

iir1

Generated by Doxygen 1.8.17

1 DSP IIR Realtime C++ filter library	1
1.1 C++ code	1
1.1.1 Setting the filter parameters	1
1.1.2 Realtime filtering sample by sample	3
1.1.3 Error handling	3
1.2 Linking	3
1.2.1 CMake setup	3
1.2.2 Generic linker setup	3
1.3 Packages for Ubuntu (xenial / bionic / focal):	3
1.4 Package for MacOS	3
1.5 Compilation from source	4
1.5.1 Linux / Mac	4
1.5.2 Windows	4
1.5.3 Unit tests	4
1.6 Documentation	4
1.6.1 Learn from the demos	4
1.6.2 Detailed documentation	4
1.7 Example filter responses	5
1.8 Credits	5
1.9 Bibliography	5
2 Namespace Index	5
2.1 Namespace List	5
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	9
4.1 Class List	9
5 Namespace Documentation	12
5.1 Iir Namespace Reference	12
5.1.1 Detailed Description	13
5.1.2 Enumeration Type 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	15
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
6.1.1 Detailed Description	16
6.1.2 Member Function Documentation	16
6.2 lir::Butterworth::AnalogLowPass Class Reference	17
6.2.1 Detailed Description	17
6.3 lir::ChebyshevI::AnalogLowPass Class Reference	17
6.3.1 Detailed Description	17
6.4 lir::ChebyshevII::AnalogLowPass Class Reference	18
6.4.1 Detailed Description	18
6.5 lir::ChebyshevII::AnalogLowShelf Class Reference	18
6.5.1 Detailed Description	18
6.6 lir::Butterworth::AnalogLowShelf Class Reference	18
6.6.1 Detailed Description	18
6.7 lir::ChebyshevI::AnalogLowShelf Class Reference	19
6.7.1 Detailed Description	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	21
6.9.1 Detailed Description	21
6.9.2 Member Function Documentation	21
6.10 lir::ChebyshevII::BandPass< FilterOrder, StateType > Struct Template Reference	23
6.10.1 Detailed Description	23
6.10.2 Member Function Documentation	23
6.11 lir::RBJ::BandPass1 Struct Reference	24
6.11.1 Detailed Description	25
6.11.2 Member Function Documentation	25
6.12 lir::RBJ::BandPass2 Struct Reference	25
6.12.1 Detailed Description	26
6.12.2 Member Function Documentation	26
6.13 lir::ChebyshevII::BandPassBase Struct Reference	26
6.14 lir::ChebyshevI::BandPassBase Struct Reference	27
6.15 lir::Butterworth::BandPassBase Struct Reference	27
6.16 lir::BandPassTransform Class Reference	28
6.16.1 Detailed Description	28
6.17 lir::Butterworth::BandShelf< FilterOrder, StateType > Struct Template Reference	28
6.17.1 Detailed Description	28
6.17.2 Member Function Documentation	28
6.18 lir::RBJ::BandShelf Struct Reference	30
6.18.1 Detailed Description	30
6.18.2 Member Function Documentation	30
6.19 lir::ChebyshevI::BandShelf< FilterOrder, StateType > Struct Template Reference	31

6.19.1 Detailed Description	31
6.19.2 Member Function Documentation	31
6.20 <code>lir::ChebyshevII::BandShelf< FilterOrder, StateType > Struct Template Reference</code>	33
6.20.1 Detailed Description	33
6.20.2 Member Function Documentation	34
6.21 <code>lir::Butterworth::BandShelfBase Struct Reference</code>	35
6.22 <code>lir::ChebyshevII::BandShelfBase Struct Reference</code>	35
6.23 <code>lir::ChebyshevI::BandShelfBase Struct Reference</code>	36
6.24 <code>lir::ChebyshevI::BandStop< FilterOrder, StateType > Struct Template Reference</code>	36
6.24.1 Detailed Description	37
6.24.2 Member Function Documentation	37
6.25 <code>lir::RBJ::BandStop Struct Reference</code>	38
6.25.1 Detailed Description	38
6.25.2 Member Function Documentation	38
6.26 <code>lir::ChebyshevII::BandStop< FilterOrder, StateType > Struct Template Reference</code>	39
6.26.1 Detailed Description	39
6.26.2 Member Function Documentation	40
6.27 <code>lir::Butterworth::BandStop< FilterOrder, StateType > Struct Template Reference</code>	41
6.27.1 Detailed Description	41
6.27.2 Member Function Documentation	42
6.28 <code>lir::Butterworth::BandStopBase Struct Reference</code>	43
6.29 <code>lir::ChebyshevII::BandStopBase Struct Reference</code>	43
6.30 <code>lir::ChebyshevI::BandStopBase Struct Reference</code>	44
6.31 <code>lir::BandStopTransform Class Reference</code>	44
6.31.1 Detailed Description	44
6.32 <code>lir::Biquad Class Reference</code>	44
6.32.1 Member Function Documentation	45
6.33 <code>lir::BiquadPoleState Struct Reference</code>	47
6.33.1 Detailed Description	48
6.34 <code>lir::Cascade Class Reference</code>	48
6.34.1 Detailed Description	48
6.34.2 Member Function Documentation	48
6.35 <code>lir::CascadeStages< MaxStages, StateType > Class Template Reference</code>	49
6.35.1 Detailed Description	49
6.35.2 Member Function Documentation	49
6.36 <code>lir::ComplexPair Struct Reference</code>	50
6.36.1 Detailed Description	50
6.36.2 Member Function Documentation	50
6.37 <code>lir::DirectFormI Class Reference</code>	50
6.37.1 Detailed Description	51
6.38 <code>lir::DirectFormII Class Reference</code>	51
6.38.1 Detailed Description	51

6.39	lir::ChebyshevI::HighPass< FilterOrder, StateType > Struct Template Reference	51
6.39.1	Detailed Description	51
6.39.2	Member Function Documentation	52
6.40	lir::RBJ::HighPass Struct Reference	53
6.40.1	Detailed Description	53
6.40.2	Member Function Documentation	53
6.41	lir::ChebyshevII::HighPass< FilterOrder, StateType > Struct Template Reference	54
6.41.1	Detailed Description	54
6.41.2	Member Function Documentation	54
6.42	lir::Butterworth::HighPass< FilterOrder, StateType > Struct Template Reference	55
6.42.1	Detailed Description	56
6.42.2	Member Function Documentation	56
6.43	lir::ChebyshevII::HighPassBase Struct Reference	57
6.44	lir::ChebyshevI::HighPassBase Struct Reference	57
6.45	lir::Butterworth::HighPassBase Struct Reference	58
6.46	lir::HighPassTransform Class Reference	58
6.46.1	Detailed Description	58
6.47	lir::Butterworth::HighShelf< FilterOrder, StateType > Struct Template Reference	58
6.47.1	Detailed Description	59
6.47.2	Member Function Documentation	59
6.48	lir::RBJ::HighShelf Struct Reference	60
6.48.1	Detailed Description	61
6.48.2	Member Function Documentation	61
6.49	lir::ChebyshevI::HighShelf< FilterOrder, StateType > Struct Template Reference	61
6.49.1	Detailed Description	62
6.49.2	Member Function Documentation	62
6.50	lir::ChebyshevII::HighShelf< FilterOrder, StateType > Struct Template Reference	63
6.50.1	Detailed Description	63
6.50.2	Member Function Documentation	64
6.51	lir::ChebyshevII::HighShelfBase Struct Reference	65
6.52	lir::Butterworth::HighShelfBase Struct Reference	65
6.53	lir::ChebyshevI::HighShelfBase Struct Reference	66
6.54	lir::RBJ::IIRNotch Struct Reference	66
6.54.1	Detailed Description	67
6.54.2	Member Function Documentation	67
6.55	lir::Layout< MaxPoles > Class Template Reference	67
6.55.1	Detailed Description	67
6.56	lir::LayoutBase Class Reference	67
6.56.1	Detailed Description	68
6.57	lir::RBJ::LowPass Struct Reference	68
6.57.1	Detailed Description	68
6.57.2	Member Function Documentation	68

6.58 lir::ChebyshevII::LowPass< FilterOrder, StateType > Struct Template Reference	69
6.58.1 Detailed Description	69
6.58.2 Member Function Documentation	69
6.59 lir::Butterworth::LowPass< FilterOrder, StateType > Struct Template Reference	70
6.59.1 Detailed Description	71
6.59.2 Member Function Documentation	71
6.60 lir::ChebyshevI::LowPass< FilterOrder, StateType > Struct Template Reference	72
6.60.1 Detailed Description	72
6.60.2 Member Function Documentation	73
6.61 lir::ChebyshevI::LowPassBase Struct Reference	74
6.62 lir::Butterworth::LowPassBase Struct Reference	74
6.63 lir::ChebyshevII::LowPassBase Struct Reference	75
6.64 lir::LowPassTransform Class Reference	75
6.64.1 Detailed Description	75
6.65 lir::ChebyshevI::LowShelf< FilterOrder, StateType > Struct Template Reference	75
6.65.1 Detailed Description	76
6.65.2 Member Function Documentation	76
6.66 lir::Butterworth::LowShelf< FilterOrder, StateType > Struct Template Reference	77
6.66.1 Detailed Description	77
6.66.2 Member Function Documentation	78
6.67 lir::RBJ::LowShelf Struct Reference	79
6.67.1 Detailed Description	79
6.67.2 Member Function Documentation	79
6.68 lir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference	80
6.68.1 Detailed Description	80
6.68.2 Member Function Documentation	80
6.69 lir::Butterworth::LowShelfBase Struct Reference	82
6.70 lir::ChebyshevI::LowShelfBase Struct Reference	82
6.71 lir::ChebyshevII::LowShelfBase Struct Reference	83
6.72 lir::Custom::OnePole Struct Reference	83
6.72.1 Detailed Description	83
6.73 lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > Struct Template Reference	83
6.73.1 Detailed Description	84
6.74 lir::PoleFilterBase< AnalogPrototype > Class Template Reference	84
6.74.1 Detailed Description	84
6.75 lir::PoleFilterBase2 Class Reference	84
6.75.1 Detailed Description	85
6.76 lir::PoleZeroPair Struct Reference	85
6.76.1 Detailed Description	85
6.77 lir::RBJ::RBJbase Struct Reference	85
6.77.1 Detailed Description	86
6.78 lir::Custom::SOSCascade< NSOS, StateType > Struct Template Reference	86

6.78.1 Detailed Description	87
6.78.2 Constructor & Destructor Documentation	87
6.78.3 Member Function Documentation	87
6.79 <code>lir::Cascade::Storage</code> Struct Reference	88
6.79.1 Detailed Description	88
6.79.2 Constructor & Destructor Documentation	88
6.80 <code>lir::TransposedDirectFormII</code> Class Reference	88
6.81 <code>lir::Custom::TwoPole</code> Struct Reference	88
6.81.1 Detailed Description	88
Index	91

1 DSP IIR 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. Memory is allocated at compile time so that there is never the risk of memory leaks.

1.1 C++ code

Add the following include statement to your code:

```
#include "Iir.h"
```

The general coding approach is that first the filter is instantiated specifying its order, then the parameters are set with the function `setup` and then it's ready to be used for sample by sample realtime filtering.

1.1.1 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.

The frequencies can either be analogue ones against the sampling rate or normalised ones between 0..1/2 where 1/2 is the Nyquist frequency. Note that normalised frequencies are simply $f = F/F_s$ and are in units of 1/samples. Internally the library uses normalised frequencies and the setup commands simply divide by the sampling rate if given. Choose between:

1. `setup`: sampling rate and the analogue cutoff frequencies
2. `setupN`: normalised frequencies in 1/samples between $f = 0..1/2$ where $1/2 = \text{Nyquist}$.

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);
```

or specify a normalised frequency between 0..1/2:

```
f.setupN(norm_cutoff_frequency);
```

1. 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);
```

or specify a normalised frequency between 0..1/2:

```
f.setupN(norm_cutoff_frequency, passband_ripple_in_db);
```

1. 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);
```

or specify a normalised frequency between 0..1/2:

```
f.setupN(norm_cutoff_frequency, stopband_ripple_in_db);
```

1. 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);
```

or specify a normalised frequency between 0..1/2:

```
f.setupN(norm_cutoff_frequency, Q_factor);
```

1. 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"
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}
};
const int nSOS = sizeof(coeff) / sizeof(coeff[0]); // here: nSOS = 2 = order / 2
Iir::Custom::SOSCascade<nSOS> cust(coeff);
```


1.1.2 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 types of `x` and `y` can either be float or double (integer is also allowed but is still processed internally as floating point):

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

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

1.1.3 Error handling

Invalid values provided to `setup()` will throw an exception. Parameters provided to `setup()` which result in coefficients being NAN will also throw an exception.

1.2 Linking

1.2.1 CMake setup

If you use cmake as your build system then just add to your `CMakeLists.txt` the following lines for the dynamic library:

```
find_package(iir)
target_link_libraries(... iir::iir)
```

or for the static one:

```
find_package(iir)
target_link_libraries(... iir::iir_static)
```

1.2.2 Generic linker setup

Link it against the dynamic library (Unix/Mac: `-liir`, Windows: `iir.lib`) or the static library (Unix/Mac: `-libiir_static.a`, Windows: `libiir_static.lib`).

1.3 Packages for Ubuntu (xenial / bionic / focal):

If you have Ubuntu's LTS distros xenial, bionic or focal then install it as a pre-compiled package:

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

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/
```

1.4 Package for MacOS

Make sure you have the homebrew package manager installed: <https://brew.sh/>

Add the homebrew tap:

```
brew tap berndporr/dsp
```

and then install the iir filter package with:

```
brew install iir
```

1.5 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.

1.5.1 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.

1.5.2 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.

1.5.3 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.

1.6 Documentation

1.6.1 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.

1.6.2 Detailed documentation

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

The online documentation is here: <http://berndporr.github.io/iir1>

1.7 Example filter responses

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

1.8 Credits

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

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

While the original library processes audio arrays this library has been adapted to do fast realtime processing sample by sample. The `setup` command won't require the filter order and instead remembers it from the template argument. The class structure has been simplified and all functions documented for doxygen. Instead of having `assert()` statements this library throws exceptions in case a parameter is wrong. Any filter design requiring optimisation (for example Elliptic filters) has been removed and instead a function has been added which can import easily coefficients from scipy.

1.9 Bibliography

"High-Order Digital Parametric Equalizer Design"
Sophocles J. Orfanidis
(Journal of the Audio Engineering Society, vol 53. pp 1026-1046)
"Spectral Transformations for digital filters"
A. G. Constantinides, B.Sc.(Eng.) Ph.D.
(Proceedings of the IEEE, vol. 117, pp. 1585-1590, August 1970)

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	12
lir::Butterworth	14
lir::ChebyshevI	15
lir::ChebyshevII	15
lir::Custom	16

3 Hierarchical Index

3.1 Class Hierarchy

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

BandPassBase

lir::PoleFilter < BandPassBase , DirectFormII , 4 , 4 * 2 >	83
lir::Butterworth::BandPass < FilterOrder , StateType >	21
lir::ChebyshevI::BandPass < FilterOrder , StateType >	19
lir::ChebyshevII::BandPass < FilterOrder , StateType >	23
lir::BandPassTransform	28
BandShelfBase	
lir::PoleFilter < BandShelfBase , DirectFormII , 4 , 4 * 2 >	83
lir::Butterworth::BandShelf < FilterOrder , StateType >	28
lir::ChebyshevI::BandShelf < FilterOrder , StateType >	31
lir::ChebyshevII::BandShelf < FilterOrder , StateType >	33
BandStopBase	
lir::PoleFilter < BandStopBase , DirectFormII , 4 , 4 * 2 >	83
lir::Butterworth::BandStop < FilterOrder , StateType >	41
lir::ChebyshevI::BandStop < FilterOrder , StateType >	36
lir::ChebyshevII::BandStop < FilterOrder , StateType >	39
lir::BandStopTransform	44
BaseClass	
lir::PoleFilter < BaseClass , StateType , MaxAnalogPoles , MaxDigitalPoles >	83
lir::Biquad	44
lir::Custom::OnePole	83
lir::Custom::TwoPole	88
lir::RBJ::RBJbase	85
lir::RBJ::AllPass	16
lir::RBJ::BandPass1	24
lir::RBJ::BandPass2	25
lir::RBJ::BandShelf	30
lir::RBJ::BandStop	38
lir::RBJ::HighPass	53
lir::RBJ::HighShelf	60
lir::RBJ::IIRNotch	66
lir::RBJ::LowPass	68
lir::RBJ::LowShelf	79
lir::Cascade	48

lir::PoleFilterBase2	84
lir::PoleFilterBase< AnalogPrototype >	84
lir::PoleFilterBase< AnalogLowPass >	84
lir::Butterworth::BandPassBase	27
lir::Butterworth::BandStopBase	43
lir::Butterworth::HighPassBase	58
lir::Butterworth::LowPassBase	74
lir::ChebyshevI::BandPassBase	27
lir::ChebyshevI::BandStopBase	44
lir::ChebyshevI::HighPassBase	57
lir::ChebyshevI::LowPassBase	74
lir::ChebyshevII::BandPassBase	26
lir::ChebyshevII::BandStopBase	43
lir::ChebyshevII::HighPassBase	57
lir::ChebyshevII::LowPassBase	75
lir::PoleFilterBase< AnalogLowShelf >	84
lir::Butterworth::BandShelfBase	35
lir::Butterworth::HighShelfBase	65
lir::Butterworth::LowShelfBase	82
lir::ChebyshevI::BandShelfBase	36
lir::ChebyshevI::HighShelfBase	66
lir::ChebyshevI::LowShelfBase	82
lir::ChebyshevII::BandShelfBase	35
lir::ChebyshevII::HighShelfBase	65
lir::ChebyshevII::LowShelfBase	83
lir::CascadeStages< MaxStages, StateType >	49
lir::CascadeStages< NSOS, DirectFormII >	49
lir::Custom::SOSCascade< NSOS, StateType >	86
lir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >	49
lir::PoleFilter< HighPassBase, DirectFormII, 4 >	83
lir::Butterworth::HighPass< FilterOrder, StateType >	55
lir::ChebyshevI::HighPass< FilterOrder, StateType >	51

lir::ChebyshevII::HighPass< FilterOrder, StateType >	54
lir::PoleFilter< HighShelfBase, DirectFormII, 4 >	83
lir::Butterworth::HighShelf< FilterOrder, StateType >	58
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	61
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	63
lir::PoleFilter< LowPassBase, DirectFormII, 4 >	83
lir::Butterworth::LowPass< FilterOrder, StateType >	70
lir::ChebyshevI::LowPass< FilterOrder, StateType >	72
lir::ChebyshevII::LowPass< FilterOrder, StateType >	69
lir::PoleFilter< LowShelfBase, DirectFormII, 4 >	83
lir::Butterworth::LowShelf< FilterOrder, StateType >	77
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	75
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	80
lir::CascadeStages<(MaxAnalogPoles+1)/2, StateType >	49
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	83
lir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >	49
lir::PoleFilter< BandPassBase, DirectFormII, 4, 4 *2 >	83
lir::PoleFilter< BandShelfBase, DirectFormII, 4, 4 *2 >	83
lir::PoleFilter< BandStopBase, DirectFormII, 4, 4 *2 >	83
complex_pair_t	
lir::ComplexPair	50
lir::DirectFormI	50
lir::DirectFormII	51
HighPassBase	
lir::PoleFilter< HighPassBase, DirectFormII, 4 >	83
lir::HighPassTransform	58
HighShelfBase	
lir::PoleFilter< HighShelfBase, DirectFormII, 4 >	83
lir::Layout< MaxPoles >	67
lir::Layout< MaxAnalogPoles >	67
lir::Layout< MaxDigitalPoles >	67
lir::LayoutBase	67
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, DirectFormII, 4 >	83
lir::LowPassTransform	75
LowShelfBase	
lir::PoleFilter< LowShelfBase, DirectFormII, 4 >	83
lir::PoleZeroPair	85
lir::BiquadPoleState	47
lir::Cascade::Storage	88
lir::TransposedDirectFormII	88

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 >	21
lir::ChebyshevII::BandPass< FilterOrder, StateType >	23
lir::RBJ::BandPass1	24
lir::RBJ::BandPass2	25
lir::ChebyshevII::BandPassBase	26
lir::ChebyshevI::BandPassBase	27

lir::Butterworth::BandPassBase	27
lir::BandPassTransform	28
lir::Butterworth::BandShelf< FilterOrder, StateType >	28
lir::RBJ::BandShelf	30
lir::ChebyshevI::BandShelf< FilterOrder, StateType >	31
lir::ChebyshevII::BandShelf< FilterOrder, StateType >	33
lir::Butterworth::BandShelfBase	35
lir::ChebyshevII::BandShelfBase	35
lir::ChebyshevI::BandShelfBase	36
lir::ChebyshevI::BandStop< FilterOrder, StateType >	36
lir::RBJ::BandStop	38
lir::ChebyshevII::BandStop< FilterOrder, StateType >	39
lir::Butterworth::BandStop< FilterOrder, StateType >	41
lir::Butterworth::BandStopBase	43
lir::ChebyshevII::BandStopBase	43
lir::ChebyshevI::BandStopBase	44
lir::BandStopTransform	44
lir::Biquad	44
lir::BiquadPoleState	47
lir::Cascade	48
lir::CascadeStages< MaxStages, StateType >	49
lir::ComplexPair	50
lir::DirectFormI	50
lir::DirectFormII	51
lir::ChebyshevI::HighPass< FilterOrder, StateType >	51
lir::RBJ::HighPass	53
lir::ChebyshevII::HighPass< FilterOrder, StateType >	54
lir::Butterworth::HighPass< FilterOrder, StateType >	55
lir::ChebyshevII::HighPassBase	57
lir::ChebyshevI::HighPassBase	57
lir::Butterworth::HighPassBase	58
lir::HighPassTransform	58

lir::Butterworth::HighShelf< FilterOrder, StateType >	58
lir::RBJ::HighShelf	60
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	61
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	63
lir::ChebyshevII::HighShelfBase	65
lir::Butterworth::HighShelfBase	65
lir::ChebyshevI::HighShelfBase	66
lir::RBJ::IIRNotch	66
lir::Layout< MaxPoles >	67
lir::LayoutBase	67
lir::RBJ::LowPass	68
lir::ChebyshevII::LowPass< FilterOrder, StateType >	69
lir::Butterworth::LowPass< FilterOrder, StateType >	70
lir::ChebyshevI::LowPass< FilterOrder, StateType >	72
lir::ChebyshevI::LowPassBase	74
lir::Butterworth::LowPassBase	74
lir::ChebyshevII::LowPassBase	75
lir::LowPassTransform	75
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	75
lir::Butterworth::LowShelf< FilterOrder, StateType >	77
lir::RBJ::LowShelf	79
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	80
lir::Butterworth::LowShelfBase	82
lir::ChebyshevI::LowShelfBase	82
lir::ChebyshevII::LowShelfBase	83
lir::Custom::OnePole	83
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	83
lir::PoleFilterBase< AnalogPrototype >	84
lir::PoleFilterBase2	84
lir::PoleZeroPair	85
lir::RBJ::RBJbase	85
lir::Custom::SOSCascade< NSOS, StateType >	86

lir::Cascade::Storage	88
lir::TransposedDirectFormII	88
lir::Custom::TwoPole	88

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

Enumerations

- enum [Kind](#)

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) 2011-2021 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) 2011-2021 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

5.2 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. The filter order is usually set via the template parameter which reserves the correct space and is then automatically passed to the setup function. Optionally one can also provide the filter order at setup time to force a lower order than the default one.

5.3 `lir::ChebyshevI` 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 `lir::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

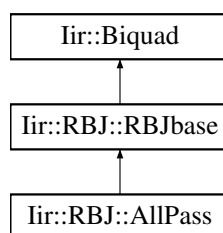
Single pole, [Biquad](#) and cascade of Biquads with parameters allowing for directly setting the parameters.

6 Class Documentation

6.1 Iir::RBJ::AllPass Struct Reference

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

Inheritance diagram for Iir::RBJ::AllPass:



Public Member Functions

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

6.1.1 Detailed Description

Allpass filter

6.1.2 Member Function Documentation

6.1.2.1 setup() void Iir::RBJ::AllPass::setup (
double *sampleRate*,
double *phaseFrequency*,
double *q* = (1/sqrt(2))) [inline]

Calculates the coefficients

Parameters

<i>sampleRate</i>	Sampling rate
<i>phaseFrequency</i>	Frequency where the phase flips
<i>q</i>	Q-factor

6.1.2.2 setupN() `void Iir::RBJ::AllPass::setupN (`
`double phaseFrequency,`
`double q = (1/sqrt(2)))`

Calculates the coefficients

Parameters

<i>phaseFrequency</i>	Normalised frequency where the phase flips
<i>q</i>	Q-factor

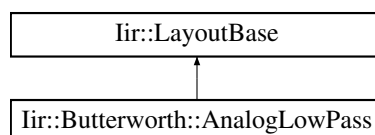
The documentation for this struct was generated from the following files:

- 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

Analogue lowpass prototypes (s-plane)

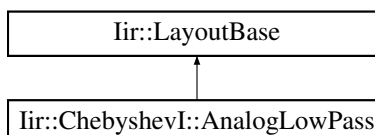
The documentation for this class was generated from the following files:

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

6.3 Iir::ChebyshevI::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::ChebyshevI::AnalogLowPass:



6.3.1 Detailed Description

Analog lowpass prototypes (s-plane)

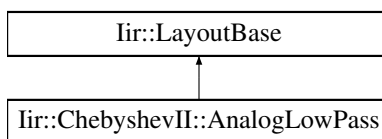
The documentation for this class was generated from the following files:

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

6.4 Iir::ChebyshevII::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::ChebyshevII::AnalogLowPass:



6.4.1 Detailed Description

Analogue lowpass prototype (s-plane)

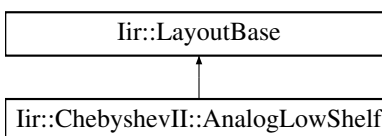
The documentation for this class was generated from the following files:

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

6.5 Iir::ChebyshevII::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::ChebyshevII::AnalogLowShelf:



6.5.1 Detailed Description

Analogue shelf lowpass prototype (s-plane)

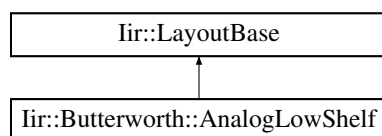
The documentation for this class was generated from the following files:

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

6.6 Iir::Butterworth::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::Butterworth::AnalogLowShelf:



6.6.1 Detailed Description

Analogue low shelf prototypes (s-plane)

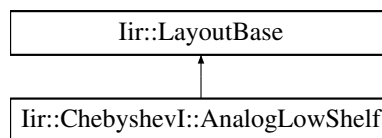
The documentation for this class was generated from the following files:

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

6.7 Iir::ChebyshevI::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::ChebyshevI::AnalogLowShelf:



6.7.1 Detailed Description

Analog lowpass shelf prototype (s-plane)

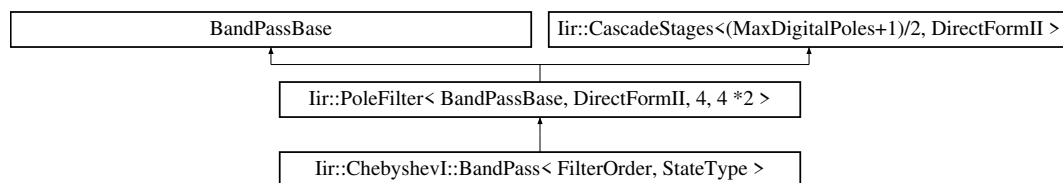
The documentation for this class was generated from the following files:

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

6.8.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
```

```
struct Iir::ChebyshevI::BandPass< FilterOrder, StateType >
```

[ChebyshevI](#) bandpass filter

Parameters

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

6.8.2 Member Function Documentation

6.8.2.1 setup() [1/2] `template<int FilterOrder = 4, 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 at the order FilterOrder

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

```
6.8.2.2 setup() [2/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setup (
    int reqOrder,
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<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

```
6.8.2.3 setupN() [1/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setupN (
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>centerFrequency</i>	Normalised center frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Frequency with of the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

```
6.8.2.4 setupN() [2/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setupN (
    int reqOrder,
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>centerFrequency</i>	Normalised center frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Frequency width of the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

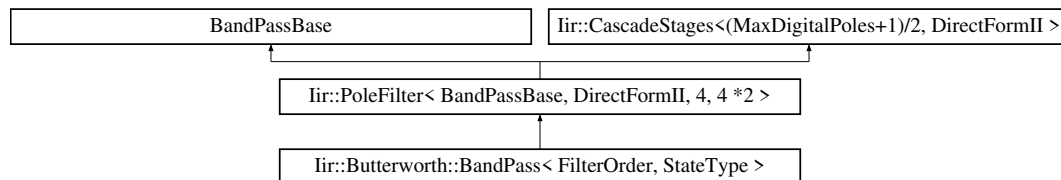
The documentation for this struct was generated from the following file:

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

6.9.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::Butterworth::BandPass< FilterOrder, StateType >
```

[Butterworth](#) Bandpass filter.

Parameters

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

6.9.2 Member Function Documentation

6.9.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandPass< FilterOrder, StateType >::setup (`
 double *sampleRate*,
 double *centerFrequency*,
 double *widthFrequency*) `[inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

6.9.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandPass< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double centerFrequency,`
`double widthFrequency) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass

6.9.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandPass< FilterOrder, StateType >::setupN (`
`double centerFrequency,`
`double widthFrequency) [inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass in normalised freq

6.9.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandPass< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double centerFrequency,`
`double widthFrequency) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass in normalised freq

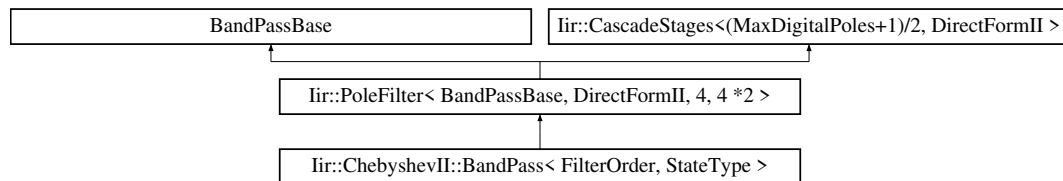
The documentation for this struct was generated from the following file:

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

6.10.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
```

```
struct Iir::ChebyshevII::BandPass< FilterOrder, StateType >
```

[ChebyshevII](#) bandpass filter

Parameters

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

6.10.2 Member Function Documentation

6.10.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, 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

6.10.2.2 [setup\(\)](#) [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setup (
```

```
    int reqOrder,
```

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

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<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

6.10.2.3 **setupN()** [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setupN (
    double centerFrequency,
    double widthFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.10.2.4 **setupN()** [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setupN (
    int reqOrder,
    double centerFrequency,
    double widthFrequency,
    double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

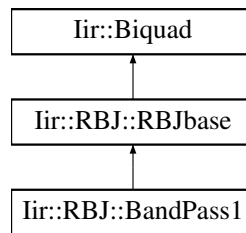
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.11 **Iir::RBJ::BandPass1** Struct Reference

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

Inheritance diagram for Iir::RBJ::BandPass1:



Public Member Functions

- void [setupN](#) (double centerFrequency, double bandWidth)
- 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*) [inline]

Calculates the coefficients

Parameters

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

6.11.2.2 [setupN\(\)](#) void Iir::RBJ::BandPass1::setupN (
 double *centerFrequency*,
 double *bandWidth*)

Calculates the coefficients

Parameters

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

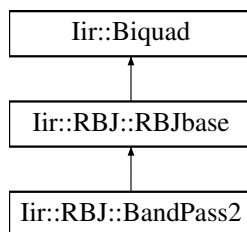
The documentation for this struct was generated from the following files:

- 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 [setupN](#) (double centerFrequency, double bandWidth)
- 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*) [inline]

Calculates the coefficients

Parameters

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

6.12.2.2 [setupN\(\)](#) void Iir::RBJ::BandPass2::setupN (
 double *centerFrequency*,
 double *bandWidth*)

Calculates the coefficients

Parameters

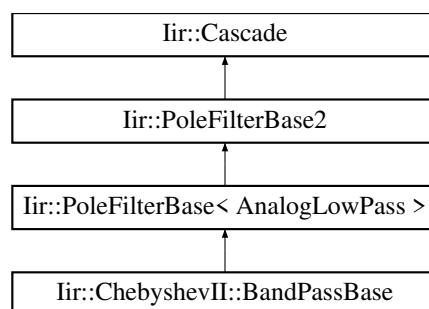
<i>centerFrequency</i>	Normalised centre frequency of the bandpass
<i>bandWidth</i>	Bandwidth in octaves

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:



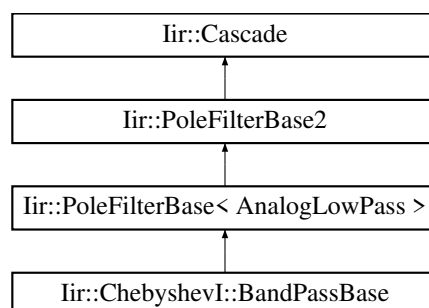
Additional Inherited Members

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:



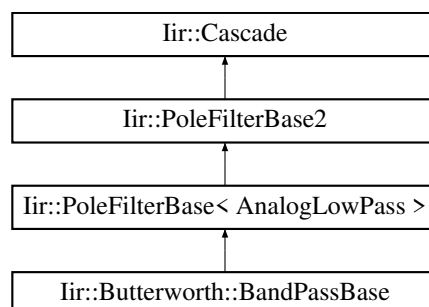
Additional Inherited Members

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

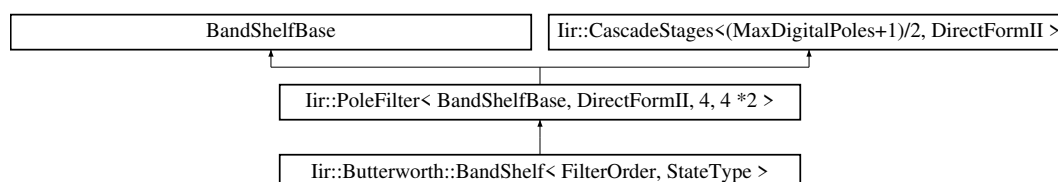
The documentation for this class was generated from the following files:

- 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)
- void [setup](#) (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency, double gain↔ Db)
- void [setupN](#) (double centerFrequency, double widthFrequency, double gainDb)
- void [setupN](#) (int reqOrder, double centerFrequency, double widthFrequency, double gainDb)

6.17.1 Detailed Description

```
template<int FilterOrder = 4, 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>	Reserves memory for a filter of the order FilterOrder
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.17.2 Member Function Documentation

6.17.2.1 setup() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setup (
```

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

Calculates the coefficients with the filter order provided by the instantiation

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

6.17.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setup (`

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

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<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

6.17.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setupN (`

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the passband
<i>widthFrequency</i>	Width of the passband
<i>gainDb</i>	The gain in the passband

6.17.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setupN (`

```
int reqOrder,
double centerFrequency,
double widthFrequency,
double gainDb ) [inline]
```

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the passband
<i>widthFrequency</i>	Width of the passband
<i>gainDb</i>	The gain in the passband

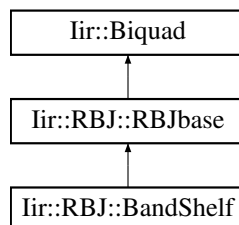
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.18 Iir::RBJ::BandShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::BandShelf:



Public Member Functions

- void [setupN](#) (double centerFrequency, double gainDb, double bandWidth)
- 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*) [inline]

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

6.18.2.2 setupN() void Iir::RBJ::BandShelf::setupN (

```
double centerFrequency,
double gainDb,
double bandWidth )
```

Calculates the coefficients

Parameters

<i>centerFrequency</i>	Normalised centre frequency
<i>gainDb</i>	Gain in the passband
<i>bandWidth</i>	Bandwidth in octaves

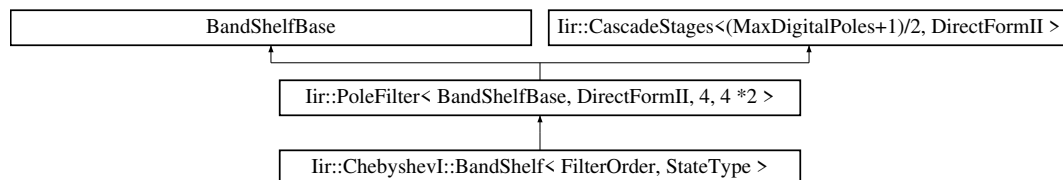
The documentation for this struct was generated from the following files:

- 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)
- void [setup](#) (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency, double gain↔ Db, double rippleDb)
- void [setupN](#) (double centerFrequency, double widthFrequency, double gainDb, double rippleDb)
- void [setupN](#) (int reqOrder, double centerFrequency, double widthFrequency, double gainDb, double rippleDb)

6.19.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::ChebyshevI::BandShelf< FilterOrder, StateType >
```

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

Parameters

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

6.19.2 Member Function Documentation

6.19.2.1 setup() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::BandShelf< FilterOrder, StateType >::setup (`
 double sampleRate,

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

Calculates the coefficients of the filter at the order FilterOrder

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.

6.19.2.2 **setup()** [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::BandShelf< FilterOrder, StateType >::setup (
    int reqOrder,
    double sampleRate,
    double centerFrequency,
    double widthFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<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.

6.19.2.3 **setupN()** [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::BandShelf< FilterOrder, StateType >::setupN (
    double centerFrequency,
    double widthFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) 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.

```

6.19.2.4 setupN() [2/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevI::BandShelf< FilterOrder, StateType >::setupN (
    int reqOrder,
    double centerFrequency,
    double widthFrequency,
    double gainDb,
    double rippleDb ) [inline]

```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) 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.

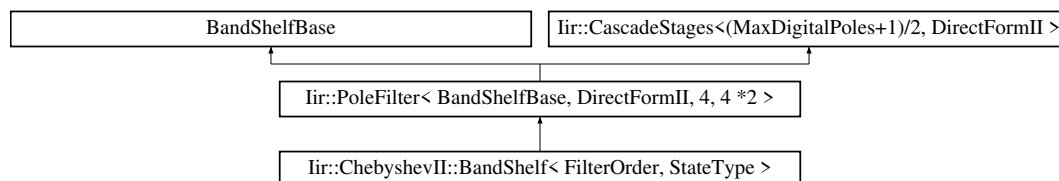
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

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

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



Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void [setup](#) (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency, double gain↔ Db, double stopBandDb)
- void [setupN](#) (double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void [setupN](#) (int reqOrder, double centerFrequency, double widthFrequency, double gainDb, double stop↔ BandDb)

6.20.1 Detailed Description

```

template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::ChebyshevII::BandShelf< FilterOrder, StateType >

```

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

Parameters

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

6.20.2 Member Function Documentation

6.20.2.1 setup() [1/2] `template<int FilterOrder = 4, 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

6.20.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandShelf< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double centerFrequency,`
`double widthFrequency,`
`double gainDb,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<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

6.20.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandShelf< FilterOrder, StateType >::setupN (`
`double centerFrequency,`
`double widthFrequency,`
`double gainDb,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass

Parameters

<i>gainDb</i>	Gain in the passband. The stopband has always 0dB.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.20.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandShelf< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double centerFrequency,`
`double widthFrequency,`
`double gainDb,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

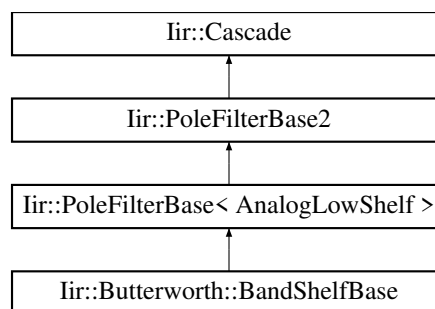
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>gainDb</i>	Gain in the passband. The stopband has always 0dB.
<i>stopBandDb</i>	Permitted ripples in dB 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:



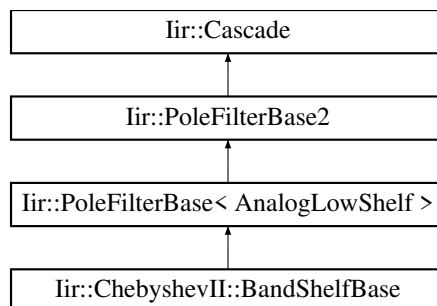
Additional Inherited Members

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:



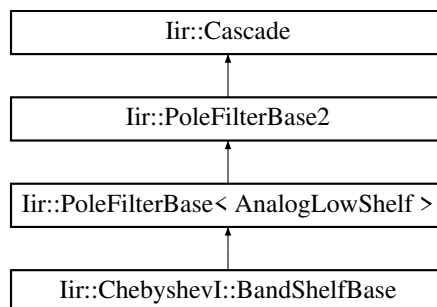
Additional Inherited Members

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

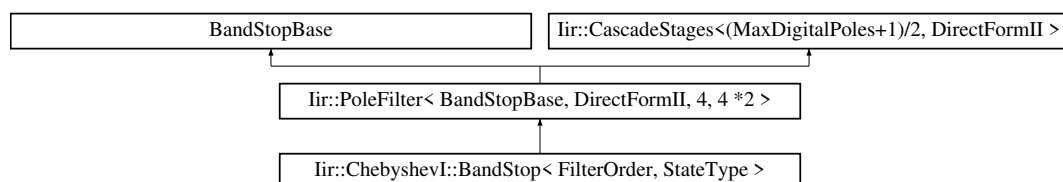
The documentation for this struct was generated from the following files:

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

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

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

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



Public Member Functions

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

6.24.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::ChebyshevI::BandStop< FilterOrder, StateType >
```

[ChebyshevI](#) bandstop filter

Parameters

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

6.24.2 Member Function Documentation

6.24.2.1 setup() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setup (`
 `double sampleRate,`
 `double centerFrequency,`
 `double widthFrequency,`
 `double rippleDb) [inline]`

Calculates the coefficients of the filter at the order *FilterOrder*

Parameters

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

6.24.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setup (`
 `int reqOrder,`
 `double sampleRate,`
 `double centerFrequency,`
 `double widthFrequency,`
 `double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the notch
<i>widthFrequency</i>	Frequency with of the notch
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.24.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setupN (`
 `double centerFrequency,`

```
double widthFrequency,
double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the notch
<i>widthFrequency</i>	Frequency width of the notch
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.24.2.4 **setupN()** [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setupN (
    int reqOrder,
    double centerFrequency,
    double widthFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the notch
<i>widthFrequency</i>	Frequency width of the notch
<i>rippleDb</i>	Permitted ripples in dB in the passband

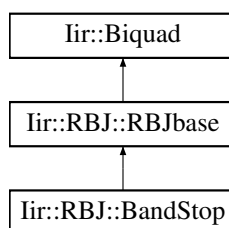
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.25 Iir::RBJ::BandStop Struct Reference

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

Inheritance diagram for Iir::RBJ::BandStop:



Public Member Functions

- void [setupN](#) (double centerFrequency, double bandWidth)
- 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*) [inline]

Calculates the coefficients

Parameters

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

6.25.2.2 setupN() void Iir::RBJ::BandStop::setupN (
double *centerFrequency*,
double *bandWidth*)

Calculates the coefficients

Parameters

<i>centerFrequency</i>	Normalised Centre frequency of the bandstop
<i>bandWidth</i>	Bandwidth in octaves

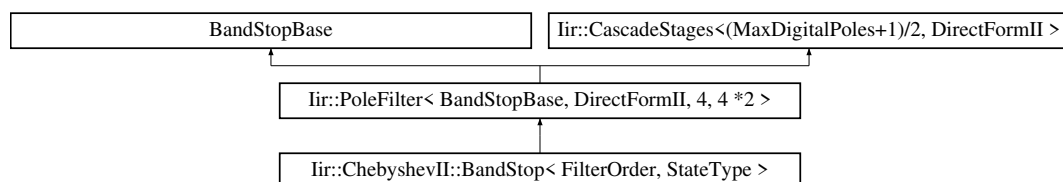
The documentation for this struct was generated from the following files:

- 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*)
- void [setup](#) (int *reqOrder*, double *sampleRate*, double *centerFrequency*, double *widthFrequency*, double *stopBandDb*)
- void [setupN](#) (double *centerFrequency*, double *widthFrequency*, double *stopBandDb*)
- void [setupN](#) (int *reqOrder*, double *centerFrequency*, double *widthFrequency*, double *stopBandDb*)

6.26.1 Detailed Description

template<int FilterOrder = 4, class StateType = DirectFormII>

struct Iir::ChebyshevII::BandStop< FilterOrder, StateType >

[ChebyshevII](#) bandstop filter.

Parameters

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

6.26.2 Member Function Documentation

6.26.2.1 `setup()` [1/2] `template<int FilterOrder = 4, 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

6.26.2.2 `setup()` [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandStop< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double centerFrequency,`
`double widthFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<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

6.26.2.3 `setupN()` [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandStop< FilterOrder, StateType >::setupN (`
`double centerFrequency,`
`double widthFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandstop
------------------------	--

Parameters

<i>widthFrequency</i>	Width of the bandstop
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.26.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::BandStop< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double centerFrequency,`
`double widthFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandstop
<i>widthFrequency</i>	Width of the bandstop
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

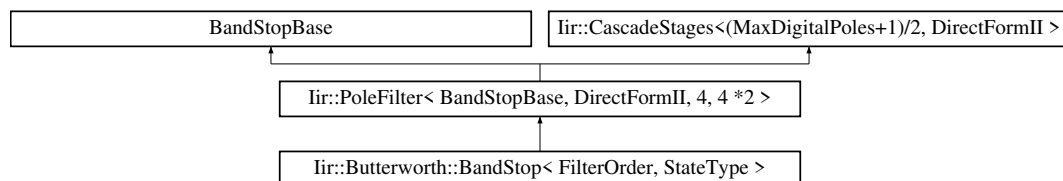
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

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

`#include <Butterworth.h>`

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

**Public Member Functions**

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

6.27.1 Detailed Description

`template<int FilterOrder = 4, class StateType = DirectFormII>`
`struct Iir::Butterworth::BandStop< FilterOrder, StateType >`

[Butterworth](#) Bandstop filter.

Parameters

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

6.27.2 Member Function Documentation

6.27.2.1 setup() `template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::BandStop< FilterOrder, StateType >::setup (`
 double *sampleRate*,
 double *centerFrequency*,
 double *widthFrequency*) `[inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

6.27.2.2 setupN() `[1/3] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::BandStop< FilterOrder, StateType >::setupN (`
 double *centerFrequency*,
 double *widthFrequency*) `[inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandstop
<i>widthFrequency</i>	Normalised width of the bandstop

6.27.2.3 setupN() `[2/3] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::BandStop< FilterOrder, StateType >::setupN (`
 int *reqOrder*,
 double *centerFrequency*,
 double *widthFrequency*) `[inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandstop
<i>widthFrequency</i>	Normalised width of the bandstop

6.27.2.4 setupN() `[3/3] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::BandStop< FilterOrder, StateType >::setupN (`
 int *reqOrder*,
 double *sampleRate*,
 double *centerFrequency*,
 double *widthFrequency*) `[inline]`

Calculates the coefficients

Parameters

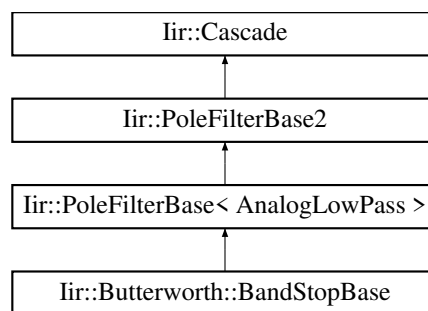
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop

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:



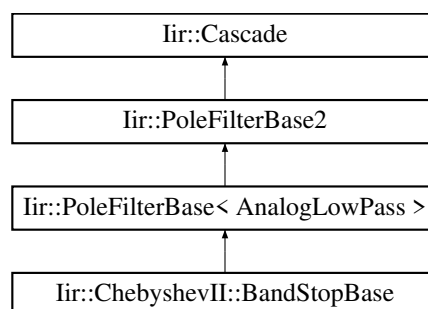
Additional Inherited Members

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:



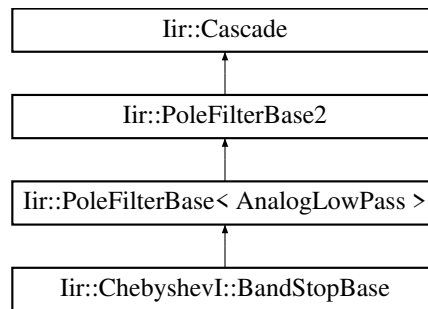
Additional Inherited Members

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

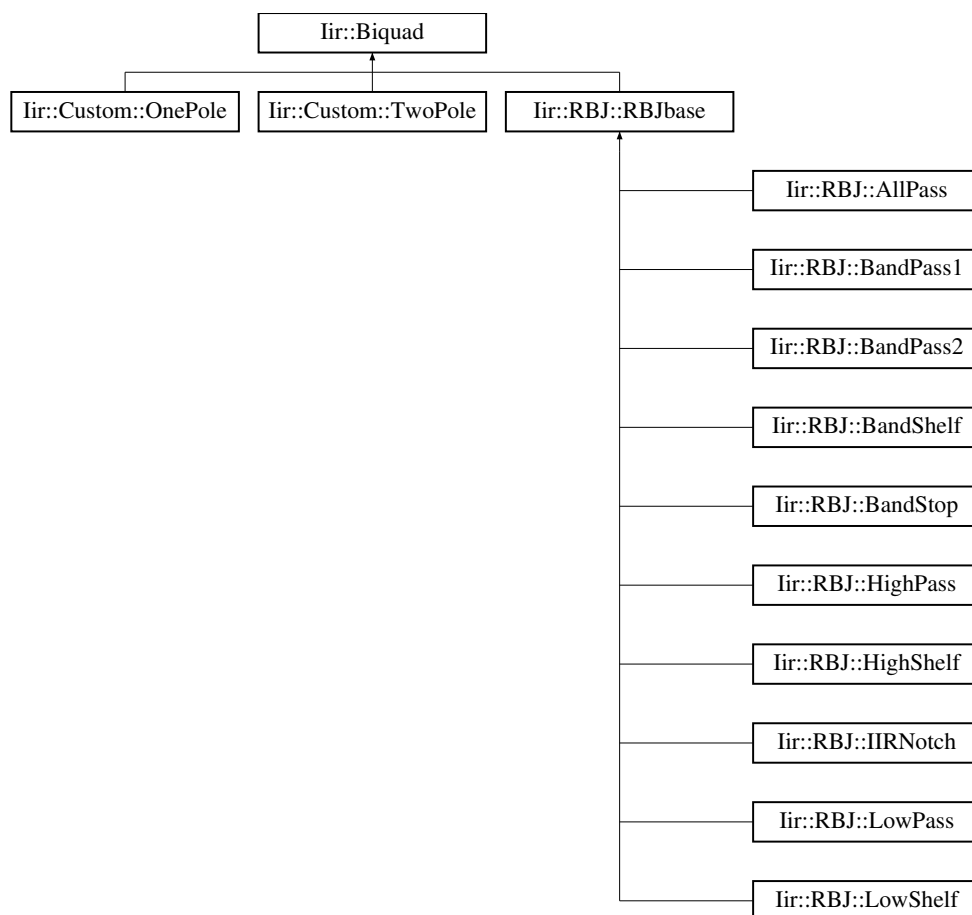
low pass to band stop transform

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 > double filter (double 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 >`

```
double Iir::Biquad::filter (
    double 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)

Returns

The filtered sample.

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 and return the complex response.

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/zeros as a pair. Needs to be complex conjugate.

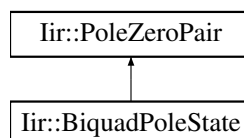
The documentation for this class was generated from the following files:

- 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

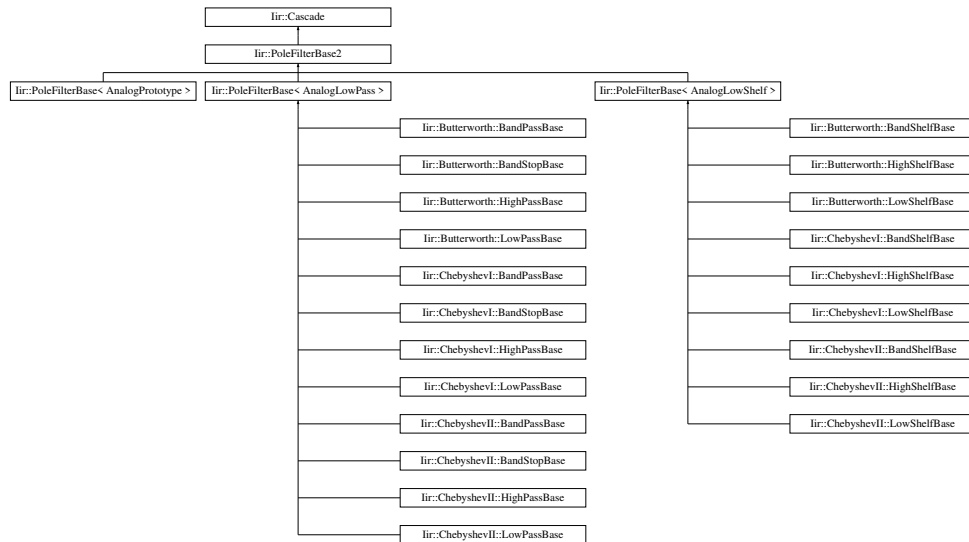
Expresses a biquad as a pair of pole/zeros, with gain values so that the coefficients can be reconstructed precisely. The documentation for this struct was generated from the following files:

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

6.34 Iir::Cascade Class Reference

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

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 Detailed Description

Holds coefficients for a cascade of second order sections.

6.34.2 Member Function Documentation

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

Returns the number of Biquads kept here

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

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

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

Returns a reference to a biquad

6.34.2.4 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)
- const [Cascade::Storage](#) [getCascadeStorage](#) ()
- const StateType(& [getCascadeState](#) ()) [MaxStages]

6.35.1 Detailed Description

```
template<int MaxStages, class StateType>
class Iir::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 Iir::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
-----------	-----------------------

Returns

filtered sample

6.35.2.2 getCascadeState() `template<int MaxStages, class StateType > const StateType(& Iir::CascadeStages< MaxStages, StateType >::getCascadeState ()) [MaxStages] [inline]`

Returns the current state of the entire [Biquad](#) chain

6.35.2.3 getCascadeStorage() `template<int MaxStages, class StateType >`
`const Cascade::Storage Iir::CascadeStages< MaxStages, StateType >::getCascadeStorage ()`
`[inline]`

Returns the coefficients of the entire Biquad chain

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

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

6.35.2.5 setup() `template<int MaxStages, class StateType >`
`void Iir::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.
------------------------	--

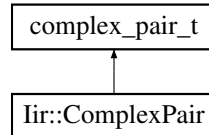
The documentation for this class was generated from the following file:

- iir/Cascade.h

6.36 Iir::ComplexPair Struct Reference

`#include <Types.h>`

Inheritance diagram for Iir::ComplexPair:



Public Member Functions

- bool `isMatchedPair () const`

6.36.1 Detailed Description

A conjugate or real pair

6.36.2 Member Function Documentation

6.36.2.1 isMatchedPair() `bool Iir::ComplexPair::isMatchedPair () const [inline]`

Returns true if this is either a conjugate pair, or a pair of reals where neither is zero.

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:

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

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

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:

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

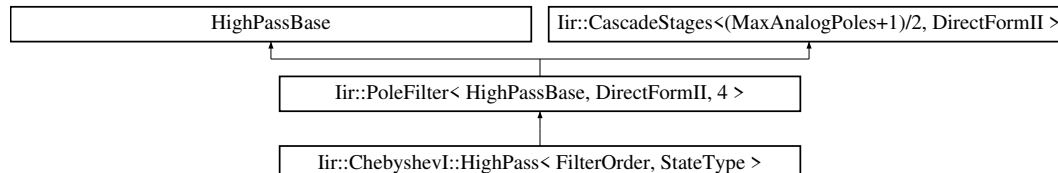
The documentation for this class was generated from the following file:

- iir/State.h

6.39 Iir::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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double rippleDb)
- void [setupN](#) (double cutoffFrequency, double rippleDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double rippleDb)

6.39.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
```

```
struct Iir::ChebyshevI::HighPass< FilterOrder, StateType >
```

[ChebyshevI](#) highpass filter

Parameters

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

6.39.2 Member Function Documentation

6.39.2.1 setup() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setup (`
`double sampleRate,`
`double cutoffFrequency,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at the order `FilterOrder`

Parameters

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

6.39.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double cutoffFrequency,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.39.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at the order `FilterOrder`

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.39.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>rippleDb</i>	Permitted ripples in dB in the passband

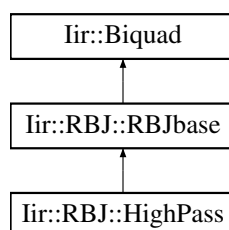
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.40 Iir::RBJ::HighPass Struct Reference

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

Inheritance diagram for Iir::RBJ::HighPass:



Public Member Functions

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

6.40.1 Detailed Description

Highpass.

6.40.2 Member Function Documentation

6.40.2.1 setup() void Iir::RBJ::HighPass::setup (
double *sampleRate*,
double *cutoffFrequency*,
double *q* = (1/sqrt(2))) [inline]

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.

6.40.2.2 setupN() void Iir::RBJ::HighPass::setupN (
double *cutoffFrequency*,
double *q* = (1/sqrt(2)))

Calculates the coefficients

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>q</i>	Q factor determines the resonance peak at the cutoff.

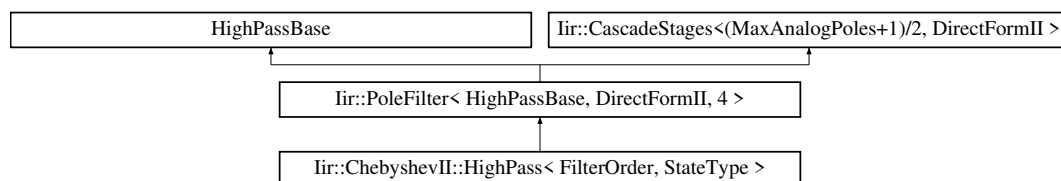
The documentation for this struct was generated from the following files:

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

6.41 `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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double stopBandDb)
- void [setupN](#) (double cutoffFrequency, double stopBandDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double stopBandDb)

6.41.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
```

```
struct Iir::ChebyshevII::HighPass< FilterOrder, StateType >
```

[ChebyshevII](#) highpass filter

Parameters

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

6.41.2 Member Function Documentation

6.41.2.1 `setup()` [1/2] `template<int FilterOrder = 4, 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.

Parameters

<i>stopBandDb</i>	Permitted ripples in dB in the stopband
-------------------	---

6.41.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighPass< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.41.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighPass< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.41.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighPass< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

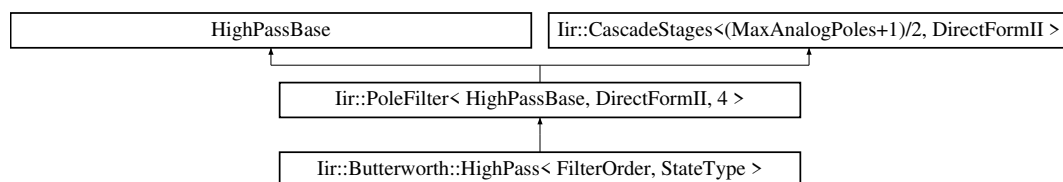
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.42 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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency)
- void [setupN](#) (double cutoffFrequency)
- void [setupN](#) (int reqOrder, double cutoffFrequency)

6.42.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::Butterworth::HighPass< FilterOrder, StateType >
```

[Butterworth](#) Highpass filter.

Parameters

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

6.42.2 Member Function Documentation

6.42.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::HighPass< FilterOrder, StateType >::setup (`
 double *sampleRate*,
 double *cutoffFrequency*) `[inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

6.42.2.2 [setup\(\)](#) [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::HighPass< FilterOrder, StateType >::setup (`
 int *reqOrder*,
 double *sampleRate*,
 double *cutoffFrequency*) `[inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate

Parameters

<i>cutoffFrequency</i>	Cutoff frequency
------------------------	------------------

6.42.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::HighPass< FilterOrder, StateType >::setupN (
double cutoffFrequency) [inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
------------------------	--------------------------------------

6.42.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::HighPass< FilterOrder, StateType >::setupN (
int reqOrder,
double cutoffFrequency) [inline]`

Calculates the coefficients

Parameters

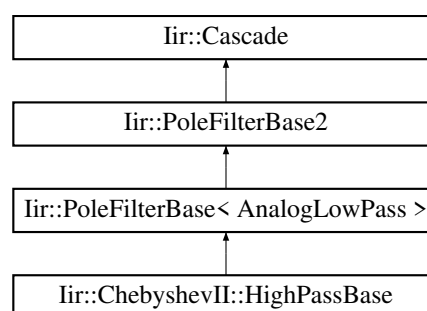
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)

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

- iir/Butterworth.h

6.43 Iir::ChebyshevII::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighPassBase:



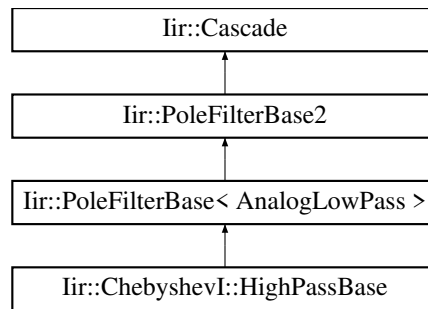
Additional Inherited Members

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

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

6.44 Iir::ChebyshevI::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighPassBase:



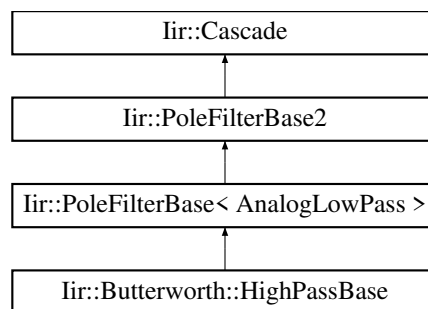
Additional Inherited Members

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

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

6.45 Iir::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

6.46 Iir::HighPassTransform Class Reference

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

6.46.1 Detailed Description

low pass to high pass

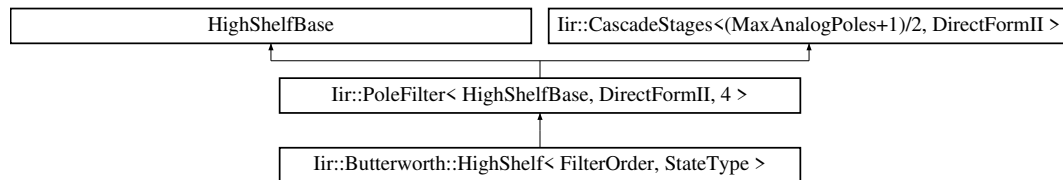
The documentation for this class was generated from the following files:

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

6.47 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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double gainDb)
- void [setupN](#) (double cutoffFrequency, double gainDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double gainDb)

6.47.1 Detailed Description

```
template<int FilterOrder = 4, 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>	Reserves memory for a filter of the order FilterOrder
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.47.2 Member Function Documentation

6.47.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

6.47.2.2 [setup\(\)](#) [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

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

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
-----------------	--

Parameters

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

6.47.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in dB of the filter in the passband

6.47.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in dB of the filter in the passband

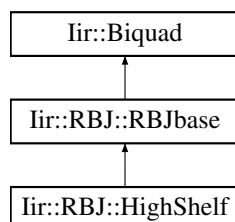
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.48 Iir::RBJ::HighShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::HighShelf:



Public Member Functions

- void [setupN](#) (double cutoffFrequency, double gainDb, double shelfSlope=1)
- 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) [inline]`

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.

6.48.2.2 setupN() `void Iir::RBJ::HighShelf::setupN (`
`double cutoffFrequency,`
`double gainDb,`
`double shelfSlope = 1)`

Calculates the coefficients

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency
<i>gainDb</i>	Gain in the passband
<i>shelfSlope</i>	Slope between stop/passband. 1 = as steep as it can.

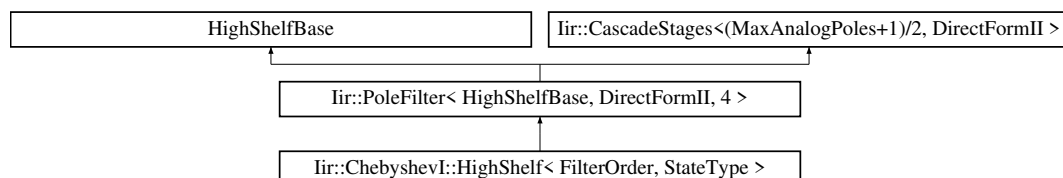
The documentation for this struct was generated from the following files:

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

- void [setupN](#) (double cutoffFrequency, double gainDb, double rippleDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double gainDb, double rippleDb)

6.49.1 Detailed Description

```
template<int FilterOrder = 4, 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>	Reserves memory for a filter of the order FilterOrder
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.49.2 Member Function Documentation

6.49.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, 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 at the order FilterOrder

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

6.49.2.2 [setup\(\)](#) [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setup (`
 int *reqOrder*,
 double *sampleRate*,
 double *cutoffFrequency*,
 double *gainDb*,
 double *rippleDb*) [inline]

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<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

6.49.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double gainDb,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.49.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double gainDb,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

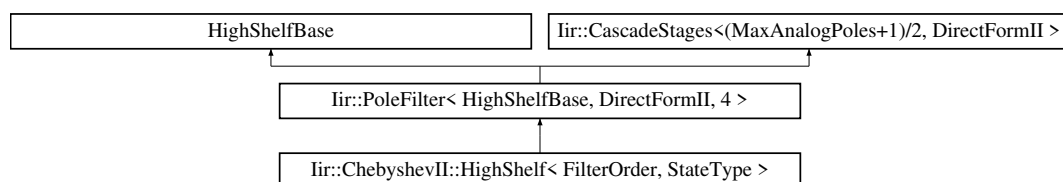
The documentation for this struct was generated from the following file:

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

6.50.1 Detailed Description

```
template<int FilterOrder = 4, 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>	Reserves memory for a filter of the order <i>FilterOrder</i>
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.50.2 Member Function Documentation

6.50.2.1 setup() [1/2] `template<int FilterOrder = 4, 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

6.50.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighShelf< FilterOrder, StateType >::setup (`
 `int reqOrder,`
 `double sampleRate,`
 `double cutoffFrequency,`
 `double gainDb,`
 `double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<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

6.50.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighShelf< FilterOrder, StateType >::setupN (`
 `double cutoffFrequency,`
 `double gainDb,`
 `double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.50.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::HighShelf< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double gainDb,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

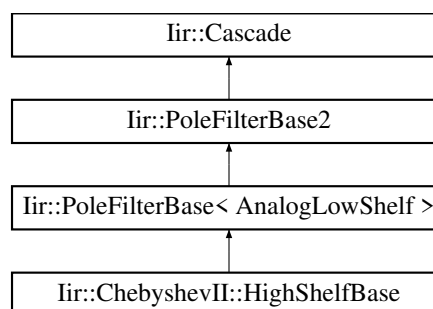
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

- iir/ChebyshevII.h

6.51 Iir::ChebyshevII::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighShelfBase:



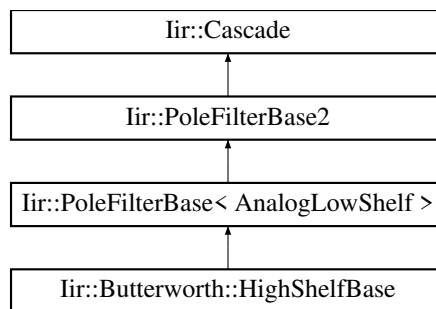
Additional Inherited Members

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

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

6.52 Iir::Butterworth::HighShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighShelfBase:



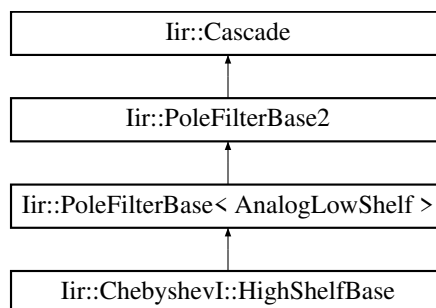
Additional Inherited Members

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

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

6.53 Iir::ChebyshevI::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighShelfBase:



Additional Inherited Members

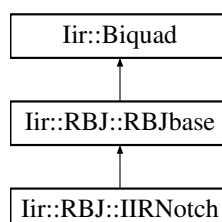
The documentation for this struct was generated from the following files:

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

6.54 Iir::RBJ::IRNotch Struct Reference

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

Inheritance diagram for Iir::RBJ::IRNotch:



Public Member Functions

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

6.54.1 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.

6.54.2 Member Function Documentation

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

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)

6.54.2.2 setupN() `void Iir::RBJ::IIRNotch::setupN (`
`double centerFrequency,`
`double q_factor = 10)`

Calculates the coefficients

Parameters

<i>centerFrequency</i>	Normalised centre 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

6.55 Iir::Layout< MaxPoles > Class Template Reference

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

6.55.1 Detailed Description

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

Storage for [Layout](#)

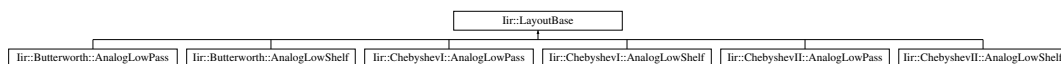
The documentation for this class was generated from the following file:

- iir/Layout.h

6.56 Iir::LayoutBase Class Reference

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

Inheritance diagram for Iir::LayoutBase:



6.56.1 Detailed Description

Base uses pointers to reduce template instantiations

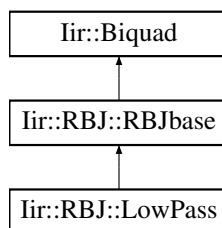
The documentation for this class was generated from the following file:

- iir/Layout.h

6.57 Iir::RBJ::LowPass Struct Reference

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

Inheritance diagram for Iir::RBJ::LowPass:



Public Member Functions

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

6.57.1 Detailed Description

Lowpass.

6.57.2 Member Function Documentation

6.57.2.1 setup() void Iir::RBJ::LowPass::setup (
double *sampleRate*,
double *cutoffFrequency*,
double *q* = (1/sqrt(2))) [inline]

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.

6.57.2.2 setupN() void Iir::RBJ::LowPass::setupN (
double *cutoffFrequency*,
double *q* = (1/sqrt(2)))

Calculates the coefficients

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency
<i>q</i>	Q factor determines the resonance peak at the cutoff.

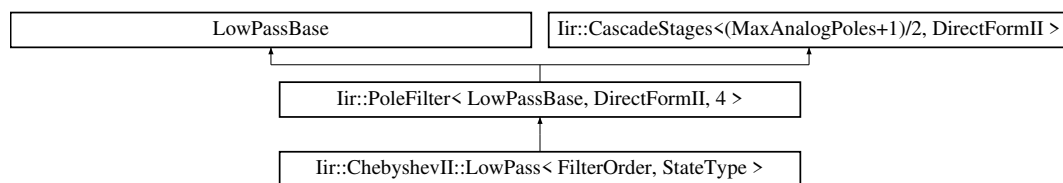
The documentation for this struct was generated from the following files:

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

6.58 `lir::ChebyshevII::LowPass< FilterOrder, StateType >` Struct Template Reference

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

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



Public Member Functions

- void [setup](#) (double *sampleRate*, double *cutoffFrequency*, double *stopBandDb*)
- void [setup](#) (int *reqOrder*, double *sampleRate*, double *cutoffFrequency*, double *stopBandDb*)
- void [setupN](#) (double *cutoffFrequency*, double *stopBandDb*)
- void [setupN](#) (int *reqOrder*, double *cutoffFrequency*, double *stopBandDb*)

6.58.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct lir::ChebyshevII::LowPass< FilterOrder, StateType >
```

[ChebyshevII](#) lowpass filter

Parameters

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

6.58.2 Member Function Documentation

```
6.58.2.1 setup() [1/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevII::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.

Parameters

<i>stopBandDb</i>	Permitted ripples in dB in the stopband
-------------------	---

6.58.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowPass< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.58.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowPass< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.58.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowPass< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double stopBandDb) [inline]`

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

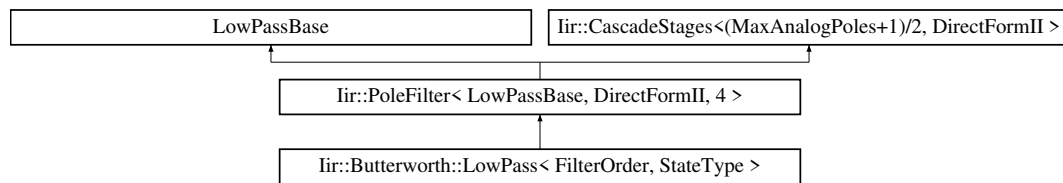
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.59 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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency)
- void [setupN](#) (double cutoffFrequency)
- void [setupN](#) (int reqOrder, double cutoffFrequency)

6.59.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::Butterworth::LowPass< FilterOrder, StateType >
```

[Butterworth](#) Lowpass filter.

Parameters

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

6.59.2 Member Function Documentation

6.59.2.1 [setup\(\)](#) [1/2] `template<int FilterOrder = 4, 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

6.59.2.2 [setup\(\)](#) [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::LowPass< FilterOrder, StateType >::setup (`
 int *reqOrder*,
 double *sampleRate*,
 double *cutoffFrequency*) `[inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate

Parameters

<i>cutoffFrequency</i>	Cutoff
------------------------	--------

6.59.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::LowPass< FilterOrder, StateType >::setupN (
 double cutoffFrequency) [inline]`

Calculates the coefficients

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
------------------------	--------------------------------------

6.59.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::Butterworth::LowPass< FilterOrder, StateType >::setupN (
 int reqOrder,
 double cutoffFrequency) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)

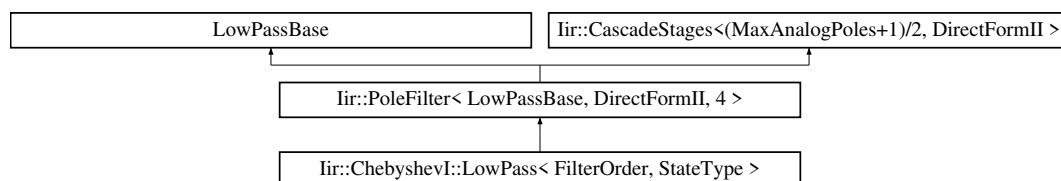
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.60 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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double rippleDb)
- void [setupN](#) (double cutoffFrequency, double rippleDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double rippleDb)

6.60.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>  
struct Iir::ChebyshevI::LowPass< FilterOrder, StateType >
```

[ChebyshevI](#) lowpass filter

Parameters

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

6.60.2 Member Function Documentation

6.60.2.1 setup() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup (
```

```
    double sampleRate,
    double cutoffFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

6.60.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup (
```

```
    int reqOrder,
    double sampleRate,
    double cutoffFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.60.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setupN (
```

```
    double cutoffFrequency,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.60.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

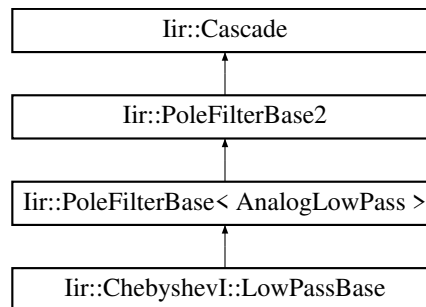
<i>reqOrder</i>	Actual order for the filter calculations
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>rippleDb</i>	Permitted ripples in dB in the passband

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

- iir/ChebyshevI.h

6.61 Iir::ChebyshevI::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowPassBase:



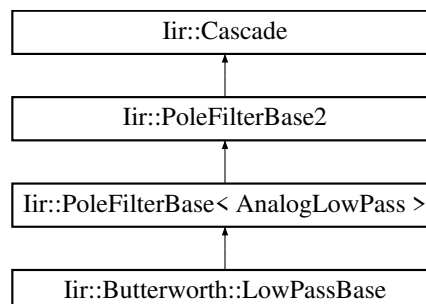
Additional Inherited Members

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

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

6.62 Iir::Butterworth::LowPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowPassBase:



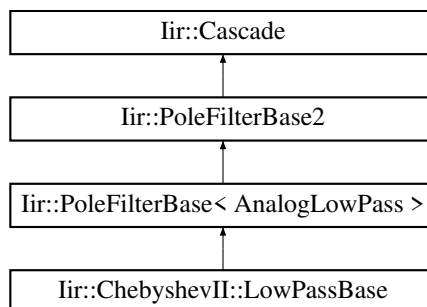
Additional Inherited Members

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::LowPassTransform Class Reference

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

6.64.1 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

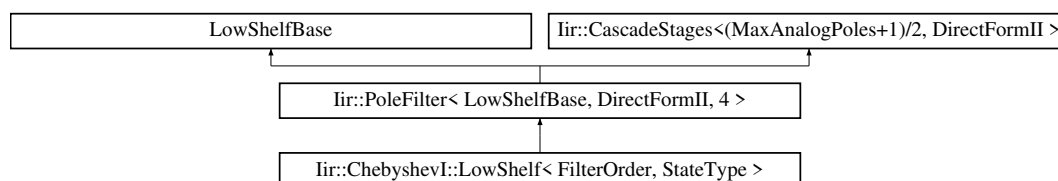
The documentation for this class was generated from the following files:

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

6.65 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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double gainDb, double rippleDb)
- void [setupN](#) (double cutoffFrequency, double gainDb, double rippleDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double gainDb, double rippleDb)

6.65.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct Iir::ChebyshevI::LowShelf< FilterOrder, StateType >
```

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

Parameters

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

6.65.2 Member Function Documentation

6.65.2.1 setup() [1/2] `template<int FilterOrder = 4, 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 at the order *FilterOrder*

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

6.65.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::LowShelf< FilterOrder, StateType >::setup (`
 `int reqOrder,`
 `double sampleRate,`
 `double cutoffFrequency,`
 `double gainDb,`
 `double rippleDb) [inline]`

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<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

6.65.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevI::LowShelf< FilterOrder, StateType >::setupN (`
 `double cutoffFrequency,`

```
double gainDb,
double rippleDb ) [inline]
```

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

6.65.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`

```
void Iir::ChebyshevI::LowShelf< FilterOrder, StateType >::setupN (
    int reqOrder,
    double cutoffFrequency,
    double gainDb,
    double rippleDb ) [inline]
```

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

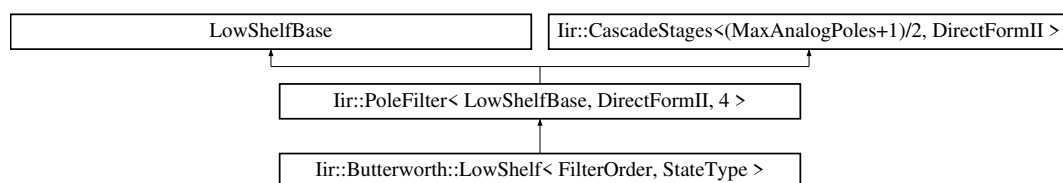
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.66 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)
- void `setup` (int reqOrder, double sampleRate, double cutoffFrequency, double gainDb)
- void `setupN` (double cutoffFrequency, double gainDb)
- void `setupN` (int reqOrder, double cutoffFrequency, double gainDb)

6.66.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
```

```
struct Iir::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>	Reserves memory for a filter of the order <i>FilterOrder</i>
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.66.2 Member Function Documentation

6.66.2.1 `setup()` [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setup (`
`double sampleRate,`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

6.66.2.2 `setup()` [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setup (`
`int reqOrder,`
`double sampleRate,`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff
<i>gainDb</i>	Gain in dB of the filter in the passband

6.66.2.3 `setupN()` [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setupN (`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients with the filter order provided by the instantiation

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in dB of the filter in the passband

6.66.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setupN (`
`int reqOrder,`
`double cutoffFrequency,`
`double gainDb) [inline]`

Calculates the coefficients

Parameters

<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain in dB of the filter in the passband

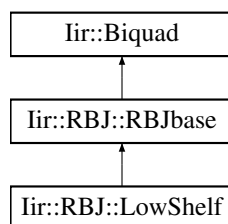
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.67 Iir::RBJ::LowShelf Struct Reference

`#include <RBJ.h>`

Inheritance diagram for Iir::RBJ::LowShelf:



Public Member Functions

- void [setupN](#) (double cutoffFrequency, double gainDb, double shelfSlope=1)
- 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) [inline]`

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.

6.67.2.2 setupN() `void Iir::RBJ::LowShelf::setupN (`
`double cutoffFrequency,`
`double gainDb,`
`double shelfSlope = 1)`

Calculates the coefficients

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency
<i>gainDb</i>	Gain in the passband
<i>shelfSlope</i>	Slope between stop/passband. 1 = as steep as it can.

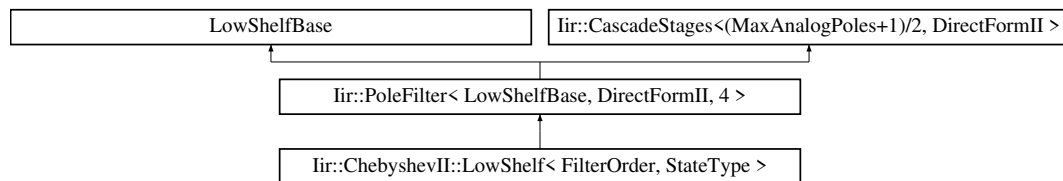
The documentation for this struct was generated from the following files:

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

6.68 Iir::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)
- void [setup](#) (int reqOrder, double sampleRate, double cutoffFrequency, double gainDb, double stopBandDb)
- void [setupN](#) (double cutoffFrequency, double gainDb, double stopBandDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double gainDb, double stopBandDb)

6.68.1 Detailed Description

```
template<int FilterOrder = 4, 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>	Reserves memory for a filter of the order <code>FilterOrder</code>
<i>StateType</i>	The filter topology: DirectFormI , DirectFormII , ...

6.68.2 Member Function Documentation

6.68.2.1 setup() [1/2] `template<int FilterOrder = 4, 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 of the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.68.2.2 setup() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowShelf< FilterOrder, StateType >::setup (`

```
int reqOrder,
double sampleRate,
double cutoffFrequency,
double gainDb,
double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>gainDb</i>	Gain of the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.68.2.3 setupN() [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowShelf< FilterOrder, StateType >::setupN (`

```
double cutoffFrequency,
double gainDb,
double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain of the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

6.68.2.4 setupN() [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void Iir::ChebyshevII::LowShelf< FilterOrder, StateType >::setupN (`

```
int reqOrder,
double cutoffFrequency,
double gainDb,
double stopBandDb ) [inline]
```

Calculates the coefficients of the filter

Parameters

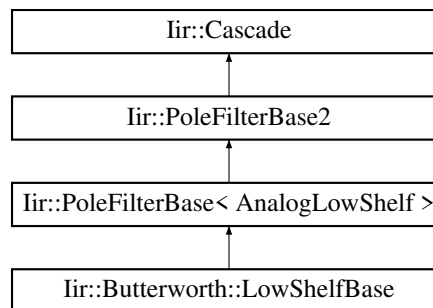
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

- iir/ChebyshevII.h

6.69 Iir::Butterworth::LowShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowShelfBase:



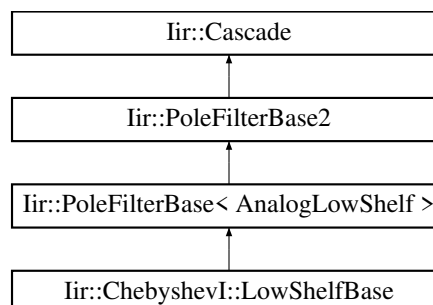
Additional Inherited Members

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

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

6.70 Iir::ChebyshevI::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowShelfBase:



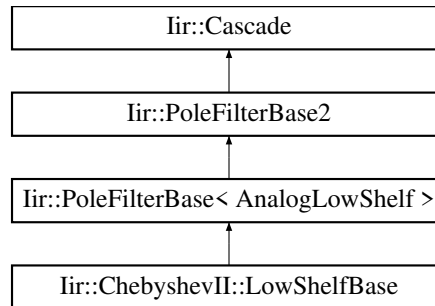
Additional Inherited Members

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

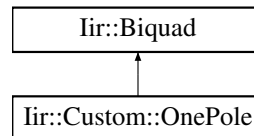
The documentation for this struct was generated from the following files:

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

6.72 Iir::Custom::OnePole Struct Reference

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

Inheritance diagram for Iir::Custom::OnePole:



Additional Inherited Members

6.72.1 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

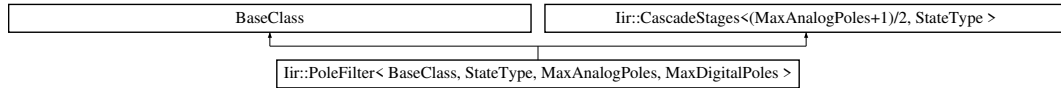
The documentation for this struct was generated from the following files:

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

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

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

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



Additional Inherited Members

6.73.1 Detailed Description

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

Storage for pole filters

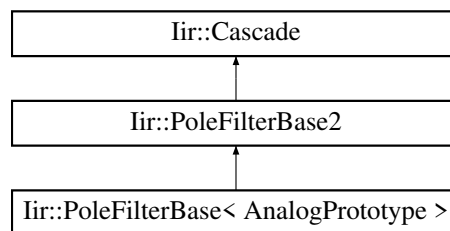
The documentation for this struct was generated from the following file:

- iir/PoleFilter.h

6.74 Iir::PoleFilterBase< AnalogPrototype > Class Template Reference

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

Inheritance diagram for Iir::PoleFilterBase< AnalogPrototype >:



Additional Inherited Members

6.74.1 Detailed Description

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

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

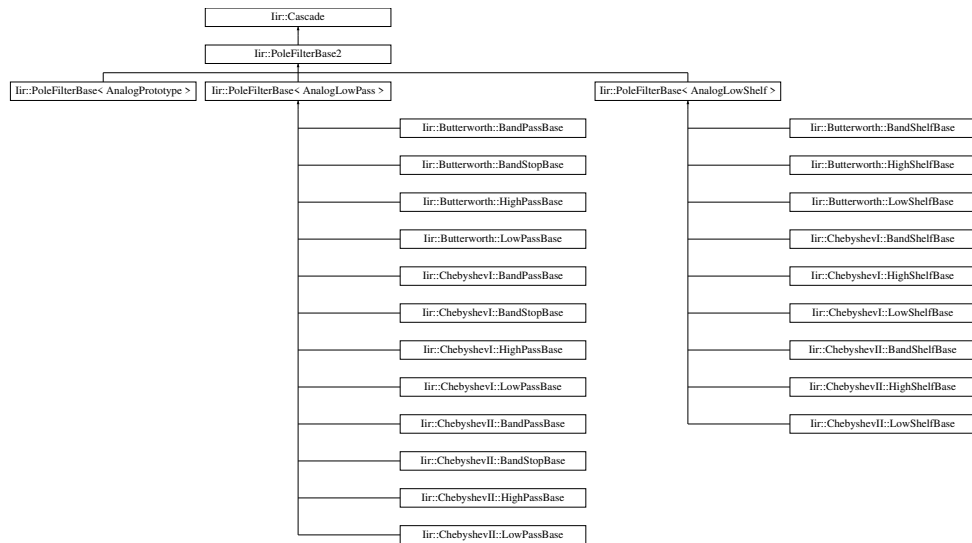
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.75 Iir::PoleFilterBase2 Class Reference

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

Inheritance diagram for Iir::PoleFilterBase2:



Additional Inherited Members

6.75.1 Detailed Description

Factored implementations to reduce template instantiations

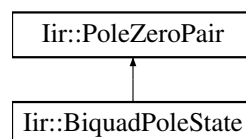
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.76 Iir::PoleZeroPair Struct Reference

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

Inheritance diagram for Iir::PoleZeroPair:



6.76.1 Detailed Description

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

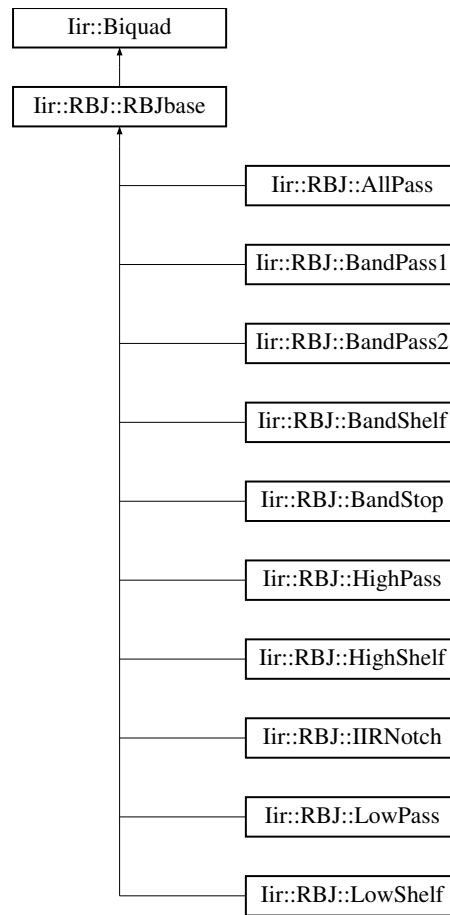
The documentation for this struct was generated from the following file:

- iir/Types.h

6.77 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

6.77.1 Detailed Description

The base class of all RBJ filters

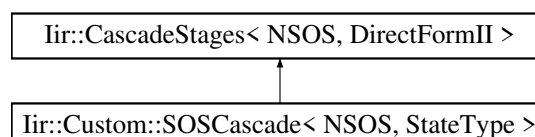
The documentation for this struct was generated from the following file:

- `iir/RBJ.h`

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

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

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



Public Member Functions

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

6.78.1 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 , ...

6.78.2 Constructor & Destructor Documentation

6.78.2.1 SOSCascade() [1/2] `template<int NSOS, class StateType = DirectFormII>`

`Iir::Custom::SOSCascade< NSOS, StateType >::SOSCascade ()`

Default constructor which creates a unity gain filter of NSOS biquads. Set the filter coefficients later with the [setup\(\)](#) method.

6.78.2.2 SOSCascade() [2/2] `template<int NSOS, class StateType = DirectFormII>`

`Iir::Custom::SOSCascade< NSOS, StateType >::SOSCascade (`
`const double(&) sosCoefficients[NSOS][6]) [inline]`

Python scipy.signal-friendly setting of coefficients. Initialises 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.
------------------------	---

6.78.3 Member Function Documentation

6.78.3.1 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

6.79 Iir::Cascade::Storage Struct Reference

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

Public Member Functions

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

6.79.1 Detailed Description

Pointer to an array of Biquads

6.79.2 Constructor & Destructor Documentation

6.79.2.1 Storage() Iir::Cascade::Storage::Storage (
 int maxStages_,
 [Biquad](#) *const stageArray_) [inline]

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

Parameters

<i>maxStages_</i>	Number of biquads
<i>stageArray_</i>	The array of the Biquads

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

- iir/Cascade.h

6.80 Iir::TransposedDirectFormII Class Reference

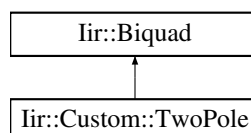
The documentation for this class was generated from the following file:

- iir/State.h

6.81 Iir::Custom::TwoPole Struct Reference

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

Inheritance diagram for Iir::Custom::TwoPole:



Additional Inherited Members

6.81.1 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

- applyScale
 - lir::Biquad, [45](#)
- filter
 - lir::Biquad, [46](#)
 - lir::CascadeStages< MaxStages, StateType >, [49](#)
- getA0
 - lir::Biquad, [46](#)
- getA1
 - lir::Biquad, [46](#)
- getA2
 - lir::Biquad, [46](#)
- getB0
 - lir::Biquad, [46](#)
- getB1
 - lir::Biquad, [46](#)
- getB2
 - lir::Biquad, [46](#)
- getCascadeState
 - lir::CascadeStages< MaxStages, StateType >, [49](#)
- getCascadeStorage
 - lir::CascadeStages< MaxStages, StateType >, [49](#)
- getNumStages
 - lir::Cascade, [48](#)
- getPoleZeros
 - lir::Biquad, [46](#)
 - lir::Cascade, [48](#)
- lir, [12](#)
 - Kind, [14](#)
- lir::BandPassTransform, [28](#)
- lir::BandStopTransform, [44](#)
- lir::Biquad, [44](#)
 - applyScale, [45](#)
 - filter, [46](#)
 - getA0, [46](#)
 - getA1, [46](#)
 - getA2, [46](#)
 - getB0, [46](#)
 - getB1, [46](#)
 - getB2, [46](#)
 - getPoleZeros, [46](#)
 - response, [46](#)
 - setCoefficients, [46](#)
 - setIdentity, [47](#)
 - setOnePole, [47](#)
 - setPoleZeroPair, [47](#)
 - setTwoPole, [47](#)
- lir::BiquadPoleState, [47](#)
- lir::Butterworth, [14](#)
- lir::Butterworth::AnalogLowPass, [17](#)
- lir::Butterworth::AnalogLowShelf, [18](#)
- lir::Butterworth::BandPass< FilterOrder, StateType >, [21](#)
 - setup, [21](#), [22](#)
 - setupN, [22](#)
- lir::Butterworth::BandPassBase, [27](#)
- lir::Butterworth::BandShelf< FilterOrder, StateType >, [28](#)
 - setup, [28](#), [29](#)
 - setupN, [29](#)
- lir::Butterworth::BandShelfBase, [35](#)
- lir::Butterworth::BandStop< FilterOrder, StateType >, [41](#)
 - setup, [42](#)
 - setupN, [42](#)
- lir::Butterworth::BandStopBase, [43](#)
- lir::Butterworth::HighPass< FilterOrder, StateType >, [55](#)
 - setup, [56](#)
 - setupN, [57](#)
- lir::Butterworth::HighPassBase, [58](#)
- lir::Butterworth::HighShelf< FilterOrder, StateType >, [58](#)
 - setup, [59](#)
 - setupN, [60](#)
- lir::Butterworth::HighShelfBase, [65](#)
- lir::Butterworth::LowPass< FilterOrder, StateType >, [70](#)
 - setup, [71](#)
 - setupN, [72](#)
- lir::Butterworth::LowPassBase, [74](#)
- lir::Butterworth::LowShelf< FilterOrder, StateType >, [77](#)
 - setup, [78](#)
 - setupN, [78](#)
- lir::Butterworth::LowShelfBase, [82](#)
- lir::Cascade, [48](#)
 - getNumStages, [48](#)
 - getPoleZeros, [48](#)
 - operator[], [48](#)
 - response, [48](#)
- lir::Cascade::Storage, [88](#)
 - Storage, [88](#)
- lir::CascadeStages< MaxStages, StateType >, [49](#)
 - filter, [49](#)
 - getCascadeState, [49](#)
 - getCascadeStorage, [49](#)
 - reset, [50](#)
 - setup, [50](#)
- lir::ChebyshevI, [15](#)
- lir::ChebyshevI::AnalogLowPass, [17](#)
- lir::ChebyshevI::AnalogLowShelf, [19](#)
- lir::ChebyshevI::BandPass< FilterOrder, StateType >, [19](#)
 - setup, [19](#), [20](#)
 - setupN, [20](#)
- lir::ChebyshevI::BandPassBase, [27](#)
- lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [31](#)
 - setup, [31](#), [32](#)
 - setupN, [32](#)

- lir::ChebyshevI::BandShelfBase, [36](#)
- lir::ChebyshevI::BandStop< FilterOrder, StateType >, [36](#)
 - setup, [37](#)
 - setupN, [37](#), [38](#)
- lir::ChebyshevI::BandStopBase, [44](#)
- lir::ChebyshevI::HighPass< FilterOrder, StateType >, [51](#)
 - setup, [52](#)
 - setupN, [52](#)
- lir::ChebyshevI::HighPassBase, [57](#)
- lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [61](#)
 - setup, [62](#)
 - setupN, [62](#), [63](#)
- lir::ChebyshevI::HighShelfBase, [66](#)
- lir::ChebyshevI::LowPass< FilterOrder, StateType >, [72](#)
 - setup, [73](#)
 - setupN, [73](#)
- lir::ChebyshevI::LowPassBase, [74](#)
- lir::ChebyshevI::LowShelf< FilterOrder, StateType >, [75](#)
 - setup, [76](#)
 - setupN, [76](#), [77](#)
- lir::ChebyshevI::LowShelfBase, [82](#)
- lir::ChebyshevII, [15](#)
- lir::ChebyshevII::AnalogLowPass, [18](#)
- lir::ChebyshevII::AnalogLowShelf, [18](#)
- lir::ChebyshevII::BandPass< FilterOrder, StateType >, [23](#)
 - setup, [23](#)
 - setupN, [24](#)
- lir::ChebyshevII::BandPassBase, [26](#)
- lir::ChebyshevII::BandShelf< FilterOrder, StateType >, [33](#)
 - setup, [34](#)
 - setupN, [34](#), [35](#)
- lir::ChebyshevII::BandShelfBase, [35](#)
- lir::ChebyshevII::BandStop< FilterOrder, StateType >, [39](#)
 - setup, [40](#)
 - setupN, [40](#), [41](#)
- lir::ChebyshevII::BandStopBase, [43](#)
- lir::ChebyshevII::HighPass< FilterOrder, StateType >, [54](#)
 - setup, [54](#), [55](#)
 - setupN, [55](#)
- lir::ChebyshevII::HighPassBase, [57](#)
- lir::ChebyshevII::HighShelf< FilterOrder, StateType >, [63](#)
 - setup, [64](#)
 - setupN, [64](#), [65](#)
- lir::ChebyshevII::HighShelfBase, [65](#)
- lir::ChebyshevII::LowPass< FilterOrder, StateType >, [69](#)
 - setup, [69](#), [70](#)
 - setupN, [70](#)
- lir::ChebyshevII::LowPassBase, [75](#)
- lir::ChebyshevII::LowShelf< FilterOrder, StateType >, [80](#)
 - setup, [80](#), [81](#)
 - setupN, [81](#)
- lir::ChebyshevII::LowShelfBase, [83](#)
- lir::ComplexPair, [50](#)
 - isMatchedPair, [50](#)
- lir::Custom, [16](#)
- lir::Custom::OnePole, [83](#)
- lir::Custom::SOSCascade< NSOS, StateType >, [86](#)
 - setup, [87](#)
 - SOSCascade, [87](#)
- lir::Custom::TwoPole, [88](#)
- lir::DirectFormI, [50](#)
- lir::DirectFormII, [51](#)
- lir::HighPassTransform, [58](#)
- lir::Layout< MaxPoles >, [67](#)
- lir::LayoutBase, [67](#)
- lir::LowPassTransform, [75](#)
- lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >, [83](#)
- lir::PoleFilterBase< AnalogPrototype >, [84](#)
- lir::PoleFilterBase2, [84](#)
- lir::PoleZeroPair, [85](#)
- lir::RBJ::AllPass, [16](#)
 - setup, [16](#)
 - setupN, [17](#)
- lir::RBJ::BandPass1, [24](#)
 - setup, [25](#)
 - setupN, [25](#)
- lir::RBJ::BandPass2, [25](#)
 - setup, [26](#)
 - setupN, [26](#)
- lir::RBJ::BandShelf, [30](#)
 - setup, [30](#)
 - setupN, [30](#)
- lir::RBJ::BandStop, [38](#)
 - setup, [38](#)
 - setupN, [39](#)
- lir::RBJ::HighPass, [53](#)
 - setup, [53](#)
 - setupN, [53](#)
- lir::RBJ::HighShelf, [60](#)
 - setup, [61](#)
 - setupN, [61](#)
- lir::RBJ::IIRNotch, [66](#)
 - setup, [67](#)
 - setupN, [67](#)
- lir::RBJ::LowPass, [68](#)
 - setup, [68](#)
 - setupN, [68](#)
- lir::RBJ::LowShelf, [79](#)
 - setup, [79](#)
 - setupN, [80](#)
- lir::RBJ::RBJbase, [85](#)
- lir::TransposedDirectFormII, [88](#)
- isMatchedPair
 - lir::ComplexPair, [50](#)

- Kind
 - [lir](#), [14](#)
- operator[]
 - [lir::Cascade](#), [48](#)
- reset
 - [lir::CascadeStages](#)< MaxStages, StateType >, [50](#)
- response
 - [lir::Biquad](#), [46](#)
 - [lir::Cascade](#), [48](#)
- setCoefficients
 - [lir::Biquad](#), [46](#)
- setIdentity
 - [lir::Biquad](#), [47](#)
- setOnePole
 - [lir::Biquad](#), [47](#)
- setPoleZeroPair
 - [lir::Biquad](#), [47](#)
- setTwoPole
 - [lir::Biquad](#), [47](#)
- setup
 - [lir::Butterworth::BandPass](#)< FilterOrder, StateType >, [21](#), [22](#)
 - [lir::Butterworth::BandShelf](#)< FilterOrder, StateType >, [28](#), [29](#)
 - [lir::Butterworth::BandStop](#)< FilterOrder, StateType >, [42](#)
 - [lir::Butterworth::HighPass](#)< FilterOrder, StateType >, [56](#)
 - [lir::Butterworth::HighShelf](#)< FilterOrder, StateType >, [59](#)
 - [lir::Butterworth::LowPass](#)< FilterOrder, StateType >, [71](#)
 - [lir::Butterworth::LowShelf](#)< FilterOrder, StateType >, [78](#)
 - [lir::CascadeStages](#)< MaxStages, StateType >, [50](#)
 - [lir::ChebyshevI::BandPass](#)< FilterOrder, StateType >, [19](#), [20](#)
 - [lir::ChebyshevI::BandShelf](#)< FilterOrder, StateType >, [31](#), [32](#)
 - [lir::ChebyshevI::BandStop](#)< FilterOrder, StateType >, [37](#)
 - [lir::ChebyshevI::HighPass](#)< FilterOrder, StateType >, [52](#)
 - [lir::ChebyshevI::HighShelf](#)< FilterOrder, StateType >, [62](#)
 - [lir::ChebyshevI::LowPass](#)< FilterOrder, StateType >, [73](#)
 - [lir::ChebyshevI::LowShelf](#)< FilterOrder, StateType >, [76](#)
 - [lir::ChebyshevII::BandPass](#)< FilterOrder, StateType >, [23](#)
 - [lir::ChebyshevII::BandShelf](#)< FilterOrder, StateType >, [34](#)
 - [lir::ChebyshevII::BandStop](#)< FilterOrder, StateType >, [40](#)
 - [lir::ChebyshevII::HighPass](#)< FilterOrder, StateType >, [54](#), [55](#)
 - [lir::ChebyshevII::HighShelf](#)< FilterOrder, StateType >, [64](#)
 - [lir::ChebyshevII::LowPass](#)< FilterOrder, StateType >, [69](#), [70](#)
 - [lir::ChebyshevII::LowShelf](#)< FilterOrder, StateType >, [80](#), [81](#)
 - [lir::Custom::SOSCascade](#)< NSOS, StateType >, [87](#)
 - [lir::RBJ::AllPass](#), [16](#)
 - [lir::RBJ::BandPass1](#), [25](#)
 - [lir::RBJ::BandPass2](#), [26](#)
 - [lir::RBJ::BandShelf](#), [30](#)
 - [lir::RBJ::BandStop](#), [38](#)
 - [lir::RBJ::HighPass](#), [53](#)
 - [lir::RBJ::HighShelf](#), [61](#)
 - [lir::RBJ::IIRNotch](#), [67](#)
 - [lir::RBJ::LowPass](#), [68](#)
 - [lir::RBJ::LowShelf](#), [79](#)
- setupN
 - [lir::Butterworth::BandPass](#)< FilterOrder, StateType >, [22](#)
 - [lir::Butterworth::BandShelf](#)< FilterOrder, StateType >, [29](#)
 - [lir::Butterworth::BandStop](#)< FilterOrder, StateType >, [42](#)
 - [lir::Butterworth::HighPass](#)< FilterOrder, StateType >, [57](#)
 - [lir::Butterworth::HighShelf](#)< FilterOrder, StateType >, [60](#)
 - [lir::Butterworth::LowPass](#)< FilterOrder, StateType >, [72](#)
 - [lir::Butterworth::LowShelf](#)< FilterOrder, StateType >, [78](#)
 - [lir::ChebyshevI::BandPass](#)< FilterOrder, StateType >, [20](#)
 - [lir::ChebyshevI::BandShelf](#)< FilterOrder, StateType >, [32](#)
 - [lir::ChebyshevI::BandStop](#)< FilterOrder, StateType >, [37](#), [38](#)
 - [lir::ChebyshevI::HighPass](#)< FilterOrder, StateType >, [52](#)
 - [lir::ChebyshevI::HighShelf](#)< FilterOrder, StateType >, [62](#), [63](#)
 - [lir::ChebyshevI::LowPass](#)< FilterOrder, StateType >, [73](#)
 - [lir::ChebyshevI::LowShelf](#)< FilterOrder, StateType >, [76](#), [77](#)
 - [lir::ChebyshevII::BandPass](#)< FilterOrder, StateType >, [24](#)
 - [lir::ChebyshevII::BandShelf](#)< FilterOrder, StateType >, [34](#), [35](#)
 - [lir::ChebyshevII::BandStop](#)< FilterOrder, StateType >, [40](#), [41](#)
 - [lir::ChebyshevII::HighPass](#)< FilterOrder, StateType >, [55](#)
 - [lir::ChebyshevII::HighShelf](#)< FilterOrder, StateType >

Type >, [64](#), [65](#)
lir::ChebyshevII::LowPass< FilterOrder, StateType
 >, [70](#)
lir::ChebyshevII::LowShelf< FilterOrder, StateType
 >, [81](#)
lir::RBJ::AllPass, [17](#)
lir::RBJ::BandPass1, [25](#)
lir::RBJ::BandPass2, [26](#)
lir::RBJ::BandShelf, [30](#)
lir::RBJ::BandStop, [39](#)
lir::RBJ::HighPass, [53](#)
lir::RBJ::HighShelf, [61](#)
lir::RBJ::IIRNotch, [67](#)
lir::RBJ::LowPass, [68](#)
lir::RBJ::LowShelf, [80](#)
SOSCascade
 lir::Custom::SOSCascade< NSOS, StateType >,
 [87](#)
Storage
 lir::Cascade::Storage, [88](#)