
00067 }
```

### 7.20.3.2 bool DSG::EPTR::EPTRsaw::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 67 of file [EPTRsaw.h](#).

```
00067                                     {
00068     signal.Flush();
00069     while (!signal.Full()) {
00070         if (Perform(_storage)) {
00071             if(signal.Write(_storage)){
00072                 }else return false;
00073             }else return false;
00074         }return true;
00075     }
```

## 7.20.4 Member Data Documentation

### 7.20.4.1 DSG::DSGSample DSG::EPTR::EPTRsaw::\_register [protected]

Definition at line 43 of file [EPTRsaw.h](#).

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

- [EPTRsaw.h](#)
- [EPTRsaw.cpp](#)

## 7.21 DSG::Factorial< N > Struct Template Reference

[DSG::Factorial](#) - Compute integer factorial.

```
#include <DSGMath.h>
```

### Public Types

- enum { [value](#) = N \* Factorial<N-1>::value }

#### 7.21.1 Detailed Description

```
template<unsigned long N>struct DSG::Factorial< N >
```

[DSG::Factorial](#) - Compute integer factorial.

Definition at line 36 of file [DSGMath.h](#).

#### 7.21.2 Member Enumeration Documentation

7.21.2.1 template<unsigned long N> anonymous enum

Enumerator

***value***

Definition at line 37 of file [DSGMath.h](#).

```
00037 {value = N * Factorial<N-1>::value};
```

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

- [DSGMath.h](#)

## 7.22 DSG::Factorial< 0 > Struct Template Reference

[DSG::Factorial](#) - Compute integer factorial.

```
#include <DSGMath.h>
```

### Public Types

- enum { [value](#) = 1 }

#### 7.22.1 Detailed Description

```
template<>struct DSG::Factorial< 0 >
```

[DSG::Factorial](#) - Compute integer factorial.

Definition at line 41 of file [DSGMath.h](#).

## 7.22.2 Member Enumeration Documentation

### 7.22.2.1 anonymous enum

Enumerator

**value**

Definition at line 42 of file [DSGMath.h](#).

```
00042 { value = 1 };
```

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

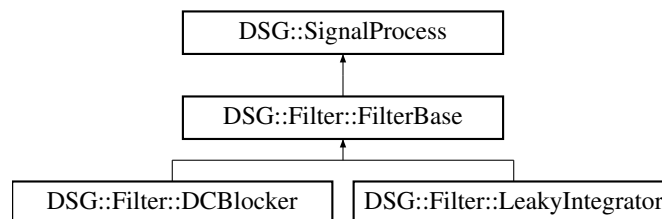
- [DSGMath.h](#)

## 7.23 DSG::Filter::FilterBase Class Reference

[DSG::Filter::FilterBase](#) - [Filter](#) Base Class, implements interface for cutoff frequency.

```
#include <Filter.h>
```

Inheritance diagram for [DSG::Filter::FilterBase](#):



### Public Member Functions

- [FilterBase](#) ()
- virtual [~FilterBase](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual bool [Cutoff](#) ([DSG::DSGFrequency](#) const &cutoff)

### Protected Attributes

- [DSG::DSGSample \\_temp](#)
- unsigned long [count](#)

### 7.23.1 Detailed Description

[DSG::Filter::FilterBase](#) - [Filter](#) Base Class, implements interface for cutoff frequency.

Definition at line 34 of file [Filter.h](#).

## 7.23.2 Constructor & Destructor Documentation

### 7.23.2.1 DSG::Filter::FilterBase::FilterBase ( )

Definition at line 25 of file [Filter.cpp](#).

```
00025 : _temp(0), count(0) {}
```

### 7.23.2.2 DSG::Filter::FilterBase::~~FilterBase ( ) [virtual]

Definition at line 26 of file [Filter.cpp](#).

```
00026 {}
```

## 7.23.3 Member Function Documentation

### 7.23.3.1 bool DSG::Filter::FilterBase::Cutoff ( DSG::DSGFrequency const & *cutoff* ) [inline], [virtual]

Reimplemented in [DSG::Filter::LeakyIntegrator](#).

Definition at line 60 of file [Filter.h](#).

```
00060                                     {
00061         return false;
00062     }
```

### 7.23.3.2 bool DSG::Filter::FilterBase::Perform ( DSG::DSGSample & *signal* ) [inline], [virtual]

Implements [DSG::SignalProcess](#).

Reimplemented in [DSG::Filter::LeakyIntegrator](#), and [DSG::Filter::DCBlocker](#).

Definition at line 45 of file [Filter.h](#).

```
00045                                     {
00046         return true;
00047     }
```

### 7.23.3.3 bool DSG::Filter::FilterBase::Perform ( DSG::RingBuffer & *signal* ) [inline], [virtual]

Implements [DSG::SignalProcess](#).

Reimplemented in [DSG::Filter::LeakyIntegrator](#), and [DSG::Filter::DCBlocker](#).

Definition at line 48 of file [Filter.h](#).

```
00048                                     {
00049         if (!signal.Empty()) {
00050             count = signal.Count();
00051             while (count-- > 0) {
00052                 if (signal.Read(_temp)) {
00053                     if (Perform(_temp)) {
00054                         signal.Write(_temp);
00055                     } else return false;
00056                 } else return false;
00057             } return true;
00058         } else return false;
00059     }
```



### 7.23.4 Member Data Documentation

#### 7.23.4.1 DSG::DSGSample DSG::Filter::FilterBase::\_temp [protected]

Definition at line 42 of file [Filter.h](#).

#### 7.23.4.2 unsigned long DSG::Filter::FilterBase::count [protected]

Definition at line 43 of file [Filter.h](#).

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

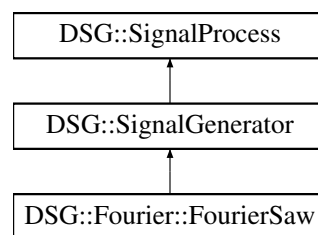
- [Filter.h](#)
- [Filter.cpp](#)

## 7.24 DSG::Fourier::FourierSaw Class Reference

[DSG::Fourier::FourierSaw](#) - [Fourier](#) Series Sawtooth Wave Generator.

```
#include <FourierSaw.h>
```

Inheritance diagram for [DSG::Fourier::FourierSaw](#):



### Public Member Functions

- [FourierSaw](#) ()
- [FourierSaw](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~FourierSaw](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ([DSG::DSGFrequency](#) const &value)

### Protected Attributes

- unsigned long [\\_h](#)
- const double [\\_a](#)
- double [phs](#)
- double [value](#)
- int [i](#)

### Additional Inherited Members

#### 7.24.1 Detailed Description

[DSG::Fourier::FourierSaw](#) - [Fourier](#) Series Sawtooth Wave Generator.

Definition at line 34 of file [FourierSaw.h](#).

## 7.24.2 Constructor & Destructor Documentation

### 7.24.2.1 DSG::Fourier::FourierSaw::FourierSaw ( )

Definition at line 25 of file [FourierSaw.cpp](#).

```
00025 :DSG::SignalGenerator(),_a(1.7/PI),phs(0),value(0),
      i(0){}
```

### 7.24.2.2 DSG::Fourier::FourierSaw::FourierSaw ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [FourierSaw.cpp](#).

```
00026 :
      DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),
      value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
```

### 7.24.2.3 DSG::Fourier::FourierSaw::~~FourierSaw ( ) [virtual]

Definition at line 29 of file [FourierSaw.cpp](#).

```
00029 {}
```

## 7.24.3 Member Function Documentation

### 7.24.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierSaw::Frequency ( DSG::DSGFrequency const & *value* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 69 of file [FourierSaw.h](#).

```
00069 :
00070     _frequency = value;
00071     _dt = _frequency/DSG::SampleRate();
00072     _h = MaxHarms(_frequency);
00073     return _frequency;
00074 }
```

### 7.24.3.2 bool DSG::Fourier::FourierSaw::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 49 of file [FourierSaw.h](#).

```
00049 :
00050     // _h Sine Calls Per Sample where _h is theoretically nyquist / frequency
00051     value=DSG::Sin(_phasor);
00052     for (i=2; i<_h; ++i) {
00053         value += (1.0/i) * DSG::Sin(_phasor*i);
00054     }
00055     value*=_a;
00056     signal = value;
00057     step();
00058     return true;
00059 }
```

7.24.3.3 `bool DSG::Fourier::FourierSaw::Perform ( DSG::RingBuffer & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 60 of file [FourierSaw.h](#).

```

00060                                     {
00061         signal.Flush();
00062         while (!signal.Full()) {
00063             if (Perform(_storage)) {
00064                 if(signal.Write(_storage)){
00065                     }else return false;
00066                 }else return false;
00067             }return true;
00068         }

```

## 7.24.4 Member Data Documentation

7.24.4.1 `const double DSG::Fourier::FourierSaw::_a` `[protected]`

Definition at line 44 of file [FourierSaw.h](#).

7.24.4.2 `unsigned long DSG::Fourier::FourierSaw::_h` `[protected]`

Definition at line 43 of file [FourierSaw.h](#).

7.24.4.3 `int DSG::Fourier::FourierSaw::i` `[protected]`

Definition at line 47 of file [FourierSaw.h](#).

7.24.4.4 `double DSG::Fourier::FourierSaw::phs` `[protected]`

Definition at line 45 of file [FourierSaw.h](#).

7.24.4.5 `double DSG::Fourier::FourierSaw::value` `[protected]`

Definition at line 46 of file [FourierSaw.h](#).

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

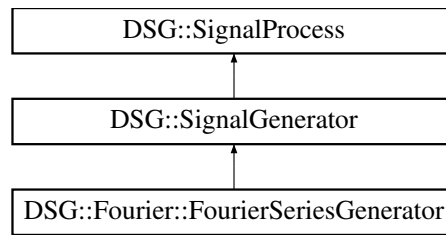
- [FourierSaw.h](#)
- [FourierSaw.cpp](#)

## 7.25 DSG::Fourier::FourierSeriesGenerator Class Reference

[DSG::Fourier::FourierSeriesGenerator](#) - Generates a wave form using a user specified [Fourier](#) Series.

```
#include <FourierSeries.h>
```

Inheritance diagram for [DSG::Fourier::FourierSeriesGenerator](#):



## Public Types

- typedef std::vector< [Harmonic](#) > [FourierSeries](#)

## Public Member Functions

- [FourierSeriesGenerator](#) ()
- [FourierSeriesGenerator](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~FourierSeriesGenerator](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- void [Series](#) ([FourierSeries](#) const &series)
- [FourierSeries](#) & [Series](#) ()

## Protected Attributes

- [FourierSeries \\_series](#)
- [DSG::DSGSample](#) value

## Additional Inherited Members

### 7.25.1 Detailed Description

[DSG::Fourier::FourierSeriesGenerator](#) - Generates a wave form using a user specified [Fourier](#) Series.

Definition at line 48 of file [FourierSeries.h](#).

### 7.25.2 Member Typedef Documentation

#### 7.25.2.1 typedef std::vector<Harmonic> DSG::Fourier::FourierSeriesGenerator::FourierSeries

Definition at line 50 of file [FourierSeries.h](#).

### 7.25.3 Constructor & Destructor Documentation

#### 7.25.3.1 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator ( )

Definition at line 45 of file [FourierSeries.cpp](#).

```
00045 :DSG::SignalGenerator() {}
```

### 7.25.3.2 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 46 of file [FourierSeries.cpp](#).

```
00046 :DSG::SignalGenerator(frequency,offset){}
```

### 7.25.3.3 DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator ( ) [virtual]

Definition at line 47 of file [FourierSeries.cpp](#).

```
00047 {}
```

## 7.25.4 Member Function Documentation

### 7.25.4.1 bool DSG::Fourier::FourierSeriesGenerator::Perform ( DSG::DSGSample & *signal* ) [inline], [virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 62 of file [FourierSeries.h](#).

```
00062                                     {
00063     value = _phasor;
00064     signal=0;
00065     for (auto i = _series.begin(); i!=_series.end(); ++i) {
00066         signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
00067     }
00068     step();
00069     return true;
00070 }
```

### 7.25.4.2 bool DSG::Fourier::FourierSeriesGenerator::Perform ( DSG::RingBuffer & *signal* ) [inline], [virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 71 of file [FourierSeries.h](#).

```
00071                                     {
00072     signal.Flush();
00073     while (!signal.Full()) {
00074         if (Perform(_storage)) {
00075             if(signal.Write(_storage)){
00076                 }else return false;
00077             }else return false;
00078         }return true;
00079     }
```

### 7.25.4.3 void DSG::Fourier::FourierSeriesGenerator::Series ( FourierSeries const & *series* ) [inline]

Definition at line 80 of file [FourierSeries.h](#).

```
00080     {
00081         _series = series;
00082     }
```

#### 7.25.4.4 DSG::Fourier::FourierSeriesGenerator::FourierSeries & DSG::Fourier::FourierSeriesGenerator::Series ( ) [inline]

Definition at line 83 of file [FourierSeries.h](#).

```
00083                                     {
00084         return _series;
00085     }
```

### 7.25.5 Member Data Documentation

#### 7.25.5.1 FourierSeries DSG::Fourier::FourierSeriesGenerator::\_series [protected]

Definition at line 59 of file [FourierSeries.h](#).

#### 7.25.5.2 DSG::DSGSample DSG::Fourier::FourierSeriesGenerator::value [protected]

Definition at line 60 of file [FourierSeries.h](#).

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

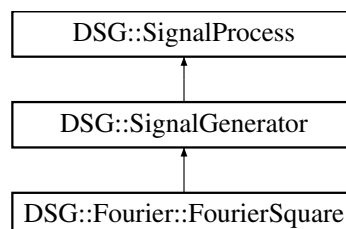
- [FourierSeries.h](#)
- [FourierSeries.cpp](#)

## 7.26 DSG::Fourier::FourierSquare Class Reference

[DSG::Fourier::FourierSquare](#) - [Fourier](#) Series Square Wave Generator.

```
#include <FourierSquare.h>
```

Inheritance diagram for DSG::Fourier::FourierSquare:



### Public Member Functions

- [FourierSquare](#) ( )
- [FourierSquare](#) (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual [~FourierSquare](#) ( )
- virtual bool [Perform](#) (DSG::DSGSample &signal)
- virtual bool [Perform](#) (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & [Frequency](#) (DSG::DSGFrequency const &value)

### Protected Attributes

- unsigned long [\\_h](#)
- const double [\\_a](#)
- double [phs](#)

- double [value](#)
- int [i](#)

## Additional Inherited Members

### 7.26.1 Detailed Description

[DSG::Fourier::FourierSquare](#) - [Fourier](#) Series Square Wave Generator.

Definition at line 34 of file [FourierSquare.h](#).

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 DSG::Fourier::FourierSquare::FourierSquare ( )

Definition at line 25 of file [FourierSquare.cpp](#).

```
00025 :DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),
      i(0){}
```

#### 7.26.2.2 DSG::Fourier::FourierSquare::FourierSquare ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [FourierSquare.cpp](#).

```
00026 :
      DSG::SignalGenerator(frequency,offset),_a(3.6/PI),phs(0),
      value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
```

#### 7.26.2.3 DSG::Fourier::FourierSquare::~~FourierSquare ( ) [virtual]

Definition at line 29 of file [FourierSquare.cpp](#).

```
00029 {}
```

### 7.26.3 Member Function Documentation

#### 7.26.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierSquare::Frequency ( DSG::DSGFrequency const & *value* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 69 of file [FourierSquare.h](#).

```
00069 :
00070     _frequency = value;
00071     _dt = _frequency/DSG::SampleRate();
00072     _h = MaxHarms(_frequency);
00073     return _frequency;
00074 }
```

### 7.26.3.2 `bool DSG::Fourier::FourierSquare::Perform ( DSG::DSGSample & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 49 of file [FourierSquare.h](#).

```

00049                                     {
00050         //(_h/2)+1 Sine Calls Per Sample
00051         value=DSG::Sin(_phasor);//i=1
00052         for (i=3; i<_h; i+=2) { //i=3..5..7..
00053             value += (1.0/i) * DSG::Sin(_phasor*i);
00054         }
00055         value*=_a;
00056         signal = value;
00057         step();
00058         return true;
00059     }

```

### 7.26.3.3 `bool DSG::Fourier::FourierSquare::Perform ( DSG::RingBuffer & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 60 of file [FourierSquare.h](#).

```

00060                                     {
00061         signal.Flush();
00062         while (!signal.Full()) {
00063             if (Perform(_storage)) {
00064                 if(signal.Write(_storage)){
00065                     }else return false;
00066                 }else return false;
00067             }return true;
00068         }

```

## 7.26.4 Member Data Documentation

### 7.26.4.1 `const double DSG::Fourier::FourierSquare::_a` `[protected]`

Definition at line 44 of file [FourierSquare.h](#).

### 7.26.4.2 `unsigned long DSG::Fourier::FourierSquare::_h` `[protected]`

Definition at line 43 of file [FourierSquare.h](#).

### 7.26.4.3 `int DSG::Fourier::FourierSquare::i` `[protected]`

Definition at line 47 of file [FourierSquare.h](#).

### 7.26.4.4 `double DSG::Fourier::FourierSquare::phs` `[protected]`

Definition at line 45 of file [FourierSquare.h](#).

### 7.26.4.5 `double DSG::Fourier::FourierSquare::value` `[protected]`

Definition at line 46 of file [FourierSquare.h](#).

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

- [FourierSquare.h](#)
- [FourierSquare.cpp](#)

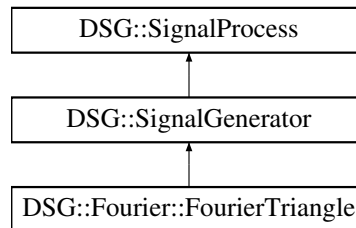


## 7.27 DSG::Fourier::FourierTriangle Class Reference

[DSG::Fourier::FourierTriangle](#) - [Fourier](#) Series Triangle Wave Generator.

```
#include <FourierTriangle.h>
```

Inheritance diagram for DSG::Fourier::FourierTriangle:



### Public Member Functions

- [FourierTriangle](#) ()
- [FourierTriangle](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~FourierTriangle](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ([DSG::DSGFrequency](#) const &value)

### Protected Attributes

- unsigned long [\\_h](#)
- const double [\\_a](#)
- double [phs](#)
- double [value](#)
- int [i](#)

### Additional Inherited Members

#### 7.27.1 Detailed Description

[DSG::Fourier::FourierTriangle](#) - [Fourier](#) Series Triangle Wave Generator.

Definition at line 34 of file [FourierTriangle.h](#).

#### 7.27.2 Constructor & Destructor Documentation

##### 7.27.2.1 DSG::Fourier::FourierTriangle::FourierTriangle ( )

Definition at line 25 of file [FourierTriangle.cpp](#).

```
00025 :DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),
      value(0),i(0){}
```

### 7.27.2.2 DSG::Fourier::FourierTriangle::FourierTriangle ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [FourierTriangle.cpp](#).

```
00026                                     :
    DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),
    phs(0),value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
```

### 7.27.2.3 DSG::Fourier::FourierTriangle::~~FourierTriangle ( ) [virtual]

Definition at line 29 of file [FourierTriangle.cpp](#).

```
00029 {}
```

## 7.27.3 Member Function Documentation

### 7.27.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierTriangle::Frequency ( DSG::DSGFrequency const & *value* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 71 of file [FourierTriangle.h](#).

```
00071                                     {
00072     _frequency = value;
00073     _dt = _frequency/DSG::SampleRate();
00074     _h = MaxHarms(_frequency);
00075     return _frequency;
00076 }
```

### 7.27.3.2 bool DSG::Fourier::FourierTriangle::Perform ( DSG::DSGSample & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 49 of file [FourierTriangle.h](#).

```
00049                                     {
00050     //(_h/2)+1 Sine Calls Per Sample
00051     value=DSG::Sin(_phasor);//i=1
00052     double sgn = -1;
00053     for (i=3; i<_h; i+=2) { //i=3..5..7..
00054         value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*
i);
00055         sgn*=-1;
00056     }
00057     value*=_a;
00058     signal = value;
00059     step();
00060     return true;
00061 }
```

### 7.27.3.3 bool DSG::Fourier::FourierTriangle::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 62 of file [FourierTriangle.h](#).

```

00062                                     {
00063         signal.Flush();
00064         while (!signal.Full()) {
00065             if (Perform(_storage)) {
00066                 if(signal.Write(_storage)){
00067                     }else return false;
00068                 }else return false;
00069             }return true;
00070         }

```

## 7.27.4 Member Data Documentation

### 7.27.4.1 const double DSG::Fourier::FourierTriangle::\_a [protected]

Definition at line 44 of file [FourierTriangle.h](#).

### 7.27.4.2 unsigned long DSG::Fourier::FourierTriangle::\_h [protected]

Definition at line 43 of file [FourierTriangle.h](#).

### 7.27.4.3 int DSG::Fourier::FourierTriangle::i [protected]

Definition at line 47 of file [FourierTriangle.h](#).

### 7.27.4.4 double DSG::Fourier::FourierTriangle::phs [protected]

Definition at line 45 of file [FourierTriangle.h](#).

### 7.27.4.5 double DSG::Fourier::FourierTriangle::value [protected]

Definition at line 46 of file [FourierTriangle.h](#).

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

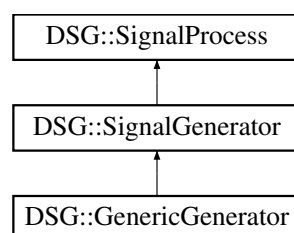
- [FourierTriangle.h](#)
- [FourierTriangle.cpp](#)

## 7.28 DSG::GenericGenerator Class Reference

[DSG::GenericGenerator](#) - Generator designed to use a stateless generator function such as [DSG::Sin\(\)](#)

```
#include <GenericGenerator.h>
```

Inheritance diagram for DSG::GenericGenerator:



## Public Member Functions

- [GenericGenerator](#) ()
- [GenericGenerator](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset, [DSG::DSGSample](#) (\*signalFunction)([DSG::DSGSample](#) const &))
- virtual [~GenericGenerator](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

## Protected Attributes

- [DSG::DSGSample](#) (\*\_callback) ([DSG::DSGSample](#) const &)

## Additional Inherited Members

### 7.28.1 Detailed Description

[DSG::GenericGenerator](#) - Generator designed to use a stateless generator function such as [DSG::Sin\(\)](#)

Definition at line 29 of file [GenericGenerator.h](#).

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 [DSG::GenericGenerator::GenericGenerator](#) ( )

Definition at line 25 of file [GenericGenerator.cpp](#).

```
00025 :DSG::SignalGenerator() {}
```

#### 7.28.2.2 [DSG::GenericGenerator::GenericGenerator](#) ( [DSG::DSGFrequency](#) const & frequency, [DSG::DSGPhase](#) const & offset, [DSG::DSGSample](#) (\*) ([DSG::DSGSample](#) const &) signalFunction )

Definition at line 26 of file [GenericGenerator.cpp](#).

```
00026 :DSG::SignalGenerator(frequency,offset),_callback(signalFunction) {}
```

#### 7.28.2.3 [DSG::GenericGenerator::~~GenericGenerator](#) ( ) [virtual]

Definition at line 27 of file [GenericGenerator.cpp](#).

```
00027 {}
```

### 7.28.3 Member Function Documentation

#### 7.28.3.1 bool [DSG::GenericGenerator::Perform](#) ( [DSG::DSGSample](#) & signal ) [inline],[virtual]

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 39 of file [GenericGenerator.h](#).

```
00039                                     {
00040         if (_callback!=nullptr) {
00041             signal = _callback(_phasor);
00042         }else signal = 0;
00043         step();
00044         return true;
00045     }
```

7.28.3.2 `bool DSG::GenericGenerator::Perform ( DSG::RingBuffer & signal )` `[inline]`, `[virtual]`

Reimplemented from [DSG::SignalGenerator](#).

Definition at line 46 of file [GenericGenerator.h](#).

```

00046                                     {
00047         signal.Flush();
00048         while (!signal.Full()) {
00049             if (Perform(_storage)) {
00050                 if (signal.Write(_storage)) {
00051                     }else return false;
00052                 }else return false;
00053             }return true;
00054         }

```

## 7.28.4 Member Data Documentation

7.28.4.1 `DSG::DSGSample(* DSG::GenericGenerator::_callback)(DSG::DSGSample const &)` `[protected]`

Definition at line 37 of file [GenericGenerator.h](#).

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

- [GenericGenerator.h](#)
- [GenericGenerator.cpp](#)

## 7.29 DSG::Fourier::Harmonic Class Reference

[DSG::Fourier::Harmonic](#) - Represents a single harmonic in a [Fourier](#) Series.

```
#include <FourierSeries.h>
```

### Public Member Functions

- [Harmonic](#) ()
- [Harmonic](#) ([DSG::DSGSample](#) const &ratio, [DSG::DSGSample](#) const &amplitude)
- virtual [~Harmonic](#) ()
- [DSG::DSGSample](#) const & [Ratio](#) () const
- [DSG::DSGSample](#) const & [Ratio](#) ([DSG::DSGSample](#) const &value)
- [DSG::DSGSample](#) const & [Amplitude](#) () const
- [DSG::DSGSample](#) const & [Amplitude](#) ([DSG::DSGSample](#) const &value)

### Protected Attributes

- [DSG::DSGSample \\_ratio](#)
- [DSG::DSGSample \\_amplitude](#)

### 7.29.1 Detailed Description

[DSG::Fourier::Harmonic](#) - Represents a single harmonic in a [Fourier](#) Series.

Definition at line 34 of file [FourierSeries.h](#).

## 7.29.2 Constructor & Destructor Documentation

### 7.29.2.1 DSG::Fourier::Harmonic::Harmonic ( )

Definition at line 25 of file [FourierSeries.cpp](#).

```
00025 :_ratio(0),_amplitude(0){}
```

### 7.29.2.2 DSG::Fourier::Harmonic::Harmonic ( DSG::DSGSample const & ratio, DSG::DSGSample const & amplitude )

Definition at line 26 of file [FourierSeries.cpp](#).

```
00026 :_ratio(ratio),_amplitude(amplitude){}
```

### 7.29.2.3 DSG::Fourier::Harmonic::~Harmonic ( ) [virtual]

Definition at line 27 of file [FourierSeries.cpp](#).

```
00027                                     {
00028     _ratio=0;
00029     _amplitude=0;
00030 }
```

## 7.29.3 Member Function Documentation

### 7.29.3.1 DSG::DSGSample const & DSG::Fourier::Harmonic::Amplitude ( ) const

Definition at line 38 of file [FourierSeries.cpp](#).

```
00038                                     {
00039     return _amplitude;
00040 }
```

### 7.29.3.2 DSG::DSGSample const & DSG::Fourier::Harmonic::Amplitude ( DSG::DSGSample const & value )

Definition at line 41 of file [FourierSeries.cpp](#).

```
00041                                     {
00042     _amplitude=value;
00043     return _amplitude;
00044 }
```

### 7.29.3.3 DSG::DSGSample const & DSG::Fourier::Harmonic::Ratio ( ) const

Definition at line 31 of file [FourierSeries.cpp](#).

```
00031                                     {
00032     return _ratio;
00033 }
```

### 7.29.3.4 DSG::DSGSample const & DSG::Fourier::Harmonic::Ratio ( DSG::DSGSample const & value )

Definition at line 34 of file [FourierSeries.cpp](#).

```
00034                                     {
00035     _ratio = value;
00036     return _ratio;
00037 }
```

### 7.29.4 Member Data Documentation

#### 7.29.4.1 DSG::DSGSample DSG::Fourier::Harmonic::\_amplitude [protected]

Definition at line 45 of file [FourierSeries.h](#).

#### 7.29.4.2 DSG::DSGSample DSG::Fourier::Harmonic::\_ratio [protected]

Definition at line 44 of file [FourierSeries.h](#).

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

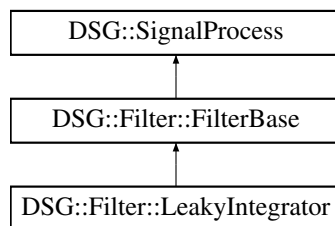
- [FourierSeries.h](#)
- [FourierSeries.cpp](#)

## 7.30 DSG::Filter::LeakyIntegrator Class Reference

[DSG::Filter::LeakyIntegrator](#) - Leaky integrator.

```
#include <Leaky.h>
```

Inheritance diagram for DSG::Filter::LeakyIntegrator:



### Public Member Functions

- [LeakyIntegrator](#) ()
- [LeakyIntegrator](#) (DSG::DSGFrequency const &cutoff)
- virtual [~LeakyIntegrator](#) ()
- virtual bool [Perform](#) (DSG::DSGSample &signal)
- virtual bool [Perform](#) (DSG::RingBuffer &signal)
- virtual bool [Cutoff](#) (DSG::DSGFrequency const &cutoff)

### Protected Attributes

- double [x1](#)
- double [y1](#)
- double [a](#)
- double [b](#)
- double [y](#)

### 7.30.1 Detailed Description

[DSG::Filter::LeakyIntegrator](#) - Leaky integrator.

Definition at line 34 of file [Leaky.h](#).

## 7.30.2 Constructor & Destructor Documentation

### 7.30.2.1 DSG::Filter::LeakyIntegrator::LeakyIntegrator ( )

Definition at line 25 of file [Leaky.cpp](#).

```
00025                                     :DSG::Filter::FilterBase() {
00026     x1=0;
00027     y1=0;
00028     a=0;
00029     b=0;
00030     y=0;
00031 }
```

### 7.30.2.2 DSG::Filter::LeakyIntegrator::LeakyIntegrator ( DSG::DSGFrequency const & cutoff )

Definition at line 32 of file [Leaky.cpp](#).

```
00032                                     :
00033     DSG::Filter::FilterBase() {
00034     x1=0;
00035     y1=0;
00036     a=0;
00037     b=0;
00038     y=0;
00039     Cutoff(cutoff);
00040 }
```

### 7.30.2.3 DSG::Filter::LeakyIntegrator::~LeakyIntegrator ( ) [virtual]

Definition at line 40 of file [Leaky.cpp](#).

```
00040                                     {
00041     x1=0;
00042     y1=0;
00043     a=0;
00044     b=0;
00045     y=0;
00046 }
```

## 7.30.3 Member Function Documentation

### 7.30.3.1 bool DSG::Filter::LeakyIntegrator::Cutoff ( DSG::DSGFrequency const & cutoff ) [inline], [virtual]

Reimplemented from [DSG::Filter::FilterBase](#).

Definition at line 65 of file [Leaky.h](#).

```
00065                                     {
00066     double Omega;
00067     x1 = y1 = 0.0;
00068     Omega = atan(PI * cutoff);
00069     a = -(1.0 - Omega) / (1.0 + Omega);
00070     b = (1.0 - b) / 2.0;
00071     return true;
00072 }
```

### 7.30.3.2 bool DSG::Filter::LeakyIntegrator::Perform ( DSG::DSGSample & signal ) [inline], [virtual]

Reimplemented from [DSG::Filter::FilterBase](#).

Definition at line 46 of file [Leaky.h](#).



```

00046                                     {
00047         y = b * (signal + x1) - a * y1;
00048         x1=signal;
00049         y1=y;
00050         signal=y;
00051         return true;
00052     }

```

### 7.30.3.3 bool DSG::Filter::LeakyIntegrator::Perform ( DSG::RingBuffer & signal ) [inline],[virtual]

Reimplemented from [DSG::Filter::FilterBase](#).

Definition at line 53 of file [Leaky.h](#).

```

00053                                     {
00054         if (!signal.Empty()) {
00055             count = signal.Count();
00056             while (count-- > 0) {
00057                 if(signal.Read(_temp)){
00058                     if (Perform(_temp)) {
00059                         signal.Write(_temp);
00060                     }else return false;
00061                 }else return false;
00062             }return true;
00063         }else return false;
00064     }

```

## 7.30.4 Member Data Documentation

### 7.30.4.1 double DSG::Filter::LeakyIntegrator::a [protected]

Definition at line 43 of file [Leaky.h](#).

### 7.30.4.2 double DSG::Filter::LeakyIntegrator::b [protected]

Definition at line 43 of file [Leaky.h](#).

### 7.30.4.3 double DSG::Filter::LeakyIntegrator::x1 [protected]

Definition at line 43 of file [Leaky.h](#).

### 7.30.4.4 double DSG::Filter::LeakyIntegrator::y [protected]

Definition at line 44 of file [Leaky.h](#).

### 7.30.4.5 double DSG::Filter::LeakyIntegrator::y1 [protected]

Definition at line 43 of file [Leaky.h](#).

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

- [Leaky.h](#)
- [Leaky.cpp](#)

## 7.31 DSG::LUT< element, size > Class Template Reference

[DSG::LUT](#) - Look Up Table.

```
#include <LUT.h>
```

## Public Types

- typedef element(\* [FillFunction](#) )(element)
- typedef element(\* [FillFunctionConstRef](#) )(element const &)

## Public Member Functions

- [LUT](#) ()
- [LUT](#) ([FillFunction](#) fill, double const &range=1.0)
- [LUT](#) ([FillFunctionConstRef](#) fill, double const &range=1.0)
- [~LUT](#) ()
- element const & [operator\[\]](#) (unsigned long const &index) const
- element & [operator\[\]](#) (unsigned long const &index)
- element const & [operator\(\)](#) (double const &x)
- unsigned long const & [Size](#) () const

## Protected Attributes

- element [\\_table](#) [size]
- const unsigned long [\\_size](#)
- double [phs](#)

### 7.31.1 Detailed Description

template<typename element, unsigned long size>class DSG::LUT< element, size >

[DSG::LUT](#) - Look Up Table.

Definition at line 33 of file [LUT.h](#).

### 7.31.2 Member Typedef Documentation

7.31.2.1 template<typename element, unsigned long size> typedef element(\* [DSG::LUT](#)< element, size >::[FillFunction](#))(element)

Definition at line 35 of file [LUT.h](#).

7.31.2.2 template<typename element, unsigned long size> typedef element(\* [DSG::LUT](#)< element, size >::[FillFunctionConstRef](#))(element const &)

Definition at line 36 of file [LUT.h](#).

### 7.31.3 Constructor & Destructor Documentation

7.31.3.1 template<typename element, unsigned long size> [DSG::LUT](#)< element, size >::[LUT](#) ( ) [inline]

Definition at line 37 of file [LUT.h](#).

```
00037 :_size(size){}
```

**7.31.3.2** `template<typename element, unsigned long size> DSG::LUT< element, size >::LUT ( FillFunction fill, double const & range = 1.0 ) [inline]`

Definition at line 38 of file [LUT.h](#).

```
00038                                     :_size(size){
00039         //range is the expected input range for the function
00040         //example would be 0-2pi or 0-1
00041         //would be provided a 2pi or 1
00042         //defaults to 1
00043         double step = range/(double)_size;
00044         phs = 0;
00045         for (int i=0; i<_size; ++i) {
00046             _table[i] = fill(phs);
00047             phs+=step;
00048         }
00049     }
```

**7.31.3.3** `template<typename element, unsigned long size> DSG::LUT< element, size >::LUT ( FillFunctionConstRef fill, double const & range = 1.0 ) [inline]`

Definition at line 50 of file [LUT.h](#).

```
00050                                     :_size(size){
00051         //range is the expected input range for the function
00052         //example would be 0-2pi or 0-1
00053         //would be provided a 2pi or 1
00054         //defaults to 1
00055         double step = range/_size;
00056         phs = 0;
00057         for (int i=0; i<_size; ++i) {
00058             _table[i] = fill(phs);
00059             phs+=step;
00060         }
00061     }
```

**7.31.3.4** `template<typename element, unsigned long size> DSG::LUT< element, size >::~~LUT ( ) [inline]`

Definition at line 62 of file [LUT.h](#).

```
00062 {}
```

## 7.31.4 Member Function Documentation

**7.31.4.1** `template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator() ( double const & x ) [inline]`

Definition at line 75 of file [LUT.h](#).

```
00075                                     {
00076         phs=x;
00077         //need range checking on x to ensure 0-1 range
00078         phs<0 ? phs = 1-(phs*-1):0;
00079         phs-=((int)phs);
00080         return this->_table[(unsigned) (phs*(this->_size-1))];
00081     }
```

**7.31.4.2** `template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator[] ( unsigned long const & index ) const [inline]`

Definition at line 63 of file [LUT.h](#).

```

00063                                     {
00064 #ifdef DEBUG
00065     assert(index<_size);
00066 #endif
00067     return _table[index];
00068 }

```

**7.31.4.3** `template<typename element, unsigned long size> element& DSG::LUT< element, size >::operator[] ( unsigned long const & index ) [inline]`

Definition at line 69 of file [LUT.h](#).

```

00069                                     {
00070 #ifdef DEBUG
00071     assert(index<_size);
00072 #endif
00073     return _table[index];
00074 }

```

**7.31.4.4** `template<typename element, unsigned long size> unsigned long const& DSG::LUT< element, size >::Size ( ) const [inline]`

Definition at line 82 of file [LUT.h](#).

```

00082                                     {
00083     return _size;
00084 }

```

## 7.31.5 Member Data Documentation

**7.31.5.1** `template<typename element, unsigned long size> const unsigned long DSG::LUT< element, size >::_size [protected]`

Definition at line 87 of file [LUT.h](#).

**7.31.5.2** `template<typename element, unsigned long size> element DSG::LUT< element, size >::_table[size] [protected]`

Definition at line 86 of file [LUT.h](#).

**7.31.5.3** `template<typename element, unsigned long size> double DSG::LUT< element, size >::phs [protected]`

Definition at line 88 of file [LUT.h](#).

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

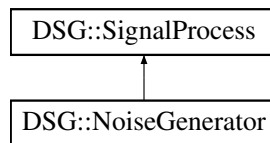
- [LUT.h](#)

## 7.32 DSG::NoiseGenerator Class Reference

[DSG::NoiseGenerator](#) - Generator that uses noise functions such as [DSG::White\(\)](#) to generate signal.

```
#include <NoiseGenerator.h>
```

Inheritance diagram for [DSG::NoiseGenerator](#):



## Public Member Functions

- [NoiseGenerator](#) ([DSGSample](#)(\*[StatelessFunction](#))([DSGSample](#)))
- virtual [~NoiseGenerator](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)

## Protected Attributes

- [DSGSample](#)(\* [\\_function](#) )([DSGSample](#))
- [DSG::DSGSample](#) [\\_storage](#)

### 7.32.1 Detailed Description

[DSG::NoiseGenerator](#) - Generator that uses noise functions such as [DSG::White\(\)](#) to generate signal.

Definition at line 29 of file [NoiseGenerator.h](#).

### 7.32.2 Constructor & Destructor Documentation

#### 7.32.2.1 DSG::NoiseGenerator::NoiseGenerator ( [DSGSample](#)(\*)([DSGSample](#)) *StatelessFunction* )

Definition at line 25 of file [NoiseGenerator.cpp](#).

```

00025                                     :
      DSG::SignalProcess() {
00026     _function = StatelessFunction;
00027 }
```

#### 7.32.2.2 DSG::NoiseGenerator::~~NoiseGenerator ( ) [virtual]

Definition at line 28 of file [NoiseGenerator.cpp](#).

```

00028 {}
```

### 7.32.3 Member Function Documentation

#### 7.32.3.1 bool DSG::NoiseGenerator::Perform ( [DSG::DSGSample](#) & *signal* ) [inline], [virtual]

Implements [DSG::SignalProcess](#).

Definition at line 39 of file [NoiseGenerator.h](#).

```

00039                                     {
00040     signal = _function(0);
00041     return true;
00042 }
```

### 7.32.3.2 bool DSG::NoiseGenerator::Perform ( DSG::RingBuffer & *signal* ) [inline],[virtual]

Implements [DSG::SignalProcess](#).

Definition at line 43 of file [NoiseGenerator.h](#).

```

00043                                     {
00044         signal.Flush();
00045         while (!signal.Full()) {
00046             if (Perform(_storage)) {
00047                 if(signal.Write(_storage)){
00048                     }else return false;
00049                 }else return false;
00050             }return true;
00051         }

```

## 7.32.4 Member Data Documentation

### 7.32.4.1 DSGSample(\* DSG::NoiseGenerator::\_function)(DSGSample) [protected]

Definition at line 36 of file [NoiseGenerator.h](#).

### 7.32.4.2 DSG::DSGSample DSG::NoiseGenerator::\_storage [protected]

Definition at line 37 of file [NoiseGenerator.h](#).

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

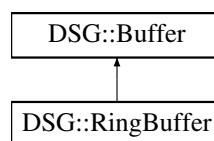
- [NoiseGenerator.h](#)
- [NoiseGenerator.cpp](#)

## 7.33 DSG::RingBuffer Class Reference

[DSG::RingBuffer](#) - Circular [Buffer](#) of Audio.

```
#include <RingBuffer.h>
```

Inheritance diagram for DSG::RingBuffer:



### Public Member Functions

- [RingBuffer](#) ()
- [RingBuffer](#) (const size\_t size)
- [RingBuffer](#) ([RingBuffer](#) &buffer)
- [RingBuffer](#) & operator= ([RingBuffer](#) &buffer)
- virtual ~[RingBuffer](#) ()
- bool [Write](#) (const [DSGSample](#) &elem)
- bool [Read](#) ([DSG::DSGSample](#) &elem)
- size\_t const & [Count](#) () const
- bool [Full](#) () const
- bool [Empty](#) () const
- void [Flush](#) ()

## Protected Member Functions

- `size_t next` (`size_t current`)
- `size_t make_pow_2` (`size_t number`)

## Protected Attributes

- `std::atomic< size_t > _write`
- `std::atomic< size_t > _read`
- `size_t _count`
- `size_t MASK`
- `size_t write`
- `size_t read`

## Friends

- `bool operator>>` (`DSG::DSGSample const &signal`, `DSG::RingBuffer &buffer`)
- `bool operator<<` (`DSG::DSGSample &signal`, `DSG::RingBuffer &buffer`)

### 7.33.1 Detailed Description

`DSG::RingBuffer` - Circular `Buffer` of Audio.

Definition at line 35 of file `RingBuffer.h`.

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 `DSG::RingBuffer::RingBuffer ( )`

Definition at line 25 of file `RingBuffer.cpp`.

```
00025 :Buffer(0),_read(0),_write(0),_count(0),MASK(0){}
```

#### 7.33.2.2 `DSG::RingBuffer::RingBuffer ( const size_t size )`

Definition at line 26 of file `RingBuffer.cpp`.

```
00026                                     :Buffer(make_pow_2(size)),
    _read(0),_write(0),_count(0){
00027     MASK = this->_size-1;
00028 }
```

#### 7.33.2.3 `DSG::RingBuffer::RingBuffer ( RingBuffer & buffer )`

Definition at line 29 of file `RingBuffer.cpp`.

```
00029                                     :Buffer(buffer){
00030     _write.store(buffer._write.load(std::memory_order_acquire));
00031     _read.store(buffer._read.load(std::memory_order_acquire));
00032     _count = buffer._count;
00033     MASK = buffer._size-1;
00034 }
```

#### 7.33.2.4 DSG::RingBuffer::~~RingBuffer ( ) [virtual]

Definition at line 43 of file [RingBuffer.cpp](#).

```
00043 {Flush();}
```

### 7.33.3 Member Function Documentation

#### 7.33.3.1 size\_t const & DSG::RingBuffer::Count ( ) const [inline]

Definition at line 106 of file [RingBuffer.h](#).

```
00106                                     {
00107     return _count;
00108 }
```

#### 7.33.3.2 bool DSG::RingBuffer::Empty ( ) const [inline]

Definition at line 80 of file [RingBuffer.h](#).

```
00080                                     {
00081     return _count==0;
00082 }
```

#### 7.33.3.3 void DSG::RingBuffer::Flush ( ) [inline]

Definition at line 83 of file [RingBuffer.h](#).

```
00083                                     {
00084     _write.store(0,std::memory_order_relaxed);
00085     _read.store(0,std::memory_order_relaxed);
00086     _count=0;
00087 }
```

#### 7.33.3.4 bool DSG::RingBuffer::Full ( ) const [inline]

Definition at line 77 of file [RingBuffer.h](#).

```
00077                                     {
00078     return _count==this->_size;
00079 }
```

#### 7.33.3.5 size\_t DSG::RingBuffer::make\_pow\_2 ( size\_t *number* ) [inline], [protected]

Definition at line 111 of file [RingBuffer.h](#).

```
00111                                     {
00112     return pow(2, ceil(log(number)/log(2)));
00113 }
```

#### 7.33.3.6 size\_t DSG::RingBuffer::next ( size\_t *current* ) [inline], [protected]

Definition at line 110 of file [RingBuffer.h](#).

```
00110 {return (current+1) & MASK;}
```



### 7.33.3.7 DSG::RingBuffer & DSG::RingBuffer::operator= ( RingBuffer & buffer )

Definition at line 35 of file [RingBuffer.cpp](#).

```
00035                                     {
00036     Buffer::operator=(buffer);
00037     _write.store(buffer._write.load(std::memory_order_acquire));
00038     _read.store(buffer._read.load(std::memory_order_acquire));
00039     _count = buffer._count;
00040     MASK = buffer._size-1;
00041     return *this;
00042 }
```

### 7.33.3.8 bool DSG::RingBuffer::Read ( DSG::DSGSample & elem ) [inline]

Definition at line 97 of file [RingBuffer.h](#).

```
00097                                     {
00098     if (!Empty()) {
00099         read = _read.load(std::memory_order_acquire);
00100         _read.store(next(read), std::memory_order_release);
00101         elem = this->_buffer[read];
00102         --_count;
00103         return true;
00104     }else return false;
00105 }
```

### 7.33.3.9 bool DSG::RingBuffer::Write ( const DSGSample & elem ) [inline]

Definition at line 88 of file [RingBuffer.h](#).

```
00088                                     {
00089     if (!Full()) {
00090         write = _write.load(std::memory_order_acquire);
00091         _write.store(next(write), std::memory_order_release);
00092         this->_buffer[write] = elem;
00093         ++_count;
00094         return true;
00095     }else return false;
00096 }
```

## 7.33.4 Friends And Related Function Documentation

### 7.33.4.1 bool operator<< ( DSG::DSGSample & signal, DSG::RingBuffer & buffer ) [friend]

Definition at line 60 of file [RingBuffer.h](#).

```
00060                                     {
00061     return buffer.Read(signal);
00062 }
```

### 7.33.4.2 bool operator>> ( DSG::DSGSample const & signal, DSG::RingBuffer & buffer ) [friend]

Definition at line 57 of file [RingBuffer.h](#).

```
00057                                     {
00058     return buffer.Write(signal);
00059 }
```

## 7.33.5 Member Data Documentation

### 7.33.5.1 size\_t DSG::RingBuffer::\_count [protected]

Definition at line 39 of file [RingBuffer.h](#).

#### 7.33.5.2 `std::atomic<size_t> DSG::RingBuffer::_read` [protected]

Definition at line 38 of file [RingBuffer.h](#).

#### 7.33.5.3 `std::atomic<size_t> DSG::RingBuffer::_write` [protected]

Definition at line 37 of file [RingBuffer.h](#).

#### 7.33.5.4 `size_t DSG::RingBuffer::MASK` [protected]

Definition at line 40 of file [RingBuffer.h](#).

#### 7.33.5.5 `size_t DSG::RingBuffer::read` [protected]

Definition at line 42 of file [RingBuffer.h](#).

#### 7.33.5.6 `size_t DSG::RingBuffer::write` [protected]

Definition at line 41 of file [RingBuffer.h](#).

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

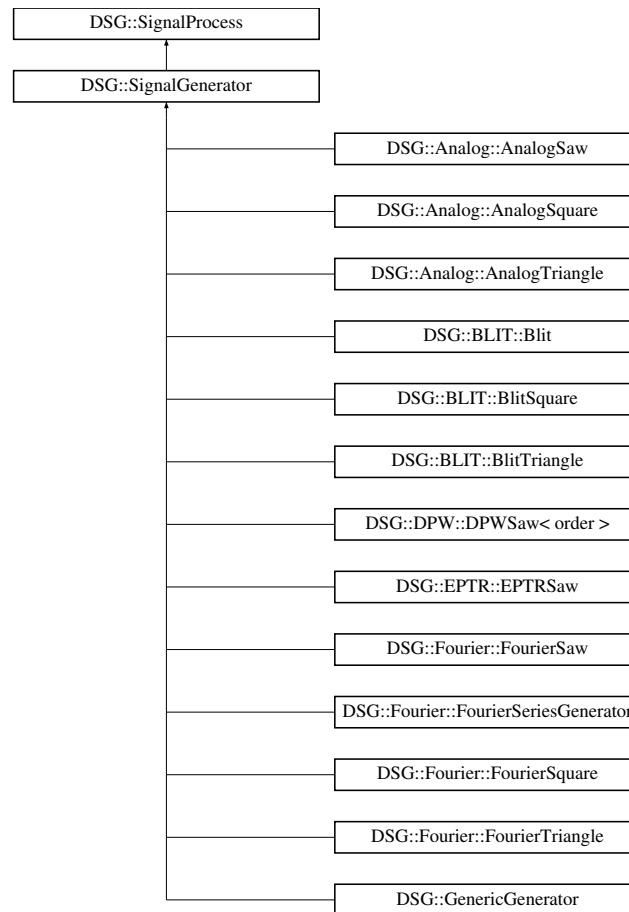
- [RingBuffer.h](#)
- [RingBuffer.cpp](#)

## 7.34 DSG::SignalGenerator Class Reference

[DSG::SignalGenerator](#) - Extends DSG::Signal Process With Tools For Signal Generation.

```
#include <SignalGenerator.h>
```

Inheritance diagram for DSG::SignalGenerator:



## Public Member Functions

- [SignalGenerator](#) ()
- [SignalGenerator](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~SignalGenerator](#) ()
- virtual bool [Perform](#) ([DSG::DSGSample](#) &signal)
- virtual bool [Perform](#) ([DSG::RingBuffer](#) &signal)
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ()
- virtual [DSG::DSGFrequency](#) const & [Frequency](#) ([DSG::DSGFrequency](#) const &value)
- virtual [DSG::DSGPhase](#) const & [Phase](#) ()
- virtual [DSG::DSGPhase](#) const & [Phase](#) ([DSG::DSGPhase](#) const &value)

## Protected Member Functions

- void [step](#) ()
- void [sync](#) ()

## Protected Attributes

- [DSG::DSGFrequency](#) [\\_frequency](#)
- [DSG::DSGPhase](#) [\\_dt](#)
- [DSG::DSGPhase](#) [\\_offset](#)
- [DSG::DSGPhase](#) [\\_phasor](#)
- [DSG::DSGSample](#) [\\_storage](#)

### 7.34.1 Detailed Description

[DSG::SignalGenerator](#) - Extends DSG::Signal Process With Tools For Signal Generation.

Definition at line 33 of file [SignalGenerator.h](#).

### 7.34.2 Constructor & Destructor Documentation

#### 7.34.2.1 DSG::SignalGenerator::SignalGenerator ( )

Definition at line 25 of file [SignalGenerator.cpp](#).

```
00025 :DSG::SignalProcess(),_phasor(0),_frequency(0),
      _dt(0),_offset(0){}
```

#### 7.34.2.2 DSG::SignalGenerator::SignalGenerator ( DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset* )

Definition at line 26 of file [SignalGenerator.cpp](#).

```
00026 :
00026     _phasor(0),_frequency(frequency),_dt(0),_offset(offset){
00027         Frequency(frequency);
00028         Phase(offset);
00029 }
```

#### 7.34.2.3 DSG::SignalGenerator::~SignalGenerator ( ) [virtual]

Definition at line 30 of file [SignalGenerator.cpp](#).

```
00030 {}
```

### 7.34.3 Member Function Documentation

#### 7.34.3.1 DSG::DSGFrequency const & DSG::SignalGenerator::Frequency ( ) [inline],[virtual]

Definition at line 70 of file [SignalGenerator.h](#).

```
00070 {
00071     return _frequency;
00072 }
```

#### 7.34.3.2 DSG::DSGFrequency const & DSG::SignalGenerator::Frequency ( DSG::DSGFrequency const & *value* ) [inline],[virtual]

Reimplemented in [DSG::BLIT::Blit](#), [DSG::BLIT::BlitSaw](#), [DSG::Fourier::FourierSaw](#), [DSG::Fourier::FourierSquare](#), and [DSG::Fourier::FourierTriangle](#).

Definition at line 73 of file [SignalGenerator.h](#).

```
00073 {
00074     _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
00075     _dt = _frequency/DSG::SampleRate();
00076     return _frequency;
00077 }
```

**7.34.3.3** `bool DSG::SignalGenerator::Perform ( DSG::DSGSample & signal )` `[inline], [virtual]`

Implements [DSG::SignalProcess](#).

Reimplemented in [DSG::Fourier::FourierSeriesGenerator](#), [DSG::BLIT::Blit](#), [DSG::DPW::DPWSaw< order >](#), [DSG::EPTR::EPTRSaw](#), [DSG::Analog::AnalogSaw](#), [DSG::Analog::AnalogSquare](#), [DSG::Analog::AnalogTriangle](#), [DSG::BLIT::BlitSaw](#), [DSG::Fourier::FourierSaw](#), [DSG::Fourier::FourierSquare](#), [DSG::Fourier::FourierTriangle](#), and [DSG::GenericGenerator](#).

Definition at line 62 of file [SignalGenerator.h](#).

```
00062                                     {
00063     signal=0;
00064     return false;
00065 }
```

**7.34.3.4** `bool DSG::SignalGenerator::Perform ( DSG::RingBuffer & signal )` `[inline], [virtual]`

Implements [DSG::SignalProcess](#).

Reimplemented in [DSG::DPW::DPWSaw< order >](#), [DSG::Fourier::FourierSeriesGenerator](#), [DSG::BLIT::Blit](#), [DSG::EPTR::EPTRSaw](#), [DSG::Analog::AnalogSaw](#), [DSG::Analog::AnalogSquare](#), [DSG::Analog::AnalogTriangle](#), [DSG::BLIT::BlitSaw](#), [DSG::Fourier::FourierSaw](#), [DSG::Fourier::FourierSquare](#), [DSG::Fourier::FourierTriangle](#), and [DSG::GenericGenerator](#).

Definition at line 66 of file [SignalGenerator.h](#).

```
00066                                     {
00067     signal.Flush();
00068     return false;
00069 }
```

**7.34.3.5** `DSG::DSGPhase const & DSG::SignalGenerator::Phase ( )` `[inline], [virtual]`

Definition at line 78 of file [SignalGenerator.h](#).

```
00078                                     {
00079     return _offset;
00080 }
```

**7.34.3.6** `DSG::DSGPhase const & DSG::SignalGenerator::Phase ( DSG::DSGPhase const & value )` `[inline], [virtual]`

Definition at line 81 of file [SignalGenerator.h](#).

```
00081                                     {
00082     _offset-=value;
00083     _phasor-=_offset;
00084     _offset=value;
00085     return _offset;
00086 }
```

**7.34.3.7** `void DSG::SignalGenerator::step ( )` `[inline], [protected]`

Definition at line 87 of file [SignalGenerator.h](#).

```
00087                                     {
00088     _phasor+=_dt;
00089     _phasor>1.0 ? --_phasor:0;
00090 }
```

#### 7.34.3.8 void DSG::SignalGenerator::sync ( ) [inline],[protected]

Definition at line 91 of file [SignalGenerator.h](#).

```
00091                                     {
00092     _phasor=_offset;
00093 }
```

### 7.34.4 Member Data Documentation

#### 7.34.4.1 DSG::DSGPhase DSG::SignalGenerator::\_dt [protected]

Definition at line 51 of file [SignalGenerator.h](#).

#### 7.34.4.2 DSG::DSGFrequency DSG::SignalGenerator::\_frequency [protected]

Definition at line 50 of file [SignalGenerator.h](#).

#### 7.34.4.3 DSG::DSGPhase DSG::SignalGenerator::\_offset [protected]

Definition at line 52 of file [SignalGenerator.h](#).

#### 7.34.4.4 DSG::DSGPhase DSG::SignalGenerator::\_phasor [protected]

Definition at line 53 of file [SignalGenerator.h](#).

#### 7.34.4.5 DSG::DSGSample DSG::SignalGenerator::\_storage [protected]

Definition at line 54 of file [SignalGenerator.h](#).

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

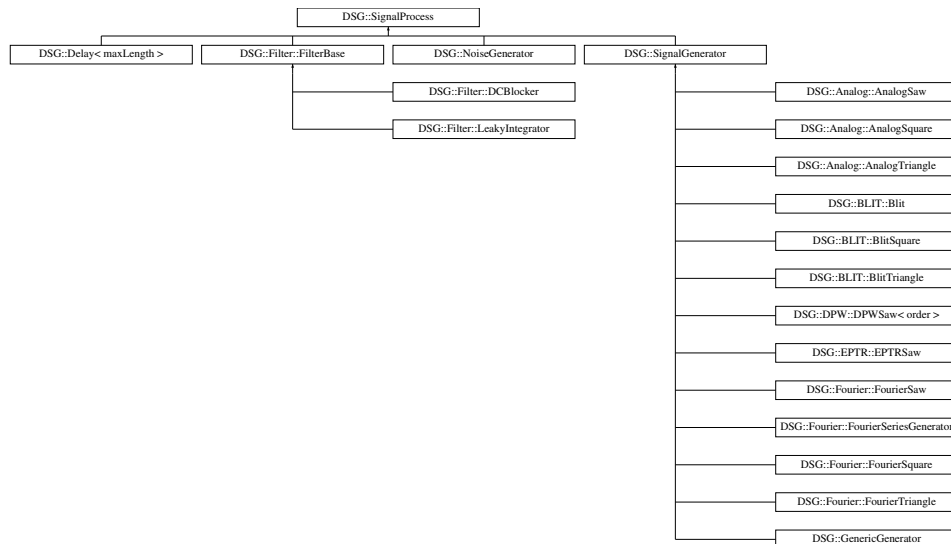
- [SignalGenerator.h](#)
- [SignalGenerator.cpp](#)

## 7.35 DSG::SignalProcess Class Reference

[DSG::SignalProcess](#) - Defines Base Interface For Audio Processing.

```
#include <SignalProcess.h>
```

Inheritance diagram for DSG::SignalProcess:



## Public Member Functions

- [SignalProcess](#) ()
- virtual [~SignalProcess](#) ()
- virtual bool [Perform](#) (DSG::DSGSample &signal)=0
- virtual bool [Perform](#) (DSG::RingBuffer &signal)=0

### 7.35.1 Detailed Description

[DSG::SignalProcess](#) - Defines Base Interface For Audio Processing.

Definition at line 31 of file [SignalProcess.h](#).

### 7.35.2 Constructor & Destructor Documentation

#### 7.35.2.1 DSG::SignalProcess::SignalProcess ( )

Definition at line 25 of file [SignalProcess.cpp](#).

```
00025 {}
```

#### 7.35.2.2 DSG::SignalProcess::~~SignalProcess ( ) [virtual]

Definition at line 26 of file [SignalProcess.cpp](#).

```
00026 {}
```

### 7.35.3 Member Function Documentation

#### 7.35.3.1 virtual bool DSG::SignalProcess::Perform ( DSG::DSGSample & signal ) [inline],[pure virtual]

Implemented in [DSG::Delay< maxLength >](#), [DSG::Fourier::FourierSeriesGenerator](#), [DSG::BLIT::Blit](#), [DSG::DPW::DPWSaw< order >](#), [DSG::EPTR::EPTRSaw](#), [DSG::Analog::AnalogSaw](#), [DSG::Analog::AnalogSquare](#), [DSG::Analog::AnalogTriangle](#), [DSG::BLIT::BlitSaw](#), [DSG::Fourier::FourierSaw](#), [DSG::Fourier::FourierSquare](#), [DSG::Fourier::FourierTriangle](#), [DSG::Filter::LeakyIntegrator](#), [DSG::Filter::FilterBase](#), [DSG::SignalGenerator](#), [DSG::Filter::DCBlocker](#), [DSG::GenericGenerator](#), and [DSG::NoiseGenerator](#).

7.35.3.2 `virtual bool DSG::SignalProcess::Perform ( DSG::RingBuffer & signal )` `[inline],[pure virtual]`

Implemented in [DSG::Delay< maxLength >](#), [DSG::DPW::DPWSaw< order >](#), [DSG::Fourier::FourierSeriesGenerator](#), [DSG::BLIT::Blit](#), [DSG::EPTR::EPTRSaw](#), [DSG::Analog::AnalogSaw](#), [DSG::Analog::AnalogSquare](#), [DSG::Analog::AnalogTriangle](#), [DSG::BLIT::BlitSaw](#), [DSG::Fourier::FourierSaw](#), [DSG::Fourier::FourierSquare](#), [DSG::Fourier::FourierTriangle](#), [DSG::Filter::LeakyIntegrator](#), [DSG::Filter::FilterBase](#), [DSG::SignalGenerator](#), [DSG::Filter::DCBlocker](#), [DSG::GenericGenerator](#), and [DSG::NoiseGenerator](#).

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

- [SignalProcess.h](#)
- [SignalProcess.cpp](#)



## Chapter 8

# File Documentation

### 8.1 AnalogSaw.cpp File Reference

```
#include "AnalogSaw.h"
```

### 8.2 AnalogSaw.cpp

```
00001 //
00002 // AnalogSaw.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "AnalogSaw.h"
00025 DSG::Analog::AnalogSaw():DSG::
SignalGenerator(){}
00026 DSG::Analog::AnalogSaw(
DSG::DSGfrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
00027 DSG::Analog::AnalogSaw::~AnalogSaw(){}

```

### 8.3 AnalogSaw.h File Reference

```
#include "SignalGenerator.h"
```

#### Classes

- class `DSG::Analog::AnalogSaw`  
*DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.*

## Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Analog](#)  
*DSG::Analog - Namespace Containing [Analog](#) Style Oscillators.*

## 8.4 AnalogSaw.h

```

00001 //
00002 //  AnalogSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__AnalogSaw__
00025 #define __DSG__AnalogSaw__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*! DSG::Analog - Namespace Containing Analog Style Oscillators
00032     namespace Analog{
00033         /*!\brief DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator
00034         class AnalogSaw : public DSG::SignalGenerator {
00035         public:
00036             AnalogSaw();
00037             AnalogSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00038             virtual ~AnalogSaw();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041         protected:
00042             DSG::DSGSample _stor;
00043         };
00044         inline bool DSG::Analog::AnalogSaw::Perform(
DSG::DSGSample& signal){
00045             _stor=_phasor;
00046             _stor+=0.5;
00047             if (_stor>1.0) {
00048                 --_stor;
00049             }
00050             _stor-=0.5;
00051             _stor*=2.0;
00052             signal=_stor;
00053             step();
00054             return true;
00055         }
00056         inline bool DSG::Analog::AnalogSaw::Perform(
DSG::RingBuffer& signal){
00057             signal.Flush();
00058             while (!signal.Full()) {
00059                 if (Perform(_storage)) {
00060                     if(signal.Write(_storage)){
00061                         }else return false;
00062                     }else return false;
00063                 }return true;
00064             }
00065         }
00066     }
00067 #endif /* defined(__DSG__AnalogSaw__) */

```

## 8.5 AnalogSquare.cpp File Reference

```
#include "AnalogSquare.h"
```

## 8.6 AnalogSquare.cpp

```
00001 //
00002 // AnalogSquare.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "AnalogSquare.h"
00025 DSG::Analog::AnalogSquare():
    DSG::SignalGenerator(){}
00026 DSG::Analog::AnalogSquare(
    DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
    DSG::SignalGenerator(frequency,offset){}
00027 DSG::Analog::AnalogSquare::~AnalogSquare(){}

```

## 8.7 AnalogSquare.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Analog::AnalogSquare](#)  
[DSG::Analog::AnalogSquare](#) - *Analog Style Square Wave Generator.*

### Namespaces

- [DSG](#)  
[DSG](#) - *A Collection of tools for Digital Signal Generation.*
- [DSG::Analog](#)  
[DSG::Analog](#) - *Namespace Containing [Analog](#) Style Oscillators.*

## 8.8 AnalogSquare.h

```
00001 //
00002 // AnalogSquare.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.

```

```

00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__AnalogSquare__
00025 #define __DSG__AnalogSquare__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*! DSG::Analog - Namespace Containing Analog Style Oscillators
00032     namespace Analog{
00033         /*!\brief DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator
00034         class AnalogSquare : public DSG::SignalGenerator {
00035         public:
00036             AnalogSquare();
00037             AnalogSquare(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00038             virtual ~AnalogSquare();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041         };
00042         inline bool DSG::Analog::AnalogSquare::Perform(
DSG::DSGSample& signal){
00043             signal=_phasor < 0.5 ? 1.0:-1.0;
00044             step();
00045             return true;
00046         }
00047         inline bool DSG::Analog::AnalogSquare::Perform(
DSG::RingBuffer& signal){
00048             signal.Flush();
00049             while (!signal.Full()) {
00050                 if (Perform(_storage)) {
00051                     if(signal.Write(_storage)){
00052                         }else return false;
00053                     }else return false;
00054                 }return true;
00055             }
00056         }
00057     }
00058 #endif /* defined(__DSG__AnalogSquare__) */

```

## 8.9 AnalogTriangle.cpp File Reference

```
#include "AnalogTriangle.h"
```

## 8.10 AnalogTriangle.cpp

```

00001 //
00002 // AnalogTriangle.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.

```

```

00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "AnalogTriangle.h"
00025 DSG::Analog::AnalogTriangle::AnalogTriangle():
00026     DSG::SignalGenerator({})
00027 DSG::Analog::AnalogTriangle::AnalogTriangle(
00028     DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
00029     DSG::SignalGenerator(frequency, offset){}
00030 DSG::Analog::AnalogTriangle::~AnalogTriangle(){}

```

## 8.11 AnalogTriangle.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Analog::AnalogTriangle](#)  
*DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.*

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Analog](#)  
*DSG::Analog - Namespace Containing Analog Style Oscillators.*

## 8.12 AnalogTriangle.h

```

00001 //
00002 // AnalogTriangle.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__AnalogTriangle__
00025 #define __DSG__AnalogTriangle__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     //! DSG::Analog - Namespace Containing Analog Style Oscillators
00032     namespace Analog{

```

```

00033         //!

```

## 8.13 AudioSettings.cpp File Reference

```
#include "AudioSettings.h"
```

## 8.14 AudioSettings.cpp

```

00001 //
00002 // AudioSettings.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "AudioSettings.h"
00025 DSG::DSGFrequency DSG::AudioSettings::_sampleRate;
00026 DSG::DSGFrequency DSG::AudioSettings::_nyquist;
00027 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(){
00028     return _sampleRate;

```

```

00029 }
00030 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(
    DSG::DSGFrequency const& value){
00031     _sampleRate = value;
00032     _nyquist = _sampleRate*0.5;
00033     return _sampleRate;
00034 }
00035 DSG::DSGFrequency const& DSG::AudioSettings::Nyquist() {
00036     return _nyquist;
00037 }

```

## 8.15 AudioSettings.h File Reference

```
#include "DSGTypes.h"
```

### Classes

- class [DSG::AudioSettings](#)  
*DSG::AudioSettings* - Global Storage For Audio Settings Such As Sample Rate.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.

### Functions

- [DSG::DSGFrequency const & DSG::SampleRate \(\)](#)  
*DSG::SampleRate* - Get Global Sample Rate.
- [DSG::DSGFrequency const & DSG::SampleRate \(DSG::DSGFrequency const &value\)](#)  
*DSG::SampleRate* - Set Global Sample Rate.
- [DSG::DSGFrequency DSG::Nyquist \(\)](#)  
*DSG::Nyquist()* - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.

## 8.16 AudioSettings.h

```

00001 //
00002 // AudioSettings.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__AudioSettings__

```

```

00025 #define __DSG__AudioSettings__
00026 #include "DSGTypes.h"
00027 namespace DSG {
00028     /*!\brief DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate
00029     */
00030     class AudioSettings{
00031     public:
00032         static DSG::DSGFrequency const& SampleRate();
00033         static DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value);
00034         static DSG::DSGFrequency const& Nyquist();
00035     protected:
00036         static DSG::DSGFrequency _sampleRate;
00037         static DSG::DSGFrequency _nyquist;
00038     };
00039     /*!\brief DSG::SampleRate - Get Global Sample Rate
00040     inline DSG::DSGFrequency const& SampleRate(){
00041         return DSG::AudioSettings::SampleRate();
00042     }
00043     /*!\brief DSG::SampleRate - Set Global Sample Rate
00044     inline DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value){
00045         return DSG::AudioSettings::SampleRate(value);
00046     }
00047     /*!\brief DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed.
    This value will be updated whenever the sample rate changes.
00048     inline DSG::DSGFrequency Nyquist(){
00049         return DSG::AudioSettings::Nyquist();
00050     }
00051 }
00052 #endif /* defined(__DSG__AudioSettings__) */

```

## 8.17 Blackman.h File Reference

```

#include "PI.h"
#include "LUT.h"
#include "Sine.h"

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Window](#)  
*DSG::Window - Window functions and utilities.*

### Functions

- `template<typename decimal >`  
decimal [DSG::Window::Blackman](#) (decimal const &x)  
*DSG::Window::Blackman - Blackman Window Function.*

## 8.18 Blackman.h

```

00001 //
00002 //  Blackman.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/24/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or

```



```

00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Blackman_h
00025 #define DSG_Blackman_h
00026 #include "PI.h"
00027 #include "LUT.h"
00028 #include "Sine.h"
00029 namespace DSG {
00030 #ifdef DSG_Short_Names
00031     inline
00032 #endif
00033     namespace Window{
00034         //!\brief DSG::Window::Blackman - Blackman Window Function
00035         template<typename decimal>
00036         inline decimal Blackman(decimal const& x){
00037             // Generate Blackman Window
00038             /*
00039             Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
00040             */
00041             static_assert(std::is_floating_point<decimal>::value==true,"DSG::Blackman Function Requires
Floating Point Type");
00042             //we will implement the blackman window as a function as if it were sin(x)
00043             //cos input domain 0-1 not 0-2pi
00044             //range checking is handles within DSG::Cos
00045             decimal phs=x;
00046             while (phs>1.0) {
00047                 phs-=1.0;
00048             }
00049             return 0.42 - (0.5 * DSG::Cos(phs))+(0.08 * DSG::Cos(2.0*phs));
00050         }
00051     }
00052 }
00053 #endif

```

## 8.19 BLIT.cpp File Reference

```
#include "BLIT.h"
```

## 8.20 BLIT.cpp

```

00001 //
00002 // BLIT.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "BLIT.h"
00025 DSG::BLIT::Blit::Blit():DSG::SignalGenerator(){
00026     Frequency(0);
00027 }
00028 DSG::BLIT::Blit::Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){

```

```

00029     Frequency(frequency);
00030 }
00031 DSG::BLIT::Blit::~Blit() {}

```

## 8.21 BLIT.h File Reference

```

#include "SignalGenerator.h"
#include "Denormal.h"
#include "Sinc.h"
#include "DSGMath.h"

```

### Classes

- class [DSG::BLIT::Blit](#)  
[DSG::BLIT::Blit](#) - Band-Limited Impulse Train Generator.

### Namespaces

- [DSG](#)  
[DSG](#) - A Collection of tools for Digital Signal Generation.
- [DSG::BLIT](#)  
[DSG::BLIT](#) - Namespace Containing [BLIT](#) Based Oscillators.

## 8.22 BLIT.h

```

00001 //
00002 //  BLIT.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__BLIT__
00025 #define __DSG__BLIT__
00026 #include "SignalGenerator.h"
00027 #include "Denormal.h"
00028 #include "Sinc.h"
00029 #include "DSGMath.h"
00030 namespace DSG{
00031 #ifdef DSG_Short_Names
00032     inline
00033 #endif
00034     //!DSG::BLIT - Namespace Containing BLIT Based Oscillators
00035     namespace BLIT{
00036         /*!\brief DSG::BLIT::Blit - Band-Limited Impulse Train Generator
00037         */
00038         /*!\todo Re-write DSG::BLIT::Blit algorithm
00039         class Blit:public DSG::SignalGenerator{
00040         public:
00041             Blit();

```

```

00042         Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00043     virtual ~Blit();
00044     virtual inline bool Perform(DSG::DSGSample& signal);
00045     virtual inline bool Perform(DSG::RingBuffer& signal);
00046     virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00047     protected:
00048         unsigned long p_;
00049         unsigned long m_;
00050         unsigned long _h;
00051         double a_;
00052         DSG::DSGSample denominator;
00053         DSG::DSGSample value;
00054     };
00055     inline bool DSG::BLIT::Blit::Perform(
DSG::DSGSample& signal){
00056         //found better results in this case with built in sine function. not performance wise but
algorithmically
00057         denominator = m_ * sin(PI*_phasor);
00058         if (DSG::IsDenormal(denominator)) {
00059             signal = a_;
00060         }else{
00061             value = sin(PI*_phasor * m_);
00062             value/=denominator;
00063             value*=a_;
00064             signal = value;
00065         }
00066         step();
00067         return true;
00068     }
00069     inline bool DSG::BLIT::Blit::Perform(
DSG::RingBuffer& signal){
00070         signal.Flush();
00071         while (!signal.Full()) {
00072             if (Perform(_storage)) {
00073                 if(signal.Write(_storage)){
00074                     }else return false;
00075                 }else return false;
00076             }return true;
00077         }
00078     inline DSG::DSGFrequency const&
DSG::BLIT::Blit::Frequency(DSG::DSGFrequency const& value){
00079         this->SignalGenerator::Frequency(value);
00080         p_ = DSG::SampleRate()/_frequency;
00081         _h = (unsigned)floor(p_*0.5);
00082         m_ = 2 * (_h)+1;
00083         a_ = m_/(double)p_;
00084         return _frequency;
00085     }
00086 }
00087 }
00088 #endif /* defined(__DSG__BLIT__) */

```

## 8.23 BLITSaw.cpp File Reference

```
#include "BLITSaw.h"
```

## 8.24 BLITSaw.cpp

```

00001 //
00002 // BLITSaw.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of

```

```

00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024  #include "BLITSaw.h"
00025  DSG::BLIT::BlitSaw::BlitSaw():DSG::BLIT::Blit(),Register_(0){
00026      Frequency(0);
00027  }
00028  DSG::BLIT::BlitSaw::BlitSaw(DSG::DSGFrequency const& frequency,
00029      DSG::DSGPhase const& offset):DSG::BLIT::Blit(frequency,offset),Register_(0){
00029      Frequency(frequency);
00030  }
00031  DSG::BLIT::BlitSaw::~~BlitSaw(){}

```

## 8.25 BLITSaw.h File Reference

```
#include "BLIT.h"
```

### Classes

- class [DSG::BLIT::BlitSaw](#)  
*DSG::BLIT::BlitSaw* - Saw Wave Generator Based on *BLIT* Algorithm.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.
- [DSG::BLIT](#)  
*DSG::BLIT* - Namespace Containing *BLIT* Based Oscillators.

## 8.26 BLITSaw.h

```

00001  //
00002  //  BLITSaw.h
00003  //  DSG
00004  //
00005  //  Created by Alexander Zywicki on 9/17/14.
00006  //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007  //
00008  /*
00009   This file is part of the Digital Signal Generation Project or "DSG".
00010
00011   DSG is free software: you can redistribute it and/or modify
00012   it under the terms of the GNU General Public License as published by
00013   the Free Software Foundation, either version 3 of the License, or
00014   (at your option) any later version.
00015
00016   DSG is distributed in the hope that it will be useful,
00017   but WITHOUT ANY WARRANTY; without even the implied warranty of
00018   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019   GNU General Public License for more details.
00020
00021   You should have received a copy of the GNU General Public License
00022   along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023   */
00024  #ifndef __DSG__BLITSaw__
00025  #define __DSG__BLITSaw__
00026  #include "BLIT.h"
00027  namespace DSG{
00028  #ifdef DSG_Short_Names
00029      inline
00030  #endif
00031      namespace BLIT{
00032          //!\brief DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm
00033          //!\todo Re-write DSG::BLIT::BlitSaw algorithm

```

```

00034     class BlitSaw : public Blit{
00035     public:
00036         BlitSaw();
00037         BlitSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00038         virtual ~BlitSaw();
00039         virtual inline bool Perform(DSG::DSGSample& signal);
00040         virtual inline bool Perform(DSG::RingBuffer& signal);
00041         virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00042     protected:
00043         DSG::DSGSample C2_;
00044         DSG::DSGSample Register_;
00045     };
00046     inline bool DSG::BLIT::BlitSaw::Perform(
DSG::DSGSample& signal){
00047         denominator = m_ * sin(PI*_phasor);
00048         if (DSG::IsDenormal(denominator)) {
00049             signal = a_;
00050         }else{
00051             value = sin(PI*_phasor * m_);
00052             value/=denominator;
00053             value*=a_;
00054             signal = value;
00055         }
00056         step();
00057         signal += (Register_ - C2_);
00058         Register_ = signal * 0.995;
00059         C2_+=signal;
00060         C2_*=0.5;
00061         return true;
00062     }
00063     inline bool DSG::BLIT::BlitSaw::Perform(
DSG::RingBuffer& signal){
00064         signal.Flush();
00065         while (!signal.Full()) {
00066             if (Perform(_storage)) {
00067                 if(signal.Write(_storage)){
00068                     }else return false;
00069                 }else return false;
00070             }return true;
00071         }
00072     inline DSG::DSGFrequency const&
DSG::BLIT::BlitSaw::Frequency(DSG::DSGFrequency const& value)
{
00073         this->SignalGenerator::Frequency(value);
00074         p_ = DSG::SampleRate()/_frequency;
00075         _h = (unsigned) floor(p_*0.5);
00076         m_ = 2 * (_h)+1;
00077         a_ = m_/(double)p_;
00078         C2_ = 1.0/(double)p_;
00079         return _frequency;
00080     }
00081 }
00082 }
00083 #endif /* defined(__DSG__BLITSaw__) */

```

## 8.27 BLITSquare.cpp File Reference

```
#include "BLITSquare.h"
```

## 8.28 BLITSquare.cpp

```

00001 //
00002 // BLITSquare.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.

```

```

00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "BLITSquare.h"

```

## 8.29 BLITSquare.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::BLIT::BlitSquare](#)

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::BLIT](#)  
*DSG::BLIT - Namespace Containing [BLIT](#) Based Oscillators.*

## 8.30 BLITSquare.h

```

00001 //
00002 // BLITSquare.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__BLITSquare__
00025 #define __DSG__BLITSquare__
00026 #include "SignalGenerator.h"
00027 namespace DSG {
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     namespace BLIT{
00032         //!\todo Write DSG::BLIT::BlitSquare algorithm
00033         class BlitSquare:public DSG::SignalGenerator{};
00034     }
00035 }
00036 #endif /* defined(__DSG__BLITSquare__) */

```

## 8.31 BLITTriangle.cpp File Reference

```
#include "BLITTriangle.h"
```

## 8.32 BLITTriangle.cpp

```
00001 //
00002 //  BLITTriangle.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "BLITTriangle.h"
```

## 8.33 BLITTriangle.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::BLIT::BlitTriangle](#)

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::BLIT](#)  
*DSG::BLIT - Namespace Containing BLIT Based Oscillators.*

## 8.34 BLITTriangle.h

```
00001 //
00002 //  BLITTriangle.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
```

```

00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__BLITTriangle__
00025 #define __DSG__BLITTriangle__
00026
00027 #include "SignalGenerator.h"
00028 namespace DSG {
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     namespace BLIT{
00033         //!\todo Write DSG::BLIT::BlitTriangle algorithm
00034         class BlitTriangle:public DSG::SignalGenerator{};
00035     }
00036 }
00037 #endif /* defined(__DSG__BLITTriangle__) */

```

## 8.35 Bounds.h File Reference

```
#include <assert.h>
```

### Namespaces

- [DSG](#)  
*[DSG](#) - A Collection of tools for Digital Signal Generation.*

### Functions

- `template<int lower, int upper, typename decimal >`  
`decimal DSG::EnforceBounds (decimal const &value)`  
*[DSG::EnforceBounds](#) - Clip value to set bounds.*
- `template<int lower, int upper, int value>`  
`void DSG::StaticAssertBounds ()`  
*[DSG::StaticAssertBounds](#) - Fails on compile time if value is not within bounds.*
- `template<int lower, int upper, typename T >`  
`void DSG::AssertBounds (T const &value)`  
*[DSG::AssertBounds](#) - Fails on runtime if value is not within bounds.*

## 8.36 Bounds.h

```

00001 //
00002 //  Bounds.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/11/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,

```



```

00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Bounds_h
00025 #define DSG_Bounds_h
00026 #include <assert.h>
00027 namespace DSG{
00028     ///\brief DSG::EnforceBounds - Clip value to set bounds
00029     template<int lower,int upper,typename decimal>
00030     decimal EnforceBounds(decimal const& value){
00031         if (value<lower) {
00032             return lower;
00033         }else if(value> upper){
00034             return upper;
00035         }else return value;
00036     }
00037     ///\brief DSG::StaticAssertBounds - Fails on compile time if value is not within bounds
00038     template<int lower,int upper,int value>
00039     void StaticAssertBounds(){
00040         static_assert(value>=lower && value<=upper,"Failed Static Bounds Assert");
00041     }
00042     ///\brief DSG::AssertBounds - Fails on runtime if value is not within bounds
00043     template<int lower,int upper,typename T>
00044     void AssertBounds(T const& value){
00045         assert(value>=lower && value<=upper);
00046     }
00047 }
00048 #endif

```

## 8.37 Buffer.cpp File Reference

```
#include "Buffer.h"
```

## 8.38 Buffer.cpp

```

00001 //
00002 // Buffer.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "Buffer.h"
00025 DSG::Buffer::Buffer():_size(0),_buffer(nullptr){}
00026 DSG::Buffer::Buffer(size_t size):_size(size),_buffer(new
DSG::DSGSample[size]){}
00027 DSG::Buffer::Buffer(Buffer const& other) {
00028     _buffer = new DSG::DSGSample[_size];
00029     _size = other._size;
00030     *this = other;
00031 }
00032 DSG::Buffer& DSG::Buffer::operator=(Buffer const& other){
00033     if (_size!=other._size) {
00034         if (_buffer!=nullptr) {
00035             delete [] _buffer;
00036         }
00037         _size = other._size;

```

```

00038     _buffer = new DSG::DSGSample[_size];
00039 }
00040 for (int i=0; i<_size; ++i) {
00041     _buffer[i] = other._buffer[i];
00042 }
00043 return *this;
00044 }
00045 DSG::Buffer::~~Buffer(){
00046     if (_buffer!=nullptr) {
00047         delete [] _buffer;
00048     }
00049 }
00050 DSG::DSGSample& DSG::Buffer::operator[](size_t const& index){
00051 #ifdef DEBUG
00052     assert(index<_size);
00053 #endif
00054     return _buffer[index];
00055 }

```

## 8.39 Buffer.h File Reference

```

#include <stddef.h>
#include "DSGTypes.h"

```

### Classes

- class [DSG::Buffer](#)  
*DSG::Buffer* - Base Class For *DSG::RingBuffer*. Not For Direct Use.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.

## 8.40 Buffer.h

```

00001 //
00002 // Buffer.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__Buffer__
00025 #define __DSG__Buffer__
00026 #include <stddef.h>
00027 #include "DSGTypes.h"
00028 #ifdef DEBUG
00029 #include <assert.h>
00030 #endif
00031 namespace DSG{
00032     /*!\brief DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use

```

```

00033     */
00034     class Buffer {
00035     public:
00036         Buffer();
00037         Buffer(size_t size);
00038         Buffer(Buffer const& other);
00039         Buffer& operator=(Buffer const& other);
00040         virtual ~Buffer();
00041         DSG::DSGSample& operator[](size_t const& index);
00042         inline size_t const& Size() const;
00043     protected:
00044         DSG::DSGSample* _buffer;
00045         size_t _size;
00046     };
00047     inline size_t const& DSG::Buffer::Size() const{
00048         return _size;
00049     }
00050 }
00051 #endif /* defined(__DSG__Buffer__) */

```

## 8.41 BufferConversion.h File Reference

```
#include "RingBuffer.h"
```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

### Functions

- [bool DSG::RingToArray](#) (DSG::RingBuffer &ring, [DSG::DSGSample](#) \*array, unsigned long length)  
*DSG::RingToArray - Move Ring Buffer data to an array.*
- [bool DSG::ArrayToRing](#) (DSG::RingBuffer &ring, [DSG::DSGSample](#) \*array, unsigned long length)  
*DSG::ArrayToRing - Move array data to a Ring Buffer.*

## 8.42 BufferConversion.h

```

00001 //
00002 //  BufferConversion.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/14/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__BufferConversion__
00025 #define __DSG__BufferConversion__
00026 #include "RingBuffer.h"
00027 namespace DSG {
00028     //!\brief DSG::RingToArray - Move Ring Buffer data to an array
00029     inline bool RingToArray(DSG::RingBuffer& ring,

```

```

    DSG::DSGSample* array,unsigned long length){
00030     for (int i=0; i<length; ++i) {
00031         if (!ring.Empty()) {
00032             ring.Read(array[i]);
00033         }
00034     }return true;
00035 }
00036 //!\brief DSG::ArrayToRing - Move array data to a Ring Buffer
00037 inline bool ArrayToRing(DSG::RingBuffer& ring,
    DSG::DSGSample* array, unsigned long length){
00038     int i=0;
00039     ring.Flush();
00040     while (!ring.Full()) {
00041         ring.Write(array[i]);
00042         ++i;
00043     }return true;
00044 }
00045 }
00046 #endif /* defined(__DSG__BufferConversion__) */

```

## 8.43 DCBlocker.cpp File Reference

```
#include "DCBlocker.h"
```

## 8.44 DCBlocker.cpp

```

00001 //
00002 //  DCBlocker.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/13/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "DCBlocker.h"
00025 DSG::Filter::DCBlocker():DSG::Filter::
    FilterBase(),_a(0.995),xml(0),yaml(0),x(0),_temp(0){}
00026 DSG::Filter::DCBlocker::~DCBlocker(){}

```

## 8.45 DCBlocker.h File Reference

```
#include "Filter.h"
```

### Classes

- class `DSG::Filter::DCBlocker`  
*DSG::Filter::DCBlocker* - DC blocking filter.

## Namespaces

- **DSG**  
*DSG - A Collection of tools for Digital Signal Generation.*
- **DSG::Filter**  
*DSG::Filter - Filters.*

## 8.46 DCBlocker.h

```

00001 //
00002 //  DCBlocker.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/13/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_DCBlocker__
00025 #define __DSG_DCBlocker__
00026 #include "Filter.h"
00027 namespace DSG {
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     namespace Filter{
00032         //!\brief DSG::Filter::DCBlocker - DC blocking filter
00033         class DCBlocker:public DSG::Filter::FilterBase {
00034         public:
00035             DCBlocker();
00036             virtual ~DCBlocker();
00037             virtual inline bool Perform(DSG::DSGSample& signal);
00038             virtual inline bool Perform(DSG::RingBuffer& signal);
00039         protected:
00040             unsigned long count;
00041             DSG::DSGSample _temp;
00042             DSG::DSGSample xml;
00043             DSG::DSGSample yml;
00044             DSG::DSGSample x;
00045             DSG::DSGSample _a;
00046         };
00047         inline bool DSG::Filter::DCBlocker::Perform(
DSG::DSGSample& signal){
00048             x = signal;
00049             signal= x - xml+ (_a * yml);
00050             xml = x;
00051             yml=signal;
00052             return true;
00053         }
00054         inline bool DSG::Filter::DCBlocker::Perform(
DSG::RingBuffer& signal){
00055             if (!signal.Empty()) {
00056                 count = signal.Count();
00057                 while (count-- > 0) {
00058                     if(signal.Read(_temp)){
00059                         if (Perform(_temp)) {
00060                             signal.Write(_temp);
00061                         }else return false;
00062                     }else return false;
00063                 }return true;
00064             }else return false;
00065         }
00066     }
00067 }
00068 #endif /* defined(__DSG_DCBlocker__) */

```

## 8.47 Delay.h File Reference

```
#include "DSGTypes.h"
#include "SignalProcess.h"
#include "Interpolate.h"
#include "AudioSettings.h"
```

### Classes

- class [DSG::Delay< maxLength >](#)  
[DSG::Delay](#) - General purpose delay line.

### Namespaces

- [DSG](#)  
[DSG](#) - A Collection of tools for Digital Signal Generation.

## 8.48 Delay.h

```
00001 //
00002 // Delay.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/23/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__Delay__
00025 #define __DSG__Delay__
00026 #include "DSGTypes.h"
00027 #include "SignalProcess.h"
00028 #include "Interpolate.h"
00029 #include "AudioSettings.h"
00030 namespace DSG{
00031     //!\brief DSG::Delay - General purpose delay line
00032     template<unsigned long maxLength>
00033     class Delay:public DSG::SignalProcess{
00034     public:
00035         Delay():DSG::SignalProcess(),_max(maxLength),
00036         _swap(0),_temp(0),count(0),_index(0),_delay(0){
00037             for (int i=0; i<_max; ++i) {
00038                 _buffer[i]=0;
00039             }
00040         Delay(double const& samples):DSG::SignalProcess(),
00041         _max(maxLength),_swap(0),_temp(0),count(0),_index(0),
00042         _delay(0){
00043             for (int i=0; i<_max; ++i) {
00044                 _buffer[i]=0;
00045             }
00046             if (samples>maxLength) {
00047                 _delay = maxLength;
00048             }else{
00049                 _delay = samples;
00050             }
00051         }
00052     };
00053 }
```

```

00049     }
00050     virtual ~Delay(){}
00051     virtual inline unsigned long const& Length()const{
00052         return _delay;
00053     }
00054     virtual inline unsigned long const& Length(unsigned long const& samples){
00055         if (samples>maxLength) {
00056             _delay = maxLength;
00057         }else{
00058             _delay = samples;
00059         }
00060         return _delay;
00061     }
00062     virtual inline bool Perform(DSG::DSGSample& signal);
00063     virtual inline bool Perform(DSG::RingBuffer& signal);
00064 protected:
00065     unsigned long count;
00066     unsigned long _delay;
00067     unsigned long _index;
00068     const unsigned long _max;
00069     DSG::DSGSample _buffer[maxLength];
00070     DSG::DSGSample _swap;
00071     DSG::DSGSample _temp;
00072     virtual inline void increment(){
00073         ++_index;
00074         if (_index>_delay) {
00075             _index-= _delay;
00076         }
00077     }
00078 };
00079 template<unsigned long maxLength>
00080 inline bool DSG::Delay<maxLength>::Perform(
DSG::DSGSample& signal){
00081     _swap = _buffer[_index-1];
00082     _buffer[_index-1]=signal;
00083     signal = _swap;
00084     increment();
00085     return true;
00086 }
00087 template<unsigned long maxLength>
00088 inline bool DSG::Delay<maxLength>::Perform(
DSG::RingBuffer& signal){
00089     if (!signal.Empty()) {
00090         count = signal.Count();
00091         while (count-- > 0) {
00092             if(signal.Read(_temp)){
00093                 if (Perform(_temp)) {
00094                     signal.Write(_temp);
00095                 }else return false;
00096             }else return false;
00097         }return true;
00098     }else return false;
00099 }
00100 }
00101 #endif /* defined(__DSG_Delay__) */

```

## 8.49 Denormal.h File Reference

```

#include <limits>
#include "DSGMath.h"

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

### Functions

- `template<typename T>`  
`bool DSG::IsDenormal (T const &value)`  
*DSG::IsDenormal - Returns True if number is Denormal.*

## 8.50 Denormal.h

```

00001 //
00002 //  Denormal.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef DSG_Denormal_h
00025 #define DSG_Denormal_h
00026 #include <limits>
00027 #include "DSGMath.h"
00028 namespace DSG{
00029     //!\brief DSG::IsDenormal - Returns True if number is Denormal
00030     template<typename T>
00031     inline bool IsDenormal(T const& value){
00032         return DSG::Abs(value)<=std::numeric_limits<T>::epsilon();//return true if number is
            denormal
00033     }
00034 }
00035 #endif

```

## 8.51 DPW.h File Reference

```

#include "DSGTypes.h"
#include "DSGMath.h"
#include "SignalGenerator.h"
#include "Bounds.h"

```

### Classes

- class [DSG::DPW::DPW\\_Differentiator< order >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 1 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 1st order [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 2 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 2nd order [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 3 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 3rd order [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 4 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 4th order [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 5 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 5th order [DPW](#) Algorithm.
- class [DSG::DPW::DPW\\_Differentiator< 6 >](#)  
[DSG::DPW::DPW\\_Differentiator](#) - Class Performing Differentiation for the 6th order [DPW](#) Algorithm.



## Namespaces

- **DSG**  
*DSG - A Collection of tools for Digital Signal Generation.*
- **DSG::DPW**  
*DSG::DPW - Generators using the DPW method.*

## Functions

- `template<unsigned order>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 1 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 1st Order Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 2 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 2nd order Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 3 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 3rd order Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 4 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 4th order Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 5 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 5th order Polynomial used in DPW Algorithm.*
- `template<>`  
`DSG::DSGSample DSG::DPW::DPW_Polynomial< 6 > (DSG::DSGSample const &value)`  
*DSG::DPW::DPW\_Polynomial - 6th order Polynomial used in DPW Algorithm.*

## 8.52 DPW.h

```

00001 //
00002 //  DPW.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/11/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef DSG_DPW_h
00025 #define DSG_DPW_h
00026 #include "DSGTypes.h"
00027 #include "DSGMath.h"
00028 #include "SignalGenerator.h"
00029 #include "Bounds.h"
00030 namespace DSG{
00031 #ifdef DSG_Short_Names

```

```

00032     inline
00033 #endif
00034     ///\brief DSG::DPW - Generators using the DPW method
00035     namespace DPW{
00036         ///\brief DSG::DPW::DPW_Polynomial - Polynomial used in DPW Algorithm
00037         template<unsigned order>
00038         inline DSG::DSGSample DPW_Polynomial(
00039             DSG::DSGSample const& value){
00039             DSG::StaticAssertBounds<1,6,order>(); //must be 1-6 order
00040             return value;
00041         }
00042         ///\brief DSG::DPW::DPW_Polynomial - 1st Order Polynomial used in DPW Algorithm
00043         template<>
00044         inline DSG::DSGSample DPW_Polynomial<1>(
00045             DSG::DSGSample const& value){
00045             return value;
00046         }
00047         ///\brief DSG::DPW::DPW_Polynomial - 2nd order Polynomial used in DPW Algorithm
00048         template<>
00049         inline DSG::DSGSample DPW_Polynomial<2>(
00050             DSG::DSGSample const& value){
00050             return DSG::Pow<2>(value);
00051         }
00052         ///\brief DSG::DPW::DPW_Polynomial - 3rd order Polynomial used in DPW Algorithm
00053         template<>
00054         inline DSG::DSGSample DPW_Polynomial<3>(
00055             DSG::DSGSample const& value){
00055             return DSG::Pow<3>(value)-value;
00056         }
00057         ///\brief DSG::DPW::DPW_Polynomial - 4th order Polynomial used in DPW Algorithm
00058         template<>
00059         inline DSG::DSGSample DPW_Polynomial<4>(
00060             DSG::DSGSample const& value){
00060             return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
00061         }
00062         ///\brief DSG::DPW::DPW_Polynomial - 5th order Polynomial used in DPW Algorithm
00063         template<>
00064         inline DSG::DSGSample DPW_Polynomial<5>(
00065             DSG::DSGSample const& value){
00065             return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
00066         }
00067         ///\brief DSG::DPW::DPW_Polynomial - 6th order Polynomial used in DPW Algorithm
00068         template<>
00069         inline DSG::DSGSample DPW_Polynomial<6>(
00070             DSG::DSGSample const& value){
00070             return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
00071             DPW_Polynomial<2>(value);
00072         }
00072 #ifdef __APPLE__
00073 #warning DSG::DPW - differentiators order 3-6 need verification. they cause major clipping
00074 #endif
00075         ///\todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 3-6
00076         //differentiators
00077         ///\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm
00078         template<unsigned order>
00079         class DPW_Differentiator{
00080         public:
00081             DPW_Differentiator(){
00082                 DSG::StaticAssertBounds<1, 6,order>(); //order must be 1-6
00083             }
00084         };
00085         ///\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW
00086         Algorithm
00087         template<>
00088         class DPW_Differentiator<1>{
00089         public:
00090             inline DSG::DSGSample operator()(
00091                 DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00092                 return signal;
00093             }
00094         };
00095         ///\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW
00096         Algorithm
00097         template<>
00098         class DPW_Differentiator<2>{
00099         public:
00100             inline DSG::DSGSample operator()(
00101                 DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00102                 output = (signal - _delay)/(4.0 * dt);
00103                 _delay = signal;
00104                 return output;
00105             }
00106         protected:
00107             DSG::DSGSample output;
00108             DSG::DSGSample _delay;
00109         };
00110         ///\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW

```

```

Algorithm
00107     template<>
00108     class DPW_Differentiator<3>{
00109     public:
00110         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00111             output = (signal - _delay[0]);
00112             output -= (_delay[0] - _delay[1]);
00113             output /= (24.*DSG::Pow<2>(dt));
00114             _delay[1]=_delay[0];
00115             _delay[0]=signal;
00116             return output;
00117         }
00118     protected:
00119         DSG::DSGSample output;
00120         DSG::DSGSample _delay[2];
00121     };
00122     //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW
Algorithm
00123     template<>
00124     class DPW_Differentiator<4>{
00125     public:
00126         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00127             output = (signal - _delay[0]);
00128             output -= (_delay[0] - _delay[1]);
00129             output -= (_delay[1] - _delay[2]);
00130             output /= 144*DSG::Pow<3>(dt);
00131             _delay[2]=_delay[1];
00132             _delay[1]=_delay[0];
00133             _delay[0]=signal;
00134             return output;
00135         }
00136     protected:
00137         DSG::DSGSample output;
00138         DSG::DSGSample _delay[3];
00139     };
00140     //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW
Algorithm
00141     template<>
00142     class DPW_Differentiator<5>{
00143     public:
00144         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00145             output = (signal - _delay[0]);
00146             output -= (_delay[0] - _delay[1]);
00147             output -= (_delay[1] - _delay[2]);
00148             output -= (_delay[2] - _delay[3]);
00149             output /= 960*DSG::Pow<4>(dt);
00150             _delay[3]=_delay[2];
00151             _delay[2]=_delay[1];
00152             _delay[1]=_delay[0];
00153             _delay[0]=signal;
00154             return output;
00155         }
00156     protected:
00157         DSG::DSGSample output;
00158         DSG::DSGSample _delay[4];
00159     };
00160     //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW
Algorithm
00161     template<>
00162     class DPW_Differentiator<6>{
00163     public:
00164         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00165             output = (signal - _delay[0]);
00166             output -= (_delay[0] - _delay[1]);
00167             output -= (_delay[1] - _delay[2]);
00168             output -= (_delay[2] - _delay[3]);
00169             output -= (_delay[3] - _delay[4]);
00170             output /= 7200*DSG::Pow<5>(dt);
00171             _delay[4]=_delay[3];
00172             _delay[3]=_delay[2];
00173             _delay[2]=_delay[1];
00174             _delay[1]=_delay[0];
00175             _delay[0]=signal;
00176             return output;
00177         }
00178     protected:
00179         DSG::DSGSample output;
00180         DSG::DSGSample _delay[5];
00181     };
00182 }
00183 }
00184 #endif

```

## 8.53 DPWSaw.h File Reference

```
#include "DPW.h"
```

### Classes

- class `DSG::DPW::DPWSaw< order >`  
*DSG::DPW::DPWSaw* - Sawtooth Generator using the Nth Order *DPW* algorithm.

### Namespaces

- `DSG`  
*DSG* - A Collection of tools for Digital Signal Generation.
- `DSG::DPW`  
*DSG::DPW* - Generators using the *DPW* method.

## 8.54 DPWSaw.h

```
00001 //
00002 //  DPWSaw.h
00003 //  DSG
00004 //
00005 //   Created by Alexander Zywicki on 9/27/14.
00006 //   Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__DPWSaw__
00025 #define __DSG__DPWSaw__
00026 #include "DPW.h"
00027 namespace DSG {
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     namespace DPW{
00032         //!\brief DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm
00033         template<unsigned order>
00034         class DPWSaw:public DSG::SignalGenerator{
00035         public:
00036             DPWSaw():DSG::SignalGenerator(),_register(0){
00037                 DSG::StaticAssertBounds<1, 6,order>();
00038             }
00039             DPWSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset),
_register(0){DSG::StaticAssertBounds<1, 6,order>();}
00040             virtual ~DPWSaw(){}
00041             virtual inline bool Perform(DSG::DSGSample& signal){
00042                 //trivial saw ramping from -1 to 1
00043                 _register = _phasor;
00044                 _register-=0.5;
00045                 _register*=2.0;
00046                 /*-----*/
00047                 //DPW algorithm
00048                 //polynomial shaping
00049                 _register=DSG::DPW::DPW_Polynomial<order>(_register);
00050                 //differentiating
```

```

00051         signal = _diff(_register, _dt);
00052         /*-----*/
00053         //signal = DSG::EnforceBounds<-1, 1>(signal);
00054         //advance phase
00055         step();
00056         return true;
00057     }
00058     virtual inline bool Perform(DSG::RingBuffer& signal){
00059         signal.Flush();
00060         while (!signal.Full()) {
00061             if (Perform(_storage)) {
00062                 if(signal.Write(_storage)){
00063                     }else return false;
00064                 }else return false;
00065             }return true;
00066         }
00067     protected:
00068         DSG::DSGSample _register;
00069         DSG::DPW::DPW_Differentiator<order>
        _diff;
00070     };
00071 }
00072 }
00073 #endif /* defined(__DSG__DPWSaw__) */

```

## 8.55 Driver.cpp File Reference

```
#include "Driver.h"
```

### Macros

- #define [BufferSize](#) 512

### Functions

- int [DriverInit](#) (void \*data)
- int [DriverExit](#) ()
- int [Callback](#) (const void \*input, void \*output, unsigned long frameCount, const PaStreamCallbackTimeInfo \*timeInfo, PaStreamCallbackFlags statusFlags, void \*userData)

### Variables

- PaStream \* [stream](#)
- [DSG::RingBuffer \\_buffer](#) ([BufferSize](#))

### 8.55.1 Macro Definition Documentation

#### 8.55.1.1 #define BufferSize 512

Definition at line 10 of file [Driver.cpp](#).

### 8.55.2 Function Documentation

#### 8.55.2.1 int Callback ( const void \* input, void \* output, unsigned long frameCount, const PaStreamCallbackTimeInfo \* timeInfo, PaStreamCallbackFlags statusFlags, void \* userData )

Definition at line 61 of file [Driver.cpp](#).

```

00066         {
00067             DSG::DSGSample* _out = (DSG::DSGSample*)output;
00068             DSG:: DSGSample _sample;
00069             DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
00070             if (_out!=nullptr) {
00071                 _buffer.Flush();
00072                 _osc->Perform(_buffer);
00073                 for (int i=0; i<frameCount; ++i) {
00074                     _buffer.Read(_sample);
00075                     *_out++ = _sample;
00076                     *_out++ = _sample;
00077                 }
00078             }
00079             return 0;
00080 }

```

### 8.55.2.2 int DriverExit ( )

Definition at line 38 of file [Driver.cpp](#).

```

00038         {
00039             PaError err=0;
00040             err = Pa_StopStream(stream);
00041             if (err!=paNoError) {
00042                 #ifdef DEBUG
00043                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00044                 #endif
00045                 return 1;
00046             }
00047             err = Pa_CloseStream( stream );
00048             if( err != paNoError ){
00049                 #ifdef DEBUG
00050                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00051                 #endif
00052             }
00053             err = Pa_Terminate();
00054             if( err != paNoError ){
00055                 #ifdef DEBUG
00056                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00057                 #endif
00058             }
00059             return 0;
00060 }

```

### 8.55.2.3 int DriverInit ( void \* data )

Definition at line 12 of file [Driver.cpp](#).

```

00012         {
00013             PaError err=0;
00014             err=Pa_Initialize();
00015             if (err!=paNoError) {
00016                 #ifdef DEBUG
00017                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00018                 #endif
00019                 return 1;
00020             }
00021             err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
00022 BufferSize, Callback, data);
00023             if (err!=paNoError) {
00024                 #ifdef DEBUG
00025                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00026                 #endif
00027                 return 1;
00028             }
00029             err = Pa_StartStream(stream);
00030             if (err!=paNoError) {
00031                 #ifdef DEBUG
00032                     printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00033                 #endif
00034                 return 1;
00035             }
00036             return 0;
00037 }

```

### 8.55.3 Variable Documentation

#### 8.55.3.1 DSG:: RingBuffer \_buffer(BufferSize)

#### 8.55.3.2 PaStream\* stream

Definition at line 9 of file [Driver.cpp](#).

## 8.56 Driver.cpp

```

00001 //
00002 //  Driver.cpp
00003 //  Waveform
00004 //
00005 //  Created by Alexander Zywicki on 8/25/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "Driver.h"
00009 PaStream* stream;
00010 #define BufferSize 512
00011 DSG:: RingBuffer _buffer(BufferSize);
00012 int DriverInit(void * data){
00013     PaError err=0;
00014
00015     err=Pa_Initialize();
00016     if (err!=paNoError) {
00017 #ifdef DEBUG
00018         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00019 #endif
00020         return 1;
00021     }
00022     err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
00023                               BufferSize, Callback, data);
00024     if (err!=paNoError) {
00025 #ifdef DEBUG
00026         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00027 #endif
00028         return 1;
00029     }
00030     err = Pa_StartStream(stream);
00031     if (err!=paNoError) {
00032 #ifdef DEBUG
00033         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00034 #endif
00035         return 1;
00036     }
00037     return 0;
00038 }
00039 int DriverExit(){
00040     PaError err=0;
00041     err = Pa_StopStream(stream);
00042     if (err!=paNoError) {
00043 #ifdef DEBUG
00044         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00045 #endif
00046         return 1;
00047     }
00048     err = Pa_CloseStream( stream );
00049     if( err != paNoError ){
00050 #ifdef DEBUG
00051         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00052 #endif
00053     }
00054     err = Pa_Terminate();
00055     if( err != paNoError ){
00056 #ifdef DEBUG
00057         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00058 #endif
00059     }
00060     return 0;
00061 }
00062 int Callback(const void *input,
00063             void *output,
00064             unsigned long frameCount,
00065             const PaStreamCallbackTimeInfo* timeInfo,
00066             PaStreamCallbackFlags statusFlags,
00067             void *userData) {
00068     DSG::DSGSample* _out = (DSG::DSGSample*)output;
00069     DSG:: DSGSample _sample;
00070     DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;

```

```

00070     if (_out!=nullptr) {
00071         _buffer.Flush();
00072         _osc->Perform(_buffer);
00073         for (int i=0; i<frameCount; ++i) {
00074             _buffer.Read(_sample);
00075             *_out++ = _sample;
00076             *_out++ = _sample;
00077         }
00078     }
00079     return 0;
00080 }

```

## 8.57 Driver.h File Reference

```

#include <portaudio.h>
#include "DSG.h"

```

### Functions

- int [DriverInit](#) (void \*data)
- int [DriverExit](#) ()
- int [Callback](#) (const void \*input, void \*output, unsigned long frameCount, const PaStreamCallbackTimeInfo \*timeInfo, PaStreamCallbackFlags statusFlags, void \*userData)

### 8.57.1 Function Documentation

**8.57.1.1** int [Callback](#) ( const void \* *input*, void \* *output*, unsigned long *frameCount*, const PaStreamCallbackTimeInfo \* *timeInfo*, PaStreamCallbackFlags *statusFlags*, void \* *userData* )

Definition at line 61 of file [Driver.cpp](#).

```

00066     {
00067         DSG::DSGSample* _out = (DSG::DSGSample*)output;
00068         DSG::DSGSample _sample;
00069         DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
00070         if (_out!=nullptr) {
00071             _buffer.Flush();
00072             _osc->Perform(_buffer);
00073             for (int i=0; i<frameCount; ++i) {
00074                 _buffer.Read(_sample);
00075                 *_out++ = _sample;
00076                 *_out++ = _sample;
00077             }
00078         }
00079         return 0;
00080 }

```

**8.57.1.2** int [DriverExit](#) ( )

Definition at line 38 of file [Driver.cpp](#).

```

00038     {
00039         PaError err=0;
00040         err = Pa_StopStream(stream);
00041         if (err!=paNoError) {
00042             #ifdef DEBUG
00043                 printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00044             #endif
00045             return 1;
00046         }
00047         err = Pa_CloseStream( stream );
00048         if( err != paNoError ){
00049             #ifdef DEBUG
00050                 printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00051             #endif

```



```

00052     }
00053     err = Pa_Terminate();
00054     if( err != paNoError ){
00055 #ifdef DEBUG
00056         printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00057 #endif
00058     }
00059     return 0;
00060 }

```

### 8.57.1.3 int DriverInit( void \* data )

Definition at line 12 of file [Driver.cpp](#).

```

00012     {
00013         PaError err=0;
00014
00015         err=Pa_Initialize();
00016         if (err!=paNoError) {
00017 #ifdef DEBUG
00018             printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00019 #endif
00020             return 1;
00021         }
00022         err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
00023             BufferSize, Callback, data);
00024         if (err!=paNoError) {
00025 #ifdef DEBUG
00026             printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00027 #endif
00028             return 1;
00029         }
00030         err = Pa_StartStream(stream);
00031         if (err!=paNoError) {
00032 #ifdef DEBUG
00033             printf( "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00034 #endif
00035             return 1;
00036         }
00037         return 0;
00038     }

```

## 8.58 Driver.h

```

00001 //
00002 // Driver.h
00003 // Waveform
00004 //
00005 // Created by Alexander Zywicki on 8/25/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __Waveform_Driver__
00009 #define __Waveform_Driver__
00010 #ifdef DEBUG
00011 #include <iostream>
00012 #endif
00013 #include <portaudio.h>
00014 #include "DSG.h"
00015 int DriverInit(void * data);
00016 int DriverExit();
00017 int Callback( const void *input,
00018             void *output,
00019             unsigned long frameCount,
00020             const PaStreamCallbackTimeInfo* timeInfo,
00021             PaStreamCallbackFlags statusFlags,
00022             void *userData );
00023 #endif /* defined(__Waveform_Driver__) */

```

## 8.59 DSG.h File Reference

```

#include "DSGMath.h"
#include "DSGTypes.h"

```

## Namespaces

- [DSG](#)

*[DSG](#) - A Collection of tools for Digital Signal Generation.*

## Functions

- `template<typename decimal = DSG::DSGSample>`  
`decimal DSG::DSF (decimal const &beta, decimal const &theta, decimal const &N, decimal const &a)`

## 8.60 DSF.h

```

00001 //
00002 // DSF.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 11/5/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_DS_F__
00025 #define __DSG_DS_F__
00026 #include "DSGMath.h"
00027 #include "DSGTypes.h"
00028 namespace DSG{
00029     template<typename decimal=DSG::DSGSample>
00030     decimal DSF(decimal const& beta,decimal const& theta,decimal const& N,decimal const& a){
00031 #ifdef __APPLE__
00032 #warning Untested DSG::DSF()
00033 #endif
00034         decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
00035         decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
sin(theta + (N*beta)));
00036         return numerator/denominator;
00037     }
00038 }
00039 #endif /* defined(__DSG_DS_F__) */

```

## 8.61 DSG.h File Reference

```
#include "AudioSettings.h"
#include "SignalProcess.h"
#include "Buffer.h"
#include "RingBuffer.h"
#include "SignalGenerator.h"
#include "Sine.h"
#include "Sinc.h"
#include "Denormal.h"
#include "Math.h"
#include "Blackman.h"
#include "LUT.h"
#include "Window.h"
#include "Bounds.h"
#include "GenericGenerator.h"
#include "Delay.h"
#include "Sleep.h"
#include "BufferConversion.h"
#include "FourierSeries.h"
#include "FourierSaw.h"
#include "FourierSquare.h"
#include "FourierTriangle.h"
#include "AnalogSaw.h"
#include "AnalogSquare.h"
#include "AnalogTriangle.h"
#include "BLIT.h"
#include "BLITSaw.h"
#include "DSF.h"
#include "DPW.h"
#include "DPWSaw.h"
#include "EPTRSaw.h"
#include "Noise.h"
#include "DCBlocker.h"
#include "Filter.h"
#include "Leaky.h"
```

### Namespaces

- [DSG](#)  
*[DSG](#) - A Collection of tools for Digital Signal Generation.*

### Macros

- `#define` [DSG\\_Short\\_Names](#)

#### 8.61.1 Macro Definition Documentation

##### 8.61.1.1 `#define` [DSG\\_Short\\_Names](#)

Definition at line [26](#) of file [DSG.h](#).

## 8.62 DSG.h

```

00001 //
00002 //  DSG.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef DSG_DSG_h
00025 #define DSG_DSG_h
00026 #define DSG_Short_Names // enables inlining of nested namespaces to allow shorter explicit typenames
00027 //Example: DSG::Analog::AnalogSaw (Long Name)...DSG::AnalogSaw (Short Name) (only available with this macro
//enabled
00028 //!\brief DSG - A Collection of tools for Digital Signal Generation
00029 namespace DSG {
00030 #include "AudioSettings.h"
00031 #include "SignalProcess.h"
00032 #include "Buffer.h"
00033 #include "RingBuffer.h"
00034 #include "SignalGenerator.h"
00035 #include "Sine.h"
00036 #include "Sinc.h"
00037 #include "Denormal.h"
00038 #include "Math.h"
00039 #include "Blackman.h"
00040 #include "LUT.h"
00041 #include "Window.h"
00042 #include "Bounds.h"
00043
00044 #include "GenericGenerator.h"
00045
00046 #include "Delay.h"
00047
00048
00049 #include "Sleep.h"
00050 #include "BufferConversion.h"
00051
00052 #include "FourierSeries.h"
00053 #include "FourierSaw.h"
00054 #include "FourierSquare.h"
00055 #include "FourierTriangle.h"
00056
00057 #include "AnalogSaw.h"
00058 #include "AnalogSquare.h"
00059 #include "AnalogTriangle.h"
00060
00061 #include "BLIT.h"
00062 #include "BLITSaw.h"
00063
00064 #include "DSF.h"
00065
00066 #include "DPW.h"
00067 #include "DPWSaw.h"
00068
00069 #include "EPTRSaw.h"
00070
00071 #include "Noise.h"
00072
00073 #include "DCBlocker.h"
00074
00075 #include "Filter.h"
00076 #include "Leaky.h"
00077
00078 #endif

```

## 8.63 DSGMath.h File Reference

```
#include <math.h>
#include <type_traits>
```

### Classes

- struct [DSG::Factorial< N >](#)  
*DSG::Factorial* - Compute integer factorial.
- struct [DSG::Factorial< 0 >](#)  
*DSG::Factorial* - Compute integer factorial.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.

### Functions

- [template<typename T >](#)  
[T DSG::Abs](#) (T const &value)  
*DSG::Abs* - Calculate absolute value.
- [template<unsigned exponent, class T >](#)  
[T constexpr DSG::Pow](#) (T const base)  
*DSG::Pow* - Any type to an integer power, i.e.  $N^I$ .

## 8.64 DSGMath.h

```
00001 //
00002 //  Math.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef DSG_Math_h
00025 #define DSG_Math_h
00026 #include <math.h>
00027 #include <type_traits>
00028 namespace DSG {
00029     //!\brief DSG::Abs - Calculate absolute value
00030     template<typename T>
00031     inline T Abs(T const& value){
00032         return value < 0.0 ? -1.0 * value : value;
00033     }
00034     //!\brief DSG::Factorial - Compute integer factorial
00035     template<unsigned long N>
```

```

00036     struct Factorial{
00037         enum {value = N * Factorial<N-1>::value};
00038     };
00039     //!\brief DSG::Factorial - Compute integer factorial
00040     template<>
00041     struct Factorial<0>{
00042         enum{ value = 1 };
00043     };
00044     namespace{
00045         template<class T, unsigned N>
00046         struct power{
00047             static constexpr T value(const T x){
00048                 return power<T, N-1>::value(x) * x;
00049             }
00050         };
00051         template<class T>
00052         struct power<T, 0>{
00053             static constexpr T value(const T x){
00054                 return 1;
00055             }
00056         };
00057     }
00058     //!\brief DSG::Pow - Any type to an integer power, i.e.  $N^I$ 
00059     template<unsigned exponent, class T>
00060     T constexpr Pow(T const base){
00061         return power<T, exponent>::value(base);
00062     }
00063 }
00064 #endif

```

## 8.65 DSGTypes.h File Reference

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

### Typedefs

- typedef float [DSG::DSGFrequency](#)  
*DSG::DSGFrequency - Type for representing a frequency value.*
- typedef float [DSG::DSGPhase](#)  
*DSG::DSGPhase - Type for representing a phase value.*
- typedef float [DSG::DSGSample](#)  
*DSG::DSGSample - Type for representing an audio sample.*

## 8.66 DSGTypes.h

```

00001 //
00002 //  DSGTypes.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.

```

```

00023  */
00024  #ifndef DSG_DSGTypes_h
00025  #define DSG_DSGTypes_h
00026  namespace DSG {
00027      //!\brief DSG::DSGFrequency - Type for representing a frequency value
00028      typedef float DSGFrequency;
00029      //!\brief DSG::DSGPhase - Type for representing a phase value
00030      typedef float DSGPhase;
00031      //!\brief DSG::DSGSample - Type for representing an audio sample
00032      typedef float DSGSample;
00033  }
00034  #endif

```

## 8.67 EPTRsaw.cpp File Reference

```
#include "EPTRsaw.h"
```

## 8.68 EPTRsaw.cpp

```

00001  //
00002  //  EPTRsaw.cpp
00003  //  DSG
00004  //
00005  //  Created by Alexander Zywicki on 9/29/14.
00006  //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007  //
00008  /*
00009   This file is part of the Digital Signal Generation Project or "DSG".
00010
00011   DSG is free software: you can redistribute it and/or modify
00012   it under the terms of the GNU General Public License as published by
00013   the Free Software Foundation, either version 3 of the License, or
00014   (at your option) any later version.
00015
00016   DSG is distributed in the hope that it will be useful,
00017   but WITHOUT ANY WARRANTY; without even the implied warranty of
00018   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019   GNU General Public License for more details.
00020
00021   You should have received a copy of the GNU General Public License
00022   along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024  #include "EPTRsaw.h"
00025  DSG::EPTR::EPTRsaw():DSG::SignalGenerator(){}
00026  DSG::EPTR::EPTRsaw(DSG::DSGFrequency const& frequency,
00027                     DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){}
00027  DSG::EPTR::EPTRsaw::~EPTRsaw(){}

```

## 8.69 EPTRsaw.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::EPTR::EPTRsaw](#)  
*DSG::EPTR::EPTRsaw*-Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.
- [DSG::EPTR](#)

*DSG::EPTR - Generators Based On The Efficient Polynomial Transfer Region Algorithm.*

## 8.70 EPTRsaw.h

```

00001 //
00002 // EPTRsaw.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/29/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__EPTRsaw__
00025 #define __DSG__EPTRsaw__
00026 #include "SignalGenerator.h"
00027 namespace DSG {
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*!DSG::EPTR - Generators Based On The Efficient Polynomial Transfer Region Algorithm
00032     namespace EPTR{
00033         /*!brief DSG::EPTR::EPTRsaw-Sawtooth Wave Generator Using The Efficient Polynomial Transfer
00034         Region Algorithm
00035         /*!\todo Test and Possibly Re-Write DSG::EPTR::EPTRsaw algorithm
00036         class EPTRsaw : public DSG::SignalGenerator{
00037         public:
00038             EPTRsaw();
00039             EPTRsaw(DSG::DSGFrequency const& frequency,
00040 DSG::DSGPhase const& offset);
00041             virtual ~EPTRsaw();
00042             virtual inline bool Perform(DSG::DSGSample& signal);
00043             virtual inline bool Perform(DSG::RingBuffer& signal);
00044         protected:
00045             DSG::DSGSample _register;
00046             inline bool DSG::EPTR::EPTRsaw::Perform(
00047 DSG::DSGSample& signal){
00048 #ifdef __APPLE__
00049 #warning Untested For Aliasing DSG::EPTR::EPTRsaw::Perform()
00050 #endif
00051             //generate trivial saw
00052             _register = _phasor;
00053             _register+=0.5;
00054             if (_register>1.0) {
00055                 --_register;
00056             }
00057             _register-=0.5;
00058             _register*=2.0;
00059             if (_register > 1.0-_dt) {
00060                 //transition region detected
00061                 //apply eptr correction
00062                 signal = _register - (_register/_dt) + (1.0/_dt) -1;
00063             }else{
00064                 signal = _register;
00065             }
00066             step();//avance phase
00067             return true;
00068         }
00069         inline bool DSG::EPTR::EPTRsaw::Perform(
00070 DSG::RingBuffer& signal){
00071             signal.Flush();
00072             while (!signal.Full()) {
00073                 if (Perform(_storage)) {
00074                     if(signal.Write(_storage)){
00075                         }else return false;
00076                     }else return false;
00077                 }return true;
00078             }
00079         }

```



```

00076     }
00077 }
00078 #endif /* defined(__DSG__EPTRSAW__) */

```

## 8.71 Filter.cpp File Reference

```
#include "Filter.h"
```

## 8.72 Filter.cpp

```

00001 //
00002 //  Filter.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "Filter.h"
00025 DSG::Filter::FilterBase() : _temp(0), count(0) {}
00026 DSG::Filter::FilterBase::~FilterBase() {}

```

## 8.73 Filter.h File Reference

```
#include "SignalProcess.h"
```

### Classes

- class [DSG::Filter::FilterBase](#)  
*[DSG::Filter::FilterBase](#) - [Filter](#) Base Class, implements interface for cutoff frequency.*

### Namespaces

- [DSG](#)  
*[DSG](#) - A Collection of tools for Digital Signal Generation.*
- [DSG::Filter](#)  
*[DSG::Filter](#) - [Filters](#).*

## 8.74 Filter.h

```

00001 //
00002 //  Filter.h

```

```

00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/27/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG_Filter__
00025 #define __DSG_Filter__
00026 #include "SignalProcess.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*!\brief DSG::Filter - Filters
00032     namespace Filter{
00033         /*!\brief DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency
00034         class FilterBase:public DSG::SignalProcess{
00035         public:
00036             FilterBase();
00037             virtual ~FilterBase();
00038             virtual inline bool Perform(DSG::DSGSample& signal);
00039             virtual inline bool Perform(DSG::RingBuffer& signal);
00040             virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
00041         protected:
00042             DSG::DSGSample _temp;
00043             unsigned long count;
00044         };
00045         inline bool DSG::Filter::FilterBase::Perform(
DSG::DSGSample& signal){
00046             return true;
00047         }
00048         inline bool DSG::Filter::FilterBase::Perform(
DSG::RingBuffer& signal){
00049             if (!signal.Empty()) {
00050                 count = signal.Count();
00051                 while (count-- > 0) {
00052                     if(signal.Read(_temp)){
00053                         if (Perform(_temp)) {
00054                             signal.Write(_temp);
00055                         }else return false;
00056                     }else return false;
00057                 }return true;
00058             }else return false;
00059         }
00060         inline bool DSG::Filter::FilterBase::Cutoff(
DSG::DSGFrequency const& cutoff){
00061             return false;
00062         }
00063     }
00064 }
00065 #endif /* defined(__DSG_Filter__) */

```

## 8.75 FourierSaw.cpp File Reference

```
#include "FourierSaw.h"
```

## 8.76 FourierSaw.cpp

```

00001 //
00002 // FourierSaw.cpp
00003 // DSG
00004 //

```

```

00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "FourierSaw.h"
00025 DSG::Fourier::FourierSaw():DSG::
SignalGenerator(),_a(1.7/PI),phs(0),value(0),i(0){}
00026 DSG::Fourier::FourierSaw::FourierSaw(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
00029 DSG::Fourier::FourierSaw::~FourierSaw(){}

```

## 8.77 FourierSaw.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Fourier::FourierSaw](#)  
*DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator.*

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Fourier](#)  
*DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.*

## 8.78 FourierSaw.h

```

00001 //
00002 // FourierSaw.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020

```

```

00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__FourierSaw__
00025 #define __DSG__FourierSaw__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00032     namespace Fourier{
00033         /*!\brief DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator
00034         class FourierSaw : public DSG::SignalGenerator {
00035         public:
00036             FourierSaw();
00037             FourierSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00038             virtual ~FourierSaw();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041             virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00042         protected:
00043             unsigned long _h;
00044             const double _a;
00045             double phs;
00046             double value;
00047             int i;
00048         };
00049         inline bool DSG::Fourier::FourierSaw::Perform(
DSG::DSGSample& signal){
00050             /*!_h Sine Calls Per Sample where _h is theoretically nyquist / frequency
00051             value=DSG::Sin(_phasor);
00052             for (i=2; i<_h; ++i) {
00053                 value += (1.0/i) * DSG::Sin(_phasor*i);
00054             }
00055             value*=_a;
00056             signal = value;
00057             step();
00058             return true;
00059         }
00060         inline bool DSG::Fourier::FourierSaw::Perform(
DSG::RingBuffer& signal){
00061             signal.Flush();
00062             while (!signal.Full()) {
00063                 if (Perform(_storage)) {
00064                     if(signal.Write(_storage)){
00065                         }else return false;
00066                     }else return false;
00067                 }return true;
00068             }
00069             inline DSG::DSGFrequency const&
DSG::Fourier::FourierSaw::Frequency(
DSG::DSGFrequency const& value){
00070                 _frequency = value;
00071                 _dt = _frequency/DSG::SampleRate();
00072                 _h = MaxHarms(_frequency);
00073                 return _frequency;
00074             }
00075         }
00076     }
00077 #endif /* defined(__DSG__FourierSaw__) */

```

## 8.79 FourierSeries.cpp File Reference

```
#include "FourierSeries.h"
```

## 8.80 FourierSeries.cpp

```

00001 //
00002 // FourierSeries.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 11/18/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //

```

```

00008  /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024  #include "FourierSeries.h"
00025  DSG::Fourier::Harmonic::Harmonic():_ratio(0),_amplitude(0){}
00026  DSG::Fourier::Harmonic::Harmonic(DSG::DSGSample const& ratio,
00027  DSG::DSGSample const& amplitude):_ratio(ratio),_amplitude(amplitude){}
00027  DSG::Fourier::Harmonic::~Harmonic(){
00028  _ratio=0;
00029  _amplitude=0;
00030  }
00031  DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio()const{
00032  return _ratio;
00033  }
00034  DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio(
00035  DSG::DSGSample const& value){
00036  _ratio = value;
00037  return _ratio;
00038  }
00038  DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude()
00039  const{
00040  return _amplitude;
00041  }
00041  DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude(
00042  DSG::DSGSample const& value){
00043  _amplitude=value;
00044  return _amplitude;
00045  }
00045  DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
00046  ():DSG::SignalGenerator(){}
00046  DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
00047  (DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
00048  DSG::SignalGenerator(frequency,offset){}
00047  DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator
00048  (){}

```

## 8.81 FourierSeries.h File Reference

```

#include "SignalGenerator.h"
#include <vector>

```

### Classes

- class [DSG::Fourier::Harmonic](#)  
*DSG::Fourier::Harmonic* - Represents a single harmonic in a *Fourier* Series.
- class [DSG::Fourier::FourierSeriesGenerator](#)  
*DSG::Fourier::FourierSeriesGenerator* - Generates a wave form using a user specified *Fourier* Series.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.
- [DSG::Fourier](#)  
*DSG::Fourier* - Namespace Containing *Fourier* Series Based Oscillators.

## 8.82 FourierSeries.h

```

00001 //
00002 //  FourierSeries.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/18/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_FourierSeries__
00025 #define __DSG_FourierSeries__
00026 #include "SignalGenerator.h"
00027 #include <vector>
00028 namespace DSG{
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     namespace Fourier{
00033         //!\brief DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.
00034         class Harmonic{
00035         public:
00036             Harmonic();
00037             Harmonic(DSG::DSGSample const& ratio,
00038 DSG::DSGSample const& amplitude);
00039             virtual ~Harmonic();
00040             DSG::DSGSample const& Ratio()const;
00041             DSG::DSGSample const& Ratio(DSG::DSGSample const& value);
00042             DSG::DSGSample const& Amplitude()const;
00043             DSG::DSGSample const& Amplitude(
DSG::DSGSample const& value);
00044             protected:
00045                 DSG::DSGSample _ratio;
00046                 DSG::DSGSample _amplitude;
00047             };
00048         //!\brief DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified
Fourier Series
00049         class FourierSeriesGenerator: public
DSG::SignalGenerator{
00050         public:
00051             typedef std::vector<Harmonic> FourierSeries;
00052             FourierSeriesGenerator();
00053             FourierSeriesGenerator(DSG::DSGFrequency const&
frequency, DSG::DSGPhase const& offset);
00054             virtual ~FourierSeriesGenerator();
00055             virtual inline bool Perform(DSG::DSGSample& signal);
00056             virtual inline bool Perform(DSG::RingBuffer& signal);
00057             inline void Series(FourierSeries const& series);
00058             inline FourierSeries& Series();
00059             protected:
00060                 FourierSeries _series;
00061                 DSG::DSGSample value;
00062             };
00063             inline bool DSG::Fourier::FourierSeriesGenerator::Perform
(DSG::DSGSample& signal){
00064                 value = _phasor;
00065                 signal=0;
00066                 for (auto i = _series.begin(); i!=_series.end(); ++i) {
00067                     signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
00068                 }
00069                 step();
00070                 return true;
00071             }
00072             inline bool DSG::Fourier::FourierSeriesGenerator::Perform
(DSG::RingBuffer& signal){
00073                 signal.Flush();
00074                 while (!signal.Full()) {
00075                     if (Perform(_storage)) {
00076                         if(signal.Write(_storage)){
00077                             }else return false;
00078                     }else return false;
00079                 }
00080             }

```

```

00078         }return true;
00079     }
00080     inline void DSG::Fourier::FourierSeriesGenerator::Series
(DSG::Fourier::FourierSeriesGenerator::FourierSeries
const& series){
00081         _series = series;
00082     }
00083     inline DSG::Fourier::FourierSeriesGenerator::FourierSeries
& DSG::Fourier::FourierSeriesGenerator::Series() {
00084         return _series;
00085     }
00086 }
00087 }
00088 #endif /* defined(__DSG__FourierSeries__) */

```

## 8.83 FourierSquare.cpp File Reference

```
#include "FourierSquare.h"
```

## 8.84 FourierSquare.cpp

```

00001 //
00002 //  FourierSquare.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "FourierSquare.h"
00025 DSG::Fourier::FourierSquare::FourierSquare():
DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),i(0){}
00026 DSG::Fourier::FourierSquare::FourierSquare(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset),_a(3.6/PI),phs(0),value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
00029 DSG::Fourier::FourierSquare::~FourierSquare(){}

```

## 8.85 FourierSquare.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Fourier::FourierSquare](#)  
*DSG::Fourier::FourierSquare* - *Fourier Series Square Wave Generator.*

## Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Fourier](#)  
*DSG::Fourier - Namespace Containing [Fourier](#) Series Based Oscillators.*

## 8.86 FourierSquare.h

```

00001 //
00002 //  FourierSquare.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_FourierSquare__
00025 #define __DSG_FourierSquare__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00032     namespace Fourier{
00033         //!\brief DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator
00034         class FourierSquare : public DSG::SignalGenerator {
00035         public:
00036             FourierSquare();
00037             FourierSquare(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00038             virtual ~FourierSquare();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041             virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00042         protected:
00043             unsigned long _h;
00044             const double _a;
00045             double phs;
00046             double value;
00047             int i;
00048         };
00049         inline bool DSG::Fourier::FourierSquare::Perform(
DSG::DSGSample& signal){
00050             //(_h/2)+1 Sine Calls Per Sample
00051             value=DSG::Sin(_phasor);//i=1
00052             for (i=3; i<_h; i+=2) {i=3..5..7..
00053                 value += (1.0/i) * DSG::Sin(_phasor*i);
00054             }
00055             value*=_a;
00056             signal = value;
00057             step();
00058             return true;
00059         }
00060         inline bool DSG::Fourier::FourierSquare::Perform(
DSG::RingBuffer& signal){
00061             signal.Flush();
00062             while (!signal.Full()) {
00063                 if (Perform(_storage)) {
00064                     if(signal.Write(_storage)){
00065                         }else return false;
00066                     }else return false;
00067                 }return true;

```



```

00068     }
00069     inline DSG::DSGFrequency const&
DSG::Fourier::FourierSquare::Frequency(
DSG::DSGFrequency const& value){
00070         _frequency = value;
00071         _dt = _frequency/DSG::SampleRate();
00072         _h = MaxHarms(_frequency);
00073         return _frequency;
00074     }
00075 }
00076 }
00077 #endif /* defined(__DSG__FourierSquare__) */

```

## 8.87 FourierTriangle.cpp File Reference

```
#include "FourierTriangle.h"
```

## 8.88 FourierTriangle.cpp

```

00001 //
00002 // FourierTriangle.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "FourierTriangle.h"
00025 DSG::Fourier::FourierTriangle():
DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),value(0),i(0){}
00026 DSG::Fourier::FourierTriangle(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),phs(0),value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
00029 DSG::Fourier::FourierTriangle::~FourierTriangle(){}

```

## 8.89 FourierTriangle.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Fourier::FourierTriangle](#)  
*DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator.*

### Namespaces

- [DSG](#)

*DSG - A Collection of tools for Digital Signal Generation.*

- **DSG::Fourier**

*DSG::Fourier - Namespace Containing **Fourier** Series Based Oscillators.*

## 8.90 FourierTriangle.h

```

00001 //
00002 //  FourierTriangle.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_FourierTriangle__
00025 #define __DSG_FourierTriangle__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00032     namespace Fourier{
00033         //!\brief DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator
00034         class FourierTriangle : public DSG::SignalGenerator {
00035         public:
00036             FourierTriangle();
00037             FourierTriangle(DSG::DSGFrequency const& frequency,
00038 DSG::DSGPhase const& offset);
00038             virtual ~FourierTriangle();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041             virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00042         protected:
00043             unsigned long _h;
00044             const double _a;
00045             double phs;
00046             double value;
00047             int i;
00048         };
00049         inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::DSGSample& signal){
00050             //(_h/2)+1 Sine Calls Per Sample
00051             value=DSG::Sin(_phasor);//i=1
00052             double sgn = -1;
00053             for (i=3; i<_h; i+=2) {i=3..5..7..
00054                 value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*i);
00055                 sgn*=-1;
00056             }
00057             value*=_a;
00058             signal = value;
00059             step();
00060             return true;
00061         }
00062         inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::RingBuffer& signal){
00063             signal.Flush();
00064             while (!signal.Full()) {
00065                 if (Perform(_storage)) {
00066                     if (signal.Write(_storage)){
00067                         }else return false;
00068                     }else return false;
00069                 }return true;
00070             }
00071             inline DSG::DSGFrequency const&
DSG::Fourier::FourierTriangle::Frequency(

```

```

    DSG::DSGFrequency const& value){
00072     _frequency = value;
00073     _dt = _frequency/DSG::SampleRate();
00074     _h = MaxHarms(_frequency);
00075     return _frequency;
00076 }
00077 }
00078 }
00079 #endif /* defined(__DSG__FourierTriangle__) */

```

## 8.91 Gaussian.h File Reference

```

#include "Sine.h"
#include "White.h"

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Noise](#)  
*DSG::Noise - Noise Generators.*

### Functions

- `template<typename decimal = DSG::DSGSample>`  
`decimal DSG::Noise::Gaussian (decimal=0.0)`  
*[DSG::Noise::Gaussian](#) - Gaussian Noise Generator Function.*

## 8.92 Gaussian.h

```

00001 //
00002 //  Gaussian.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/6/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef DSG_Gaussian_h
00025 #define DSG_Gaussian_h
00026 #include "Sine.h"
00027 #include "White.h"
00028 namespace DSG{
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     namespace Noise{
00033         //!\brief DSG::Noise::Gaussian - Gaussian Noise Generator Function
00034         template<typename decimal=DSG::DSGSample>
00035         decimal Gaussian(decimal=0.0){
00036             static decimal normalizer=1;//variable used to actively normalize the output

```

```

00037         //to enforce compatability with DSG::LUT a dummy parameter is applied
00038         //this parameter is useless except for compatability reasons
00039         decimal R1 = DSG::Noise::White();
00040         decimal R2 = DSG::Noise::White();
00041         decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
00042         if (DSG::Abs(x)>normalizer) {
00043             //store highest output
00044             normalizer=DSG::Abs(x);
00045         }
00046         x/=normalizer;//normalize
00047         return x;
00048     }
00049 }
00050 }
00051 #endif

```

## 8.93 GenericGenerator.cpp File Reference

```
#include "GenericGenerator.h"
```

## 8.94 GenericGenerator.cpp

```

00001 //
00002 //  GenericGenerator.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "GenericGenerator.h"
00025 DSG::GenericGenerator::GenericGenerator():
00026     DSG::SignalGenerator(){}
00026 DSG::GenericGenerator::GenericGenerator(
00027     DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset,
00028     DSG::DSGSample (*signalFunction)(DSG::DSGSample const&)):
00029     DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
00027 DSG::GenericGenerator::~~GenericGenerator(){}

```

## 8.95 GenericGenerator.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class `DSG::GenericGenerator`

*DSG::GenericGenerator* - Generator designed to use a stateless generator function such as *DSG::Sin()*

## Namespaces

- **DSG**

*DSG - A Collection of tools for Digital Signal Generation.*

## 8.96 GenericGenerator.h

```

00001 //
00002 //  GenericGenerator.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__GenericGenerator__
00025 #define __DSG__GenericGenerator__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028     //!brief DSG::GenericGenerator - Generator designed to use a stateless generator function such as
00029     DSG::Sin()
00030     class GenericGenerator:public DSG::SignalGenerator{
00031     public:
00032         GenericGenerator();
00033         GenericGenerator(DSG::DSGFrequency const& frequency,
00034         DSG::DSGPhase const& offset,DSG::DSGSample (*signalFunction)(
00035         DSG::DSGSample const&));
00036         virtual ~GenericGenerator();
00037         virtual inline bool Perform(DSG::DSGSample& signal);
00038         virtual inline bool Perform(DSG::RingBuffer& signal);
00039     protected:
00040         DSG::DSGSample (*_callback) (DSG::DSGSample const&);
00041     };
00042     inline bool DSG::GenericGenerator::Perform(
00043     DSG::DSGSample& signal){
00044         if (_callback!=nullptr) {
00045             signal = _callback(_phasor);
00046         }else signal = 0;
00047         step();
00048         return true;
00049     }
00050     inline bool DSG::GenericGenerator::Perform(
00051     DSG::RingBuffer& signal){
00052         signal.Flush();
00053         while (!signal.Full()) {
00054             if (Perform(_storage)) {
00055                 if(signal.Write(_storage)){
00056                     }else return false;
00057                 }else return false;
00058             }return true;
00059         }
00060     }
00061 #endif /* defined(__DSG__GenericGenerator__) */

```

## 8.97 Interpolate.h File Reference

```

#include "DSGMath.h"
#include "PI.h"

```

## Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

## Functions

- `template<typename decimal >`  
`decimal DSG::LinearInterpolate (decimal const &y1, decimal const &y2, decimal const &mu)`  
*[DSG::LinearInterpolate](#) - Linear Interpolation.*
- `template<typename decimal >`  
`decimal DSG::CosineInterpolate (decimal y1, decimal y2, decimal mu)`  
*[DSG::CosineInterpolate](#) - Cosine Interpolation.*
- `template<typename decimal >`  
`decimal DSG::CubicInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu)`  
*[DSG::CubicInterpolate](#) - Cubic Interpolation.*
- `template<typename decimal >`  
`decimal DSG::HermiteInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu, decimal const &tension, decimal const &bias)`  
*[DSG::HermiteInterpolate](#) - Hermite Interpolation.*

## 8.98 Interpolate.h

```

00001 //
00002 //  Interpolate.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 //Code In this file was adapted from the code provided on this website
00009 //http://paulbourke.net/miscellaneous/interpolation/
00010 //
00011 /*
00012  This file is part of the Digital Signal Generation Project or "DSG".
00013
00014  DSG is free software: you can redistribute it and/or modify
00015  it under the terms of the GNU General Public License as published by
00016  the Free Software Foundation, either version 3 of the License, or
00017  (at your option) any later version.
00018
00019  DSG is distributed in the hope that it will be useful,
00020  but WITHOUT ANY WARRANTY; without even the implied warranty of
00021  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00022  GNU General Public License for more details.
00023
00024  You should have received a copy of the GNU General Public License
00025  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00026  */
00027 #ifndef DSG_Interpolate_h
00028 #define DSG_Interpolate_h
00029 #include "DSGMath.h"
00030 #include "PI.h"
00031 namespace DSG{
00032     //!\brief DSG::LinearInterpolate - Linear Interpolation
00033     template<typename decimal>
00034     decimal LinearInterpolate(decimal const& y1,decimal const& y2,decimal const& mu){
00035         return(y1*(1-mu)+y2*mu);
00036     }
00037     //!\brief DSG::CosineInterpolate - Cosine Interpolation
00038     template<typename decimal>
00039     decimal CosineInterpolate(
00040         decimal y1,decimal y2,
00041         decimal mu)
00042     {
00043         decimal mu2;
00044         mu2 = (1-cos(mu*PI))/2.0;
00045         return(y1*(1-mu2)+y2*mu2);
00046     }

```

```

00047     //!

```

## 8.99 Leaky.cpp File Reference

```
#include "Leaky.h"
```

## 8.100 Leaky.cpp

```

00001 //
00002 // Leaky.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/27/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "Leaky.h"
00025 DSG::Filter::LeakyIntegrator::LeakyIntegrator():
    DSG::Filter::FilterBase() {

```

```

00026     x1=0;
00027     y1=0;
00028     a=0;
00029     b=0;
00030     y=0;
00031 }
00032 DSG::Filter::LeakyIntegrator::LeakyIntegrator(
    DSG::DSGFrequency const& cutoff):DSG::Filter::FilterBase() {
00033     x1=0;
00034     y1=0;
00035     a=0;
00036     b=0;
00037     y=0;
00038     Cutoff(cutoff);
00039 }
00040 DSG::Filter::LeakyIntegrator::~~LeakyIntegrator() {
00041     x1=0;
00042     y1=0;
00043     a=0;
00044     b=0;
00045     y=0;
00046 }

```

## 8.101 Leaky.h File Reference

```

#include "Filter.h"
#include "PI.h"

```

### Classes

- class [DSG::Filter::LeakyIntegrator](#)  
[DSG::Filter::LeakyIntegrator](#) - *Leaky integrator.*

### Namespaces

- [DSG](#)  
[DSG](#) - *A Collection of tools for Digital Signal Generation.*
- [DSG::Filter](#)  
[DSG::Filter](#) - *Filters.*

## 8.102 Leaky.h

```

00001 //
00002 //  Leaky.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_Leaky__
00025 #define __DSG_Leaky__

```



```

00026 #include "Filter.h"
00027 #include "PI.h"
00028 namespace DSG {
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     namespace Filter{
00033         //!\brief DSG::Filter::LeakyIntegrator - Leaky integrator
00034         class LeakyIntegrator:public DSG::Filter::FilterBase{
00035         public:
00036             LeakyIntegrator();
00037             LeakyIntegrator(DSG::DSGFrequency const& cutoff);
00038             virtual ~LeakyIntegrator();
00039             virtual inline bool Perform(DSG::DSGSample& signal);
00040             virtual inline bool Perform(DSG::RingBuffer& signal);
00041             virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
00042         protected:
00043             double x1,y1,a,b;
00044             double y;
00045         };
00046         inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::DSGSample& signal){
00047             y = b * (signal + x1) - a * y1;
00048             x1=signal;
00049             y1=y;
00050             signal=y;
00051             return true;
00052         }
00053         inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::RingBuffer& signal){
00054             if (!signal.Empty()) {
00055                 count = signal.Count();
00056                 while (count-- > 0) {
00057                     if(signal.Read(_temp)){
00058                         if (Perform(_temp)) {
00059                             signal.Write(_temp);
00060                         }else return false;
00061                     }else return false;
00062                 }return true;
00063             }else return false;
00064         }
00065         inline bool DSG::Filter::LeakyIntegrator::Cutoff(
DSG::DSGFrequency const& cutoff){
00066             double Omega;
00067             x1 = y1 = 0.0;
00068             Omega = atan(PI * cutoff);
00069             a = -(1.0 - Omega) / (1.0 + Omega);
00070             b = (1.0 - b) / 2.0;
00071             return true;
00072         }
00073     }
00074 }
00075 #endif /* defined(__DSG_Leaky__) */

```

## 8.103 LUT.h File Reference

```
#include "Interpolate.h"
```

### Classes

- class [DSG::LUT< element, size >](#)

*DSG::LUT - Look Up Table.*

### Namespaces

- [DSG](#)

*DSG - A Collection of tools for Digital Signal Generation.*

## 8.104 LUT.h

```

00001 //
00002 //  LUT.h
00003 //  Waveform
00004 //
00005 //  Created by Alexander Zywicki on 8/25/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef Waveform_LUT_h
00025 #define Waveform_LUT_h
00026 #ifdef DEBUG
00027 #include <assert.h>
00028 #endif
00029 #include "Interpolate.h"
00030 namespace DSG{
00031     //!\brief DSG::LUT - Look Up Table
00032     template <typename element,unsigned long size>
00033     class LUT {
00034     public:
00035         typedef element (*FillFunction)(element);
00036         typedef element (*FillFunctionConstRef)(element const&);
00037         LUT():_size(size){}
00038         LUT(FillFunction fill,double const& range = 1.0):_size(size){
00039             //range is the expected input range for the function
00040             //example would be 0-2pi or 0-1
00041             //would be provided a 2pi or 1
00042             //defaults to 1
00043             double step = range/(double)_size;
00044             phs = 0;
00045             for (int i=0; i<_size; ++i) {
00046                 _table[i] = fill(phs);
00047                 phs+=step;
00048             }
00049         }
00050         LUT(FillFunctionConstRef fill,double const& range = 1.0):
00051         _size(size){
00052             //range is the expected input range for the function
00053             //example would be 0-2pi or 0-1
00054             //would be provided a 2pi or 1
00055             //defaults to 1
00056             double step = range/_size;
00057             phs = 0;
00058             for (int i=0; i<_size; ++i) {
00059                 _table[i] = fill(phs);
00060                 phs+=step;
00061             }
00062         }
00063         ~LUT(){}
00064         element const& operator[](unsigned long const& index)const{
00065             #ifdef DEBUG
00066                 assert(index<_size);
00067             #endif
00068             return _table[index];
00069         }
00070         element& operator[](unsigned long const& index){
00071             #ifdef DEBUG
00072                 assert(index<_size);
00073             #endif
00074             return _table[index];
00075         }
00076         inline element const& operator()(double const& x){
00077             phs=x;
00078             //need range checking on x to ensure 0-1 range
00079             phs<0 ? phs = 1-(phs*-1):0;
00080             phs-=((int)phs);
00081             return this->_table[(unsigned)(phs*(this->_size-1))];
00082         }
00083         unsigned long const& Size()const{
00084             return _size;

```

```

00084     }
00085     protected:
00086         element _table[size];
00087         const unsigned long _size;
00088         double phs;
00089     };
00090 }
00091 #endif

```

## 8.105 Noise.h File Reference

```

#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"

```

## 8.106 Noise.h

```

00001 //
00002 // Noise.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/20/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Noise_h
00025 #define DSG_Noise_h
00026 #include "Random.h"
00027 #include "Gaussian.h"
00028 #include "White.h"
00029 #include "Pink.h"
00030 #include "NoiseGenerator.h"
00031 #endif

```

## 8.107 NoiseGenerator.cpp File Reference

```

#include "NoiseGenerator.h"

```

## 8.108 NoiseGenerator.cpp

```

00001 //
00002 // NoiseGenerator.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/20/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*

```

```

00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "NoiseGenerator.h"
00025 DSG::NoiseGenerator::NoiseGenerator(DSGSample (*
    StatelessFunction)(DSGSample)):DSG::SignalProcess(){
00026     _function = StatelessFunction;
00027 }
00028 DSG::NoiseGenerator::~NoiseGenerator(){}

```

## 8.109 NoiseGenerator.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::NoiseGenerator](#)  
[DSG::NoiseGenerator](#) - Generator that uses noise functions such as [DSG::White\(\)](#) to generate signal.

### Namespaces

- [DSG](#)  
[DSG](#) - A Collection of tools for Digital Signal Generation.

## 8.110 NoiseGenerator.h

```

00001 //
00002 // NoiseGenerator.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/20/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG_NoiseGenerator__
00025 #define __DSG_NoiseGenerator__
00026 #include "SignalGenerator.h"
00027 namespace DSG{
00028     //!\brief DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate
    signal
00029     class NoiseGenerator:public SignalProcess{
00030     public:

```

```

00031     NoiseGenerator(DSGSample (*StatelessFunction) (
DSGSample));
00032     virtual ~NoiseGenerator();
00033     virtual inline bool Perform(DSG::DSGSample& signal);
00034     virtual inline bool Perform(DSG::RingBuffer& signal);
00035     protected:
00036     DSGSample (*_function) (DSGSample);
00037     DSG::DSGSample _storage;
00038 };
00039     inline bool DSG::NoiseGenerator::Perform(
DSG::DSGSample& signal){
00040     signal = _function(0);
00041     return true;
00042 }
00043     inline bool DSG::NoiseGenerator::Perform(
DSG::RingBuffer& signal){
00044     signal.Flush();
00045     while (!signal.Full()) {
00046         if (Perform(_storage)) {
00047             if(signal.Write(_storage)){
00048                 }else return false;
00049             }else return false;
00050         }return true;
00051     }
00052 }
00053 #endif /* defined(__DSG__NoiseGenerator__) */

```

## 8.111 PI.h File Reference

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

### Macros

- #define [PI](#) 3.14159265358979323846264338327
- #define [TWOPI](#) 6.28318530717958647692528676656

#### 8.111.1 Macro Definition Documentation

##### 8.111.1.1 #define PI 3.14159265358979323846264338327

Definition at line 27 of file [PI.h](#).

##### 8.111.1.2 #define TWOPI 6.28318530717958647692528676656

Definition at line 28 of file [PI.h](#).

## 8.112 PI.h

```

00001 //
00002 // PI.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.

```

```

00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_PI_h
00025 #define DSG_PI_h
00026 namespace DSG{
00027 #define PI 3.14159265358979323846264338327
00028 #define TWOPI 6.28318530717958647692528676656
00029 }
00030 #endif

```

## 8.113 Pink.h File Reference

```

#include "Gaussian.h"
#include "DCBlocker.h"

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Noise](#)  
*DSG::Noise - Noise Generators.*

### Functions

- `template<typename decimal = DSG::DSGSample>`  
`decimal DSG::Noise::Pink (decimal=0.0)`  
*DSG::Noise::Pink - Pink Noise Generator Function.*

## 8.114 Pink.h

```

00001 //
00002 // Pink.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/8/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Pink_h
00025 #define DSG_Pink_h
00026 #include "Gaussian.h"
00027 #include "DCBlocker.h"
00028 namespace DSG{
00029 #ifdef DSG_Short_Names

```

```

00030     inline
00031 #endif
00032     namespace Noise{
00033         //!

```

## 8.115 Random.h File Reference

```

#include "DSGTypes.h"
#include <random>

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Noise](#)  
*DSG::Noise - Noise Generators.*

### Functions

- template<typename decimal = DSG::DSGSample>  
decimal [DSG::Noise::Random](#) (decimal=0.0)  
*DSG::Noise::Random - Random Number Function.*

#### 8.115.1 Variable Documentation

8.115.1.1 `const decimal max = static_cast<decimal>(RAND_MAX)`

Definition at line 45 of file [Random.h](#).

## 8.116 Random.h

```

00001 //
00002 // Random.h
00003 // DSG

```

```

00004 //
00005 // Created by Alexander Zywicki on 10/28/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Random_h
00025 #define DSG_Random_h
00026 #include "DSGTypes.h"
00027 #include <random>
00028 namespace DSG{
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     /*!\brief DSG::Noise - Noise Generators
00033     namespace Noise{
00034         namespace{
00035             template<typename decimal>
00036             class random_helper{
00037             public:
00038                 random_helper() {
00039                     srand(static_cast<unsigned>(time(NULL)));
00040                 }
00041                 inline decimal next() {
00042                     return static_cast<decimal>(rand()/max);
00043                 }
00044             protected:
00045                 const decimal max = static_cast<decimal>(RAND_MAX);
00046             };
00047         }
00048         /*!\brief DSG::Noise::Random - Random Number Function
00049         template<typename decimal = DSG::DSGSample>
00050         inline decimal Random(decimal=0.0){
00051             static DSG::Noise::random_helper<decimal> _rand{};
00052             return _rand.next();
00053         }
00054     }
00055 }
00056 #endif

```

## 8.117 RingBuffer.cpp File Reference

```
#include "RingBuffer.h"
```

## 8.118 RingBuffer.cpp

```

00001 //
00002 // RingBuffer.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of

```



```

00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024  #include "RingBuffer.h"
00025  DSG::RingBuffer::RingBuffer():Buffer(0),_read(0),_write(0),_count(0),
    MASK(0){}
00026  DSG::RingBuffer::RingBuffer(const size_t size):
    Buffer(make_pow_2(size)),_read(0),_write(0),_count(0){
00027      MASK = this->_size-1;
00028  }
00029  DSG::RingBuffer::RingBuffer(RingBuffer& buffer):
    Buffer(buffer){
00030      _write.store(buffer._write.load(std::memory_order_acquire));
00031      _read.store(buffer._read.load(std::memory_order_acquire));
00032      _count = buffer._count;
00033      MASK = buffer._size-1;
00034  }
00035  DSG::RingBuffer& DSG::RingBuffer::operator=(
    RingBuffer& buffer){
00036      Buffer::operator=(buffer);
00037      _write.store(buffer._write.load(std::memory_order_acquire));
00038      _read.store(buffer._read.load(std::memory_order_acquire));
00039      _count = buffer._count;
00040      MASK = buffer._size-1;
00041      return *this;
00042  }
00043  DSG::RingBuffer::~RingBuffer() {Flush();}
00044

```

## 8.119 RingBuffer.h File Reference

```

#include <atomic>
#include "DSGMath.h"
#include "Buffer.h"

```

### Classes

- class [DSG::RingBuffer](#)  
*DSG::RingBuffer* - Circular *Buffer* of Audio.

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.

## 8.120 RingBuffer.h

```

00001  //
00002  //  RingBuffer.h
00003  //  DSG
00004  //
00005  //  Created by Alexander Zywicki on 9/16/14.
00006  //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007  //
00008  /*
00009   This file is part of the Digital Signal Generation Project or "DSG".
00010
00011   DSG is free software: you can redistribute it and/or modify
00012   it under the terms of the GNU General Public License as published by
00013   the Free Software Foundation, either version 3 of the License, or
00014   (at your option) any later version.
00015
00016   DSG is distributed in the hope that it will be useful,
00017   but WITHOUT ANY WARRANTY; without even the implied warranty of
00018   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

```

```

00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__RingBuffer__
00025 #define __DSG__RingBuffer__
00026 #ifdef DEBUG
00027 #include <iostream>
00028 #endif
00029 #include <atomic>
00030 #include "DSGMath.h"
00031 #include "Buffer.h"
00032 namespace DSG {
00033     /*!\brief DSG::RingBuffer - Circular Buffer of Audio
00034     */
00035     class RingBuffer:public DSG::Buffer {
00036     protected:
00037         std::atomic<size_t> _write;
00038         std::atomic<size_t> _read;
00039         size_t _count;
00040         size_t MASK;
00041         size_t write;
00042         size_t read;
00043         inline size_t next(size_t current);
00044         inline size_t make_pow_2(size_t number);
00045     public:
00046         RingBuffer();
00047         RingBuffer(const size_t size);
00048         RingBuffer(RingBuffer& buffer);
00049         RingBuffer& operator=(RingBuffer& buffer);
00050         virtual ~RingBuffer();
00051         inline bool Write(const DSGSample& elem);
00052         inline bool Read(DSG::DSGSample& elem);
00053         inline size_t const& Count()const;
00054         inline bool Full()const;
00055         inline bool Empty()const;
00056         inline void Flush();
00057         friend bool operator>>(DSG::DSGSample const& signal,
DSG::RingBuffer& buffer){
00058             return buffer.Write(signal);
00059         }
00060         friend bool operator<<(DSG::DSGSample& signal,
DSG::RingBuffer& buffer){
00061             return buffer.Read(signal);
00062         }
00063         #ifdef DEBUG
00064             friend std::ostream& operator<<(std::ostream& os,
DSG::RingBuffer const& buffer){
00065                 if (!buffer.Empty()) {
00066                     size_t index= buffer._read;
00067                     size_t count=buffer.Count();
00068                     size_t size = buffer.Size();
00069                     for (int i=0; i<count; ++i) {
00070                         os<<index<<": "<<buffer._buffer[index]<<std::endl;
00071                         index = ((index+1)%size);
00072                     }
00073                 }return os;
00074             }
00075         #endif
00076     };
00077     inline bool DSG::RingBuffer::Full()const{
00078         return _count==this->_size;
00079     }
00080     inline bool DSG::RingBuffer::Empty()const{
00081         return _count==0;
00082     }
00083     inline void DSG::RingBuffer::Flush(){
00084         _write.store(0,std::memory_order_relaxed);
00085         _read.store(0,std::memory_order_relaxed);
00086         _count=0;
00087     }
00088     inline bool DSG::RingBuffer::Write(const DSGSample& elem){
00089         if (!Full()) {
00090             write = _write.load(std::memory_order_acquire);
00091             _write.store(next(write),std::memory_order_release);
00092             this->_buffer[write] = elem;
00093             ++_count;
00094             return true;
00095         }else return false;
00096     }
00097     inline bool DSG::RingBuffer::Read(DSGSample& elem){
00098         if (!Empty()) {
00099             read = _read.load(std::memory_order_acquire);
00100             _read.store(next(read),std::memory_order_release);
00101             elem = this->_buffer[read];
00102             --_count;

```

```

00103         return true;
00104     }else return false;
00105 }
00106 inline size_t const& DSG::RingBuffer::Count()const{
00107     return _count;
00108 }
00109 //note: RingBuffer implementation will force a power of 2 size to allow use of bitwise increment.
00110 inline size_t DSG::RingBuffer::next(size_t current){return (current+1) & MASK;}
00111 inline size_t DSG::RingBuffer::make_pow_2(size_t number){
00112     return pow(2, ceil(log(number)/log(2)));
00113 }
00114 }
00115 #endif /* defined(__DSG__RingBuffer__) */

```

## 8.121 SignalGenerator.cpp File Reference

```
#include "SignalGenerator.h"
```

## 8.122 SignalGenerator.cpp

```

00001 //
00002 //  SignalGenerator.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "SignalGenerator.h"
00025 DSG::SignalGenerator::SignalGenerator():DSG::
SignalProcess(),_phasor(0),_frequency(0),_dt(0),_offset(0){}
00026 DSG::SignalGenerator::SignalGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):_phasor(0),
_frequency(frequency),_dt(0),_offset(offset){
00027     Frequency(frequency);
00028     Phase(offset);
00029 }
00030 DSG::SignalGenerator::~SignalGenerator(){}

```

## 8.123 SignalGenerator.h File Reference

```

#include "SignalProcess.h"
#include "AudioSettings.h"
#include "Sine.h"
#include "Bounds.h"

```

### Classes

- class [DSG::SignalGenerator](#)  
[DSG::SignalGenerator](#) - Extends DSG::Signal Process With Tools For Signal Generation.

## Namespaces

- [DSG](#)

*[DSG](#) - A Collection of tools for Digital Signal Generation.*

## Functions

- unsigned long [DSG::MaxHarms](#) ([DSG::DSGFrequency](#) const &frequency)

## 8.124 SignalGenerator.h

```

00001 //
00002 //  SignalGenerator.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__SignalGenerator__
00025 #define __DSG__SignalGenerator__
00026 #include "SignalProcess.h"
00027 #include "AudioSettings.h"
00028 #include "Sine.h"
00029 #include "Bounds.h"
00030 namespace DSG{
00031     /*!\brief DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation
00032     */
00033     class SignalGenerator:public DSG::SignalProcess{
00034     public:
00035         SignalGenerator();
00036         SignalGenerator(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00037         virtual ~SignalGenerator();
00038         virtual inline bool Perform(DSG::DSGSample& signal);
00039         virtual inline bool Perform(DSG::RingBuffer& signal);
00040         //Adds interface for control rate processing
00041         virtual inline DSG::DSGFrequency const& Frequency();
00042         virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00043         virtual inline DSG::DSGPhase const& Phase();
00044         virtual inline DSG::DSGPhase const& Phase(
DSG::DSGPhase const& value);
00045     protected:
00046         //extends sample rate interface
00047         inline void step();
00048         inline void sync();
00049         //-----
00050         DSG::DSGFrequency _frequency;//frequency in Hz
00051         DSG::DSGPhase _dt;//delta time (change in phase per sample) unit: phase 0-1
00052         DSG::DSGPhase _offset;//phase shift
00053         DSG::DSGPhase _phasor;//phase counter
00054         DSG::DSGSample _storage;//storage variable for calculations
00055     };
00056     inline unsigned long MaxHarms(DSG::DSGFrequency const& frequency){
00057         double _s = DSG::SampleRate()* 20000.0/DSG::SampleRate();
00058         _s/=frequency;
00059         return _s;
00060     }
00061 }
00062 inline bool DSG::SignalGenerator::Perform(
DSG::DSGSample& signal){
00063     signal=0;

```

```

00064     return false;
00065 }
00066 inline bool DSG::SignalGenerator::Perform(
    DSG::RingBuffer& signal){
00067     signal.Flush();
00068     return false;
00069 }
00070 inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
    (){
00071     return _frequency;
00072 }
00073 inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
    (DSG::DSGFrequency const& value){
00074     _frequency = DSG::EnforceBounds<0, 20000, DSG::DSGSample>(value);
00075     _dt = _frequency/DSG::SampleRate();
00076     return _frequency;
00077 }
00078 inline DSG::DSGPhase const& DSG::SignalGenerator::Phase() {
00079     return _offset;
00080 }
00081 inline DSG::DSGPhase const& DSG::SignalGenerator::Phase(
    DSG::DSGPhase const& value){
00082     _offset-=value;
00083     _phasor-=_offset;
00084     _offset=value;
00085     return _offset;
00086 }
00087 inline void DSG::SignalGenerator::step() {
00088     _phasor+=_dt;
00089     _phasor>1.0 ? --_phasor:0;
00090 }
00091 inline void DSG::SignalGenerator::sync() {
00092     _phasor=_offset;
00093 }
00094 #endif /* defined(__DSG__SignalGenerator__) */

```

## 8.125 SignalProcess.cpp File Reference

```
#include "SignalProcess.h"
```

## 8.126 SignalProcess.cpp

```

00001 //
00002 // SignalProcess.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "SignalProcess.h"
00025 DSG::SignalProcess::SignalProcess() {}
00026 DSG::SignalProcess::~SignalProcess() {}

```

## 8.127 SignalProcess.h File Reference

```
#include "DSGTypes.h"
#include "RingBuffer.h"
```

### Classes

- class [DSG::SignalProcess](#)  
*DSG::SignalProcess - Defines Base Interface For Audio Processing.*

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

## 8.128 SignalProcess.h

```
00001 //
00002 //  SignalProcess.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__SignalProcess__
00025 #define __DSG__SignalProcess__
00026 #include "DSGTypes.h"
00027 #include "RingBuffer.h"
00028 namespace DSG {
00029     /*!\brief DSG::SignalProcess - Defines Base Interface For Audio Processing
00030     */
00031     class SignalProcess{
00032     public:
00033         SignalProcess();
00034         virtual ~SignalProcess();
00035         //Defines Interface for sample rate processing
00036         virtual inline bool Perform(DSG::DSGSample& signal)=0;
00037         virtual inline bool Perform(DSG::RingBuffer& signal)=0;
00038     };
00039 }
00040 #endif /* defined(__DSG__SignalProcess__) */
```

## 8.129 Sinc.h File Reference

```
#include "PI.h"
#include "Sine.h"
#include "Denormal.h"
#include <type_traits>
#include "DSGMath.h"
```

## Namespaces

- **DSG**

*DSG - A Collection of tools for Digital Signal Generation.*

## Functions

- `template<typename decimal >`  
`decimal DSG::Sinc (decimal const &x)`  
*DSG::Sinc - Implements the Sinc() function ( $\sin(\text{PI} \times x) / \text{PI} \times x$ )*

## 8.130 Sinc.h

```

00001 //
00002 //  Sinc.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009  This file is part of the Digital Signal Generation Project or "DSG".
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019  GNU General Public License for more details.
00020
00021  You should have received a copy of the GNU General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__Sinc__
00025 #define __DSG__Sinc__
00026 #include "PI.h"
00027 #include "Sine.h"
00028 #include "Denormal.h"
00029 #include <type_traits>
00030 #include "DSGMath.h"
00031 namespace DSG{
00032     //!\brief DSG::Sinc - Implements the Sinc() function ( $\sin(\text{PI} \times x) / \text{PI} \times x$ )
00033     template<typename decimal>
00034     inline decimal Sinc(decimal const& x) {
00035         static_assert(std::is_floating_point<decimal>::value==true, "DSG::Sinc Function Requires Floating
Point Type");
00036         decimal pix;
00037         if (DSG::IsDenormal(x)) {
00038             return 1.0;
00039         }else{
00040             pix = PI*x;
00041             return DSG::Sin(pix)/pix;
00042         }
00043     }
00044 }
00045 #endif /* defined(__DSG__Sinc__) */

```

## 8.131 Sine.h File Reference

```

#include "LUT.h"
#include "PI.h"

```

## Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

## Macros

- `#define LUT_SIZE 16384`

## Enumerations

- enum **Sine\_Implementations**

## Functions

- double [DSG::Sin](#) (double const &x)  
*DSG::Sin() - General Purpose Sin Function, double precision.*
- float [DSG::Sin](#) (float const &x)  
*DSG::Sin() - General Purpose Sin Function, single precision.*
- double [DSG::Cos](#) (double const &x)  
*DSG::Cos() - General Purpose Cos Function, double precision.*
- float [DSG::Cos](#) (float const &x)  
*DSG::Cos() - General Purpose Cos Function, single precision.*

### 8.131.1 Macro Definition Documentation

#### 8.131.1.1 `#define LUT_SIZE 16384`

Definition at line 30 of file [Sine.h](#).

### 8.132 Sine.h

```
00001 //
00002 // Sine.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 9/16/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__Sine__
00025 #define __DSG__Sine__
00026 #include "LUT.h"
00027 #include "PI.h"
00028 namespace DSG {
00029     namespace {
00030         #define LUT_SIZE 16384
```



```

00031     typedef enum Sine_Implementations{
00032         /*!\brief DSG::Sine_Implementations - Specifies The Implementation Option For DSG::Sin<>()*/
00033         Sine_Taylor =1,
00034         Sine_LUT =2,
00035         Sine_Default = Sine_LUT
00036     }Sine_Implementations;
00037     /*!\brief DSG::Sin() - Templated Sin Function With Optional Implementation
00038     */
00039     template<unsigned implementation> inline double Sin(double const& x){
00040         return 0;
00041     }
00042     /*!\brief DSG::Sin() - Templated Cos Function With Optional Implementation
00043     */
00044     template<unsigned implementation> inline double Cos(double const& x){
00045         return 0;
00046     }
00047     template<> inline double Sin<Sine_LUT>(double const& x){
00048         static DSG::LUT<double, LUT_SIZE> _lut(&sin,
TWOPI);
00049         return _lut(x);
00050     }
00051     template<> inline double Cos<Sine_LUT>(double const& x){
00052         static DSG::LUT<double, LUT_SIZE> _lut(&cos,
TWOPI);
00053         return _lut(x);
00054     }
00055     template<> inline double Sin<Sine_Taylor>(double const& x){
00056         //taylor serie version here
00057         return 0;
00058     }
00059     template<> inline double Cos<Sine_Taylor>(double const& x){
00060         //taylor series version here
00061         return 0;
00062     }
00063 }
00064 /*!\brief DSG::Sin() - General Purpose Sin Function, double precision
00065 */
00066 inline double Sin(double const& x){
00067     return static_cast<double>(Sin<Sine_Default>(x)); //wrap default implementation as non template
00068 }
00069 /*!\brief DSG::Sin() - General Purpose Sin Function, single precision
00070 */
00071 inline float Sin(float const& x){
00072     return static_cast<float>(Sin<Sine_Default>(x));
00073 }
00074 /*!\brief DSG::Cos() - General Purpose Cos Function, double precision
00075 */
00076 inline double Cos(double const& x){
00077     return static_cast<double>(Cos<Sine_Default>(x)); //wrap default implementation as non template
00078 }
00079 /*!\brief DSG::Cos() - General Purpose Cos Function, single precision
00080 */
00081 inline float Cos(float const& x){
00082     return static_cast<float>(Cos<Sine_Default>(x));
00083 }
00084 }
00085 #endif /* defined(__DSG__Sine__) */

```

## 8.133 Sleep.h File Reference

```

#include <chrono>
#include <thread>

```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*

### Functions

- `template<typename integer >`  
`void DSG::Sleep (integer const &milliseconds)`

*DSG::Sleep* - Millisecond Sleep Function.

## 8.134 Sleep.h

```

00001 //
00002 // Sleep.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/5/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__Sleep__
00025 #define __DSG__Sleep__
00026 #include <chrono>
00027 #include <thread>
00028 namespace DSG{
00029     //!\brief DSG::Sleep - Millisecond Sleep Function
00030     template<typename integer>
00031     void Sleep(integer const& milliseconds){
00032         std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
00033     }
00034 }
00035 #endif /* defined(__DSG__Sleep__) */

```

## 8.135 White.h File Reference

```

#include "DSGTypes.h"
#include "Random.h"

```

### Namespaces

- [DSG](#)  
*DSG* - A Collection of tools for Digital Signal Generation.
- [DSG::Noise](#)  
*DSG::Noise* - Noise Generators.

### Functions

- `template<typename decimal = DSG::DSGSample>`  
`decimal DSG::Noise::White (decimal=0.0)`  
*DSG::Noise::White* - White Noise Generator Function.

## 8.136 White.h

```

00001 //
00002 // White.h

```

```

00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/14/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_White_h
00025 #define DSG_White_h
00026 #include "DSGTypes.h"
00027 #include "Random.h"
00028 namespace DSG{
00029 #ifdef DSG_Short_Names
00030     inline
00031 #endif
00032     namespace Noise{
00033         //!\brief DSG::Noise::White - White Noise Generator Function
00034         template<typename decimal = DSG::DSGSample>
00035         inline decimal White(decimal=0.0){
00036             return DSG::Random<decimal>();
00037         }
00038     }
00039 }
00040 #endif

```

## 8.137 Window.h File Reference

```
#include "LUT.h"
```

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Window](#)  
*DSG::Window - Window functions and utilities.*

### Functions

- template<typename decimal , unsigned long lutsz>  
void [DSG::Window::ApplyWindow](#) ([DSG::LUT](#)< decimal, lutsz > &lut, decimal(&>windowFunction)(decimal const &), decimal range=1.0)  
*DSG::Window::ApplyWindow - Apply a window function to a [LUT](#).*
- template<typename decimal , unsigned long lutsz>  
void [DSG::Window::ApplyWindow](#) ([DSG::LUT](#)< decimal, lutsz > &lut, decimal(&>windowFunction)(decimal), decimal range=1.0)  
*DSG::Window::ApplyWindow - Apply a window function to a [LUT](#).*

## 8.138 Window.h

```
00001 //
```

```

00002 // Window.h
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 10/17/14.
00006 // Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 /*
00009 This file is part of the Digital Signal Generation Project or "DSG".
00010
00011 DSG is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU General Public License as published by
00013 the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 DSG is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU General Public License for more details.
00020
00021 You should have received a copy of the GNU General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef DSG_Window_h
00025 #define DSG_Window_h
00026 #include "LUT.h"
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     /*! \brief DSG::Window - Window functions and utilities
00032     namespace Window{
00033         /*! \brief DSG::Window::ApplyWindow - Apply a window function to a LUT
00034         template<typename decimal, unsigned long lutsizes>
00035         void ApplyWindow(DSG::LUT<decimal, lutsizes>& lut, decimal (&
00036         windowFunction)(decimal const&), decimal range = 1.0){
00037             decimal step = range/(decimal)lut.Size();
00038             decimal phs=0;
00039             for (int i=0; i<lut.Size(); ++i) {
00040                 lut[i]*=windowFunction(phs);
00041                 phs+=step;
00042             }
00043             /*! \brief DSG::Window::ApplyWindow - Apply a window function to a LUT
00044             template<typename decimal, unsigned long lutsizes>
00045             void ApplyWindow(DSG::LUT<decimal, lutsizes>& lut, decimal (&
00046             windowFunction)(decimal), decimal range = 1.0){
00047                 decimal step = range/(decimal)lut.Size();
00048                 decimal phs=0;
00049                 for (int i=0; i<lut.Size(); ++i) {
00050                     lut[i]*=windowFunction(phs);
00051                     phs+=step;
00052                 }
00053             }
00054         }
00055     #endif

```

# Index

DSG::Factorial

value, [58](#)

DSG::Factorial< 0 >

value, [59](#)

value

DSG::Factorial, [58](#)

DSG::Factorial< 0 >, [59](#)