

iir1

Generated by Doxygen 1.8.13

Contents

1	IIR1 -- Realtime C++ filter library	1
2	Namespace Index	4
2.1	Namespace List	4
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	8
4.1	Class List	8
5	Namespace Documentation	11
5.1	Iir Namespace Reference	11
5.1.1	Detailed Description	12
5.1.2	Enumeration Type Documentation	13
5.1.3	Function Documentation	14
5.2	Iir::Butterworth Namespace Reference	14
5.2.1	Detailed Description	15
5.3	Iir::ChebyshevI Namespace Reference	15
5.3.1	Detailed Description	15
5.4	Iir::ChebyshevII Namespace Reference	16
5.4.1	Detailed Description	16
5.5	Iir::Custom Namespace Reference	16
5.5.1	Detailed Description	16

6	Class Documentation	16
6.1	lir::RBJ::AllPass Struct Reference	16
6.2	lir::Butterworth::AnalogLowPass Class Reference	17
6.2.1	Detailed Description	17
6.3	lir::ChebyshevI::AnalogLowPass Class Reference	17
6.4	lir::ChebyshevII::AnalogLowPass Class Reference	18
6.5	lir::ChebyshevII::AnalogLowShelf Class Reference	18
6.6	lir::Butterworth::AnalogLowShelf Class Reference	18
6.7	lir::ChebyshevI::AnalogLowShelf Class Reference	19
6.8	lir::ChebyshevI::BandPass< FilterOrder, StateType > Struct Template Reference	19
6.8.1	Detailed Description	19
6.8.2	Member Function Documentation	19
6.9	lir::Butterworth::BandPass< FilterOrder, StateType > Struct Template Reference	20
6.9.1	Detailed Description	20
6.9.2	Member Function Documentation	21
6.10	lir::ChebyshevII::BandPass< FilterOrder, StateType > Struct Template Reference	21
6.10.1	Detailed Description	21
6.10.2	Member Function Documentation	22
6.11	lir::RBJ::BandPass1 Struct Reference	22
6.11.1	Detailed Description	23
6.11.2	Member Function Documentation	23
6.12	lir::RBJ::BandPass2 Struct Reference	23
6.12.1	Detailed Description	24
6.12.2	Member Function Documentation	24
6.13	lir::ChebyshevII::BandPassBase Struct Reference	24
6.14	lir::ChebyshevI::BandPassBase Struct Reference	25
6.15	lir::Butterworth::BandPassBase Struct Reference	25
6.16	lir::BandPassTransform Class Reference	25
6.16.1	Detailed Description	26
6.17	lir::Butterworth::BandShelf< FilterOrder, StateType > Struct Template Reference	26

6.17.1 Detailed Description	26
6.17.2 Member Function Documentation	26
6.18 lir::RBJ::BandShelf Struct Reference	27
6.18.1 Detailed Description	27
6.18.2 Member Function Documentation	27
6.19 lir::ChebyshevI::BandShelf< FilterOrder, StateType > Struct Template Reference	28
6.19.1 Detailed Description	28
6.19.2 Member Function Documentation	28
6.20 lir::ChebyshevII::BandShelf< FilterOrder, StateType > Struct Template Reference	29
6.20.1 Detailed Description	29
6.20.2 Member Function Documentation	30
6.21 lir::Butterworth::BandShelfBase Struct Reference	30
6.22 lir::ChebyshevII::BandShelfBase Struct Reference	31
6.23 lir::ChebyshevI::BandShelfBase Struct Reference	31
6.24 lir::ChebyshevI::BandStop< FilterOrder, StateType > Struct Template Reference	32
6.24.1 Detailed Description	32
6.24.2 Member Function Documentation	32
6.25 lir::RBJ::BandStop Struct Reference	33
6.25.1 Detailed Description	33
6.25.2 Member Function Documentation	33
6.26 lir::ChebyshevII::BandStop< FilterOrder, StateType > Struct Template Reference	34
6.26.1 Detailed Description	34
6.26.2 Member Function Documentation	34
6.27 lir::Butterworth::BandStop< FilterOrder, StateType > Struct Template Reference	35
6.27.1 Detailed Description	35
6.27.2 Member Function Documentation	35
6.28 lir::Butterworth::BandStopBase Struct Reference	36
6.29 lir::ChebyshevII::BandStopBase Struct Reference	37
6.30 lir::ChebyshevI::BandStopBase Struct Reference	37
6.31 lir::BandStopTransform Class Reference	37

6.31.1 Detailed Description	38
6.32 lir::Biquad Class Reference	38
6.32.1 Member Function Documentation	39
6.33 lir::BiquadPoleState Struct Reference	42
6.33.1 Detailed Description	42
6.34 lir::Cascade Class Reference	42
6.34.1 Member Function Documentation	43
6.35 lir::CascadeStages< MaxStages, StateType > Class Template Reference	43
6.35.1 Detailed Description	44
6.35.2 Member Function Documentation	44
6.36 lir::ComplexPair Struct Reference	45
6.36.1 Detailed Description	45
6.37 lir::DirectFormI Class Reference	45
6.37.1 Detailed Description	45
6.38 lir::DirectFormII Class Reference	46
6.38.1 Detailed Description	46
6.39 lir::EnvelopeFollower< Channels, Value > Class Template Reference	46
6.40 lir::ChebyshevI::HighPass< FilterOrder, StateType > Struct Template Reference	46
6.40.1 Detailed Description	46
6.40.2 Member Function Documentation	47
6.41 lir::RBJ::HighPass Struct Reference	47
6.41.1 Detailed Description	48
6.41.2 Member Function Documentation	48
6.42 lir::ChebyshevII::HighPass< FilterOrder, StateType > Struct Template Reference	48
6.42.1 Detailed Description	48
6.42.2 Member Function Documentation	49
6.43 lir::Butterworth::HighPass< FilterOrder, StateType > Struct Template Reference	49
6.43.1 Detailed Description	50
6.43.2 Member Function Documentation	51
6.44 lir::ChebyshevI::HighPassBase Struct Reference	51

6.45	lir::ChebyshevII::HighPassBase Struct Reference	52
6.46	lir::Butterworth::HighPassBase Struct Reference	52
6.47	lir::HighPassTransform Class Reference	52
6.47.1	Detailed Description	53
6.48	lir::RBJ::HighShelf Struct Reference	53
6.48.1	Detailed Description	53
6.48.2	Member Function Documentation	53
6.49	lir::ChebyshevI::HighShelf< FilterOrder, StateType > Struct Template Reference	54
6.49.1	Detailed Description	54
6.49.2	Member Function Documentation	54
6.50	lir::ChebyshevII::HighShelf< FilterOrder, StateType > Struct Template Reference	55
6.50.1	Detailed Description	55
6.50.2	Member Function Documentation	55
6.51	lir::Butterworth::HighShelf< FilterOrder, StateType > Struct Template Reference	56
6.51.1	Detailed Description	56
6.51.2	Member Function Documentation	57
6.52	lir::Butterworth::HighShelfBase Struct Reference	57
6.53	lir::ChebyshevII::HighShelfBase Struct Reference	58
6.54	lir::ChebyshevI::HighShelfBase Struct Reference	58
6.55	lir::RBJ::IIRNotch Struct Reference	59
6.55.1	Detailed Description	59
6.55.2	Member Function Documentation	59
6.56	lir::Layout< MaxPoles > Class Template Reference	60
6.56.1	Detailed Description	60
6.57	lir::LayoutBase Class Reference	60
6.57.1	Detailed Description	60
6.58	lir::RBJ::LowPass Struct Reference	60
6.58.1	Detailed Description	61
6.58.2	Member Function Documentation	61
6.59	lir::Butterworth::LowPass< FilterOrder, StateType > Struct Template Reference	61

6.59.1 Detailed Description	62
6.59.2 Member Function Documentation	62
6.60 lir::ChebyshevII::LowPass< FilterOrder, StateType > Struct Template Reference	62
6.60.1 Detailed Description	63
6.60.2 Member Function Documentation	63
6.61 lir::ChebyshevI::LowPass< FilterOrder, StateType > Struct Template Reference	63
6.61.1 Detailed Description	64
6.61.2 Member Function Documentation	64
6.62 lir::Butterworth::LowPassBase Struct Reference	65
6.63 lir::ChebyshevII::LowPassBase Struct Reference	65
6.64 lir::ChebyshevI::LowPassBase Struct Reference	66
6.65 lir::LowPassTransform Class Reference	66
6.65.1 Detailed Description	66
6.66 lir::Butterworth::LowShelf< FilterOrder, StateType > Struct Template Reference	66
6.66.1 Detailed Description	67
6.66.2 Member Function Documentation	67
6.67 lir::RBJ::LowShelf Struct Reference	67
6.67.1 Detailed Description	68
6.67.2 Member Function Documentation	68
6.68 lir::ChebyshevI::LowShelf< FilterOrder, StateType > Struct Template Reference	68
6.68.1 Detailed Description	69
6.68.2 Member Function Documentation	69
6.69 lir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference	70
6.69.1 Detailed Description	70
6.69.2 Member Function Documentation	70
6.70 lir::ChebyshevI::LowShelfBase Struct Reference	71
6.71 lir::ChebyshevII::LowShelfBase Struct Reference	71
6.72 lir::Butterworth::LowShelfBase Struct Reference	72
6.73 lir::Custom::OnePole Struct Reference	72
6.73.1 Detailed Description	72

6.74	lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > Struct Template Reference	73
6.74.1	Detailed Description	73
6.75	lir::PoleFilterBase< AnalogPrototype > Class Template Reference	73
6.75.1	Detailed Description	73
6.76	lir::PoleFilterBase2 Class Reference	74
6.76.1	Detailed Description	74
6.77	lir::PoleZeroPair Struct Reference	74
6.77.1	Detailed Description	74
6.78	lir::RBJ::RBJbase Struct Reference	75
6.78.1	Detailed Description	75
6.79	lir::SlopeDetector< Channels, Value > Class Template Reference	76
6.80	lir::Custom::SOSCascade< NSOS, StateType > Struct Template Reference	76
6.80.1	Detailed Description	76
6.80.2	Member Function Documentation	76
6.81	lir::Cascade::Storage Struct Reference	77
6.81.1	Detailed Description	77
6.81.2	Constructor & Destructor Documentation	77
6.82	lir::TransposedDirectFormII Class Reference	77
6.83	lir::Custom::TwoPole Struct Reference	78
6.83.1	Detailed Description	78
	Index	79

1 IIR1 -- Realtime C++ filter library

An infinite impulse response (IIR) filter library for Linux, Mac OSX and Windows which implements Butterworth, RBJ, Chebychev filters and can easily import coefficients generated by Python (scipy).

The filter processes the data sample by sample for realtime processing.

It uses templates to allocate the required memory so that it can run without any malloc / new commands for example on embedded systems. Memory is allocated at compile time so that there is never the risk of memory leaks.

How to use the filter

First the filter is instantiated, then the parameters are set with the function `setup` and then it's ready to be used for sample by sample realtime filtering.

Setting the filter parameters

All filters are available as lowpass, highpass, bandpass and bandstop/notch filters. Butterworth / Chebyshev offer also low/high/band-shelves with specified passband gain and 0dB gain in the stopband.

See the header files in `\iir` or the documentation for the arguments of the `setup` commands.

The examples below are for lowpass filters:

1. Butterworth – [Butterworth.h](#) Standard filter suitable for most applications. Monotonic response.

```
const int order = 4; // 4th order (=2 biquads)
Iir::Butterworth::LowPass<order> f;
const float samplingrate = 1000; // Hz
const float cutoff_frequency = 5; // Hz
f.setup (samplingrate, cutoff_frequency);
```

2. Chebyshev Type I – [ChebyshevI.h](#) With permissible passband ripple in dB.

```
Iir::ChebyshevI::LowPass<order> f;
const float passband_ripple_in_db = 5;
f.setup (samplingrate,
        cutoff_frequency,
        passband_ripple_in_db);
```

3. Chebyshev Type II – [ChebyshevII.h](#) With worst permissible stopband rejection in dB.

```
Iir::ChebyshevII::LowPass<order> f;
double stopband_ripple_in_db = 20;
f.setup (samplingrate,
        cutoff_frequency,
        stopband_ripple_in_db);
```

4. RBJ – [RBJ.h](#) 2nd order filters with cutoff and Q factor.

```
Iir::RBJ::LowPass f;
const float cutoff_frequency = 100;
const float Q_factor = 5;
f.setup (samplingrate, cutoff_frequency, Q_factor);
```

5. Designing filters with Python's `scipy.signal` – [Custom.h](#)

```
#####
# Python
# See "elliptic_design.py" for the complete code.
from scipy import signal
order = 4
sos = signal.ellip(order, 5, 40, 0.2, 'low', output='sos')
print(sos) # copy/paste the coefficients over & replace [] with {}

/////
// C++
// part of "iirdemo.cpp"
Iir::Custom::SOSCascade<2> cust;
const double coeff[][6] = {
    {1.665623674062209972e-02,
     -3.924801366970616552e-03,
     1.665623674062210319e-02,
     1.000000000000000000e+00,
     -1.715403014004022175e+00,
     8.100474793174089472e-01},
    {1.000000000000000000e+00,
     -1.369778997100624895e+00,
     1.000000000000000022e+00,
     1.000000000000000000e+00,
     -1.605878925999785656e+00,
     9.538657786383895054e-01}
};
cust.setup(coeff);
```

Realtime filtering sample by sample

Samples are processed one by one. In the example below a sample `x` is processed with the `filter` command and then saved in `y`. The type of `x` can either be float or double:

```
float y = f.filter(x);
```

This is then repeated for every incoming sample in a loop or event handler.

Packages for Ubuntu (xenial / bionic):

If you have Ubuntu xenial or bionic then install it as a pre-compiled package:

```
sudo add-apt-repository ppa:berndporr/usbdx
```

It's available for 32,64 bit PC and 32,64 bit ARM (Raspberry PI etc). The documentation and the example programs are in:

```
/usr/share/doc/iirl-dev/
```

Compilation from source

The build tool is `cmake` which generates the make- or project files for the different platforms. `cmake` is available for Linux, Windows and Mac. It also compiles directly on a Raspberry PI.

Linux / Mac

Run

```
cmake .
```

which generates the Makefile. Then run:

```
make
sudo make install
```

which installs it under `/usr/local/lib` and `/usr/local/include`.

Both gcc and clang have been tested.

Windows

```
cmake -G "Visual Studio 15 2017 Win64" .
```

See `cmake` for the different build-options. Above is for a 64 bit build. Then start Visual C++ and open the solution. This will create the DLL and the LIB files. Under Windows it's highly recommended to use the static library and link it into the application program.

Unit tests

Run unit tests by typing `make test` or just `ctest`. These test if after a delta pulse all filters relax to zero and that their outputs never become NaN.

Documentation

Overview

For an overview of the class structure and general concepts have a look at `Documentation.txt`.

Learn from the demos

The easiest way to learn is from the examples which are in the `demo` directory. A delta pulse as a test signal is sent into the different filters and saved in a file. With the Python script `plot_impulse_fresponse.py` you can then plot the frequency responses.

Also the directory containing the unit tests provides examples for every filter type.

Detailed documentation

A PDF of all classes, methods and in particular `setup` functions is in the `doc/pdf` directory.

Run `doxygen` to generate the HTML documentation.

Example filter responses

These responses have been generated by `iirdemo.cpp` in the `/demo/` directory and then plotted with `plot_impulse_fresponse.py`.

Credits

This library has been adapted from Vinnie Falco's original work which can be found here:

<https://github.com/vinniefalco/DSPFilters>

While his original library processes audio arrays this library has been adapted to do realtime processing sample by sample. Also, in contrast to the original library the `setup` command won't require the filter order and instead remembers it from the template argument.

Enjoy!

Bernd Porr – <http://www.berndporr.me.uk>

2 Namespace Index

2.1 Namespace List

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

lir	11
lir::Butterworth	14
lir::ChebyshevI	15
lir::ChebyshevII	16
lir::Custom	16

3 Hierarchical Index

3.1 Class Hierarchy

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

BandPassBase

lir::PoleFilter< BandPassBase, StateType, FilterOrder, FilterOrder *2 >	73
lir::Butterworth::BandPass< FilterOrder, StateType >	20
lir::ChebyshevI::BandPass< FilterOrder, StateType >	19
lir::ChebyshevII::BandPass< FilterOrder, StateType >	21

[lir::BandPassTransform](#)

25

BandShelfBase

lir::PoleFilter< BandShelfBase, StateType, FilterOrder, FilterOrder *2 >	73
lir::Butterworth::BandShelf< FilterOrder, StateType >	26
lir::ChebyshevI::BandShelf< FilterOrder, StateType >	28
lir::ChebyshevII::BandShelf< FilterOrder, StateType >	29

BandStopBase

lir::PoleFilter< BandStopBase, StateType, FilterOrder, FilterOrder *2 >	73
lir::Butterworth::BandStop< FilterOrder, StateType >	35
lir::ChebyshevI::BandStop< FilterOrder, StateType >	32
lir::ChebyshevII::BandStop< FilterOrder, StateType >	34

[lir::BandStopTransform](#)

37

[lir::Biquad](#)

38

lir::Custom::OnePole	72
lir::Custom::TwoPole	78
lir::RBJ::RBJbase	75
lir::RBJ::AllPass	16
lir::RBJ::BandPass1	22

lir::RBJ::BandPass2	23
lir::RBJ::BandShelf	27
lir::RBJ::BandStop	33
lir::RBJ::HighPass	47
lir::RBJ::HighShelf	53
lir::RBJ::IIRNotch	59
lir::RBJ::LowPass	60
lir::RBJ::LowShelf	67
lir::Cascade	42
lir::PoleFilterBase2	74
lir::PoleFilterBase< AnalogPrototype >	73
lir::PoleFilterBase< AnalogLowPass >	73
lir::Butterworth::BandPassBase	25
lir::Butterworth::BandStopBase	36
lir::Butterworth::HighPassBase	52
lir::Butterworth::LowPassBase	65
lir::ChebyshevI::BandPassBase	25
lir::ChebyshevI::BandStopBase	37
lir::ChebyshevI::HighPassBase	51
lir::ChebyshevI::LowPassBase	66
lir::ChebyshevII::BandPassBase	24
lir::ChebyshevII::BandStopBase	37
lir::ChebyshevII::HighPassBase	52
lir::ChebyshevII::LowPassBase	65
lir::PoleFilterBase< AnalogLowShelf >	73
lir::Butterworth::BandShelfBase	30
lir::Butterworth::HighShelfBase	57
lir::Butterworth::LowShelfBase	72
lir::ChebyshevI::BandShelfBase	31
lir::ChebyshevI::HighShelfBase	58
lir::ChebyshevI::LowShelfBase	71
lir::ChebyshevII::BandShelfBase	31

lir::ChebyshevII::HighShelfBase	58
lir::ChebyshevII::LowShelfBase	71
lir::CascadeStages< MaxStages, StateType >	43
lir::CascadeStages< NSOS, StateType >	43
lir::Custom::SOSCascade< NSOS, StateType >	76
lir::CascadeStages<(MaxAnalogPoles+1)/2, StateType >	43
lir::PoleFilter< HighPassBase, StateType, FilterOrder >	73
lir::Butterworth::HighPass< FilterOrder, StateType >	49
lir::ChebyshevI::HighPass< FilterOrder, StateType >	46
lir::ChebyshevII::HighPass< FilterOrder, StateType >	48
lir::PoleFilter< HighShelfBase, StateType, FilterOrder >	73
lir::Butterworth::HighShelf< FilterOrder, StateType >	56
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	54
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	55
lir::PoleFilter< LowPassBase, StateType, FilterOrder >	73
lir::Butterworth::LowPass< FilterOrder, StateType >	61
lir::ChebyshevI::LowPass< FilterOrder, StateType >	63
lir::ChebyshevII::LowPass< FilterOrder, StateType >	62
lir::PoleFilter< LowShelfBase, StateType, FilterOrder >	73
lir::Butterworth::LowShelf< FilterOrder, StateType >	66
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	68
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	70
lir::CascadeStages<(MaxDigitalPoles+1)/2, StateType >	43
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	73
lir::PoleFilter< BandPassBase, StateType, FilterOrder, FilterOrder *2 >	73
lir::PoleFilter< BandShelfBase, StateType, FilterOrder, FilterOrder *2 >	73
lir::PoleFilter< BandStopBase, StateType, FilterOrder, FilterOrder *2 >	73
complex_pair_t	
lir::ComplexPair	45
lir::DirectFormI	45
lir::DirectFormII	46
lir::EnvelopeFollower< Channels, Value >	46
HighPassBase	

lir::PoleFilter< HighPassBase, StateType, FilterOrder >	73
lir::HighPassTransform	52
HighShelfBase	
lir::PoleFilter< HighShelfBase, StateType, FilterOrder >	73
lir::Layout< MaxPoles >	60
lir::Layout< MaxAnalogPoles >	60
lir::Layout< MaxDigitalPoles >	60
lir::LayoutBase	60
lir::Butterworth::AnalogLowPass	17
lir::Butterworth::AnalogLowShelf	18
lir::ChebyshevI::AnalogLowPass	17
lir::ChebyshevI::AnalogLowShelf	19
lir::ChebyshevII::AnalogLowPass	18
lir::ChebyshevII::AnalogLowShelf	18
LowPassBase	
lir::PoleFilter< LowPassBase, StateType, FilterOrder >	73
lir::LowPassTransform	66
LowShelfBase	
lir::PoleFilter< LowShelfBase, StateType, FilterOrder >	73
lir::PoleZeroPair	74
lir::BiquadPoleState	42
lir::SlopeDetector< Channels, Value >	76
lir::Cascade::Storage	77
lir::TransposedDirectFormII	77
BaseClass	
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	73

4 Class Index

4.1 Class List

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

lir::RBJ::AllPass	16
lir::Butterworth::AnalogLowPass	17
lir::ChebyshevI::AnalogLowPass	17

lir::ChebyshevII::AnalogLowPass	18
lir::ChebyshevII::AnalogLowShelf	18
lir::Butterworth::AnalogLowShelf	18
lir::ChebyshevI::AnalogLowShelf	19
lir::ChebyshevI::BandPass< FilterOrder, StateType >	19
lir::Butterworth::BandPass< FilterOrder, StateType >	20
lir::ChebyshevII::BandPass< FilterOrder, StateType >	21
lir::RBJ::BandPass1	22
lir::RBJ::BandPass2	23
lir::ChebyshevII::BandPassBase	24
lir::ChebyshevI::BandPassBase	25
lir::Butterworth::BandPassBase	25
lir::BandPassTransform	25
lir::Butterworth::BandShelf< FilterOrder, StateType >	26
lir::RBJ::BandShelf	27
lir::ChebyshevI::BandShelf< FilterOrder, StateType >	28
lir::ChebyshevII::BandShelf< FilterOrder, StateType >	29
lir::Butterworth::BandShelfBase	30
lir::ChebyshevII::BandShelfBase	31
lir::ChebyshevI::BandShelfBase	31
lir::ChebyshevI::BandStop< FilterOrder, StateType >	32
lir::RBJ::BandStop	33
lir::ChebyshevII::BandStop< FilterOrder, StateType >	34
lir::Butterworth::BandStop< FilterOrder, StateType >	35
lir::Butterworth::BandStopBase	36
lir::ChebyshevII::BandStopBase	37
lir::ChebyshevI::BandStopBase	37
lir::BandStopTransform	37
lir::Biquad	38
lir::BiquadPoleState	42
lir::Cascade	42
lir::CascadeStages< MaxStages, StateType >	43

lir::ComplexPair	45
lir::DirectFormI	45
lir::DirectFormII	46
lir::EnvelopeFollower< Channels, Value >	46
lir::ChebyshevI::HighPass< FilterOrder, StateType >	46
lir::RBJ::HighPass	47
lir::ChebyshevII::HighPass< FilterOrder, StateType >	48
lir::Butterworth::HighPass< FilterOrder, StateType >	49
lir::ChebyshevI::HighPassBase	51
lir::ChebyshevII::HighPassBase	52
lir::Butterworth::HighPassBase	52
lir::HighPassTransform	52
lir::RBJ::HighShelf	53
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	54
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	55
lir::Butterworth::HighShelf< FilterOrder, StateType >	56
lir::Butterworth::HighShelfBase	57
lir::ChebyshevII::HighShelfBase	58
lir::ChebyshevI::HighShelfBase	58
lir::RBJ::IIRNotch	59
lir::Layout< MaxPoles >	60
lir::LayoutBase	60
lir::RBJ::LowPass	60
lir::Butterworth::LowPass< FilterOrder, StateType >	61
lir::ChebyshevII::LowPass< FilterOrder, StateType >	62
lir::ChebyshevI::LowPass< FilterOrder, StateType >	63
lir::Butterworth::LowPassBase	65
lir::ChebyshevII::LowPassBase	65
lir::ChebyshevI::LowPassBase	66
lir::LowPassTransform	66
lir::Butterworth::LowShelf< FilterOrder, StateType >	66
lir::RBJ::LowShelf	67

lir::ChebyshevI::LowShelf< FilterOrder, StateType >	68
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	70
lir::ChebyshevI::LowShelfBase	71
lir::ChebyshevII::LowShelfBase	71
lir::Butterworth::LowShelfBase	72
lir::Custom::OnePole	72
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	73
lir::PoleFilterBase< AnalogPrototype >	73
lir::PoleFilterBase2	74
lir::PoleZeroPair	74
lir::RBJ::RBJbase	75
lir::SlopeDetector< Channels, Value >	76
lir::Custom::SOSCascade< NSOS, StateType >	76
lir::Cascade::Storage	77
lir::TransposedDirectFormII	77
lir::Custom::TwoPole	78

5 Namespace Documentation

5.1 lir Namespace Reference

Namespaces

- [Butterworth](#)
- [ChebyshevI](#)
- [ChebyshevII](#)
- [Custom](#)

Classes

- class [BandPassTransform](#)
- class [BandStopTransform](#)
- class [Biquad](#)
- struct [BiquadPoleState](#)
- class [Cascade](#)
- class [CascadeStages](#)
- struct [ComplexPair](#)
- class [DirectFormI](#)
- class [DirectFormII](#)
- class [EnvelopeFollower](#)

- class [HighPassTransform](#)
- class [Layout](#)
- class [LayoutBase](#)
- class [LowPassTransform](#)
- struct [PoleFilter](#)
- class [PoleFilterBase](#)
- class [PoleFilterBase2](#)
- struct [PoleZeroPair](#)
- class [SlopeDetector](#)
- class [TransposedDirectFormII](#)

Enumerations

- enum [Kind](#)

Functions

- `template<class Td , class Ts >`
void [add](#) (int samples, Td *dest, Ts const *src, int destSkip=0, int srcSkip=0)
- `template<typename Td , typename Ts >`
void [add](#) (int channels, int samples, Td *const *dest, Ts const *const *src)
- `template<typename Td , typename Ts >`
void [copy](#) (int samples, Td *dest, Ts const *src, int destSkip=0, int srcSkip=0)

5.1.1 Detailed Description

"A Collection of Useful C++ Classes for Digital Signal Processing" By Vinnie Falco and Bernd Porr

Official project location: <https://github.com/berndporr/iir1>

See Documentation.cpp for contact information, notes, and bibliography.

License: MIT License (<http://www.opensource.org/licenses/mit-license.php>) Copyright (c) 2009 by Vinnie Falco Copyright (c) 2011 by Bernd Porr

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"A Collection of Useful C++ Classes for Digital Signal Processing" By Vinnie Falco and Bernd Porr

Official project location: <https://github.com/berndporr/iir1>

See Documentation.txt for contact information, notes, and bibliography.

License: MIT License (<http://www.opensource.org/licenses/mit-license.php>) Copyright (c) 2009 by Vinnie Falco Copyright (c) 2011 by Bernd Porr

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"A Collection of Useful C++ Classes for Digital Signal Processing" By Vinnie Falco and Bernd Porr

Official project location: <https://github.com/berndporr/iir1>

See Documentation.cpp for contact information, notes, and bibliography.

License: MIT License (<http://www.opensource.org/licenses/mit-license.php>) Copyright (c) 2009 by Vinnie Falco Copyright (c) 2011 by Bernd Porr

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Describes a filter as a collection of poles and zeros along with normalization information to achieve a specified gain at a specified frequency. The poles and zeros may lie either in the s or the z plane.

5.1.2 Enumeration Type Documentation

5.1.2.1 Kind

```
enum Iir::Kind
```

Identifies the general class of filter

5.1.3 Function Documentation

5.1.3.1 `add()` [1/2]

```
template<class Td , class Ts >
void Iir::add (
    int samples,
    Td * dest,
    Ts const * src,
    int destSkip = 0,
    int srcSkip = 0 )
```

Utilities

These routines are handy for manipulating buffers of samples. Add src samples to dest, without clip or overflow checking.

5.1.3.2 `add()` [2/2]

```
template<typename Td , typename Ts >
void Iir::add (
    int channels,
    int samples,
    Td *const * dest,
    Ts const *const * src )
```

Multichannel add

5.1.3.3 `copy()`

```
template<typename Td , typename Ts >
void Iir::copy (
    int samples,
    Td * dest,
    Ts const * src,
    int destSkip = 0,
    int srcSkip = 0 )
```

Copy samples from src to dest, which may not overlap. Performs an implicit type conversion if Ts and Td are different (for example, float to double).

5.2 `Iir::Butterworth` Namespace Reference

Classes

- class [AnalogLowPass](#)
- class [AnalogLowShelf](#)
- struct [BandPass](#)
- struct [BandPassBase](#)
- struct [BandShelf](#)
- struct [BandShelfBase](#)
- struct [BandStop](#)

- struct [BandStopBase](#)
- struct [HighPass](#)
- struct [HighPassBase](#)
- struct [HighShelf](#)
- struct [HighShelfBase](#)
- struct [LowPass](#)
- struct [LowPassBase](#)
- struct [LowShelf](#)
- struct [LowShelfBase](#)

5.2.1 Detailed Description

Filters with [Butterworth](#) response characteristics

5.3 `lir::Chebyshev` Namespace Reference

Classes

- class [AnalogLowPass](#)
- class [AnalogLowShelf](#)
- struct [BandPass](#)
- struct [BandPassBase](#)
- struct [BandShelf](#)
- struct [BandShelfBase](#)
- struct [BandStop](#)
- struct [BandStopBase](#)
- struct [HighPass](#)
- struct [HighPassBase](#)
- struct [HighShelf](#)
- struct [HighShelfBase](#)
- struct [LowPass](#)
- struct [LowPassBase](#)
- struct [LowShelf](#)
- struct [LowShelfBase](#)

5.3.1 Detailed Description

Filters with Chebyshev response characteristics. The last parameter defines the passband ripple in decibel.

5.4 Iir::ChebyshevII Namespace Reference

Classes

- class [AnalogLowPass](#)
- class [AnalogLowShelf](#)
- struct [BandPass](#)
- struct [BandPassBase](#)
- struct [BandShelf](#)
- struct [BandShelfBase](#)
- struct [BandStop](#)
- struct [BandStopBase](#)
- struct [HighPass](#)
- struct [HighPassBase](#)
- struct [HighShelf](#)
- struct [HighShelfBase](#)
- struct [LowPass](#)
- struct [LowPassBase](#)
- struct [LowShelf](#)
- struct [LowShelfBase](#)

5.4.1 Detailed Description

Filters with [ChebyshevII](#) response characteristics. The last parameter defines the minimal stopband rejection requested. Generally there will be frequencies where the rejection is much better but this parameter guarantees that the rejection is at least as specified.

5.5 Iir::Custom Namespace Reference

Classes

- struct [OnePole](#)
- struct [SOSCascade](#)
- struct [TwoPole](#)

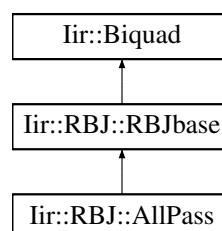
5.5.1 Detailed Description

Single pole, [Biquad](#) and cascade of Biquads with parameters allowing for directly setting the parameters.

6 Class Documentation

6.1 Iir::RBJ::AllPass Struct Reference

Inheritance diagram for Iir::RBJ::AllPass:



Additional Inherited Members

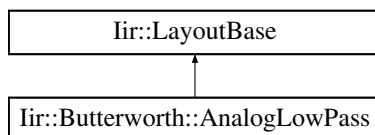
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.2 Iir::Butterworth::AnalogLowPass Class Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::AnalogLowPass:



6.2.1 Detailed Description

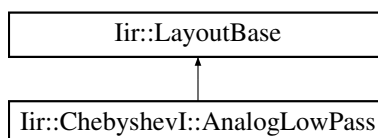
Half-band analog prototypes (s-plane)

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.3 Iir::ChebyshevI::AnalogLowPass Class Reference

Inheritance diagram for Iir::ChebyshevI::AnalogLowPass:

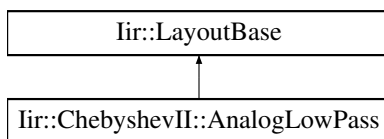


The documentation for this class was generated from the following files:

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.4 Iir::ChebyshevII::AnalogLowPass Class Reference

Inheritance diagram for Iir::ChebyshevII::AnalogLowPass:

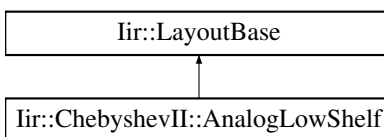


The documentation for this class was generated from the following files:

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.5 Iir::ChebyshevII::AnalogLowShelf Class Reference

Inheritance diagram for Iir::ChebyshevII::AnalogLowShelf:

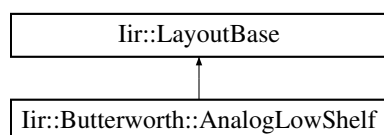


The documentation for this class was generated from the following files:

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.6 Iir::Butterworth::AnalogLowShelf Class Reference

Inheritance diagram for Iir::Butterworth::AnalogLowShelf:

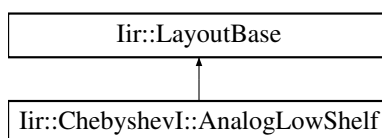


The documentation for this class was generated from the following files:

- iir/Butterworth.h
- iir/Butterworth.cpp

6.7 Iir::ChebyshevI::AnalogLowShelf Class Reference

Inheritance diagram for Iir::ChebyshevI::AnalogLowShelf:



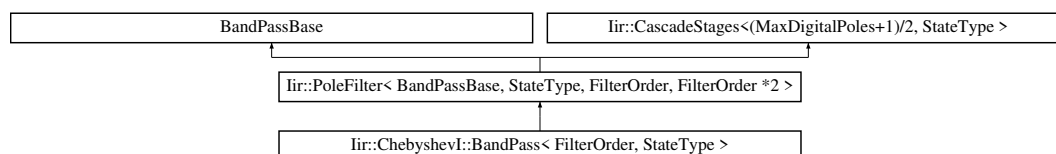
The documentation for this class was generated from the following files:

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.8 Iir::ChebyshevI::BandPass< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for Iir::ChebyshevI::BandPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double rippleDb)

6.8.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevI::BandPass< FilterOrder, StateType >
```

[ChebyshevI](#) bandpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.8.2 Member Function Documentation

6.8.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>widthFrequency</i>	Frequency with of the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

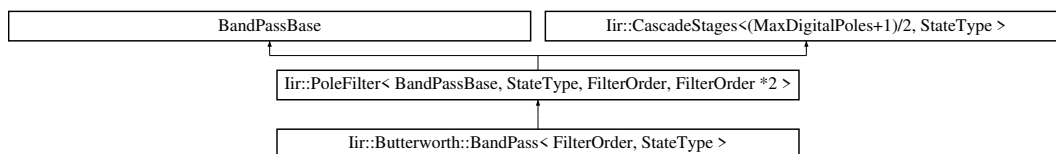
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.9 Iir::Butterworth::BandPass< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::BandPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency)

6.9.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::Butterworth::BandPass< FilterOrder, StateType >
```

[Butterworth](#) Bandpass filter.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.9.2 Member Function Documentation

6.9.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::BandPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency ) [inline]
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass

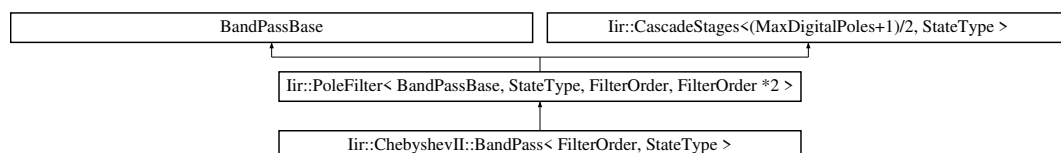
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.10 Iir::ChebyshevII::BandPass< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for Iir::ChebyshevII::BandPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double stopBandDb)

6.10.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevII::BandPass< FilterOrder, StateType >
```

[ChebyshevII](#) bandpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.10.2 Member Function Documentation**6.10.2.1 setup()**

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

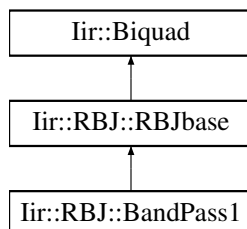
The documentation for this struct was generated from the following file:

- [iir/ChebyshevII.h](#)

6.11 Iir::RBJ::BandPass1 Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for [Iir::RBJ::BandPass1](#):

**Public Member Functions**

- void [setup](#) (double *sampleRate*, double *centerFrequency*, double *bandWidth*)

6.11.1 Detailed Description

Bandpass with constant skirt gain

6.11.2 Member Function Documentation

6.11.2.1 setup()

```
void Iir::RBJ::BandPass1::setup (
    double sampleRate,
    double centerFrequency,
    double bandWidth )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>bandWidth</i>	Bandwidth in octaves

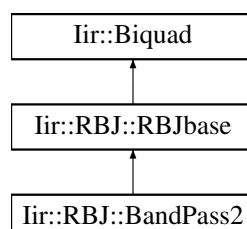
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.12 Iir::RBJ::BandPass2 Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::BandPass2:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double bandWidth)

6.12.1 Detailed Description

Bandpass with constant 0 dB peak gain

6.12.2 Member Function Documentation

6.12.2.1 setup()

```
void Iir::RBJ::BandPass2::setup (
    double sampleRate,
    double centerFrequency,
    double bandWidth )
```

Calculates the coefficients

Parameters

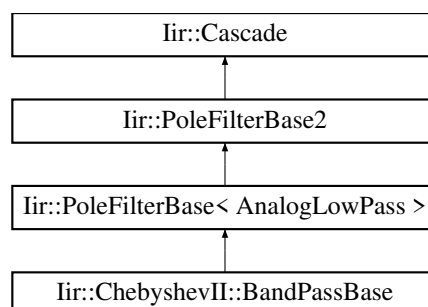
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>bandWidth</i>	Bandwidth in octaves

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

- iir/RBJ.h
- iir/RBJ.cpp

6.13 Iir::ChebyshevII::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandPassBase:



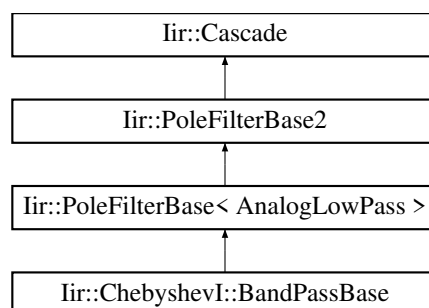
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.14 Iir::ChebyshevI::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandPassBase:



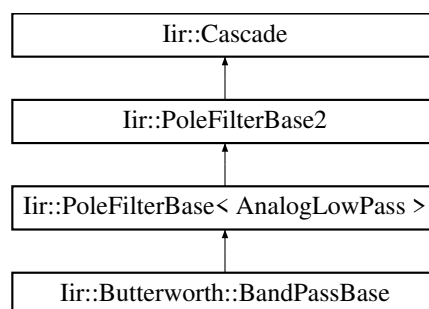
Additional Inherited Members

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

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.15 Iir::Butterworth::BandPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandPassBase:



Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.16 Iir::BandPassTransform Class Reference

```
#include <PoleFilter.h>
```


6.16.1 Detailed Description

low pass to band pass transform

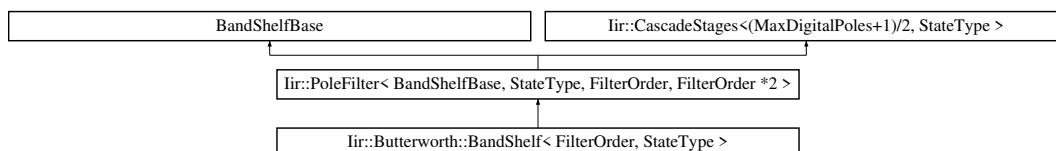
The documentation for this class was generated from the following files:

- iir/PoleFilter.h
- iir/PoleFilter.cpp

6.17 Iir::Butterworth::BandShelf< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::BandShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double gainDb)

6.17.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::Butterworth::BandShelf< FilterOrder, StateType >
```

[Butterworth](#) Bandshelf filter: it is a bandpass filter which amplifies at a specified gain in dB the frequencies in the passband.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.17.2 Member Function Documentation

6.17.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setup (
```

```
double sampleRate,
double centerFrequency,
double widthFrequency,
double gainDb ) [inline]
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the passband
<i>widthFrequency</i>	Width of the passband
<i>gainDb</i>	The gain in the passband

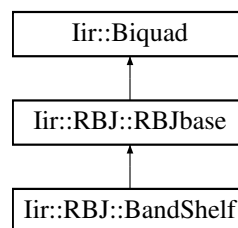
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.18 Iir::RBJ::BandShelf Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::BandShelf:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double gainDb, double bandWidth)

6.18.1 Detailed Description

Band shelf: 0db in the stopband and gainDb in the passband.

6.18.2 Member Function Documentation

6.18.2.1 setup()

```
void Iir::RBJ::BandShelf::setup (
    double sampleRate,
    double centerFrequency,
    double gainDb,
    double bandWidth )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	frequency
<i>gainDb</i>	Gain in the passband
<i>bandWidth</i>	Bandwidth in octaves

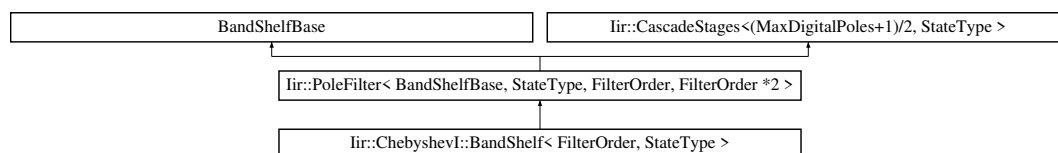
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.19 iir::ChebyshevI::BandShelf< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for iir::ChebyshevI::BandShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double gainDb, double rippleDb)

6.19.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct iir::ChebyshevI::BandShelf< FilterOrder, StateType >
```

[ChebyshevI](#) bandshelf filter. Specified gain in the passband. Otherwise 0 dB.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.19.2 Member Function Documentation

6.19.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void lir::ChebyshevI::BandShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the passband
<i>widthFrequency</i>	Width of the passband.
<i>gainDb</i>	Gain in the passband. The stopband has 0 dB.
<i>rippleDb</i>	Permitted ripples in dB in the passband.

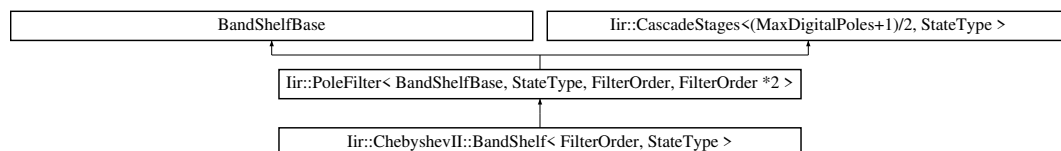
The documentation for this struct was generated from the following file:

- `iir/ChebyshevI.h`

6.20 `lir::ChebyshevII::BandShelf< FilterOrder, StateType >` Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for `lir::ChebyshevII::BandShelf< FilterOrder, StateType >`:



Public Member Functions

- void `setup` (double *sampleRate*, double *centerFrequency*, double *widthFrequency*, double *gainDb*, double *stopBandDb*)

6.20.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::ChebyshevII::BandShelf< FilterOrder, StateType >
```

[ChebyshevII](#) bandshelf filter. Bandpass with specified gain and 0 dB gain in the stopband.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.20.2 Member Function Documentation

6.20.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::BandShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double gainDb,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

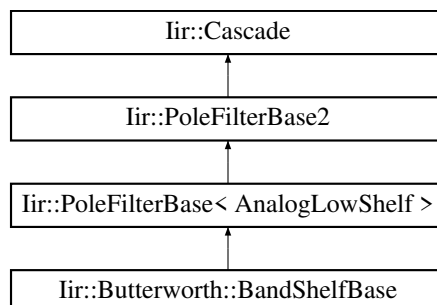
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>gainDb</i>	Gain in the passband. The stopband has always 0dB.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

- `iir/ChebyshevII.h`

6.21 `Iir::Butterworth::BandShelfBase` Struct Reference

Inheritance diagram for `Iir::Butterworth::BandShelfBase`:



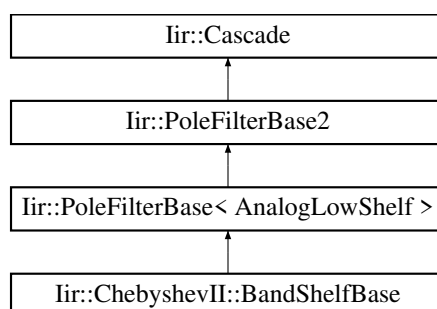
Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.22 Iir::ChebyshevII::BandShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandShelfBase:



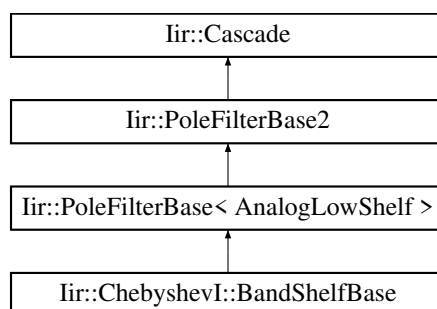
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.23 Iir::ChebyshevI::BandShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandShelfBase:



Additional Inherited Members

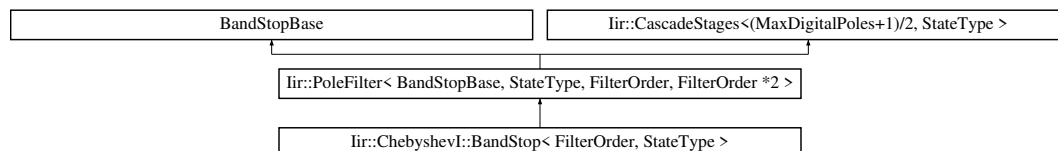
The documentation for this struct was generated from the following files:

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.24 `lir::ChebyshevI::BandStop< FilterOrder, StateType >` Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for `lir::ChebyshevI::BandStop< FilterOrder, StateType >`:



Public Member Functions

- void [setup](#) (double *sampleRate*, double *centerFrequency*, double *widthFrequency*, double *rippleDb*)

6.24.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::ChebyshevI::BandStop< FilterOrder, StateType >
```

[ChebyshevI](#) bandstop filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.24.2 Member Function Documentation

6.24.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void lir::ChebyshevI::BandStop< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the notch
<i>widthFrequency</i>	Frequency width of the notch
<i>rippleDb</i>	Permitted ripples in dB in the passband

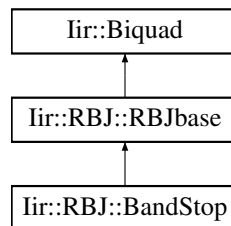
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.25 Iir::RBJ::BandStop Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::BandStop:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double bandWidth)

6.25.1 Detailed Description

Bandstop filter. Warning: the bandwidth might not be accurate for narrow notches.

6.25.2 Member Function Documentation

6.25.2.1 setup()

```
void Iir::RBJ::BandStop::setup (
    double sampleRate,
    double centerFrequency,
    double bandWidth )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandstop
<i>bandWidth</i>	Bandwidth in octaves

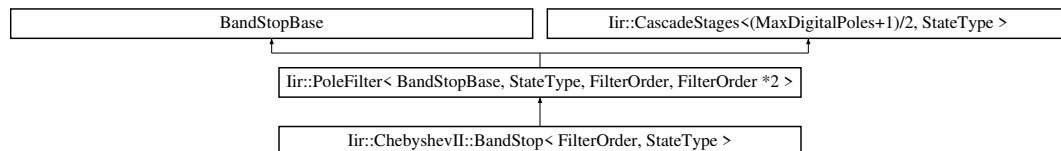
The documentation for this struct was generated from the following files:

- `iir/RBJ.h`
- `iir/RBJ.cpp`

6.26 `Iir::ChebyshevII::BandStop< FilterOrder, StateType >` Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for `Iir::ChebyshevII::BandStop< FilterOrder, StateType >`:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double stopBandDb)

6.26.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevII::BandStop< FilterOrder, StateType >
```

[ChebyshevII](#) bandstop filter.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.26.2 Member Function Documentation

6.26.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::BandStop< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

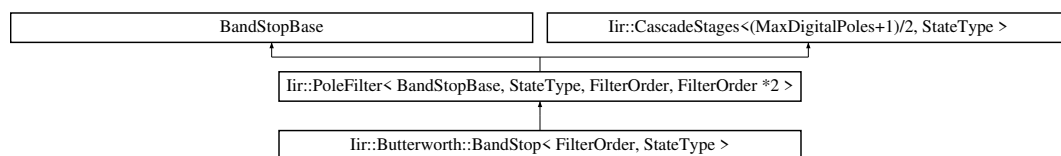
The documentation for this struct was generated from the following file:

- `iir/ChebyshevII.h`

6.27 `lir::Butterworth::BandStop< FilterOrder, StateType >` Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for `lir::Butterworth::BandStop< FilterOrder, StateType >`:



Public Member Functions

- `void setup` (double `sampleRate`, double `centerFrequency`, double `widthFrequency`)

6.27.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::Butterworth::BandStop< FilterOrder, StateType >
```

[Butterworth](#) Bandstop filter.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.27.2 Member Function Documentation

6.27.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::BandStop< FilterOrder, StateType >::setup (
    double sampleRate,
    double centerFrequency,
    double widthFrequency ) [inline]
```

Calculates the coefficients

Parameters

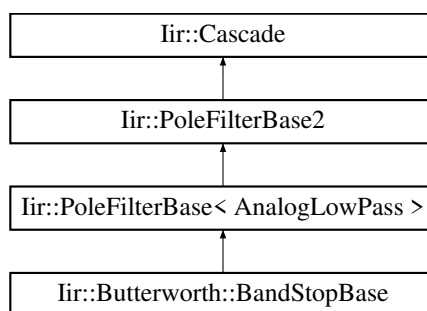
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop

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

- iir/Butterworth.h

6.28 Iir::Butterworth::BandStopBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandStopBase:



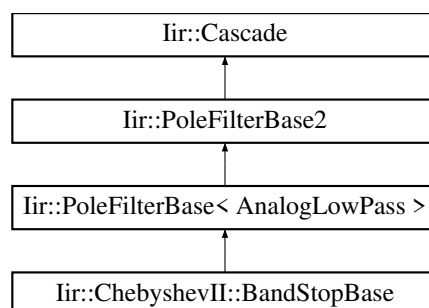
Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.29 Iir::ChebyshevII::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandStopBase:



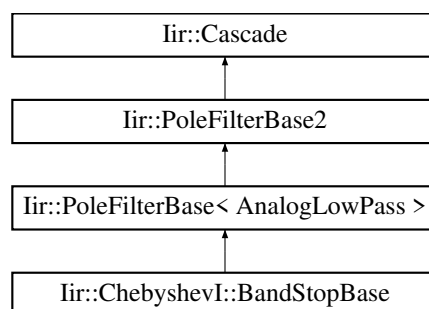
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.30 Iir::ChebyshevI::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandStopBase:



Additional Inherited Members

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

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.31 Iir::BandStopTransform Class Reference

```
#include <PoleFilter.h>
```

6.31.1 Detailed Description

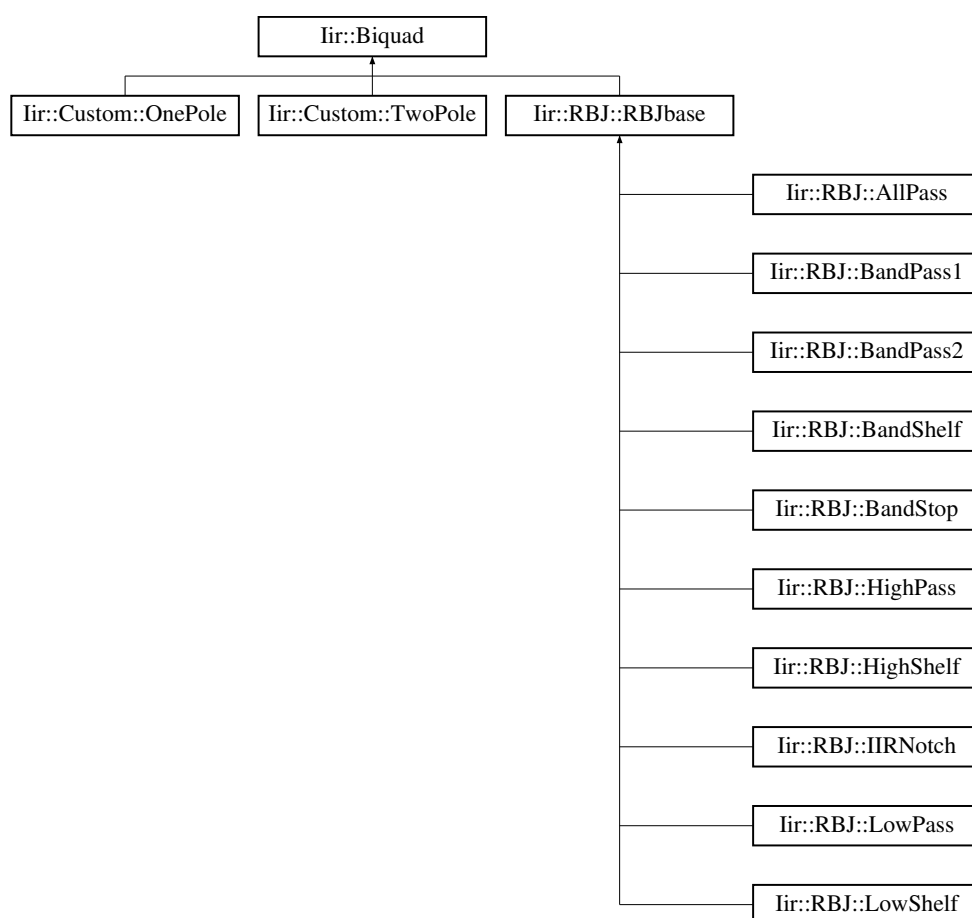
low pass to band stop transform

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

- iir/PoleFilter.h
- iir/PoleFilter.cpp

6.32 Iir::Biquad Class Reference

Inheritance diagram for Iir::Biquad:



Public Member Functions

- `complex_t response (double normalizedFrequency) const`
- `std::vector< PoleZeroPair > getPoleZeros () const`
- `double getA0 () const`
- `double getA1 () const`
- `double getA2 () const`
- `double getB0 () const`
- `double getB1 () const`
- `double getB2 () const`

- `template<class StateType , typename Sample >`
`Sample filter (Sample s, StateType &state) const`
- `void setCoefficients (double a0, double a1, double a2, double b0, double b1, double b2)`
- `void setOnePole (complex_t pole, complex_t zero)`
- `void setTwoPole (complex_t pole1, complex_t zero1, complex_t pole2, complex_t zero2)`
- `void setPoleZeroPair (const PoleZeroPair &pair)`
- `void setIdentity ()`
- `void applyScale (double scale)`

6.32.1 Member Function Documentation

6.32.1.1 applyScale()

```
void Iir::Biquad::applyScale (
    double scale )
```

Performs scaling operation on the FIR coefficients

Parameters

<i>scale</i>	Multplies the coefficients b0,b1,b2 with the scaling factor scale.
--------------	--

6.32.1.2 filter()

```
template<class StateType , typename Sample >
Sample Iir::Biquad::filter (
    Sample s,
    StateType & state ) const [inline]
```

Filter a sample with the coefficients provided here and the State provided as an argument.

Parameters

<i>s</i>	The sample to be filtered.
<i>state</i>	The Delay lines (instance of a state from State.h)

6.32.1.3 getA0()

```
double Iir::Biquad::getA0 ( ) const [inline]
```

Returns 1st IIR coefficient (usually one)

6.32.1.4 getA1()

```
double Iir::Biquad::getA1 ( ) const [inline]
```

Returns 2nd IIR coefficient

6.32.1.5 getA2()

```
double Iir::Biquad::getA2 ( ) const [inline]
```

Returns 3rd IIR coefficient

6.32.1.6 getB0()

```
double Iir::Biquad::getB0 ( ) const [inline]
```

Returns 1st FIR coefficient

6.32.1.7 getB1()

```
double Iir::Biquad::getB1 ( ) const [inline]
```

Returns 2nd FIR coefficient

6.32.1.8 getB2()

```
double Iir::Biquad::getB2 ( ) const [inline]
```

Returns 3rd FIR coefficient

6.32.1.9 getPoleZeros()

```
std::vector< PoleZeroPair > Iir::Biquad::getPoleZeros ( ) const
```

Returns the pole / zero Pairs as a vector

6.32.1.10 response()

```
complex_t Iir::Biquad::response (
    double normalizedFrequency ) const
```

Calculate filter response at the given normalized frequency.

Gets the frequency response of the [Biquad](#)

Parameters

<i>normalizedFrequency</i>	Normalised frequency (0 to 0.5)
----------------------------	---------------------------------

6.32.1.11 setCoefficients()

```
void Iir::Biquad::setCoefficients (
    double a0,
    double a1,
    double a2,
```

```
double b0,
double b1,
double b2 )
```

Sets all coefficients

Parameters

<i>a0</i>	1st IIR coefficient
<i>a1</i>	2nd IIR coefficient
<i>a2</i>	3rd IIR coefficient
<i>b0</i>	1st FIR coefficient
<i>b1</i>	2nd FIR coefficient
<i>b2</i>	3rd FIR coefficient

6.32.1.12 setIdentity()

```
void Iir::Biquad::setIdentity ( )
```

Sets the coefficients as pass through. (b0=1,a0=1, rest zero)

6.32.1.13 setOnePole()

```
void Iir::Biquad::setOnePole (
    complex_t pole,
    complex_t zero )
```

Sets one (real) pole and zero. Throws exception if imaginary components.

6.32.1.14 setPoleZeroPair()

```
void Iir::Biquad::setPoleZeroPair (
    const PoleZeroPair & pair ) [inline]
```

Sets a complex conjugate pair

6.32.1.15 setTwoPole()

```
void Iir::Biquad::setTwoPole (
    complex_t pole1,
    complex_t zero1,
    complex_t pole2,
    complex_t zero2 )
```

Sets two poles/zoes as a pair. Needs to be complex conjugate.

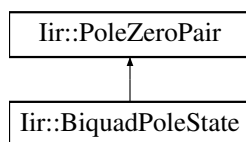
The documentation for this class was generated from the following files:

- iir/Biquad.h
- iir/Biquad.cpp

6.33 Iir::BiquadPoleState Struct Reference

```
#include <Biquad.h>
```

Inheritance diagram for Iir::BiquadPoleState:



6.33.1 Detailed Description

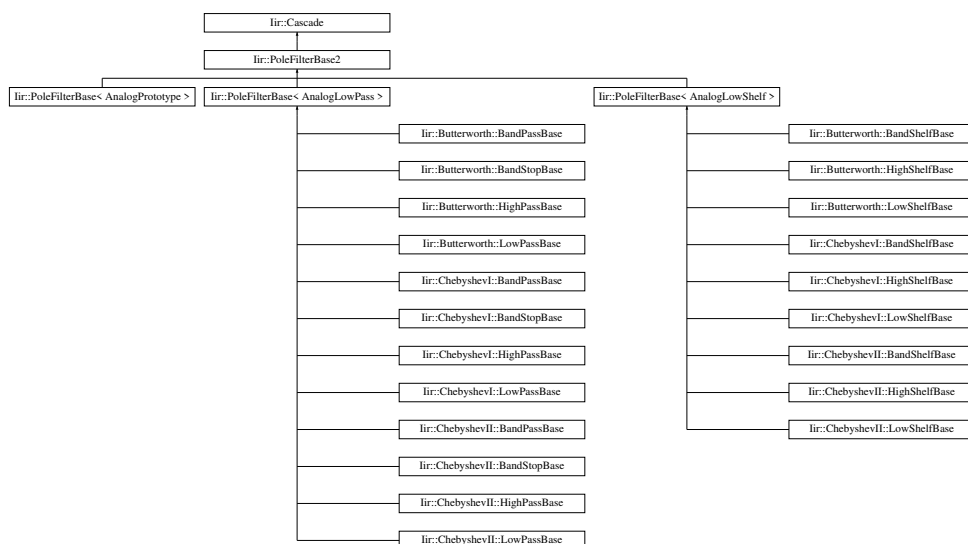
Expresses a biquad as a pair of pole/zeros, with gain values so that the coefficients can be reconstructed precisely.

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

- iir/Biquad.h
- iir/Biquad.cpp

6.34 Iir::Cascade Class Reference

Inheritance diagram for Iir::Cascade:



Classes

- struct [Storage](#)

Public Member Functions

- int [getNumStages](#) () const
- const [Biquad](#) & [operator\[\]](#) (int index)
- complex_t [response](#) (double normalizedFrequency) const
- std::vector< [PoleZeroPair](#) > [getPoleZeros](#) () const

6.34.1 Member Function Documentation

6.34.1.1 getNumStages()

```
int Iir::Cascade::getNumStages ( ) const [inline]
```

Returns the number of Biquads kept here

6.34.1.2 getPoleZeros()

```
std::vector< PoleZeroPair > Iir::Cascade::getPoleZeros ( ) const
```

Returns a vector with all pole/zero pairs of the whole Biquad cascade

6.34.1.3 operator[]()

```
const Biquad& Iir::Cascade::operator[] (
    int index ) [inline]
```

returns the points to a biquad

6.34.1.4 response()

```
complex_t Iir::Cascade::response (
    double normalizedFrequency ) const
```

Calculate filter response at the given normalized frequency

Parameters

<i>normalizedFrequency</i>	Frequency from 0 to 0.5 (Nyquist)
----------------------------	-----------------------------------

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

- iir/Cascade.h
- iir/Cascade.cpp

6.35 Iir::CascadeStages< MaxStages, StateType > Class Template Reference

```
#include <Cascade.h>
```

Public Member Functions

- void [reset](#) ()
- void [setup](#) (const double(&sosCoefficients)[MaxStages][6])
- template<typename Sample >
Sample [filter](#) (const Sample in)

6.35.1 Detailed Description

```
template<int MaxStages, class StateType>
class lir::CascadeStages< MaxStages, StateType >
```

Storage for [Cascade](#): This holds a chain of 2nd order filters with its coefficients.

6.35.2 Member Function Documentation

6.35.2.1 filter()

```
template<int MaxStages, class StateType>
template<typename Sample >
Sample lir::CascadeStages< MaxStages, StateType >::filter (
    const Sample in ) [inline]
```

Filters one sample through the whole chain of biquads and return the result

Parameters

<i>in</i>	Sample to be filtered
-----------	-----------------------

6.35.2.2 reset()

```
template<int MaxStages, class StateType>
void lir::CascadeStages< MaxStages, StateType >::reset ( ) [inline]
```

Resets all biquads (i.e. the delay lines but not the coefficients)

6.35.2.3 setup()

```
template<int MaxStages, class StateType>
void lir::CascadeStages< MaxStages, StateType >::setup (
    const double(&) sosCoefficients[MaxStages][6] ) [inline]
```

Sets the coefficients of the whole chain of biquads.

Parameters

<i>sosCoefficients</i>	2D array in Python style sos ordering: 0-2: FIR, 3-5: IIR coeff.
------------------------	--

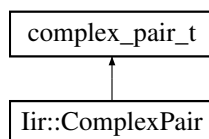
The documentation for this class was generated from the following file:

- iir/Cascade.h

6.36 Iir::ComplexPair Struct Reference

```
#include <Types.h>
```

Inheritance diagram for Iir::ComplexPair:



6.36.1 Detailed Description

A conjugate or real pair

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

- iir/Types.h

6.37 Iir::DirectFormI Class Reference

```
#include <State.h>
```

6.37.1 Detailed Description

State for applying a second order section to a sample using Direct Form I

Difference equation:

$$y[n] = (b0/a0)*x[n] + (b1/a0)*x[n-1] + (b2/a0)*x[n-2]$$

- $(a1/a0)*y[n-1] - (a2/a0)*y[n-2]$

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

- iir/State.h

6.38 Iir::DirectFormII Class Reference

```
#include <State.h>
```

6.38.1 Detailed Description

State for applying a second order section to a sample using Direct Form II

Difference equation:

$$v[n] = x[n] - (a1/a0)*v[n-1] - (a2/a0)*v[n-2] \quad y(n) = (b0/a0)*v[n] + (b1/a0)*v[n-1] + (b2/a0)*v[n-2]$$

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

- iir/State.h

6.39 Iir::EnvelopeFollower< Channels, Value > Class Template Reference

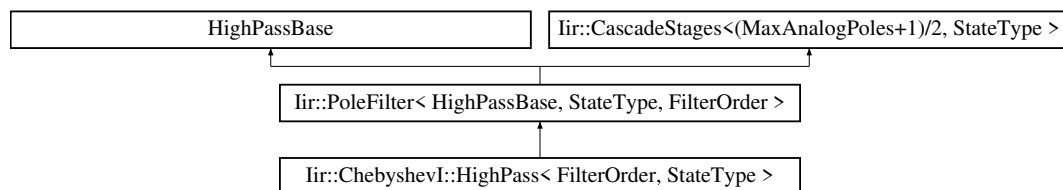
The documentation for this class was generated from the following file:

- iir/Utilities.h

6.40 Iir::ChebyshevI::HighPass< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for Iir::ChebyshevI::HighPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double rippleDb)

6.40.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevI::HighPass< FilterOrder, StateType >
```

[ChebyshevI](#) highpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.40.2 Member Function Documentation

6.40.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>rippleDb</i>	Permitted ripples in dB in the passband

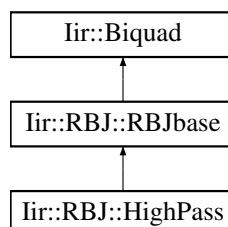
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.41 Iir::RBJ::HighPass Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::HighPass:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double q=(1/sqrt(2)))

6.41.1 Detailed Description

Highpass.

6.41.2 Member Function Documentation

6.41.2.1 setup()

```
void Iir::RBJ::HighPass::setup (
    double sampleRate,
    double cutoffFrequency,
    double q = (1/sqrt(2)) )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>q</i>	Q factor determines the resonance peak at the cutoff.

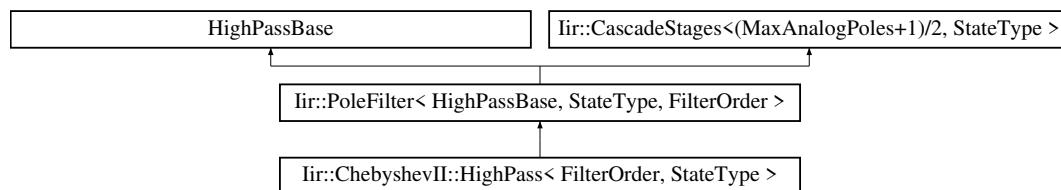
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.42 Iir::ChebyshevII::HighPass< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for Iir::ChebyshevII::HighPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double stopBandDb)

6.42.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevII::HighPass< FilterOrder, StateType >
```

[ChebyshevII](#) highpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.42.2 Member Function Documentation

6.42.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::HighPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

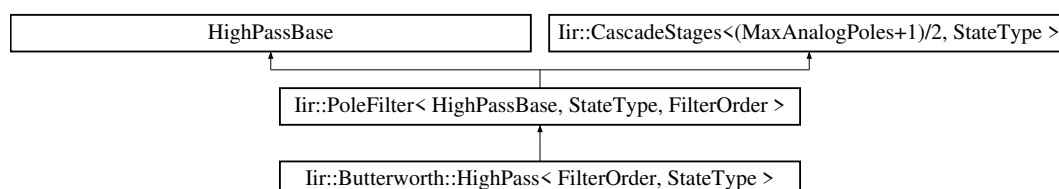
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.43 Iir::Butterworth::HighPass< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::HighPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency)

6.43.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>  
struct ltr::Butterworth::HighPass< FilterOrder, StateType >
```

[Butterworth](#) Highpass filter.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>State Type</i>	The filter topology: DirectFormI , DirectFormII , ...

6.43.2 Member Function Documentation

6.43.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::HighPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency ) [inline]
```

Calculates the coefficients

Parameters

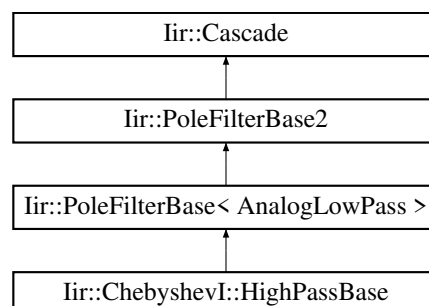
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff

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

- iir/Butterworth.h

6.44 Iir::ChebyshevI::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighPassBase:



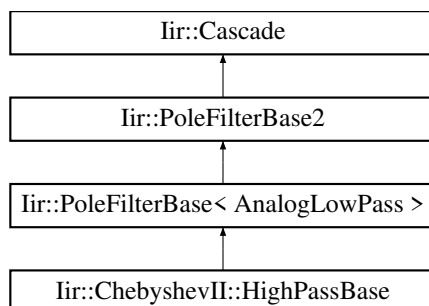
Additional Inherited Members

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

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.45 Iir::ChebyshevII::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighPassBase:



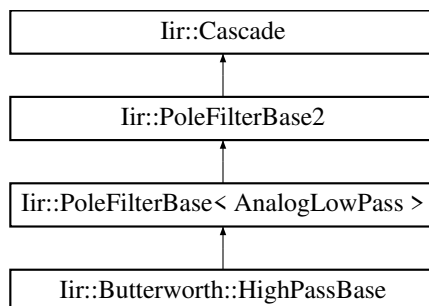
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.46 Iir::Butterworth::HighPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighPassBase:



Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.47 Iir::HighPassTransform Class Reference

```
#include <PoleFilter.h>
```

6.47.1 Detailed Description

low pass to high pass

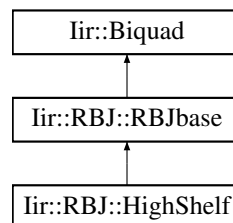
The documentation for this class was generated from the following files:

- iir/PoleFilter.h
- iir/PoleFilter.cpp

6.48 Iir::RBJ::HighShelf Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::HighShelf:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double gainDb, double shelfSlope=1)

6.48.1 Detailed Description

High shelf: 0db in the stopband and gainDb in the passband.

6.48.2 Member Function Documentation

6.48.2.1 setup()

```
void Iir::RBJ::HighShelf::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double shelfSlope = 1 )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>gainDb</i>	Gain in the passband
<i>shelfSlope</i>	Slope between stop/passband. 1 = as steep as it can.

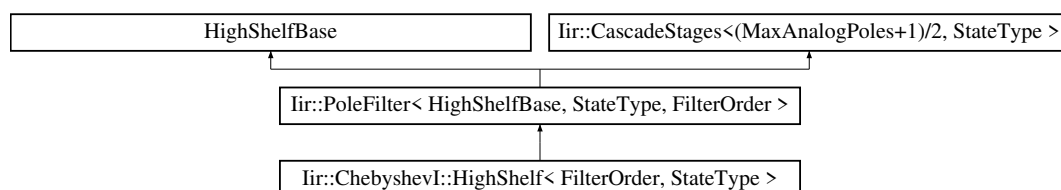
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.49 Iir::ChebyshevI::HighShelf< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for Iir::ChebyshevI::HighShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double *sampleRate*, double *cutoffFrequency*, double *gainDb*, double *rippleDb*)

6.49.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevI::HighShelf< FilterOrder, StateType >
```

[ChebyshevI](#) high shelf filter. Specified gain in the passband. Otherwise 0 dB.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.49.2 Member Function Documentation

6.49.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

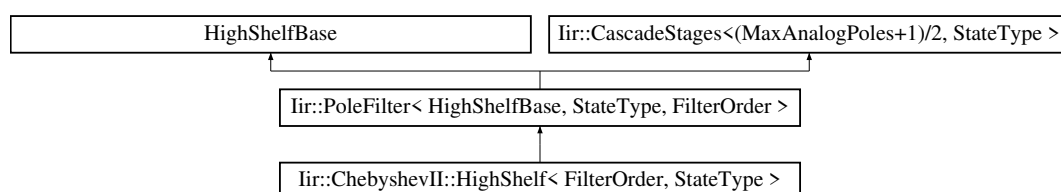
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.50 Iir::ChebyshevII::HighShelf< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for Iir::ChebyshevII::HighShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double gainDb, double stopBandDb)

6.50.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevII::HighShelf< FilterOrder, StateType >
```

[ChebyshevII](#) high shelf filter. Specified gain in the passband and 0dB in the stopband.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.50.2 Member Function Documentation

6.50.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::HighShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

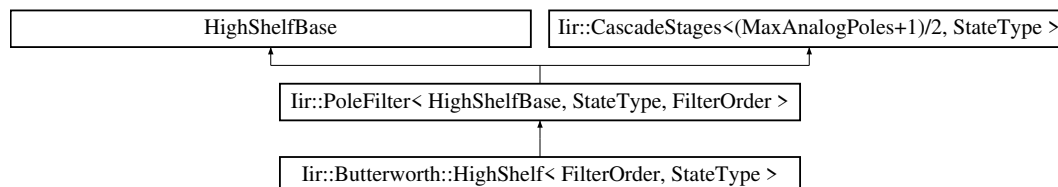
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.51 Iir::Butterworth::HighShelf< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::HighShelf< FilterOrder, StateType >:



Public Member Functions

- void `setup` (double sampleRate, double cutoffFrequency, double gainDb)

6.51.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::Butterworth::HighShelf< FilterOrder, StateType >
```

Butterworth high shelf filter. Above the cutoff the filter has a specified gain and below it has 0 dB.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.51.2 Member Function Documentation

6.51.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb ) [inline]
```

Calculates the coefficients

Parameters

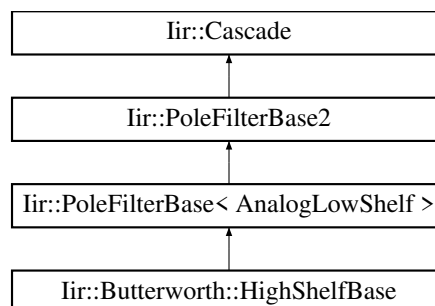
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff
<i>gainDb</i>	Gain in dB of the filter in the passband

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

- iir/Butterworth.h

6.52 Iir::Butterworth::HighShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighShelfBase:



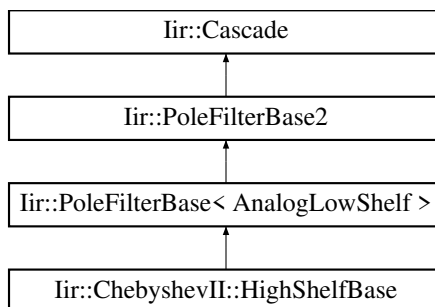
Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.53 Iir::ChebyshevII::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighShelfBase:



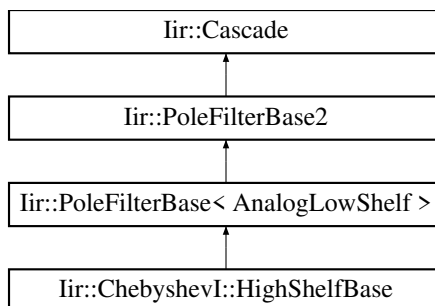
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.54 Iir::ChebyshevI::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighShelfBase:



Additional Inherited Members

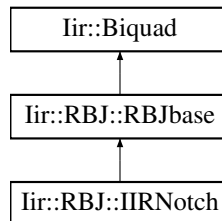
The documentation for this struct was generated from the following files:

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.55 Iir::RBJ::IIRNotch Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::IIRNotch:



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double q_factor=10)

6.55.1 Detailed Description

Bandstop with Q factor: the higher the Q factor the more narrow is the notch. However, a narrow notch has a long impulse response (= ringing) and numerical problems might prevent perfect damping. Practical values of the Q factor are about $Q = 10$ to 20 . In terms of the design the Q factor defines the radius of the poles as $r = \exp(-\pi * (\text{centerFrequency}/\text{sampleRate})/q_factor)$ whereas the angles of the poles/zeros define the bandstop frequency. The higher Q the closer r moves towards the unit circle.

6.55.2 Member Function Documentation

6.55.2.1 setup()

```
void Iir::RBJ::IIRNotch::setup (
    double sampleRate,
    double centerFrequency,
    double q_factor = 10 )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the notch
<i>q_factor</i>	Q factor of the notch (1 to ~20)

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

- iir/RBJ.h
- iir/RBJ.cpp

6.56 Iir::Layout< MaxPoles > Class Template Reference

```
#include <Layout.h>
```

6.56.1 Detailed Description

```
template<int MaxPoles>
class Iir::Layout< MaxPoles >
```

Storage for [Layout](#)

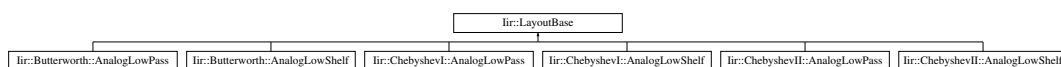
The documentation for this class was generated from the following file:

- iir/Layout.h

6.57 Iir::LayoutBase Class Reference

```
#include <Layout.h>
```

Inheritance diagram for Iir::LayoutBase:



6.57.1 Detailed Description

Base uses pointers to reduce template instantiations

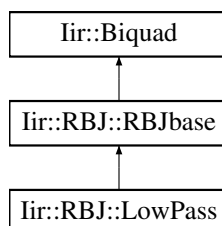
The documentation for this class was generated from the following file:

- iir/Layout.h

6.58 Iir::RBJ::LowPass Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::LowPass:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double q=(1/sqrt(2)))

6.58.1 Detailed Description

Lowpass.

6.58.2 Member Function Documentation

6.58.2.1 setup()

```
void Iir::RBJ::LowPass::setup (
    double sampleRate,
    double cutoffFrequency,
    double q = (1/sqrt(2)) )
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>q</i>	Q factor determines the resonance peak at the cutoff.

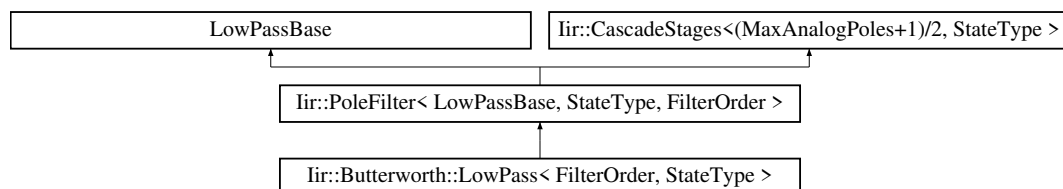
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.59 Iir::Butterworth::LowPass< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::LowPass< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency)

6.59.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::Butterworth::LowPass< FilterOrder, StateType >
```

Butterworth Lowpass filter.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.59.2 Member Function Documentation

6.59.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::Butterworth::LowPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency ) [inline]
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff

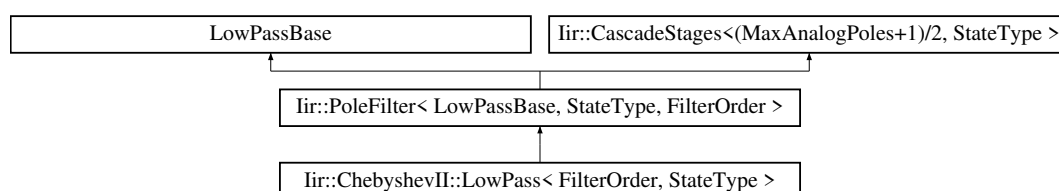
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.60 Iir::ChebyshevII::LowPass< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for Iir::ChebyshevII::LowPass< FilterOrder, StateType >:



Public Member Functions

- void `setup` (double `sampleRate`, double `cutoffFrequency`, double `stopBandDb`)

6.60.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct iir::ChebyshevI::LowPass< FilterOrder, StateType >
```

`ChebyshevI` lowpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: <code>DirectFormI</code> , <code>DirectFormII</code> , ...

6.60.2 Member Function Documentation

6.60.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

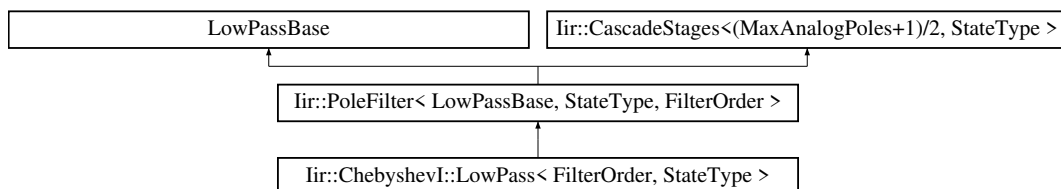
The documentation for this struct was generated from the following file:

- `iir/ChebyshevI.h`

6.61 `iir::ChebyshevI::LowPass< FilterOrder, StateType >` Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for `iir::ChebyshevI::LowPass< FilterOrder, StateType >`:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double rippleDb)

6.61.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevI::LowPass< FilterOrder, StateType >
```

[ChebyshevI](#) lowpass filter

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.61.2 Member Function Documentation

6.61.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

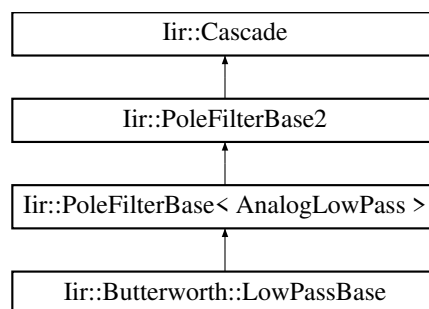
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>rippleDb</i>	Permitted ripples in dB in the passband

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

- `iir/ChebyshevI.h`

6.62 Iir::Butterworth::LowPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowPassBase:



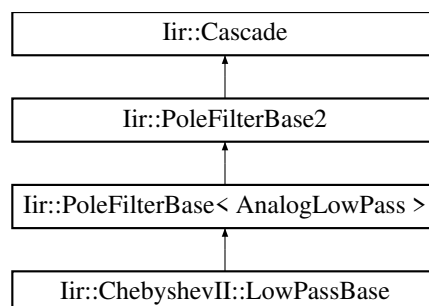
Additional Inherited Members

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

- iir/Butterworth.h
- iir/Butterworth.cpp

6.63 Iir::ChebyshevII::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowPassBase:



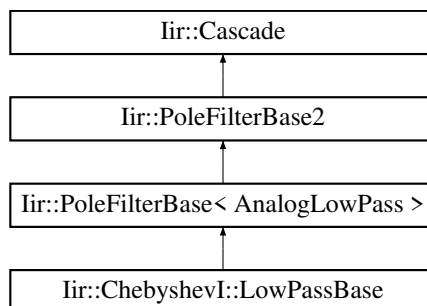
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.64 Iir::ChebyshevI::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowPassBase:



Additional Inherited Members

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

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.65 Iir::LowPassTransform Class Reference

```
#include <PoleFilter.h>
```

6.65.1 Detailed Description

s-plane to z-plane transforms

For pole filters, an analog prototype is created via placement of poles and zeros in the s-plane. The analog prototype is either a halfband low pass or a halfband low shelf. The poles, zeros, and normalization parameters are transformed into the z-plane using variants of the bilinear transformation. low pass to low pass

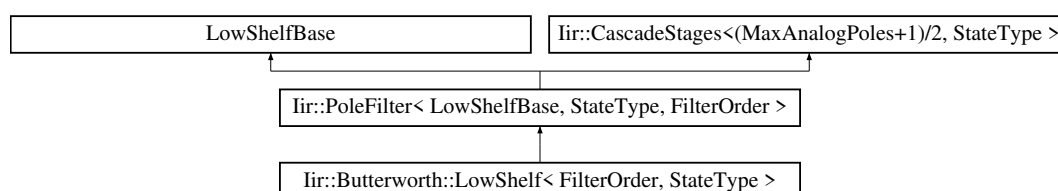
The documentation for this class was generated from the following files:

- iir/PoleFilter.h
- iir/PoleFilter.cpp

6.66 Iir::Butterworth::LowShelf< FilterOrder, StateType > Struct Template Reference

```
#include <Butterworth.h>
```

Inheritance diagram for Iir::Butterworth::LowShelf< FilterOrder, StateType >:



Public Member Functions

- void `setup` (double `sampleRate`, double `cutoffFrequency`, double `gainDb`)

6.66.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::Butterworth::LowShelf< FilterOrder, StateType >
```

`Butterworth` low shelf filter: below the cutoff it has a specified gain and above the cutoff the gain is 0 dB.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: <code>DirectFormI</code> , <code>DirectFormII</code> , ...

6.66.2 Member Function Documentation

6.66.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void lir::Butterworth::LowShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb ) [inline]
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff
<i>gainDb</i>	Gain in dB of the filter in the passband

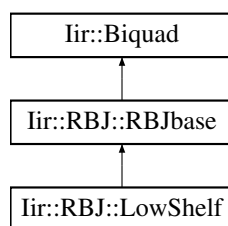
The documentation for this struct was generated from the following file:

- `iir/Butterworth.h`

6.67 `lir::RBJ::LowShelf` Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for `lir::RBJ::LowShelf`:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double gainDb, double shelfSlope=1)

6.67.1 Detailed Description

Low shelf: 0db in the stopband and gainDb in the passband.

6.67.2 Member Function Documentation

6.67.2.1 setup()

```

void Iir::RBJ::LowShelf::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double shelfSlope = 1 )
  
```

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>gainDb</i>	Gain in the passband
<i>shelfSlope</i>	Slope between stop/passband. 1 = as steep as it can.

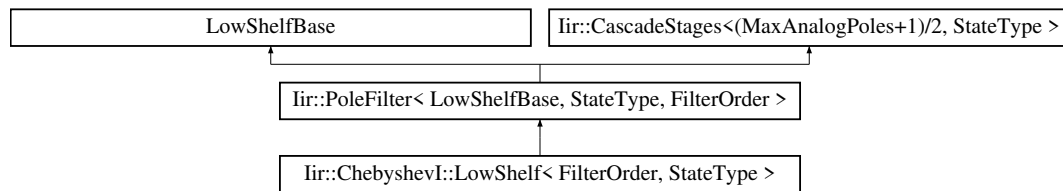
The documentation for this struct was generated from the following files:

- iir/RBJ.h
- iir/RBJ.cpp

6.68 Iir::ChebyshevI::LowShelf< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevI.h>
```

Inheritance diagram for Iir::ChebyshevI::LowShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double gainDb, double rippleDb)

6.68.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::ChebyshevI::LowShelf< FilterOrder, StateType >
```

[ChebyshevI](#) low shelf filter. Specified gain in the passband. Otherwise 0 dB.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.68.2 Member Function Documentation

6.68.2.1 `setup()`

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::LowShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

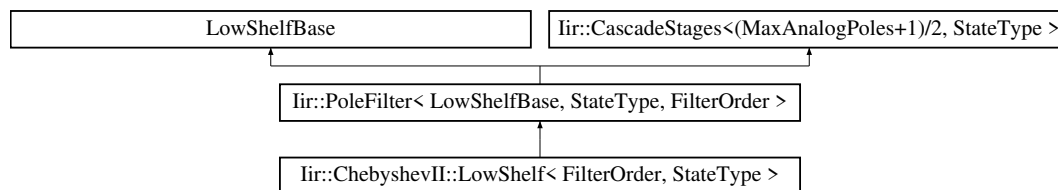
The documentation for this struct was generated from the following file:

- `iir/ChebyshevI.h`

6.69 Iir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference

```
#include <ChebyshevII.h>
```

Inheritance diagram for Iir::ChebyshevII::LowShelf< FilterOrder, StateType >:



Public Member Functions

- void [setup](#) (double sampleRate, double cutoffFrequency, double gainDb, double stopBandDb)

6.69.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct Iir::ChebyshevII::LowShelf< FilterOrder, StateType >
```

[ChebyshevII](#) low shelf filter. Specified gain in the passband and 0dB in the stopband.

Parameters

<i>FilterOrder</i>	The order of the filter.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.69.2 Member Function Documentation

6.69.2.1 setup()

```
template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::LowShelf< FilterOrder, StateType >::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

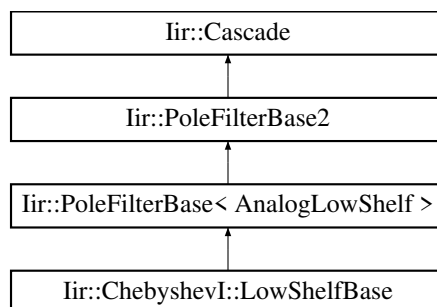
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

- iir/ChebyshevI.h

6.70 Iir::ChebyshevI::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowShelfBase:



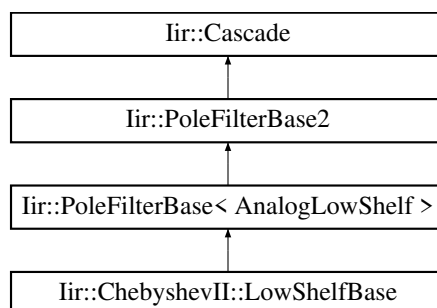
Additional Inherited Members

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

- iir/ChebyshevI.h
- iir/ChebyshevI.cpp

6.71 Iir::ChebyshevII::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowShelfBase:



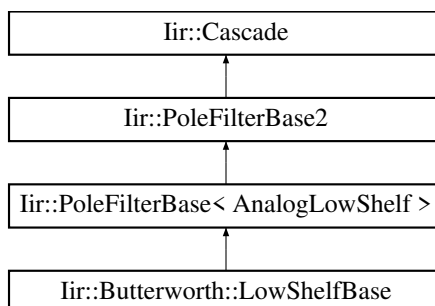
Additional Inherited Members

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

- iir/ChebyshevII.h
- iir/ChebyshevII.cpp

6.72 Iir::Butterworth::LowShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowShelfBase:



Additional Inherited Members

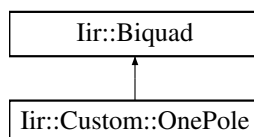
The documentation for this struct was generated from the following files:

- iir/Butterworth.h
- iir/Butterworth.cpp

6.73 Iir::Custom::OnePole Struct Reference

```
#include <Custom.h>
```

Inheritance diagram for Iir::Custom::OnePole:



Additional Inherited Members

6.73.1 Detailed Description

Setting up a filter with with one real pole, real zero and scale it by the scale factor

Parameters

<i>scale</i>	Scale the FIR coefficients by this factor
<i>pole</i>	Position of the pole on the real axis
<i>zero</i>	Position of the zero on the real axis

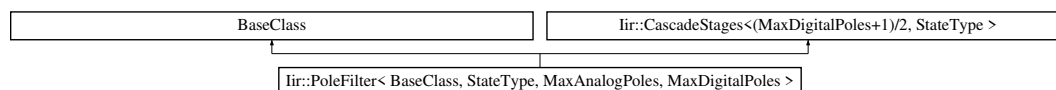
The documentation for this struct was generated from the following files:

- iir/Custom.h
- iir/Custom.cpp

6.74 Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > Struct Template Reference

```
#include <PoleFilter.h>
```

Inheritance diagram for Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >:



Additional Inherited Members

6.74.1 Detailed Description

```
template<class BaseClass, class StateType, int MaxAnalogPoles, int MaxDigitalPoles = MaxAnalogPoles>
struct Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >
```

Storage for pole filters

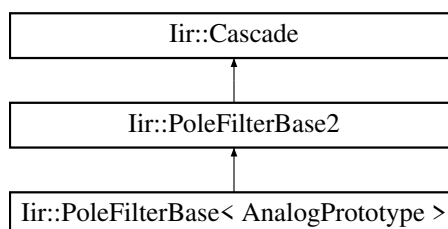
The documentation for this struct was generated from the following file:

- iir/PoleFilter.h

6.75 Iir::PoleFilterBase< AnalogPrototype > Class Template Reference

```
#include <PoleFilter.h>
```

Inheritance diagram for Iir::PoleFilterBase< AnalogPrototype >:



Additional Inherited Members

6.75.1 Detailed Description

```
template<class AnalogPrototype>
class Iir::PoleFilterBase< AnalogPrototype >
```

Serves a container to hold the analog prototype and the digital pole/zero layout.

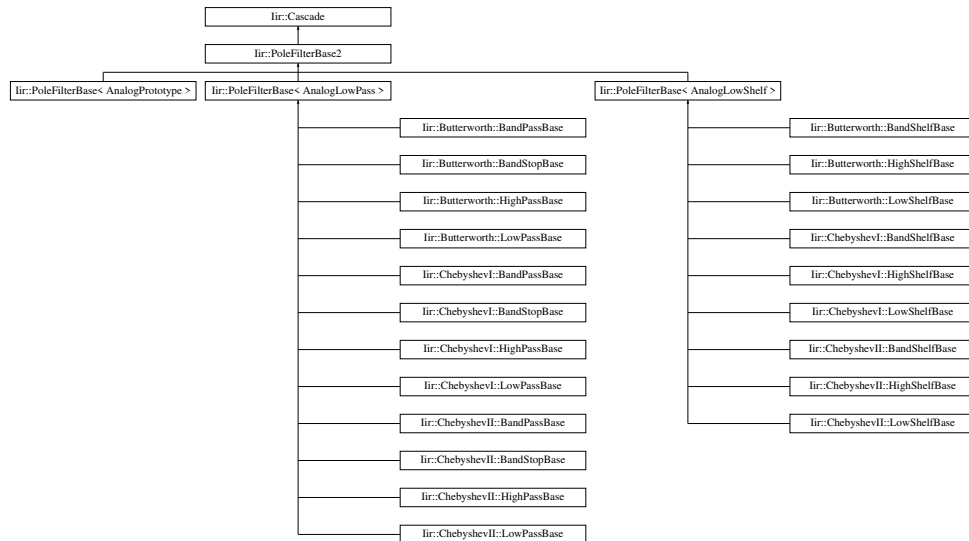
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.76 Iir::PoleFilterBase2 Class Reference

```
#include <PoleFilter.h>
```

Inheritance diagram for Iir::PoleFilterBase2:



Additional Inherited Members

6.76.1 Detailed Description

Factored implementations to reduce template instantiations

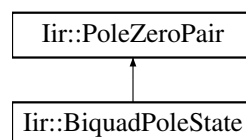
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.77 Iir::PoleZeroPair Struct Reference

```
#include <Types.h>
```

Inheritance diagram for Iir::PoleZeroPair:



6.77.1 Detailed Description

A pair of pole/zeros. This fits in a biquad (but is missing the gain)

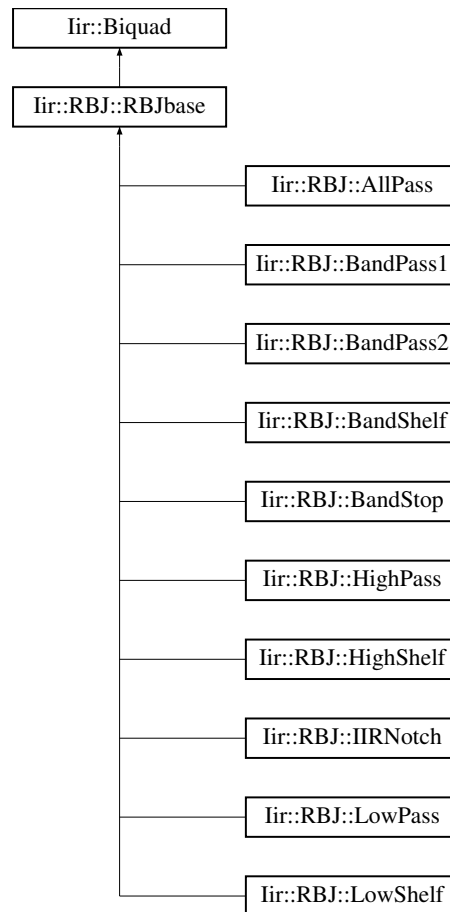
The documentation for this struct was generated from the following file:

- iir/Types.h

6.78 Iir::RBJ::RBJbase Struct Reference

```
#include <RBJ.h>
```

Inheritance diagram for Iir::RBJ::RBJbase:



Public Member Functions

- template<typename Sample >
Sample [filter](#) (Sample s)
filter operation
- void [reset](#) ()
resets the delay lines to zero
- const [DirectFormI](#) & [getState](#) ()
gets the delay lines (=state) of the filter

6.78.1 Detailed Description

The base class of all RBJ filters

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

- iir/RBJ.h

6.79 Iir::SlopeDetector< Channels, Value > Class Template Reference

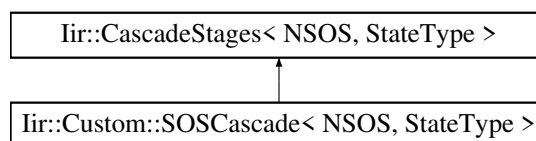
The documentation for this class was generated from the following file:

- iir/Utilities.h

6.80 Iir::Custom::SOSCascade< NSOS, StateType > Struct Template Reference

```
#include <Custom.h>
```

Inheritance diagram for Iir::Custom::SOSCascade< NSOS, StateType >:



Public Member Functions

- void [setup](#) (const double(& sosCoefficients)[NSOS][6])

6.80.1 Detailed Description

```
template<int NSOS, class StateType = DirectFormII>
struct Iir::Custom::SOSCascade< NSOS, StateType >
```

A custom cascade of 2nd order (SOS / biquads) filters.

Parameters

<i>NSOS</i>	The number of 2nd order filters / biquads.
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.80.2 Member Function Documentation

6.80.2.1 setup()

```
template<int NSOS, class StateType = DirectFormII>
void Iir::Custom::SOSCascade< NSOS, StateType >::setup (
    const double(&) sosCoefficients[NSOS][6] ) [inline]
```

Python scipy.signal-friendly setting of coefficients. Sets the coefficients of the whole chain of biquads / SOS. The argument is a 2D array where the 1st dimension holds an array of 2nd order biquad / SOS coefficients. The six SOS coefficients are ordered "Python" style with first the FIR coefficients (B) and then the IIR coefficients (A). The 2D const double array needs to have exactly the size [NSOS][6].

Parameters

<i>sosCoefficients</i>	2D array Python style sos[NSOS][6]. Indexing: 0-2: FIR-, 3-5: IIR-coefficients.
------------------------	---

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

- iir/Custom.h

6.81 Iir::Cascade::Storage Struct Reference

```
#include <Cascade.h>
```

Public Member Functions

- [Storage](#) (int maxStages_, [Biquad](#) *stageArray_)

6.81.1 Detailed Description

Pointer to an array of Biquads

6.81.2 Constructor & Destructor Documentation

6.81.2.1 Storage()

```
Iir::Cascade::Storage::Storage (
    int maxStages_,
    Biquad * stageArray_ ) [inline]
```

Constructor which receives the pointer to the [Biquad](#) array and the number of Biquads

Parameters

<i>max↔ Stages_</i>	Number of biquads
<i>stage↔ Array_</i>	The array of the Biquads

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

- iir/Cascade.h

6.82 Iir::TransposedDirectFormII Class Reference

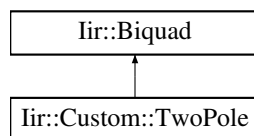
The documentation for this class was generated from the following file:

- iir/State.h

6.83 Iir::Custom::TwoPole Struct Reference

```
#include <Custom.h>
```

Inheritance diagram for Iir::Custom::TwoPole:



Additional Inherited Members

6.83.1 Detailed Description

Set a pole/zero pair in polar coordinates and scale the FIR filter coefficients

Parameters

<i>poleRho</i>	Radius of the pole
<i>poleTheta</i>	Angle of the pole
<i>zeroRho</i>	Radius of the zero
<i>zeroTheta</i>	Angle of the zero

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

- iir/Custom.h
- iir/Custom.cpp

Index

- add
 - [lir, 14](#)
- applyScale
 - [lir::Biquad, 39](#)
- copy
 - [lir, 14](#)
- filter
 - [lir::Biquad, 39](#)
 - [lir::CascadeStages, 44](#)
- getA0
 - [lir::Biquad, 39](#)
- getA1
 - [lir::Biquad, 39](#)
- getA2
 - [lir::Biquad, 39](#)
- getB0
 - [lir::Biquad, 40](#)
- getB1
 - [lir::Biquad, 40](#)
- getB2
 - [lir::Biquad, 40](#)
- getNumStages
 - [lir::Cascade, 43](#)
- getPoleZeros
 - [lir::Biquad, 40](#)
 - [lir::Cascade, 43](#)
- [lir, 11](#)
 - [add, 14](#)
 - [copy, 14](#)
 - [Kind, 13](#)
- [lir::BandPassTransform, 25](#)
- [lir::BandStopTransform, 37](#)
- [lir::Biquad, 38](#)
 - [applyScale, 39](#)
 - [filter, 39](#)
 - [getA0, 39](#)
 - [getA1, 39](#)
 - [getA2, 39](#)
 - [getB0, 40](#)
 - [getB1, 40](#)
 - [getB2, 40](#)
 - [getPoleZeros, 40](#)
 - [response, 40](#)
 - [setCoefficients, 40](#)
 - [setIdentity, 41](#)
 - [setOnePole, 41](#)
 - [setPoleZeroPair, 41](#)
 - [setTwoPole, 41](#)
- [lir::BiquadPoleState, 42](#)
- [lir::Butterworth, 14](#)
- [lir::Butterworth::AnalogLowPass, 17](#)
- [lir::Butterworth::AnalogLowShelf, 18](#)
- [lir::Butterworth::BandPass](#)
 - [setup, 21](#)
- [lir::Butterworth::BandPass< FilterOrder, StateType >, 20](#)
- [lir::Butterworth::BandPassBase, 25](#)
- [lir::Butterworth::BandShelf](#)
 - [setup, 26](#)
- [lir::Butterworth::BandShelf< FilterOrder, StateType >, 26](#)
- [lir::Butterworth::BandShelfBase, 30](#)
- [lir::Butterworth::BandStop](#)
 - [setup, 35](#)
- [lir::Butterworth::BandStop< FilterOrder, StateType >, 35](#)
- [lir::Butterworth::BandStopBase, 36](#)
- [lir::Butterworth::HighPass](#)
 - [setup, 51](#)
- [lir::Butterworth::HighPass< FilterOrder, StateType >, 49](#)
- [lir::Butterworth::HighPassBase, 52](#)
- [lir::Butterworth::HighShelf](#)
 - [setup, 57](#)
- [lir::Butterworth::HighShelf< FilterOrder, StateType >, 56](#)
- [lir::Butterworth::HighShelfBase, 57](#)
- [lir::Butterworth::LowPass](#)
 - [setup, 62](#)
- [lir::Butterworth::LowPass< FilterOrder, StateType >, 61](#)
- [lir::Butterworth::LowPassBase, 65](#)
- [lir::Butterworth::LowShelf](#)
 - [setup, 67](#)
- [lir::Butterworth::LowShelf< FilterOrder, StateType >, 66](#)
- [lir::Butterworth::LowShelfBase, 72](#)
- [lir::Cascade, 42](#)
 - [getNumStages, 43](#)
 - [getPoleZeros, 43](#)
 - [operator\[\], 43](#)
 - [response, 43](#)
- [lir::Cascade::Storage, 77](#)
 - [Storage, 77](#)
- [lir::CascadeStages](#)
 - [filter, 44](#)
 - [reset, 44](#)
 - [setup, 44](#)
- [lir::CascadeStages< MaxStages, StateType >, 43](#)
- [lir::ChebyshevI::AnalogLowPass, 17](#)
- [lir::ChebyshevI::AnalogLowShelf, 19](#)
- [lir::ChebyshevI::BandPass](#)
 - [setup, 19](#)
- [lir::ChebyshevI::BandPass< FilterOrder, StateType >, 19](#)
- [lir::ChebyshevI::BandPassBase, 25](#)
- [lir::ChebyshevI::BandShelf](#)
 - [setup, 28](#)

- lir::ChebyshevI::BandShelf< FilterOrder, StateType >, 28
- lir::ChebyshevI::BandShelfBase, 31
- lir::ChebyshevI::BandStop
 - setup, 32
- lir::ChebyshevI::BandStop< FilterOrder, StateType >, 32
- lir::ChebyshevI::BandStopBase, 37
- lir::ChebyshevI::HighPass
 - setup, 47
- lir::ChebyshevI::HighPass< FilterOrder, StateType >, 46
- lir::ChebyshevI::HighPassBase, 51
- lir::ChebyshevI::HighShelf
 - setup, 54
- lir::ChebyshevI::HighShelf< FilterOrder, StateType >, 54
- lir::ChebyshevI::HighShelfBase, 58
- lir::ChebyshevI::LowPass
 - setup, 64
- lir::ChebyshevI::LowPass< FilterOrder, StateType >, 63
- lir::ChebyshevI::LowPassBase, 66
- lir::ChebyshevI::LowShelf
 - setup, 69
- lir::ChebyshevI::LowShelf< FilterOrder, StateType >, 68
- lir::ChebyshevI::LowShelfBase, 71
- lir::ChebyshevII::AnalogLowPass, 18
- lir::ChebyshevII::AnalogLowShelf, 18
- lir::ChebyshevII::BandPass
 - setup, 22
- lir::ChebyshevII::BandPass< FilterOrder, StateType >, 21
- lir::ChebyshevII::BandPassBase, 24
- lir::ChebyshevII::BandShelf
 - setup, 30
- lir::ChebyshevII::BandShelf< FilterOrder, StateType >, 29
- lir::ChebyshevII::BandShelfBase, 31
- lir::ChebyshevII::BandStop
 - setup, 34
- lir::ChebyshevII::BandStop< FilterOrder, StateType >, 34
- lir::ChebyshevII::BandStopBase, 37
- lir::ChebyshevII::HighPass
 - setup, 49
- lir::ChebyshevII::HighPass< FilterOrder, StateType >, 48
- lir::ChebyshevII::HighPassBase, 52
- lir::ChebyshevII::HighShelf
 - setup, 55
- lir::ChebyshevII::HighShelf< FilterOrder, StateType >, 55
- lir::ChebyshevII::HighShelfBase, 58
- lir::ChebyshevII::LowPass
 - setup, 63
- lir::ChebyshevII::LowPass< FilterOrder, StateType >, 62
- lir::ChebyshevII::LowPassBase, 65
- lir::ChebyshevII::LowShelf
 - setup, 70
- lir::ChebyshevII::LowShelf< FilterOrder, StateType >, 70
- lir::ChebyshevII::LowShelfBase, 71
- lir::ChebyshevII, 16
- lir::ChebyshevI, 15
- lir::ComplexPair, 45
- lir::Custom, 16
- lir::Custom::OnePole, 72
- lir::Custom::SOSCascade
 - setup, 76
- lir::Custom::SOSCascade< NSOS, StateType >, 76
- lir::Custom::TwoPole, 78
- lir::DirectFormII, 46
- lir::DirectFormI, 45
- lir::EnvelopeFollower< Channels, Value >, 46
- lir::HighPassTransform, 52
- lir::Layout< MaxPoles >, 60
- lir::LayoutBase, 60
- lir::LowPassTransform, 66
- lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >, 73
- lir::PoleFilterBase< AnalogPrototype >, 73
- lir::PoleFilterBase2, 74
- lir::PoleZeroPair, 74
- lir::RBJ::AllPass, 16
- lir::RBJ::BandPass1, 22
 - setup, 23
- lir::RBJ::BandPass2, 23
 - setup, 24
- lir::RBJ::BandShelf, 27
 - setup, 27
- lir::RBJ::BandStop, 33
 - setup, 33
- lir::RBJ::HighPass, 47
 - setup, 48
- lir::RBJ::HighShelf, 53
 - setup, 53
- lir::RBJ::IIRNotch, 59
 - setup, 59
- lir::RBJ::LowPass, 60
 - setup, 61
- lir::RBJ::LowShelf, 67
 - setup, 68
- lir::RBJ::RBJbase, 75
- lir::SlopeDetector< Channels, Value >, 76
- lir::TransposedDirectFormII, 77
- Kind
 - lir, 13
- operator[]
 - lir::Cascade, 43
- reset
 - lir::CascadeStages, 44
- response
 - lir::Biquad, 40

- [lir::Cascade](#), [43](#)
- [setCoefficients](#)
 - [lir::Biquad](#), [40](#)
- [setIdentity](#)
 - [lir::Biquad](#), [41](#)
- [setOnePole](#)
 - [lir::Biquad](#), [41](#)
- [setPoleZeroPair](#)
 - [lir::Biquad](#), [41](#)
- [setTwoPole](#)
 - [lir::Biquad](#), [41](#)
- [setup](#)
 - [lir::Butterworth::BandPass](#), [21](#)
 - [lir::Butterworth::BandShelf](#), [26](#)
 - [lir::Butterworth::BandStop](#), [35](#)
 - [lir::Butterworth::HighPass](#), [51](#)
 - [lir::Butterworth::HighShelf](#), [57](#)
 - [lir::Butterworth::LowPass](#), [62](#)
 - [lir::Butterworth::LowShelf](#), [67](#)
 - [lir::CascadeStages](#), [44](#)
 - [lir::ChebyshevI::BandPass](#), [19](#)
 - [lir::ChebyshevI::BandShelf](#), [28](#)
 - [lir::ChebyshevI::BandStop](#), [32](#)
 - [lir::ChebyshevI::HighPass](#), [47](#)
 - [lir::ChebyshevI::HighShelf](#), [54](#)
 - [lir::ChebyshevI::LowPass](#), [64](#)
 - [lir::ChebyshevI::LowShelf](#), [69](#)
 - [lir::ChebyshevII::BandPass](#), [22](#)
 - [lir::ChebyshevII::BandShelf](#), [30](#)
 - [lir::ChebyshevII::BandStop](#), [34](#)
 - [lir::ChebyshevII::HighPass](#), [49](#)
 - [lir::ChebyshevII::HighShelf](#), [55](#)
 - [lir::ChebyshevII::LowPass](#), [63](#)
 - [lir::ChebyshevII::LowShelf](#), [70](#)
 - [lir::Custom::SOSCascade](#), [76](#)
 - [lir::RBJ::BandPass1](#), [23](#)
 - [lir::RBJ::BandPass2](#), [24](#)
 - [lir::RBJ::BandShelf](#), [27](#)
 - [lir::RBJ::BandStop](#), [33](#)
 - [lir::RBJ::HighPass](#), [48](#)
 - [lir::RBJ::HighShelf](#), [53](#)
 - [lir::RBJ::IIRNotch](#), [59](#)
 - [lir::RBJ::LowPass](#), [61](#)
 - [lir::RBJ::LowShelf](#), [68](#)
- [Storage](#)
 - [lir::Cascade::Storage](#), [77](#)