

iir1

Generated by Doxygen 1.8.17

1 DSP IIR Realtime C++ filter library	1
2 Namespace Index	2
2.1 Namespace List	2
3 Hierarchical Index	2
3.1 Class Hierarchy	2
4 Class Index	6
4.1 Class List	6
5 Namespace Documentation	8
5.1 Iir Namespace Reference	8
5.1.1 Detailed Description	9
5.1.2 Enumeration Type Documentation	11
5.2 Iir::Butterworth Namespace Reference	11
5.2.1 Detailed Description	11
5.3 Iir::ChebyshevI Namespace Reference	11
5.3.1 Detailed Description	12
5.4 Iir::ChebyshevII Namespace Reference	12
5.4.1 Detailed Description	12
5.5 Iir::Custom Namespace Reference	12
5.5.1 Detailed Description	12
6 Class Documentation	12
6.1 Iir::RBJ::AllPass Struct Reference	12
6.1.1 Detailed Description	13
6.1.2 Member Function Documentation	13
6.2 Iir::Butterworth::AnalogLowPass Class Reference	13
6.2.1 Detailed Description	13
6.3 Iir::ChebyshevI::AnalogLowPass Class Reference	14
6.3.1 Detailed Description	14
6.4 Iir::ChebyshevII::AnalogLowPass Class Reference	14
6.4.1 Detailed Description	14
6.5 Iir::ChebyshevII::AnalogLowShelf Class Reference	14
6.5.1 Detailed Description	14
6.6 Iir::Butterworth::AnalogLowShelf Class Reference	15
6.6.1 Detailed Description	15
6.7 Iir::ChebyshevI::AnalogLowShelf Class Reference	15
6.7.1 Detailed Description	15
6.8 Iir::ChebyshevI::BandPass< FilterOrder, StateType > Struct Template Reference	15
6.8.1 Detailed Description	16
6.8.2 Member Function Documentation	16
6.9 Iir::Butterworth::BandPass< FilterOrder, StateType > Struct Template Reference	17

6.9.1 Detailed Description	17
6.9.2 Member Function Documentation	18
6.10 lir::ChebyshevII::BandPass< FilterOrder, StateType > Struct Template Reference	19
6.10.1 Detailed Description	19
6.10.2 Member Function Documentation	19
6.11 lir::RBJ::BandPass1 Struct Reference	21
6.11.1 Detailed Description	21
6.11.2 Member Function Documentation	21
6.12 lir::RBJ::BandPass2 Struct Reference	22
6.12.1 Detailed Description	22
6.12.2 Member Function Documentation	22
6.13 lir::ChebyshevII::BandPassBase Struct Reference	23
6.14 lir::ChebyshevI::BandPassBase Struct Reference	23
6.15 lir::Butterworth::BandPassBase Struct Reference	24
6.16 lir::BandPassTransform Class Reference	24
6.16.1 Detailed Description	24
6.17 lir::Butterworth::BandShelf< FilterOrder, StateType > Struct Template Reference	24
6.17.1 Detailed Description	25
6.17.2 Member Function Documentation	25
6.18 lir::RBJ::BandShelf Struct Reference	26
6.18.1 Detailed Description	26
6.18.2 Member Function Documentation	26
6.19 lir::ChebyshevI::BandShelf< FilterOrder, StateType > Struct Template Reference	27
6.19.1 Detailed Description	27
6.19.2 Member Function Documentation	28
6.20 lir::ChebyshevII::BandShelf< FilterOrder, StateType > Struct Template Reference	29
6.20.1 Detailed Description	30
6.20.2 Member Function Documentation	30
6.21 lir::Butterworth::BandShelfBase Struct Reference	31
6.22 lir::ChebyshevII::BandShelfBase Struct Reference	32
6.23 lir::ChebyshevI::BandShelfBase Struct Reference	32
6.24 lir::ChebyshevI::BandStop< FilterOrder, StateType > Struct Template Reference	33
6.24.1 Detailed Description	33
6.24.2 Member Function Documentation	33
6.25 lir::RBJ::BandStop Struct Reference	35
6.25.1 Detailed Description	35
6.25.2 Member Function Documentation	35
6.26 lir::ChebyshevII::BandStop< FilterOrder, StateType > Struct Template Reference	36
6.26.1 Detailed Description	36
6.26.2 Member Function Documentation	36
6.27 lir::Butterworth::BandStop< FilterOrder, StateType > Struct Template Reference	37
6.27.1 Detailed Description	38

6.27.2 Member Function Documentation	38
6.28 <code>lir::Butterworth::BandStopBase</code> Struct Reference	39
6.29 <code>lir::ChebyshevII::BandStopBase</code> Struct Reference	40
6.30 <code>lir::ChebyshevI::BandStopBase</code> Struct Reference	40
6.31 <code>lir::BandStopTransform</code> Class Reference	40
6.31.1 Detailed Description	40
6.32 <code>lir::Biquad</code> Class Reference	41
6.32.1 Member Function Documentation	41
6.33 <code>lir::BiquadPoleState</code> Struct Reference	43
6.33.1 Detailed Description	44
6.34 <code>lir::Cascade</code> Class Reference	44
6.34.1 Detailed Description	44
6.34.2 Member Function Documentation	44
6.35 <code>lir::CascadeStages< MaxStages, StateType ></code> Class Template Reference	45
6.35.1 Detailed Description	45
6.35.2 Member Function Documentation	45
6.36 <code>lir::ComplexPair</code> Struct Reference	46
6.36.1 Detailed Description	46
6.36.2 Member Function Documentation	46
6.37 <code>lir::DirectFormI</code> Class Reference	46
6.37.1 Detailed Description	46
6.38 <code>lir::DirectFormII</code> Class Reference	47
6.38.1 Detailed Description	47
6.39 <code>lir::ChebyshevI::HighPass< FilterOrder, StateType ></code> Struct Template Reference	47
6.39.1 Detailed Description	47
6.39.2 Member Function Documentation	47
6.40 <code>lir::RBJ::HighPass</code> Struct Reference	49
6.40.1 Detailed Description	49
6.40.2 Member Function Documentation	49
6.41 <code>lir::ChebyshevII::HighPass< FilterOrder, StateType ></code> Struct Template Reference	50
6.41.1 Detailed Description	50
6.41.2 Member Function Documentation	50
6.42 <code>lir::Butterworth::HighPass< FilterOrder, StateType ></code> Struct Template Reference	51
6.42.1 Detailed Description	52
6.42.2 Member Function Documentation	52
6.43 <code>lir::ChebyshevII::HighPassBase</code> Struct Reference	53
6.44 <code>lir::ChebyshevI::HighPassBase</code> Struct Reference	53
6.45 <code>lir::Butterworth::HighPassBase</code> Struct Reference	54
6.46 <code>lir::HighPassTransform</code> Class Reference	54
6.46.1 Detailed Description	54
6.47 <code>lir::Butterworth::HighShelf< FilterOrder, StateType ></code> Struct Template Reference	54
6.47.1 Detailed Description	54

6.47.2 Member Function Documentation	55
6.48 lir::RBJ::HighShelf Struct Reference	56
6.48.1 Detailed Description	56
6.48.2 Member Function Documentation	56
6.49 lir::ChebyshevI::HighShelf< FilterOrder, StateType > Struct Template Reference	57
6.49.1 Detailed Description	57
6.49.2 Member Function Documentation	57
6.50 lir::ChebyshevII::HighShelf< FilterOrder, StateType > Struct Template Reference	59
6.50.1 Detailed Description	59
6.50.2 Member Function Documentation	59
6.51 lir::ChebyshevII::HighShelfBase Struct Reference	61
6.52 lir::Butterworth::HighShelfBase Struct Reference	61
6.53 lir::ChebyshevI::HighShelfBase Struct Reference	61
6.54 lir::RBJ::IIRNotch Struct Reference	62
6.54.1 Detailed Description	62
6.54.2 Member Function Documentation	62
6.55 lir::Layout< MaxPoles > Class Template Reference	63
6.55.1 Detailed Description	63
6.56 lir::LayoutBase Class Reference	63
6.56.1 Detailed Description	63
6.57 lir::RBJ::LowPass Struct Reference	63
6.57.1 Detailed Description	64
6.57.2 Member Function Documentation	64
6.58 lir::ChebyshevII::LowPass< FilterOrder, StateType > Struct Template Reference	64
6.58.1 Detailed Description	65
6.58.2 Member Function Documentation	65
6.59 lir::Butterworth::LowPass< FilterOrder, StateType > Struct Template Reference	66
6.59.1 Detailed Description	67
6.59.2 Member Function Documentation	67
6.60 lir::ChebyshevI::LowPass< FilterOrder, StateType > Struct Template Reference	68
6.60.1 Detailed Description	68
6.60.2 Member Function Documentation	68
6.61 lir::ChebyshevI::LowPassBase Struct Reference	70
6.62 lir::Butterworth::LowPassBase Struct Reference	70
6.63 lir::ChebyshevII::LowPassBase Struct Reference	70
6.64 lir::LowPassTransform Class Reference	71
6.64.1 Detailed Description	71
6.65 lir::ChebyshevI::LowShelf< FilterOrder, StateType > Struct Template Reference	71
6.65.1 Detailed Description	71
6.65.2 Member Function Documentation	71
6.66 lir::Butterworth::LowShelf< FilterOrder, StateType > Struct Template Reference	73
6.66.1 Detailed Description	73

6.66.2 Member Function Documentation	73
6.67 lir::RBJ::LowShelf Struct Reference	75
6.67.1 Detailed Description	75
6.67.2 Member Function Documentation	75
6.68 lir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference	76
6.68.1 Detailed Description	76
6.68.2 Member Function Documentation	76
6.69 lir::Butterworth::LowShelfBase Struct Reference	77
6.70 lir::ChebyshevI::LowShelfBase Struct Reference	78
6.71 lir::ChebyshevII::LowShelfBase Struct Reference	78
6.72 lir::Custom::OnePole Struct Reference	79
6.72.1 Detailed Description	79
6.73 lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > Struct Template Reference	79
6.73.1 Detailed Description	79
6.74 lir::PoleFilterBase< AnalogPrototype > Class Template Reference	80
6.74.1 Detailed Description	80
6.75 lir::PoleFilterBase2 Class Reference	80
6.75.1 Detailed Description	80
6.76 lir::PoleZeroPair Struct Reference	81
6.76.1 Detailed Description	81
6.77 lir::RBJ::RBJbase Struct Reference	81
6.77.1 Detailed Description	82
6.78 lir::Custom::SOSCascade< NSOS, StateType > Struct Template Reference	82
6.78.1 Detailed Description	82
6.78.2 Constructor & Destructor Documentation	82
6.78.3 Member Function Documentation	83
6.79 lir::Cascade::Storage Struct Reference	83
6.79.1 Detailed Description	83
6.79.2 Constructor & Destructor Documentation	83
6.80 lir::TransposedDirectFormII Class Reference	84
6.81 lir::Custom::TwoPole Struct Reference	84
6.81.1 Detailed Description	84
Index	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 any risk of memory leaks.

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

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

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

2 Namespace Index

2.1 Namespace List

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

lir	8
lir::Butterworth	11
lir::ChebyshevI	11
lir::ChebyshevII	12
lir::Custom	12

3 Hierarchical Index

3.1 Class Hierarchy

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

BandPassBase

lir::PoleFilter< BandPassBase, DirectFormII, 4, 4 *2 >	79
lir::Butterworth::BandPass< FilterOrder, StateType >	17
lir::ChebyshevI::BandPass< FilterOrder, StateType >	15
lir::ChebyshevII::BandPass< FilterOrder, StateType >	19

[lir::BandPassTransform](#)

BandShelfBase

lir::PoleFilter< BandShelfBase, DirectFormII, 4, 4 *2 >	79
lir::Butterworth::BandShelf< FilterOrder, StateType >	24
lir::ChebyshevI::BandShelf< FilterOrder, StateType >	27
lir::ChebyshevII::BandShelf< FilterOrder, StateType >	29

BandStopBase

lir::PoleFilter< BandStopBase, DirectFormII, 4, 4 *2 >	79
lir::Butterworth::BandStop< FilterOrder, StateType >	37

lir::ChebyshevI::BandStop< FilterOrder, StateType >	33
lir::ChebyshevII::BandStop< FilterOrder, StateType >	36
lir::BandStopTransform	40
BaseClass	
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	79
lir::Biquad	41
lir::Custom::OnePole	79
lir::Custom::TwoPole	84
lir::RBJ::RBJbase	81
lir::RBJ::AllPass	12
lir::RBJ::BandPass1	21
lir::RBJ::BandPass2	22
lir::RBJ::BandShelf	26
lir::RBJ::BandStop	35
lir::RBJ::HighPass	49
lir::RBJ::HighShelf	56
lir::RBJ::IIRNotch	62
lir::RBJ::LowPass	63
lir::RBJ::LowShelf	75
lir::Cascade	44
lir::PoleFilterBase2	80
lir::PoleFilterBase< AnalogPrototype >	80
lir::PoleFilterBase< AnalogLowPass >	80
lir::Butterworth::BandPassBase	24
lir::Butterworth::BandStopBase	39
lir::Butterworth::HighPassBase	54
lir::Butterworth::LowPassBase	70
lir::ChebyshevI::BandPassBase	23
lir::ChebyshevI::BandStopBase	40
lir::ChebyshevI::HighPassBase	53
lir::ChebyshevI::LowPassBase	70
lir::ChebyshevII::BandPassBase	23

lir::ChebyshevII::BandStopBase	40
lir::ChebyshevII::HighPassBase	53
lir::ChebyshevII::LowPassBase	70
lir::PoleFilterBase< AnalogLowShelf >	80
lir::Butterworth::BandShelfBase	31
lir::Butterworth::HighShelfBase	61
lir::Butterworth::LowShelfBase	77
lir::ChebyshevI::BandShelfBase	32
lir::ChebyshevI::HighShelfBase	61
lir::ChebyshevI::LowShelfBase	78
lir::ChebyshevII::BandShelfBase	32
lir::ChebyshevII::HighShelfBase	61
lir::ChebyshevII::LowShelfBase	78
lir::CascadeStages< MaxStages, StateType >	45
lir::CascadeStages< NSOS, DirectFormII >	45
lir::Custom::SOSCascade< NSOS, StateType >	82
lir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >	45
lir::PoleFilter< HighPassBase, DirectFormII, 4 >	79
lir::Butterworth::HighPass< FilterOrder, StateType >	51
lir::ChebyshevI::HighPass< FilterOrder, StateType >	47
lir::ChebyshevII::HighPass< FilterOrder, StateType >	50
lir::PoleFilter< HighShelfBase, DirectFormII, 4 >	79
lir::Butterworth::HighShelf< FilterOrder, StateType >	54
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	57
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	59
lir::PoleFilter< LowPassBase, DirectFormII, 4 >	79
lir::Butterworth::LowPass< FilterOrder, StateType >	66
lir::ChebyshevI::LowPass< FilterOrder, StateType >	68
lir::ChebyshevII::LowPass< FilterOrder, StateType >	64
lir::PoleFilter< LowShelfBase, DirectFormII, 4 >	79
lir::Butterworth::LowShelf< FilterOrder, StateType >	73
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	71

lir::ChebyshevII::LowShelf< FilterOrder, StateType >	76
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, 4, 4 *2 >	79
lir::PoleFilter< BandShelfBase, DirectFormII, 4, 4 *2 >	79
lir::PoleFilter< BandStopBase, DirectFormII, 4, 4 *2 >	79
complex_pair_t	
lir::ComplexPair	46
lir::DirectFormI	46
lir::DirectFormII	47
HighPassBase	
lir::PoleFilter< HighPassBase, DirectFormII, 4 >	79
lir::HighPassTransform	54
HighShelfBase	
lir::PoleFilter< HighShelfBase, DirectFormII, 4 >	79
lir::Layout< MaxPoles >	63
lir::Layout< MaxAnalogPoles >	63
lir::Layout< MaxDigitalPoles >	63
lir::LayoutBase	63
lir::Butterworth::AnalogLowPass	13
lir::Butterworth::AnalogLowShelf	15
lir::ChebyshevI::AnalogLowPass	14
lir::ChebyshevI::AnalogLowShelf	15
lir::ChebyshevII::AnalogLowPass	14
lir::ChebyshevII::AnalogLowShelf	14
LowPassBase	
lir::PoleFilter< LowPassBase, DirectFormII, 4 >	79
lir::LowPassTransform	71
LowShelfBase	
lir::PoleFilter< LowShelfBase, DirectFormII, 4 >	79
lir::PoleZeroPair	81
lir::BiquadPoleState	43
lir::Cascade::Storage	83

4 Class Index

4.1 Class List

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

lir::RBJ::AllPass	12
lir::Butterworth::AnalogLowPass	13
lir::ChebyshevI::AnalogLowPass	14
lir::ChebyshevII::AnalogLowPass	14
lir::ChebyshevII::AnalogLowShelf	14
lir::Butterworth::AnalogLowShelf	15
lir::ChebyshevI::AnalogLowShelf	15
lir::ChebyshevI::BandPass< FilterOrder, StateType >	15
lir::Butterworth::BandPass< FilterOrder, StateType >	17
lir::ChebyshevII::BandPass< FilterOrder, StateType >	19
lir::RBJ::BandPass1	21
lir::RBJ::BandPass2	22
lir::ChebyshevII::BandPassBase	23
lir::ChebyshevI::BandPassBase	23
lir::Butterworth::BandPassBase	24
lir::BandPassTransform	24
lir::Butterworth::BandShelf< FilterOrder, StateType >	24
lir::RBJ::BandShelf	26
lir::ChebyshevI::BandShelf< FilterOrder, StateType >	27
lir::ChebyshevII::BandShelf< FilterOrder, StateType >	29
lir::Butterworth::BandShelfBase	31
lir::ChebyshevII::BandShelfBase	32
lir::ChebyshevI::BandShelfBase	32
lir::ChebyshevI::BandStop< FilterOrder, StateType >	33
lir::RBJ::BandStop	35
lir::ChebyshevII::BandStop< FilterOrder, StateType >	36

lir::Butterworth::BandStop< FilterOrder, StateType >	37
lir::Butterworth::BandStopBase	39
lir::ChebyshevII::BandStopBase	40
lir::ChebyshevI::BandStopBase	40
lir::BandStopTransform	40
lir::Biquad	41
lir::BiquadPoleState	43
lir::Cascade	44
lir::CascadeStages< MaxStages, StateType >	45
lir::ComplexPair	46
lir::DirectFormI	46
lir::DirectFormII	47
lir::ChebyshevI::HighPass< FilterOrder, StateType >	47
lir::RBJ::HighPass	49
lir::ChebyshevII::HighPass< FilterOrder, StateType >	50
lir::Butterworth::HighPass< FilterOrder, StateType >	51
lir::ChebyshevII::HighPassBase	53
lir::ChebyshevI::HighPassBase	53
lir::Butterworth::HighPassBase	54
lir::HighPassTransform	54
lir::Butterworth::HighShelf< FilterOrder, StateType >	54
lir::RBJ::HighShelf	56
lir::ChebyshevI::HighShelf< FilterOrder, StateType >	57
lir::ChebyshevII::HighShelf< FilterOrder, StateType >	59
lir::ChebyshevII::HighShelfBase	61
lir::Butterworth::HighShelfBase	61
lir::ChebyshevI::HighShelfBase	61
lir::RBJ::IIRNotch	62
lir::Layout< MaxPoles >	63
lir::LayoutBase	63
lir::RBJ::LowPass	63
lir::ChebyshevII::LowPass< FilterOrder, StateType >	64

lir::Butterworth::LowPass< FilterOrder, StateType >	66
lir::ChebyshevI::LowPass< FilterOrder, StateType >	68
lir::ChebyshevI::LowPassBase	70
lir::Butterworth::LowPassBase	70
lir::ChebyshevII::LowPassBase	70
lir::LowPassTransform	71
lir::ChebyshevI::LowShelf< FilterOrder, StateType >	71
lir::Butterworth::LowShelf< FilterOrder, StateType >	73
lir::RBJ::LowShelf	75
lir::ChebyshevII::LowShelf< FilterOrder, StateType >	76
lir::Butterworth::LowShelfBase	77
lir::ChebyshevI::LowShelfBase	78
lir::ChebyshevII::LowShelfBase	78
lir::Custom::OnePole	79
lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >	79
lir::PoleFilterBase< AnalogPrototype >	80
lir::PoleFilterBase2	80
lir::PoleZeroPair	81
lir::RBJ::RBJbase	81
lir::Custom::SOSCascade< NSOS, StateType >	82
lir::Cascade::Storage	83
lir::TransposedDirectFormII	84
lir::Custom::TwoPole	84

5 Namespace Documentation

5.1 lir Namespace Reference

Namespaces

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

Classes

- class [BandPassTransform](#)
- class [BandStopTransform](#)
- class [Biquad](#)
- struct [BiquadPoleState](#)
- class [Cascade](#)
- class [CascadeStages](#)
- struct [ComplexPair](#)
- class [DirectFormI](#)
- class [DirectFormII](#)
- class [HighPassTransform](#)
- class [Layout](#)
- class [LayoutBase](#)
- class [LowPassTransform](#)
- struct [PoleFilter](#)
- class [PoleFilterBase](#)
- class [PoleFilterBase2](#)
- struct [PoleZeroPair](#)
- class [TransposedDirectFormII](#)

Enumerations

- enum [Kind](#)

5.1.1 Detailed Description

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

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

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

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

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

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

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

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

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

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

License: MIT License (<http://www.opensource.org/licenses/mit-license.php>) Copyright (c) 2009 by Vinnie Falco Copyright (c) 2011-2021 by Bernd Porr

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

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

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

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

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

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

License: MIT License (<http://www.opensource.org/licenses/mit-license.php>) Copyright (c) 2009 by Vinnie Falco Copyright (c) 2011-2021 by Bernd Porr

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

normalization information to achieve a specified gain at a specified frequency. The poles and zeros may lie either in the s or the z plane.

5.1.2 Enumeration Type Documentation

5.1.2.1 `Kind` enum `lir::Kind`

Identifies the general class of filter

5.2 `lir::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 `lir::Custom` Namespace Reference

Classes

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

5.5.1 Detailed Description

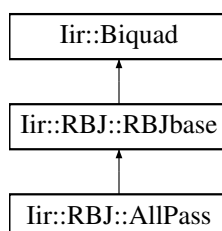
Single pole, [Biquad](#) and cascade of Biquads with parameters allowing for directly setting the parameters.

6 Class Documentation

6.1 `lir::RBJ::AllPass` Struct Reference

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

Inheritance diagram for `lir::RBJ::AllPass`:



Public Member Functions

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

6.1.1 Detailed Description

Allpass filter

6.1.2 Member Function Documentation

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

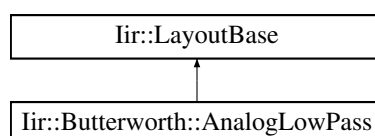
The documentation for this struct was generated from the following files:

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

6.2 Iir::Butterworth::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::Butterworth::AnalogLowPass:



6.2.1 Detailed Description

Analogue lowpass prototypes (s-plane)

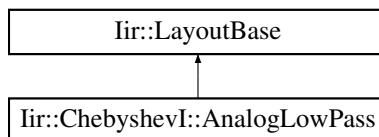
The documentation for this class was generated from the following files:

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

6.3 Iir::ChebyshevI::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::ChebyshevI::AnalogLowPass:



6.3.1 Detailed Description

Analog lowpass prototypes (s-plane)

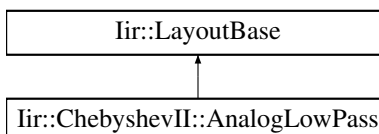
The documentation for this class was generated from the following files:

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

6.4 Iir::ChebyshevII::AnalogLowPass Class Reference

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

Inheritance diagram for Iir::ChebyshevII::AnalogLowPass:



6.4.1 Detailed Description

Analogue lowpass prototype (s-plane)

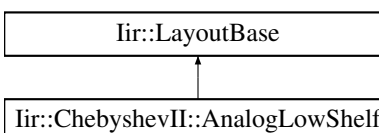
The documentation for this class was generated from the following files:

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

6.5 Iir::ChebyshevII::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::ChebyshevII::AnalogLowShelf:



6.5.1 Detailed Description

Analogue shelf lowpass prototype (s-plane)

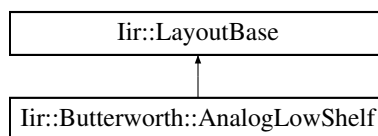
The documentation for this class was generated from the following files:

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

6.6 Iir::Butterworth::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::Butterworth::AnalogLowShelf:



6.6.1 Detailed Description

Analogue low shelf prototypes (s-plane)

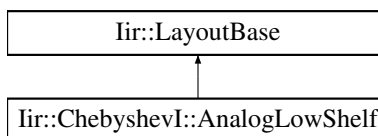
The documentation for this class was generated from the following files:

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

6.7 Iir::ChebyshevI::AnalogLowShelf Class Reference

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

Inheritance diagram for Iir::ChebyshevI::AnalogLowShelf:



6.7.1 Detailed Description

Analog lowpass shelf prototype (s-plane)

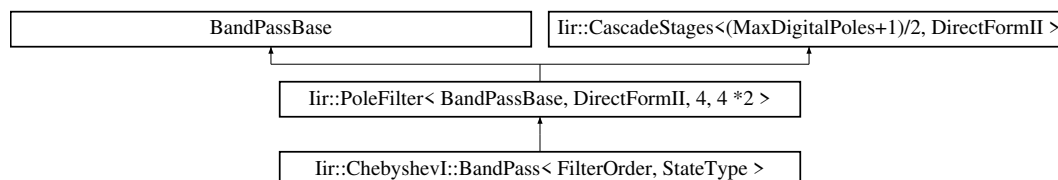
The documentation for this class was generated from the following files:

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

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

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

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



Public Member Functions

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

6.8.1 Detailed Description

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

[ChebyshevI](#) bandpass filter

Parameters

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

6.8.2 Member Function Documentation

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

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

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

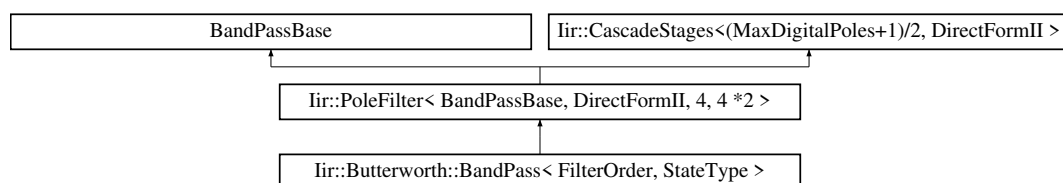
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

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

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



Public Member Functions

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

6.9.1 Detailed Description

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

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

[Butterworth](#) Bandpass filter.

Parameters

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

6.9.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

```

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

```

Calculates the coefficients

Parameters

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

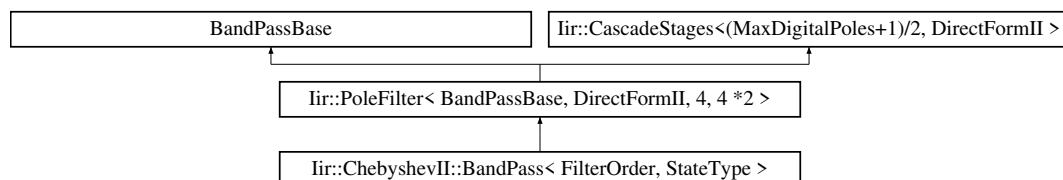
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

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

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

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



Public Member Functions

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

6.10.1 Detailed Description

```

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

```

[ChebyshevII](#) bandpass filter

Parameters

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

6.10.2 Member Function Documentation

```

6.10.2.1 setup() [1/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevII::BandPass< FilterOrder, StateType >::setup (
    double sampleRate,

```



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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

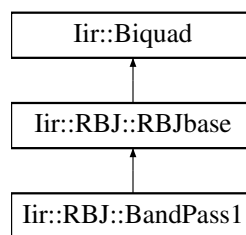
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

6.11 Iir::RBJ::BandPass1 Struct Reference

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

Inheritance diagram for Iir::RBJ::BandPass1:



Public Member Functions

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

6.11.1 Detailed Description

Bandpass with constant skirt gain

6.11.2 Member Function Documentation

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

Calculates the coefficients

Parameters

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

6.11.2.2 setupN() void Iir::RBJ::BandPass1::setupN (
double *centerFrequency*,
double *bandWidth*)

Calculates the coefficients

Parameters

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

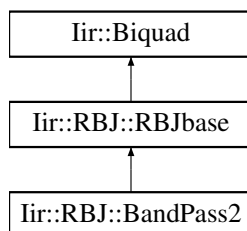
The documentation for this struct was generated from the following files:

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

6.12 Iir::RBJ::BandPass2 Struct Reference

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

Inheritance diagram for Iir::RBJ::BandPass2:



Public Member Functions

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

6.12.1 Detailed Description

Bandpass with constant 0 dB peak gain

6.12.2 Member Function Documentation

6.12.2.1 setup() void Iir::RBJ::BandPass2::setup (
double *sampleRate*,
double *centerFrequency*,
double *bandWidth*) [inline]

Calculates the coefficients

Parameters

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

6.12.2.2 setupN() void Iir::RBJ::BandPass2::setupN (
double *centerFrequency*,
double *bandWidth*)

Calculates the coefficients

Parameters

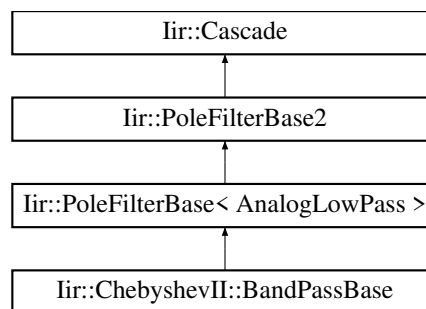
<i>centerFrequency</i>	Normalised centre frequency of the bandpass
<i>bandWidth</i>	Bandwidth in octaves

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

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

6.13 Iir::ChebyshevII::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandPassBase:



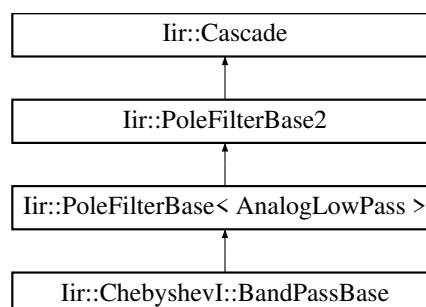
Additional Inherited Members

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

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

6.14 Iir::ChebyshevI::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandPassBase:



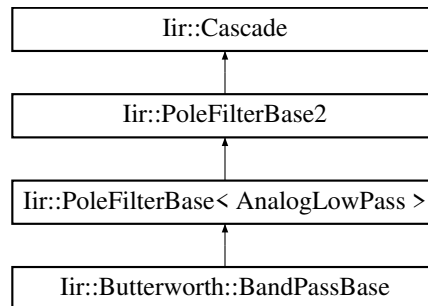
Additional Inherited Members

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

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

6.15 Iir::Butterworth::BandPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandPassBase:



Additional Inherited Members

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

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

6.16 Iir::BandPassTransform Class Reference

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

6.16.1 Detailed Description

low pass to band pass transform

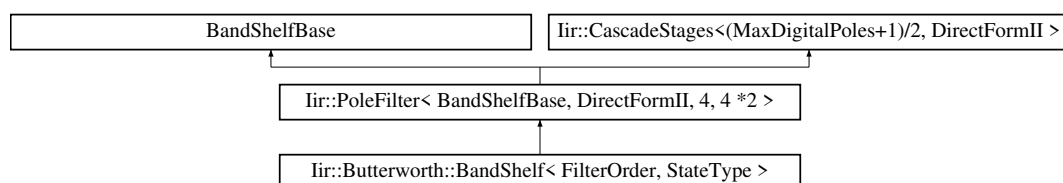
The documentation for this class was generated from the following files:

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

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

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

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



Public Member Functions

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

6.17.1 Detailed Description

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

Butterworth Bandsshelf 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: DirectFormI , DirectFormII , ...

6.17.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

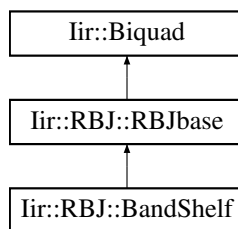
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.18 Iir::RBJ::BandShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::BandShelf:



Public Member Functions

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

6.18.1 Detailed Description

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

6.18.2 Member Function Documentation

6.18.2.1 setup() `void Iir::RBJ::BandShelf::setup (`
`double sampleRate,`
`double centerFrequency,`
`double gainDb,`
`double bandWidth) [inline]`

Calculates the coefficients

Parameters

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

6.18.2.2 setupN() `void Iir::RBJ::BandShelf::setupN (`
`double centerFrequency,`
`double gainDb,`
`double bandWidth)`

Calculates the coefficients

Parameters

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

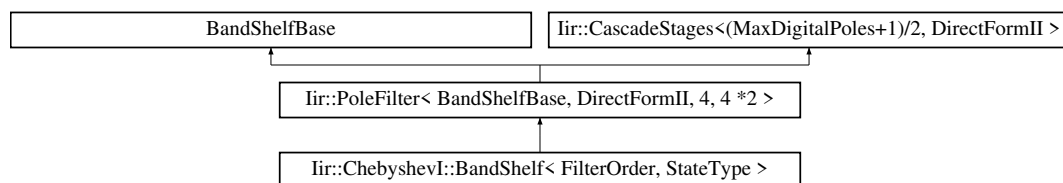
The documentation for this struct was generated from the following files:

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

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

`#include <ChebyshevI.h>`

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



Public Member Functions

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

6.19.1 Detailed Description


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

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

Parameters

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

6.19.2 Member Function Documentation

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

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

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

Parameters

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

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the passband
<i>widthFrequency</i>	Width of the passband.
<i>gainDb</i>	Gain in the passband. The stopband has 0 dB.
<i>rippleDb</i>	Permitted ripples in dB in the passband.

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

Calculates the coefficients of the filter at specified order

Parameters

<i>reqOrder</i>	Actual order for the filter calculations
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the passband
<i>widthFrequency</i>	Width of the passband.
<i>gainDb</i>	Gain in the passband. The stopband has 0 dB.
<i>rippleDb</i>	Permitted ripples in dB in the passband.

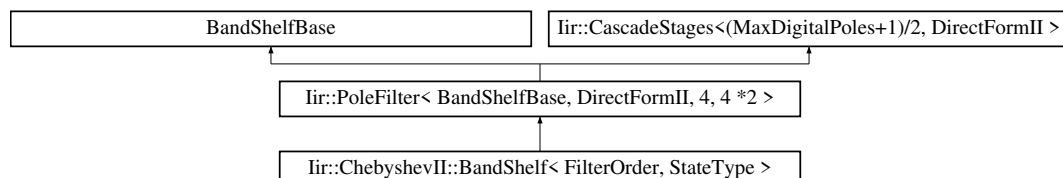
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

`#include <ChebyshevII.h>`

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



Public Member Functions

- void `setup` (double sampleRate, double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void `setup` (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency, double gain↵ Db, double stopBandDb)

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

6.20.1 Detailed Description

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

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

Parameters

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

6.20.2 Member Function Documentation

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

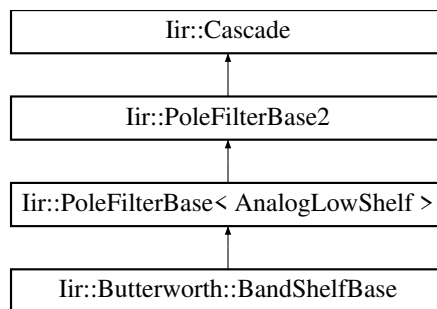
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>centerFrequency</i>	Normalised centre frequency (0..1/2) of the bandpass
<i>widthFrequency</i>	Width of the bandpass
<i>gainDb</i>	Gain in the passband. The stopband has always 0dB.
<i>stopBandDb</i>	Permitted ripples in dB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

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

- iir/ChebyshevII.h

6.21 Iir::Butterworth::BandShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandShelfBase:



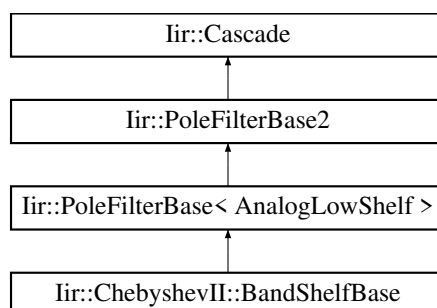
Additional Inherited Members

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

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

6.22 Iir::ChebyshevII::BandShelfBase Struct Reference

Inheritance diagram for `Iir::ChebyshevII::BandShelfBase`:



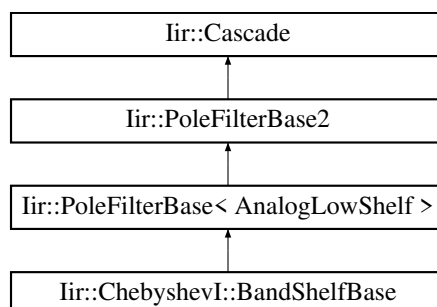
Additional Inherited Members

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

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

6.23 Iir::ChebyshevI::BandShelfBase Struct Reference

Inheritance diagram for `Iir::ChebyshevI::BandShelfBase`:



Additional Inherited Members

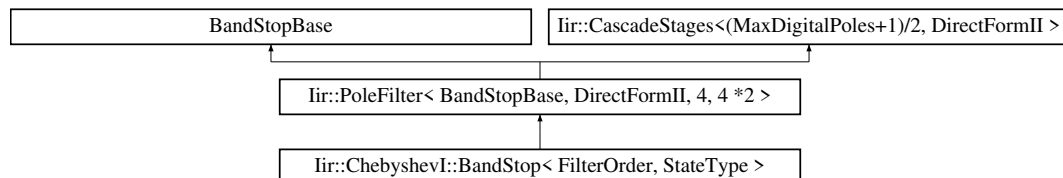
The documentation for this struct was generated from the following files:

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

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

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

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



Public Member Functions

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

6.24.1 Detailed Description

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

[ChebyshevI](#) bandstop filter

Parameters

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

6.24.2 Member Function Documentation

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

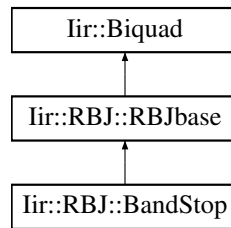
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.25 Iir::RBJ::BandStop Struct Reference

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

Inheritance diagram for Iir::RBJ::BandStop:



Public Member Functions

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

6.25.1 Detailed Description

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

6.25.2 Member Function Documentation

6.25.2.1 [setup\(\)](#) void Iir::RBJ::BandStop::setup (
 double *sampleRate*,
 double *centerFrequency*,
 double *bandWidth*) [inline]

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

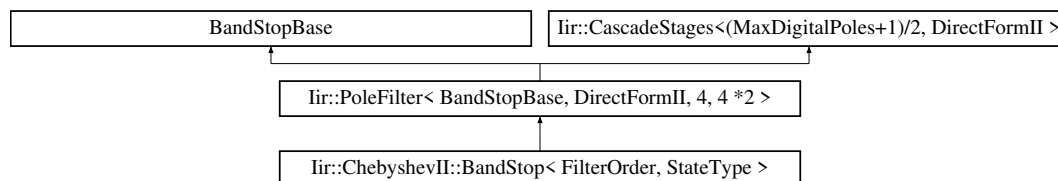
The documentation for this struct was generated from the following files:

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

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

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

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



Public Member Functions

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

6.26.1 Detailed Description

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

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

[ChebyshevII](#) bandstop filter.

Parameters

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

6.26.2 Member Function Documentation

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

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

```
    int reqOrder,
```

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

Calculates the coefficients of the filter

Parameters

<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Center frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

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

Calculates the coefficients of the filter

Parameters

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

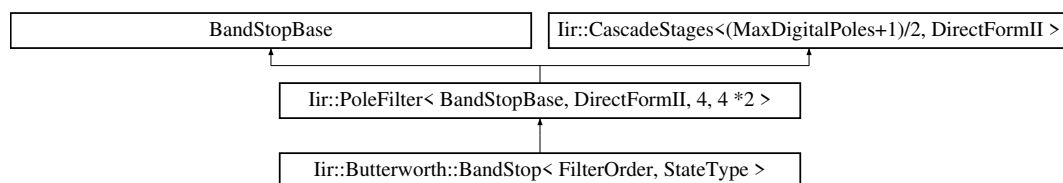
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

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

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

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



Public Member Functions

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

6.27.1 Detailed Description

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

[Butterworth](#) Bandstop filter.

Parameters

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

6.27.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

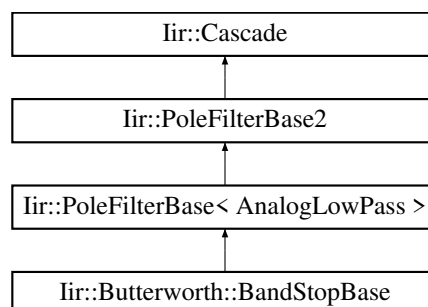
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>sampleRate</i>	Sampling rate
<i>centerFrequency</i>	Centre frequency of the bandstop
<i>widthFrequency</i>	Width of the bandstop

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

- iir/Butterworth.h

6.28 Iir::Butterworth::BandStopBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandStopBase:



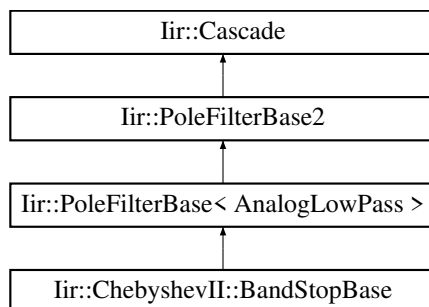
Additional Inherited Members

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

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

6.29 Iir::ChebyshevII::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandStopBase:



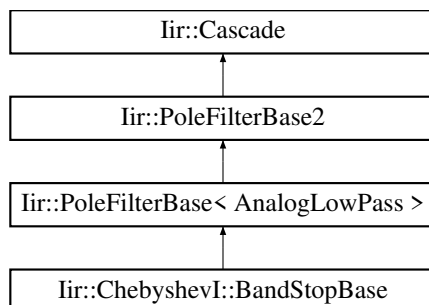
Additional Inherited Members

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

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

6.30 Iir::ChebyshevI::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandStopBase:



Additional Inherited Members

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

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

6.31 Iir::BandStopTransform Class Reference

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

6.31.1 Detailed Description

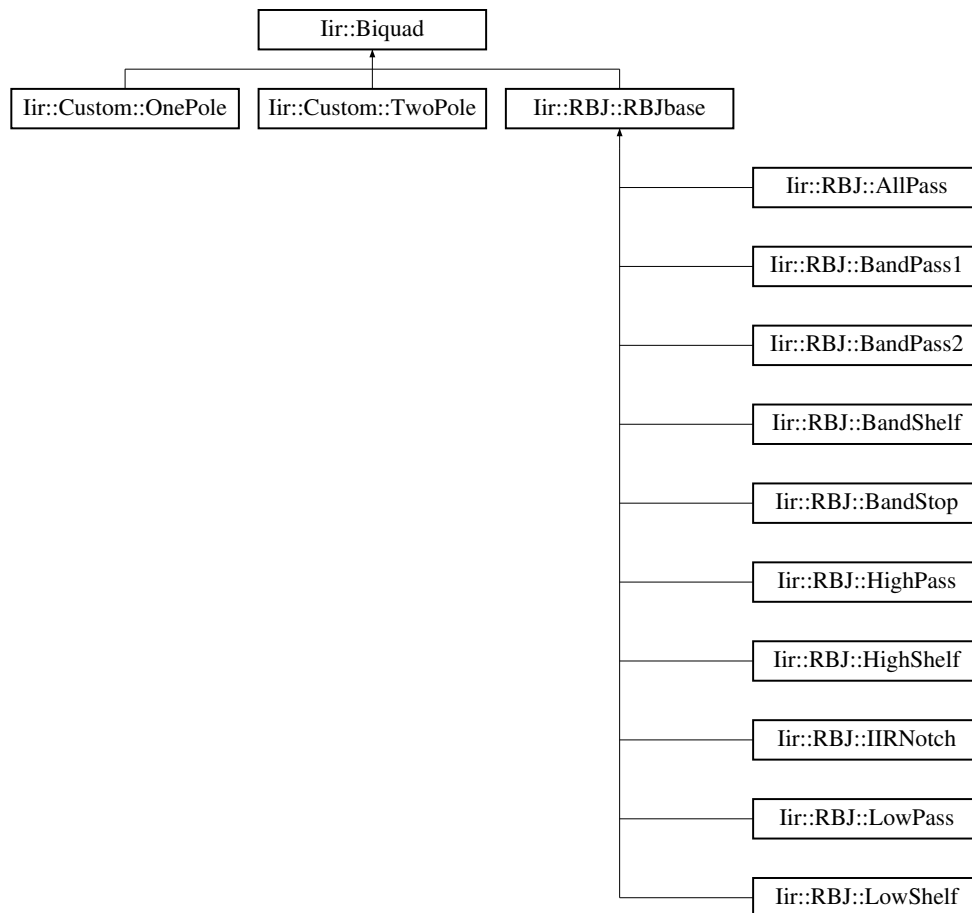
low pass to band stop transform

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

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

6.32 Iir::Biquad Class Reference

Inheritance diagram for Iir::Biquad:



Public Member Functions

- `complex_t response` (double normalizedFrequency) const
- `std::vector< PoleZeroPair > getPoleZeros` () const
- `double getA0` () const
- `double getA1` () const
- `double getA2` () const
- `double getB0` () const
- `double getB1` () const
- `double getB2` () const
- `template<class StateType >`
`double filter` (double s, StateType &state) const
- `void setCoefficients` (double a0, double a1, double a2, double b0, double b1, double b2)
- `void setOnePole` (complex_t pole, complex_t zero)
- `void setTwoPole` (complex_t pole1, complex_t zero1, complex_t pole2, complex_t zero2)
- `void setPoleZeroPair` (const PoleZeroPair &pair)
- `void setIdentity` ()
- `void applyScale` (double scale)

6.32.1 Member Function Documentation

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

Performs scaling operation on the FIR coefficients

Parameters

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

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

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

Parameters

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

Returns

The filtered sample.

6.32.1.3 getA0() `double Iir::Biquad::getA0 () const [inline]`
Returns 1st IIR coefficient (usually one)

6.32.1.4 getA1() `double Iir::Biquad::getA1 () const [inline]`
Returns 2nd IIR coefficient

6.32.1.5 getA2() `double Iir::Biquad::getA2 () const [inline]`
Returns 3rd IIR coefficient

6.32.1.6 getB0() `double Iir::Biquad::getB0 () const [inline]`
Returns 1st FIR coefficient

6.32.1.7 getB1() `double Iir::Biquad::getB1 () const [inline]`
Returns 2nd FIR coefficient

6.32.1.8 getB2() `double Iir::Biquad::getB2 () const [inline]`
Returns 3rd FIR coefficient

6.32.1.9 getPoleZeros() `std::vector< PoleZeroPair > Iir::Biquad::getPoleZeros () const`
Returns the pole / zero Pairs as a vector.

6.32.1.10 response() `complex_t Iir::Biquad::response (`
`double normalizedFrequency) const`
Calculate filter response at the given normalized frequency and return the complex response.
Gets the frequency response of the [Biquad](#)

Parameters

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

6.32.1.11 setCoefficients() `void Iir::Biquad::setCoefficients (`
`double a0,`
`double a1,`
`double a2,`
`double b0,`
`double b1,`
`double b2)`

Sets all coefficients

Parameters

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

6.32.1.12 setIdentity() `void Iir::Biquad::setIdentity ()`
 Sets the coefficients as pass through. (b0=1,a0=1, rest zero)

6.32.1.13 setOnePole() `void Iir::Biquad::setOnePole (`
`complex_t pole,`
`complex_t zero)`
 Sets one (real) pole and zero. Throws exception if imaginary components.

6.32.1.14 setPoleZeroPair() `void Iir::Biquad::setPoleZeroPair (`
`const PoleZeroPair & pair) [inline]`
 Sets a complex conjugate pair

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

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

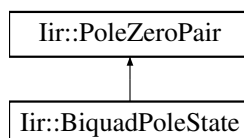
The documentation for this class was generated from the following files:

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

6.33 Iir::BiquadPoleState Struct Reference

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

Inheritance diagram for Iir::BiquadPoleState:



6.33.1 Detailed Description

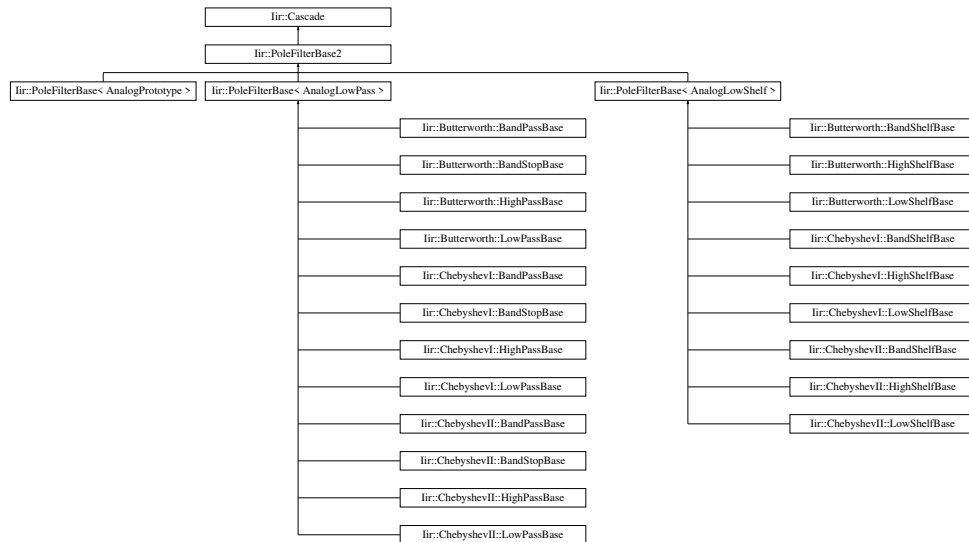
Expresses a biquad as a pair of pole/zeros, with gain values so that the coefficients can be reconstructed precisely. The documentation for this struct was generated from the following files:

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

6.34 Iir::Cascade Class Reference

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

Inheritance diagram for Iir::Cascade:



Classes

- struct [Storage](#)

Public Member Functions

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

6.34.1 Detailed Description

Holds coefficients for a cascade of second order sections.

6.34.2 Member Function Documentation

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

Returns the number of Biquads kept here

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

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

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

Returns a reference to a biquad

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

Calculate filter response at the given normalized frequency

Parameters

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

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

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

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

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

Public Member Functions

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

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

Returns the coefficients of the entire [Biquad](#) chain

6.35.2.3 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)

```
6.35.2.4 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.
------------------------	--

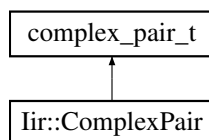
The documentation for this class was generated from the following file:

- 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:

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

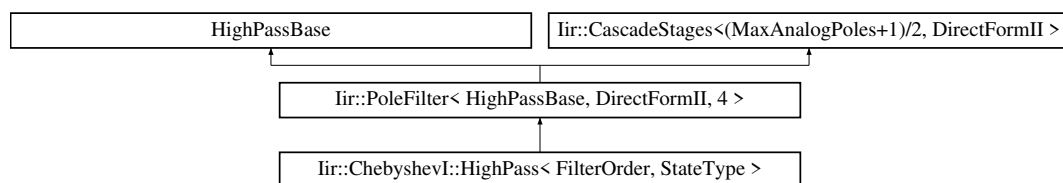
The documentation for this class was generated from the following file:

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

6.39.1 Detailed Description

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

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

[ChebyshevI](#) highpass filter

Parameters

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

6.39.2 Member Function Documentation

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

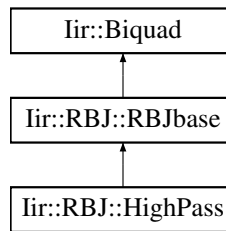
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

6.40 Iir::RBJ::HighPass Struct Reference

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

Inheritance diagram for Iir::RBJ::HighPass:



Public Member Functions

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

6.40.1 Detailed Description

Highpass.

6.40.2 Member Function Documentation

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

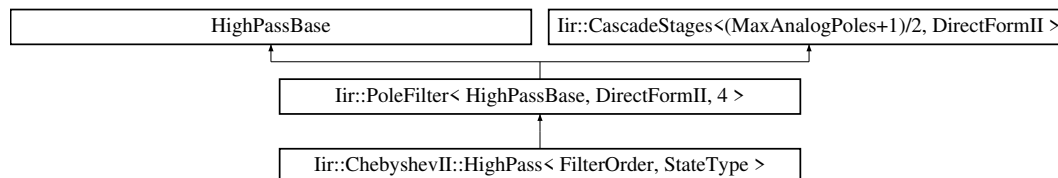
The documentation for this struct was generated from the following files:

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

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

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

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



Public Member Functions

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

6.41.1 Detailed Description

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

[ChebyshevII](#) highpass filter

Parameters

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

6.41.2 Member Function Documentation

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

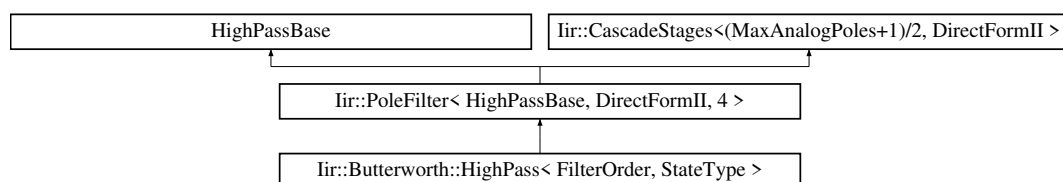
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

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

#include <Butterworth.h>

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



Public Member Functions

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

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

6.42.1 Detailed Description

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

[Butterworth](#) Highpass filter.

Parameters

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

6.42.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

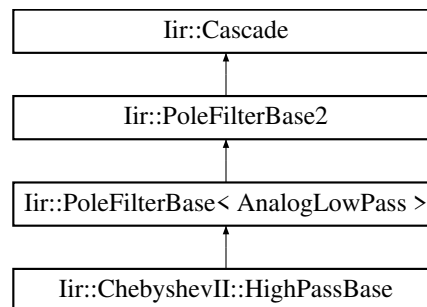
<i>reqOrder</i>	The actual order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)

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

- iir/Butterworth.h

6.43 Iir::ChebyshevII::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighPassBase:



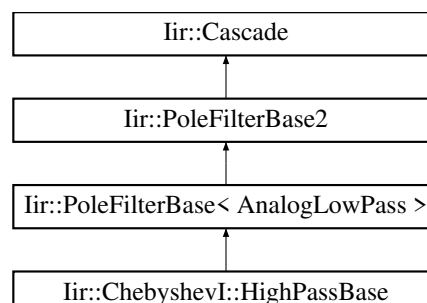
Additional Inherited Members

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

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

6.44 Iir::ChebyshevI::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighPassBase:



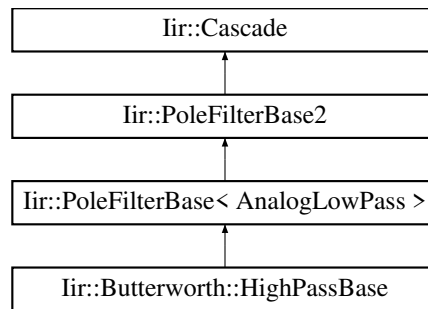
Additional Inherited Members

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

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

6.45 Iir::Butterworth::HighPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighPassBase:



Additional Inherited Members

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

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

6.46 Iir::HighPassTransform Class Reference

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

6.46.1 Detailed Description

low pass to high pass

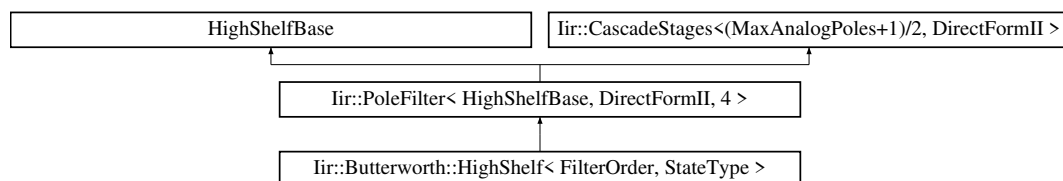
The documentation for this class was generated from the following files:

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

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

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

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



Public Member Functions

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

6.47.1 Detailed Description

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

[Butterworth](#) high shelf filter. Above the cutoff the filter has a specified gain and below it has 0 dB.

Parameters

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

6.47.2 Member Function Documentation

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

```

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

```

Calculates the coefficients

Parameters

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

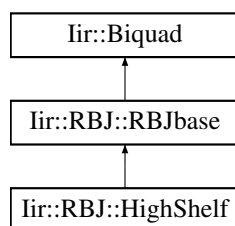
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.48 Iir::RBJ::HighShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::HighShelf:



Public Member Functions

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

6.48.1 Detailed Description

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

6.48.2 Member Function Documentation

```

6.48.2.1 setup() void Iir::RBJ::HighShelf::setup (
    double sampleRate,
    double cutoffFrequency,
    double gainDb,
    double shelfSlope = 1 ) [inline]

```

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

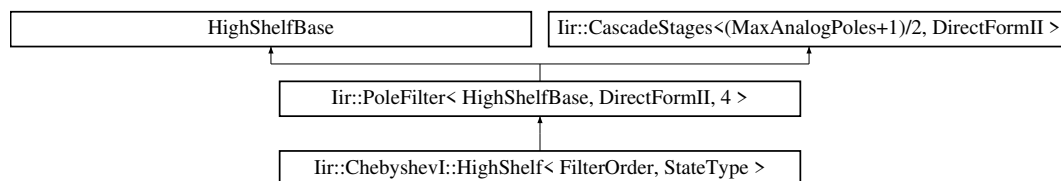
The documentation for this struct was generated from the following files:

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

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

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

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



Public Member Functions

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

6.49.1 Detailed Description

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

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

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

Parameters

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

6.49.2 Member Function Documentation

6.49.2.1 setup() [1/2] template<int FilterOrder = 4, class StateType = DirectFormII>
void Iir::ChebyshevI::HighShelf< FilterOrder, StateType >::setup (

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

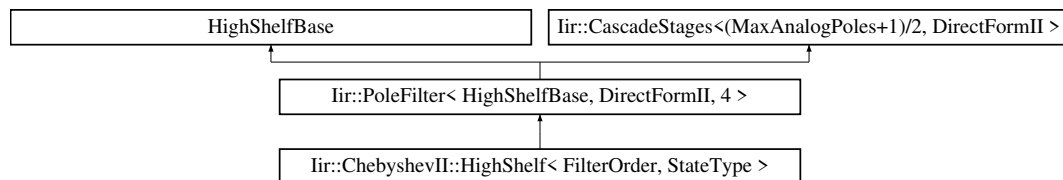
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

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

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



Public Member Functions

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

6.50.1 Detailed Description

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

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

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

Parameters

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

6.50.2 Member Function Documentation

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

Parameters

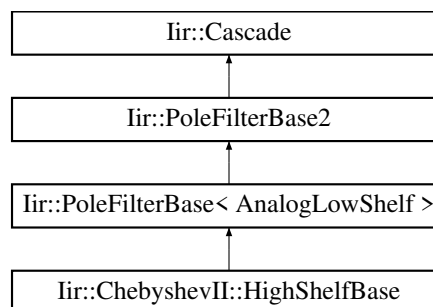
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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:



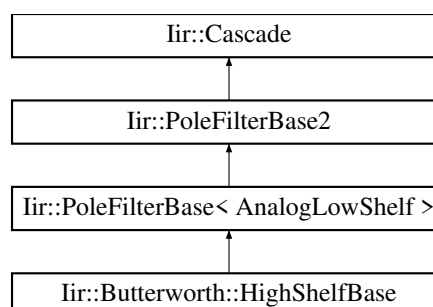
Additional Inherited Members

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:



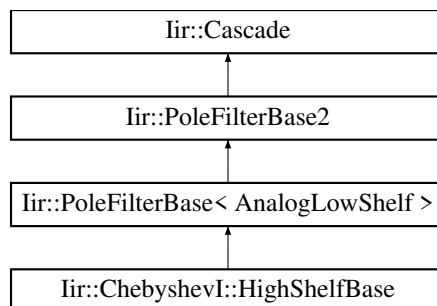
Additional Inherited Members

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

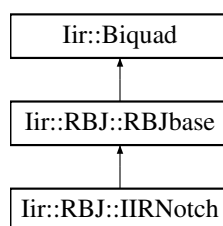
The documentation for this struct was generated from the following files:

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

6.54.1 Detailed Description

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

6.54.2 Member Function Documentation

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

Calculates the coefficients

Parameters

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

Parameters

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

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

Calculates the coefficients

Parameters

<i>centerFrequency</i>	Normalised centre frequency of the notch
<i>q_factor</i>	Q factor of the notch (1 to ~20)

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

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

6.55 `lir::Layout< MaxPoles >` Class Template Reference

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

6.55.1 Detailed Description

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

Storage for [Layout](#)

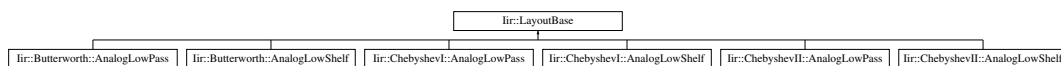
The documentation for this class was generated from the following file:

- `iir/Layout.h`

6.56 `lir::LayoutBase` Class Reference

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

Inheritance diagram for `lir::LayoutBase`:

**6.56.1 Detailed Description**

Base uses pointers to reduce template instantiations

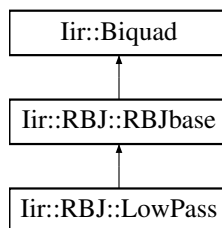
The documentation for this class was generated from the following file:

- `iir/Layout.h`

6.57 `lir::RBJ::LowPass` Struct Reference

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

Inheritance diagram for `lir::RBJ::LowPass`:



Public Member Functions

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

6.57.1 Detailed Description

Lowpass.

6.57.2 Member Function Documentation

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

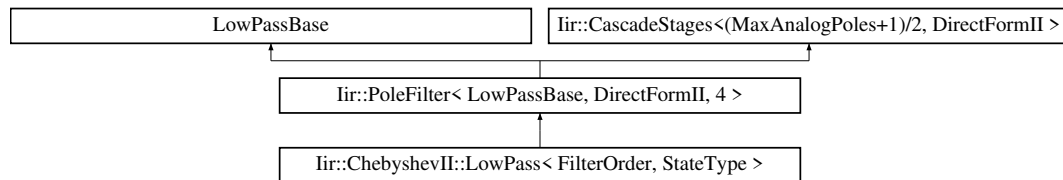
The documentation for this struct was generated from the following files:

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

6.58 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 [setupN](#) (double cutoffFrequency, double stopBandDb)
- void [setupN](#) (int reqOrder, double cutoffFrequency, double stopBandDb)

6.58.1 Detailed Description

```
template<int FilterOrder = 4, class StateType = DirectFormII>
struct 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: DirectFormI , DirectFormII , ...

6.58.2 Member Function Documentation

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

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

Calculates the coefficients of the filter

Parameters

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

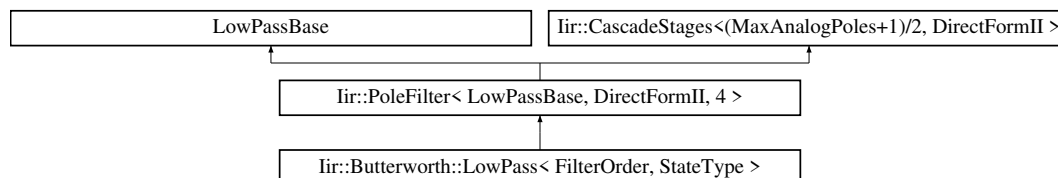
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

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

`#include <Butterworth.h>`

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



Public Member Functions

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

6.59.1 Detailed Description

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

[Butterworth](#) Lowpass filter.

Parameters

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

6.59.2 Member Function Documentation

6.59.2.1 `setup()` [1/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void lir::Butterworth::LowPass< FilterOrder, StateType >::setup (`
 `double sampleRate,`
 `double cutoffFrequency) [inline]`

Calculates the coefficients

Parameters

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

6.59.2.2 `setup()` [2/2] `template<int FilterOrder = 4, class StateType = DirectFormII>`
`void lir::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

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

Calculates the coefficients

Parameters

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

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


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

Calculates the coefficients

Parameters

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

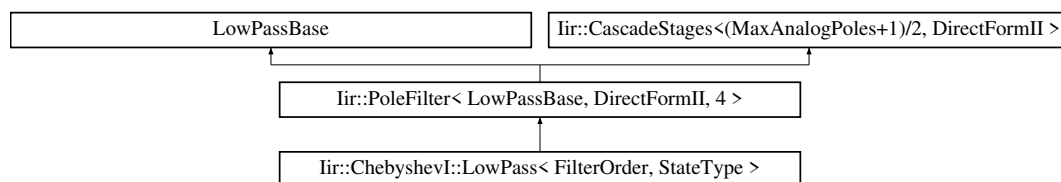
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

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

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

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



Public Member Functions

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

6.60.1 Detailed Description

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

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

[ChebyshevI](#) lowpass filter

Parameters

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

6.60.2 Member Function Documentation

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

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

Calculates the coefficients of the filter at the order FilterOrder

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

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

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

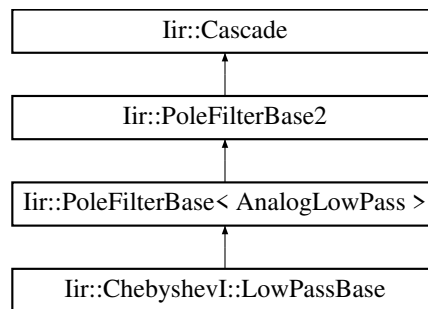
<i>reqOrder</i>	Actual order for the filter calculations
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>rippleDb</i>	Permitted ripples in dB in the passband

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

- `iir/ChebyshevI.h`

6.61 Iir::ChebyshevI::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowPassBase:



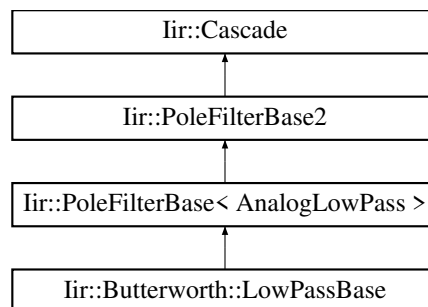
Additional Inherited Members

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

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

6.62 Iir::Butterworth::LowPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowPassBase:



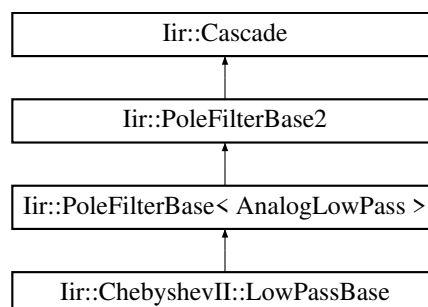
Additional Inherited Members

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

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

6.63 Iir::ChebyshevII::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowPassBase:



Additional Inherited Members

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

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

6.64 `lir::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

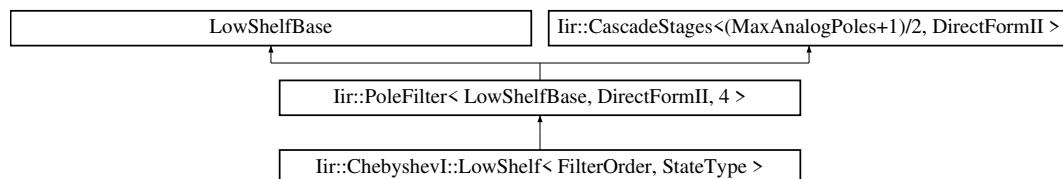
The documentation for this class was generated from the following files:

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

6.65.1 Detailed Description

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

```
struct 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: DirectFormI , DirectFormII , ...

6.65.2 Member Function Documentation

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

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

Parameters

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

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

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

Parameters

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

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

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

Calculates the coefficients of the filter at specified order

Parameters

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

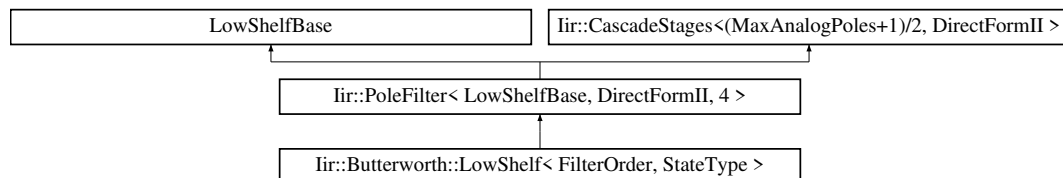
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

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

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



Public Member Functions

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

6.66.1 Detailed Description

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

[Butterworth](#) low shelf filter: below the cutoff it has a specified gain and above the cutoff the gain is 0 dB.

Parameters

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

6.66.2 Member Function Documentation

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

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

Calculates the coefficients with the filter order provided by the instantiation

Parameters

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

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

Calculates the coefficients

Parameters

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

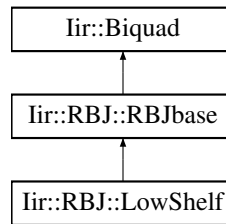
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

6.67 Iir::RBJ::LowShelf Struct Reference

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

Inheritance diagram for Iir::RBJ::LowShelf:



Public Member Functions

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

6.67.1 Detailed Description

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

6.67.2 Member Function Documentation

6.67.2.1 setup() void Iir::RBJ::LowShelf::setup (
 double *sampleRate*,
 double *cutoffFrequency*,
 double *gainDb*,
 double *shelfSlope* = 1) [inline]

Calculates the coefficients

Parameters

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

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

Calculates the coefficients

Parameters

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

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

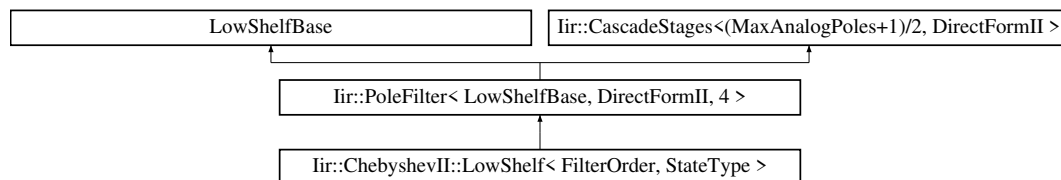
- iir/RBJ.h

- iir/RBJ.cpp

6.68 iir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference

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

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



Public Member Functions

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

6.68.1 Detailed Description

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

```
struct iir::ChebyshevII::LowShelf< FilterOrder, StateType >
```

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

Parameters

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

6.68.2 Member Function Documentation

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

Calculates the coefficients of the filter

Parameters

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

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

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

Calculates the coefficients of the filter

Parameters

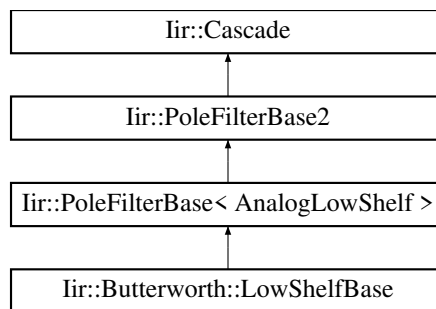
<i>reqOrder</i>	Requested order which can be less than the instantiated one
<i>cutoffFrequency</i>	Normalised cutoff frequency (0..1/2)
<i>gainDb</i>	Gain the passband. The stopband has 0 dB gain.
<i>stopBandDb</i>	Permitted ripples in dB in the stopband

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

- iir/ChebyshevII.h

6.69 Iir::Butterworth::LowShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowShelfBase:



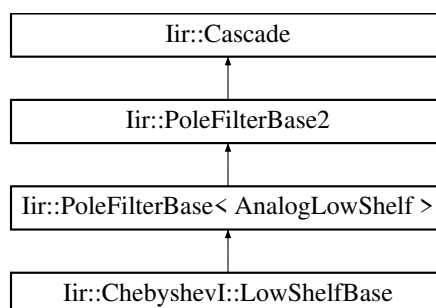
Additional Inherited Members

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

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

6.70 Iir::ChebyshevI::LowShelfBase Struct Reference

Inheritance diagram for `Iir::ChebyshevI::LowShelfBase`:



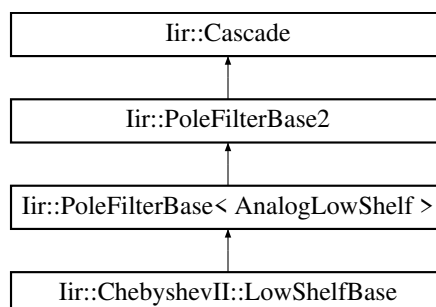
Additional Inherited Members

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

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

6.71 Iir::ChebyshevII::LowShelfBase Struct Reference

Inheritance diagram for `Iir::ChebyshevII::LowShelfBase`:



Additional Inherited Members

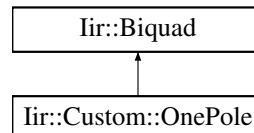
The documentation for this struct was generated from the following files:

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

6.72 `Iir::Custom::OnePole` Struct Reference

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

Inheritance diagram for `Iir::Custom::OnePole`:



Additional Inherited Members

6.72.1 Detailed Description

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

Parameters

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

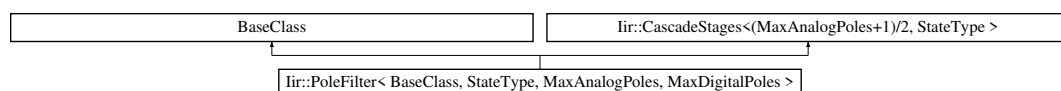
The documentation for this struct was generated from the following files:

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

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

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

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



Additional Inherited Members

6.73.1 Detailed Description

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

Storage for pole filters

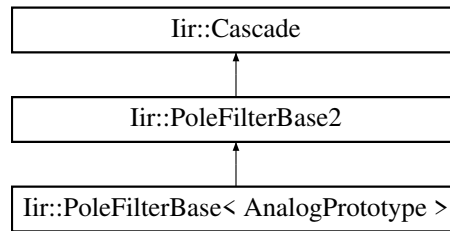
The documentation for this struct was generated from the following file:

- `iir/PoleFilter.h`

6.74 Iir::PoleFilterBase< AnalogPrototype > Class Template Reference

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

Inheritance diagram for Iir::PoleFilterBase< AnalogPrototype >:



Additional Inherited Members

6.74.1 Detailed Description

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

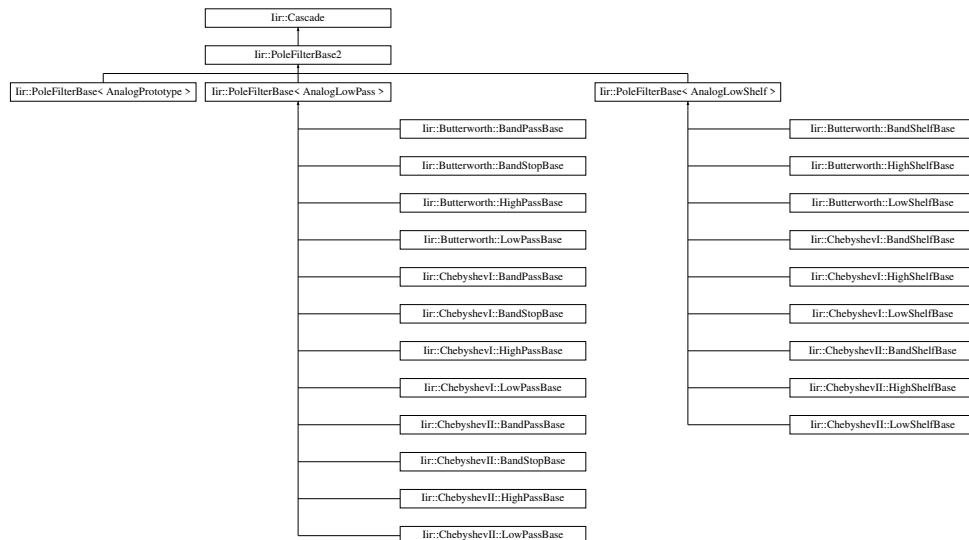
Serves a container to hold the analog prototype and the digital pole/zero layout.
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.75 Iir::PoleFilterBase2 Class Reference

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

Inheritance diagram for Iir::PoleFilterBase2:



Additional Inherited Members

6.75.1 Detailed Description

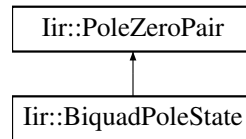
Factored implementations to reduce template instantiations
The documentation for this class was generated from the following file:

- iir/PoleFilter.h

6.76 Iir::PoleZeroPair Struct Reference

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

Inheritance diagram for Iir::PoleZeroPair:



6.76.1 Detailed Description

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

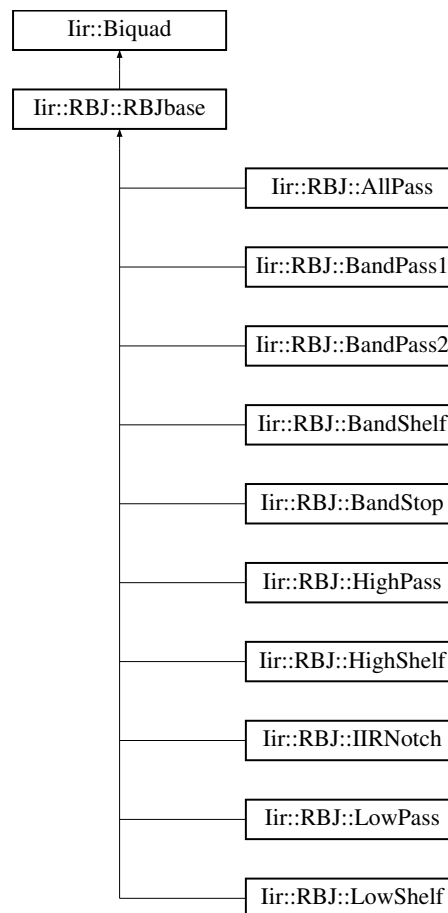
The documentation for this struct was generated from the following file:

- iir/Types.h

6.77 Iir::RBJ::RBJbase Struct Reference

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

Inheritance diagram for Iir::RBJ::RBJbase:



Public Member Functions

- template<typename Sample >
Sample [filter](#) (Sample s)
filter operation

- void [reset](#) ()
resets the delay lines to zero
- const [DirectFormI](#) & [getState](#) ()
gets the delay lines (=state) of the filter

6.77.1 Detailed Description

The base class of all RBJ filters

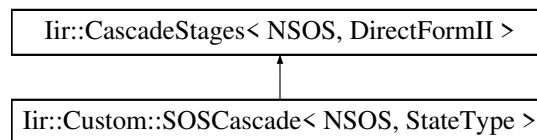
The documentation for this struct was generated from the following file:

- iir/RBJ.h

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

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

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



Public Member Functions

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

6.78.1 Detailed Description

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

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

Parameters

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

6.78.2 Constructor & Destructor Documentation

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

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

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

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

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

Python scipy.signal-friendly setting of coefficients. Initialises the coefficients of the whole chain of biquads / SOS. The argument is a 2D array where the 1st dimension holds an array of 2nd order biquad / SOS coefficients. The six

SOS coefficients are ordered "Python" style with first the FIR coefficients (B) and then the IIR coefficients (A). The 2D const double array needs to have exactly the size [NSOS][6].

Parameters

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

6.78.3 Member Function Documentation

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

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

Parameters

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

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

- iir/Custom.h

6.79 Iir::Cascade::Storage Struct Reference

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

Public Member Functions

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

6.79.1 Detailed Description

Pointer to an array of Biquads

6.79.2 Constructor & Destructor Documentation

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

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

Parameters

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

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

- iir/Cascade.h

6.80 Iir::TransposedDirectFormII Class Reference

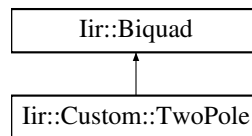
The documentation for this class was generated from the following file:

- iir/State.h

6.81 Iir::Custom::TwoPole Struct Reference

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

Inheritance diagram for Iir::Custom::TwoPole:



Additional Inherited Members

6.81.1 Detailed Description

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

Parameters

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

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

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

Index

applyScale
 lir::Biquad, [41](#)

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, [42](#)

getCascadeStorage
 lir::CascadeStages< MaxStages, StateType >, [45](#)

getNumStages
 lir::Cascade, [44](#)

getPoleZeros
 lir::Biquad, [42](#)
 lir::Cascade, [44](#)

lir, [8](#)
 Kind, [11](#)

lir::BandPassTransform, [24](#)

lir::BandStopTransform, [40](#)

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

lir::BiquadPoleState, [43](#)

lir::Butterworth, [11](#)

lir::Butterworth::AnalogLowPass, [13](#)

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

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

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

lir::Butterworth::BandShelf< FilterOrder, StateType >, [24](#)
 setup, [25](#)
 setupN, [25](#), [26](#)

lir::Butterworth::BandShelfBase, [31](#)

lir::Butterworth::BandStop< FilterOrder, StateType >, [37](#)
 setup, [38](#)
 setupN, [38](#), [39](#)

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

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

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

lir::Butterworth::HighShelf< FilterOrder, StateType >, [54](#)
 setup, [55](#)
 setupN, [55](#)

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

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

lir::Butterworth::LowPassBase, [70](#)

lir::Butterworth::LowShelf< FilterOrder, StateType >, [73](#)
 setup, [73](#), [74](#)
 setupN, [74](#)

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

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

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

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

lir::ChebyshevI, [11](#)

lir::ChebyshevI::AnalogLowPass, [14](#)

lir::ChebyshevI::AnalogLowShelf, [15](#)

lir::ChebyshevI::BandPass< FilterOrder, StateType >, [15](#)
 setup, [16](#)
 setupN, [16](#), [17](#)

lir::ChebyshevI::BandPassBase, [23](#)

lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [27](#)
 setup, [28](#)
 setupN, [28](#), [29](#)

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

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

- setup, [33](#), [34](#)
- setupN, [34](#)
- lir::ChebyshevI::BandStopBase, [40](#)
- lir::ChebyshevI::HighPass< FilterOrder, StateType >, [47](#)
 - setup, [47](#), [48](#)
 - setupN, [48](#)
- lir::ChebyshevI::HighPassBase, [53](#)
- lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [57](#)
 - setup, [57](#), [58](#)
 - setupN, [58](#)
- lir::ChebyshevI::HighShelfBase, [61](#)
- lir::ChebyshevI::LowPass< FilterOrder, StateType >, [68](#)
 - setup, [68](#), [69](#)
 - setupN, [69](#)
- lir::ChebyshevI::LowPassBase, [70](#)
- lir::ChebyshevI::LowShelf< FilterOrder, StateType >, [71](#)
 - setup, [71](#), [72](#)
 - setupN, [72](#)
- lir::ChebyshevI::LowShelfBase, [78](#)
- lir::ChebyshevII, [12](#)
- lir::ChebyshevII::AnalogLowPass, [14](#)
- lir::ChebyshevII::AnalogLowShelf, [14](#)
- lir::ChebyshevII::BandPass< FilterOrder, StateType >, [19](#)
 - setup, [19](#), [20](#)
 - setupN, [20](#)
- lir::ChebyshevII::BandPassBase, [23](#)
- lir::ChebyshevII::BandShelf< FilterOrder, StateType >, [29](#)
 - setup, [30](#)
 - setupN, [31](#)
- lir::ChebyshevII::BandShelfBase, [32](#)
- lir::ChebyshevII::BandStop< FilterOrder, StateType >, [36](#)
 - setup, [36](#)
 - setupN, [37](#)
- lir::ChebyshevII::BandStopBase, [40](#)
- lir::ChebyshevII::HighPass< FilterOrder, StateType >, [50](#)
 - setup, [50](#)
 - setupN, [51](#)
- lir::ChebyshevII::HighPassBase, [53](#)
- lir::ChebyshevII::HighShelf< FilterOrder, StateType >, [59](#)
 - setup, [59](#), [60](#)
 - setupN, [60](#)
- lir::ChebyshevII::HighShelfBase, [61](#)
- lir::ChebyshevII::LowPass< FilterOrder, StateType >, [64](#)
 - setup, [65](#)
 - setupN, [66](#)
- lir::ChebyshevII::LowPassBase, [70](#)
- lir::ChebyshevII::LowShelf< FilterOrder, StateType >, [76](#)
 - setup, [76](#)
- setupN, [77](#)
- lir::ChebyshevII::LowShelfBase, [78](#)
- lir::ComplexPair, [46](#)
 - isMatchedPair, [46](#)
- lir::Custom, [12](#)
- lir::Custom::OnePole, [79](#)
- lir::Custom::SOSCascade< NSOS, StateType >, [82](#)
 - setup, [83](#)
 - SOSCascade, [82](#)
- lir::Custom::TwoPole, [84](#)
- lir::DirectFormI, [46](#)
- lir::DirectFormII, [47](#)
- lir::HighPassTransform, [54](#)
- lir::Layout< MaxPoles >, [63](#)
- lir::LayoutBase, [63](#)
- lir::LowPassTransform, [71](#)
- lir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >, [79](#)
- lir::PoleFilterBase< AnalogPrototype >, [80](#)
- lir::PoleFilterBase2, [80](#)
- lir::PoleZeroPair, [81](#)
- lir::RBJ::AllPass, [12](#)
 - setup, [13](#)
 - setupN, [13](#)
- lir::RBJ::BandPass1, [21](#)
 - setup, [21](#)
 - setupN, [21](#)
- lir::RBJ::BandPass2, [22](#)
 - setup, [22](#)
 - setupN, [22](#)
- lir::RBJ::BandShelf, [26](#)
 - setup, [26](#)
 - setupN, [27](#)
- lir::RBJ::BandStop, [35](#)
 - setup, [35](#)
 - setupN, [35](#)
- lir::RBJ::HighPass, [49](#)
 - setup, [49](#)
 - setupN, [49](#)
- lir::RBJ::HighShelf, [56](#)
 - setup, [56](#)
 - setupN, [57](#)
- lir::RBJ::IIRNotch, [62](#)
 - setup, [62](#)
 - setupN, [63](#)
- lir::RBJ::LowPass, [63](#)
 - setup, [64](#)
 - setupN, [64](#)
- lir::RBJ::LowShelf, [75](#)
 - setup, [75](#)
 - setupN, [75](#)
- lir::RBJ::RBJbase, [81](#)
- lir::TransposedDirectFormII, [84](#)
- isMatchedPair
 - lir::ComplexPair, [46](#)
- Kind
 - lir, [11](#)

- operator[]
 - lir::Cascade, [44](#)
- reset
 - lir::CascadeStages< MaxStages, StateType >, [45](#)
- response
 - lir::Biquad, [42](#)
 - lir::Cascade, [44](#)
- 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 >, [18](#)
 - lir::Butterworth::BandShelf< FilterOrder, StateType >, [25](#)
 - lir::Butterworth::BandStop< FilterOrder, StateType >, [38](#)
 - lir::Butterworth::HighPass< FilterOrder, StateType >, [52](#)
 - lir::Butterworth::HighShelf< FilterOrder, StateType >, [55](#)
 - lir::Butterworth::LowPass< FilterOrder, StateType >, [67](#)
 - lir::Butterworth::LowShelf< FilterOrder, StateType >, [73](#), [74](#)
 - lir::CascadeStages< MaxStages, StateType >, [46](#)
 - lir::ChebyshevI::BandPass< FilterOrder, StateType >, [16](#)
 - lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [28](#)
 - lir::ChebyshevI::BandStop< FilterOrder, StateType >, [33](#), [34](#)
 - lir::ChebyshevI::HighPass< FilterOrder, StateType >, [47](#), [48](#)
 - lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [57](#), [58](#)
 - lir::ChebyshevI::LowPass< FilterOrder, StateType >, [68](#), [69](#)
 - lir::ChebyshevI::LowShelf< FilterOrder, StateType >, [71](#), [72](#)
 - lir::ChebyshevII::BandPass< FilterOrder, StateType >, [19](#), [20](#)
 - lir::ChebyshevII::BandShelf< FilterOrder, StateType >, [30](#)
 - lir::ChebyshevII::BandStop< FilterOrder, StateType >, [36](#)
 - lir::ChebyshevII::HighPass< FilterOrder, StateType >, [50](#)
 - lir::ChebyshevII::HighShelf< FilterOrder, StateType >, [59](#), [60](#)
 - lir::ChebyshevII::LowPass< FilterOrder, StateType >, [65](#)
 - lir::ChebyshevII::LowShelf< FilterOrder, StateType >, [76](#)
 - lir::Custom::SOSCascade< NSOS, StateType >, [83](#)
 - lir::RBJ::AllPass, [13](#)
 - lir::RBJ::BandPass1, [21](#)
 - lir::RBJ::BandPass2, [22](#)
 - lir::RBJ::BandShelf, [26](#)
 - lir::RBJ::BandStop, [35](#)
 - lir::RBJ::HighPass, [49](#)
 - lir::RBJ::HighShelf, [56](#)
 - lir::RBJ::IIRNotch, [62](#)
 - lir::RBJ::LowPass, [64](#)
 - lir::RBJ::LowShelf, [75](#)
- setupN
 - lir::Butterworth::BandPass< FilterOrder, StateType >, [18](#)
 - lir::Butterworth::BandShelf< FilterOrder, StateType >, [25](#), [26](#)
 - lir::Butterworth::BandStop< FilterOrder, StateType >, [38](#), [39](#)
 - lir::Butterworth::HighPass< FilterOrder, StateType >, [52](#), [53](#)
 - lir::Butterworth::HighShelf< FilterOrder, StateType >, [55](#)
 - lir::Butterworth::LowPass< FilterOrder, StateType >, [67](#)
 - lir::Butterworth::LowShelf< FilterOrder, StateType >, [74](#)
 - lir::ChebyshevI::BandPass< FilterOrder, StateType >, [16](#), [17](#)
 - lir::ChebyshevI::BandShelf< FilterOrder, StateType >, [28](#), [29](#)
 - lir::ChebyshevI::BandStop< FilterOrder, StateType >, [34](#)
 - lir::ChebyshevI::HighPass< FilterOrder, StateType >, [48](#)
 - lir::ChebyshevI::HighShelf< FilterOrder, StateType >, [58](#)
 - lir::ChebyshevI::LowPass< FilterOrder, StateType >, [69](#)
 - lir::ChebyshevI::LowShelf< FilterOrder, StateType >, [72](#)
 - lir::ChebyshevII::BandPass< FilterOrder, StateType >, [20](#)
 - lir::ChebyshevII::BandShelf< FilterOrder, StateType >, [31](#)
 - lir::ChebyshevII::BandStop< FilterOrder, StateType >, [37](#)
 - lir::ChebyshevII::HighPass< FilterOrder, StateType >, [51](#)
 - lir::ChebyshevII::HighShelf< FilterOrder, StateType >, [60](#)
 - lir::ChebyshevII::LowPass< FilterOrder, StateType >, [66](#)
 - lir::ChebyshevII::LowShelf< FilterOrder, StateType >

>, [77](#)
lir::RBJ::AllPass, [13](#)
lir::RBJ::BandPass1, [21](#)
lir::RBJ::BandPass2, [22](#)
lir::RBJ::BandShelf, [27](#)
lir::RBJ::BandStop, [35](#)
lir::RBJ::HighPass, [49](#)
lir::RBJ::HighShelf, [57](#)
lir::RBJ::IIRNotch, [63](#)
lir::RBJ::LowPass, [64](#)
lir::RBJ::LowShelf, [75](#)
SOSCascade
lir::Custom::SOSCascade< NSOS, StateType >,
[82](#)
Storage
lir::Cascade::Storage, [83](#)