

DSG

Generated by Doxygen 1.8.8

Sun Nov 30 2014 00:49:54



# Contents

<b>1</b>	<b>DSG - A Collection Of Tools For Digital Signal Generation</b>	<b>1</b>
1.1	Intoduction . . . . .	1
1.1.1	Scope . . . . .	1
1.2	License . . . . .	1
<b>2</b>	<b>Todo List</b>	<b>3</b>
<b>3</b>	<b>Bug List</b>	<b>5</b>
<b>4</b>	<b>Namespace Index</b>	<b>7</b>
4.1	Namespace List . . . . .	7
<b>5</b>	<b>Hierarchical Index</b>	<b>9</b>
5.1	Class Hierarchy . . . . .	9
<b>6</b>	<b>Class Index</b>	<b>11</b>
6.1	Class List . . . . .	11
<b>7</b>	<b>File Index</b>	<b>13</b>
7.1	File List . . . . .	13
<b>8</b>	<b>Namespace Documentation</b>	<b>15</b>
8.1	DSG Namespace Reference . . . . .	15
8.1.1	Detailed Description . . . . .	17
8.1.2	Typedef Documentation . . . . .	17
8.1.2.1	DSGFrequency . . . . .	17
8.1.2.2	DSGPhase . . . . .	18
8.1.2.3	DSGSample . . . . .	18
8.1.3	Function Documentation . . . . .	18
8.1.3.1	Abs . . . . .	18
8.1.3.2	AddSampleRateListener . . . . .	18
8.1.3.3	ArrayToRing . . . . .	18
8.1.3.4	AssertBounds . . . . .	18
8.1.3.5	Cos . . . . .	19

8.1.3.6	Cos	19
8.1.3.7	CosineInterpolate	19
8.1.3.8	CubicInterpolate	19
8.1.3.9	DSF	19
8.1.3.10	EnforceBounds	20
8.1.3.11	HermiteInterpolate	20
8.1.3.12	IsDenormal	20
8.1.3.13	LinearInterpolate	20
8.1.3.14	MaxHarms	21
8.1.3.15	Nyquist	21
8.1.3.16	Pow	21
8.1.3.17	RingToArray	21
8.1.3.18	SampleRate	21
8.1.3.19	SampleRate	22
8.1.3.20	Sin	22
8.1.3.21	Sin	22
8.1.3.22	Sinc	22
8.1.3.23	Sleep	22
8.1.3.24	StaticAssertBounds	23
8.1.3.25	VerifySampleRateSet	23
8.2	DSG::Analog Namespace Reference	23
8.2.1	Detailed Description	23
8.3	DSG::BLIT Namespace Reference	23
8.3.1	Detailed Description	24
8.4	DSG::DPW Namespace Reference	24
8.4.1	Detailed Description	25
8.4.2	Function Documentation	25
8.4.2.1	DPW_Polynomial	25
8.4.2.2	DPW_Polynomial< 1 >	25
8.4.2.3	DPW_Polynomial< 2 >	25
8.4.2.4	DPW_Polynomial< 3 >	25
8.4.2.5	DPW_Polynomial< 4 >	25
8.4.2.6	DPW_Polynomial< 5 >	26
8.4.2.7	DPW_Polynomial< 6 >	26
8.5	DSG::EPTR Namespace Reference	26
8.5.1	Detailed Description	26
8.6	DSG::Filter Namespace Reference	26
8.6.1	Detailed Description	26
8.7	DSG::Fourier Namespace Reference	27
8.7.1	Detailed Description	27

8.8	DSG::MIDI Namespace Reference	27
8.8.1	Detailed Description	27
8.8.2	Function Documentation	27
8.8.2.1	FTOM	27
8.8.2.2	MTOF	28
8.9	DSG::Noise Namespace Reference	28
8.9.1	Detailed Description	28
8.9.2	Function Documentation	28
8.9.2.1	Gaussian	28
8.9.2.2	Pink	29
8.9.2.3	Random	29
8.9.2.4	White	29
8.10	DSG::Window Namespace Reference	29
8.10.1	Detailed Description	30
8.10.2	Function Documentation	30
8.10.2.1	ApplyWindow	30
8.10.2.2	ApplyWindow	30
8.10.2.3	Blackman	30
9	<b>Class Documentation</b>	<b>33</b>
9.1	DSG::Analog::AnalogSaw Class Reference	33
9.1.1	Detailed Description	33
9.1.2	Constructor & Destructor Documentation	34
9.1.2.1	AnalogSaw	34
9.1.2.2	AnalogSaw	34
9.1.2.3	~AnalogSaw	34
9.1.3	Member Function Documentation	34
9.1.3.1	Perform	34
9.1.3.2	Perform	34
9.1.4	Member Data Documentation	35
9.1.4.1	_stor	35
9.2	DSG::Analog::AnalogSquare Class Reference	35
9.2.1	Detailed Description	35
9.2.2	Constructor & Destructor Documentation	35
9.2.2.1	AnalogSquare	35
9.2.2.2	AnalogSquare	36
9.2.2.3	~AnalogSquare	36
9.2.3	Member Function Documentation	36
9.2.3.1	Perform	36
9.2.3.2	Perform	36

9.3	DSG::Analog::AnalogTriangle Class Reference	36
9.3.1	Detailed Description	37
9.3.2	Constructor & Destructor Documentation	37
9.3.2.1	AnalogTriangle	37
9.3.2.2	AnalogTriangle	37
9.3.2.3	~AnalogTriangle	37
9.3.3	Member Function Documentation	38
9.3.3.1	Perform	38
9.3.3.2	Perform	38
9.3.4	Member Data Documentation	38
9.3.4.1	_stor	38
9.4	DSG::AudioSettings Class Reference	38
9.4.1	Detailed Description	39
9.4.2	Member Function Documentation	39
9.4.2.1	AddSampleRateListener	39
9.4.2.2	IsSampleRateSet	39
9.4.2.3	Nyquist	39
9.4.2.4	SampleRate	39
9.4.2.5	SampleRate	40
9.4.3	Member Data Documentation	40
9.4.3.1	_listeners	40
9.4.3.2	_nyquist	40
9.4.3.3	_sampleRate	40
9.4.3.4	_set	40
9.5	DSG::BLIT::Blit Class Reference	40
9.5.1	Detailed Description	41
9.5.2	Constructor & Destructor Documentation	41
9.5.2.1	Blit	41
9.5.2.2	Blit	41
9.5.2.3	~Blit	41
9.5.3	Member Function Documentation	42
9.5.3.1	Frequency	42
9.5.3.2	Perform	42
9.5.3.3	Perform	42
9.5.4	Member Data Documentation	42
9.5.4.1	_h	42
9.5.4.2	a_	43
9.5.4.3	denominator	43
9.5.4.4	m_	43
9.5.4.5	p_	43

9.5.4.6	value	43
9.6	DSG::BLIT::BlitSaw Class Reference	43
9.6.1	Detailed Description	44
9.6.2	Constructor & Destructor Documentation	44
9.6.2.1	BlitSaw	44
9.6.2.2	BlitSaw	44
9.6.2.3	~BlitSaw	44
9.6.3	Member Function Documentation	44
9.6.3.1	Frequency	44
9.6.3.2	Perform	45
9.6.3.3	Perform	45
9.6.4	Member Data Documentation	45
9.6.4.1	C2_	45
9.6.4.2	Register_	45
9.7	DSG::BLIT::BlitSquare Class Reference	45
9.7.1	Detailed Description	46
9.8	DSG::BLIT::BlitTriangle Class Reference	46
9.8.1	Detailed Description	46
9.9	DSG::Buffer Class Reference	46
9.9.1	Detailed Description	47
9.9.2	Constructor & Destructor Documentation	47
9.9.2.1	Buffer	47
9.9.2.2	Buffer	47
9.9.2.3	Buffer	47
9.9.2.4	~Buffer	48
9.9.3	Member Function Documentation	48
9.9.3.1	operator=	48
9.9.3.2	operator[]	48
9.9.3.3	Size	48
9.9.4	Member Data Documentation	48
9.9.4.1	_buffer	48
9.9.4.2	_size	48
9.10	DSG::Filter::DCBlocker Class Reference	49
9.10.1	Detailed Description	49
9.10.2	Constructor & Destructor Documentation	49
9.10.2.1	DCBlocker	49
9.10.2.2	~DCBlocker	49
9.10.3	Member Function Documentation	50
9.10.3.1	Perform	50
9.10.3.2	Perform	50

9.10.4	Member Data Documentation . . . . .	50
9.10.4.1	_a . . . . .	50
9.10.4.2	_temp . . . . .	50
9.10.4.3	count . . . . .	50
9.10.4.4	x . . . . .	50
9.10.4.5	xm1 . . . . .	50
9.10.4.6	ym1 . . . . .	51
9.11	DSG::Delay< maxLength > Class Template Reference . . . . .	51
9.11.1	Detailed Description . . . . .	51
9.11.2	Constructor & Destructor Documentation . . . . .	52
9.11.2.1	Delay . . . . .	52
9.11.2.2	Delay . . . . .	52
9.11.2.3	~Delay . . . . .	52
9.11.3	Member Function Documentation . . . . .	52
9.11.3.1	increment . . . . .	52
9.11.3.2	Length . . . . .	52
9.11.3.3	Length . . . . .	53
9.11.3.4	Perform . . . . .	53
9.11.3.5	Perform . . . . .	53
9.11.4	Member Data Documentation . . . . .	53
9.11.4.1	_buffer . . . . .	53
9.11.4.2	_delay . . . . .	53
9.11.4.3	_index . . . . .	53
9.11.4.4	_max . . . . .	54
9.11.4.5	_swap . . . . .	54
9.11.4.6	_temp . . . . .	54
9.11.4.7	count . . . . .	54
9.12	DSG::DPW::DPW_Differentiator< order > Class Template Reference . . . . .	54
9.12.1	Detailed Description . . . . .	54
9.12.2	Constructor & Destructor Documentation . . . . .	54
9.12.2.1	DPW_Differentiator . . . . .	54
9.13	DSG::DPW::DPW_Differentiator< 1 > Class Template Reference . . . . .	55
9.13.1	Detailed Description . . . . .	55
9.13.2	Member Function Documentation . . . . .	55
9.13.2.1	operator() . . . . .	55
9.14	DSG::DPW::DPW_Differentiator< 2 > Class Template Reference . . . . .	55
9.14.1	Detailed Description . . . . .	56
9.14.2	Member Function Documentation . . . . .	56
9.14.2.1	operator() . . . . .	56
9.14.3	Member Data Documentation . . . . .	56



9.14.3.1	<a href="#">_delay</a>	56
9.14.3.2	<a href="#">output</a>	56
9.15	<a href="#">DSG::DPW::DPW_Differentiator&lt; 3 &gt; Class Template Reference</a>	56
9.15.1	<a href="#">Detailed Description</a>	56
9.15.2	<a href="#">Member Function Documentation</a>	57
9.15.2.1	<a href="#">operator()</a>	57
9.15.3	<a href="#">Member Data Documentation</a>	57
9.15.3.1	<a href="#">_delay</a>	57
9.15.3.2	<a href="#">output</a>	57
9.16	<a href="#">DSG::DPW::DPW_Differentiator&lt; 4 &gt; Class Template Reference</a>	57
9.16.1	<a href="#">Detailed Description</a>	57
9.16.2	<a href="#">Member Function Documentation</a>	58
9.16.2.1	<a href="#">operator()</a>	58
9.16.3	<a href="#">Member Data Documentation</a>	58
9.16.3.1	<a href="#">_delay</a>	58
9.16.3.2	<a href="#">output</a>	58
9.17	<a href="#">DSG::DPW::DPW_Differentiator&lt; 5 &gt; Class Template Reference</a>	58
9.17.1	<a href="#">Detailed Description</a>	58
9.17.2	<a href="#">Member Function Documentation</a>	59
9.17.2.1	<a href="#">operator()</a>	59
9.17.3	<a href="#">Member Data Documentation</a>	59
9.17.3.1	<a href="#">_delay</a>	59
9.17.3.2	<a href="#">output</a>	59
9.18	<a href="#">DSG::DPW::DPW_Differentiator&lt; 6 &gt; Class Template Reference</a>	59
9.18.1	<a href="#">Detailed Description</a>	60
9.18.2	<a href="#">Member Function Documentation</a>	60
9.18.2.1	<a href="#">operator()</a>	60
9.18.3	<a href="#">Member Data Documentation</a>	60
9.18.3.1	<a href="#">_delay</a>	60
9.18.3.2	<a href="#">output</a>	60
9.19	<a href="#">DSG::DPW::DPWSaw&lt; order &gt; Class Template Reference</a>	60
9.19.1	<a href="#">Detailed Description</a>	61
9.19.2	<a href="#">Constructor &amp; Destructor Documentation</a>	61
9.19.2.1	<a href="#">DPWSaw</a>	61
9.19.2.2	<a href="#">DPWSaw</a>	61
9.19.2.3	<a href="#">~DPWSaw</a>	62
9.19.3	<a href="#">Member Function Documentation</a>	62
9.19.3.1	<a href="#">Perform</a>	62
9.19.3.2	<a href="#">Perform</a>	62
9.19.4	<a href="#">Member Data Documentation</a>	62

9.19.4.1	<code>_diff</code>	62
9.19.4.2	<code>_register</code>	62
9.20	<code>DSG::EPTR::EPTR</code> Class Reference	63
9.20.1	Detailed Description	63
9.20.2	Constructor & Destructor Documentation	63
9.20.2.1	<code>EPTR</code>	63
9.20.2.2	<code>EPTR</code>	64
9.20.2.3	<code>~EPTR</code>	64
9.20.3	Member Function Documentation	64
9.20.3.1	<code>Perform</code>	64
9.20.3.2	<code>Perform</code>	64
9.20.4	Member Data Documentation	64
9.20.4.1	<code>_register</code>	64
9.21	<code>DSG::Factorial&lt; N &gt;</code> Struct Template Reference	65
9.21.1	Detailed Description	65
9.21.2	Member Enumeration Documentation	65
9.21.2.1	anonymous enum	65
9.22	<code>DSG::Factorial&lt; 0 &gt;</code> Struct Template Reference	65
9.22.1	Detailed Description	65
9.22.2	Member Enumeration Documentation	66
9.22.2.1	anonymous enum	66
9.23	<code>DSG::Filter::FilterBase</code> Class Reference	66
9.23.1	Detailed Description	66
9.23.2	Constructor & Destructor Documentation	67
9.23.2.1	<code>FilterBase</code>	67
9.23.2.2	<code>~FilterBase</code>	67
9.23.3	Member Function Documentation	67
9.23.3.1	<code>Cutoff</code>	67
9.23.3.2	<code>Perform</code>	67
9.23.3.3	<code>Perform</code>	67
9.23.4	Member Data Documentation	68
9.23.4.1	<code>_temp</code>	68
9.23.4.2	<code>count</code>	68
9.24	<code>DSG::Fourier::Fourier</code> Class Reference	68
9.24.1	Detailed Description	68
9.24.2	Constructor & Destructor Documentation	69
9.24.2.1	<code>Fourier</code>	69
9.24.2.2	<code>Fourier</code>	69
9.24.2.3	<code>~Fourier</code>	69
9.24.3	Member Function Documentation	69

9.24.3.1	Frequency	69
9.24.3.2	Perform	69
9.24.3.3	Perform	70
9.24.4	Member Data Documentation	70
9.24.4.1	_a	70
9.24.4.2	_h	70
9.24.4.3	i	70
9.24.4.4	phs	70
9.24.4.5	value	70
9.25	DSG::Fourier::FourierSeriesGenerator Class Reference	70
9.25.1	Detailed Description	71
9.25.2	Member Typedef Documentation	71
9.25.2.1	FourierSeries	71
9.25.3	Constructor & Destructor Documentation	71
9.25.3.1	FourierSeriesGenerator	71
9.25.3.2	FourierSeriesGenerator	72
9.25.3.3	~FourierSeriesGenerator	72
9.25.4	Member Function Documentation	72
9.25.4.1	Perform	72
9.25.4.2	Perform	72
9.25.4.3	Series	72
9.25.4.4	Series	73
9.25.5	Member Data Documentation	73
9.25.5.1	_series	73
9.25.5.2	value	73
9.26	DSG::Fourier::FourierSquare Class Reference	73
9.26.1	Detailed Description	74
9.26.2	Constructor & Destructor Documentation	74
9.26.2.1	FourierSquare	74
9.26.2.2	FourierSquare	74
9.26.2.3	~FourierSquare	74
9.26.3	Member Function Documentation	74
9.26.3.1	Frequency	74
9.26.3.2	Perform	75
9.26.3.3	Perform	75
9.26.4	Member Data Documentation	75
9.26.4.1	_a	75
9.26.4.2	_h	75
9.26.4.3	i	75
9.26.4.4	phs	75

9.26.4.5	value	75
9.27	DSG::Fourier::FourierTriangle Class Reference	76
9.27.1	Detailed Description	76
9.27.2	Constructor & Destructor Documentation	76
9.27.2.1	FourierTriangle	76
9.27.2.2	FourierTriangle	77
9.27.2.3	~FourierTriangle	77
9.27.3	Member Function Documentation	77
9.27.3.1	Frequency	77
9.27.3.2	Perform	77
9.27.3.3	Perform	77
9.27.4	Member Data Documentation	78
9.27.4.1	_a	78
9.27.4.2	_h	78
9.27.4.3	i	78
9.27.4.4	phs	78
9.27.4.5	value	78
9.28	DSG::GenericGenerator Class Reference	78
9.28.1	Detailed Description	79
9.28.2	Constructor & Destructor Documentation	79
9.28.2.1	GenericGenerator	79
9.28.2.2	GenericGenerator	79
9.28.2.3	~GenericGenerator	79
9.28.3	Member Function Documentation	79
9.28.3.1	Perform	79
9.28.3.2	Perform	80
9.28.4	Member Data Documentation	80
9.28.4.1	_callback	80
9.29	DSG::Fourier::Harmonic Class Reference	80
9.29.1	Detailed Description	80
9.29.2	Constructor & Destructor Documentation	81
9.29.2.1	Harmonic	81
9.29.2.2	Harmonic	81
9.29.2.3	~Harmonic	81
9.29.3	Member Function Documentation	81
9.29.3.1	Amplitude	81
9.29.3.2	Amplitude	81
9.29.3.3	Ratio	81
9.29.3.4	Ratio	81
9.29.4	Member Data Documentation	82

9.29.4.1	<a href="#">_amplitude</a>	82
9.29.4.2	<a href="#">_ratio</a>	82
9.30	<a href="#">DSG::Filter::LeakyIntegrator Class Reference</a>	82
9.30.1	<a href="#">Detailed Description</a>	82
9.30.2	<a href="#">Constructor &amp; Destructor Documentation</a>	83
9.30.2.1	<a href="#">LeakyIntegrator</a>	83
9.30.2.2	<a href="#">LeakyIntegrator</a>	83
9.30.2.3	<a href="#">~LeakyIntegrator</a>	83
9.30.3	<a href="#">Member Function Documentation</a>	83
9.30.3.1	<a href="#">Cutoff</a>	83
9.30.3.2	<a href="#">Perform</a>	83
9.30.3.3	<a href="#">Perform</a>	84
9.30.4	<a href="#">Member Data Documentation</a>	84
9.30.4.1	<a href="#">a</a>	84
9.30.4.2	<a href="#">b</a>	84
9.30.4.3	<a href="#">x1</a>	84
9.30.4.4	<a href="#">y</a>	84
9.30.4.5	<a href="#">y1</a>	84
9.31	<a href="#">DSG::LUT&lt; element, size &gt; Class Template Reference</a>	84
9.31.1	<a href="#">Detailed Description</a>	85
9.31.2	<a href="#">Member Typedef Documentation</a>	85
9.31.2.1	<a href="#">FillFunction</a>	85
9.31.2.2	<a href="#">FillFunctionConstRef</a>	85
9.31.3	<a href="#">Constructor &amp; Destructor Documentation</a>	85
9.31.3.1	<a href="#">LUT</a>	85
9.31.3.2	<a href="#">LUT</a>	86
9.31.3.3	<a href="#">LUT</a>	86
9.31.3.4	<a href="#">~LUT</a>	86
9.31.4	<a href="#">Member Function Documentation</a>	86
9.31.4.1	<a href="#">operator()</a>	86
9.31.4.2	<a href="#">operator[]</a>	86
9.31.4.3	<a href="#">operator[]</a>	87
9.31.4.4	<a href="#">Size</a>	87
9.31.5	<a href="#">Member Data Documentation</a>	87
9.31.5.1	<a href="#">_size</a>	87
9.31.5.2	<a href="#">_table</a>	87
9.31.5.3	<a href="#">phs</a>	87
9.32	<a href="#">DSG::NoiseGenerator Class Reference</a>	87
9.32.1	<a href="#">Detailed Description</a>	88
9.32.2	<a href="#">Constructor &amp; Destructor Documentation</a>	88

9.32.2.1	NoiseGenerator	88
9.32.2.2	~NoiseGenerator	88
9.32.3	Member Function Documentation	88
9.32.3.1	Perform	88
9.32.3.2	Perform	89
9.32.4	Member Data Documentation	89
9.32.4.1	_function	89
9.32.4.2	_storage	89
9.33	DSG::Phasor Class Reference	89
9.33.1	Detailed Description	91
9.33.2	Constructor & Destructor Documentation	91
9.33.2.1	Phasor	91
9.33.2.2	Phasor	91
9.33.2.3	~Phasor	91
9.33.3	Member Function Documentation	91
9.33.3.1	Frequency	91
9.33.3.2	Frequency	91
9.33.3.3	Phase	92
9.33.3.4	Phase	92
9.33.3.5	step	92
9.33.3.6	sync	92
9.33.4	Member Data Documentation	92
9.33.4.1	_dt	92
9.33.4.2	_frequency	92
9.33.4.3	_offset	92
9.33.4.4	_phasor	93
9.34	DSG::RingBuffer Class Reference	93
9.34.1	Detailed Description	94
9.34.2	Constructor & Destructor Documentation	94
9.34.2.1	RingBuffer	94
9.34.2.2	RingBuffer	94
9.34.2.3	RingBuffer	94
9.34.2.4	~RingBuffer	94
9.34.3	Member Function Documentation	94
9.34.3.1	Count	94
9.34.3.2	Empty	95
9.34.3.3	Flush	95
9.34.3.4	Full	95
9.34.3.5	make_pow_2	95
9.34.3.6	next	95

9.34.3.7	operator=	95
9.34.3.8	Read	96
9.34.3.9	Write	96
9.34.4	Friends And Related Function Documentation	96
9.34.4.1	operator<<	96
9.34.4.2	operator>>	96
9.34.5	Member Data Documentation	96
9.34.5.1	_count	96
9.34.5.2	_read	96
9.34.5.3	_write	96
9.34.5.4	MASK	97
9.34.5.5	read	97
9.34.5.6	write	97
9.35	DSG::SignalGenerator Class Reference	97
9.35.1	Detailed Description	98
9.35.2	Constructor & Destructor Documentation	98
9.35.2.1	SignalGenerator	98
9.35.2.2	SignalGenerator	98
9.35.2.3	~SignalGenerator	98
9.35.3	Member Function Documentation	98
9.35.3.1	Perform	98
9.35.3.2	Perform	99
9.35.3.3	SampleRateChanged	99
9.35.4	Member Data Documentation	99
9.35.4.1	_storage	99
9.36	DSG::SignalProcess Class Reference	99
9.36.1	Detailed Description	100
9.36.2	Constructor & Destructor Documentation	100
9.36.2.1	SignalProcess	100
9.36.2.2	~SignalProcess	100
9.36.3	Member Function Documentation	100
9.36.3.1	Perform	101
9.36.3.2	Perform	101
9.36.3.3	SampleRateChanged	101
<b>10</b>	<b>File Documentation</b>	<b>103</b>
10.1	AnalogSaw.cpp File Reference	103
10.2	AnalogSaw.cpp	103
10.3	AnalogSaw.h File Reference	103
10.4	AnalogSaw.h	104

10.5 AnalogSquare.cpp File Reference . . . . .	105
10.6 AnalogSquare.cpp . . . . .	105
10.7 AnalogSquare.h File Reference . . . . .	105
10.8 AnalogSquare.h . . . . .	105
10.9 AnalogTriangle.cpp File Reference . . . . .	106
10.10AnalogTriangle.cpp . . . . .	106
10.11AnalogTriangle.h File Reference . . . . .	107
10.12AnalogTriangle.h . . . . .	107
10.13AudioSettings.cpp File Reference . . . . .	108
10.14AudioSettings.cpp . . . . .	108
10.15AudioSettings.h File Reference . . . . .	109
10.15.1 Macro Definition Documentation . . . . .	110
10.15.1.1 SampleRateDefault . . . . .	110
10.16AudioSettings.h . . . . .	110
10.17Blackman.h File Reference . . . . .	111
10.18Blackman.h . . . . .	111
10.19BLIT.cpp File Reference . . . . .	112
10.20BLIT.cpp . . . . .	112
10.21BLIT.h File Reference . . . . .	112
10.22BLIT.h . . . . .	113
10.23BLITSaw.cpp File Reference . . . . .	114
10.24BLITSaw.cpp . . . . .	114
10.25BLITSaw.h File Reference . . . . .	114
10.26BLITSaw.h . . . . .	115
10.27BLITSquare.cpp File Reference . . . . .	116
10.28BLITSquare.cpp . . . . .	116
10.29BLITSquare.h File Reference . . . . .	116
10.30BLITSquare.h . . . . .	117
10.31BLITTriangle.cpp File Reference . . . . .	117
10.32BLITTriangle.cpp . . . . .	117
10.33BLITTriangle.h File Reference . . . . .	118
10.34BLITTriangle.h . . . . .	118
10.35Bounds.h File Reference . . . . .	119
10.36Bounds.h . . . . .	119
10.37Buffer.cpp File Reference . . . . .	120
10.38Buffer.cpp . . . . .	120
10.39Buffer.h File Reference . . . . .	120
10.40Buffer.h . . . . .	121
10.41BufferConversion.h File Reference . . . . .	121
10.42BufferConversion.h . . . . .	122



10.43DCBlocker.cpp File Reference . . . . .	122
10.44DCBlocker.cpp . . . . .	123
10.45DCBlocker.h File Reference . . . . .	123
10.46DCBlocker.h . . . . .	123
10.47Delay.h File Reference . . . . .	124
10.48Delay.h . . . . .	125
10.49Denormal.h File Reference . . . . .	126
10.50Denormal.h . . . . .	126
10.51DPW.h File Reference . . . . .	127
10.52DPW.h . . . . .	128
10.53DPWSaw.h File Reference . . . . .	130
10.54DPWSaw.h . . . . .	131
10.55Driver.cpp File Reference . . . . .	132
10.55.1 Macro Definition Documentation . . . . .	132
10.55.1.1 BufferSize . . . . .	132
10.55.2 Function Documentation . . . . .	132
10.55.2.1 Callback . . . . .	132
10.55.2.2 DriverExit . . . . .	133
10.55.2.3 DriverInit . . . . .	133
10.55.3 Variable Documentation . . . . .	133
10.55.3.1 _buffer . . . . .	133
10.55.3.2 stream . . . . .	133
10.56Driver.cpp . . . . .	133
10.57Driver.h File Reference . . . . .	135
10.57.1 Function Documentation . . . . .	135
10.57.1.1 Callback . . . . .	135
10.57.1.2 DriverExit . . . . .	135
10.57.1.3 DriverInit . . . . .	136
10.58Driver.h . . . . .	136
10.59DSF.h File Reference . . . . .	136
10.60DSF.h . . . . .	137
10.61DSG.h File Reference . . . . .	138
10.61.1 Macro Definition Documentation . . . . .	138
10.61.1.1 DSG_Short_Names . . . . .	138
10.62DSG.h . . . . .	139
10.63DSGMath.h File Reference . . . . .	140
10.64DSGMath.h . . . . .	140
10.65DSGTypes.h File Reference . . . . .	141
10.66DSGTypes.h . . . . .	141
10.67EPTRSaw.cpp File Reference . . . . .	142

10.68EPTRSaw.cpp . . . . .	142
10.69EPTRSaw.h File Reference . . . . .	142
10.70EPTRSaw.h . . . . .	143
10.71Filter.cpp File Reference . . . . .	144
10.72Filter.cpp . . . . .	144
10.73Filter.h File Reference . . . . .	144
10.74Filter.h . . . . .	145
10.75FourierSaw.cpp File Reference . . . . .	145
10.76FourierSaw.cpp . . . . .	146
10.77FourierSaw.h File Reference . . . . .	146
10.78FourierSaw.h . . . . .	146
10.79FourierSeries.cpp File Reference . . . . .	147
10.80FourierSeries.cpp . . . . .	148
10.81FourierSeries.h File Reference . . . . .	148
10.82FourierSeries.h . . . . .	149
10.83FourierSquare.cpp File Reference . . . . .	150
10.84FourierSquare.cpp . . . . .	150
10.85FourierSquare.h File Reference . . . . .	150
10.86FourierSquare.h . . . . .	151
10.87FourierTriangle.cpp File Reference . . . . .	152
10.88FourierTriangle.cpp . . . . .	152
10.89FourierTriangle.h File Reference . . . . .	152
10.90FourierTriangle.h . . . . .	153
10.91Gaussian.h File Reference . . . . .	154
10.92Gaussian.h . . . . .	154
10.93GenericGenerator.cpp File Reference . . . . .	155
10.94GenericGenerator.cpp . . . . .	155
10.95GenericGenerator.h File Reference . . . . .	155
10.96GenericGenerator.h . . . . .	156
10.97Interpolate.h File Reference . . . . .	157
10.98Interpolate.h . . . . .	157
10.99Leaky.cpp File Reference . . . . .	158
10.100Leaky.cpp . . . . .	158
10.101Leaky.h File Reference . . . . .	159
10.102Leaky.h . . . . .	159
10.103LUT.h File Reference . . . . .	160
10.104LUT.h . . . . .	161
10.105MTOF.cpp File Reference . . . . .	162
10.106MTOF.cpp . . . . .	162
10.107MTOF.h File Reference . . . . .	162

10.108	TOF.h	163
10.109	Noise.h File Reference	163
10.110	Noise.h	163
10.111	NoiseGenerator.cpp File Reference	164
10.112	NoiseGenerator.cpp	164
10.113	NoiseGenerator.h File Reference	164
10.114	NoiseGenerator.h	165
10.115	Phasor.cpp File Reference	166
10.116	Phasor.cpp	166
10.117	Phasor.h File Reference	166
10.118	Phasor.h	166
10.119	PI.h File Reference	167
10.119.1	Macro Definition Documentation	168
10.119.1.1	PI	168
10.119.1.2	TWOPI	168
10.120	PI.h	168
10.121	Pink.h File Reference	168
10.122	Pink.h	169
10.123	Random.h File Reference	169
10.123.1	Variable Documentation	170
10.123.1.1	max	170
10.124	Random.h	170
10.125	RingBuffer.cpp File Reference	171
10.126	RingBuffer.cpp	171
10.127	RingBuffer.h File Reference	171
10.128	RingBuffer.h	172
10.129	SignalGenerator.cpp File Reference	173
10.130	SignalGenerator.cpp	173
10.131	SignalGenerator.h File Reference	174
10.132	SignalGenerator.h	174
10.133	SignalProcess.cpp File Reference	175
10.134	SignalProcess.cpp	175
10.135	SignalProcess.h File Reference	176
10.136	SignalProcess.h	176
10.137	Sinc.h File Reference	177
10.138	Sinc.h	177
10.139	Sine.h File Reference	178
10.139.1	Macro Definition Documentation	178
10.139.1.1	LUT_SIZE	178
10.140	Sine.h	178

---

10.14 <del>3</del> Sleep.h File Reference . . . . .	179
10.14 <del>2</del> Sleep.h . . . . .	180
10.14 <del>3</del> White.h File Reference . . . . .	180
10.14 <del>4</del> White.h . . . . .	181
10.14 <del>5</del> Window.h File Reference . . . . .	181
10.14 <del>6</del> Window.h . . . . .	182
<b>Index</b>	<b>183</b>

# Chapter 1

## DSG - A Collection Of Tools For Digital Signal Generation

### 1.1 Introduction

The Digital Signal Generation Project or [DSG](#) is a collection of tools used for the generation of digital signals, more specifically the generation of band-limited waveforms.

#### 1.1.1 Scope

Though [DSG](#) has a focus on Bandlimited Waveform Generation it is not limited to it. [DSG](#) defines a signal processing interface that is compatible with any form of audio based signal processing work. The interface defined in [DSG::SignalProcess](#) is the base interface for signal processing in [DSG](#). It is further expanded by [DSG::SignalGenerator](#) which adds functionality geared towards waveform generation. See the documentation for each for their specifics.

### 1.2 License

[DSG](#) is released under the Lesser GNU Public License (LGPL).

A copy of the LGPL and the GNU Public License should be included with the distribution in the files: COPYING (GPL), and COPYING.LESSER (LGPL). Additionally each source file should contain a copy of the license notice which reads as follows:

#### Copyright

This file is part of the Digital Signal Generation Project or "DSG".

[DSG](#) is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

[DSG](#) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with [DSG](#). If not, see <http://www.gnu.org/licenses/>.



## Chapter 2

# Todo List

### Namespace **DSG**

Increase documentation level. Add documentation for every variable, parameter...

Implement Blep Based Algorithms

### 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

### Member **DSG::Cos (double const &x)**

Implement Taylor Series implementation of Cos Function

### Class **DSG::DPW::DPW\_Differentiator< order >**

Fix **DSG::DPW::DPW\_Differentiator** algorithms for orders 4-6

### Class **DSG::EPTR::EPTRSaw**

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

### Class **DSG::LUT< element, size >**

Implement interpolation into lookup algorithm

### Member **DSG::Sin (double const &x)**

Implement Taylor Series implementation of Sin Function





## Chapter 3

# Bug List

**Class** `DSG::DPW::DPW_Differentiator< 4 >`

Causes major clipping

**Class** `DSG::DPW::DPW_Differentiator< 5 >`

Causes major clipping

**Class** `DSG::DPW::DPW_Differentiator< 6 >`

Causes major clipping

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

Algorithm is not performing in a band limited manor



## Chapter 4

# Namespace Index

### 4.1 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 . . . . .	<a href="#">15</a>
<a href="#">DSG::Analog</a>	<a href="#">DSG::Analog</a> - Namespace Containing <a href="#">Analog</a> Style Oscillators . . . . .	<a href="#">23</a>
<a href="#">DSG::BLIT</a>	<a href="#">DSG::BLIT</a> - Namespace Containing <a href="#">BLIT</a> Based Oscillators . . . . .	<a href="#">23</a>
<a href="#">DSG::DPW</a>	<a href="#">DSG::DPW</a> - Generators using the <a href="#">DPW</a> method . . . . .	<a href="#">24</a>
<a href="#">DSG::EPTR</a>	<a href="#">DSG::EPTR</a> - Generators Based On The Efficient Polynomial Transfer Region Algorithm . . . . .	<a href="#">26</a>
<a href="#">DSG::Filter</a>	<a href="#">DSG::Filter</a> - Filters . . . . .	<a href="#">26</a>
<a href="#">DSG::Fourier</a>	<a href="#">DSG::Fourier</a> - Namespace Containing <a href="#">Fourier</a> Series Based Oscillators . . . . .	<a href="#">27</a>
<a href="#">DSG::MIDI</a>	<a href="#">DSG::MIDI</a> - Namespace enclosing <a href="#">MIDI</a> processing tools . . . . .	<a href="#">27</a>
<a href="#">DSG::Noise</a>	<a href="#">DSG::Noise</a> - <a href="#">Noise</a> Generators . . . . .	<a href="#">28</a>
<a href="#">DSG::Window</a>	<a href="#">DSG::Window</a> - <a href="#">Window</a> functions and utilities . . . . .	<a href="#">29</a>



## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

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

DSG::AudioSettings . . . . .	38
DSG::Buffer . . . . .	46
DSG::RingBuffer . . . . .	93
DSG::DPW::DPW_Differentiator< order > . . . . .	54
DSG::DPW::DPW_Differentiator< 1 > . . . . .	55
DSG::DPW::DPW_Differentiator< 2 > . . . . .	55
DSG::DPW::DPW_Differentiator< 3 > . . . . .	56
DSG::DPW::DPW_Differentiator< 4 > . . . . .	57
DSG::DPW::DPW_Differentiator< 5 > . . . . .	58
DSG::DPW::DPW_Differentiator< 6 > . . . . .	59
DSG::Factorial< N > . . . . .	65
DSG::Factorial< 0 > . . . . .	65
DSG::Fourier::Harmonic . . . . .	80
DSG::LUT< element, size > . . . . .	84
DSG::Phasor . . . . .	89
DSG::SignalGenerator . . . . .	97
DSG::Analog::AnalogSaw . . . . .	33
DSG::Analog::AnalogSquare . . . . .	35
DSG::Analog::AnalogTriangle . . . . .	36
DSG::BLIT::Blit . . . . .	40
DSG::BLIT::BlitSaw . . . . .	43
DSG::BLIT::BlitSquare . . . . .	45
DSG::BLIT::BlitTriangle . . . . .	46
DSG::DPW::DPWSaw< order > . . . . .	60
DSG::EPTR::EPTRSaw . . . . .	63
DSG::Fourier::FourierSaw . . . . .	68
DSG::Fourier::FourierSeriesGenerator . . . . .	70
DSG::Fourier::FourierSquare . . . . .	73
DSG::Fourier::FourierTriangle . . . . .	76
DSG::GenericGenerator . . . . .	78
DSG::SignalProcess . . . . .	99
DSG::Delay< maxLength > . . . . .	51
DSG::Filter::FilterBase . . . . .	66
DSG::Filter::DCBlocker . . . . .	49
DSG::Filter::LeakyIntegrator . . . . .	82
DSG::NoiseGenerator . . . . .	87
DSG::SignalGenerator . . . . .	97



## Chapter 6

# Class Index

### 6.1 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 . . . . .	33
<a href="#">DSG::Analog::AnalogSquare</a>	
<a href="#">DSG::Analog::AnalogSquare</a> - Analog Syle Square Wave Generator . . . . .	35
<a href="#">DSG::Analog::AnalogTriangle</a>	
<a href="#">DSG::Analog::AnalogTriangle</a> - Analog Syle Triangle Wave Generator . . . . .	36
<a href="#">DSG::AudioSettings</a>	
<a href="#">DSG::AudioSettings</a> - Global Storage For Audio Settings Such As Sample Rate . . . . .	38
<a href="#">DSG::BLIT::Blit</a>	
<a href="#">DSG::BLIT::Blit</a> - Band-Limited Impulse Train Generator . . . . .	40
<a href="#">DSG::BLIT::BlitSaw</a>	
<a href="#">DSG::BLIT::BlitSaw</a> - Saw Wave Generator Based on <a href="#">BLIT</a> Algorithm . . . . .	43
<a href="#">DSG::BLIT::BlitSquare</a> . . . . .	45
<a href="#">DSG::BLIT::BlitTriangle</a> . . . . .	46
<a href="#">DSG::Buffer</a>	
<a href="#">DSG::Buffer</a> - Base Class For <a href="#">DSG::RingBuffer</a> . Not For Direct Use . . . . .	46
<a href="#">DSG::Filter::DCBlocker</a>	
<a href="#">DSG::Filter::DCBlocker</a> - DC blocking filter . . . . .	49
<a href="#">DSG::Delay&lt; maxLength &gt;</a>	
<a href="#">DSG::Delay</a> - General purpose delay line . . . . .	51
<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 . . .	54
<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 . . . . .	55
<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 . . . . .	55
<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 . . . . .	56
<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 . . . . .	57
<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 . . . . .	58

DSG::DPW::DPW_Differentiator< 6 >	
DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order <a href="#">DPW</a> Algorithm	59
DSG::DPW::DPWSaw< order >	
DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order <a href="#">DPW</a> algorithm	60
DSG::EPTR::EPTRSaw	
DSG::EPTR::EPTRSaw - Sawtooth Wave Generator Using The Efficient Polynomial Transfer Region Algorithm	63
DSG::Factorial< N >	
DSG::Factorial - Compute integer factorial	65
DSG::Factorial< 0 >	
DSG::Factorial - Compute integer factorial	65
DSG::Filter::FilterBase	
DSG::Filter::FilterBase - <a href="#">Filter</a> Base Class, implements interface for cutoff frequency	66
DSG::Fourier::FourierSaw	
DSG::Fourier::FourierSaw - <a href="#">Fourier</a> Series Sawtooth Wave Generator	68
DSG::Fourier::FourierSeriesGenerator	
DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified <a href="#">Fourier</a> Series	70
DSG::Fourier::FourierSquare	
DSG::Fourier::FourierSquare - <a href="#">Fourier</a> Series Square Wave Generator	73
DSG::Fourier::FourierTriangle	
DSG::Fourier::FourierTriangle - <a href="#">Fourier</a> Series Triangle Wave Generator	76
DSG::GenericGenerator	
DSG::GenericGenerator - Generator designed to use a stateless generator function such as <a href="#">DSG::Sin()</a>	78
DSG::Fourier::Harmonic	
DSG::Fourier::Harmonic - Represents a single harmonic in a <a href="#">Fourier</a> Series	80
DSG::Filter::LeakyIntegrator	
DSG::Filter::LeakyIntegrator - Leaky integrator	82
DSG::LUT< element, size >	
DSG::LUT - Look Up Table	84
DSG::NoiseGenerator	
DSG::NoiseGenerator - Generator that uses noise functions such as <a href="#">DSG::White()</a> to generate signal	87
DSG::Phasor	
DSG::Phasor - Linear Phase Generator	89
DSG::RingBuffer	
DSG::RingBuffer - Circular <a href="#">Buffer</a> of Audio	93
DSG::SignalGenerator	
DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation	97
DSG::SignalProcess	
DSG::SignalProcess - Defines Base Interface For Audio Processing	99



## Chapter 7

# File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

AnalogSaw.cpp	103
AnalogSaw.h	103
AnalogSquare.cpp	105
AnalogSquare.h	105
AnalogTriangle.cpp	106
AnalogTriangle.h	107
AudioSettings.cpp	108
AudioSettings.h	109
Blackman.h	111
BLIT.cpp	112
BLIT.h	112
BLITSaw.cpp	114
BLITSaw.h	114
BLITSquare.cpp	116
BLITSquare.h	116
BLITTriangle.cpp	117
BLITTriangle.h	118
Bounds.h	119
Buffer.cpp	120
Buffer.h	120
BufferConversion.h	121
DCBlocker.cpp	122
DCBlocker.h	123
Delay.h	124
Denormal.h	126
DPW.h	127
DPWSaw.h	130
Driver.cpp	132
Driver.h	135
DSF.h	136
DSG.h	138
DSGMath.h	140
DSGTypes.h	141
EPTRSaw.cpp	142
EPTRSaw.h	142
Filter.cpp	144
Filter.h	144
FourierSaw.cpp	145

FourierSaw.h	146
FourierSeries.cpp	147
FourierSeries.h	148
FourierSquare.cpp	150
FourierSquare.h	150
FourierTriangle.cpp	152
FourierTriangle.h	152
Gaussian.h	154
GenericGenerator.cpp	155
GenericGenerator.h	155
Interpolate.h	157
Leaky.cpp	158
Leaky.h	159
LUT.h	160
MTOF.cpp	162
MTOF.h	162
Noise.h	163
NoiseGenerator.cpp	164
NoiseGenerator.h	164
Phasor.cpp	166
Phasor.h	166
PI.h	167
Pink.h	168
Random.h	169
RingBuffer.cpp	171
RingBuffer.h	171
SignalGenerator.cpp	173
SignalGenerator.h	174
SignalProcess.cpp	175
SignalProcess.h	176
Sinc.h	177
Sine.h	178
Sleep.h	179
White.h	180
Window.h	181

## Chapter 8

# Namespace Documentation

### 8.1 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.
- **MIDI**  
*DSG::MIDI* - Namespace enclosing **MIDI** processing tools.
- **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 [Phasor](#)
  - [\*DSG::Phasor\*](#) - Linear Phase Generator.
- 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](#)
- typedef float [DSGPhase](#)
- typedef float [DSGSample](#)

## 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.
- bool [AddSampleRateListener](#) ([DSG::SignalProcess](#) \*listner)
  - [\*DSG::AddSampleRateListener\(\)\*](#) - Allows Generators to be notified if the sample rate changes.
- void [VerifySampleRateSet](#) ()
  - [\*DSG::VerifySampleRateSet\(\)\*](#) - Allows a Generator to ask if a valid sample rate has been set.
- 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.*

### 8.1.1 Detailed Description

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

#### Copyright

Lesser GNU Public License

**Todo** Increase documentation level. Add documentation for every variable, parameter...  
Implement Blep Based Algorithms

### 8.1.2 Typedef Documentation

#### 8.1.2.1 typedef float [DSG::DSGFrequency](#)

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

### 8.1.2.2 typedef float DSG::DSGPhase

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

### 8.1.2.3 typedef float DSG::DSGSample

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

## 8.1.3 Function Documentation

### 8.1.3.1 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     }
```

### 8.1.3.2 bool DSG::AddSampleRateListener ( DSG::SignalProcess \* *listner* ) [inline]

[DSG::AddSampleRateListener\(\)](#) - Allows Generators to be notified if the sample rate changes.

Definition at line 61 of file [AudioSettings.h](#).

```
00061         {
00062         return AudioSettings::AddSampleRateListener(listner);
00063     }
```

### 8.1.3.3 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     }
```

### 8.1.3.4 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     }
```

**8.1.3.5** `double DSG::Cos ( double const & x ) [inline]`[DSG::Cos\(\)](#) - General Purpose Cos Function, double precision.**Todo** Implement Taylor Series implementation of Cos FunctionDefinition at line 78 of file [Sine.h](#).

```

00078         {
00079             return static_cast<double>(Cos<Sine_Default>(x)); //wrap default implementation as non template
00080         }

```

**8.1.3.6** `float DSG::Cos ( float const & x ) [inline]`[DSG::Cos\(\)](#) - General Purpose Cos Function, single precision.Definition at line 83 of file [Sine.h](#).

```

00083         {
00084             return static_cast<float>(Cos<Sine_Default>(x));
00085         }

```

**8.1.3.7** `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     }

```

**8.1.3.8** `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     }

```

**8.1.3.9** `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         }

```

**8.1.3.10** `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     }
```

**8.1.3.11** `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     }
```

**8.1.3.12** `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
denormal
00033     }
```

**8.1.3.13** `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     }
```



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

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

```
00043
00044     double _s = DSG::SampleRate() * 20000.0/DSG::SampleRate(); {
00045     _s/=frequency;
00046     return _s;
00047 }
```

**8.1.3.15** 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 57 of file [AudioSettings.h](#).

```
00057
00058     return DSG::AudioSettings::Nyquist();
00059 }
```

**8.1.3.16** 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 }
```

**8.1.3.17** 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 }
```

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

[DSG::SampleRate](#) - Get Global Sample Rate.

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

```
00049
00050     return DSG::AudioSettings::SampleRate();
00051 }
```

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

[DSG::SampleRate](#) - Set Global Sample Rate.

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

```
00053                                     {
00054     return DSG::AudioSettings::SampleRate(value);
00055 }
```

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

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

**Todo** Implement Taylor Series implementation of Sin Function

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

```
00067                                     {
00068     return static_cast<double>(Sin<Sine_Default>(x)); //wrap default implementation as non template
00069 }
```

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

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

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

```
00072                                     {
00073     return static_cast<float>(Sin<Sine_Default>(x));
00074 }
```

### 8.1.3.22 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 }
```

### 8.1.3.23 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 }
```

**8.1.3.24** `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     }
```

**8.1.3.25** `void DSG::VerifySampleRateSet ( ) [inline]`

[DSG::VerifySampleRateSet\(\)](#) - Allows a Generator to ask if a valid sample rate has been set.

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

```
00065         {
00066     if (!DSG::AudioSettings::IsSampleRateSet ()) {
00067         SampleRate (SampleRateDefault);
00068     }
00069 }
```

## 8.2 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.*

### 8.2.1 Detailed Description

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

## 8.3 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](#)

### 8.3.1 Detailed Description

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

## 8.4 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.

### 8.4.1 Detailed Description

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

### 8.4.2 Function Documentation

**8.4.2.1** `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     }
```

**8.4.2.2** `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     }
```

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

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

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

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

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

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

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

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

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

[DSG::DPW::DPW\\_Polynomial](#) - 4th order Polynomial 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     }
```

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

[DSG::DPW::DPW\\_Polynomial](#) - 5th order Polynomial 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     }
```

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

[DSG::DPW::DPW\\_Polynomial](#) - 6th order Polynomial 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 *
00071         DPW_Polynomial<2>(value);
00071     }
```

## 8.5 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.

### 8.5.1 Detailed Description

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

## 8.6 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.

### 8.6.1 Detailed Description

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

## 8.7 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.*

### 8.7.1 Detailed Description

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

## 8.8 DSG::MIDI Namespace Reference

DSG::MIDI - Namespace enclosing [MIDI](#) processing tools.

### Functions

- double [MTOF](#) (unsigned char const &MIDI\_Number)  
*DSG::MIDI:MTOF - MIDI to Frequency Conversion.*
- unsigned char [FTOM](#) (double const &Frequency)  
*DSG::MIDI:FTOM - Frequency to MIDI Conversion.*

### 8.8.1 Detailed Description

DSG::MIDI - Namespace enclosing [MIDI](#) processing tools.

### 8.8.2 Function Documentation

#### 8.8.2.1 unsigned char DSG::MIDI::FTOM ( double const &Frequency )

DSG::MIDI:FTOM - Frequency to [MIDI](#) Conversion.

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

```
00028
00029     return ( (log2 ( (Frequency/440.0) ) ) *12.0) +69.0;
00030 }
```

### 8.8.2.2 double DSG::MIDI::MTOF ( unsigned char const & *MIDI\_Number* )

[DSG::MIDI::MTOF](#) - [MIDI](#) to Frequency Conversion.

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

```
00025                                     {
00026     return 440.0 *pow(2.0, (MIDI_Number-69.0)/12.0);
00027 }
```

## 8.9 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.*

### 8.9.1 Detailed Description

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

### 8.9.2 Function Documentation

#### 8.9.2.1 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 }
```



**8.9.2.2** `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         }

```

**8.9.2.3** `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         }

```

**8.9.2.4** `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         }

```

## 8.10 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 lutsizes>`  
`void ApplyWindow (DSG::LUT< decimal, lutsizes > &lut, decimal(&>windowFunction)(decimal const &), decimal range=1.0)`

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

- `template<typename decimal, unsigned long lutsize>`  
`void ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&>windowFunction)(decimal), decimal range=1.0)`

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

### 8.10.1 Detailed Description

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

### 8.10.2 Function Documentation

- 8.10.2.1 `template<typename decimal, unsigned long lutsize> void DSG::Window::ApplyWindow ( DSG::LUT< decimal, lutsize > &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         }
```

- 8.10.2.2 `template<typename decimal, unsigned long lutsize> void DSG::Window::ApplyWindow ( DSG::LUT< decimal, lutsize > &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         }
```

- 8.10.2.3 `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 9

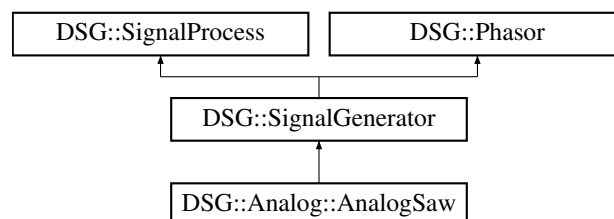
# Class Documentation

### 9.1 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

##### 9.1.1 Detailed Description

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

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

## 9.1.2 Constructor & Destructor Documentation

### 9.1.2.1 DSG::Analog::AnalogSaw::AnalogSaw ( )

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

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

### 9.1.2.2 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) {}
```

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

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

```
00027 {}
```

## 9.1.3 Member Function Documentation

### 9.1.3.1 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     }
```

### 9.1.3.2 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         }
```

### 9.1.4 Member Data Documentation

#### 9.1.4.1 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:

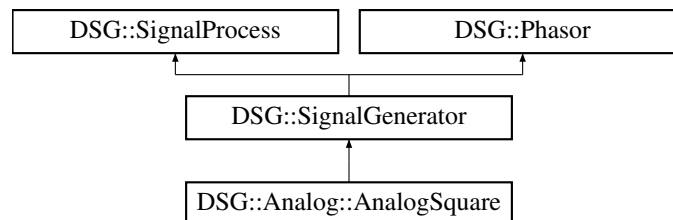
- [AnalogSaw.h](#)
- [AnalogSaw.cpp](#)

## 9.2 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

#### 9.2.1 Detailed Description

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

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

#### 9.2.2 Constructor & Destructor Documentation

##### 9.2.2.1 DSG::Analog::AnalogSquare::AnalogSquare ( )

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

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

### 9.2.2.2 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){}
```

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

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

```
00027 {}
```

## 9.2.3 Member Function Documentation

### 9.2.3.1 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     }
```

### 9.2.3.2 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](#)

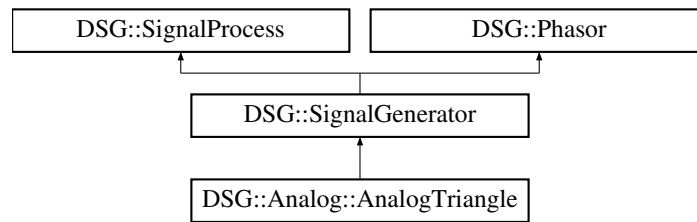
## 9.3 DSG::Analog::AnalogTriangle Class Reference

[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

### 9.3.1 Detailed Description

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

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

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 DSG::Analog::AnalogTriangle::AnalogTriangle ( )

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

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

#### 9.3.2.2 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) {}
```

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

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

```
00027 {}
```

### 9.3.3 Member Function Documentation

#### 9.3.3.1 `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     }

```

#### 9.3.3.2 `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         }

```

### 9.3.4 Member Data Documentation

#### 9.3.4.1 `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](#)

## 9.4 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 bool [AddSampleRateListener](#) ([SignalProcess](#) \*listener)
- static bool const & [IsSampleRateSet](#) ()

## Static Protected Attributes

- static [DSG::DSGFrequency \\_sampleRate](#)
- static [DSG::DSGFrequency \\_nyquist](#)
- static std::vector  
     < [DSG::SignalProcess](#) \* > [\\_listeners](#)
- static bool [\\_set](#) =false

### 9.4.1 Detailed Description

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

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

### 9.4.2 Member Function Documentation

#### 9.4.2.1 bool DSG::AudioSettings::AddSampleRateListener ( [DSG::SignalProcess](#) \* *listener* ) [static]

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

```
00047                                     {
00048     \_listeners.push_back(listener);
00049     return true;
00050 }
```

#### 9.4.2.2 bool const & DSG::AudioSettings::IsSampleRateSet ( ) [static]

Definition at line 51 of file [AudioSettings.cpp](#).

```
00051                                     {
00052     return \_set;
00053 }
```

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

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

```
00044                                     {
00045     return \_nyquist;
00046 }
```

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

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

```
00030                                     {
00031     return \_sampleRate;
00032 }
```

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

Definition at line 33 of file [AudioSettings.cpp](#).

```

00033                                     {
00034     if (!_set) {
00035         _set=true;
00036     }
00037     _sampleRate = value;
00038     _nyquist = _sampleRate*0.5;
00039     for (auto i:_listeners) {
00040         i->SampleRateChanged(_sampleRate);
00041     }
00042     return _sampleRate;
00043 }
```

### 9.4.3 Member Data Documentation

#### 9.4.3.1 std::vector< DSG::SignalProcess \* > DSG::AudioSettings::\_listeners [static],[protected]

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

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

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

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

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

#### 9.4.3.4 bool DSG::AudioSettings::\_set =false [static],[protected]

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

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

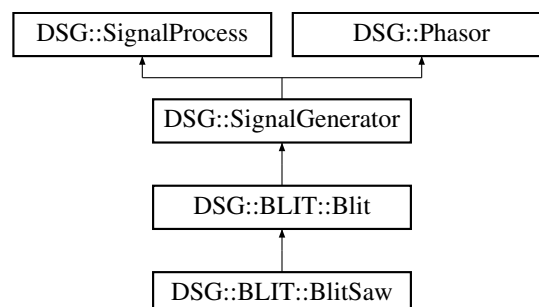
- [AudioSettings.h](#)
- [AudioSettings.cpp](#)

## 9.5 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

### 9.5.1 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](#).

### 9.5.2 Constructor & Destructor Documentation

#### 9.5.2.1 DSG::BLIT::Blit::Blit ( )

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

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

#### 9.5.2.2 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 }
```

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

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

```
00031 {}
```

### 9.5.3 Member Function Documentation

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

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

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     }
```

#### 9.5.3.2 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
00057         algorithmically
00058         denominator = m_ * sin(_phasor);
00059         if (DSG::IsDenormal(denominator)) {
00060             signal = a_;
00061         }else{
00062             value = sin(PI*_phasor * m_);
00063             value/=denominator;
00064             value*=a_;
00065             signal = value;
00066         }
00067         step();
00068         return true;
00069     }
```

#### 9.5.3.3 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         }
```

### 9.5.4 Member Data Documentation

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

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

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

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

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

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

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

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

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

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

## 9.5.4.6 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:

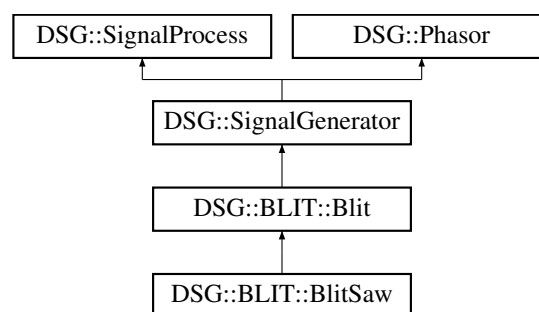
- [BLIT.h](#)
- [BLIT.cpp](#)

## 9.6 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

### 9.6.1 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](#).

### 9.6.2 Constructor & Destructor Documentation

#### 9.6.2.1 DSG::BLIT::BlitSaw::BlitSaw ( )

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

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

#### 9.6.2.2 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 }
```

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

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

```
00031 {}
```

### 9.6.3 Member Function Documentation

#### 9.6.3.1 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 }
```



**9.6.3.2** `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     }

```

**9.6.3.3** `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         }

```

## 9.6.4 Member Data Documentation

**9.6.4.1** `DSG::DSGSample DSG::BLIT::BlitSaw::C2_` `[protected]`

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

**9.6.4.2** `DSG::DSGSample DSG::BLIT::BlitSaw::Register_` `[protected]`

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

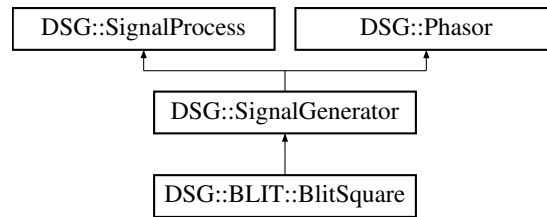
The documentation for this class was generated from the following files:

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

## 9.7 DSG::BLIT::BlitSquare Class Reference

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

Inheritance diagram for `DSG::BLIT::BlitSquare`:



## Additional Inherited Members

### 9.7.1 Detailed Description

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

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

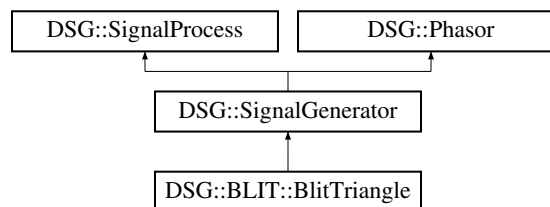
The documentation for this class was generated from the following file:

- [BLITSquare.h](#)

## 9.8 DSG::BLIT::BlitTriangle Class Reference

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

Inheritance diagram for DSG::BLIT::BlitTriangle:



## Additional Inherited Members

### 9.8.1 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:

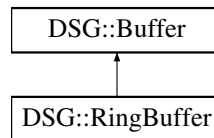
- [BLITTriangle.h](#)

## 9.9 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](#)

### 9.9.1 Detailed Description

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

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

### 9.9.2 Constructor & Destructor Documentation

#### 9.9.2.1 DSG::Buffer::Buffer ( )

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

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

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

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

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

#### 9.9.2.3 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 }
```

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

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

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

### 9.9.3 Member Function Documentation

#### 9.9.3.1 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 }
```

#### 9.9.3.2 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 }
```

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

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

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

### 9.9.4 Member Data Documentation

#### 9.9.4.1 DSG::DSGSample\* DSG::Buffer::\_buffer [protected]

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

#### 9.9.4.2 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:

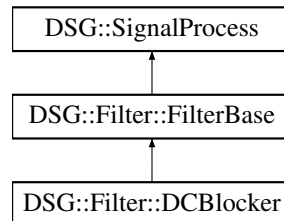
- [Buffer.h](#)
- [Buffer.cpp](#)

## 9.10 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](#)

### 9.10.1 Detailed Description

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

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

### 9.10.2 Constructor & Destructor Documentation

#### 9.10.2.1 DSG::Filter::DCBlocker::DCBlocker ( )

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

```
00025 :DSG::Filter::FilterBase(), _a(0.995), xm1(0), ym1(0),
      x(0), _temp(0) {}
```

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

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

```
00026 {}
```

### 9.10.3 Member Function Documentation

#### 9.10.3.1 `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     }
```

#### 9.10.3.2 `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     }
```

### 9.10.4 Member Data Documentation

#### 9.10.4.1 `DSG::DSGSample DSG::Filter::DCBlocker::_a` `[protected]`

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

#### 9.10.4.2 `DSG::DSGSample DSG::Filter::DCBlocker::_temp` `[protected]`

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

#### 9.10.4.3 `unsigned long DSG::Filter::DCBlocker::count` `[protected]`

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

#### 9.10.4.4 `DSG::DSGSample DSG::Filter::DCBlocker::x` `[protected]`

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

#### 9.10.4.5 `DSG::DSGSample DSG::Filter::DCBlocker::xml` `[protected]`

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

## 9.10.4.6 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:

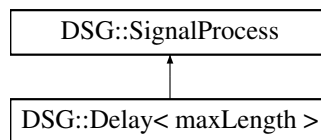
- [DCBlocker.h](#)
- [DCBlocker.cpp](#)

## 9.11 DSG::Delay&lt; maxLength &gt; 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](#)

## 9.11.1 Detailed Description

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

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

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

## 9.11.2 Constructor & Destructor Documentation

### 9.11.2.1 `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     }
```

### 9.11.2.2 `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     }
```

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

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

```
00050 {}
```

## 9.11.3 Member Function Documentation

### 9.11.3.1 `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     }
```

### 9.11.3.2 `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     }
```



**9.11.3.3** `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     }
```

**9.11.3.4** `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     }
```

**9.11.3.5** `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     }
```

## 9.11.4 Member Data Documentation

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

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

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

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

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

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

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

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

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

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

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

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

9.11.4.7 `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](#)

## 9.12 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](#) ()

#### 9.12.1 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 4-6

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

#### 9.12.2 Constructor & Destructor Documentation

9.12.2.1 `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](#)

## 9.13 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)

#### 9.13.1 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](#).

#### 9.13.2 Member Function Documentation

9.13.2.1 **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](#)

## 9.14 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](#)

### 9.14.1 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](#).

### 9.14.2 Member Function Documentation

9.14.2.1 `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     }
```

### 9.14.3 Member Data Documentation

9.14.3.1 `DSG::DSGSample DSG::DPW::DPW_Differentiator< 2 >::_delay` `[protected]`

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

9.14.3.2 `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](#)

## 9.15 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]

### 9.15.1 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](#).

### 9.15.2 Member Function Documentation

**9.15.2.1** `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     }
```

### 9.15.3 Member Data Documentation

**9.15.3.1** `DSG::DSGSample DSG::DPW::DPW_Differentiator< 3 >::_delay[2]` `[protected]`

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

**9.15.3.2** `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](#)

## 9.16 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]

### 9.16.1 Detailed Description

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

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

**Bug** Causes major clipping

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

## 9.16.2 Member Function Documentation

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

Definition at line 128 of file [DPW.h](#).

```

00128                                     {
00129         output = (signal - _delay[0]);
00130         output -= (_delay[0] - _delay[1]);
00131         output -= (_delay[1] - _delay[2]);
00132         output /= 144*DSG::Pow<3>(dt);
00133         _delay[2]=_delay[1];
00134         _delay[1]=_delay[0];
00135         _delay[0]=signal;
00136         return output;
00137     }
```

## 9.16.3 Member Data Documentation

### 9.16.3.1 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 4 >::\_delay[3] [protected]

Definition at line 140 of file [DPW.h](#).

### 9.16.3.2 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 4 >::output [protected]

Definition at line 139 of file [DPW.h](#).

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

- [DPW.h](#)

## 9.17 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]

### 9.17.1 Detailed Description

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

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

**Bug** Causes major clipping

Definition at line 146 of file [DPW.h](#).

## 9.17.2 Member Function Documentation

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

Definition at line 148 of file [DPW.h](#).

```

00148
00149         output = (signal - _delay[0]);
00150         output -= (_delay[0] - _delay[1]);
00151         output -= (_delay[1] - _delay[2]);
00152         output -= (_delay[2] - _delay[3]);
00153         output /= 960*DSG::Pow<4>(dt);
00154         _delay[3]=_delay[2];
00155         _delay[2]=_delay[1];
00156         _delay[1]=_delay[0];
00157         _delay[0]=signal;
00158         return output;
00159     }

```

## 9.17.3 Member Data Documentation

### 9.17.3.1 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 5 >::\_delay[4] [protected]

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

### 9.17.3.2 DSG::DSGSample DSG::DPW::DPW\_Differentiator< 5 >::output [protected]

Definition at line 161 of file [DPW.h](#).

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

- [DPW.h](#)

## 9.18 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]

### 9.18.1 Detailed Description

template<>class DSG::DPW::DPW\_Differentiator< 6 >

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

**Bug** Causes major clipping

Definition at line 168 of file [DPW.h](#).

### 9.18.2 Member Function Documentation

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

Definition at line 170 of file [DPW.h](#).

```

00170                                     {
00171         output = (signal - _delay[0]);
00172         output -= (_delay[0] - _delay[1]);
00173         output -= (_delay[1] - _delay[2]);
00174         output -= (_delay[2] - _delay[3]);
00175         output -= (_delay[3] - _delay[4]);
00176         output /= 7200*DSG::Pow<5>(dt);
00177         _delay[4]=_delay[3];
00178         _delay[3]=_delay[2];
00179         _delay[2]=_delay[1];
00180         _delay[1]=_delay[0];
00181         _delay[0]=signal;
00182         return output;
00183     }
```

### 9.18.3 Member Data Documentation

9.18.3.1 **DSG::DSGSample DSG::DPW::DPW\_Differentiator< 6 >::\_delay[5]** [protected]

Definition at line 186 of file [DPW.h](#).

9.18.3.2 **DSG::DSGSample DSG::DPW::DPW\_Differentiator< 6 >::output** [protected]

Definition at line 185 of file [DPW.h](#).

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

- [DPW.h](#)

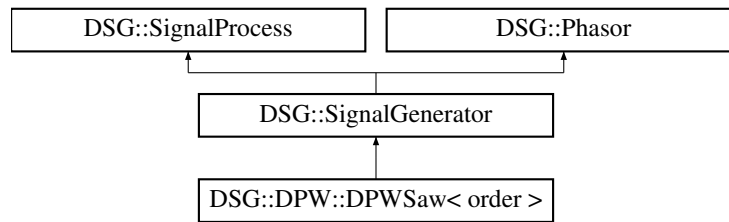
## 9.19 DSG::DPW::DPWSaw< order > Class Template Reference

[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

### 9.19.1 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](#).

### 9.19.2 Constructor & Destructor Documentation

9.19.2.1 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     }
```

9.19.2.2 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>();}
```

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

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

```
00040 {}
```

## 9.19.3 Member Function Documentation

**9.19.3.1** `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     }
```

**9.19.3.2** `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         }
```

## 9.19.4 Member Data Documentation

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

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

**9.19.4.2** `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:

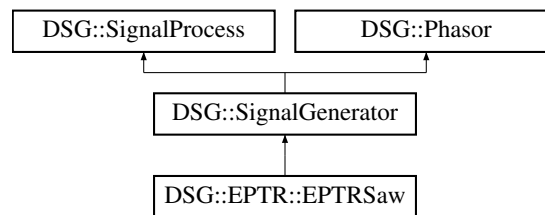
- [DPWSaw.h](#)

## 9.20 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::DSG Frequency const &frequency, DSG::DSG Phase const &offset)
- virtual [~EPTR Saw](#) ()
- virtual bool [Perform](#) (DSG::DSG Sample &signal)
- virtual bool [Perform](#) (DSG::Ring Buffer &signal)

### Protected Attributes

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

### Additional Inherited Members

#### 9.20.1 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

**Bug** Algorithm is not performing in a band limited manor

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

#### 9.20.2 Constructor & Destructor Documentation

##### 9.20.2.1 DSG::EPTR::EPTR Saw::EPTR Saw ( )

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

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

### 9.20.2.2 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){}
```

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

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

```
00027 {}
```

## 9.20.3 Member Function Documentation

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

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

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

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

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

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

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

```
00068                                     {
00069         signal.Flush();
00070         while (!signal.Full()) {
00071             if (Perform(_storage)) {
00072                 if(signal.Write(_storage)){
00073                     }else return false;
00074                 }else return false;
00075             }return true;
00076         }
```

## 9.20.4 Member Data Documentation

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

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

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

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

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

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

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

### Public Types

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

#### 9.21.1 Detailed Description

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

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

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

#### 9.21.2 Member Enumeration Documentation

9.21.2.1 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](#)

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

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

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

### Public Types

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

#### 9.22.1 Detailed Description

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

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

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

## 9.22.2 Member Enumeration Documentation

### 9.22.2.1 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:

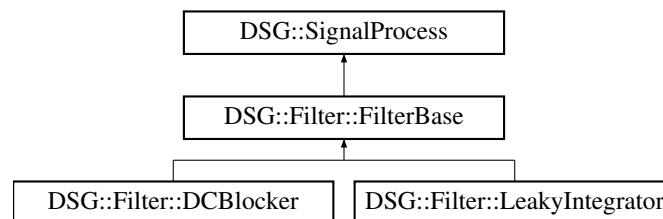
- [DSGMath.h](#)

## 9.23 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](#)

### 9.23.1 Detailed Description

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

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

## 9.23.2 Constructor & Destructor Documentation

### 9.23.2.1 DSG::Filter::FilterBase::FilterBase ( )

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

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

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

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

```
00026 {}
```

## 9.23.3 Member Function Documentation

### 9.23.3.1 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     }
```

### 9.23.3.2 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     }
```

### 9.23.3.3 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     }
```

### 9.23.4 Member Data Documentation

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

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

#### 9.23.4.2 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:

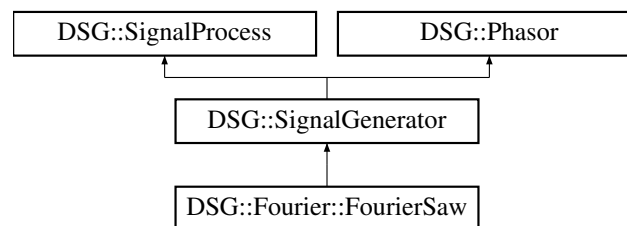
- [Filter.h](#)
- [Filter.cpp](#)

## 9.24 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

#### 9.24.1 Detailed Description

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

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



## 9.24.2 Constructor & Destructor Documentation

### 9.24.2.1 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){}
```

### 9.24.2.2 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 }
```

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

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

```
00029 {}
```

## 9.24.3 Member Function Documentation

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

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

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 }
```

### 9.24.3.2 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 }
```

### 9.24.3.3 `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         }

```

## 9.24.4 Member Data Documentation

### 9.24.4.1 `const double DSG::Fourier::FourierSaw::_a` `[protected]`

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

### 9.24.4.2 `unsigned long DSG::Fourier::FourierSaw::_h` `[protected]`

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

### 9.24.4.3 `int DSG::Fourier::FourierSaw::i` `[protected]`

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

### 9.24.4.4 `double DSG::Fourier::FourierSaw::phs` `[protected]`

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

### 9.24.4.5 `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:

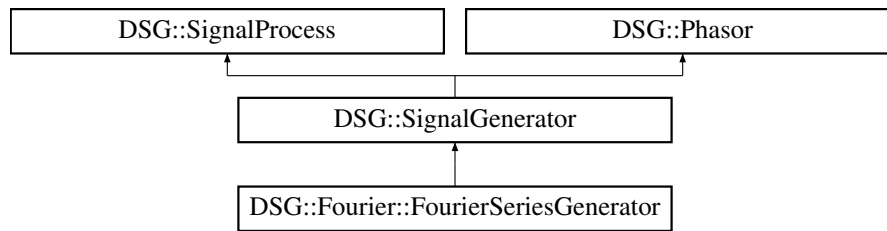
- [FourierSaw.h](#)
- [FourierSaw.cpp](#)

## 9.25 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

### 9.25.1 Detailed Description

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

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

### 9.25.2 Member Typedef Documentation

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

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

### 9.25.3 Constructor & Destructor Documentation

#### 9.25.3.1 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator ( )

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

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

### 9.25.3.2 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){}
```

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

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

```
00047 {}
```

## 9.25.4 Member Function Documentation

### 9.25.4.1 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 }
```

### 9.25.4.2 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     }
```

### 9.25.4.3 void DSG::Fourier::FourierSeriesGenerator::Series ( FourierSeries const & *series* ) [inline]

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

```
00080
00081     {
00082         _series = series;
00082     }
```

#### 9.25.4.4 DSG::Fourier::FourierSeriesGenerator::FourierSeries & DSG::Fourier::FourierSeriesGenerator::Series ( ) [inline]

Definition at line 83 of file [FourierSeries.h](#).

```
00083                                     {
00084         return _series;
00085     }
```

### 9.25.5 Member Data Documentation

#### 9.25.5.1 FourierSeries DSG::Fourier::FourierSeriesGenerator::\_series [protected]

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

#### 9.25.5.2 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:

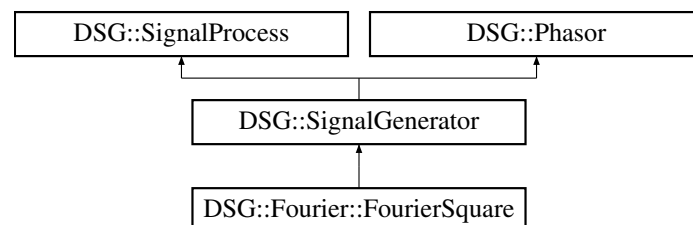
- [FourierSeries.h](#)
- [FourierSeries.cpp](#)

## 9.26 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

### 9.26.1 Detailed Description

[DSG::Fourier::FourierSquare](#) - [Fourier](#) Series Square Wave Generator.

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

### 9.26.2 Constructor & Destructor Documentation

#### 9.26.2.1 [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){}
```

#### 9.26.2.2 [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 }
```

#### 9.26.2.3 [DSG::Fourier::FourierSquare::~~FourierSquare \( \) \[virtual\]](#)

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

```
00029 {}
```

### 9.26.3 Member Function Documentation

#### 9.26.3.1 [DSG::DSGFrequency const & DSG::Fourier::FourierSquare::Frequency \( DSG::DSGFrequency const & value \) \[inline\],\[virtual\]](#)

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

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 }
```

### 9.26.3.2 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     }

```

### 9.26.3.3 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         }

```

## 9.26.4 Member Data Documentation

### 9.26.4.1 const double DSG::Fourier::FourierSquare::\_a [protected]

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

### 9.26.4.2 unsigned long DSG::Fourier::FourierSquare::\_h [protected]

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

### 9.26.4.3 int DSG::Fourier::FourierSquare::i [protected]

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

### 9.26.4.4 double DSG::Fourier::FourierSquare::phs [protected]

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

### 9.26.4.5 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:

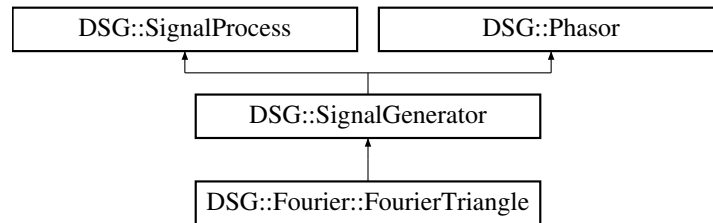
- [FourierSquare.h](#)
- [FourierSquare.cpp](#)

## 9.27 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

#### 9.27.1 Detailed Description

[DSG::Fourier::FourierTriangle](#) - [Fourier](#) Series Triangle Wave Generator.

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

#### 9.27.2 Constructor & Destructor Documentation

##### 9.27.2.1 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){}
```



### 9.27.2.2 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 }
```

### 9.27.2.3 DSG::Fourier::FourierTriangle::~~FourierTriangle ( ) [virtual]

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

```
00029 {}
```

## 9.27.3 Member Function Documentation

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

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

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 }
```

### 9.27.3.2 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 }
```

### 9.27.3.3 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         }

```

## 9.27.4 Member Data Documentation

### 9.27.4.1 `const double DSG::Fourier::FourierTriangle::_a` [protected]

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

### 9.27.4.2 `unsigned long DSG::Fourier::FourierTriangle::_h` [protected]

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

### 9.27.4.3 `int DSG::Fourier::FourierTriangle::i` [protected]

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

### 9.27.4.4 `double DSG::Fourier::FourierTriangle::phs` [protected]

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

### 9.27.4.5 `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:

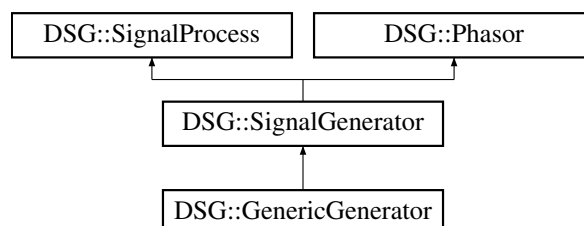
- [FourierTriangle.h](#)
- [FourierTriangle.cpp](#)

## 9.28 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

### 9.28.1 Detailed Description

[DSG::GenericGenerator](#) - Generator designed to use a stateless generator function such as [DSG::Sin\(\)](#)

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

### 9.28.2 Constructor & Destructor Documentation

#### 9.28.2.1 DSG::GenericGenerator::GenericGenerator ( )

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

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

#### 9.28.2.2 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) {}
```

#### 9.28.2.3 DSG::GenericGenerator::~~GenericGenerator ( ) [virtual]

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

```
00027 {}
```

### 9.28.3 Member Function Documentation

#### 9.28.3.1 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     }
```

### 9.28.3.2 `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         }

```

## 9.28.4 Member Data Documentation

### 9.28.4.1 `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](#)

## 9.29 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

### 9.29.1 Detailed Description

[DSG::Fourier::Harmonic](#) - Represents a single harmonic in a [Fourier](#) Series.

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

## 9.29.2 Constructor & Destructor Documentation

### 9.29.2.1 DSG::Fourier::Harmonic::Harmonic ( )

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

```
00025 :_ratio(0),_amplitude(0){}
```

### 9.29.2.2 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){}
```

### 9.29.2.3 DSG::Fourier::Harmonic::~~Harmonic ( ) [virtual]

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

```
00027                                     {
00028     _ratio=0;
00029     _amplitude=0;
00030 }
```

## 9.29.3 Member Function Documentation

### 9.29.3.1 DSG::DSGSample const & DSG::Fourier::Harmonic::Amplitude ( ) const

Definition at line 38 of file [FourierSeries.cpp](#).

```
00038                                     {
00039     return _amplitude;
00040 }
```

### 9.29.3.2 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 }
```

### 9.29.3.3 DSG::DSGSample const & DSG::Fourier::Harmonic::Ratio ( ) const

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

```
00031                                     {
00032     return _ratio;
00033 }
```

### 9.29.3.4 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 }
```

### 9.29.4 Member Data Documentation

#### 9.29.4.1 DSG::DSGSample DSG::Fourier::Harmonic::\_amplitude [protected]

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

#### 9.29.4.2 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:

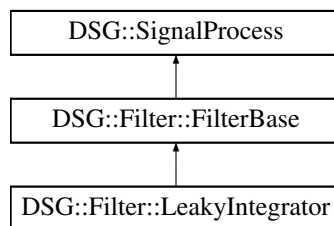
- [FourierSeries.h](#)
- [FourierSeries.cpp](#)

## 9.30 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](#)

### 9.30.1 Detailed Description

[DSG::Filter::LeakyIntegrator](#) - Leaky integrator.

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

## 9.30.2 Constructor & Destructor Documentation

### 9.30.2.1 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 }
```

### 9.30.2.2 DSG::Filter::LeakyIntegrator::LeakyIntegrator ( DSG::DSGFrequency const & cutoff )

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

```
00032                                     :
DSG::Filter::FilterBase() {
00033     x1=0;
00034     y1=0;
00035     a=0;
00036     b=0;
00037     y=0;
00038     Cutoff(cutoff);
00039 }
```

### 9.30.2.3 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 }
```

## 9.30.3 Member Function Documentation

### 9.30.3.1 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 }
```

### 9.30.3.2 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     }

```

### 9.30.3.3 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     }

```

## 9.30.4 Member Data Documentation

### 9.30.4.1 double DSG::Filter::LeakyIntegrator::a [protected]

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

### 9.30.4.2 double DSG::Filter::LeakyIntegrator::b [protected]

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

### 9.30.4.3 double DSG::Filter::LeakyIntegrator::x1 [protected]

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

### 9.30.4.4 double DSG::Filter::LeakyIntegrator::y [protected]

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

### 9.30.4.5 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](#)

## 9.31 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](#)

### 9.31.1 Detailed Description

template<typename element, unsigned long size>class DSG::LUT< element, size >

[DSG::LUT](#) - Look Up Table.

**Todo** Implement interpolation into lookup algorithm

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

### 9.31.2 Member Typedef Documentation

9.31.2.1 template<typename element, unsigned long size> typedef element(\* [DSG::LUT< element, size >::FillFunction](#))(element)

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

9.31.2.2 template<typename element, unsigned long size> typedef element(\* [DSG::LUT< element, size >::FillFunctionConstRef](#))(element const &)

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

### 9.31.3 Constructor & Destructor Documentation

9.31.3.1 template<typename element, unsigned long size> [DSG::LUT< element, size >::LUT](#) ( ) [[inline](#)]

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

```
00038 :_size(size){}
```

**9.31.3.2** `template<typename element, unsigned long size> DSG::LUT< element, size >::LUT ( FillFunction fill, double const & range = 1.0 ) [inline]`

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

```

00039                                     :_size(size){
00040         //range is the expected input range for the function
00041         //example would be 0-2pi or 0-1
00042         //would be provided a 2pi or 1
00043         //defaults to 1
00044         double step = range/(double)_size;
00045         phs = 0;
00046         for (int i=0; i<_size; ++i) {
00047             _table[i] = fill(phs);
00048             phs+=step;
00049         }
00050     }

```

**9.31.3.3** `template<typename element, unsigned long size> DSG::LUT< element, size >::LUT ( FillFunctionConstRef fill, double const & range = 1.0 ) [inline]`

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

```

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     }

```

**9.31.3.4** `template<typename element, unsigned long size> DSG::LUT< element, size >::~~LUT ( ) [inline]`

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

```

00063 {}

```

## 9.31.4 Member Function Documentation

**9.31.4.1** `template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator() ( double const & x ) [inline]`

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

```

00076                                     {
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     }

```

**9.31.4.2** `template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator[] ( unsigned long const & index ) const [inline]`

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

```

00064                                     {
00065     #ifdef DEBUG
00066         assert(index<_size);
00067     #endif
00068         return _table[index];
00069     }

```

**9.31.4.3** `template<typename element, unsigned long size> element& DSG::LUT< element, size >::operator[] ( unsigned long const & index ) [inline]`

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

```

00070                                     {
00071     #ifdef DEBUG
00072         assert(index<_size);
00073     #endif
00074         return _table[index];
00075     }

```

**9.31.4.4** `template<typename element, unsigned long size> unsigned long const& DSG::LUT< element, size >::Size ( ) const [inline]`

Definition at line 83 of file [LUT.h](#).

```

00083                                     {
00084         return _size;
00085     }

```

## 9.31.5 Member Data Documentation

**9.31.5.1** `template<typename element, unsigned long size> const unsigned long DSG::LUT< element, size >::_size [protected]`

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

**9.31.5.2** `template<typename element, unsigned long size> element DSG::LUT< element, size >::_table[size] [protected]`

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

**9.31.5.3** `template<typename element, unsigned long size> double DSG::LUT< element, size >::phs [protected]`

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

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

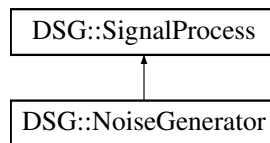
- [LUT.h](#)

## 9.32 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](#)

### 9.32.1 Detailed Description

[DSG::NoiseGenerator](#) - Generator that uses noise functions such as [DSG::White\(\)](#) to generate signal.

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

### 9.32.2 Constructor & Destructor Documentation

#### 9.32.2.1 DSG::NoiseGenerator::NoiseGenerator ( [DSGSample](#)(\*)([DSGSample](#)) *StatelessFunction* )

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

```

00025                                     :
      DSG::SignalProcess() {
00026     \_function = StatelessFunction;
00027 }
```

#### 9.32.2.2 DSG::NoiseGenerator::~~NoiseGenerator ( ) [virtual]

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

```

00028 {}
```

### 9.32.3 Member Function Documentation

#### 9.32.3.1 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 }
```

9.32.3.2 `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     }

```

## 9.32.4 Member Data Documentation

9.32.4.1 `DSGSample(* DSG::NoiseGenerator::_function)(DSGSample)` `[protected]`

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

9.32.4.2 `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:

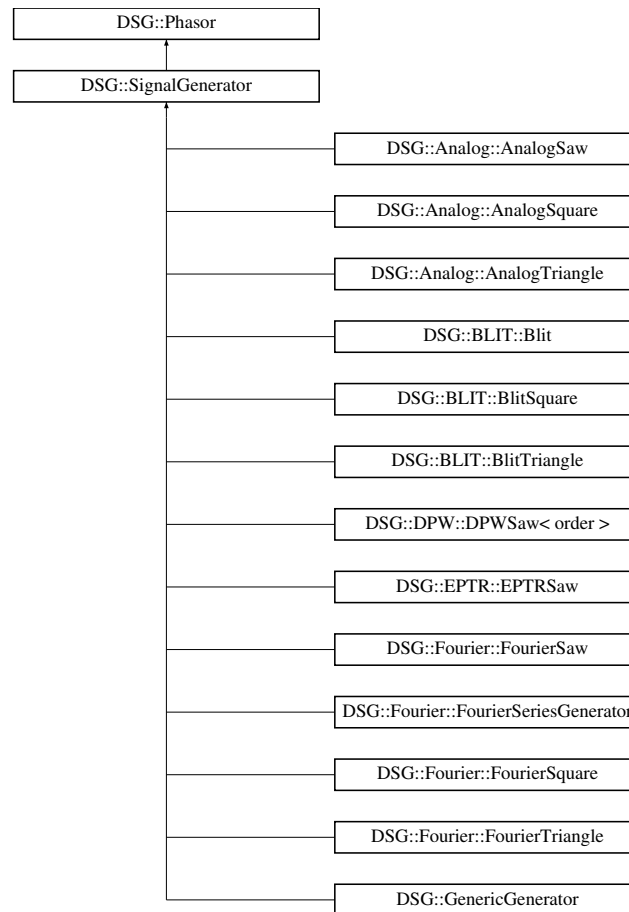
- [NoiseGenerator.h](#)
- [NoiseGenerator.cpp](#)

## 9.33 DSG::Phasor Class Reference

[DSG::Phasor](#) - Linear Phase Generator.

```
#include <Phasor.h>
```

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



## Public Member Functions

- [Phasor](#) ()
- [Phasor](#) ([DSG::DSGFrequency](#) const &frequency, [DSG::DSGPhase](#) const &offset)
- virtual [~Phasor](#) ()
- 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](#)

### 9.33.1 Detailed Description

[DSG::Phasor](#) - Linear Phase Generator.

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

### 9.33.2 Constructor & Destructor Documentation

#### 9.33.2.1 DSG::Phasor::Phasor ( )

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

```
00025 : _phasor(0), _frequency(0), _dt(0), _offset(0) {}
```

#### 9.33.2.2 DSG::Phasor::Phasor ( DSG::DSGFrequency const & frequency, DSG::DSGPhase const & offset )

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

```
00026 :
00026     _phasor(0), _frequency(frequency), _dt(0), _offset(offset) {
00027         Frequency(frequency);
00028         Phase(offset);
00029 }
```

#### 9.33.2.3 DSG::Phasor::~~Phasor ( ) [virtual]

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

```
00030 {}
```

### 9.33.3 Member Function Documentation

#### 9.33.3.1 DSG::DSGFrequency const & DSG::Phasor::Frequency ( ) [inline], [virtual]

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

```
00050 {
00051     return _frequency;
00052 }
```

#### 9.33.3.2 DSG::DSGFrequency const & DSG::Phasor::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 53 of file [Phasor.h](#).

```
00053 {
00054     _frequency = DSG::EnforceBounds<0, 20000, DSG::DSGSample>(value);
00055     _dt = _frequency/DSG::SampleRate();
00056     return _frequency;
00057 }
```

### 9.33.3.3 DSG::DSGPhase const & DSG::Phasor::Phase ( ) [inline],[virtual]

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

```
00058                                     {
00059     return _offset;
00060 }
```

### 9.33.3.4 DSG::DSGPhase const & DSG::Phasor::Phase ( DSG::DSGPhase const & value ) [inline],[virtual]

Definition at line 61 of file [Phasor.h](#).

```
00061                                     {
00062     _offset-=value;
00063     _phasor-=_offset;
00064     _offset=value;
00065     return _offset;
00066 }
```

### 9.33.3.5 void DSG::Phasor::step ( ) [inline],[protected]

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

```
00067     {
00068     _phasor+=_dt;
00069     _phasor>1.0 ? --_phasor:0;
00070 }
```

### 9.33.3.6 void DSG::Phasor::sync ( ) [inline],[protected]

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

```
00071     {
00072     _phasor=_offset;
00073 }
```

## 9.33.4 Member Data Documentation

### 9.33.4.1 DSG::DSGPhase DSG::Phasor::\_dt [protected]

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

### 9.33.4.2 DSG::DSGFrequency DSG::Phasor::\_frequency [protected]

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

### 9.33.4.3 DSG::DSGPhase DSG::Phasor::\_offset [protected]

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



## 9.33.4.4 DSG::DSGPhase DSG::Phasor::\_phasor [protected]

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

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

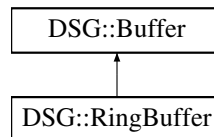
- [Phasor.h](#)
- [Phasor.cpp](#)

## 9.34 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)

### 9.34.1 Detailed Description

DSG::RingBuffer - Circular Buffer of Audio.

Definition at line 35 of file RingBuffer.h.

### 9.34.2 Constructor & Destructor Documentation

#### 9.34.2.1 DSG::RingBuffer::RingBuffer ( )

Definition at line 25 of file RingBuffer.cpp.

```
00025 :Buffer(0),_read(0),_write(0),_count(0),MASK(0){}
```

#### 9.34.2.2 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 }
```

#### 9.34.2.3 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 }
```

#### 9.34.2.4 DSG::RingBuffer::~RingBuffer ( ) [virtual]

Definition at line 43 of file RingBuffer.cpp.

```
00043 {Flush();}
```

### 9.34.3 Member Function Documentation

#### 9.34.3.1 size\_t const & DSG::RingBuffer::Count ( ) const [inline]

Definition at line 106 of file RingBuffer.h.

```
00106                                     {
00107     return _count;
00108 }
```

**9.34.3.2** `bool DSG::RingBuffer::Empty ( ) const` `[inline]`

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

```
00080                                     {
00081         return _count==0;
00082     }
```

**9.34.3.3** `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     }
```

**9.34.3.4** `bool DSG::RingBuffer::Full ( ) const` `[inline]`

Definition at line 77 of file [RingBuffer.h](#).

```
00077                                     {
00078         return _count==this->_size;
00079     }
```

**9.34.3.5** `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     }
```

**9.34.3.6** `size_t DSG::RingBuffer::next ( size_t current )` `[inline],[protected]`

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

```
00110 {return (current+1) & MASK;}
```

**9.34.3.7** `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     }
```

#### 9.34.3.8 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 }
```

#### 9.34.3.9 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 }
```

### 9.34.4 Friends And Related Function Documentation

#### 9.34.4.1 bool operator<< ( DSG::DSGSample & *signal*, DSG::RingBuffer & *buffer* ) [friend]

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

```

00060                                     {
00061     return buffer.Read(signal);
00062 }
```

#### 9.34.4.2 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 }
```

### 9.34.5 Member Data Documentation

#### 9.34.5.1 size\_t DSG::RingBuffer::\_count [protected]

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

#### 9.34.5.2 std::atomic<size\_t> DSG::RingBuffer::\_read [protected]

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

#### 9.34.5.3 std::atomic<size\_t> DSG::RingBuffer::\_write [protected]

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

9.34.5.4 `size_t DSG::RingBuffer::MASK` [protected]

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

9.34.5.5 `size_t DSG::RingBuffer::read` [protected]

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

9.34.5.6 `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:

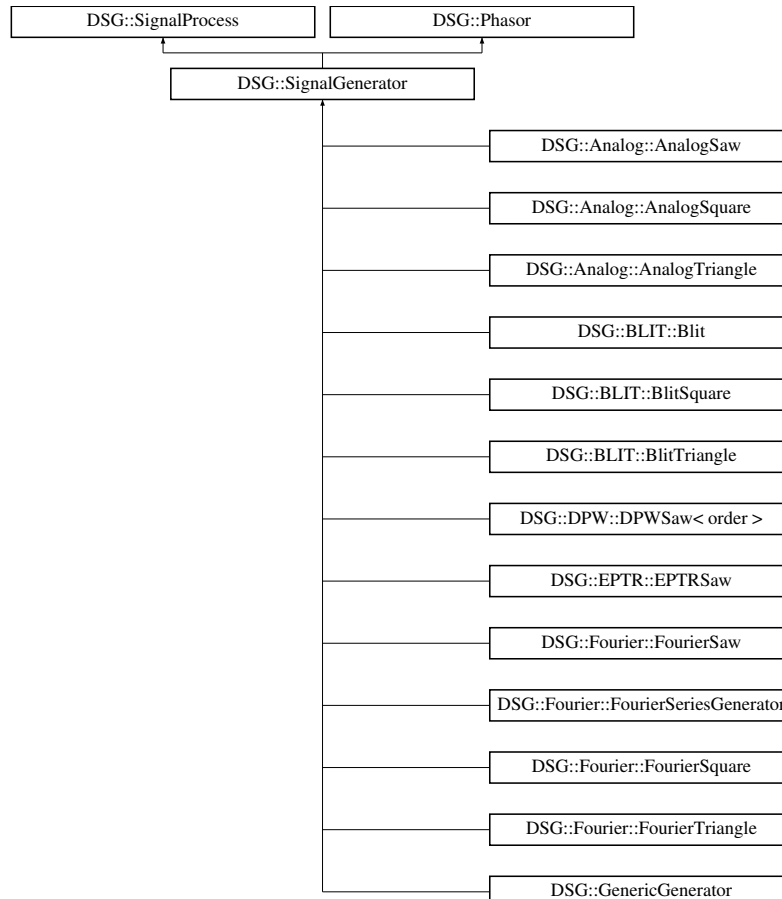
- [RingBuffer.h](#)
- [RingBuffer.cpp](#)

## 9.35 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 bool [SampleRateChanged](#) ([DSG::DSGFrequency](#) const &sampleRate)

## Protected Attributes

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

## Additional Inherited Members

### 9.35.1 Detailed Description

[DSG::SignalGenerator](#) - Extends DSG::Signal Process With Tools For Signal Generation.

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

### 9.35.2 Constructor & Destructor Documentation

#### 9.35.2.1 DSG::SignalGenerator::SignalGenerator ( )

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

```
00025 :DSG::SignalProcess(),DSG::Phasor() {}
```

#### 9.35.2.2 DSG::SignalGenerator::SignalGenerator ( DSG::DSGFrequency const & frequency, DSG::DSGPhase const & offset )

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

```
00026 :DSG::SignalProcess(),DSG::Phasor(frequency,offset) {}
```

#### 9.35.2.3 DSG::SignalGenerator::~~SignalGenerator ( ) [virtual]

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

```
00027 {}
```

### 9.35.3 Member Function Documentation

#### 9.35.3.1 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 49 of file [SignalGenerator.h](#).

```

00049                                     {
00050     signal=0;
00051     return false;
00052 }

```

### 9.35.3.2 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 53 of file [SignalGenerator.h](#).

```

00053                                     {
00054     signal.Flush();
00055     return false;
00056 }

```

### 9.35.3.3 bool DSG::SignalGenerator::SampleRateChanged ( DSG::DSGFrequency const & *sampleRate* ) [inline], [virtual]

Implements [DSG::SignalProcess](#).

Definition at line 57 of file [SignalGenerator.h](#).

```

00057                                     {
00058     Frequency(_frequency);
00059     return true;
00060 }

```

## 9.35.4 Member Data Documentation

### 9.35.4.1 DSG::DSGSample DSG::SignalGenerator::\_storage [protected]

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

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

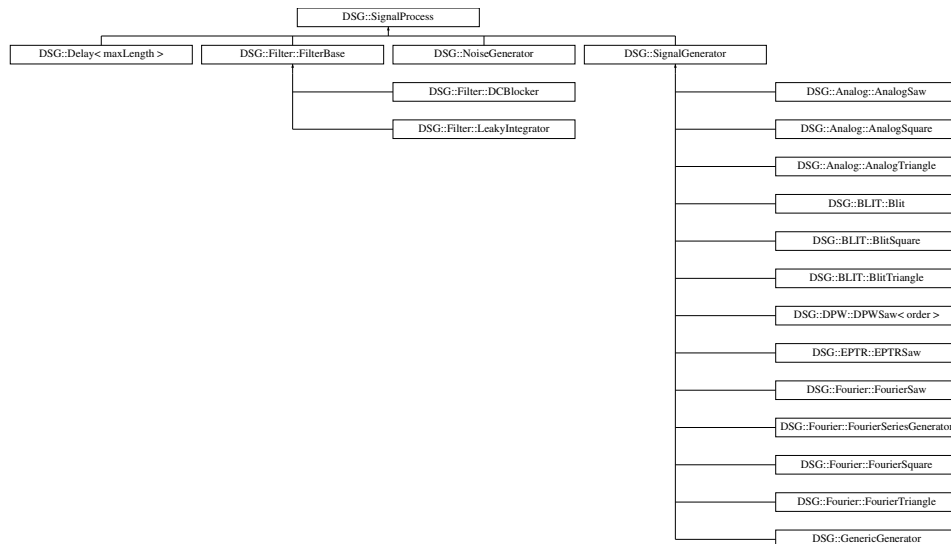
- [SignalGenerator.h](#)
- [SignalGenerator.cpp](#)

## 9.36 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
- virtual bool [SampleRateChanged](#) (DSG::DSGFrequency const &sampleRate)=0

### 9.36.1 Detailed Description

[DSG::SignalProcess](#) - Defines Base Interface For Audio Processing.

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

### 9.36.2 Constructor & Destructor Documentation

#### 9.36.2.1 DSG::SignalProcess::SignalProcess ( )

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

```

00025         {
00026     VerifySampleRateSet(); //ensure that there is a valid sample rate set
00027     AddSampleRateListener(this); //listen for changes in the sample rate
00028 }
```

#### 9.36.2.2 DSG::SignalProcess::~~SignalProcess ( ) [virtual]

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

```

00029 {}
```

### 9.36.3 Member Function Documentation



9.36.3.1 `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::Filter::DCBlocker](#), [DSG::SignalGenerator](#), [DSG::GenericGenerator](#), and [DSG::NoiseGenerator](#).

9.36.3.2 `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::Filter::DCBlocker](#), [DSG::SignalGenerator](#), [DSG::GenericGenerator](#), and [DSG::NoiseGenerator](#).

9.36.3.3 `bool DSG::SignalProcess::SampleRateChanged ( DSG::DSGFrequency const & sampleRate ) [inline],[pure virtual]`

Implemented in [DSG::SignalGenerator](#).

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

```
00041                                     {
00042         return true;
00043     }
```

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

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



## Chapter 10

# File Documentation

### 10.1 AnalogSaw.cpp File Reference

```
#include "AnalogSaw.h"
```

### 10.2 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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() {}
```

### 10.3 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.*

## 10.4 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```

## 10.5 AnalogSquare.cpp File Reference

```
#include "AnalogSquare.h"
```

## 10.6 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "AnalogSquare.h"
00025 DSG::Analog::AnalogSquare():
00026     DSG::SignalGenerator(){}
00027 DSG::Analog::AnalogSquare(
00028     DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
00029     DSG::SignalGenerator(frequency, offset){}
00030 DSG::Analog::AnalogSquare::~AnalogSquare(){}
00031
```

## 10.7 AnalogSquare.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

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

### Namespaces

- [DSG](#)  
[DSG](#) - A Collection of tools for Digital Signal Generation.
- [DSG::Analog](#)  
[DSG::Analog](#) - Namespace Containing [Analog](#) Style Oscillators.

## 10.8 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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__) */

```

## 10.9 AnalogTriangle.cpp File Reference

```
#include "AnalogTriangle.h"
```

## 10.10 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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() {}

```

## 10.11 AnalogTriangle.h File Reference

```
#include "SignalGenerator.h"
```

### Classes

- class [DSG::Analog::AnalogTriangle](#)  
*DSG::Analog::AnalogTriangle - Analog Style Triangle Wave Generator.*

### Namespaces

- [DSG](#)  
*DSG - A Collection of tools for Digital Signal Generation.*
- [DSG::Analog](#)  
*DSG::Analog - Namespace Containing Analog Style Oscillators.*

## 10.12 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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         //!

```

## 10.13 AudioSettings.cpp File Reference

```

#include "AudioSettings.h"
#include "SignalProcess.h"

```

## 10.14 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "AudioSettings.h"
00025 #include "SignalProcess.h"
00026 DSG::DSGFrequency DSG::AudioSettings::_sampleRate;
00027 DSG::DSGFrequency DSG::AudioSettings::_nyquist;

```



```

00028 bool DSG::AudioSettings::_set=false;
00029 std::vector<DSG::SignalProcess*> DSG::AudioSettings::_listeners;
00030 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate() {
00031     return _sampleRate;
00032 }
00033 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(
DSG::DSGFrequency const& value){
00034     if (!_set) {
00035         _set=true;
00036     }
00037     _sampleRate = value;
00038     _nyquist = _sampleRate*0.5;
00039     for (auto i:_listeners) {
00040         i->SampleRateChanged(_sampleRate);
00041     }
00042     return _sampleRate;
00043 }
00044 DSG::DSGFrequency const& DSG::AudioSettings::Nyquist() {
00045     return _nyquist;
00046 }
00047 bool DSG::AudioSettings::AddSampleRateListener(
DSG::SignalProcess* listener){
00048     _listeners.push_back(listener);
00049     return true;
00050 }
00051 bool const& DSG::AudioSettings::IsSampleRateSet() {
00052     return _set;
00053 }

```

## 10.15 AudioSettings.h File Reference

```

#include "DSGTypes.h"
#include <vector>

```

### 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.

### Macros

- #define [SampleRateDefault](#) 44100

### 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.
- bool [DSG::AddSampleRateListener \(DSG::SignalProcess \\*listner\)](#)  
*DSG::AddSampleRateListener()* - Allows Generators to be notified if the sample rate changes.
- void [DSG::VerifySampleRateSet \(\)](#)  
*DSG::VerifySampleRateSet()* - Allows a Generator to ask if a valid sample rate has been set.

## 10.15.1 Macro Definition Documentation

### 10.15.1.1 #define SampleRateDefault 44100

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

## 10.16 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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 #include <vector>
00028 namespace DSG {
00029     class SignalProcess;
00030     /*!\brief DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate
00031     */
00032     class AudioSettings{
00033     public:
00034         static DSG::DSGFrequency const& SampleRate();
00035         static DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value);
00036         static DSG::DSGFrequency const& Nyquist();
00037         static bool AddSampleRateListener(SignalProcess* listener);
00038         static bool const& IsSampleRateSet();
00039     protected:
00040         static DSG::DSGFrequency _sampleRate;
00041         static DSG::DSGFrequency _nyquist;
00042         static std::vector<DSG::SignalProcess*> _listeners;
00043         static bool _set;
00044     };
00045     namespace{
00046 #define SampleRateDefault 44100//hidden macro defining default sample rate
00047     }
00048     /*!\brief DSG::SampleRate - Get Global Sample Rate
00049     inline DSG::DSGFrequency const& SampleRate(){
00050         return DSG::AudioSettings::SampleRate();
00051     }
00052     /*!\brief DSG::SampleRate - Set Global Sample Rate
00053     inline DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value){
00054         return DSG::AudioSettings::SampleRate(value);
00055     }
00056     /*!\brief DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed.
    This value will be updated whenever the sample rate changes.
00057     inline DSG::DSGFrequency Nyquist(){
00058         return DSG::AudioSettings::Nyquist();
00059     }
00060     /*!\brief DSG::AddSampleRateListener() - Allows Generators to be notified if the sample rate changes
00061     inline bool AddSampleRateListener(DSG::SignalProcess* listner){
00062         return AudioSettings::AddSampleRateListener(listner);
00063     }
00064     /*!\brief DSG::VerifySampleRateSet() - Allows a Generator to ask if a valid sample rate has been set
00065     inline void VerifySampleRateSet(){
00066         if (!DSG::AudioSettings::IsSampleRateSet()) {
00067             SampleRate(SampleRateDefault);
00068         }
00069     }
00070 }

```

```
00071 #endif /* defined(__DSG__AudioSettings__) */
```

## 10.17 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.*

## 10.18 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.19 BLIT.cpp File Reference

```
#include "BLIT.h"
```

## 10.20 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
00029     DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){
00029     Frequency(frequency);
00030 }
00031 DSG::BLIT::Blit::~Blit(){}

```

## 10.21 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.*

## 10.22 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
00043                 DSG::DSGPhase const& offset);
00044             virtual ~Blit();
00045             virtual inline bool Perform(DSG::DSGSample& signal);
00046             virtual inline bool Perform(DSG::RingBuffer& signal);
00047             virtual inline DSG::DSGFrequency const& Frequency(
00048                 DSG::DSGFrequency const& value);
00049         protected:
00050             unsigned long p_;
00051             unsigned long m_;
00052             unsigned long _h;
00053             double a_;
00054             DSG::DSGSample denominator;
00055             DSG::DSGSample value;
00056         };
00057         inline bool DSG::BLIT::Blit::Perform(
00058             DSG::DSGSample& signal){
00059             //found better results in this case with built in sine function. not performance wise but
00060             algorithmically
00061             denominator = m_ * sin(_phasor);
00062             if (DSG::IsDenormal(denominator)) {
00063                 signal = a_;
00064             }else{
00065                 value = sin(PI*_phasor * m_);
00066                 value/=denominator;
00067                 value*=a_;
00068                 signal = value;
00069             }
00070             step();
00071             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__) */

```

## 10.23 BLITSaw.cpp File Reference

```
#include "BLITSaw.h"
```

## 10.24 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
DSG::DSGPhase const& offset):DSG::BLIT::Blit(frequency,offset),Register_(0){
00029     Frequency(frequency);
00030 }
00031 DSG::BLIT::BlitSaw::~BlitSaw(){}

```

## 10.25 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.

## 10.26 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
00038 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__) */

```

## 10.27 BLITSquare.cpp File Reference

```
#include "BLITSquare.h"
```

## 10.28 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "BLITSquare.h"

```

## 10.29 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.*

## 10.30 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```

## 10.31 BLITTriangle.cpp File Reference

```
#include "BLITTriangle.h"
```

## 10.32 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #include "BLITTriangle.h"

```

## 10.33 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.*

## 10.34 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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__) */

```

## 10.35 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.*

## 10.36 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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

```

## 10.37 Buffer.cpp File Reference

```
#include "Buffer.h"
```

## 10.38 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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 }

```

## 10.39 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.*

## 10.40 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```

## 10.41 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.*

## 10.42 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
00030         DSG::DSGSample* array, unsigned long length){
00031         for (int i=0; i<length; ++i) {
00032             if (!ring.Empty()) {
00033                 ring.Read(array[i]);
00034             }return true;
00035         }
00036     }
00037     //!\brief DSG::ArrayToRing - Move array data to a Ring Buffer
00038     inline bool ArrayToRing(DSG::RingBuffer& ring,
00039         DSG::DSGSample* array, unsigned long length){
00040         int i=0;
00041         ring.Flush();
00042         while (!ring.Full()) {
00043             ring.Write(array[i]);
00044             ++i;
00045         }return true;
00046     }
00047 }
00048 #endif /* defined(__DSG__BufferConversion__) */

```

## 10.43 DCBlocker.cpp File Reference

```
#include "DCBlocker.h"
```

## 10.44 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(){}

```

## 10.45 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.

## 10.46 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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__) */

```

## 10.47 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.



## 10.48 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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         virtual ~Delay(){}
00052         virtual inline unsigned long const& Length()const{
00053             return _delay;
00054         }
00055         virtual inline unsigned long const& Length(unsigned long const& samples){
00056             if (samples>maxLength) {
00057                 _delay = maxLength;
00058             }else{
00059                 _delay = samples;
00060             }
00061             return _delay;
00062         }
00063         virtual inline bool Perform(DSG::DSGSample& signal);
00064         virtual inline bool Perform(DSG::RingBuffer& signal);
00065     protected:
00066         unsigned long count;
00067         unsigned long _delay;
00068         unsigned long _index;
00069         const unsigned long _max;
00070         DSG::DSGSample _buffer[maxLength];
00071         DSG::DSGSample _swap;
00072         DSG::DSGSample _temp;
00073         virtual inline void increment(){
00074             ++_index;
00075             if (_index>_delay) {
00076                 _index-=_delay;
00077             }
00078         };
00079     template<unsigned long maxLength>
00080     inline bool DSG::Delay<maxLength>::Perform(
00081         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(
00089 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__) */

```

## 10.49 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.*

## 10.50 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.51 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.*

## 10.52 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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){
00040             DSG::StaticAssertBounds<1,6,order>(); //must be 1-6 order
00041             return value;
00042         }
00043         //!\brief DSG::DPW::DPW_Polynomial - 1st Order Polynomial used in DPW Algorithm
00044         template<>
00045         inline DSG::DSGSample DPW_Polynomial<1>(
00046             DSG::DSGSample const& value){
00047             return value;
00048         }
00049         //!\brief DSG::DPW::DPW_Polynomial - 2nd order Polynomial used in DPW Algorithm
00050         template<>
00051         inline DSG::DSGSample DPW_Polynomial<2>(
00052             DSG::DSGSample const& value){
00053             return DSG::Pow<2>(value);
00054         }
00055         //!\brief DSG::DPW::DPW_Polynomial - 3rd order Polynomial used in DPW Algorithm
00056         template<>
00057         inline DSG::DSGSample DPW_Polynomial<3>(
00058             DSG::DSGSample const& value){
00059             return DSG::Pow<3>(value)-value;
00060         }
00061         //!\brief DSG::DPW::DPW_Polynomial - 4th order Polynomial used in DPW Algorithm

```

```

00058     template<>
00059     inline DSG::DSGSample DPW_Polynomial<4>(
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 Polynoimal used in DPW Algorithm
00063     template<>
00064     inline DSG::DSGSample DPW_Polynomial<5>(
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 Polynoimal used in DPW Algorithm
00068     template<>
00069     inline DSG::DSGSample DPW_Polynomial<6>(
DSG::DSGSample const& value){
00070         return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
DPW_Polynomial<2>(value);
00071     }
00072     #ifdef __APPLE__
00073     #warning DSG::DPW - differentiators order 4-6 need verification. they cause major clipping
00074     #endif
00075     ///\todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 4-6
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
Algorithm
00086     template<>
00087     class DPW_Differentiator<1>{
00088     public:
00089         inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00090             return signal;
00091         }
00092     };
00093     ///\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW
Algorithm
00094     template<>
00095     class DPW_Differentiator<2>{
00096     public:
00097         inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00098             output = (signal - _delay)/(4.0 * dt);
00099             _delay = signal;
00100             return output;
00101         }
00102     protected:
00103         DSG::DSGSample output;
00104         DSG::DSGSample _delay;
00105     };
00106     ///\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     * \bug Causes major clipping
00124     */
00125     template<>
00126     class DPW_Differentiator<4>{
00127     public:
00128         inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00129             output = (signal - _delay[0]);
00130             output -= (_delay[0] - _delay[1]);
00131             output -= (_delay[1] - _delay[2]);
00132             output /= 144*DSG::Pow<3>(dt);

```

```

00133         _delay[2]=_delay[1];
00134         _delay[1]=_delay[0];
00135         _delay[0]=signal;
00136         return output;
00137     }
00138     protected:
00139         DSG::DSGSample output;
00140         DSG::DSGSample _delay[3];
00141     };
00142     /*!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW
Algorithm
00143     * \bug Causes major clipping
00144     */
00145     template<>
00146     class DPW_Differentiator<5>{
00147     public:
00148         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00149             output = (signal - _delay[0]);
00150             output -= (_delay[0] - _delay[1]);
00151             output -= (_delay[1] - _delay[2]);
00152             output -= (_delay[2] - _delay[3]);
00153             output /= 960*DSG::Pow<4>(dt);
00154             _delay[3]=_delay[2];
00155             _delay[2]=_delay[1];
00156             _delay[1]=_delay[0];
00157             _delay[0]=signal;
00158             return output;
00159         }
00160     protected:
00161         DSG::DSGSample output;
00162         DSG::DSGSample _delay[4];
00163     };
00164     /*!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW
Algorithm
00165     * \bug Causes major clipping
00166     */
00167     template<>
00168     class DPW_Differentiator<6>{
00169     public:
00170         inline DSG::DSGSample operator() (
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00171             output = (signal - _delay[0]);
00172             output -= (_delay[0] - _delay[1]);
00173             output -= (_delay[1] - _delay[2]);
00174             output -= (_delay[2] - _delay[3]);
00175             output -= (_delay[3] - _delay[4]);
00176             output /= 7200*DSG::Pow<5>(dt);
00177             _delay[4]=_delay[3];
00178             _delay[3]=_delay[2];
00179             _delay[2]=_delay[1];
00180             _delay[1]=_delay[0];
00181             _delay[0]=signal;
00182             return output;
00183         }
00184     protected:
00185         DSG::DSGSample output;
00186         DSG::DSGSample _delay[5];
00187     };
00188     }
00189 }
00190 #endif

```

## 10.53 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.*

## 10.54 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```

## 10.55 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](#))

## 10.55.1 Macro Definition Documentation

### 10.55.1.1 #define BufferSize 512

Definition at line 10 of file [Driver.cpp](#).

## 10.55.2 Function Documentation

### 10.55.2.1 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     }

```



## 10.55.2.2 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 }

```

## 10.55.2.3 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(),
    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 }

```

## 10.55.3 Variable Documentation

## 10.55.3.1 DSG::RingBuffer \_buffer(BufferSize)

## 10.55.3.2 PaStream\* stream

Definition at line 9 of file [Driver.cpp](#).

## 10.56 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(),
    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 }
00038 int DriverExit(){
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 }
00061 int Callback(const void *input,
00062             void *output,
00063             unsigned long frameCount,
00064             const PaStreamCallbackTimeInfo* timeInfo,
00065             PaStreamCallbackFlags statusFlags,
00066             void *userData) {
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 }

```

## 10.57 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)

### 10.57.1 Function Documentation

**10.57.1.1** 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 }
```

**10.57.1.2** 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 }
```

### 10.57.1.3 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 }

```

## 10.58 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__) */

```

## 10.59 DSF.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)`

## 10.60 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__DSF__
00025 #define __DSG__DSF__
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__DSF__) */

```

## 10.61 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](#)

#### 10.61.1 Macro Definition Documentation

##### 10.61.1.1 `#define` [DSG\\_Short\\_Names](#)

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

## 10.62 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 #ifndef DSG_DSG_h
00009 #define DSG_DSG_h
00010 /*!\mainpage DSG - A Collection Of Tools For Digital Signal Generation
00011 *\section intro_sec Introduction
00012 The Digital Signal Generation Project or DSG is a collection of tools used for the generation of digital
00013 signals, more specifically the generation of band-limited waveforms.
00014 *\subsection Scope
00015 Though DSG has a focus on Bandlimited Waveform Generation it is not limited to it.
00016 DSG defines a signal processing interface that is compatible with any form of audio based signal
00017 processing work.
00018 The interface defined in DSG::SignalProcess is the base interface for signal processing in DSG, It is
00019 further expanded by DSG::SignalGenerator which adds functionality geared towards waveform generation.
00020 See the documentation for each for their specifics.
00021 *\section License
00022 DSG is released under the Lesser GNU Public License (LGPL).
00023
00024 A copy of the LGPL and the GNU Public License should be included with the distribution in the files:
00025 COPYING (GPL), and COPYING.LESSER (LGPL)
00026 Additionally each source file should contain a copy of the license notice which reads as follows:
00027 \copyright
00028 This file is part of the Digital Signal Generation Project or "DSG".
00029
00030 DSG is free software: you can redistribute it and/or modify
00031 it under the terms of the GNU Lesser General Public License as published by
00032 the Free Software Foundation, either version 3 of the License, or
00033 (at your option) any later version.
00034
00035 DSG is distributed in the hope that it will be useful,
00036 but WITHOUT ANY WARRANTY; without even the implied warranty of
00037 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00038 GNU Lesser General Public License for more details.
00039
00040 You should have received a copy of the GNU Lesser General Public License
00041 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00042
00043 */
00044 #define DSG_Short_Names // enables inlining of nested namespaces to allow shorter explicit typenames
00045 //Example: DSG::Analog::AnalogSaw (Long Name)...DSG::AnalogSaw (Short Name) (only available with this macro
00046 enabled
00047 /*!\brief DSG - A Collection of tools for Digital Signal Generation
00048 /*!\copyright Lesser GNU Public License
00049 /*!\todo Increase documentation level. Add documentation for every variable, parameter...
00050 /*!\todo Implement Blep Based Algorithms
00051 namespace DSG {
00052 #include "AudioSettings.h"
00053 #include "SignalProcess.h"
00054 #include "Buffer.h"
00055 #include "RingBuffer.h"
00056 #include "SignalGenerator.h"
00057 #include "Sine.h"
00058 #include "Sinc.h"
00059 #include "Denormal.h"
00060 #include "Math.h"
00061 #include "Blackman.h"
00062 #include "LUT.h"
00063 #include "Window.h"
00064 #include "Bounds.h"
00065 #include "GenericGenerator.h"
00066 #include "Delay.h"
00067 #include "Sleep.h"
00068 #include "BufferConversion.h"
00069 #include "FourierSeries.h"
00070 #include "FourierSaw.h"
00071 #include "FourierSquare.h"
00072 #include "FourierTriangle.h"
00073 #include "AnalogSaw.h"
00074 #include "AnalogSquare.h"
00075 #include "AnalogTriangle.h"
00076 #include "BLIT.h"
00077 #include "BLITSaw.h"
00078 #include "DSF.h"
00079 #include "DPW.h"
00080 #include "DPWSaw.h"
00081 #include "EPTRSaw.h"
00082 #include "Noise.h"
00083 #include "DCBlocker.h"
00084 #include "Filter.h"

```

```
00080 #include "Leaky.h"
00081 #endif
```

## 10.63 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.  $N^I$ .

## 10.64 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.65 DSGTypes.h File Reference

### Namespaces

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

### Typedefs

- typedef float [DSG::DSGFrequency](#)
- typedef float [DSG::DSGPhase](#)
- typedef float [DSG::DSGSample](#)

## 10.66 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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     //!\typedef DSG::DSGFrequency - Type for representing a frequency value
00028     //!\brief DSG::DSGFrequency - Type for representing a frequency value
00029     typedef float DSGFrequency;
00030     //!\typedef DSG::DSGPhase - Type for representing a phase value
00031     //!\brief DSG::DSGPhase - Type for representing a phase value
00032     typedef float DSGPhase;
00033     //!\typedef DSG::DSGSample - Type for representing an audio sample
00034     //!\brief DSG::DSGSample - Type for representing an audio sample
00035     typedef float DSGSample;
00036 }
00037 #endif

```

## 10.67 EPTRsaw.cpp File Reference

```
#include "EPTRsaw.h"
```

## 10.68 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
    DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){}
00027 DSG::EPTR::EPTRsaw::~~EPTRsaw(){}

```

## 10.69 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.*

## 10.70 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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         ///!bug Algorithm is not performing in a band limited manor
00037         class EPTRsaw : public DSG::SignalGenerator{
00038         public:
00039             EPTRsaw();
00040             EPTRsaw(DSG::DSGFrequency const& frequency,
00041 DSG::DSGPhase const& offset);
00042             virtual ~EPTRsaw();
00043             virtual inline bool Perform(DSG::DSGSample& signal);
00044             virtual inline bool Perform(DSG::RingBuffer& signal);
00045         protected:
00046             DSG::DSGSample _register;
00047         };
00048         inline bool DSG::EPTR::EPTRsaw::Perform(
00049 DSG::DSGSample& signal){
00050 #ifdef __APPLE__
00051 #warning Untested For Aliasing DSG::EPTR::EPTRsaw::Perform()
00052 #endif
00053             //generate trivial saw
00054             _register = _phasor;
00055             _register+=0.5;
00056             if (_register>1.0) {
00057                 --_register;
00058             }
00059             _register-=0.5;
00060             _register*=2.0;
00061             if (_register > 1.0-_dt) {
00062                 //transition region detected
00063                 //apply eptr correction
00064                 signal = _register - (_register/_dt) + (1.0/_dt) -1;
00065             }else{
00066                 signal = _register;
00067             }
00068             step();//avance phase
00069             return true;
00070         }
00071         inline bool DSG::EPTR::EPTRsaw::Perform(
00072 DSG::RingBuffer& signal){
00073             signal.Flush();
00074             while (!signal.Full()) {
00075                 if (Perform(_storage)) {
00076                     if(signal.Write(_storage)){

```

```

00073         }else return false;
00074     }else return false;
00075     }return true;
00076 }
00077 }
00078 }
00079 #endif /* defined(__DSG__EPTRSaw__) */

```

## 10.71 Filter.cpp File Reference

```
#include "Filter.h"
```

## 10.72 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "Filter.h"
00025 DSG::Filter::FilterBase::FilterBase():_temp(0),count(0){}
00026 DSG::Filter::FilterBase::~FilterBase(){}

```

## 10.73 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.*

## 10.74 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(
00046         DSG::DSGSample& signal){
00047             return true;
00048         }
00049         inline bool DSG::Filter::FilterBase::Perform(
00050         DSG::RingBuffer& signal){
00051             if (!signal.Empty()) {
00052                 count = signal.Count();
00053                 while (count-- > 0) {
00054                     if(signal.Read(_temp)){
00055                         if (Perform(_temp)) {
00056                             signal.Write(_temp);
00057                         }else return false;
00058                     }return true;
00059                 }else return false;
00060             }
00061             inline bool DSG::Filter::FilterBase::Cutoff(
00062             DSG::DSGFrequency const& cutoff){
00063                 return false;
00064             }
00065         }
00066     }
00067 #endif /* defined(__DSG__Filter__) */

```

## 10.75 FourierSaw.cpp File Reference

```
#include "FourierSaw.h"
```

## 10.76 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(
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(){}

```

## 10.77 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.*

## 10.78 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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__) */

```

## 10.79 FourierSeries.cpp File Reference

```
#include "FourierSeries.h"
```

## 10.80 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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){}
00028 DSG::Fourier::Harmonic::~Harmonic(){
00029     _ratio=0;
00030     _amplitude=0;
00031 }
00032 DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio()const{
00033     return _ratio;
00034 }
00035 DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio(
00036 DSG::DSGSample const& value){
00037     _ratio = value;
00038     return _ratio;
00039 }
00040 DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude()
00041 const{
00042     return _amplitude;
00043 }
00044 DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude(
00045 DSG::DSGSample const& value){
00046     _amplitude=value;
00047     return _amplitude;
00048 }
00049 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
00050 ():DSG::SignalGenerator(){}
00051 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
00052 (DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
00053 DSG::SignalGenerator(frequency,offset){}
00054 DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator
00055 (){}

```

## 10.81 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.*

## 10.82 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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) {

```

```

00066         signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
00067     }
00068     step();
00069     return true;
00070 }
00071 inline bool DSG::Fourier::FourierSeriesGenerator::Perform
(DSG::RingBuffer& signal){
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     }
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__) */

```

## 10.83 FourierSquare.cpp File Reference

```
#include "FourierSquare.h"
```

## 10.84 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(){}

```

## 10.85 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.*

## 10.86 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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,
00038 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__) */

```

## 10.87 FourierTriangle.cpp File Reference

```
#include "FourierTriangle.h"
```

## 10.88 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(){}

```

## 10.89 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.*

## 10.90 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(
00042                 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(
00050             DSG::DSGSample& signal){
00051             //(_h/2)+1 Sine Calls Per Sample
00052             value=DSG::Sin(_phasor);//i=1
00053             double sgn = -1;
00054             for (i=3; i<_h; i+=2) {i=3..5..7..
00055                 value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*i);
00056                 sgn*=-1;
00057             }
00058             value*=_a;
00059             signal = value;
00060             step();
00061             return true;
00062         }
00063         inline bool DSG::Fourier::FourierTriangle::Perform(
00064             DSG::RingBuffer& signal){
00065             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__) */

```

## 10.91 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.*

## 10.92 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.93 GenericGenerator.cpp File Reference

```
#include "GenericGenerator.h"
```

## 10.94 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "GenericGenerator.h"
00025 DSG::GenericGenerator::GenericGenerator():
    DSG::SignalGenerator(){}
00026 DSG::GenericGenerator::GenericGenerator(
    DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset,
    DSG::DSGSample (*signalFunction)(DSG::DSGSample const&)):
    DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
00027 DSG::GenericGenerator::~GenericGenerator(){}

```

## 10.95 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.*

## 10.96 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```



## 10.97 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.*

## 10.98 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 Lesser 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 Lesser General Public License for more details.
00023
00024  You should have received a copy of the GNU Lesser 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     //!\brief DSG::CubicInterpolate - Cubic Interpolation
00048     template<typename decimal>
00049     decimal CubicInterpolate(decimal const& y0, decimal const& y1,
00050         decimal const& y2, decimal const& y3,
00051         decimal const& mu)
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     }
00061     //!\brief DSG::HermiteInterpolate - Hermite Interpolation
00062     template<typename decimal>
00063     decimal HermiteInterpolate(decimal const& y0, decimal const& y1,
00064         decimal const& y2, decimal const& y3,
00065         decimal const& mu,
00066         decimal const& tension,
00067         decimal const& bias)
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     }
00089 }
00090 #endif

```

## 10.99 Leaky.cpp File Reference

```
#include "Leaky.h"
```

## 10.100 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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 }

```

## 10.101 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.

## 10.102 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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__) */

```

## 10.103 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.*

## 10.104 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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     //!\todo Implement interpolation into lookup algorithm
00033     template <typename element, unsigned long size>
00034     class LUT {
00035     public:
00036         typedef element (*FillFunction)(element);
00037         typedef element (*FillFunctionConstRef)(element const&);
00038         LUT():_size(size){}
00039         LUT(FillFunction fill, double const& range = 1.0):_size(size){
00040             //range is the expected input range for the function
00041             //example would be 0-2pi or 0-1
00042             //would be provided a 2pi or 1
00043             //defaults to 1
00044             double step = range/(double)_size;
00045             phs = 0;
00046             for (int i=0; i<_size; ++i) {
00047                 _table[i] = fill(phs);
00048                 phs+=step;
00049             }
00050         }
00051         LUT(FillFunctionConstRef fill, double const& range = 1.0):
00052         _size(size){
00053             //range is the expected input range for the function
00054             //example would be 0-2pi or 0-1
00055             //would be provided a 2pi or 1
00056             //defaults to 1
00057             double step = range/_size;
00058             phs = 0;
00059             for (int i=0; i<_size; ++i) {
00060                 _table[i] = fill(phs);
00061                 phs+=step;
00062             }
00063         }
00064         ~LUT(){}
00065         element const& operator[] (unsigned long const& index) const{
00066             #ifdef DEBUG
00067                 assert(index<_size);
00068             #endif
00069             return _table[index];
00070         }
00071         element& operator[] (unsigned long const& index){
00072             #ifdef DEBUG
00073                 assert(index<_size);
00074             #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;
00085     }
00086     protected:
00087         element _table[size];
00088         const unsigned long _size;
00089         double phs;
00090     };
00091 }
00092 #endif

```

## 10.105 MTOF.cpp File Reference

```
#include "MTOF.h"
```

## 10.106 MTOF.cpp

```

00001 //
00002 // MTOF.cpp
00003 // DSG
00004 //
00005 // Created by Alexander Zywicki on 11/25/14.
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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "MTOF.h"
00025 double DSG::MIDI::MTOF(unsigned char const& MIDI_Number){
00026     return 440.0 *pow(2.0, (MIDI_Number-69.0)/12.0);
00027 }
00028 unsigned char DSG::MIDI::FTOM(double const& Frequency){
00029     return((log2((Frequency/440.0)))*12.0)+69.0;
00030 }

```

## 10.107 MTOF.h File Reference

```
#include <math.h>
```

### Namespaces

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

*DSG::MIDI* - Namespace enclosing *MIDI* processing tools.

## Functions

- double *DSG::MIDI::MTOF* (unsigned char const &MIDI\_Number)  
*DSG::MIDI::MTOF* - *MIDI* to Frequency Conversion.
- unsigned char *DSG::MIDI::FTOM* (double const &Frequency)  
*DSG::MIDI::FTOM* - Frequency to *MIDI* Conversion.

## 10.108 MTOF.h

```

00001 //
00002 //  MTOF.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/25/14.
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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG_MTOF__
00025 #define __DSG_MTOF__
00026 #include <math.h>
00027 namespace DSG{
00028 #ifdef DSG_Short_Names
00029     inline
00030 #endif
00031     //!\brief DSG::MIDI - Namespace enclosing MIDI processing tools
00032     namespace MIDI{
00033         //!\brief DSG::MIDI:MTOF - MIDI to Frequency Conversion
00034         double  MTOF(unsigned char const& MIDI_Number);
00035         //!\brief DSG::MIDI:FTOM - Frequency to MIDI Conversion
00036         unsigned char FTOM(double const& Frequency);
00037     }
00038 }
00039 #endif /* defined(__DSG_MTOF__) */

```

## 10.109 Noise.h File Reference

```

#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"

```

## 10.110 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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

```

## 10.111 NoiseGenerator.cpp File Reference

```
#include "NoiseGenerator.h"
```

## 10.112 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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(){}

```

## 10.113 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.*

## 10.114 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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
00029     signal
00030     class NoiseGenerator:public SignalProcess{
00031     public:
00032         NoiseGenerator(DSGSample (*StatelessFunction) (
00033         DSGSample));
00034         virtual ~NoiseGenerator();
00035         virtual inline bool Perform(DSG::DSGSample& signal);
00036         virtual inline bool Perform(DSG::RingBuffer& signal);
00037     protected:
00038         DSGSample (*_function) (DSGSample);
00039         DSG::DSGSample _storage;
00040     };
00041     inline bool DSG::NoiseGenerator::Perform (
00042     DSG::DSGSample& signal){
00043         signal = _function(0);
00044         return true;
00045     }
00046     inline bool DSG::NoiseGenerator::Perform (
00047     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_NoiseGenerator__) */

```

## 10.115 Phasor.cpp File Reference

```
#include "Phasor.h"
```

## 10.116 Phasor.cpp

```
00001 //
00002 //  Phasor.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/26/14.
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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser General Public License
00022  along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #include "Phasor.h"
00025 DSG::Phasor::Phasor():_phasor(0),_frequency(0),_dt(0),_offset(0){}
00026 DSG::Phasor::Phasor(DSG::DSGFrequency const& frequency,
00027                     DSG::DSGPhase const& offset):_phasor(0),_frequency(frequency),_dt(0),_offset(offset){
00028     Frequency(frequency);
00029     Phase(offset);
00030 }
00030 DSG::Phasor::~~Phasor(){}

```

## 10.117 Phasor.h File Reference

```
#include "DSGTypes.h"
#include "Bounds.h"
#include "AudioSettings.h"
```

### Classes

- class [DSG::Phasor](#)  
*DSG::Phasor* - Linear Phase Generator.

### Namespaces

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

## 10.118 Phasor.h

```
00001 //
00002 //  Phasor.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/26/14.

```

```

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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with DSG. If not, see <http://www.gnu.org/licenses/>.
00023 */
00024 #ifndef __DSG__Phasor__
00025 #define __DSG__Phasor__
00026 #include "DSGTypes.h"
00027 #include "Bounds.h"
00028 #include "AudioSettings.h"
00029 namespace DSG{
00030     //!\brief DSG::Phasor - Linear Phase Generator
00031     class Phasor{
00032     public:
00033         Phasor();
00034         Phasor(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00035         virtual ~Phasor();
00036         virtual inline DSG::DSGFrequency const& Frequency();
00037         virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
00038         virtual inline DSG::DSGPhase const& Phase();
00039         virtual inline DSG::DSGPhase const& Phase(
DSG::DSGPhase const& value);
00040     protected:
00041         //extends sample rate interface
00042         inline void step();
00043         inline void sync();
00044         //-----//
00045         DSG::DSGFrequency _frequency;//frequency in Hz
00046         DSG::DSGPhase _dt;//delta time (change in phase per sample) unit: phase 0-1
00047         DSG::DSGPhase _offset;//phase shift
00048         DSG::DSGPhase _phasor;//phase counter
00049     };
00050     inline DSG::DSGFrequency const& DSG::Phasor::Frequency(){
00051         return _frequency;
00052     }
00053     inline DSG::DSGFrequency const& DSG::Phasor::Frequency(
DSG::DSGFrequency const& value){
00054         _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
00055         _dt = _frequency/DSG::SampleRate();
00056         return _frequency;
00057     }
00058     inline DSG::DSGPhase const& DSG::Phasor::Phase(){
00059         return _offset;
00060     }
00061     inline DSG::DSGPhase const& DSG::Phasor::Phase(
DSG::DSGPhase const& value){
00062         _offset-=value;
00063         _phasor-=_offset;
00064         _offset=value;
00065         return _offset;
00066     }
00067     inline void DSG::Phasor::step(){
00068         _phasor+=_dt;
00069         _phasor>1.0 ? --_phasor:0;
00070     }
00071     inline void DSG::Phasor::sync(){
00072         _phasor=_offset;
00073     }
00074 }
00075 #endif /* defined(__DSG__Phasor__) */

```

## 10.119 Pl.h File Reference

### Namespaces

- [DSG](#)

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

## Macros

- `#define PI 3.14159265358979323846264338327`
- `#define TWOPI 6.28318530717958647692528676656`

### 10.119.1 Macro Definition Documentation

#### 10.119.1.1 `#define PI 3.14159265358979323846264338327`

Definition at line 27 of file [PI.h](#).

#### 10.119.1.2 `#define TWOPI 6.28318530717958647692528676656`

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

## 10.120 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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
```

### 10.121 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.*

## 10.122 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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         //!\brief DSG::Noise::Pink - Pink Noise Generator Function
00034         template<typename decimal=DSG::DSGSample>
00035         decimal Pink(decimal=0.0){
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         }
00059     }
00060 }
00061 #endif

```

## 10.123 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.*

### 10.123.1 Variable Documentation

#### 10.123.1.1 `const decimal max = static_cast<decimal>(RAND_MAX)`

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

## 10.124 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.125 RingBuffer.cpp File Reference

```
#include "RingBuffer.h"
```

## 10.126 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.127 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.*

## 10.128 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */

```

## 10.129 SignalGenerator.cpp File Reference

```
#include "SignalGenerator.h"
```

## 10.130 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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(),DSG::Phasor(){}
00026  DSG::SignalGenerator::SignalGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalProcess(),DSG::Phasor(frequency,offset){}
00027  DSG::SignalGenerator::~SignalGenerator(){}

```

## 10.131 SignalGenerator.h File Reference

```

#include "SignalProcess.h"
#include "Sine.h"
#include "Phasor.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)

## 10.132 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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 "Sine.h"
00028  #include "Phasor.h"
00029  namespace DSG{

```

```

00030      /*!\brief DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation
00031      */
00032      class SignalGenerator:public DSG::SignalProcess,public
DSG::Phasor{
00033      public:
00034          SignalGenerator();
00035          SignalGenerator(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
00036          virtual ~SignalGenerator();
00037          virtual inline bool Perform(DSG::DSGSample& signal);
00038          virtual inline bool Perform(DSG::RingBuffer& signal);
00039          virtual inline bool SampleRateChanged(DSG::DSGFrequency const&
sampleRate);
00040      protected:
00041          DSG::DSGSample _storage;//storage variable for calculations
00042      };
00043      inline unsigned long MaxHarms(DSG::DSGFrequency const& frequency){
00044          double _s = DSG::SampleRate()* 20000.0/DSG::SampleRate();
00045          _s/=frequency;
00046          return _s;
00047      }
00048  }
00049  inline bool DSG::SignalGenerator::Perform(
DSG::DSGSample& signal){
00050      signal=0;
00051      return false;
00052  }
00053  inline bool DSG::SignalGenerator::Perform(
DSG::RingBuffer& signal){
00054      signal.Flush();
00055      return false;
00056  }
00057  inline bool DSG::SignalGenerator::SampleRateChanged(
DSG::DSGFrequency const& sampleRate){
00058      Frequency(_frequency);
00059      return true;
00060  }
00061
00062  #endif /* defined(__DSG__SignalGenerator__) */

```

## 10.133 SignalProcess.cpp File Reference

```
#include "SignalProcess.h"
```

## 10.134 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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     VerifySampleRateSet();//ensure that there is a valid sample rate set
00027     AddSampleRateListener(this);//listen for changes in the sample rate
00028 }
00029 DSG::SignalProcess::~SignalProcess(){}

```

## 10.135 SignalProcess.h File Reference

```
#include "DSGTypes.h"
#include "RingBuffer.h"
#include "AudioSettings.h"
```

### Classes

- class [DSG::SignalProcess](#)  
*DSG::SignalProcess - Defines Base Interface For Audio Processing.*

### Namespaces

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

## 10.136 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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 #include "AudioSettings.h"
00029 namespace DSG {
00030     /*!\brief DSG::SignalProcess - Defines Base Interface For Audio Processing
00031     */
00032     class SignalProcess{
00033     public:
00034         SignalProcess();
00035         virtual ~SignalProcess();
00036         //Defines Interface for sample rate processing
00037         virtual inline bool Perform(DSG::DSGSample& signal)=0;
00038         virtual inline bool Perform(DSG::RingBuffer& signal)=0;
00039         virtual inline bool SampleRateChanged(DSG::DSGFrequency const&
sampleRate)=0;
00040     };
00041     inline bool DSG::SignalProcess::SampleRateChanged(
DSG::DSGFrequency const& sampleRate){
00042         return true;
00043     }
00044 }
00045 #endif /* defined(__DSG__SignalProcess__) */
```

## 10.137 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 ( $\sin(PI*x)/PI*x$ )*

## 10.138 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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(PI*x)/PI*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__) */
```

## 10.139 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.*

### 10.139.1 Macro Definition Documentation

#### 10.139.1.1 `#define` LUT\_SIZE 16384

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

## 10.140 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 Lesser 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 Lesser General Public License for more details.
```

```

00020
00021 You should have received a copy of the GNU Lesser 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     /*!\todo Implement Taylor Series implementation of Sin Function
00067     inline double Sin(double const& x){
00068         return static_cast<double>(Sin<Sine_Default>(x)); //wrap default implementation as non template
00069     }
00070     /*!\brief DSG::Sin() - General Purpose Sin Function, single precision
00071     */
00072     inline float Sin(float const& x){
00073         return static_cast<float>(Sin<Sine_Default>(x));
00074     }
00075     /*!\brief DSG::Cos() - General Purpose Cos Function, double precision
00076     */
00077     /*!\todo Implement Taylor Series implementation of Cos Function
00078     inline double Cos(double const& x){
00079         return static_cast<double>(Cos<Sine_Default>(x)); //wrap default implementation as non template
00080     }
00081     /*!\brief DSG::Cos() - General Purpose Cos Function, single precision
00082     */
00083     inline float Cos(float const& x){
00084         return static_cast<float>(Cos<Sine_Default>(x));
00085     }
00086 }
00087 #endif /* defined(__DSG__Sine__) */

```

## 10.141 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.*

## 10.142 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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__) */
```

## 10.143 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.*

## 10.144 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 Lesser 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 Lesser General Public License for more details.
00020
00021  You should have received a copy of the GNU Lesser 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

```

## 10.145 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 lutsizes>`  
`void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsizes > &lut, decimal(&>windowFunction)(decimal const &), decimal range=1.0)`

*DSG::Window::ApplyWindow* - Apply a window function to a *LUT*.

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

*DSG::Window::ApplyWindow* - Apply a window function to a *LUT*.

## 10.146 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 Lesser 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 Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser 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, [65](#)

DSG::Factorial< 0 >

value, [66](#)

value

DSG::Factorial, [65](#)

DSG::Factorial< 0 >, [66](#)