DSG

Generated by Doxygen 1.8.8

Sun Nov 30 2014 00:49:54

Contents

Chapter 1

DSG - A Collection Of Tools For Digital Signal Generation

1.1 Intoduction

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 compatable with any form fo 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 doumentation 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 distrobution 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:

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 interploation 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:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

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 Efficienct 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∗x)/PI∗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.

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

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.

```
                                                                {
        return AudioSettings::AddSampleRateListener(listner);
    }
```

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.

```
                                                                                                          {
        int i=0;
        ring.Flush();
        while (!ring.Full()) {
            ring.Write(array[i]);
            ++i;
        }return true;
    }
```

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.

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

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 Function

Definition 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.

```
                                                        {
        if (value<lower) {
            return lower;
        }else if(value> upper){
            return upper;
        }else return value;
    }
```

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.

```
    {
        /*
         Tension: 1 is high, 0 normal, -1 is low
         Bias: 0 is even,
         positive is towards first segment,
         negative towards the other
         */
        decimal m0,m1,mu2,mu3;
        decimal a0,a1,a2,a3;
        mu2 = mu * mu;
        mu3 = mu2 * mu;
        m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
        m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
        m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
        m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
        a0 =  2*mu3 - 3*mu2 + 1;
        a1 =    mu3 - 2*mu2 + mu;
        a2 =    mu3 -   mu2;
        a3 = -2*mu3 + 3*mu2;
        return(a0*y1+a1*m0+a2*m1+a3*y2);
    }
```

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.

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

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.

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

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

Definition at line 43 of file SignalGenerator.h.

```
                                                                              {
        double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
        _s/=frequency;
        return _s;
    }
```

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.

```
                                        {
        return DSG::AudioSettings::Nyquist();
    }
```

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.

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

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.

```
                                                                                                           {
        for (int i=0; i<length; ++i) {
            if (!ring.Empty()) {
                ring.Read(array[i]);
            }
        }return true;
    }
```

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

DSG::SampleRate - Get Global Sample Rate.

Definition at line 49 of file AudioSettings.h.

```
                                                  {
        return DSG::AudioSettings::SampleRate();
    }
```

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.

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

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.

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

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.

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

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.

```
                                                {
            static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
      Point Type");
            decimal pix;
            if (DSG::IsDenormal(x)) {
                return 1.0;
            }else{
                pix = PI∗x;
                return DSG::Sin(pix)/pix;
            }
        }
```

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.

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

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.

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

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.

```
                                     {
        if (!DSG::AudioSettings::IsSampleRateSet()) {
            SampleRate(SampleRateDefault);
        }
    }
```

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 - Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 1 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 2 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 3 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 4 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 5 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 6 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 6th order Polynoimal 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 - Polynoimal 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 Polynoimal 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 Polynoimal 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 Polynoimal 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 Polynoimal used in DPW Algorithm.

Definition at line 59 of file DPW.h.

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

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

DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.

Definition at line 64 of file DPW.h.

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

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

DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm.

Definition at line 69 of file DPW.h.

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

8.5 DSG::EPTR Namespace Reference

DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.

Classes

- class EPTRSaw

    DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

8.5.1 Detailed Description

DSG::EPTR - Generators Based On The Efficienct 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.

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

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.

```
                                     {
            static decimal normalizer=1;//variable used to actively normalize the output
            //to enforce compatability with DSG::LUT a dummy parameter is applied
            //this parameter is useless except for compatability reasons
            decimal R1 = DSG::Noise::White();
            decimal R2 = DSG::Noise::White();
            decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
            if (DSG::Abs(x)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(x);
            }
            x/=normalizer;//normalize
            return x;
        }
```

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.

```
                                  {
            //routine: Get white or gaussian, filter, return
            static decimal b0,b1,b2,b3,b4,b5,b6;
            static decimal normalizer=1;//variable used to actively normalize the output
            static DSG::DCBlocker _block;
            decimal white = DSG::Noise::Gaussian();
            decimal pink;
            //pinking filter
            b0 = 0.99886 * b0 + white * 0.0555179;
            b1 = 0.99332 * b1 + white * 0.0750759;
            b2 = 0.96900 * b2 + white * 0.1538520;
            b3 = 0.86650 * b3 + white * 0.3104856;
            b4 = 0.55000 * b4 + white * 0.5329522;
            b5 = -0.7616 * b5 - white * 0.0168980;
            pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
            b6 = white * 0.115926;
            if (DSG::Abs(pink)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(pink);
            }
            pink/=normalizer;
            _block.Perform(pink);
            return pink;
        }
```

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.

```
                                           {
            static DSG::Noise::random_helper<decimal> _rand{};
            return _rand.next();
        }
```

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.

```
                                         {
            return DSG::Random<decimal>();
        }
```

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 lutsize>
  void ApplyWindow (DSG::LUT< decimal, lutsize > &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.

```
               {
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
```

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.

```
        {
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
```

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.

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

Chapter 9

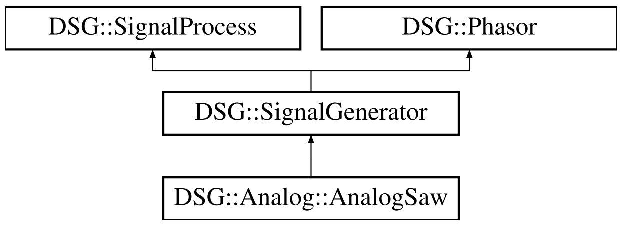
Class Documentation

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.

```
: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.

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

9.1.2.3 DSG::Analog::AnalogSaw::∼AnalogSaw ( ) [virtual]

Definition at line 27 of file AnalogSaw.cpp.

```
{}
```

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.

```
                                                                     {
            _stor=_phasor;
            _stor+=0.5;
            if (_stor>1.0) {
                --_stor;
            }
            _stor-=0.5;
            _stor*=2.0;
            signal=_stor;
            step();
            return true;
        }
```

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.

```
                                                                      {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

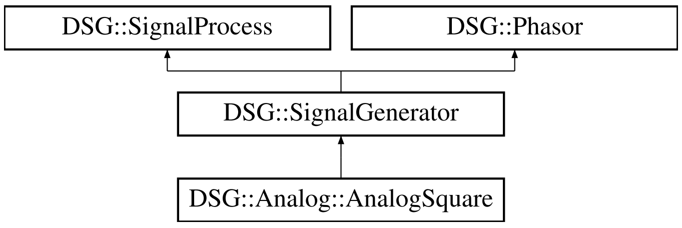
- AnalogSaw.h
- AnalogSaw.cpp

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:

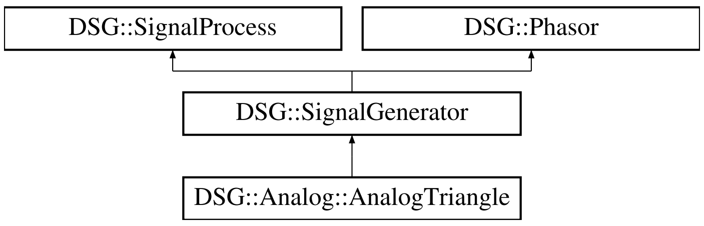
- AnalogSquare.h
- AnalogSquare.cpp

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.

```
                                                                    {
            _stor = _phasor;
            _stor+=0.25;
            while (_stor>1.0) {
                _stor-=1.0;
            }
            _stor-=0.5;
            if (_stor<0) {
                _stor*=-1.0;
            }
            _stor-=0.25;
            _stor*=-4.0;
            signal = _stor;
            step();//always last
            return true;
        }
```

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.

```
                                                                     {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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.

```
                                                                     {
    _listeners.push_back(listener);
    return true;
}
```

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

Definition at line 51 of file AudioSettings.cpp.

```
                                              {
    return _set;
}
```

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

Definition at line 44 of file AudioSettings.cpp.

```
                                                 {
    return _nyquist;
}
```

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

Definition at line 30 of file AudioSettings.cpp.

```
                                                    {
    return _sampleRate;
}
```

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

Definition at line 33 of file AudioSettings.cpp.

```
                                                                                {
    if (!_set) {
        _set=true;
    }
    _sampleRate = value;
    _nyquist = _sampleRate*0.5;
    for (auto i:_listeners) {
        i->SampleRateChanged(_sampleRate);
    }
    return _sampleRate;
}
```

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:

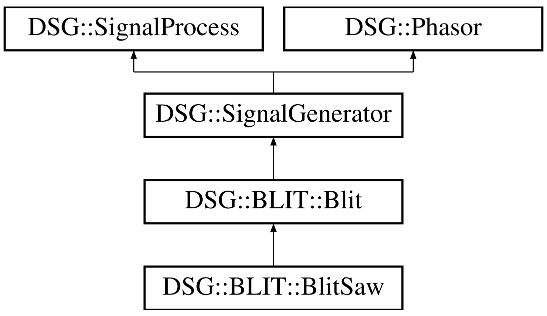
- AudioSettings.h
- AudioSettings.cpp

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                                                                    :
     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.

```
                                                                        {
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            return _frequency;
        }
```

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.

```
                                                    {
            //found better results in this case with built in sine function. not performance wise but
algorithmically
            denominator = m_ * sin(_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            return true;
        }
```

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.

```
                                                     {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

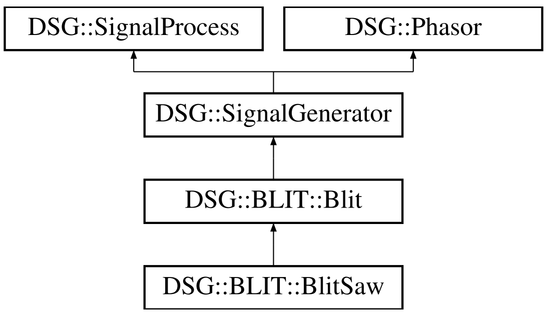
- BLIT.h
- BLIT.cpp

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.

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

9.6.2.2 DSG::BLIT::BlitSaw::BlitSaw (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 28 of file BLITSaw.cpp.

```
                                                                              :
      DSG::BLIT::Blit(frequency,offset),Register_(0){
    Frequency(frequency);
}
```

9.6.2.3 DSG::BLIT::BlitSaw::∼BlitSaw ( ) [virtual]

Definition at line 31 of file BLITSaw.cpp.

```
{}
```

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.

```
                                                                                         {
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            C2_ = 1.0/(double)p_;
            return _frequency;
        }
```

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.

```
                                                                {
            denominator = m_ * sin(PI*_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            signal += (Register_ - C2_);
            Register_ = signal * 0.995;
            C2_+=signal;
            C2_*=0.5;
            return true;
        }
```

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.

```
                                                                 {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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.

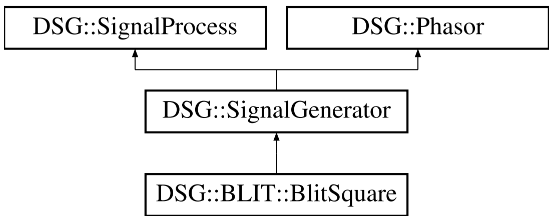
The documentation for this class was generated from the following files:

- 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.

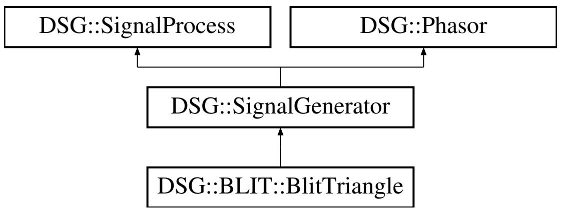
The documentation for this class was generated from the following file:

- 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:

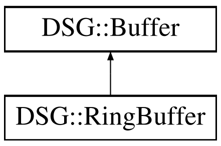
- BLITTriangle.h

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:

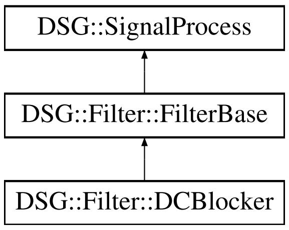
- Buffer.h
- Buffer.cpp

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.

```
                                                                          {
            x = signal;
            signal= x - xm1+ (_a * ym1);
            xm1 = x;
            ym1=signal;
            return true;
        }
```

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.

```
                                                                           {
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
```

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::xm1 [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:

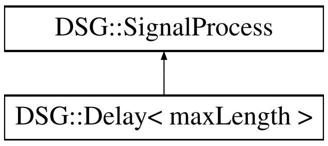
- DCBlocker.h
- DCBlocker.cpp

9.11 DSG::Delay< maxLength > Class Template Reference

DSG::Delay - General purpose delay line.

`#include <Delay.h>`

Inheritance diagram for DSG::Delay< maxLength >:



Public Member Functions

- Delay ()
- Delay (double const &samples)
- virtual ∼Delay ()
- virtual unsigned long const & Length () const
- virtual unsigned long const & Length (unsigned long const &samples)
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Member Functions

- virtual void increment ()

Protected Attributes

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

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.

```
                                                                                 {
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
            return _delay;
        }
```

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.

```
                                                                 {
        _swap = _buffer[_index-1];
        _buffer[_index-1]=signal;
        signal = _swap;
        increment();
        return true;
    }
```

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.

```
                                                                  {
        if (!signal.Empty()) {
            count = signal.Count();
            while (count-- > 0) {
                if(signal.Read(_temp)){
                    if (Perform(_temp)) {
                        signal.Write(_temp);
                    }else return false;
                }else return false;
            }return true;
        }else return false;
    }
```

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.

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

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.

```
                                                                                    {
            output = (signal - _delay)/(4.0 * dt);
            _delay = signal;
            return output;
        }
```

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.

```
                                                                                {
                    output  = (signal - _delay[0]);
                    output -= (_delay[0] - _delay[1]);
                    output /= (24.*DSG::Pow<2>(dt));
                    _delay[1]=_delay[0];
                    _delay[0]=signal;
                    return output;
                }
```

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.

```
                                                                        {
            output  = (signal - _delay[0]);
            output -= (_delay[0] - _delay[1]);
            output -= (_delay[1] - _delay[2]);
            output /= 144*DSG::Pow<3>(dt);
            _delay[2]=_delay[1];
            _delay[1]=_delay[0];
            _delay[0]=signal;
            return output;
        }
```

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.

```
                                                                                          {
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output /= 960*DSG::Pow<4>(dt);
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
```

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.

```
                                                                                                  {
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output -= (_delay[3] - _delay[4]);
                output /= 7200*DSG::Pow<5>(dt);
                _delay[4]=_delay[3];
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
```

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:

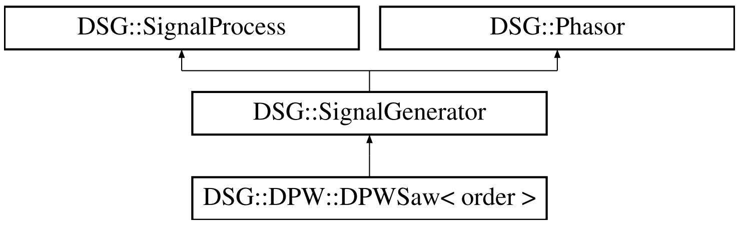
- DPW.h

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.

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

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.

```
{}
```

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.

```
                                                            {
                //trivial saw ramping from -1 to 1
                _register = _phasor;
                _register-=0.5;
                _register*=2.0;
                /*-------------------------*/
                //DPW algorithm
                //polynomial shaping
                _register=DSG::DPW::DPW_Polynomial<order>(_register);
                //differentiating
                signal = _diff(_register,_dt);
                /*-------------------------*/
                //signal = DSG::EnforceBounds<-1, 1>(signal);
                //advance phase
                step();
                return true;
            }
```

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.

```
                                                             {
                signal.Flush();
                while (!signal.Full()) {
                    if (Perform(_storage)) {
                        if(signal.Write(_storage)){
                        }else return false;
                    }else return false;
                }return true;
            }
```

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:

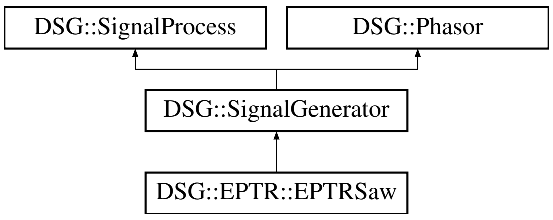
- DPWSaw.h

9.20 DSG::EPTR::EPTRSaw Class Reference

DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

```
#include <EPTRSaw.h>
```

Inheritance diagram for DSG::EPTR::EPTRSaw:



Public Member Functions

- EPTRSaw ()
- EPTRSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ∼EPTRSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample _register

Additional Inherited Members

9.20.1 Detailed Description

DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

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

Bug Algorithm is not performing in a band limited manor

Definition at line 36 of file EPTRSaw.h.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 DSG::EPTR::EPTRSaw::EPTRSaw ( )

Definition at line 25 of file EPTRSaw.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.

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

9.20.2.3 DSG::EPTR::EPTRSaw::∼EPTRSaw ( ) [virtual]

Definition at line 27 of file EPTRSaw.cpp.

```
{}
```

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.

```
                                                                    {
#ifdef __APPLE__
#warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
#endif
            //generate trivial saw
            _register = _phasor;
            _register+=0.5;
            if (_register>1.0) {
                --_register;
            }
            _register-=0.5;
            _register*=2.0;
            if (_register > 1.0-_dt) {
                //transition region detected
                //apply eptr correction
                signal = _register - (_register/_dt) + (1.0/
      _dt) -1;
            }else{
                signal = _register;
            }
            step();//avance phase
            return true;
        }
```

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.

```
                                                                     {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

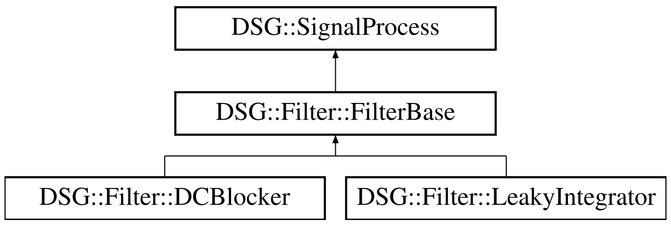
- DSGMath.h

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:

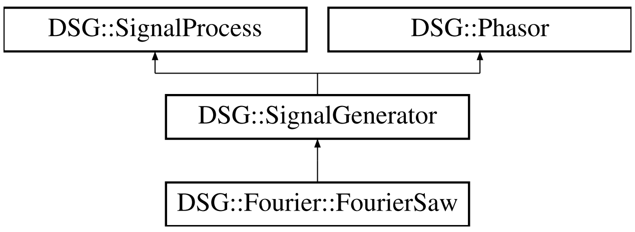
- Filter.h
- Filter.cpp

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.

```
                                                                      {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

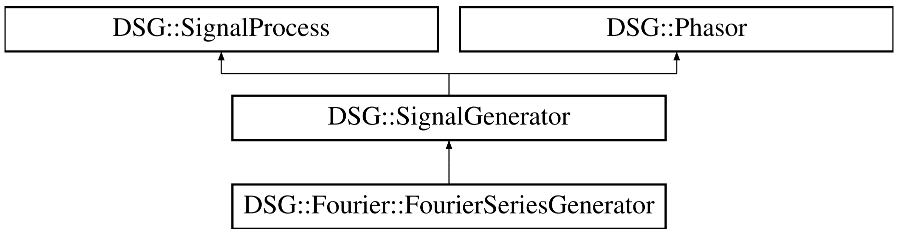
- FourierSaw.h
- FourierSaw.cpp

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.

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

9.25.3.3 DSG::Fourier::FourierSeriesGenerator::∼FourierSeriesGenerator ( ) [virtual]

Definition at line 47 of file FourierSeries.cpp.

```
{}
```

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.

```
                                                                                    {
            value = _phasor;
            signal=0;
            for (auto i = _series.begin(); i!=_series.end(); ++i) {
                signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
            }
            step();
            return true;
        }
```

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.

```
                                                                                     {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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

Definition at line 80 of file FourierSeries.h.

```

            {
            _series = series;
        }
```

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

Definition at line 83 of file FourierSeries.h.

```
                                                                                        {
                return _series;
        }
```

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:

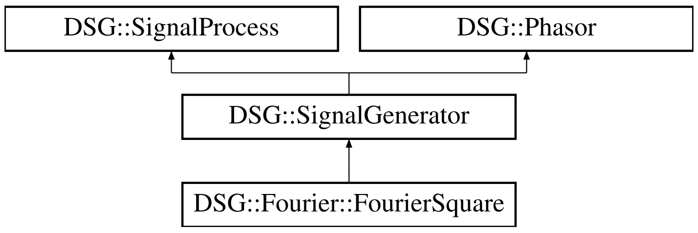
- FourierSeries.h
- FourierSeries.cpp

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.

```
: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.

```
                                                                                           :
DSG::SignalGenerator(frequency,offset),_a(3.6/PI),phs(0),
value(0),i(0){
    _h = MaxHarms(_frequency)+1;
}
```

9.26.2.3 DSG::Fourier::FourierSquare::∼FourierSquare ( ) [virtual]

Definition at line 29 of file FourierSquare.cpp.

```
{}
```

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.

```
                                                                                              {
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
```

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.

```
                                                                              {
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
```

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.

```
                                                                               {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

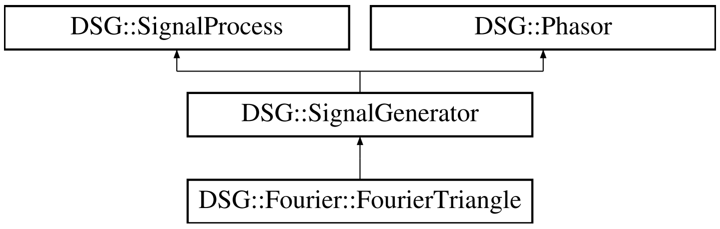
- FourierSquare.h
- FourierSquare.cpp

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.

```
                                                                                               :
      DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),
      phs(0),value(0),i(0){
    _h = MaxHarms(_frequency)+1;
}
```

9.27.2.3 DSG::Fourier::FourierTriangle::∼FourierTriangle ( ) [virtual]

Definition at line 29 of file FourierTriangle.cpp.

```
{}
```

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.

```
                                                                                                   {
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
```

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.

```
                                                                            {
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            double sgn = -1;
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*
      i);
                sgn*=-1;
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
```

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.

```
                                                                                    {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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:

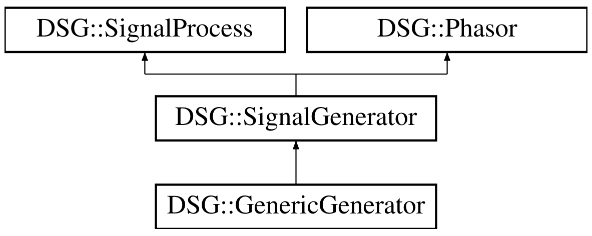
- FourierTriangle.h
- FourierTriangle.cpp

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::DSG↩
Sample(∗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.

```
: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.

```
:DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
```

9.28.2.3 DSG::GenericGenerator::∼GenericGenerator ( ) [virtual]

Definition at line 27 of file GenericGenerator.cpp.

```
{}
```

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.

```
                                                            {
        if (_callback!=nullptr) {
            signal = _callback(_phasor);
        }else signal = 0;
        step();
        return true;
    }
```

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.

```
                                                        {
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
```

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:

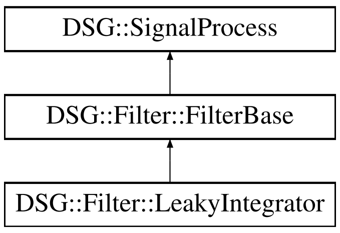
- FourierSeries.h
- FourierSeries.cpp

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.

```
                                               :DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

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

Definition at line 32 of file Leaky.cpp.

```
                                                                          :
      DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
    Cutoff(cutoff);
}
```

9.30.2.3 DSG::Filter::LeakyIntegrator::∼LeakyIntegrator ( ) [virtual]

Definition at line 40 of file Leaky.cpp.

```
                                            {
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

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.

```
                                                                                 {
            double Omega;
            x1 = y1 = 0.0;
            Omega = atan(PI * cutoff);
            a = -(1.0 - Omega) / (1.0 + Omega);
            b = (1.0 - b) / 2.0;
            return true;
        }
```

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.

```
                                                                              {
            y = b * (signal + x1) - a * y1;
            x1=signal;
            y1=y;
            signal=y;
            return true;
        }
```

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.

```
                                                                               {
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
```

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 interploation 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.

```
                                                    :_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/(double)_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
```

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.

```
                                                            :_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
```

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

Definition at line 63 of file LUT.h.

```
{}
```

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.

```
                                                        {
            phs=x;
            //need range checking on x to ensure 0-1 range
            phs<0 ? phs = 1-(phs*-1):0;
            phs-=((int)phs);
            return this->_table[(unsigned)(phs*(this->_size-1))];
        }
```

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:

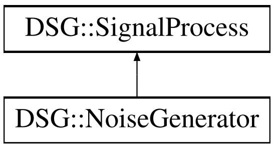
- LUT.h

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.

```
                                                                                  :
     DSG::SignalProcess(){
    _function = StatelessFunction;
}
```

9.32.2.2 DSG::NoiseGenerator::∼NoiseGenerator ( ) [virtual]

Definition at line 28 of file NoiseGenerator.cpp.

```
{}
```

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.

```
                                                            {
        signal = _function(0);
        return true;
    }
```

9.32.3.2 bool DSG::NoiseGenerator::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Definition at line 43 of file NoiseGenerator.h.

```
                                                                            {
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
```

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:

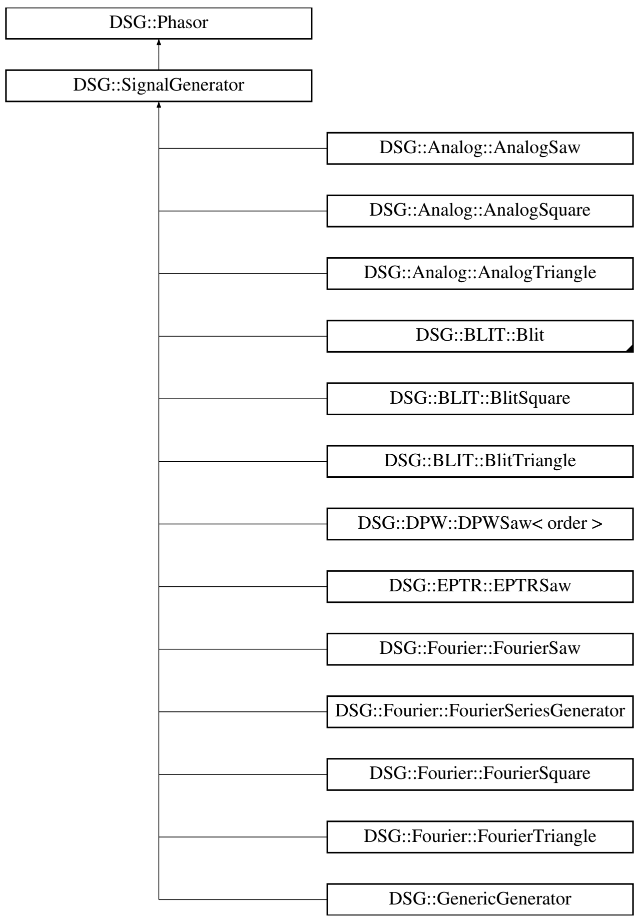
- NoiseGenerator.h
- NoiseGenerator.cpp

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                                                                                :
      _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:

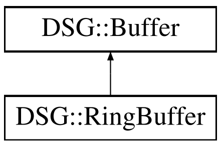
- Phasor.h
- Phasor.cpp

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.

```
                                        {
        return _count==0;
    }
```

9.34.3.3 void DSG::RingBuffer::Flush ( ) [inline]

Definition at line 83 of file RingBuffer.h.

```
                                   {
        _write.store(0,std::memory_order_relaxed);
        _read.store(0,std::memory_order_relaxed);
        _count=0;
    }
```

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

Definition at line 77 of file RingBuffer.h.

```
                                       {
        return _count==this->_size;
    }
```

9.34.3.5 size_t DSG::RingBuffer::make_pow_2 (size_t *number*) [inline], [protected]

Definition at line 111 of file RingBuffer.h.

```
                                                       {
        return pow(2, ceil(log(number)/log(2)));
    }
```

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

Definition at line 110 of file RingBuffer.h.

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

9.34.3.7 DSG::RingBuffer & DSG::RingBuffer::operator= (RingBuffer & *buffer*)

Definition at line 35 of file RingBuffer.cpp.

```
                                                          {
    Buffer::operator=(buffer);
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
    return *this;
}
```

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:

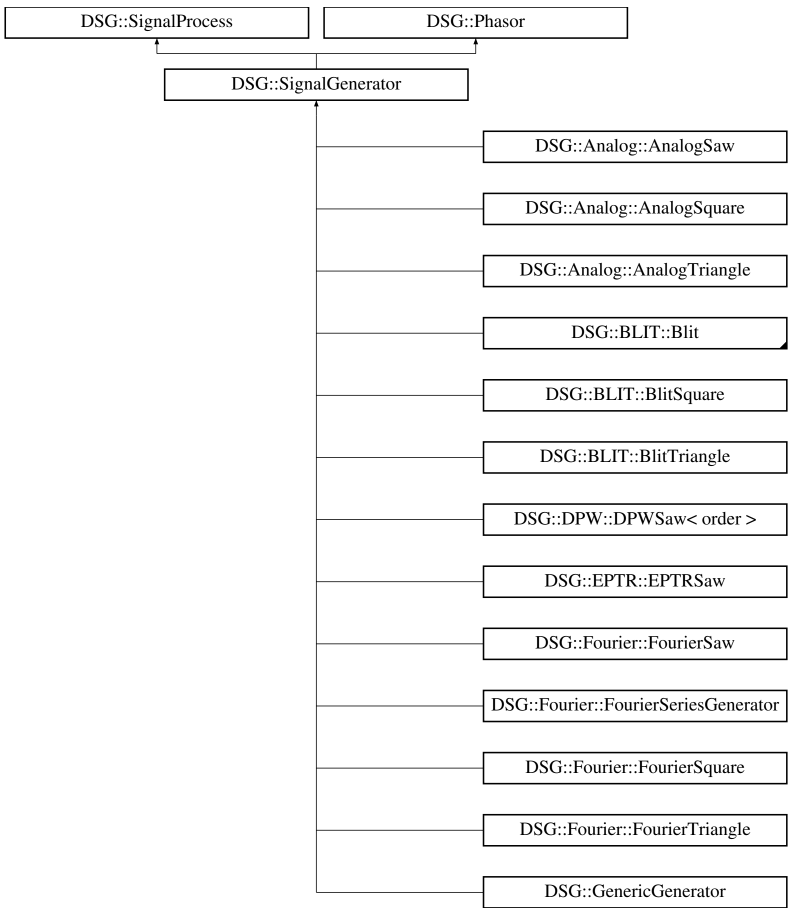
- RingBuffer.h
- RingBuffer.cpp

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 >, DS←G::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, D←SG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, and DSG::GenericGenerator.

Definition at line 49 of file SignalGenerator.h.

```
                                                            {
    signal=0;
    return false;
}
```

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.

```
                                                             {
    signal.Flush();
    return false;
}
```

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.

```
                                                                                  {
    Frequency(_frequency);
    return true;
}
```

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:

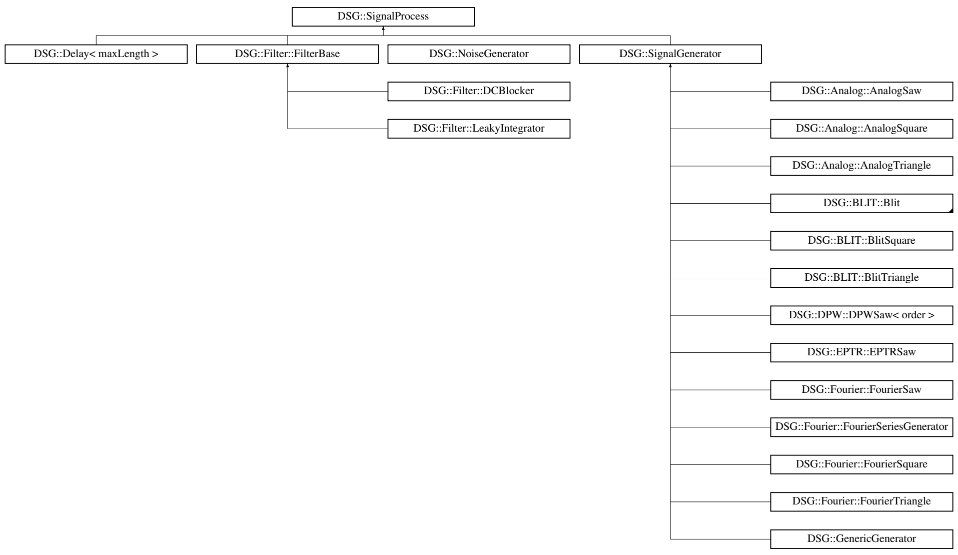
- SignalGenerator.h
- SignalGenerator.cpp

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::D↩PW::DPWSaw< order >, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, D↩SG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DS↩G::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::FourierSeries↩Generator, DSG::BLIT::Blit, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, D↩SG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DS↩G::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

```
//
//  AnalogSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "AnalogSaw.h"
DSG::Analog::AnalogSaw::AnalogSaw():DSG::
SignalGenerator(){}
DSG::Analog::AnalogSaw::AnalogSaw(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
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

```
//
//  AnalogSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__AnalogSaw__
#define __DSG__AnalogSaw__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator
        class AnalogSaw : public DSG::SignalGenerator {
        public:
            AnalogSaw();
            AnalogSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~AnalogSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _stor;
        };
        inline bool DSG::Analog::AnalogSaw::Perform(
DSG::DSGSample& signal){
            _stor=_phasor;
            _stor+=0.5;
            if (_stor>1.0) {
                --_stor;
            }
            _stor-=0.5;
            _stor*=2.0;
            signal=_stor;
            step();
            return true;
        }
        inline bool DSG::Analog::AnalogSaw::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogSaw__) */
```

10.5 AnalogSquare.cpp File Reference

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

10.6 AnalogSquare.cpp

```
//
//  AnalogSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "AnalogSquare.h"
DSG::Analog::AnalogSquare::AnalogSquare():
DSG::SignalGenerator(){}
DSG::Analog::AnalogSquare::AnalogSquare(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Analog::AnalogSquare::~AnalogSquare(){}
```

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

```
//
//  AnalogSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__AnalogSquare__
#define __DSG__AnalogSquare__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator
        class AnalogSquare : public DSG::SignalGenerator {
        public:
            AnalogSquare();
            AnalogSquare(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~AnalogSquare();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        };
        inline bool DSG::Analog::AnalogSquare::Perform(
DSG::DSGSample& signal){
            signal=_phasor < 0.5 ? 1.0:-1.0;
            step();
            return true;
        }
        inline bool DSG::Analog::AnalogSquare::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogSquare__) */
```

10.9 AnalogTriangle.cpp File Reference

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

10.10 AnalogTriangle.cpp

```
//
//  AnalogTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "AnalogTriangle.h"
DSG::Analog::AnalogTriangle::AnalogTriangle():
DSG::SignalGenerator(){}
DSG::Analog::AnalogTriangle::AnalogTriangle(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Analog::AnalogTriangle::~AnalogTriangle(){}
```

10.11 AnalogTriangle.h File Reference

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

Classes

- class DSG::Analog::AnalogTriangle

  DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.

Namespaces

- DSG

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

  DSG::Analog - Namespace Containing Analog Style Oscillators.

10.12 AnalogTriangle.h

```
//
//  AnalogTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__AnalogTriangle__
#define __DSG__AnalogTriangle__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator
        class AnalogTriangle : public DSG::SignalGenerator {
        public:
            AnalogTriangle();
            AnalogTriangle(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~AnalogTriangle();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _stor;
        };
        inline bool DSG::Analog::AnalogTriangle::Perform(
DSG::DSGSample& signal){
            _stor = _phasor;
            _stor+=0.25;
            while (_stor>1.0) {
                _stor-=1.0;
            }
            _stor-=0.5;
            if (_stor<0) {
                _stor*=-1.0;
            }
            _stor-=0.25;
            _stor*=-4.0;
            signal = _stor;
            step();//always last
            return true;
        }
        inline bool DSG::Analog::AnalogTriangle::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogTriangle__) */
```

10.13 AudioSettings.cpp File Reference

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

10.14 AudioSettings.cpp

```
//
//  AudioSettings.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "AudioSettings.h"
#include "SignalProcess.h"
DSG::DSGFrequency DSG::AudioSettings::_sampleRate;
DSG::DSGFrequency DSG::AudioSettings::_nyquist;
```

```
bool DSG::AudioSettings::_set=false;
std::vector<DSG::SignalProcess*> DSG::AudioSettings::_listeners;
DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(){
    return _sampleRate;
}
DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(
DSG::DSGFrequency const& value){
    if (!_set) {
        _set=true;
    }
    _sampleRate = value;
    _nyquist = _sampleRate*0.5;
    for (auto i:_listeners) {
        i->SampleRateChanged(_sampleRate);
    }
    return _sampleRate;
}
DSG::DSGFrequency const& DSG::AudioSettings::Nyquist(){
    return _nyquist;
}
bool DSG::AudioSettings::AddSampleRateListener(
DSG::SignalProcess* listener){
    _listeners.push_back(listener);
    return true;
}
bool const& DSG::AudioSettings::IsSampleRateSet(){
    return _set;
}
```

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 }
```

```
#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

```
//
//  Blackman.h
//  DSG
//
//  Created by Alexander Zywicki on 9/24/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Blackman_h
#define DSG_Blackman_h
#include "PI.h"
#include "LUT.h"
#include "Sine.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Window{
        //!\brief DSG::Window::Blackman - Blackman Window Function
        template<typename decimal>
        inline decimal Blackman(decimal const& x){
            // Generate Blackman Window
            /*
             Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
             }*/
            static_assert(std::is_floating_point<decimal>::value==true,"DSG::Blackman Function Requires
 Floating Point Type");
            //we will implement the blackman window as a function as if it were sin(x)
            //cos input domain 0-1 not 0-2pi
            //range checking is handles within DSG::Cos
            decimal phs=x;
            while (phs>1.0) {
                phs-=1.0;
            }
            return 0.42 - (0.5 * DSG::Cos(phs))+(0.08 * DSG::Cos(2.0*phs));
        }
    }
}
#endif
```

10.19 BLIT.cpp File Reference

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

10.20 BLIT.cpp

```
//
//  BLIT.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "BLIT.h"
DSG::BLIT::Blit::Blit():DSG::SignalGenerator(){
    Frequency(0);
}
DSG::BLIT::Blit::Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){
    Frequency(frequency);
}
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

```
//
//  BLIT.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__BLIT__
#define __DSG__BLIT__
#include "SignalGenerator.h"
#include "Denormal.h"
#include "Sinc.h"
#include "DSGMath.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::BLIT - Namespace Containing BLIT Based Oscillators
    namespace BLIT{
        /*!\brief DSG::BLIT::Blit - Band-Limited Impulse Train Generator
         */
        //!\todo Re-write DSG::BLIT::Blit algorithm
        class Blit:public DSG::SignalGenerator{
        public:
            Blit();
            Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~Blit();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long p_;
            unsigned long m_;
            unsigned long _h;
            double a_;
            DSG::DSGSample denominator;
            DSG::DSGSample value;
        };
        inline bool DSG::BLIT::Blit::Perform(
DSG::DSGSample& signal){
            //found better results in this case with built in sine function. not performance wise but
 algorithmically
            denominator = m_ * sin(_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            return true;
        }
        inline bool DSG::BLIT::Blit::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::BLIT::Blit::Frequency(DSG::DSGFrequency const& value){
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__BLIT__) */
```

10.23 BLITSaw.cpp File Reference

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

10.24 BLITSaw.cpp

```
//
//  BLITSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "BLITSaw.h"
DSG::BLIT::BlitSaw::BlitSaw():DSG::BLIT::Blit(),Register_(0){
    Frequency(0);
}
DSG::BLIT::BlitSaw::BlitSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::BLIT::Blit(frequency,offset),Register_(0){
    Frequency(frequency);
}
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

```
//
//  BLITSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__BLITSaw__
#define __DSG__BLITSaw__
#include "BLIT.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\brief DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm
        //!\todo Re-write DSG::BLIT::BlitSaw algorithm
        class BlitSaw : public Blit{
        public:
            BlitSaw();
            BlitSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~BlitSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            DSG::DSGSample C2_;
            DSG::DSGSample Register_;
        };
        inline bool DSG::BLIT::BlitSaw::Perform(
DSG::DSGSample& signal){
            denominator = m_ * sin(PI*_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            signal += (Register_ - C2_);
            Register_ = signal * 0.995;
            C2_+=signal;
            C2_*=0.5;
            return true;
        }
        inline bool DSG::BLIT::BlitSaw::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::BLIT::BlitSaw::Frequency(DSG::DSGFrequency const& value)
{
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            C2_ = 1.0/(double)p_;
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__BLITSaw__) */
```

10.27 BLITSquare.cpp File Reference

`#include "BLITSquare.h"`

10.28 BLITSquare.cpp

```
//
//  BLITSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#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

```
//
//  BLITSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__BLITSquare__
#define __DSG__BLITSquare__
#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\todo Write DSG::BLIT::BlitSquare algorithm
        class BlitSquare:public DSG::SignalGenerator{};
    }
}
#endif /* defined(__DSG__BLITSquare__) */
```

10.31 BLITTriangle.cpp File Reference

`#include "BLITTriangle.h"`

10.32 BLITTriangle.cpp

```
//
//  BLITTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#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

```
//
//  BLITTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__BLITTriangle__
#define __DSG__BLITTriangle__

#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\todo Write DSG::BLIT::BlitTriangle algorithm
        class BlitTriangle:public DSG::SignalGenerator{};
    }
}
#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

```
//
//  Bounds.h
//  DSG
//
//  Created by Alexander Zywicki on 11/11/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Bounds_h
#define DSG_Bounds_h
#include <assert.h>
namespace DSG{
    //!\brief DSG::EnforceBounds - Clip value to set bounds
    template<int lower,int upper,typename decimal>
    decimal EnforceBounds(decimal const& value){
        if (value<lower) {
            return lower;
        }else if(value> upper){
            return upper;
        }else return value;
    }
    //!\brief DSG::StaticAssertBounds - Fails on compile time if value is not within bounds
    template<int lower,int upper,int value>
    void StaticAssertBounds(){
        static_assert(value>=lower && value<=upper,"Failed Static Bounds Assert");
    }
    //!\brief DSG::AssertBounds - Fails on runtime if value is not within bounds
    template<int lower,int upper,typename T>
    void AssertBounds(T const& value){
        assert(value>=lower && value<=upper);
    }
}
#endif
```

10.37 Buffer.cpp File Reference

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

10.38 Buffer.cpp

```
//
//  Buffer.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "Buffer.h"
DSG::Buffer::Buffer():_size(0),_buffer(nullptr){}
DSG::Buffer::Buffer(size_t size):_size(size),_buffer(new
DSG::DSGSample[size]){}
DSG::Buffer::Buffer(Buffer const& other) {
    _buffer = new  DSG::DSGSample[_size];
    _size = other._size;
    *this = other;
}
DSG::Buffer& DSG::Buffer::operator=(Buffer const& other){
    if (_size!=other._size) {
        if (_buffer!=nullptr) {
            delete [] _buffer;
        }
        _size = other._size;
        _buffer = new  DSG::DSGSample[_size];
    }
    for (int i=0; i<_size; ++i) {
        _buffer[i] = other._buffer[i];
    }
    return *this;
}
DSG::Buffer::~Buffer(){
    if (_buffer!=nullptr) {
        delete [] _buffer;
    }
}
DSG::DSGSample& DSG::Buffer::operator[](size_t const& index){
#ifdef DEBUG
    assert(index<_size);
#endif
    return _buffer[index];
}
```

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

```
//
//  Buffer.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Buffer__
#define __DSG__Buffer__
#include <stddef.h>
#include "DSGTypes.h"
#ifdef DEBUG
#include <assert.h>
#endif
namespace DSG{
    /*!\brief DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use
     */
    class Buffer {
    public:
        Buffer();
        Buffer(size_t size);
        Buffer(Buffer const& other);
        Buffer& operator=(Buffer const& other);
        virtual ~Buffer();
        DSG::DSGSample& operator[](size_t const& index);
        inline size_t const& Size()const;
    protected:
        DSG::DSGSample* _buffer;
        size_t _size;
    };
    inline size_t const& DSG::Buffer::Size()const{
        return _size;
    }
}
#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

```
//
//  BufferConversion.h
//  DSG
//
//  Created by Alexander Zywicki on 10/14/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__BufferConversion__
#define __DSG__BufferConversion__
#include "RingBuffer.h"
namespace DSG {
    //!\brief DSG::RingToArray - Move Ring Buffer data to an array
    inline bool RingToArray(DSG::RingBuffer& ring,
DSG::DSGSample* array,unsigned long length){
        for (int i=0; i<length; ++i) {
            if (!ring.Empty()) {
                ring.Read(array[i]);
            }
        }return true;
    }
    //!\brief DSG::ArrayToRing - Move array data to a Ring Buffer
    inline bool ArrayToRing(DSG::RingBuffer& ring,
DSG::DSGSample* array, unsigned long length){
        int i=0;
        ring.Flush();
        while (!ring.Full()) {
            ring.Write(array[i]);
            ++i;
        }return true;
    }
}
#endif /* defined(__DSG__BufferConversion__) */
```

10.43 DCBlocker.cpp File Reference

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

10.44 DCBlocker.cpp

```
//
//  DCBlocker.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/13/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "DCBlocker.h"
DSG::Filter::DCBlocker::DCBlocker():DSG::Filter::
FilterBase(),_a(0.995),xm1(0),ym1(0),x(0),_temp(0){}
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

```
//
//  DCBlocker.h
//  DSG
//
//  Created by Alexander Zywicki on 10/13/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
*/
#ifndef __DSG__DCBlocker__
#define __DSG__DCBlocker__
#include "Filter.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Filter{
        //!\brief DSG::Filter::DCBlocker - DC blocking filter
        class DCBlocker:public DSG::Filter::FilterBase {
        public:
            DCBlocker();
            virtual ~DCBlocker();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            unsigned long count;
            DSG::DSGSample _temp;
            DSG::DSGSample xm1;
            DSG::DSGSample ym1;
            DSG::DSGSample x;
            DSG::DSGSample _a;
        };
        inline bool DSG::Filter::DCBlocker::Perform(
DSG::DSGSample& signal){
            x = signal;
            signal= x - xm1+ (_a * ym1);
            xm1 = x;
            ym1=signal;
            return true;
        }
        inline bool DSG::Filter::DCBlocker::Perform(
DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
    }
}
#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

```
//
//  Delay.h
//  DSG
//
//  Created by Alexander Zywicki on 10/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Delay__
#define __DSG__Delay__
#include "DSGTypes.h"
#include "SignalProcess.h"
#include "Interpolate.h"
#include "AudioSettings.h"
namespace DSG{
    //!\brief DSG::Delay - General purpose delay line
    template<unsigned long maxLength>
    class Delay:public DSG::SignalProcess{
    public:
        Delay():DSG::SignalProcess(),_max(maxLength),
_swap(0),_temp(0),count(0),_index(0),_delay(0){
            for (int i=0; i<_max; ++i) {
                _buffer[i]=0;
            }
        }
        Delay(double const& samples):DSG::SignalProcess(),
_max(maxLength),_swap(0),_temp(0),count(0),_index(0),
_delay(0){
            for (int i=0; i<_max; ++i) {
                _buffer[i]=0;
            }
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
        }
        virtual ~Delay(){}
        virtual inline unsigned long const& Length()const{
            return _delay;
        }
        virtual inline unsigned long const& Length(unsigned long const& samples){
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
            return _delay;
        }
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        unsigned long count;
        unsigned long  _delay;
        unsigned long _index;
        const unsigned long _max;
        DSG::DSGSample _buffer[maxLength];
        DSG::DSGSample _swap;
        DSG::DSGSample _temp;
        virtual inline void increment(){
            ++_index;
            if (_index>_delay) {
                _index-=_delay;
            }
        }
    };
    template<unsigned long maxLength>
    inline bool DSG::Delay<maxLength>::Perform(
DSG::DSGSample& signal){
        _swap = _buffer[_index-1];
        _buffer[_index-1]=signal;
        signal = _swap;
        increment();
        return true;
    }
    template<unsigned long maxLength>
    inline bool DSG::Delay<maxLength>::Perform(
DSG::RingBuffer& signal){
        if (!signal.Empty()) {
            count = signal.Count();
            while (count-- > 0) {
                if(signal.Read(_temp)){
                    if (Perform(_temp)) {
                        signal.Write(_temp);
                    }else return false;
                }else return false;
            }return true;
        }else return false;
    }
}
#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

```
//
//  Denormal.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Denormal_h
#define DSG_Denormal_h
#include <limits>
#include "DSGMath.h"
```

```
namespace DSG{
    //!\brief DSG::IsDenormal - Returns True if number is Denormal
    template<typename T>
    inline bool IsDenormal(T const& value){
        return DSG::Abs(value)<=std::numeric_limits<T>::epsilon();//return true if number is
denormal
    }
}
#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 - Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 1 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 2 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm.

- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 3 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 4 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 5 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 6 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm.

10.52 DPW.h

```
//
//  DPW.h
//  DSG
//
//  Created by Alexander Zywicki on 11/11/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_DPW_h
#define DSG_DPW_h
#include "DSGTypes.h"
#include "DSGMath.h"
#include "SignalGenerator.h"
#include "Bounds.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::DPW - Generators using the DPW method
    namespace DPW{
        //!\brief DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm
        template<unsigned order>
        inline DSG::DSGSample DPW_Polynomial(
DSG::DSGSample const& value){
            DSG::StaticAssertBounds<1,6,order>();//must be 1-6 order
            return value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<1>(
DSG::DSGSample const& value){
            return value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<2>(
DSG::DSGSample const& value){
            return DSG::Pow<2>(value);
        }
        //!\brief DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<3>(
DSG::DSGSample const& value){
            return DSG::Pow<3>(value)-value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<4>(DSG::DSGSample const& value){
            return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
        }
        //!\brief DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<5>(DSG::DSGSample const& value){
            return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<6>(DSG::DSGSample const& value){
            return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 * DPW_Polynomial<2>(value);
        }
#ifdef __APPLE__
#warning DSG::DPW - differentiators order 4-6 need verification. they cause major clipping
#endif
        //!\todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 4-6
        //differentiators
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm
        template<unsigned order>
        class DPW_Differentiator{
        public:
            DPW_Differentiator(){
                DSG::StaticAssertBounds<1, 6,order>();//order must be 1-6
            }
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm
        template<>
        class DPW_Differentiator<1>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                return signal;
            }
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm
        template<>
        class DPW_Differentiator<2>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output = (signal - _delay)/(4.0 * dt);
                _delay = signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay;
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW Algorithm
        template<>
        class DPW_Differentiator<3>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output /= (24.*DSG::Pow<2>(dt));
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[2];
        };
        /*!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW Algorithm
         * \bug Causes major clipping
         */
        template<>
        class DPW_Differentiator<4>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output /= 144*DSG::Pow<3>(dt);
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[3];
        };
        /*!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW
 Algorithm
         * \bug Causes major clipping
         */
        template<>
        class DPW_Differentiator<5>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output /= 960*DSG::Pow<4>(dt);
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[4];
        };
        /*!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW
 Algorithm
         * \bug Causes major clipping
         */
        template<>
        class DPW_Differentiator<6>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output -= (_delay[3] - _delay[4]);
                output /= 7200*DSG::Pow<5>(dt);
                _delay[4]=_delay[3];
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[5];
        };
    }
}
#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

```
//
//  DPWSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__DPWSaw__
#define __DSG__DPWSaw__
#include "DPW.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace DPW{
        //!\brief DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm
        template<unsigned order>
        class DPWSaw:public DSG::SignalGenerator{
        public:
            DPWSaw():DSG::SignalGenerator(),_register(0){
                DSG::StaticAssertBounds<1, 6,order>();
            }
            DPWSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset),
_register(0){DSG::StaticAssertBounds<1, 6,order>();}
            virtual ~DPWSaw(){}
            virtual inline bool Perform(DSG::DSGSample& signal){
                //trivial saw ramping from -1 to 1
                _register = _phasor;
                _register-=0.5;
                _register*=2.0;
                /*-------------------------*/
                //DPW algorithm
                //polynomial shaping
                _register=DSG::DPW::DPW_Polynomial<order>(_register);
                //differentiating
                signal = _diff(_register,_dt);
                /*-------------------------*/
                //signal = DSG::EnforceBounds<-1, 1>(signal);
                //advance phase
                step();
                return true;
            }
            virtual inline bool Perform(DSG::RingBuffer& signal){
                signal.Flush();
                while (!signal.Full()) {
                    if (Perform(_storage)) {
                        if(signal.Write(_storage)){
                        }else return false;
                    }else return false;
                }return true;
            }
        protected:
            DSG::DSGSample _register;
            DSG::DPW::DPW_Differentiator<order>
_diff;
        };
    }
}
#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.

```
                             {
    DSG::DSGSample* _out = (DSG::DSGSample*)output;
    DSG:: DSGSample _sample;
    DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
    if (_out!=nullptr) {
        _buffer.Flush();
        _osc->Perform(_buffer);
        for (int i=0; i<frameCount; ++i) {
            _buffer.Read(_sample);
            *_out++ = _sample;
            *_out++ = _sample;
        }
    }
    return 0;
}
```

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 //
```

```
//  Driver.cpp
//  Waveform
//
//  Created by Alexander Zywicki on 8/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
#include "Driver.h"
PaStream* stream;
#define BufferSize 512
DSG:: RingBuffer _buffer(BufferSize);
int DriverInit(void * data){
    PaError err=0;

    err=Pa_Initialize();
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
BufferSize, Callback, data);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_StartStream(stream);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    return 0;
}
int DriverExit(){
    PaError err=0;
    err = Pa_StopStream(stream);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_CloseStream( stream );
    if( err != paNoError ){
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
    }
    err = Pa_Terminate();
    if( err != paNoError ){
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
    }
    return 0;
}
int Callback(const void *input,
             void *output,
             unsigned long frameCount,
             const PaStreamCallbackTimeInfo* timeInfo,
             PaStreamCallbackFlags statusFlags,
             void *userData) {
    DSG::DSGSample* _out = (DSG::DSGSample*)output;
    DSG:: DSGSample _sample;
    DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
    if (_out!=nullptr) {
        _buffer.Flush();
        _osc->Perform(_buffer);
        for (int i=0; i<frameCount; ++i) {
            _buffer.Read(_sample);
            *_out++ = _sample;
            *_out++ = _sample;
        }
    }
    return 0;
}
```

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.

```
                            {
    PaError err=0;

    err=Pa_Initialize();
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
BufferSize, Callback, data);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_StartStream(stream);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    return 0;
}
```

10.58 Driver.h

```
//
//  Driver.h
//  Waveform
//
//  Created by Alexander Zywicki on 8/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
#ifndef __Waveform__Driver__
#define __Waveform__Driver__
#ifdef DEBUG
#include <iostream>
#endif
#include <portaudio.h>
#include "DSG.h"
int DriverInit(void * data);
int DriverExit();
int Callback( const void *input,
             void *output,
             unsigned long frameCount,
             const PaStreamCallbackTimeInfo* timeInfo,
             PaStreamCallbackFlags statusFlags,
             void *userData );
#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

```
//
//  DSF.h
//  DSG
//
//  Created by Alexander Zywicki on 11/5/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__DSF__
#define __DSG__DSF__
#include "DSGMath.h"
#include "DSGTypes.h"
namespace DSG{
    template<typename decimal=DSG::DSGSample>
    decimal DSF(decimal const& beta,decimal const& theta,decimal const& N,decimal const& a){
#ifdef __APPLE__
#warning Untested DSG::DSF()
#endif
        decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
        decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
sin(theta + (N*beta)));
        return numerator/denominator;
    }
}
#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

```
//
//  DSG.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
#ifndef DSG_DSG_h
#define DSG_DSG_h
/*!\mainpage DSG - A Collection Of Tools For Digital Signal Generation
 *\section intro_sec Intoduction
 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.
 *\subsection Scope
 Though DSG has a focus on Bandlimited Waveform Generation it is not limited to it.
 DSG defines a signal processing interface that is compatable with any form fo 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 doumentation for each for their specifics.
 *\section 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 distrobution 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/>.

 */
#define DSG_Short_Names // enables inlining of nested namespaces to allow shorter explicit typenames
//Example: DSG::Analog::AnalogSaw (Long Name)...DSG::AnalogSaw (Short Name)(only available with this macro
 enabled
//!\brief DSG - A Collection of tools for Digital Signal Generation
//!\copyright Lesser GNU Public License
//!\todo Increase documentation level. Add documentation for every variable, parameter...
//!\todo Implement Blep Based Algorithms
namespace DSG {}
#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"
#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

```
//
//  Math.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Math_h
#define DSG_Math_h
#include <math.h>
#include <type_traits>
namespace DSG {
    //!\brief DSG::Abs - Calculate absolute value
    template<typename T>
    inline T Abs(T const& value){
        return value < 0.0 ? -1.0 * value : value;
    }
    //!\brief DSG::Factorial - Compute integer factorial
    template<unsigned long N>
    struct Factorial{
        enum {value = N * Factorial<N-1>::value};
    };
    //!\brief DSG::Factorial - Compute integer factorial
    template<>
    struct Factorial<0>{
        enum{ value = 1 };
    };
    namespace{
        template<class T, unsigned N>
        struct power{
            static constexpr T value(const T x){
                return power<T, N-1>::value(x) * x;
            }
        };
        template<class T>
        struct power<T, 0>{
            static constexpr T value(const T x){
                return 1;
            }
        };
    }
    //!\brief DSG::Pow - Any type to an integer power, i.e. N ^ I
    template<unsigned exponent, class T>
    T constexpr Pow(T const base){
        return power<T, exponent>::value(base);
    }
}
#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

```
//
//  DSGTypes.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_DSGTypes_h
#define DSG_DSGTypes_h
namespace DSG {
    //!\typedef DSG::DSGFrequency - Type for representing a frequency value
    //!\brief DSG::DSGFrequency - Type for representing a frequency value
    typedef float DSGFrequency;
    //!\typedef DSG::DSGPhase - Type for representing a phase value
    //!\brief DSG::DSGPhase - Type for representing a phase value
    typedef float DSGPhase;
    //!\typedef DSG::DSGSample - Type for representing an audio sample
    //!\brief DSG::DSGSample - Type for representing an audio sample
    typedef float DSGSample;
}
#endif
```

10.67 EPTRSaw.cpp File Reference

```
#include "EPTRSaw.h"
```

10.68 EPTRSaw.cpp

```
//
//  EPTRSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/29/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "EPTRSaw.h"
DSG::EPTR::EPTRSaw::EPTRSaw():DSG::SignalGenerator(){}
DSG::EPTR::EPTRSaw::EPTRSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){}
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 Efficienct Polynomial Transfer Region Algorithm.

Namespaces

- DSG

DSG - A Collection of tools for Digital Signal Generation.

- DSG::EPTR

  DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.

10.70 EPTRSaw.h

```
//
//  EPTRSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/29/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__EPTRSaw__
#define __DSG__EPTRSaw__
#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm
    namespace EPTR{
        //!\brief DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer
 Region Algorithm
        //!\todo Test and Possibly Re-Write DSG::EPTR::EPTRSaw algorithm
        //!\bug Algorithm is not performing in a band limited manor
        class EPTRSaw : public DSG::SignalGenerator{
        public:
            EPTRSaw();
            EPTRSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
            virtual ~EPTRSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _register;
        };
        inline bool DSG::EPTR::EPTRSaw::Perform(
      DSG::DSGSample& signal){
#ifdef __APPLE__
#warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
#endif
            //generate trivial saw
            _register = _phasor;
            _register+=0.5;
            if (_register>1.0) {
                --_register;
            }
            _register-=0.5;
            _register*=2.0;
            if (_register > 1.0-_dt) {
                //transition region detected
                //apply eptr correction
                signal = _register - (_register/_dt) + (1.0/_dt) -1;
            }else{
                signal = _register;
            }
            step();//avance phase
            return true;
        }
        inline bool DSG::EPTR::EPTRSaw::Perform(
      DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__EPTRSaw__) */
```

10.71 Filter.cpp File Reference

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

10.72 Filter.cpp

```
//
//  Filter.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "Filter.h"
DSG::Filter::FilterBase::FilterBase():_temp(0),count(0){}
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

```
//
//  Filter.h
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Filter__
#define __DSG__Filter__
#include "SignalProcess.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::Filter - Filters
    namespace Filter{
        //!\brief DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency
        class FilterBase:public DSG::SignalProcess{
        public:
            FilterBase();
            virtual ~FilterBase();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
        protected:
            DSG::DSGSample _temp;
            unsigned long count;
        };
        inline bool DSG::Filter::FilterBase::Perform(
DSG::DSGSample& signal){
            return true;
        }
        inline bool DSG::Filter::FilterBase::Perform(
DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
        inline bool DSG::Filter::FilterBase::Cutoff(
DSG::DSGFrequency const& cutoff){
            return false;
        }
    }
}
#endif /* defined(__DSG__Filter__) */
```

10.75 FourierSaw.cpp File Reference

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

10.76 FourierSaw.cpp

```
//
//  FourierSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "FourierSaw.h"
DSG::Fourier::FourierSaw::FourierSaw():DSG::
SignalGenerator(),_a(1.7/PI),phs(0),value(0),i(0){}
DSG::Fourier::FourierSaw::FourierSaw(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),value(0),i(0){
    _h = MaxHarms(_frequency)+1;
}
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

```
//
//  FourierSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__FourierSaw__
#define __DSG__FourierSaw__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator
        class FourierSaw : public DSG::SignalGenerator {
        public:
            FourierSaw();
            FourierSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~FourierSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierSaw::Perform(
DSG::DSGSample& signal){
            //_h Sine Calls Per Sample where _h  is theoretically nyquist / frequency
            value=DSG::Sin(_phasor);
            for (i=2; i<_h; ++i) {
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSaw::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::Fourier::FourierSaw::Frequency(
DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__FourierSaw__) */
```

10.79 FourierSeries.cpp File Reference

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

10.80 FourierSeries.cpp

```
//
//  FourierSeries.cpp
//  DSG
//
//  Created by Alexander Zywicki on 11/18/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "FourierSeries.h"
DSG::Fourier::Harmonic::Harmonic():_ratio(0),_amplitude(0){}
DSG::Fourier::Harmonic::Harmonic(DSG::DSGSample const& ratio,
      DSG::DSGSample const& amplitude):_ratio(ratio),_amplitude(amplitude){}
DSG::Fourier::Harmonic::~Harmonic(){
    _ratio=0;
    _amplitude=0;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio()const{
    return _ratio;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio(
      DSG::DSGSample const& value){
    _ratio = value;
    return _ratio;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude()
      const{
    return _amplitude;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude(
      DSG::DSGSample const& value){
    _amplitude=value;
    return _amplitude;
}
DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
      ():DSG::SignalGenerator(){}
DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
      (DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset){}
DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator
      (){}
```

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

```
//
//  FourierSeries.h
//  DSG
//
//  Created by Alexander Zywicki on 11/18/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__FourierSeries__
#define __DSG__FourierSeries__
#include "SignalGenerator.h"
#include <vector>
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Fourier{
        //!\brief DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.
        class Harmonic{
        public:
            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:
            DSG::DSGSample _ratio;
            DSG::DSGSample _amplitude;
        };
        //!\brief DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified Fourier Series
        class FourierSeriesGenerator: public DSG::SignalGenerator{
        public:
            typedef std::vector<Harmonic> FourierSeries;
            FourierSeriesGenerator();
            FourierSeriesGenerator(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~FourierSeriesGenerator();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            inline void Series(FourierSeries const& series);
            inline FourierSeries& Series();
        protected:
            FourierSeries _series;
            DSG::DSGSample value;
        };
        inline bool DSG::Fourier::FourierSeriesGenerator::Perform(DSG::DSGSample& signal){
            value = _phasor;
            signal=0;
            for (auto i = _series.begin(); i!=_series.end(); ++i) {
                signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
            }
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSeriesGenerator::Perform(DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline void DSG::Fourier::FourierSeriesGenerator::Series(DSG::Fourier::FourierSeriesGenerator::FourierSeries const& series){
            _series = series;
        }
        inline DSG::Fourier::FourierSeriesGenerator::FourierSeries& DSG::Fourier::FourierSeriesGenerator::Series(){
            return _series;
        }
    }
}
#endif /* defined(__DSG__FourierSeries__) */
```

10.83 FourierSquare.cpp File Reference

`#include "FourierSquare.h"`

10.84 FourierSquare.cpp

```
//
//  FourierSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "FourierSquare.h"
DSG::Fourier::FourierSquare::FourierSquare():DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),i(0){}
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){
    _h = MaxHarms(_frequency)+1;
}
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

```
//
//  FourierSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__FourierSquare__
#define __DSG__FourierSquare__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator
        class FourierSquare : public DSG::SignalGenerator {
        public:
            FourierSquare();
            FourierSquare(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~FourierSquare();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierSquare::Perform(DSG::DSGSample& signal){
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSquare::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::Fourier::FourierSquare::Frequency(
DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__FourierSquare__) */
```

10.87 FourierTriangle.cpp File Reference

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

10.88 FourierTriangle.cpp

```
//
//  FourierTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "FourierTriangle.h"
DSG::Fourier::FourierTriangle::FourierTriangle():
DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),value(0),i(0){}
DSG::Fourier::FourierTriangle::FourierTriangle(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),phs(0),value(0),i(0){
    _h = MaxHarms(_frequency)+1;
}
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

```
//
//  FourierTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__FourierTriangle__
#define __DSG__FourierTriangle__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator
        class FourierTriangle : public DSG::SignalGenerator {
        public:
            FourierTriangle();
            FourierTriangle(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~FourierTriangle();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::DSGSample& signal){
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            double sgn = -1;
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*i);
                sgn*=-1;
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
      DSG::Fourier::FourierTriangle::Frequency(
      DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#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

```
//
//  Gaussian.h
//  DSG
//
//  Created by Alexander Zywicki on 10/6/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Gaussian_h
#define DSG_Gaussian_h
#include "Sine.h"
#include "White.h"
```

```
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::Gaussian - Gaussian Noise Generator Function
        template<typename decimal=DSG::DSGSample>
        decimal Gaussian(decimal=0.0){
            static decimal normalizer=1;//variable used to actively normalize the output
            //to enforce compatability with DSG::LUT a dummy parameter is applied
            //this parameter is useless except for compatability reasons
            decimal R1 = DSG::Noise::White();
            decimal R2 = DSG::Noise::White();
            decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
            if (DSG::Abs(x)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(x);
            }
            x/=normalizer;//normalize
            return x;
        }
    }
}
#endif
```

10.93 GenericGenerator.cpp File Reference

`#include "GenericGenerator.h"`

10.94 GenericGenerator.cpp

```
//
//  GenericGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "GenericGenerator.h"
DSG::GenericGenerator::GenericGenerator():
DSG::SignalGenerator(){}
DSG::GenericGenerator::GenericGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset,
DSG::DSGSample (*signalFunction)(DSG::DSGSample const&)):
DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
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

```
//
//  GenericGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__GenericGenerator__
#define __DSG__GenericGenerator__
#include "SignalGenerator.h"
namespace DSG{
    //!\brief DSG::GenericGenerator - Generator designed to use a stateless generator function such as
 DSG::Sin()
    class GenericGenerator:public DSG::SignalGenerator{
    public:
        GenericGenerator();
        GenericGenerator(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset,DSG::DSGSample (*signalFunction)(
DSG::DSGSample const&));
        virtual ~GenericGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        DSG::DSGSample (*_callback)(DSG::DSGSample const&);
    };
    inline bool DSG::GenericGenerator::Perform(
DSG::DSGSample& signal){
        if (_callback!=nullptr) {
            signal = _callback(_phasor);
        }else signal = 0;
        step();
        return true;
    }
    inline bool DSG::GenericGenerator::Perform(
DSG::RingBuffer& signal){
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
}
#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

```
//
//  Interpolate.h
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
//Code In this file was adapted from the code provided on this website
//http://paulbourke.net/miscellaneous/interpolation/
//
/*
 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/>.
 */
#ifndef DSG_Interpolate_h
#define DSG_Interpolate_h
#include "DSGMath.h"
#include "PI.h"
namespace DSG{
    //!\brief DSG::LinearInterpolate - Linear Interpolation
    template<typename decimal>
    decimal LinearInterpolate(decimal const& y1,decimal const& y2,decimal const& mu){
        return(y1*(1-mu)+y2*mu);
    }
    //!\brief DSG::CosineInterpolate - Cosine Interpolation
    template<typename decimal>
    decimal CosineInterpolate(
                                 decimal y1,decimal y2,
                                 decimal mu)
    {
        decimal mu2;
        mu2 = (1-cos(mu*PI))/2.0;
        return(y1*(1-mu2)+y2*mu2);
    }
    //!\brief DSG::CubicInterpolate - Cubic Interpolation
    template<typename decimal>
    decimal CubicInterpolate(decimal const& y0,decimal const& y1,
                                decimal const& y2,decimal const& y3,
                                decimal const& mu)
    {
        decimal a0,a1,a2,a3,mu2;
        mu2 = mu*mu;
        a0 = y3 - y2 - y0 + y1;
        a1 = y0 - y1 - a0;
        a2 = y2 - y0;
        a3 = y1;
        return(a0*mu*mu2+a1*mu2+a2*mu+a3);
    }
    //!\brief DSG::HermiteInterpolate - Hermite 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)
    {
        /*
         Tension: 1 is high, 0 normal, -1 is low
         Bias: 0 is even,
         positive is towards first segment,
         negative towards the other
         */
        decimal m0,m1,mu2,mu3;
        decimal a0,a1,a2,a3;
        mu2 = mu * mu;
        mu3 = mu2 * mu;
        m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
        m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
        m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
        m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
        a0 =  2*mu3 - 3*mu2 + 1;
        a1 =    mu3 - 2*mu2 + mu;
        a2 =    mu3 -   mu2;
        a3 = -2*mu3 + 3*mu2;
        return(a0*y1+a1*m0+a2*m1+a3*y2);
    }
}
#endif
```

10.99 Leaky.cpp File Reference

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

10.100 Leaky.cpp

```
//
//  Leaky.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "Leaky.h"
DSG::Filter::LeakyIntegrator::LeakyIntegrator():
DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
DSG::Filter::LeakyIntegrator::LeakyIntegrator(
DSG::DSGFrequency const& cutoff):DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
    Cutoff(cutoff);
}
DSG::Filter::LeakyIntegrator::~LeakyIntegrator(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

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

```
//
//  Leaky.h
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Leaky__
#define __DSG__Leaky__
#include "Filter.h"
#include "PI.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Filter{
        //!\brief DSG::Filter::LeakyIntegrator - Leaky integrator
        class LeakyIntegrator:public DSG::Filter::FilterBase{
        public:
            LeakyIntegrator();
            LeakyIntegrator(DSG::DSGFrequency const& cutoff);
            virtual ~LeakyIntegrator();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
        protected:
            double x1,y1,a,b;
            double y;
        };
        inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::DSGSample& signal){
            y = b * (signal + x1) - a * y1;
            x1=signal;
            y1=y;
            signal=y;
            return true;
        }
        inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
        inline bool DSG::Filter::LeakyIntegrator::Cutoff(
DSG::DSGFrequency const& cutoff){
            double Omega;
            x1 = y1 = 0.0;
            Omega = atan(PI * cutoff);
            a = -(1.0 - Omega) / (1.0 + Omega);
            b = (1.0 - b) / 2.0;
            return true;
        }
    }
}
#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

```
//
//  LUT.h
//  Waveform
//
//  Created by Alexander Zywicki on 8/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef Waveform_LUT_h
#define Waveform_LUT_h
#ifdef DEBUG
#include <assert.h>
#endif
#include "Interpolate.h"
namespace DSG{
    //!\brief DSG::LUT - Look Up Table
    //!\todo Implement interploation into lookup algorithm
    template <typename element,unsigned long size>
    class LUT {
    public:
        typedef element (*FillFunction)(element);
        typedef element (*FillFunctionConstRef)(element const&);
        LUT():_size(size){}
        LUT(FillFunction fill,double const& range = 1.0):_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/(double)_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
        LUT(FillFunctionConstRef fill,double const& range = 1.0):
_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
        ~LUT(){}
        element const& operator[](unsigned long const& index)const{
#ifdef DEBUG
            assert(index<_size);
#endif
            return _table[index];
        }
        element& operator[](unsigned long const& index){
#ifdef DEBUG
            assert(index<_size);
#endif
```

```
            return _table[index];
        }
        inline element const&  operator()(double const& x){
            phs=x;
            //need range checking on x to ensure 0-1 range
            phs<0 ? phs = 1-(phs*-1):0;
            phs-=((int)phs);
            return this->_table[(unsigned)(phs*(this->_size-1))];
        }
        unsigned long const& Size()const{
            return _size;
        }
    protected:
        element _table[size];
        const unsigned long _size;
        double phs;
    };
}
#endif
```

10.105 MTOF.cpp File Reference

`#include "MTOF.h"`

10.106 MTOF.cpp

```
//
//  MTOF.cpp
//  DSG
//
//  Created by Alexander Zywicki on 11/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "MTOF.h"
double DSG::MIDI::MTOF(unsigned char const& MIDI_Number){
    return 440.0 *pow(2.0, (MIDI_Number-69.0)/12.0);
}
unsigned char DSG::MIDI::FTOM(double const& Frequency){
    return((log2((Frequency/440.0)))*12.0)+69.0;
}
```

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

```
//
//  MTOF.h
//  DSG
//
//  Created by Alexander Zywicki on 11/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__MTOF__
#define __DSG__MTOF__
#include <math.h>
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::MIDI - Namespace enclosing MIDI processing tools
    namespace MIDI{
        //!\brief DSG::MIDI:MTOF - MIDI to Frequency Conversion
        double   MTOF(unsigned char const& MIDI_Number);
        //!\brief DSG::MIDI:FTOM - Frequency to MIDI Conversion
        unsigned char FTOM(double const& Frequency);
    }
}
#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

```
//
//  Noise.h
//  DSG
//
```

```
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Noise_h
#define DSG_Noise_h
#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"
#endif
```

10.111 NoiseGenerator.cpp File Reference

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

10.112 NoiseGenerator.cpp

```
//
//  NoiseGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "NoiseGenerator.h"
DSG::NoiseGenerator::NoiseGenerator(DSGSample (*
StatelessFunction)(DSGSample)):DSG::SignalProcess(){
    _function = StatelessFunction;
}
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

```
//
//  NoiseGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__NoiseGenerator__
#define __DSG__NoiseGenerator__
#include "SignalGenerator.h"
namespace DSG{
    //!\brief DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate
 signal
    class NoiseGenerator:public SignalProcess{
    public:
        NoiseGenerator(DSGSample (*StatelessFunction)(
DSGSample));
        virtual ~NoiseGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        DSGSample (*_function)(DSGSample);
        DSG::DSGSample _storage;
    };
    inline bool DSG::NoiseGenerator::Perform(
DSG::DSGSample& signal){
        signal = _function(0);
        return true;
    }
    inline bool DSG::NoiseGenerator::Perform(
DSG::RingBuffer& signal){
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
}
#endif /* defined(__DSG__NoiseGenerator__) */
```

10.115 Phasor.cpp File Reference

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

10.116 Phasor.cpp

```
//
//  Phasor.cpp
//  DSG
//
//  Created by Alexander Zywicki on 11/26/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "Phasor.h"
DSG::Phasor::Phasor():_phasor(0),_frequency(0),_dt(0),_offset(0){}
DSG::Phasor::Phasor(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):_phasor(0),_frequency(frequency),_dt(0),_offset(offset){
    Frequency(frequency);
    Phase(offset);
}
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

```
//
//  Phasor.h
//  DSG
//
//  Created by Alexander Zywicki on 11/26/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Phasor__
#define __DSG__Phasor__
#include "DSGTypes.h"
#include "Bounds.h"
#include "AudioSettings.h"
namespace DSG{
    //!\brief DSG::Phasor - Linear Phase Generator
    class Phasor{
    public:
        Phasor();
        Phasor(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
        virtual ~Phasor();
        virtual inline DSG::DSGFrequency const& Frequency();
        virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        virtual inline DSG::DSGPhase const& Phase();
        virtual inline DSG::DSGPhase const& Phase(
DSG::DSGPhase const& value);
    protected:
        //extends sample rate interface
        inline void step();
        inline void sync();
        //------------------------------//
        DSG::DSGFrequency _frequency;//frequency in Hz
        DSG::DSGPhase _dt;//delta time (change in phase per sample) unit: phase 0-1
        DSG::DSGPhase _offset;//phase shift
        DSG::DSGPhase _phasor;//phase counter
    };
    inline DSG::DSGFrequency const& DSG::Phasor::Frequency(){
        return _frequency;
    }
    inline DSG::DSGFrequency const& DSG::Phasor::Frequency(
DSG::DSGFrequency const& value){
        _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
        _dt = _frequency/DSG::SampleRate();
        return _frequency;
    }
    inline DSG::DSGPhase const& DSG::Phasor::Phase(){
        return _offset;
    }
    inline DSG::DSGPhase const& DSG::Phasor::Phase(
DSG::DSGPhase const& value){
        _offset-=value;
        _phasor-=_offset;
        _offset=value;
        return _offset;
    }
    inline void DSG::Phasor::step(){
        _phasor+=_dt;
        _phasor>1.0 ? --_phasor:0;
    }
    inline void DSG::Phasor::sync(){
        _phasor=_offset;
    }
}
#endif /* defined(__DSG__Phasor__) */
```

10.119 PI.h File Reference

Namespaces

- DSG

DSG - A Collection of tools for Digital Signal Generation.

Macros

- #define PI 3.141592653589793238462643383277
- #define TWOPI 6.283185307179586476925286766559

10.119.1 Macro Definition Documentation

10.119.1.1 #define PI 3.141592653589793238462643383277

Definition at line 27 of file PI.h.

10.119.1.2 #define TWOPI 6.283185307179586476925286766559

Definition at line 28 of file PI.h.

10.120 PI.h

```
//
//  PI.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_PI_h
#define DSG_PI_h
namespace DSG{
#define PI    3.141592653589793238462643383277
#define TWOPI 6.283185307179586476925286766559
}
#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

```
//
//  Pink.h
//  DSG
//
//  Created by Alexander Zywicki on 10/8/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Pink_h
#define DSG_Pink_h
#include "Gaussian.h"
#include "DCBlocker.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::Pink - Pink Noise Generator Function
        template<typename decimal=DSG::DSGSample>
        decimal Pink(decimal=0.0){
            //routine: Get white or gaussian, filter, return
            static decimal b0,b1,b2,b3,b4,b5,b6;
            static decimal normalizer=1;//variable used to actively normalize the output
            static DSG::DCBlocker _block;
            decimal white = DSG::Noise::Gaussian();
            decimal pink;
            //pinking filter
            b0 = 0.99886 * b0 + white * 0.0555179;
            b1 = 0.99332 * b1 + white * 0.0750759;
            b2 = 0.96900 * b2 + white * 0.1538520;
            b3 = 0.86650 * b3 + white * 0.3104856;
            b4 = 0.55000 * b4 + white * 0.5329522;
            b5 = -0.7616 * b5 - white * 0.0168980;
            pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
            b6 = white * 0.115926;
            if (DSG::Abs(pink)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(pink);
            }
            pink/=normalizer;
            _block.Perform(pink);
            return pink;
        }
    }
}
#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){
```

```
            static DSG::Noise::random_helper<decimal> _rand{};
            return _rand.next();
        }
    }
}
#endif
```

10.125 RingBuffer.cpp File Reference

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

10.126 RingBuffer.cpp

```
//
//  RingBuffer.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "RingBuffer.h"
DSG:: RingBuffer::RingBuffer():Buffer(0),_read(0),_write(0),_count(0),
      MASK(0){}
DSG:: RingBuffer::RingBuffer(const size_t size):
      Buffer(make_pow_2(size)),_read(0),_write(0),_count(0){
    MASK = this->_size-1;
}
DSG:: RingBuffer::RingBuffer(RingBuffer& buffer):
      Buffer(buffer){
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
}
DSG:: RingBuffer& DSG:: RingBuffer::operator=(
      RingBuffer& buffer){
    Buffer::operator=(buffer);
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
    return *this;
}
DSG:: RingBuffer::~RingBuffer(){Flush();}

```

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

```
//
//  RingBuffer.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__RingBuffer__
#define __DSG__RingBuffer__
#ifdef DEBUG
#include <iostream>
#endif
#include <atomic>
#include "DSGMath.h"
#include "Buffer.h"
namespace DSG {
    /*!\brief DSG::RingBuffer - Circular Buffer of Audio
     */
    class RingBuffer:public DSG::Buffer {
    protected:
        std::atomic<size_t> _write;
        std::atomic<size_t> _read;
        size_t _count;
        size_t MASK;
        size_t write;
        size_t read;
        inline size_t next(size_t current);
        inline size_t make_pow_2(size_t number);
    public:
        RingBuffer();
        RingBuffer(const size_t size);
        RingBuffer(RingBuffer& buffer);
        RingBuffer& operator=(RingBuffer& buffer);
        virtual ~RingBuffer();
        inline bool Write(const DSGSample& elem);
        inline bool Read(DSG::DSGSample& elem);
        inline size_t const& Count()const;
        inline bool Full()const;
        inline bool Empty()const;
        inline void Flush();
        friend bool operator>>(DSG::DSGSample const& signal,
DSG::RingBuffer& buffer){
            return buffer.Write(signal);
        }
        friend bool operator<<(DSG::DSGSample& signal,
DSG::RingBuffer& buffer){
            return buffer.Read(signal);
        }
#ifdef DEBUG
```

```
        friend std::ostream& operator<<(std::ostream& os, DSG:: RingBuffer const& buffer){
            if (!buffer.Empty()) {
                size_t index= buffer._read;
                size_t count=buffer.Count();
                size_t size = buffer.Size();
                for (int i=0; i<count; ++i) {
                    os<<index<<": "<<buffer._buffer[index]<<std::endl;
                    index = ((index+1)%size);
                }
            }return os;
        }
#endif
    };
    inline bool DSG::RingBuffer::Full()const{
        return _count==this->_size;
    }
    inline bool DSG::RingBuffer::Empty()const{
        return _count==0;
    }
    inline void DSG::RingBuffer::Flush(){
        _write.store(0,std::memory_order_relaxed);
        _read.store(0,std::memory_order_relaxed);
        _count=0;
    }
    inline bool DSG::RingBuffer::Write(const DSGSample& elem){
        if (!Full()) {
            write = _write.load(std::memory_order_acquire);
            _write.store(next(write),std::memory_order_release);
            this->_buffer[write] = elem;
            ++_count;
            return true;
        }else return false;
    }
    inline bool DSG::RingBuffer::Read(DSGSample& elem){
        if (!Empty()) {
            read = _read.load(std::memory_order_acquire);
            _read.store(next(read),std::memory_order_release);
            elem = this->_buffer[read];
            --_count;
            return true;
        }else return false;
    }
    inline size_t const& DSG::RingBuffer::Count()const{
        return _count;
    }
    //note: RingBuffer implementation will force a power of 2 size to allow use of bitwise increment.
    inline size_t DSG::RingBuffer::next(size_t current){return (current+1) & MASK;}
    inline size_t DSG::RingBuffer::make_pow_2(size_t number){
        return pow(2, ceil(log(number)/log(2)));
    }
}
#endif /* defined(__DSG__RingBuffer__) */
```

10.129 SignalGenerator.cpp File Reference

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

10.130 SignalGenerator.cpp

```
//
//  SignalGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "SignalGenerator.h"
DSG::SignalGenerator::SignalGenerator():DSG::
SignalProcess(),DSG::Phasor(){}
DSG::SignalGenerator::SignalGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalProcess(),DSG::Phasor(frequency,offset){}
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

```
//
//  SignalGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__SignalGenerator__
#define __DSG__SignalGenerator__
#include "SignalProcess.h"
#include "Sine.h"
#include "Phasor.h"
namespace DSG{
    /*!\brief DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation
     */
    class SignalGenerator:public DSG::SignalProcess,public DSG::Phasor{
    public:
        SignalGenerator();
        SignalGenerator(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
        virtual ~SignalGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
        virtual inline bool SampleRateChanged(DSG::DSGFrequency const& sampleRate);
    protected:
        DSG::DSGSample _storage;//storage variable for calculations
    };
    inline unsigned long MaxHarms(DSG::DSGFrequency const& frequency){
        double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
        _s/=frequency;
        return _s;
    }
}
inline bool DSG::SignalGenerator::Perform(DSG::DSGSample& signal){
    signal=0;
    return false;
}
inline bool DSG::SignalGenerator::Perform(DSG::RingBuffer& signal){
    signal.Flush();
    return false;
}
inline bool DSG::SignalGenerator::SampleRateChanged(DSG::DSGFrequency const& sampleRate){
    Frequency(_frequency);
    return true;
}

#endif /* defined(__DSG__SignalGenerator__) */
```

10.133 SignalProcess.cpp File Reference

#include "SignalProcess.h"

10.134 SignalProcess.cpp

```
//
//  SignalProcess.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#include "SignalProcess.h"
DSG::SignalProcess::SignalProcess(){
    VerifySampleRateSet();//ensure that there is a valid sample rate set
    AddSampleRateListener(this);//listen for changes in the sample rate
}
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

```
//
//  SignalProcess.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__SignalProcess__
#define __DSG__SignalProcess__
#include "DSGTypes.h"
#include "RingBuffer.h"
#include "AudioSettings.h"
namespace DSG {
    /*!\brief DSG::SignalProcess - Defines Base Interface For Audio Processing
     */
    class SignalProcess{
    public:
        SignalProcess();
        virtual ~SignalProcess();
        //Defines Interface for sample rate processing
        virtual inline bool Perform(DSG::DSGSample& signal)=0;
        virtual inline bool Perform(DSG::RingBuffer& signal)=0;
        virtual inline bool SampleRateChanged(DSG::DSGFrequency const&
sampleRate)=0;
    };
    inline bool DSG::SignalProcess::SampleRateChanged(
DSG::DSGFrequency const& sampleRate){
        return true;
    }
}
#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

```
//
//  Sinc.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Sinc__
#define __DSG__Sinc__
#include "PI.h"
#include "Sine.h"
#include "Denormal.h"
#include <type_traits>
#include "DSGMath.h"
namespace DSG{
    //!\brief DSG::Sinc - Implements the Sinc() function (sin(PI*x)/PI*x)
    template<typename decimal>
    inline decimal Sinc(decimal const& x) {
        static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
 Point Type");
        decimal pix;
        if (DSG::IsDenormal(x)) {
            return 1.0;
        }else{
            pix = PI*x;
            return DSG::Sin(pix)/pix;
        }
    }
}
#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.
```

```
 You should have received a copy of the GNU Lesser General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Sine__
#define __DSG__Sine__
#include "LUT.h"
#include "PI.h"
namespace DSG {
    namespace{
            #define LUT_SIZE 16384
        typedef enum Sine_Implementations{
            /*!\brief DSG::Sine_Implementations - Specifies The Implementation Option For DSG::Sin<>()*/
            Sine_Taylor =1,
            Sine_LUT =2,
            Sine_Default = Sine_LUT
        }Sine_Implementations;
        /*!\brief DSG::Sin() - Templated Sin Function With Optional Implementation
         */
        template<unsigned implementation> inline double Sin(double const& x){
            return 0;
        }
        /*!\brief DSG::Sin() - Templated Cos Function With Optional Implementation
         */
        template<unsigned implementation> inline double Cos(double const& x){
            return 0;
        }
        template<> inline double Sin<Sine_LUT>(double const& x){
            static DSG::LUT<double, LUT_SIZE> _lut(&sin,
TWOPI);
            return _lut(x);
        }
        template<> inline double Cos<Sine_LUT>(double const& x){
            static DSG::LUT<double, LUT_SIZE> _lut(&cos,
TWOPI);
            return _lut(x);
        }
        template<> inline double Sin<Sine_Taylor>(double const& x){
            //taylor serie version here
            return 0;
        }
        template<> inline double Cos<Sine_Taylor>(double const& x){
            //taylor series version here
            return 0;
        }
    }
    /*!\brief DSG::Sin() - General Purpose Sin Function, double precision
     */
    //!\todo Implement Taylor Series implementation of Sin Function
    inline double Sin(double const& x){
        return static_cast<double>(Sin<Sine_Default>(x));//wrap default implementation as non template
    }
    /*!\brief DSG::Sin() - General Purpose Sin Function, single precision
     */
    inline float Sin(float const& x){
        return static_cast<float>(Sin<Sine_Default>(x));
    }
    /*!\brief DSG::Cos() - General Purpose Cos Function, double precision
     */
    //!\todo Implement Taylor Series implementation of Cos Function
    inline double Cos(double const& x){
        return static_cast<double>(Cos<Sine_Default>(x));//wrap default implementation as non template
    }
    /*!\brief DSG::Cos() - General Purpose Cos Function, single precision
     */
    inline float Cos(float const& x){
        return static_cast<float>(Cos<Sine_Default>(x));
    }
}
#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

```
//
//  Sleep.h
//  DSG
//
//  Created by Alexander Zywicki on 10/5/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef __DSG__Sleep__
#define __DSG__Sleep__
#include <chrono>
#include <thread>
namespace DSG{
    //!\brief DSG::Sleep - Millisecond Sleep Function
    template<typename integer>
    void Sleep(integer const& milliseconds){
        std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
    }
}
#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

```
//
//  White.h
//  DSG
//
//  Created by Alexander Zywicki on 10/14/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_White_h
#define DSG_White_h
#include "DSGTypes.h"
#include "Random.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::White - White Noise Generator Function
        template<typename decimal = DSG::DSGSample>
        inline decimal White(decimal=0.0){
            return DSG::Random<decimal>();
        }
    }
}
#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 lutsize>
  void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > &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 DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&windowFunction)(decimal), decimal range=1.0)

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

10.146 Window.h

```
//
//  Window.h
//  DSG
//
//  Created by Alexander Zywicki on 10/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 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/>.
 */
#ifndef DSG_Window_h
#define DSG_Window_h
#include "LUT.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::Window - Window functions and utilities
    namespace Window{
        //!\brief 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 const&),decimal range = 1.0){
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
        //!\brief 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){
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
    }
}
#endif
```

Index