

iir1

Generated by Doxygen 1.8.17

<b>1 DSP IIR Realtime C++ filter library</b>	<b>1</b>
1.1 C++ code	1
1.1.1 Setting the filter parameters	1
1.1.2 Realtime filtering sample by sample	2
1.1.3 Error handling	2
1.2 Linking	2
1.2.1 CMake setup	2
1.2.2 Generic linker setup	2
1.3 Packages for Ubuntu (xenial / bionic / focal):	3
1.4 Package for MacOS	3
1.5 Compilation from source	3
1.5.1 Linux / Mac	3
1.5.2 Windows	3
1.5.3 Unit tests	3
1.6 Documentation	4
1.6.1 Learn from the demos	4
1.6.2 Detailed documentation	4
1.7 Example filter responses	4
1.8 Credits	4
1.9 Bibliography	4
<b>2 Namespace Index</b>	<b>4</b>
2.1 Namespace List	4
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>8</b>
4.1 Class List	8
<b>5 Namespace Documentation</b>	<b>11</b>
5.1 Iir Namespace Reference	11
5.1.1 Detailed Description	12
5.1.2 Enumeration Type Documentation	13
5.2 Iir::Butterworth Namespace Reference	13
5.2.1 Detailed Description	14
5.3 Iir::ChebyshevI Namespace Reference	14
5.3.1 Detailed Description	14
5.4 Iir::ChebyshevII Namespace Reference	14
5.4.1 Detailed Description	15
5.5 Iir::Custom Namespace Reference	15
5.5.1 Detailed Description	15
<b>6 Class Documentation</b>	<b>15</b>

6.1 <code>lir::RBJ::AllPass</code> Struct Reference . . . . .	15
6.2 <code>lir::Butterworth::AnalogLowPass</code> Class Reference . . . . .	15
6.2.1 Detailed Description . . . . .	15
6.3 <code>lir::ChebyshevI::AnalogLowPass</code> Class Reference . . . . .	16
6.4 <code>lir::ChebyshevII::AnalogLowPass</code> Class Reference . . . . .	16
6.5 <code>lir::ChebyshevII::AnalogLowShelf</code> Class Reference . . . . .	16
6.6 <code>lir::Butterworth::AnalogLowShelf</code> Class Reference . . . . .	16
6.7 <code>lir::ChebyshevI::AnalogLowShelf</code> Class Reference . . . . .	17
6.8 <code>lir::ChebyshevI::BandPass&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	17
6.8.1 Detailed Description . . . . .	17
6.8.2 Member Function Documentation . . . . .	17
6.9 <code>lir::Butterworth::BandPass&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	19
6.9.1 Detailed Description . . . . .	19
6.9.2 Member Function Documentation . . . . .	19
6.10 <code>lir::ChebyshevII::BandPass&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	20
6.10.1 Detailed Description . . . . .	21
6.10.2 Member Function Documentation . . . . .	21
6.11 <code>lir::RBJ::BandPass1</code> Struct Reference . . . . .	22
6.11.1 Detailed Description . . . . .	23
6.11.2 Member Function Documentation . . . . .	23
6.12 <code>lir::RBJ::BandPass2</code> Struct Reference . . . . .	23
6.12.1 Detailed Description . . . . .	23
6.12.2 Member Function Documentation . . . . .	24
6.13 <code>lir::ChebyshevII::BandPassBase</code> Struct Reference . . . . .	24
6.14 <code>lir::ChebyshevI::BandPassBase</code> Struct Reference . . . . .	24
6.15 <code>lir::Butterworth::BandPassBase</code> Struct Reference . . . . .	25
6.16 <code>lir::BandPassTransform</code> Class Reference . . . . .	25
6.16.1 Detailed Description . . . . .	25
6.17 <code>lir::Butterworth::BandShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	25
6.17.1 Detailed Description . . . . .	26
6.17.2 Member Function Documentation . . . . .	26
6.18 <code>lir::RBJ::BandShelf</code> Struct Reference . . . . .	27
6.18.1 Detailed Description . . . . .	27
6.18.2 Member Function Documentation . . . . .	27
6.19 <code>lir::ChebyshevI::BandShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	28
6.19.1 Detailed Description . . . . .	28
6.19.2 Member Function Documentation . . . . .	28
6.20 <code>lir::ChebyshevII::BandShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference . . . . .	30
6.20.1 Detailed Description . . . . .	30
6.20.2 Member Function Documentation . . . . .	31
6.21 <code>lir::Butterworth::BandShelfBase</code> Struct Reference . . . . .	32
6.22 <code>lir::ChebyshevII::BandShelfBase</code> Struct Reference . . . . .	32

6.23 <a href="#">lir::ChebyshevI::BandShelfBase Struct Reference</a>	33
6.24 <a href="#">lir::ChebyshevI::BandStop&lt; FilterOrder, StateType &gt; Struct Template Reference</a>	33
6.24.1 Detailed Description	34
6.24.2 Member Function Documentation	34
6.25 <a href="#">lir::RBJ::BandStop Struct Reference</a>	35
6.25.1 Detailed Description	35
6.25.2 Member Function Documentation	35
6.26 <a href="#">lir::ChebyshevII::BandStop&lt; FilterOrder, StateType &gt; Struct Template Reference</a>	36
6.26.1 Detailed Description	36
6.26.2 Member Function Documentation	36
6.27 <a href="#">lir::Butterworth::BandStop&lt; FilterOrder, StateType &gt; Struct Template Reference</a>	38
6.27.1 Detailed Description	38
6.27.2 Member Function Documentation	38
6.28 <a href="#">lir::Butterworth::BandStopBase Struct Reference</a>	39
6.29 <a href="#">lir::ChebyshevII::BandStopBase Struct Reference</a>	40
6.30 <a href="#">lir::ChebyshevI::BandStopBase Struct Reference</a>	40
6.31 <a href="#">lir::BandStopTransform Class Reference</a>	41
6.31.1 Detailed Description	41
6.32 <a href="#">lir::Biquad Class Reference</a>	41
6.32.1 Member Function Documentation	42
6.33 <a href="#">lir::BiquadPoleState Struct Reference</a>	44
6.33.1 Detailed Description	44
6.34 <a href="#">lir::Cascade Class Reference</a>	44
6.34.1 Detailed Description	45
6.34.2 Member Function Documentation	45
6.35 <a href="#">lir::CascadeStages&lt; MaxStages, StateType &gt; Class Template Reference</a>	45
6.35.1 Detailed Description	45
6.35.2 Member Function Documentation	45
6.36 <a href="#">lir::ComplexPair Struct Reference</a>	46
6.36.1 Detailed Description	46
6.36.2 Member Function Documentation	46
6.37 <a href="#">lir::DirectFormI Class Reference</a>	47
6.37.1 Detailed Description	47
6.38 <a href="#">lir::DirectFormII Class Reference</a>	47
6.38.1 Detailed Description	47
6.39 <a href="#">lir::ChebyshevI::HighPass&lt; FilterOrder, StateType &gt; Struct Template Reference</a>	47
6.39.1 Detailed Description	47
6.39.2 Member Function Documentation	48
6.40 <a href="#">lir::RBJ::HighPass Struct Reference</a>	49
6.40.1 Detailed Description	49
6.40.2 Member Function Documentation	49
6.41 <a href="#">lir::ChebyshevII::HighPass&lt; FilterOrder, StateType &gt; Struct Template Reference</a>	50

6.41.1 Detailed Description	50
6.41.2 Member Function Documentation	50
6.42 <code>lir::Butterworth::HighPass&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	51
6.42.1 Detailed Description	52
6.42.2 Member Function Documentation	52
6.43 <code>lir::ChebyshevI::HighPassBase Struct Reference</code>	53
6.44 <code>lir::ChebyshevII::HighPassBase Struct Reference</code>	53
6.45 <code>lir::Butterworth::HighPassBase Struct Reference</code>	54
6.46 <code>lir::HighPassTransform Class Reference</code>	54
6.46.1 Detailed Description	54
6.47 <code>lir::RBJ::HighShelf Struct Reference</code>	54
6.47.1 Detailed Description	55
6.47.2 Member Function Documentation	55
6.48 <code>lir::ChebyshevI::HighShelf&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	55
6.48.1 Detailed Description	56
6.48.2 Member Function Documentation	56
6.49 <code>lir::Butterworth::HighShelf&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	57
6.49.1 Detailed Description	57
6.49.2 Member Function Documentation	58
6.50 <code>lir::ChebyshevII::HighShelf&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	59
6.50.1 Detailed Description	59
6.50.2 Member Function Documentation	59
6.51 <code>lir::ChebyshevII::HighShelfBase Struct Reference</code>	61
6.52 <code>lir::Butterworth::HighShelfBase Struct Reference</code>	61
6.53 <code>lir::ChebyshevI::HighShelfBase Struct Reference</code>	62
6.54 <code>lir::RBJ::IIRNotch Struct Reference</code>	62
6.54.1 Detailed Description	62
6.54.2 Member Function Documentation	62
6.55 <code>lir::Layout&lt; MaxPoles &gt; Class Template Reference</code>	63
6.55.1 Detailed Description	63
6.56 <code>lir::LayoutBase Class Reference</code>	63
6.56.1 Detailed Description	63
6.57 <code>lir::RBJ::LowPass Struct Reference</code>	63
6.57.1 Detailed Description	64
6.57.2 Member Function Documentation	64
6.58 <code>lir::ChebyshevII::LowPass&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	64
6.58.1 Detailed Description	64
6.58.2 Member Function Documentation	65
6.59 <code>lir::Butterworth::LowPass&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	66
6.59.1 Detailed Description	66
6.59.2 Member Function Documentation	66
6.60 <code>lir::ChebyshevI::LowPass&lt; FilterOrder, StateType &gt; Struct Template Reference</code>	67

6.60.1 Detailed Description	68
6.60.2 Member Function Documentation	68
6.61 <code>lir::ChebyshevI::LowPassBase</code> Struct Reference	69
6.62 <code>lir::Butterworth::LowPassBase</code> Struct Reference	69
6.63 <code>lir::ChebyshevII::LowPassBase</code> Struct Reference	70
6.64 <code>lir::LowPassTransform</code> Class Reference	70
6.64.1 Detailed Description	70
6.65 <code>lir::ChebyshevI::LowShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference	71
6.65.1 Detailed Description	71
6.65.2 Member Function Documentation	71
6.66 <code>lir::ChebyshevII::LowShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference	72
6.66.1 Detailed Description	73
6.66.2 Member Function Documentation	73
6.67 <code>lir::Butterworth::LowShelf&lt; FilterOrder, StateType &gt;</code> Struct Template Reference	74
6.67.1 Detailed Description	75
6.67.2 Member Function Documentation	75
6.68 <code>lir::RBJ::LowShelf</code> Struct Reference	76
6.68.1 Detailed Description	76
6.68.2 Member Function Documentation	77
6.69 <code>lir::Butterworth::LowShelfBase</code> Struct Reference	77
6.70 <code>lir::ChebyshevII::LowShelfBase</code> Struct Reference	77
6.71 <code>lir::ChebyshevI::LowShelfBase</code> Struct Reference	78
6.72 <code>lir::Custom::OnePole</code> Struct Reference	78
6.72.1 Detailed Description	78
6.73 <code>lir::PoleFilter&lt; BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles &gt;</code> Struct Template Reference	79
6.73.1 Detailed Description	79
6.74 <code>lir::PoleFilterBase&lt; AnalogPrototype &gt;</code> Class Template Reference	79
6.74.1 Detailed Description	79
6.75 <code>lir::PoleFilterBase2</code> Class Reference	79
6.75.1 Detailed Description	80
6.76 <code>lir::PoleZeroPair</code> Struct Reference	80
6.76.1 Detailed Description	80
6.77 <code>lir::RBJ::RBJbase</code> Struct Reference	80
6.77.1 Detailed Description	81
6.78 <code>lir::Custom::SOSCascade&lt; NSOS, StateType &gt;</code> Struct Template Reference	81
6.78.1 Detailed Description	82
6.78.2 Constructor & Destructor Documentation	82
6.78.3 Member Function Documentation	82
6.79 <code>lir::Cascade::Storage</code> Struct Reference	83
6.79.1 Detailed Description	83
6.79.2 Constructor & Destructor Documentation	83
6.80 <code>lir::TransposedDirectFormII</code> Class Reference	83

6.81 <a href="#">Iir::Custom::TwoPole Struct Reference</a> . . . . .	83
6.81.1 Detailed Description . . . . .	83
<a href="#">Index</a>	85

## 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.

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

The examples below are for lowpass filters:

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

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

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

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

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

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

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

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

## 5. Designing filters with Python's `scipy.signal` – `Custom.h`

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

<a href="#">lir</a>	11
<a href="#">lir::Butterworth</a>	13
<a href="#">lir::ChebyshevI</a>	14
<a href="#">lir::ChebyshevII</a>	14
<a href="#">lir::Custom</a>	15

## 3 Hierarchical Index

### 3.1 Class Hierarchy

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

BandPassBase

[lir::PoleFilter< BandPassBase, DirectFormII, FilterOrder, FilterOrder \\*2 >](#) 79

[lir::Butterworth::BandPass< FilterOrder, StateType >](#) 19

[lir::ChebyshevI::BandPass< FilterOrder, StateType >](#) 17

[lir::ChebyshevII::BandPass< FilterOrder, StateType >](#) 20

[lir::BandPassTransform](#) 25

BandShelfBase

[lir::PoleFilter< BandShelfBase, DirectFormII, FilterOrder, FilterOrder \\*2 >](#) 79

[lir::Butterworth::BandShelf< FilterOrder, StateType >](#) 25

[lir::ChebyshevI::BandShelf< FilterOrder, StateType >](#) 28

[lir::ChebyshevII::BandShelf< FilterOrder, StateType >](#) 30

BandStopBase

[lir::PoleFilter< BandStopBase, DirectFormII, FilterOrder, FilterOrder \\*2 >](#) 79

[lir::Butterworth::BandStop< FilterOrder, StateType >](#) 38

[lir::ChebyshevI::BandStop< FilterOrder, StateType >](#) 33

[lir::ChebyshevII::BandStop< FilterOrder, StateType >](#) 36

[lir::BandStopTransform](#) 41

BaseClass

[lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >](#) 79

[lir::Biquad](#) 41

[lir::Custom::OnePole](#) 78

[lir::Custom::TwoPole](#) 83

[lir::RBJ::RBJbase](#) 80

lir::RBJ::AllPass	15
lir::RBJ::BandPass1	22
lir::RBJ::BandPass2	23
lir::RBJ::BandShelf	27
lir::RBJ::BandStop	35
lir::RBJ::HighPass	49
lir::RBJ::HighShelf	54
lir::RBJ::IIRNotch	62
lir::RBJ::LowPass	63
lir::RBJ::LowShelf	76
lir::Cascade	44
lir::PoleFilterBase2	79
lir::PoleFilterBase< AnalogPrototype >	79
lir::PoleFilterBase< AnalogLowPass >	79
lir::Butterworth::BandPassBase	25
lir::Butterworth::BandStopBase	39
lir::Butterworth::HighPassBase	54
lir::Butterworth::LowPassBase	69
lir::ChebyshevI::BandPassBase	24
lir::ChebyshevI::BandStopBase	40
lir::ChebyshevI::HighPassBase	53
lir::ChebyshevI::LowPassBase	69
lir::ChebyshevII::BandPassBase	24
lir::ChebyshevII::BandStopBase	40
lir::ChebyshevII::HighPassBase	53
lir::ChebyshevII::LowPassBase	70
lir::PoleFilterBase< AnalogLowShelf >	79
lir::Butterworth::BandShelfBase	32
lir::Butterworth::HighShelfBase	61
lir::Butterworth::LowShelfBase	77
lir::ChebyshevI::BandShelfBase	33
lir::ChebyshevI::HighShelfBase	62

lir::ChebyshevI::LowShelfBase	78
lir::ChebyshevII::BandShelfBase	32
lir::ChebyshevII::HighShelfBase	61
lir::ChebyshevII::LowShelfBase	77
lir::CascadeStages< MaxStages, StateType >	45
lir::CascadeStages< NSOS, DirectFormII >	45
lir::Custom::SOSCascade< NSOS, StateType >	81
lir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >	45
lir::PoleFilter< HighPassBase, DirectFormII, FilterOrder >	79
lir::Butterworth::HighPass< FilterOrder, StateType >	51
lir::ChebyshevI::HighPass< FilterOrder, StateType >	47
lir::ChebyshevII::HighPass< FilterOrder, StateType >	50
lir::PoleFilter< HighShelfBase, DirectFormII, FilterOrder >	79
lir::Butterworth::HighShelf< FilterOrder, StateType >	57
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	55
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	59
lir::PoleFilter< LowPassBase, DirectFormII, FilterOrder >	79
lir::Butterworth::LowPass< FilterOrder, StateType >	66
lir::ChebyshevI::LowPass< FilterOrder, StateType >	67
lir::ChebyshevII::LowPass< FilterOrder, StateType >	64
lir::PoleFilter< LowShelfBase, DirectFormII, FilterOrder >	79
lir::Butterworth::LowShelf< FilterOrder, StateType >	74
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	71
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	72
lir::CascadeStages<(MaxAnalogPoles+1)/2, StateType >	45
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	79
lir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >	45
lir::PoleFilter< BandPassBase, DirectFormII, FilterOrder, FilterOrder *2 >	79
lir::PoleFilter< BandShelfBase, DirectFormII, FilterOrder, FilterOrder *2 >	79
lir::PoleFilter< BandStopBase, DirectFormII, FilterOrder, FilterOrder *2 >	79
complex_pair_t	
lir::ComplexPair	46

<b>lir::DirectFormI</b>	<b>47</b>
<b>lir::DirectFormII</b>	<b>47</b>
HighPassBase	
<b>lir::PoleFilter&lt; HighPassBase, DirectFormII, FilterOrder &gt;</b>	<b>79</b>
<b>lir::HighPassTransform</b>	<b>54</b>
HighShelfBase	
<b>lir::PoleFilter&lt; HighShelfBase, DirectFormII, FilterOrder &gt;</b>	<b>79</b>
<b>lir::Layout&lt; MaxPoles &gt;</b>	<b>63</b>
<b>lir::Layout&lt; MaxAnalogPoles &gt;</b>	<b>63</b>
<b>lir::Layout&lt; MaxDigitalPoles &gt;</b>	<b>63</b>
<b>lir::LayoutBase</b>	<b>63</b>
<b>lir::Butterworth::AnalogLowPass</b>	<b>15</b>
<b>lir::Butterworth::AnalogLowShelf</b>	<b>16</b>
<b>lir::ChebyshevI::AnalogLowPass</b>	<b>16</b>
<b>lir::ChebyshevI::AnalogLowShelf</b>	<b>17</b>
<b>lir::ChebyshevII::AnalogLowPass</b>	<b>16</b>
<b>lir::ChebyshevII::AnalogLowShelf</b>	<b>16</b>
LowPassBase	
<b>lir::PoleFilter&lt; LowPassBase, DirectFormII, FilterOrder &gt;</b>	<b>79</b>
<b>lir::LowPassTransform</b>	<b>70</b>
LowShelfBase	
<b>lir::PoleFilter&lt; LowShelfBase, DirectFormII, FilterOrder &gt;</b>	<b>79</b>
<b>lir::PoleZeroPair</b>	<b>80</b>
<b>lir::BiquadPoleState</b>	<b>44</b>
<b>lir::Cascade::Storage</b>	<b>83</b>
<b>lir::TransposedDirectFormII</b>	<b>83</b>

## 4 Class Index

### 4.1 Class List

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

<b>lir::RBJ::AllPass</b>	<b>15</b>
<b>lir::Butterworth::AnalogLowPass</b>	<b>15</b>
<b>lir::ChebyshevI::AnalogLowPass</b>	<b>16</b>

<a href="#">lir::ChebyshevII::AnalogLowPass</a>	16
<a href="#">lir::ChebyshevII::AnalogLowShelf</a>	16
<a href="#">lir::Butterworth::AnalogLowShelf</a>	16
<a href="#">lir::ChebyshevI::AnalogLowShelf</a>	17
<a href="#">lir::ChebyshevI::BandPass&lt; FilterOrder, StateType &gt;</a>	17
<a href="#">lir::Butterworth::BandPass&lt; FilterOrder, StateType &gt;</a>	19
<a href="#">lir::ChebyshevII::BandPass&lt; FilterOrder, StateType &gt;</a>	20
<a href="#">lir::RBJ::BandPass1</a>	22
<a href="#">lir::RBJ::BandPass2</a>	23
<a href="#">lir::ChebyshevII::BandPassBase</a>	24
<a href="#">lir::ChebyshevI::BandPassBase</a>	24
<a href="#">lir::Butterworth::BandPassBase</a>	25
<a href="#">lir::BandPassTransform</a>	25
<a href="#">lir::Butterworth::BandShelf&lt; FilterOrder, StateType &gt;</a>	25
<a href="#">lir::RBJ::BandShelf</a>	27
<a href="#">lir::ChebyshevI::BandShelf&lt; FilterOrder, StateType &gt;</a>	28
<a href="#">lir::ChebyshevII::BandShelf&lt; FilterOrder, StateType &gt;</a>	30
<a href="#">lir::Butterworth::BandShelfBase</a>	32
<a href="#">lir::ChebyshevII::BandShelfBase</a>	32
<a href="#">lir::ChebyshevI::BandShelfBase</a>	33
<a href="#">lir::ChebyshevI::BandStop&lt; FilterOrder, StateType &gt;</a>	33
<a href="#">lir::RBJ::BandStop</a>	35
<a href="#">lir::ChebyshevII::BandStop&lt; FilterOrder, StateType &gt;</a>	36
<a href="#">lir::Butterworth::BandStop&lt; FilterOrder, StateType &gt;</a>	38
<a href="#">lir::Butterworth::BandStopBase</a>	39
<a href="#">lir::ChebyshevII::BandStopBase</a>	40
<a href="#">lir::ChebyshevI::BandStopBase</a>	40
<a href="#">lir::BandStopTransform</a>	41
<a href="#">lir::Biquad</a>	41
<a href="#">lir::BiquadPoleState</a>	44
<a href="#">lir::Cascade</a>	44
<a href="#">lir::CascadeStages&lt; MaxStages, StateType &gt;</a>	45

<a href="#">lir::ComplexPair</a>	<a href="#">46</a>
<a href="#">lir::DirectFormI</a>	<a href="#">47</a>
<a href="#">lir::DirectFormII</a>	<a href="#">47</a>
<a href="#">lir::ChebyshevI::HighPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">47</a>
<a href="#">lir::RBJ::HighPass</a>	<a href="#">49</a>
<a href="#">lir::ChebyshevII::HighPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">50</a>
<a href="#">lir::Butterworth::HighPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">51</a>
<a href="#">lir::ChebyshevI::HighPassBase</a>	<a href="#">53</a>
<a href="#">lir::ChebyshevII::HighPassBase</a>	<a href="#">53</a>
<a href="#">lir::Butterworth::HighPassBase</a>	<a href="#">54</a>
<a href="#">lir::HighPassTransform</a>	<a href="#">54</a>
<a href="#">lir::RBJ::HighShelf</a>	<a href="#">54</a>
<a href="#">lir::ChebyshevI::HighShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">55</a>
<a href="#">lir::Butterworth::HighShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">57</a>
<a href="#">lir::ChebyshevII::HighShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">59</a>
<a href="#">lir::ChebyshevII::HighShelfBase</a>	<a href="#">61</a>
<a href="#">lir::Butterworth::HighShelfBase</a>	<a href="#">61</a>
<a href="#">lir::ChebyshevI::HighShelfBase</a>	<a href="#">62</a>
<a href="#">lir::RBJ::IIRNotch</a>	<a href="#">62</a>
<a href="#">lir::Layout&lt; MaxPoles &gt;</a>	<a href="#">63</a>
<a href="#">lir::LayoutBase</a>	<a href="#">63</a>
<a href="#">lir::RBJ::LowPass</a>	<a href="#">63</a>
<a href="#">lir::ChebyshevII::LowPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">64</a>
<a href="#">lir::Butterworth::LowPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">66</a>
<a href="#">lir::ChebyshevI::LowPass&lt; FilterOrder, StateType &gt;</a>	<a href="#">67</a>
<a href="#">lir::ChebyshevI::LowPassBase</a>	<a href="#">69</a>
<a href="#">lir::Butterworth::LowPassBase</a>	<a href="#">69</a>
<a href="#">lir::ChebyshevII::LowPassBase</a>	<a href="#">70</a>
<a href="#">lir::LowPassTransform</a>	<a href="#">70</a>
<a href="#">lir::ChebyshevI::LowShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">71</a>
<a href="#">lir::ChebyshevII::LowShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">72</a>
<a href="#">lir::Butterworth::LowShelf&lt; FilterOrder, StateType &gt;</a>	<a href="#">74</a>



<a href="#">lir::RBJ::LowShelf</a>	76
<a href="#">lir::Butterworth::LowShelfBase</a>	77
<a href="#">lir::ChebyshevII::LowShelfBase</a>	77
<a href="#">lir::ChebyshevI::LowShelfBase</a>	78
<a href="#">lir::Custom::OnePole</a>	78
<a href="#">lir::PoleFilter&lt; BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles &gt;</a>	79
<a href="#">lir::PoleFilterBase&lt; AnalogPrototype &gt;</a>	79
<a href="#">lir::PoleFilterBase2</a>	79
<a href="#">lir::PoleZeroPair</a>	80
<a href="#">lir::RBJ::RBJbase</a>	80
<a href="#">lir::Custom::SOSCascade&lt; NSOS, StateType &gt;</a>	81
<a href="#">lir::Cascade::Storage</a>	83
<a href="#">lir::TransposedDirectFormII</a>	83
<a href="#">lir::Custom::TwoPole</a>	83

## 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/iirl1>

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/iirl1>

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/iirl1>

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](#)

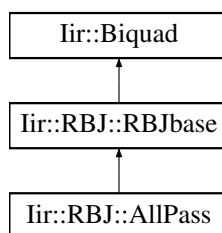
### 5.5.1 Detailed Description

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

Inheritance diagram for Iir::RBJ::AllPass:



### Additional Inherited Members

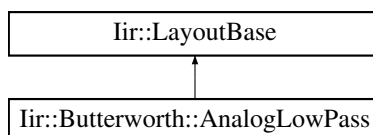
The documentation for this struct was generated from the following files:

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

### 6.2 Iir::Butterworth::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::Butterworth::AnalogLowPass:



### 6.2.1 Detailed Description

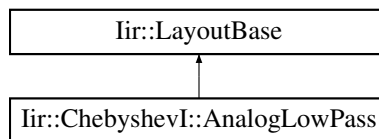
Half-band analog prototypes (s-plane)

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

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

### 6.3 Iir::ChebyshevI::AnalogLowPass Class Reference

Inheritance diagram for Iir::ChebyshevI::AnalogLowPass:

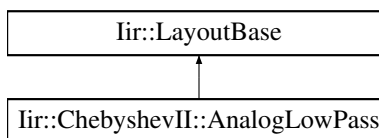


The documentation for this class was generated from the following files:

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

### 6.4 Iir::ChebyshevII::AnalogLowPass Class Reference

Inheritance diagram for Iir::ChebyshevII::AnalogLowPass:

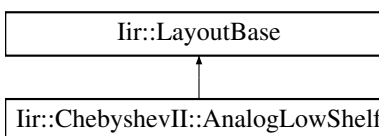


The documentation for this class was generated from the following files:

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

### 6.5 Iir::ChebyshevII::AnalogLowShelf Class Reference

Inheritance diagram for Iir::ChebyshevII::AnalogLowShelf:

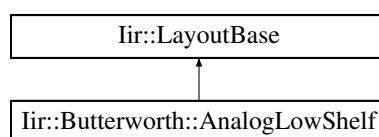


The documentation for this class was generated from the following files:

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

### 6.6 Iir::Butterworth::AnalogLowShelf Class Reference

Inheritance diagram for Iir::Butterworth::AnalogLowShelf:

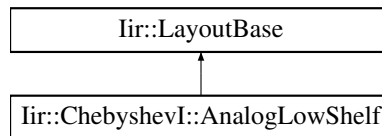


The documentation for this class was generated from the following files:

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

## 6.7 Iir::ChebyshevI::AnalogLowShelf Class Reference

Inheritance diagram for Iir::ChebyshevI::AnalogLowShelf:



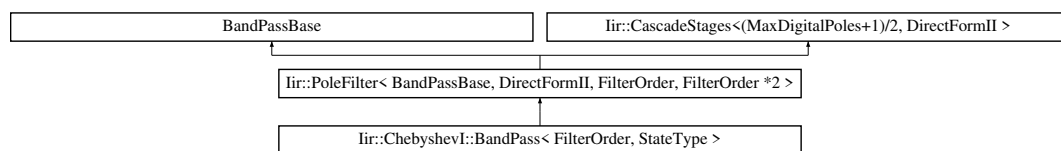
The documentation for this class was generated from the following files:

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

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

#include <ChebyshevI.h>

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



### Public Member Functions

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

### 6.8.1 Detailed Description

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

[ChebyshevI](#) bandpass filter

#### Parameters

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

### 6.8.2 Member Function Documentation

**6.8.2.1 setup() [1/4]** `template<int FilterOrder, class StateType = DirectFormII> void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setup ( 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 width of the passband
<i>rippleDb</i>	Permitted ripples in dB in the passband

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

Calculates the coefficients of the filter at the order FilterOrder

## Parameters

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

**6.8.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::BandPass< FilterOrder, StateType >::setup (`  
`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

**6.8.2.4 setup() [4/4]** `template<int FilterOrder, 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



## Parameters

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

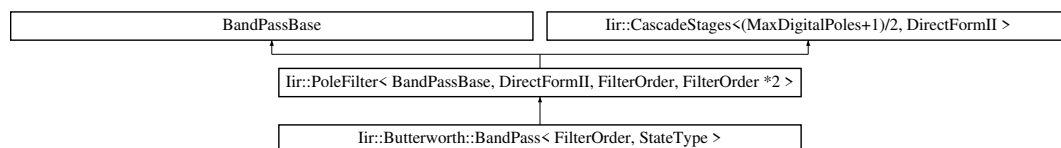
The documentation for this struct was generated from the following file:

- `iir/ChebyshevI.h`

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

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

Inheritance diagram for `lir::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 [setup](#) (double centerFrequency, double widthFrequency)
- void [setup](#) (int reqOrder, double centerFrequency, double widthFrequency)

## 6.9.1 Detailed Description

```
template<int FilterOrder, class StateType = DirectFormII>
struct lir::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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

## 6.9.2 Member Function Documentation

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

```
void lir::Butterworth::BandPass< FilterOrder, StateType >::setup (
    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

**6.9.2.2 setup() [2/4]** `template<int FilterOrder, 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

**6.9.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::BandPass< FilterOrder, StateType >::setup (`  
`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

**6.9.2.4 setup() [4/4]** `template<int FilterOrder, 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

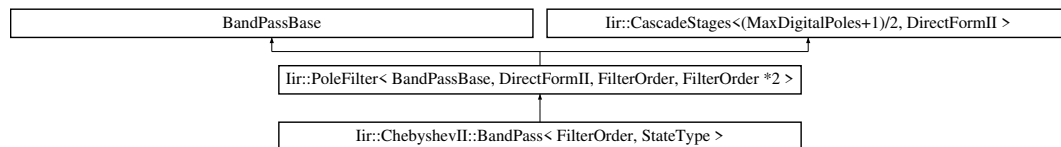
The documentation for this struct was generated from the following file:

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

### 6.10.1 Detailed Description

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

[ChebyshevII](#) bandpass filter

#### Parameters

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

### 6.10.2 Member Function Documentation

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

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

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

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

Calculates the coefficients of the filter

#### Parameters

<i>sampleRate</i>	Sampling rate
-------------------	---------------

## Parameters

<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 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setup (`  
`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

**6.10.2.4 setup() [4/4]** `template<int FilterOrder, 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

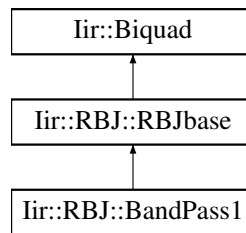
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

## 6.11 Iir::RBJ::BandPass1 Struct Reference

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

Inheritance diagram for Iir::RBJ::BandPass1:



### Public Member Functions

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

#### 6.11.1 Detailed Description

Bandpass with constant skirt gain

#### 6.11.2 Member Function Documentation

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

Calculates the coefficients

##### Parameters

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

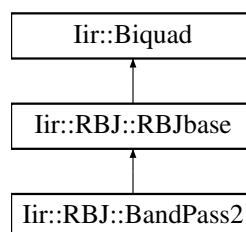
The documentation for this struct was generated from the following files:

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

## 6.12 Iir::RBJ::BandPass2 Struct Reference

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

Inheritance diagram for Iir::RBJ::BandPass2:



### Public Member Functions

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

#### 6.12.1 Detailed Description

Bandpass with constant 0 dB peak gain

## 6.12.2 Member Function Documentation

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

Calculates the coefficients

### Parameters

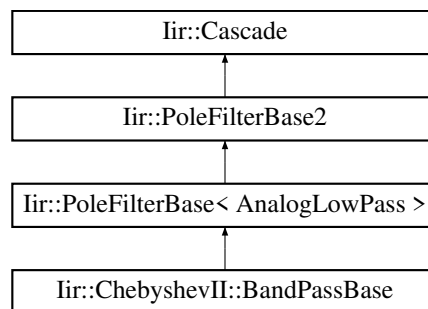
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>bandWidth</i>	Bandwidth in octaves

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

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

## 6.13 Iir::ChebyshevII::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandPassBase:



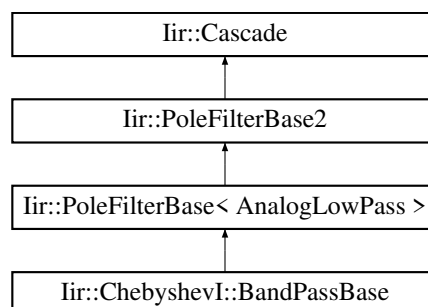
### Additional Inherited Members

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

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

## 6.14 Iir::ChebyshevI::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandPassBase:



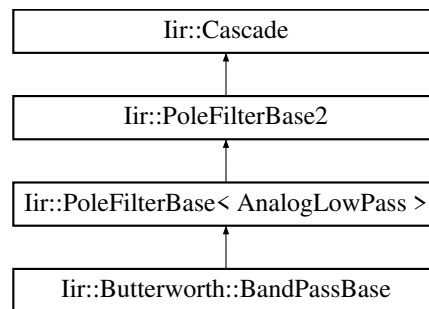
### Additional Inherited Members

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

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

## 6.15 Iir::Butterworth::BandPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandPassBase:



### Additional Inherited Members

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

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

## 6.16 Iir::BandPassTransform Class Reference

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

### 6.16.1 Detailed Description

low pass to band pass transform

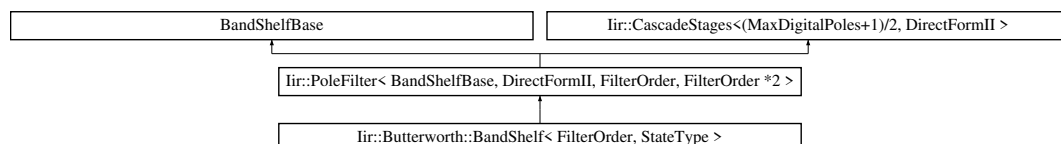
The documentation for this class was generated from the following files:

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

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

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

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



### Public Member Functions

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

### 6.17.1 Detailed Description

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

**Butterworth** Bandshelf filter: it is a bandpass filter which amplifies at a specified gain in dB the frequencies in the passband.

#### Parameters

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

### 6.17.2 Member Function Documentation

**6.17.2.1 setup() [1/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setup (`  
     `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

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

Calculates the coefficients 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

**6.17.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::BandShelf< FilterOrder, StateType >::setup (`  
     `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

```

6.17.2.4 setup() [4/4] template<int FilterOrder, class StateType = DirectFormII>
void lir::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

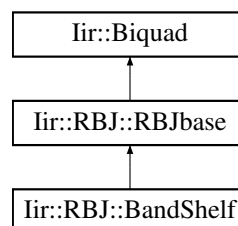
The documentation for this struct was generated from the following file:

- `iir/Butterworth.h`

## 6.18 `lir::RBJ::BandShelf` Struct Reference

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

Inheritance diagram for `lir::RBJ::BandShelf`:



### Public Member Functions

- void `setup` (double sampleRate, double centerFrequency, double gainDb, double bandWidth)

### 6.18.1 Detailed Description

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

### 6.18.2 Member Function Documentation

```

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

```

Calculates the coefficients

#### Parameters

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

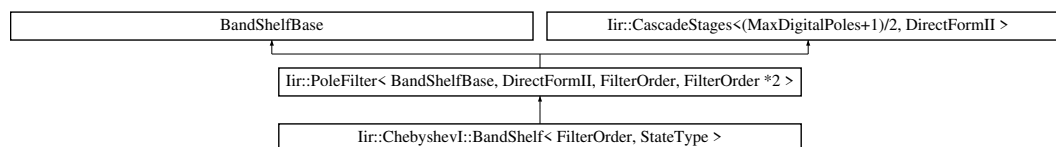
The documentation for this struct was generated from the following files:

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

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

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

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



### Public Member Functions

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

### 6.19.1 Detailed Description

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

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

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

#### Parameters

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

### 6.19.2 Member Function Documentation

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

**6.19.2.2 setup() [2/4]** `template<int FilterOrder, 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.

**6.19.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::BandShelf< FilterOrder, StateType >::setup (`  
`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.

```

6.19.2.4 setup() [4/4] template<int FilterOrder, 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.

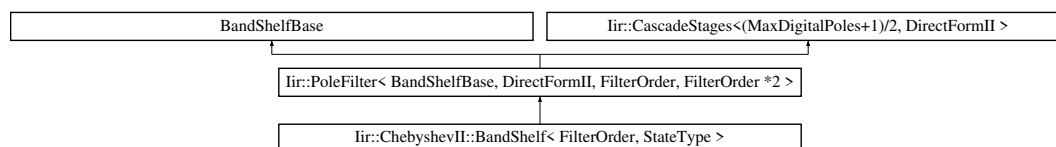
The documentation for this struct was generated from the following file:

- 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 [setup](#) (double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void [setup](#) (int reqOrder, double centerFrequency, double widthFrequency, double gainDb, double stopBand↵Db)

### 6.20.1 Detailed Description

```

template<int FilterOrder, 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

## 6.20.2 Member Function Documentation

**6.20.2.1 `setup()` [1/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void lir::ChebyshevII::BandShelf< FilterOrder, StateType >::setup (`  
`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
<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()` [2/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void lir::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

**6.20.2.3 `setup()` [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void lir::ChebyshevII::BandShelf< FilterOrder, StateType >::setup (`  
`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

```

6.20.2.4 setup() [4/4] template<int FilterOrder, 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

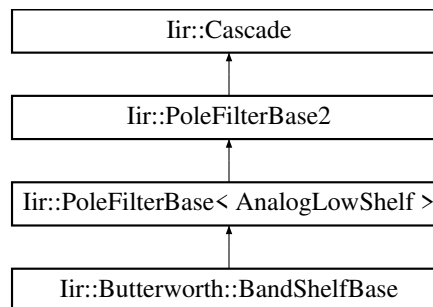
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>gainDb</i>	Gain in the passband. The stopband has always 0dB.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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:



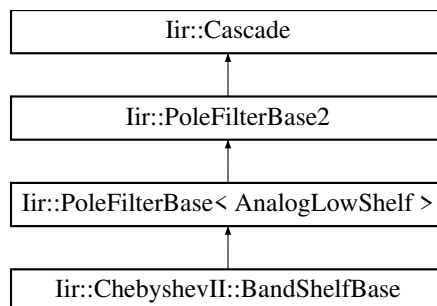
#### Additional Inherited Members

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:



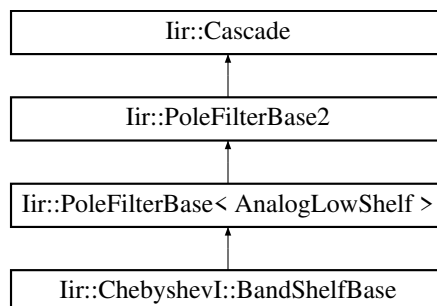
#### Additional Inherited Members

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

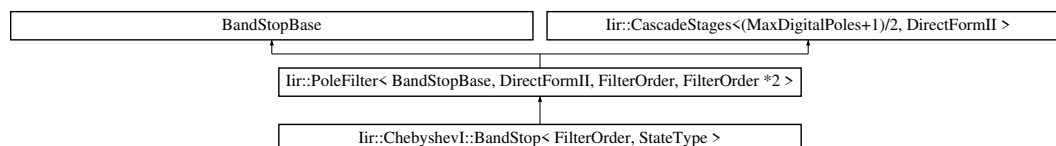
The documentation for this struct was generated from the following files:

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

### 6.24.1 Detailed Description

```
template<int FilterOrder, 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

### 6.24.2 Member Function Documentation

**6.24.2.1 [setup\(\)](#) [1/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setup (`  
`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

**6.24.2.2 [setup\(\)](#) [2/4]** `template<int FilterOrder, 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

**6.24.2.3 [setup\(\)](#) [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::BandStop< FilterOrder, StateType >::setup (`  
`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

**6.24.2.4 setup() [4/4]** `template<int FilterOrder, 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

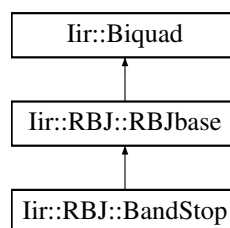
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

**6.25 Iir::RBJ::BandStop Struct Reference**

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

Inheritance diagram for Iir::RBJ::BandStop:

**Public Member Functions**

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

**6.25.1 Detailed Description**

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

**6.25.2 Member Function Documentation**

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

Calculates the coefficients

#### Parameters

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

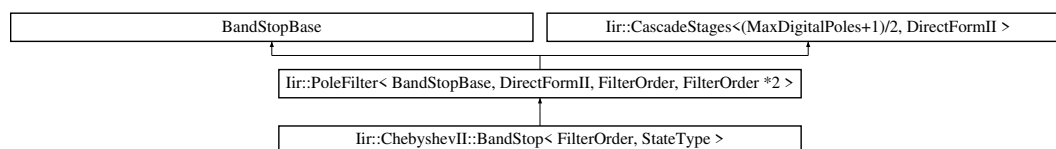
The documentation for this struct was generated from the following files:

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

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

`#include <ChebyshevII.h>`

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



### Public Member Functions

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

### 6.26.1 Detailed Description

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

[ChebyshevII](#) bandstop filter.

#### Parameters

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

### 6.26.2 Member Function Documentation

**6.26.2.1 setup()** [1/4] `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevII::BandStop< FilterOrder, StateType >::setup (`  
`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
<i>widthFrequency</i>	Width of the bandstop
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

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

Calculates the coefficients of the filter

#### Parameters

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

#### 6.26.2.3 setup() [3/4] `template<int FilterOrder, class StateType = DirectFormII>`

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

#### 6.26.2.4 setup() [4/4] `template<int FilterOrder, 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

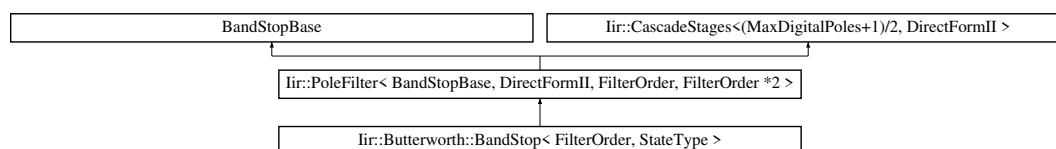
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

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

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

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



### Public Member Functions

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

#### 6.27.1 Detailed Description

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

[Butterworth](#) Bandstop filter.

## Parameters

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

#### 6.27.2 Member Function Documentation

**6.27.2.1 setup() [1/4]** `template<int FilterOrder, class StateType = DirectFormII> void Iir::Butterworth::BandStop< FilterOrder, StateType >::setup ( 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

**6.27.2.2 setup()** [2/4] `template<int FilterOrder, 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

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

Calculates the coefficients

#### Parameters

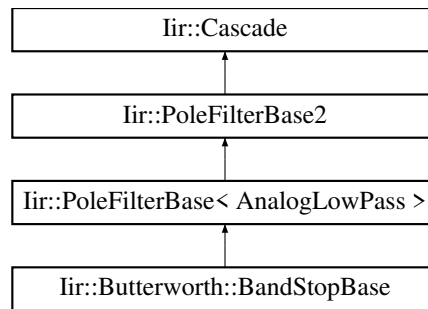
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop

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:



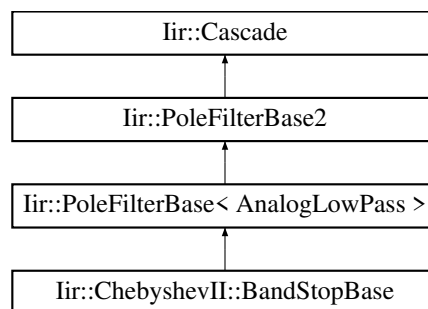
#### Additional Inherited Members

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`:



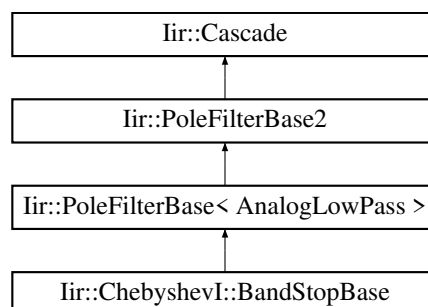
#### Additional Inherited Members

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/Chebyshev1.h
- iir/Chebyshev1.cpp

## 6.31 Iir::BandStopTransform Class Reference

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

### 6.31.1 Detailed Description

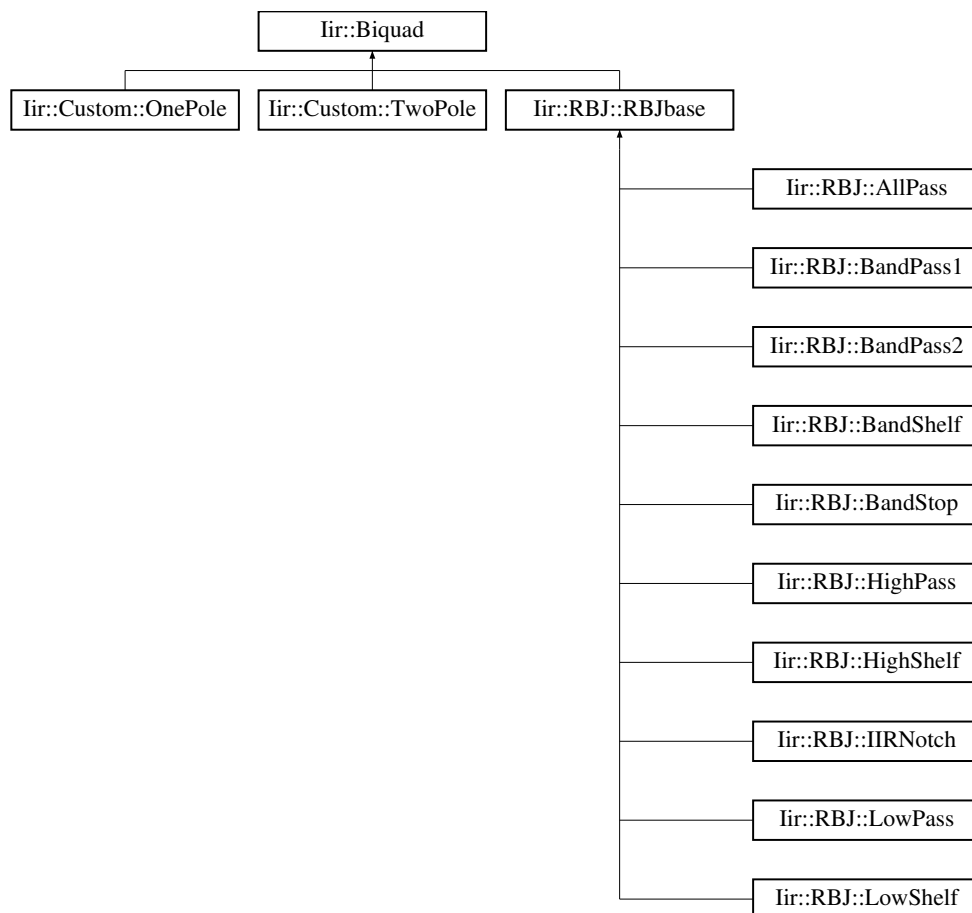
low pass to band stop transform

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 <a href="#">State.h</a> )

#### 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/zoes as a pair. Needs to be complex conjugate.

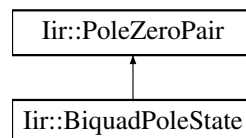
The documentation for this class was generated from the following files:

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

### 6.33 Iir::BiquadPoleState Struct Reference

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

Inheritance diagram for Iir::BiquadPoleState:



#### 6.33.1 Detailed Description

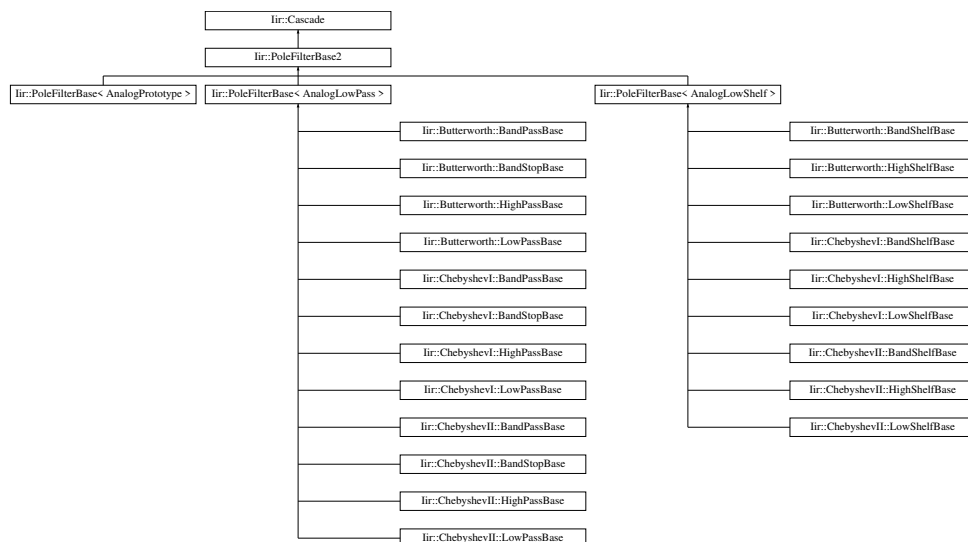
Expresses a biquad as a pair of pole/zeros, with gain values so that the coefficients can be reconstructed precisely. The documentation for this struct was generated from the following files:

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

### 6.34 Iir::Cascade Class Reference

```
#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)

### 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 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)

**6.35.2.3 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.
------------------------	--

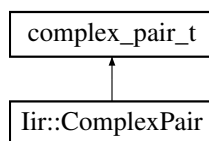
The documentation for this class was generated from the following file:

- 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:

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

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

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:

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

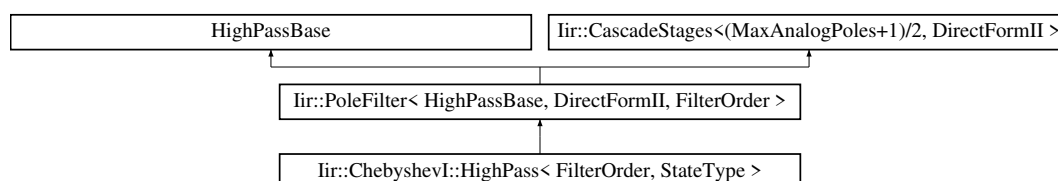
The documentation for this class was generated from the following file:

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

### 6.39.1 Detailed Description

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

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

[ChebyshevI](#) highpass filter

## Parameters

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

## 6.39.2 Member Function Documentation

**6.39.2.1 setup()** [1/4] `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setup (  
    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

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

Calculates the coefficients of the filter 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

**6.39.2.3 setup()** [3/4] `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::ChebyshevI::HighPass< FilterOrder, StateType >::setup (  
    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

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

```

    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

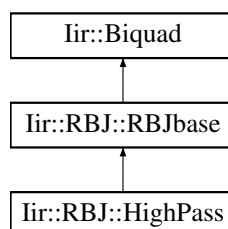
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

## 6.40 Iir::RBJ::HighPass Struct Reference

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

Inheritance diagram for Iir::RBJ::HighPass:



#### Public Member Functions

- void [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)) )

Calculates the coefficients

#### Parameters

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

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

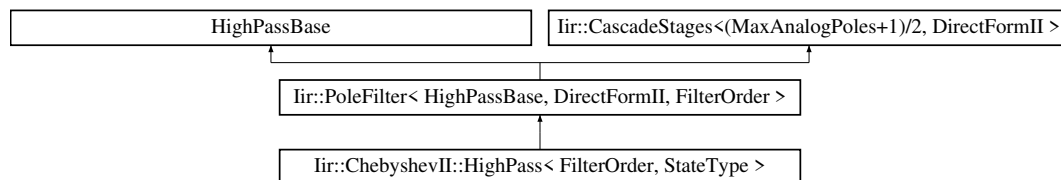
- iir/RBJ.h

- iir/RBJ.cpp

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

#### 6.41.1 Detailed Description

```
template<int FilterOrder, 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

#### 6.41.2 Member Function Documentation

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

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

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

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

Calculates the coefficients of the filter



## Parameters

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

**6.41.2.3 setup()** [3/4] `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevII::HighPass< FilterOrder, StateType >::setup (`  
`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

**6.41.2.4 setup()** [4/4] `template<int FilterOrder, 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

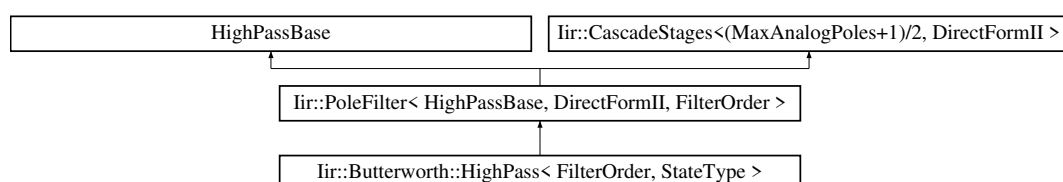
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

## 6.42 Iir::Butterworth::HighPass&lt; FilterOrder, StateType &gt; 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 [setup](#) (double cutoffFrequency)
- void [setup](#) (int reqOrder, double cutoffFrequency)

### 6.42.1 Detailed Description

```
template<int FilterOrder, 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

### 6.42.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

#### Parameters

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

**6.42.2.2 [setup\(\)](#) [2/4]** `template<int FilterOrder, 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

**6.42.2.3 [setup\(\)](#) [3/4]** `template<int FilterOrder, class StateType = DirectFormII> void Iir::Butterworth::HighPass< FilterOrder, StateType >::setup ( int reqOrder, double cutoffFrequency ) [inline]`

Calculates the coefficients

#### Parameters

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

## Parameters

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

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

Calculates the coefficients

## Parameters

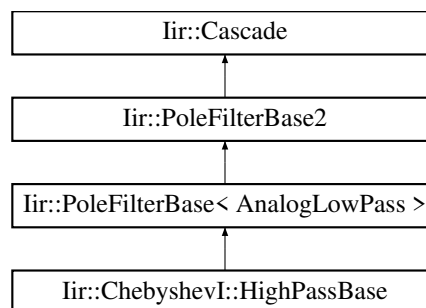
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff

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

- iir/Butterworth.h

## 6.43 Iir::ChebyshevI::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighPassBase:



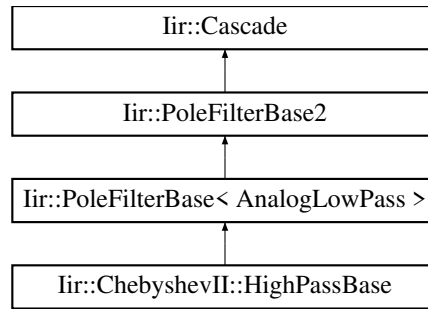
## Additional Inherited Members

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

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

## 6.44 Iir::ChebyshevII::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighPassBase:



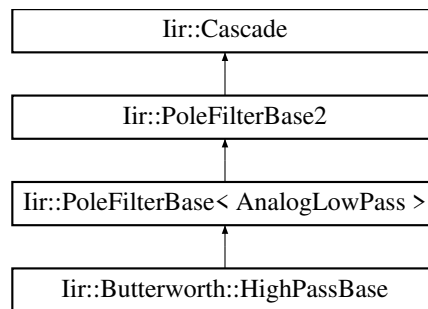
#### Additional Inherited Members

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

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

### 6.45 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

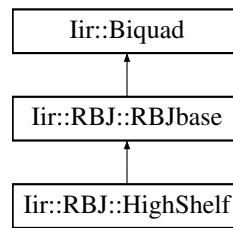
The documentation for this class was generated from the following files:

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

### 6.47 Iir::RBJ::HighShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::HighShelf:



## Public Member Functions

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

### 6.47.1 Detailed Description

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

### 6.47.2 Member Function Documentation

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

Calculates the coefficients

#### Parameters

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

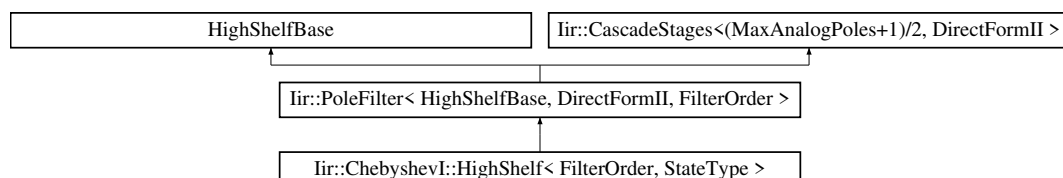
The documentation for this struct was generated from the following files:

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

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

### 6.48.1 Detailed Description

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

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

#### Parameters

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

### 6.48.2 Member Function Documentation

**6.48.2.1 [setup\(\)](#) [1/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setup (`  
`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

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

Calculates the coefficients of the filter 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

**6.48.2.3 [setup\(\)](#) [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setup (`  
`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

**6.48.2.4 setup() [4/4]** `template<int FilterOrder, 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

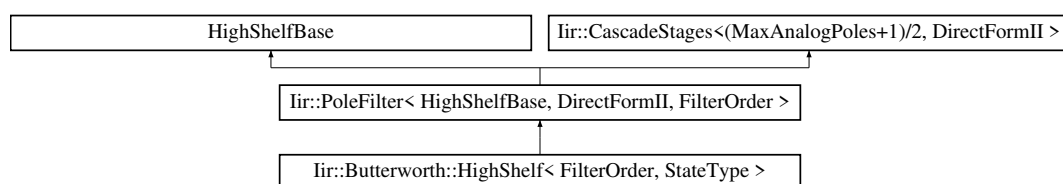
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

**6.49 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 [setup](#) (double cutoffFrequency, double gainDb)
- void [setup](#) (int reqOrder, double cutoffFrequency, double gainDb)

**6.49.1 Detailed Description**

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

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

## Parameters

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

## 6.49.2 Member Function Documentation

**6.49.2.1 setup() [1/4]** `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setup (  
    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

**6.49.2.2 setup() [2/4]** `template<int FilterOrder, 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

**6.49.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setup (  
    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

**6.49.2.4 setup() [4/4]** `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::Butterworth::HighShelf< FilterOrder, StateType >::setup (  
    double sampleRate,  
    double cutoffFrequency,  
    double gainDb ) [inline]`



```

    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

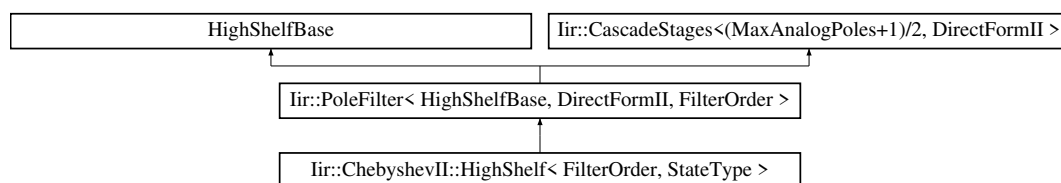
The documentation for this struct was generated from the following file:

- `iir/Butterworth.h`

## 6.50 `lir::ChebyshevII::HighShelf< FilterOrder, StateType >` Struct Template Reference

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

Inheritance diagram for `lir::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 [setup](#) (double cutoffFrequency, double gainDb, double stopBandDb)
- void [setup](#) (int reqOrder, double cutoffFrequency, double gainDb, double stopBandDb)

#### 6.50.1 Detailed Description

```

template<int FilterOrder, class StateType = DirectFormII>
struct lir::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 <code>FilterOrder</code>
<i>StateType</i>	The filter topology: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

#### 6.50.2 Member Function Documentation

```

6.50.2.1 setup\(\) [1/4] template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevII::HighShelf< FilterOrder, StateType >::setup (
    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

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

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

Calculates the coefficients of the filter

#### Parameters

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

#### 6.50.2.3 **setup()** [3/4] `template<int FilterOrder, class StateType = DirectFormII>`

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

#### 6.50.2.4 **setup()** [4/4] `template<int FilterOrder, 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

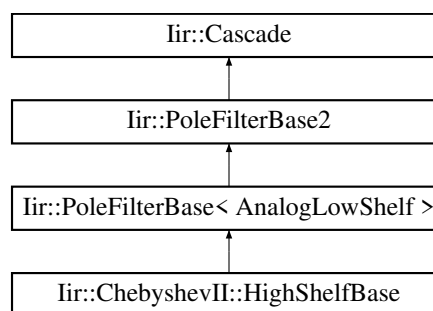
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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:



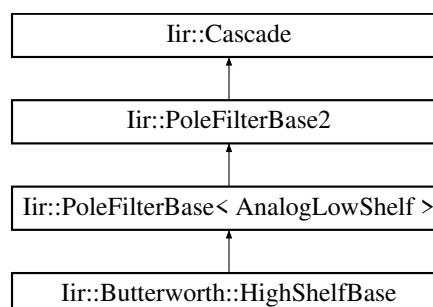
### Additional Inherited Members

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:



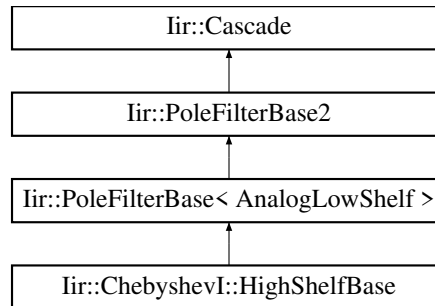
### Additional Inherited Members

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

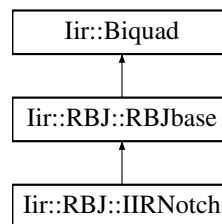
The documentation for this struct was generated from the following files:

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

### 6.54 Iir::RBJ::IIRNotch Struct Reference

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

Inheritance diagram for Iir::RBJ::IIRNotch:



#### Public Member Functions

- void [setup](#) (double sampleRate, double centerFrequency, double q\_factor=10)

#### 6.54.1 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 )

Calculates the coefficients

## Parameters

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

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

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

## 6.55 Iir::Layout< MaxPoles > Class Template Reference

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

### 6.55.1 Detailed Description

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

Storage for [Layout](#)

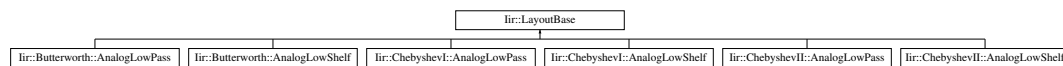
The documentation for this class was generated from the following file:

- 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

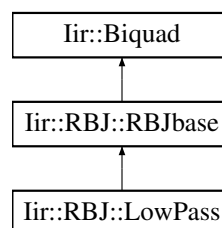
The documentation for this class was generated from the following file:

- iir/Layout.h

## 6.57 Iir::RBJ::LowPass Struct Reference

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

Inheritance diagram for Iir::RBJ::LowPass:



### Public Member Functions

- void [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)) )`

Calculates the coefficients

#### Parameters

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

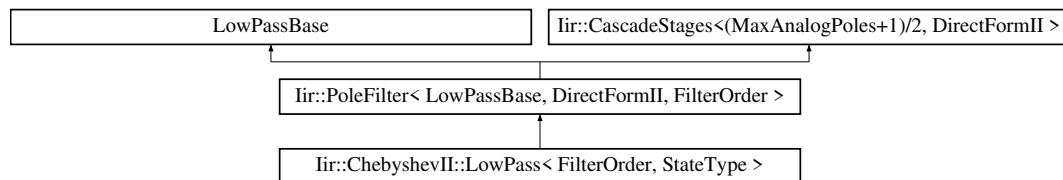
The documentation for this struct was generated from the following files:

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

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

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

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



### Public Member Functions

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

### 6.58.1 Detailed Description

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

[ChebyshevII](#) lowpass filter

#### Parameters

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

## 6.58.2 Member Function Documentation

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

**6.58.2.2 setup() [2/4]** `template<int FilterOrder, 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.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

**6.58.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>  
void Iir::ChebyshevII::LowPass< FilterOrder, StateType >::setup (  
    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

**6.58.2.4 setup() [4/4]** `template<int FilterOrder, 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

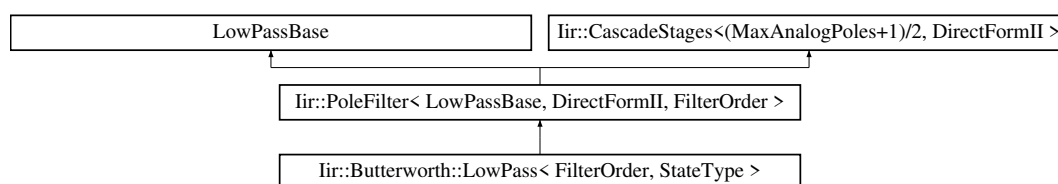
The documentation for this struct was generated from the following file:

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

#### 6.59.1 Detailed Description

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

```
struct Iir::Butterworth::LowPass< FilterOrder, StateType >
```

[Butterworth](#) Lowpass filter.

## Parameters

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

#### 6.59.2 Member Function Documentation

##### 6.59.2.1 [setup\(\)](#) [1/4]

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

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

Calculates the coefficients

## Parameters

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



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

Calculates the coefficients

Parameters

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

**6.59.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::LowPass< FilterOrder, StateType >::setup (`  
`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)

**6.59.2.4 setup() [4/4]** `template<int FilterOrder, 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
<i>cutoffFrequency</i>	Cutoff

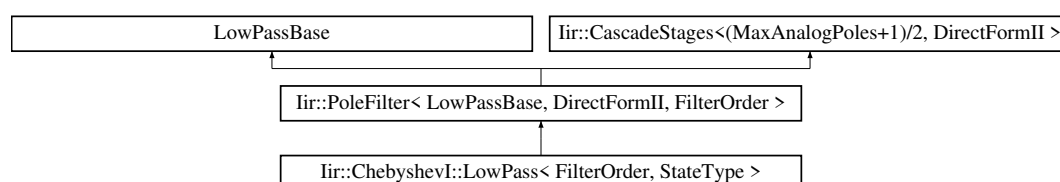
The documentation for this struct was generated from the following file:

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

### 6.60.1 Detailed Description

```
template<int FilterOrder, 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

### 6.60.2 Member Function Documentation

**6.60.2.1 [setup\(\)](#) [1/4]** `template<int FilterOrder, class StateType = DirectFormII> void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup ( 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

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

Calculates the coefficients of the filter 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

**6.60.2.3 [setup\(\)](#) [3/4]** `template<int FilterOrder, class StateType = DirectFormII> void Iir::ChebyshevI::LowPass< FilterOrder, StateType >::setup ( 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

**6.60.2.4 setup()** [4/4] `template<int FilterOrder, 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

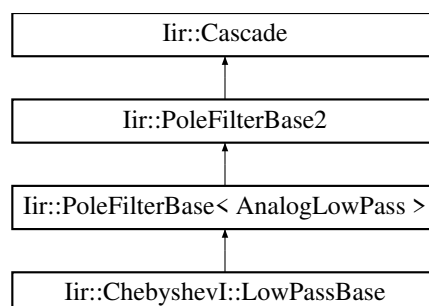
<i>reqOrder</i>	Actual order for the filter calculations
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency.
<i>rippleDb</i>	Permitted ripples in dB in the passband

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:



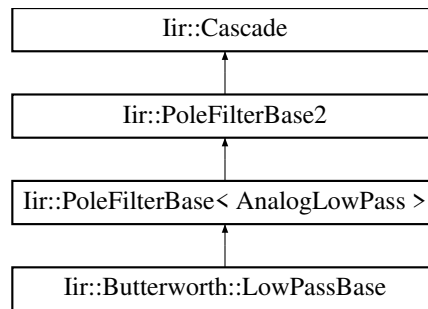
#### Additional Inherited Members

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:



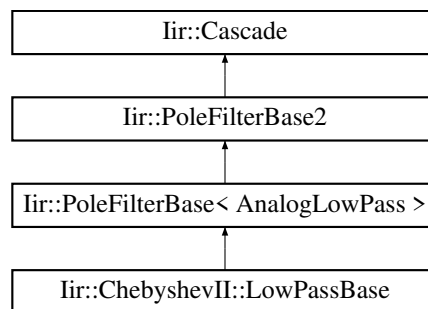
#### Additional Inherited Members

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

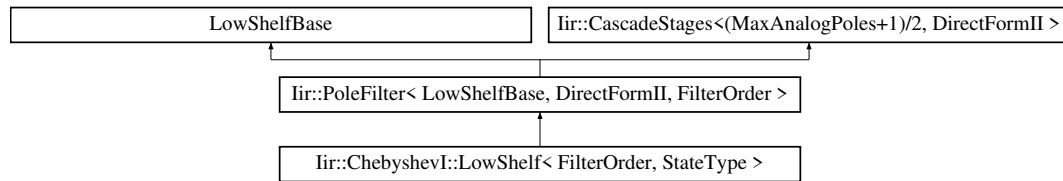
The documentation for this class was generated from the following files:

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

## 6.65 `lir::ChebyshevI::LowShelf< FilterOrder, StateType >` Struct Template Reference

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

Inheritance diagram for `lir::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 [setup](#) (double cutoffFrequency, double gainDb, double rippleDb)
- void [setup](#) (int reqOrder, double cutoffFrequency, double gainDb, double rippleDb)

#### 6.65.1 Detailed Description

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

```
struct lir::ChebyshevI::LowShelf< FilterOrder, StateType >
```

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

#### Parameters

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

### 6.65.2 Member Function Documentation

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

```
void lir::ChebyshevI::LowShelf< FilterOrder, StateType >::setup (
    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

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

```
void lir::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

```
6.65.2.3 setup() [3/4]  template<int FilterOrder, class StateType = DirectFormII>
void Iir::ChebyshevI::LowShelf< FilterOrder, StateType >::setup (
    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

```
6.65.2.4 setup() [4/4]  template<int FilterOrder, 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

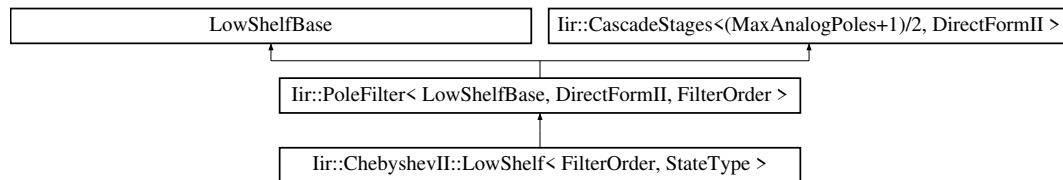
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

### 6.66.1 Detailed Description

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

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

#### Parameters

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

### 6.66.2 Member Function Documentation

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

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

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

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

Calculates the coefficients of the filter

#### Parameters

<i>sampleRate</i>	Sampling rate
-------------------	---------------

## Parameters

<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

**6.66.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::ChebyshevII::LowShelf< FilterOrder, StateType >::setup (`  
`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

**6.66.2.4 setup() [4/4]** `template<int FilterOrder, 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

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

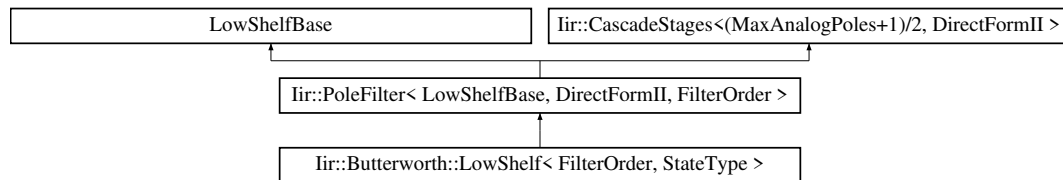
- `iir/ChebyshevII.h`

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

### 6.67.1 Detailed Description

```
template<int FilterOrder, 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 FilterOrder
<i>StateType</i>	The filter topology: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

### 6.67.2 Member Function Documentation

**6.67.2.1 [setup\(\)](#) [1/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setup (`  
     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

**6.67.2.2 [setup\(\)](#) [2/4]** `template<int FilterOrder, 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

**6.67.2.3 setup() [3/4]** `template<int FilterOrder, class StateType = DirectFormII>`  
`void Iir::Butterworth::LowShelf< FilterOrder, StateType >::setup (`  
`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

**6.67.2.4 setup() [4/4]** `template<int FilterOrder, 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

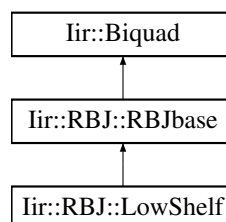
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

## 6.68 Iir::RBJ::LowShelf Struct Reference

`#include <RBJ.h>`

Inheritance diagram for Iir::RBJ::LowShelf:



### Public Member Functions

- void `setup` (double sampleRate, double cutoffFrequency, double gainDb, double shelfSlope=1)

### 6.68.1 Detailed Description

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

## 6.68.2 Member Function Documentation

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

Calculates the coefficients

### Parameters

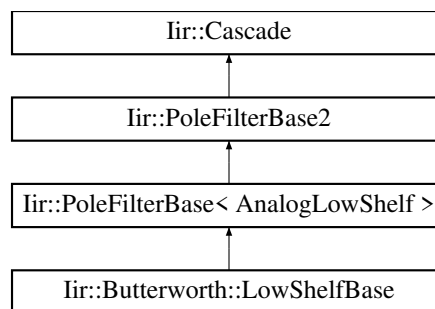
<i>sampleRate</i>	Sampling rate
<i>cutoffFrequency</i>	Cutoff frequency
<i>gainDb</i>	Gain in the passband
<i>shelfSlope</i>	Slope between stop/passband. 1 = as steep as it can.

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

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

## 6.69 Iir::Butterworth::LowShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowShelfBase:



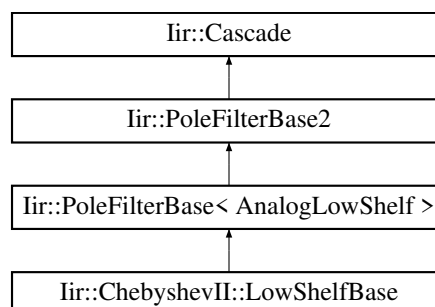
### Additional Inherited Members

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

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

## 6.70 Iir::ChebyshevII::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowShelfBase:



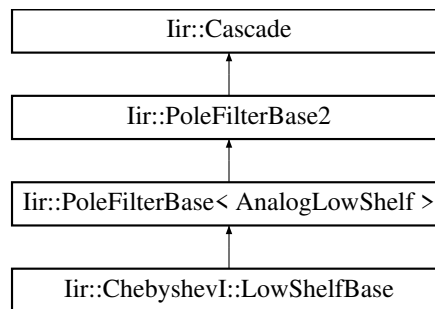
### Additional Inherited Members

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

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

## 6.71 Iir::ChebyshevI::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowShelfBase:



### Additional Inherited Members

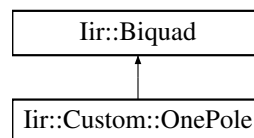
The documentation for this struct was generated from the following files:

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

## 6.72 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

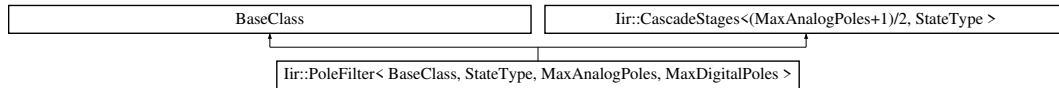
The documentation for this struct was generated from the following files:

- 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

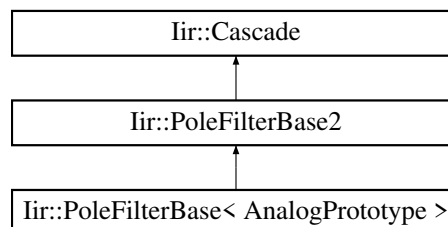
The documentation for this struct was generated from the following file:

- 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.

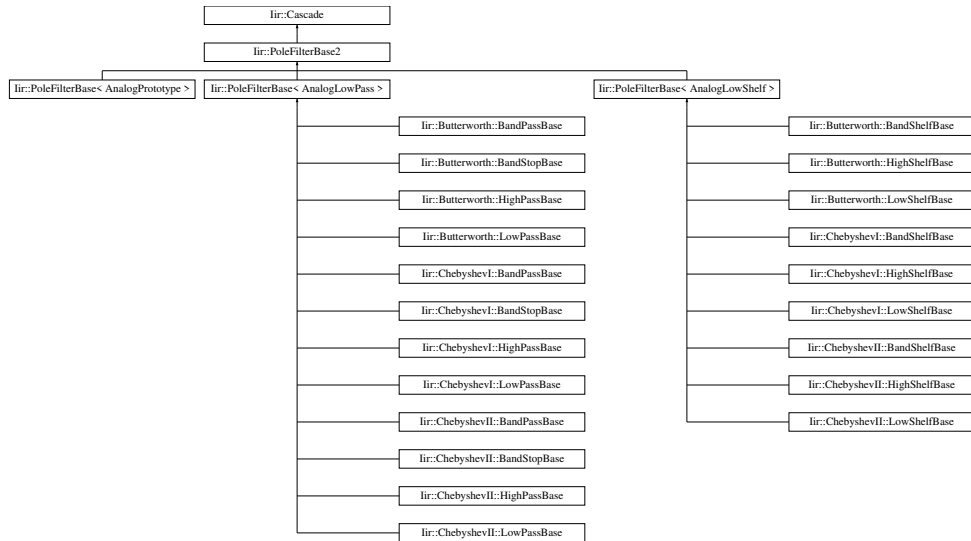
The documentation for this class was generated from the following file:

- 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

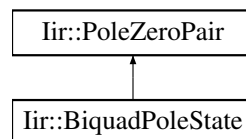
The documentation for this class was generated from the following file:

- 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)

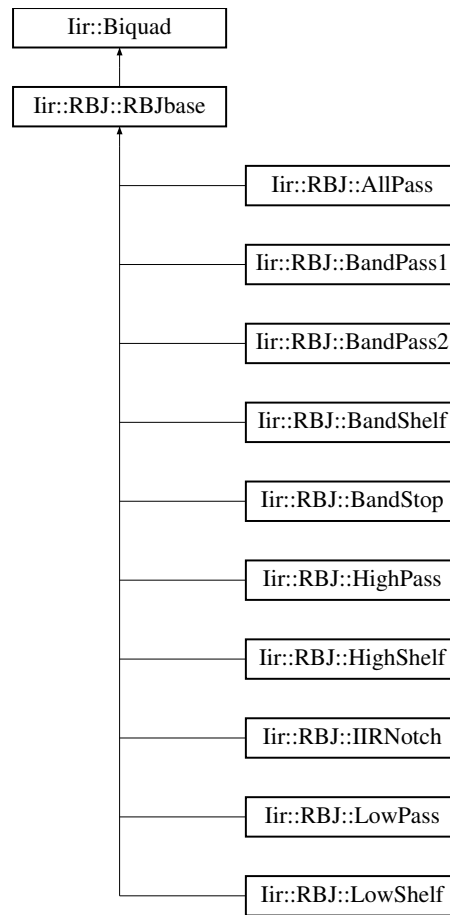
The documentation for this struct was generated from the following file:

- 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

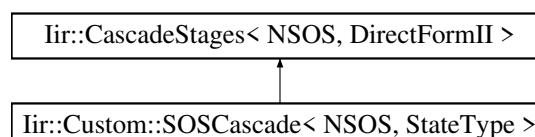
The documentation for this struct was generated from the following file:

- 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: <a href="#">DirectFormI</a> , <a href="#">DirectFormII</a> , ...

### 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](#) \*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](#) \* stageArray\_ ) [inline]

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

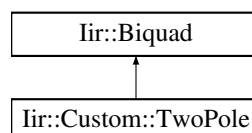
The documentation for this class was generated from the following file:

- 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, [42](#)

filter  
    lir::Biquad, [42](#)  
    lir::CascadeStages< MaxStages, StateType >, [45](#)

getA0  
    lir::Biquad, [42](#)

getA1  
    lir::Biquad, [42](#)

getA2  
    lir::Biquad, [42](#)

getB0  
    lir::Biquad, [42](#)

getB1  
    lir::Biquad, [42](#)

getB2  
    lir::Biquad, [43](#)

getNumStages  
    lir::Cascade, [45](#)

getPoleZeros  
    lir::Biquad, [43](#)  
    lir::Cascade, [45](#)

lir, [11](#)  
    Kind, [13](#)

lir::BandPassTransform, [25](#)

lir::BandStopTransform, [41](#)

lir::Biquad, [41](#)  
    applyScale, [42](#)  
    filter, [42](#)  
    getA0, [42](#)  
    getA1, [42](#)  
    getA2, [42](#)  
    getB0, [42](#)  
    getB1, [42](#)  
    getB2, [43](#)  
    getPoleZeros, [43](#)  
    response, [43](#)  
    setCoefficients, [43](#)  
    setIdentity, [43](#)  
    setOnePole, [43](#)  
    setPoleZeroPair, [43](#)  
    setTwoPole, [43](#)

lir::BiquadPoleState, [44](#)

lir::Butterworth, [13](#)

lir::Butterworth::AnalogLowPass, [15](#)

lir::Butterworth::AnalogLowShelf, [16](#)

lir::Butterworth::BandPass< FilterOrder, StateType >, [19](#)  
    setup, [19](#), [20](#)

lir::Butterworth::BandPassBase, [25](#)

lir::Butterworth::BandShelf< FilterOrder, StateType >, [25](#)  
    setup, [26](#), [27](#)

lir::Butterworth::BandShelfBase, [32](#)

lir::Butterworth::BandStop< FilterOrder, StateType >, [38](#)  
    setup, [38](#), [39](#)

lir::Butterworth::BandStopBase, [39](#)

lir::Butterworth::HighPass< FilterOrder, StateType >, [51](#)  
    setup, [52](#), [53](#)

lir::Butterworth::HighPassBase, [54](#)

lir::Butterworth::HighShelf< FilterOrder, StateType >, [57](#)  
    setup, [58](#)

lir::Butterworth::HighShelfBase, [61](#)

lir::Butterworth::LowPass< FilterOrder, StateType >, [66](#)  
    setup, [66](#), [67](#)

lir::Butterworth::LowPassBase, [69](#)

lir::Butterworth::LowShelf< FilterOrder, StateType >, [74](#)  
    setup, [75](#), [76](#)

lir::Butterworth::LowShelfBase, [77](#)

lir::Cascade, [44](#)  
    getNumStages, [45](#)  
    getPoleZeros, [45](#)  
    operator[], [45](#)  
    response, [45](#)

lir::Cascade::Storage, [83](#)  
    Storage, [83](#)

lir::CascadeStages< MaxStages, StateType >, [45](#)  
    filter, [45](#)  
    reset, [46](#)  
    setup, [46](#)

lir::ChebyshevI, [14](#)

lir::ChebyshevI::AnalogLowPass, [16](#)

lir::ChebyshevI::AnalogLowShelf, [17](#)

lir::ChebyshevI::BandPass< FilterOrder, StateType >, [17](#)  
    setup, [17](#), [18](#)

lir::ChebyshevI::BandPassBase, [24](#)

lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [28](#)  
    setup, [28](#), [29](#)

lir::ChebyshevI::BandShelfBase, [33](#)

lir::ChebyshevI::BandStop< FilterOrder, StateType >, [33](#)  
    setup, [34](#), [35](#)

lir::ChebyshevI::BandStopBase, [40](#)

lir::ChebyshevI::HighPass< FilterOrder, StateType >, [47](#)  
    setup, [48](#)

lir::ChebyshevI::HighPassBase, [53](#)

lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [55](#)  
    setup, [56](#), [57](#)

lir::ChebyshevI::HighShelfBase, [62](#)

lir::ChebyshevI::LowPass< FilterOrder, StateType >, [67](#)  
    setup, [68](#), [69](#)

- lir::ChebyshevI::LowPassBase, 69
- lir::ChebyshevI::LowShelf< FilterOrder, StateType >, 71
  - setup, 71, 72
- lir::ChebyshevI::LowShelfBase, 78
- lir::ChebyshevII, 14
- lir::ChebyshevII::AnalogLowPass, 16
- lir::ChebyshevII::AnalogLowShelf, 16
- lir::ChebyshevII::BandPass< FilterOrder, StateType >, 20
  - setup, 21, 22
- lir::ChebyshevII::BandPassBase, 24
- lir::ChebyshevII::BandShelf< FilterOrder, StateType >, 30
  - setup, 31, 32
- lir::ChebyshevII::BandShelfBase, 32
- lir::ChebyshevII::BandStop< FilterOrder, StateType >, 36
  - setup, 36, 37
- lir::ChebyshevII::BandStopBase, 40
- lir::ChebyshevII::HighPass< FilterOrder, StateType >, 50
  - setup, 50, 51
- lir::ChebyshevII::HighPassBase, 53
- lir::ChebyshevII::HighShelf< FilterOrder, StateType >, 59
  - setup, 59, 60
- lir::ChebyshevII::HighShelfBase, 61
- lir::ChebyshevII::LowPass< FilterOrder, StateType >, 64
  - setup, 65
- lir::ChebyshevII::LowPassBase, 70
- lir::ChebyshevII::LowShelf< FilterOrder, StateType >, 72
  - setup, 73, 74
- lir::ChebyshevII::LowShelfBase, 77
- lir::ComplexPair, 46
  - isMatchedPair, 46
- lir::Custom, 15
- lir::Custom::OnePole, 78
- lir::Custom::SOSCascade< NSOS, StateType >, 81
  - setup, 82
  - SOSCascade, 82
- lir::Custom::TwoPole, 83
- lir::DirectFormI, 47
- lir::DirectFormII, 47
- lir::HighPassTransform, 54
- lir::Layout< MaxPoles >, 63
- lir::LayoutBase, 63
- lir::LowPassTransform, 70
- lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >, 79
- lir::PoleFilterBase< AnalogPrototype >, 79
- lir::PoleFilterBase2, 79
- lir::PoleZeroPair, 80
- lir::RBJ::AllPass, 15
- lir::RBJ::BandPass1, 22
  - setup, 23
- lir::RBJ::BandPass2, 23
  - setup, 24
- lir::RBJ::BandShelf, 27
  - setup, 27
- lir::RBJ::BandStop, 35
  - setup, 35
- lir::RBJ::HighPass, 49
  - setup, 49
- lir::RBJ::HighShelf, 54
  - setup, 55
- lir::RBJ::IIRNotch, 62
  - setup, 62
- lir::RBJ::LowPass, 63
  - setup, 64
- lir::RBJ::LowShelf, 76
  - setup, 77
- lir::RBJ::RBJbase, 80
- lir::TransposedDirectFormII, 83
- isMatchedPair
  - lir::ComplexPair, 46
- Kind
  - lir, 13
- operator[]
  - lir::Cascade, 45
- reset
  - lir::CascadeStages< MaxStages, StateType >, 46
- response
  - lir::Biquad, 43
  - lir::Cascade, 45
- setCoefficients
  - lir::Biquad, 43
- setIdentity
  - lir::Biquad, 43
- setOnePole
  - lir::Biquad, 43
- setPoleZeroPair
  - lir::Biquad, 43
- setTwoPole
  - lir::Biquad, 43
- setup
  - lir::Butterworth::BandPass< FilterOrder, StateType >, 19, 20
  - lir::Butterworth::BandShelf< FilterOrder, StateType >, 26, 27
  - lir::Butterworth::BandStop< FilterOrder, StateType >, 38, 39
  - lir::Butterworth::HighPass< FilterOrder, StateType >, 52, 53
  - lir::Butterworth::HighShelf< FilterOrder, StateType >, 58
  - lir::Butterworth::LowPass< FilterOrder, StateType >, 66, 67
  - lir::Butterworth::LowShelf< FilterOrder, StateType >, 75, 76
  - lir::CascadeStages< MaxStages, StateType >, 46

- lir::ChebyshevI::BandPass< FilterOrder, StateType >, [17](#), [18](#)
- lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [28](#), [29](#)
- lir::ChebyshevI::BandStop< FilterOrder, StateType >, [34](#), [35](#)
- lir::ChebyshevI::HighPass< FilterOrder, StateType >, [48](#)
- lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [56](#), [57](#)
- lir::ChebyshevI::LowPass< FilterOrder, StateType >, [68](#), [69](#)
- lir::ChebyshevI::LowShelf< FilterOrder, StateType >, [71](#), [72](#)
- lir::ChebyshevII::BandPass< FilterOrder, StateType >, [21](#), [22](#)
- lir::ChebyshevII::BandShelf< FilterOrder, StateType >, [31](#), [32](#)
- lir::ChebyshevII::BandStop< FilterOrder, StateType >, [36](#), [37](#)
- lir::ChebyshevII::HighPass< FilterOrder, StateType >, [50](#), [51](#)
- lir::ChebyshevII::HighShelf< FilterOrder, StateType >, [59](#), [60](#)
- lir::ChebyshevII::LowPass< FilterOrder, StateType >, [65](#)
- lir::ChebyshevII::LowShelf< FilterOrder, StateType >, [73](#), [74](#)
- lir::Custom::SOSCascade< NSOS, StateType >, [82](#)
- lir::RBJ::BandPass1, [23](#)
- lir::RBJ::BandPass2, [24](#)
- lir::RBJ::BandShelf, [27](#)
- lir::RBJ::BandStop, [35](#)
- lir::RBJ::HighPass, [49](#)
- lir::RBJ::HighShelf, [55](#)
- lir::RBJ::IIRNotch, [62](#)
- lir::RBJ::LowPass, [64](#)
- lir::RBJ::LowShelf, [77](#)

SOSCascade

- lir::Custom::SOSCascade< NSOS, StateType >, [82](#)

Storage

- lir::Cascade::Storage, [83](#)