iir1

Generated on Wed Oct 21 2026 23:01:47 for iir1 by Doxygen 1.15.0

Wed Oct 21 2026 23:01:47

1 DSP IIR Realtime C++ filter library 1

2 Namespace Index 1

2.1 Namespace List . . . 1

3 Hierarchical Index 1

3.1 Class Hierarchy . . . 1

4 Class Index 5

4.1 Class List . . . 5

5 File Index 8

5.1 File List . . . 8

6 Namespace Documentation 8

6.1 Iir Namespace Reference . . . 8

6.1.1 Detailed Description . . . 9

6.2 Iir::Butterworth Namespace Reference . . . 11

6.2.1 Detailed Description . . . 11

6.3 Iir::ChebyshevI Namespace Reference . . . 11

6.3.1 Detailed Description . . . 11

6.4 Iir::ChebyshevII Namespace Reference . . . 12

6.4.1 Detailed Description . . . 12

6.5 Iir::Custom Namespace Reference . . . 12

6.5.1 Detailed Description . . . 12

7 Class Documentation 12

7.1 Iir::RBJ::AllPass Struct Reference . . . 12

7.1.1 Detailed Description . . . 13

7.1.2 Member Function Documentation . . . 13

7.2 Iir::Butterworth::AnalogLowPass Class Reference . . . 14

7.2.1 Detailed Description . . . 14

7.3 Iir::ChebyshevI::AnalogLowPass Class Reference . . . 14

7.3.1 Detailed Description . . . 14

7.4 Iir::ChebyshevII::AnalogLowPass Class Reference . . . 14

7.4.1 Detailed Description . . . 15

7.5 Iir::Butterworth::AnalogLowShelf Class Reference . . . 15

7.5.1 Detailed Description . . . 15

7.6 Iir::ChebyshevI::AnalogLowShelf Class Reference . . . 15

7.6.1 Detailed Description . . . 15

7.7 Iir::ChebyshevII::AnalogLowShelf Class Reference . . . 15

7.7.1 Detailed Description . . . 16

7.8 Iir::Butterworth::BandPass< FilterOrder, StateType > Struct Template Reference . . . 16

7.8.1 Detailed Description . . . 16

7.8.2 Member Function Documentation . . . 16
7.9 Iir::ChebyshevI::BandPass< FilterOrder, StateType > Struct Template Reference . . . 18
7.9.1 Detailed Description . . . 18
7.9.2 Member Function Documentation . . . 18
7.10 Iir::ChebyshevII::BandPass< FilterOrder, StateType > Struct Template Reference . . . 20
7.10.1 Detailed Description . . . 20
7.10.2 Member Function Documentation . . . 20
7.11 Iir::RBJ::BandPass1 Struct Reference . . . 22
7.11.1 Detailed Description . . . 23
7.11.2 Member Function Documentation . . . 23
7.12 Iir::RBJ::BandPass2 Struct Reference . . . 23
7.12.1 Detailed Description . . . 24
7.12.2 Member Function Documentation . . . 24
7.13 Iir::Butterworth::BandPassBase Struct Reference . . . 25
7.14 Iir::ChebyshevI::BandPassBase Struct Reference . . . 25
7.15 Iir::ChebyshevII::BandPassBase Struct Reference . . . 26
7.16 Iir::BandPassTransform Class Reference . . . 26
7.16.1 Detailed Description . . . 27
7.17 Iir::Butterworth::BandShelf< FilterOrder, StateType > Struct Template Reference . . . 27
7.17.1 Detailed Description . . . 27
7.17.2 Member Function Documentation . . . 27
7.18 Iir::ChebyshevI::BandShelf< FilterOrder, StateType > Struct Template Reference . . . 29
7.18.1 Detailed Description . . . 29
7.18.2 Member Function Documentation . . . 30
7.19 Iir::ChebyshevII::BandShelf< FilterOrder, StateType > Struct Template Reference . . . 31
7.19.1 Detailed Description . . . 32
7.19.2 Member Function Documentation . . . 32
7.20 Iir::RBJ::BandShelf Struct Reference . . . 33
7.20.1 Detailed Description . . . 34
7.20.2 Member Function Documentation . . . 34
7.21 Iir::Butterworth::BandShelfBase Struct Reference . . . 35
7.22 Iir::ChebyshevI::BandShelfBase Struct Reference . . . 36
7.23 Iir::ChebyshevII::BandShelfBase Struct Reference . . . 36
7.24 Iir::Butterworth::BandStop< FilterOrder, StateType > Struct Template Reference . . . 37
7.24.1 Detailed Description . . . 37
7.24.2 Member Function Documentation . . . 37
7.25 Iir::ChebyshevI::BandStop< FilterOrder, StateType > Struct Template Reference . . . 38
7.25.1 Detailed Description . . . 39
7.25.2 Member Function Documentation . . . 39
7.26 Iir::ChebyshevII::BandStop< FilterOrder, StateType > Struct Template Reference . . . 41
7.26.1 Detailed Description . . . 41
7.26.2 Member Function Documentation . . . 41

7.27 Iir::RBJ::BandStop Struct Reference . . . 43
7.27.1 Detailed Description . . . 44
7.27.2 Member Function Documentation . . . 44
7.28 Iir::Butterworth::BandStopBase Struct Reference . . . 44
7.29 Iir::ChebyshevI::BandStopBase Struct Reference . . . 45
7.30 Iir::ChebyshevII::BandStopBase Struct Reference . . . 45
7.31 Iir::BandStopTransform Class Reference . . . 46
7.31.1 Detailed Description . . . 46
7.32 Iir::Biquad Class Reference . . . 46
7.32.1 Member Function Documentation . . . 47
7.33 Iir::BiquadPoleState Struct Reference . . . 49
7.33.1 Detailed Description . . . 49
7.34 Iir::Cascade Class Reference . . . 49
7.34.1 Detailed Description . . . 50
7.34.2 Member Function Documentation . . . 50
7.35 Iir::CascadeStages< MaxStages, StateType > Class Template Reference . . . 51
7.35.1 Detailed Description . . . 51
7.35.2 Member Function Documentation . . . 51
7.36 Iir::ComplexPair Struct Reference . . . 52
7.36.1 Detailed Description . . . 52
7.36.2 Member Function Documentation . . . 52
7.37 Iir::DirectFormI Class Reference . . . 52
7.37.1 Detailed Description . . . 52
7.38 Iir::DirectFormII Class Reference . . . 53
7.38.1 Detailed Description . . . 53
7.39 Iir::Butterworth::HighPass< FilterOrder, StateType > Struct Template Reference . . . 53
7.39.1 Detailed Description . . . 53
7.39.2 Member Function Documentation . . . 54
7.40 Iir::ChebyshevI::HighPass< FilterOrder, StateType > Struct Template Reference . . . 55
7.40.1 Detailed Description . . . 55
7.40.2 Member Function Documentation . . . 55
7.41 Iir::ChebyshevII::HighPass< FilterOrder, StateType > Struct Template Reference . . . 57
7.41.1 Detailed Description . . . 57
7.41.2 Member Function Documentation . . . 57
7.42 Iir::RBJ::HighPass Struct Reference . . . 59
7.42.1 Detailed Description . . . 59
7.42.2 Member Function Documentation . . . 60
7.43 Iir::Butterworth::HighPassBase Struct Reference . . . 60
7.44 Iir::ChebyshevI::HighPassBase Struct Reference . . . 61
7.45 Iir::ChebyshevII::HighPassBase Struct Reference . . . 61
7.46 Iir::HighPassTransform Class Reference . . . 62
7.46.1 Detailed Description . . . 62

7.47 Iir::Butterworth::HighShelf< FilterOrder, StateType > Struct Template Reference 62
7.47.1 Detailed Description 62
7.47.2 Member Function Documentation 62
7.48 Iir::ChebyshevI::HighShelf< FilterOrder, StateType > Struct Template Reference 64
7.48.1 Detailed Description 64
7.48.2 Member Function Documentation 64
7.49 Iir::ChebyshevII::HighShelf< FilterOrder, StateType > Struct Template Reference 66
7.49.1 Detailed Description 66
7.49.2 Member Function Documentation 67
7.50 Iir::RBJ::HighShelf Struct Reference 68
7.50.1 Detailed Description 69
7.50.2 Member Function Documentation 69
7.51 Iir::Butterworth::HighShelfBase Struct Reference 70
7.52 Iir::ChebyshevI::HighShelfBase Struct Reference 70
7.53 Iir::ChebyshevII::HighShelfBase Struct Reference 71
7.54 Iir::RBJ::IIRNotch Struct Reference 71
7.54.1 Detailed Description 72
7.54.2 Member Function Documentation 72
7.55 Iir::Layout< MaxPoles > Class Template Reference 73
7.55.1 Detailed Description 73
7.56 Iir::LayoutBase Class Reference 73
7.56.1 Detailed Description 73
7.57 Iir::Butterworth::LowPass< FilterOrder, StateType > Struct Template Reference 73
7.57.1 Detailed Description 74
7.57.2 Member Function Documentation 74
7.58 Iir::ChebyshevI::LowPass< FilterOrder, StateType > Struct Template Reference 75
7.58.1 Detailed Description 75
7.58.2 Member Function Documentation 76
7.59 Iir::ChebyshevII::LowPass< FilterOrder, StateType > Struct Template Reference 77
7.59.1 Detailed Description 77
7.59.2 Member Function Documentation 78
7.60 Iir::RBJ::LowPass Struct Reference 79
7.60.1 Detailed Description 80
7.60.2 Member Function Documentation 80
7.61 Iir::Butterworth::LowPassBase Struct Reference 80
7.62 Iir::ChebyshevI::LowPassBase Struct Reference 81
7.63 Iir::ChebyshevII::LowPassBase Struct Reference 81
7.64 Iir::LowPassTransform Class Reference 82
7.64.1 Detailed Description 82
7.65 Iir::Butterworth::LowShelf< FilterOrder, StateType > Struct Template Reference 82
7.65.1 Detailed Description 83
7.65.2 Member Function Documentation 83

7.66 Iir::ChebyshevI::LowShelf< FilterOrder, StateType > Struct Template Reference 84
7.66.1 Detailed Description 85
7.66.2 Member Function Documentation 85
7.67 Iir::ChebyshevII::LowShelf< FilterOrder, StateType > Struct Template Reference 86
7.67.1 Detailed Description 87
7.67.2 Member Function Documentation 87
7.68 Iir::RBJ::LowShelf Struct Reference 88
7.68.1 Detailed Description 89
7.68.2 Member Function Documentation 89
7.69 Iir::Butterworth::LowShelfBase Struct Reference 90
7.70 Iir::ChebyshevI::LowShelfBase Struct Reference 90
7.71 Iir::ChebyshevII::LowShelfBase Struct Reference 91
7.72 Iir::Custom::OnePole Struct Reference 91
7.72.1 Detailed Description 92
7.73 Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > Struct Template Reference 92
7.73.1 Detailed Description 93
7.74 Iir::PoleFilterBase< AnalogPrototype > Class Template Reference 93
7.74.1 Detailed Description 93
7.75 Iir::PoleFilterBase2 Class Reference 93
7.75.1 Detailed Description 94
7.76 Iir::PoleZeroPair Struct Reference 94
7.76.1 Detailed Description 94
7.77 Iir::RBJ::RBJbase Struct Reference 95
7.77.1 Detailed Description 96
7.78 Iir::Custom::SOSCascade< NSOS, StateType > Struct Template Reference 96
7.78.1 Detailed Description 96
7.78.2 Constructor & Destructor Documentation 97
7.78.3 Member Function Documentation 97
7.79 Iir::Cascade::Storage Struct Reference 98
7.79.1 Detailed Description 98
7.80 Iir::TransposedDirectFormII Class Reference 98
7.81 Iir::Custom::TwoPole Struct Reference 98
7.81.1 Detailed Description 99

8 File Documentation **99**

8.1 Iir.h 99
8.2 Biquad.h 99
8.3 Butterworth.h 100
8.4 Cascade.h 105
8.5 ChebyshevI.h 106
8.6 ChebyshevII.h 111
8.7 Common.h 117

8.8 Custom.h . . . 117
8.9 Layout.h . . . 118
8.10 MathSupplement.h . . . 120
8.11 PoleFilter.h . . . 120
8.12 RBJ.h . . . 123
8.13 State.h . . . 124
8.14 Types.h . . . 126

Index **129**

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:

Iir **8**

Iir::Butterworth **11**

Iir::ChebyshevI **11**

Iir::ChebyshevII **12**

Iir::Custom **12**

3 Hierarchical Index

3.1 Class Hierarchy

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

BandPassBase

Iir::PoleFilter< BandPassBase, DirectFormII, 4, 4 *2 > 92

Iir::Butterworth::BandPass< FilterOrder, StateType > 16

Iir::ChebyshevI::BandPass< FilterOrder, StateType > 18

Iir::ChebyshevII::BandPass< FilterOrder, StateType > 20

Iir::BandPassTransform 26
BandShelfBase

Iir::PoleFilter< BandShelfBase, DirectFormII, 4, 4 *2 > 92

Iir::Butterworth::BandShelf< FilterOrder, StateType > 27

Iir::ChebyshevI::BandShelf< FilterOrder, StateType > 29

Iir::ChebyshevII::BandShelf< FilterOrder, StateType > 31
BandStopBase

Iir::PoleFilter< BandStopBase, DirectFormII, 4, 4 *2 > 92

Iir::Butterworth::BandStop< FilterOrder, StateType > 37

Iir::ChebyshevI::BandStop< FilterOrder, StateType > 38

Iir::ChebyshevII::BandStop< FilterOrder, StateType > 41

Iir::BandStopTransform 46
BaseClass

Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > 92

Iir::Biquad 46

Iir::Custom::OnePole 91

Iir::Custom::TwoPole 98

Iir::RBJ::RBJbase 95

Iir::RBJ::AllPass 12

Iir::RBJ::BandPass1 22

Iir::RBJ::BandPass2 23

Iir::RBJ::BandShelf 33

Iir::RBJ::BandStop 43

Iir::RBJ::HighPass 59

Iir::RBJ::HighShelf 68

Iir::RBJ::IIRNotch 71

Iir::RBJ::LowPass 79

Iir::RBJ::LowShelf 88

Iir::Cascade 49

- **Iir::PoleFilterBase2** 93
  - **Iir::PoleFilterBase< AnalogLowPass >** 93
    - **Iir::Butterworth::BandPassBase** 25
    - **Iir::Butterworth::BandStopBase** 44
    - **Iir::Butterworth::HighPassBase** 60
    - **Iir::Butterworth::LowPassBase** 80
    - **Iir::ChebyshevI::BandPassBase** 25
    - **Iir::ChebyshevI::BandStopBase** 45
    - **Iir::ChebyshevI::HighPassBase** 61
    - **Iir::ChebyshevI::LowPassBase** 81
    - **Iir::ChebyshevII::BandPassBase** 26
    - **Iir::ChebyshevII::BandStopBase** 45
    - **Iir::ChebyshevII::HighPassBase** 61
    - **Iir::ChebyshevII::LowPassBase** 81
  - **Iir::PoleFilterBase< AnalogLowShelf >** 93
    - **Iir::Butterworth::BandShelfBase** 35
    - **Iir::Butterworth::HighShelfBase** 70
    - **Iir::Butterworth::LowShelfBase** 90
    - **Iir::ChebyshevI::BandShelfBase** 36
    - **Iir::ChebyshevI::HighShelfBase** 70
    - **Iir::ChebyshevI::LowShelfBase** 90
    - **Iir::ChebyshevII::BandShelfBase** 36
    - **Iir::ChebyshevII::HighShelfBase** 71
    - **Iir::ChebyshevII::LowShelfBase** 91
  - **Iir::PoleFilterBase< AnalogPrototype >** 93
- **Iir::CascadeStages< MaxStages, StateType >** 51
- **Iir::CascadeStages< NSOS, DirectFormII >** 51
  - **Iir::Custom::SOSCascade< NSOS, StateType >** 96
- **Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >** 51
  - **Iir::PoleFilter< HighPassBase, DirectFormII, 4 >** 92
    - **Iir::Butterworth::HighPass< FilterOrder, StateType >** 53
    - **Iir::ChebyshevI::HighPass< FilterOrder, StateType >** 55

Iir::ChebyshevII::HighPass< FilterOrder, StateType > 57

Iir::PoleFilter< HighShelfBase, DirectFormII, 4 > 92

Iir::Butterworth::HighShelf< FilterOrder, StateType > 62

Iir::ChebyshevI::HighShelf< FilterOrder, StateType > 64

Iir::ChebyshevII::HighShelf< FilterOrder, StateType > 66

Iir::PoleFilter< LowPassBase, DirectFormII, 4 > 92

Iir::Butterworth::LowPass< FilterOrder, StateType > 73

Iir::ChebyshevI::LowPass< FilterOrder, StateType > 75

Iir::ChebyshevII::LowPass< FilterOrder, StateType > 77

Iir::PoleFilter< LowShelfBase, DirectFormII, 4 > 92

Iir::Butterworth::LowShelf< FilterOrder, StateType > 82

Iir::ChebyshevI::LowShelf< FilterOrder, StateType > 84

Iir::ChebyshevII::LowShelf< FilterOrder, StateType > 86

Iir::CascadeStages<(MaxAnalogPoles+1)/2, StateType > 51

Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > 92

Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII > 51

Iir::PoleFilter< BandPassBase, DirectFormII, 4, 4 ∗2 > 92

Iir::PoleFilter< BandShelfBase, DirectFormII, 4, 4 ∗2 > 92

Iir::PoleFilter< BandStopBase, DirectFormII, 4, 4 ∗2 > 92

complex_pair_t

Iir::ComplexPair 52

Iir::DirectFormI 52

Iir::DirectFormII 53

HighPassBase

Iir::PoleFilter< HighPassBase, DirectFormII, 4 > 92

Iir::HighPassTransform 62

HighShelfBase

Iir::PoleFilter< HighShelfBase, DirectFormII, 4 > 92

Iir::Layout< MaxPoles > 73

Iir::LayoutBase 73

Iir::Butterworth::AnalogLowPass 14

Iir::Butterworth::AnalogLowShelf 15

Iir::ChebyshevI::AnalogLowPass 14

Iir::ChebyshevI::AnalogLowShelf 15

Iir::ChebyshevII::AnalogLowPass 14

Iir::ChebyshevII::AnalogLowShelf 15

LowPassBase

Iir::PoleFilter< LowPassBase, DirectFormII, 4 > 92

Iir::LowPassTransform 82

LowShelfBase

Iir::PoleFilter< LowShelfBase, DirectFormII, 4 > 92

Iir::PoleZeroPair 94

Iir::BiquadPoleState 49

Iir::Cascade::Storage 98

Iir::TransposedDirectFormII 98

4 Class Index

4.1 Class List

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

Iir::RBJ::AllPass 12

Iir::Butterworth::AnalogLowPass 14

Iir::ChebyshevI::AnalogLowPass 14

Iir::ChebyshevII::AnalogLowPass 14

Iir::Butterworth::AnalogLowShelf 15

Iir::ChebyshevI::AnalogLowShelf 15

Iir::ChebyshevII::AnalogLowShelf 15

Iir::Butterworth::BandPass< FilterOrder, StateType > 16

Iir::ChebyshevI::BandPass< FilterOrder, StateType > 18

Iir::ChebyshevII::BandPass< FilterOrder, StateType > 20

Iir::RBJ::BandPass1 22

Iir::RBJ::BandPass2 23

Iir::Butterworth::BandPassBase 25

Iir::ChebyshevI::BandPassBase 25

Iir::ChebyshevII::BandPassBase 26

Iir::BandPassTransform 26

Iir::Butterworth::BandShelf< FilterOrder, StateType > 27

Iir::ChebyshevI::BandShelf< FilterOrder, StateType > 29

Iir::ChebyshevII::BandShelf< FilterOrder, StateType > 31

Iir::RBJ::BandShelf 33

Iir::Butterworth::BandShelfBase 35

Iir::ChebyshevI::BandShelfBase 36

Iir::ChebyshevII::BandShelfBase 36

Iir::Butterworth::BandStop< FilterOrder, StateType > 37

Iir::ChebyshevI::BandStop< FilterOrder, StateType > 38

Iir::ChebyshevII::BandStop< FilterOrder, StateType > 41

Iir::RBJ::BandStop 43

Iir::Butterworth::BandStopBase 44

Iir::ChebyshevI::BandStopBase 45

Iir::ChebyshevII::BandStopBase 45

Iir::BandStopTransform 46

Iir::Biquad 46

Iir::BiquadPoleState 49

Iir::Cascade 49

Iir::CascadeStages< MaxStages, StateType > 51

Iir::ComplexPair 52

Iir::DirectFormI 52

Iir::DirectFormII 53

Iir::Butterworth::HighPass< FilterOrder, StateType > 53

Iir::ChebyshevI::HighPass< FilterOrder, StateType > 55

Iir::ChebyshevII::HighPass< FilterOrder, StateType > 57

Iir::RBJ::HighPass 59

Iir::Butterworth::HighPassBase 60

Iir::ChebyshevI::HighPassBase 61

Iir::ChebyshevII::HighPassBase 61

Iir::HighPassTransform 62

Iir::Butterworth::HighShelf< FilterOrder, StateType > 62

Iir::ChebyshevI::HighShelf< FilterOrder, StateType > 64

Iir::ChebyshevII::HighShelf< FilterOrder, StateType > 66

Iir::RBJ::HighShelf 68

Iir::Butterworth::HighShelfBase 70

Iir::ChebyshevI::HighShelfBase 70

Iir::ChebyshevII::HighShelfBase 71

Iir::RBJ::IIRNotch 71

Iir::Layout< MaxPoles > 73

Iir::LayoutBase 73

Iir::Butterworth::LowPass< FilterOrder, StateType > 73

Iir::ChebyshevI::LowPass< FilterOrder, StateType > 75

Iir::ChebyshevII::LowPass< FilterOrder, StateType > 77

Iir::RBJ::LowPass 79

Iir::Butterworth::LowPassBase 80

Iir::ChebyshevI::LowPassBase 81

Iir::ChebyshevII::LowPassBase 81

Iir::LowPassTransform 82

Iir::Butterworth::LowShelf< FilterOrder, StateType > 82

Iir::ChebyshevI::LowShelf< FilterOrder, StateType > 84

Iir::ChebyshevII::LowShelf< FilterOrder, StateType > 86

Iir::RBJ::LowShelf 88

Iir::Butterworth::LowShelfBase 90

Iir::ChebyshevI::LowShelfBase 90

Iir::ChebyshevII::LowShelfBase 91

Iir::Custom::OnePole 91

Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles > 92

Iir::PoleFilterBase< AnalogPrototype > 93

Iir::PoleFilterBase2 93

Iir::PoleZeroPair 94

Iir::RBJ::RBJbase 95

Iir::Custom::SOSCascade< NSOS, StateType > 96

Iir::Cascade::Storage 98

Iir::TransposedDirectFormII 98

Iir::Custom::TwoPole **98**

5 File Index

5.1 File List

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

Iir.h **99**

iir/Biquad.h **99**

iir/Butterworth.h **100**

iir/Cascade.h **105**

iir/ChebyshevI.h **106**

iir/ChebyshevII.h **111**

iir/Common.h **117**

iir/Custom.h **117**

iir/Layout.h **118**

iir/MathSupplement.h **120**

iir/PoleFilter.h **120**

iir/RBJ.h **123**

iir/State.h **124**

iir/Types.h **126**

6 Namespace Documentation

6.1 Iir Namespace Reference

Namespaces

- namespace Butterworth
- namespace ChebyshevI
- namespace ChebyshevII
- namespace Custom

Classes

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

6.1.1 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.

6.2 Iir::Butterworth Namespace Reference

Classes

- class AnalogLowPass
- class AnalogLowShelf
- struct LowPassBase
- struct HighPassBase
- struct BandPassBase
- struct BandStopBase
- struct LowShelfBase
- struct HighShelfBase
- struct BandShelfBase
- struct LowPass
- struct HighPass
- struct BandPass
- struct BandStop
- struct LowShelf
- struct HighShelf
- struct BandShelf

6.2.1 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 provde the filter order at setup time to force a lower order than the default one.

6.3 Iir::ChebyshevI Namespace Reference

Classes

- class AnalogLowPass
- class AnalogLowShelf
- struct LowPassBase
- struct HighPassBase
- struct BandPassBase
- struct BandStopBase
- struct LowShelfBase
- struct HighShelfBase
- struct BandShelfBase
- struct LowPass
- struct HighPass
- struct BandPass
- struct BandStop
- struct LowShelf
- struct HighShelf
- struct BandShelf

6.3.1 Detailed Description

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

6.4 Iir::ChebyshevII Namespace Reference

Classes

- class AnalogLowPass
- class AnalogLowShelf
- struct LowPassBase
- struct HighPassBase
- struct BandPassBase
- struct BandStopBase
- struct LowShelfBase
- struct HighShelfBase
- struct BandShelfBase
- struct LowPass
- struct HighPass
- struct BandPass
- struct BandStop
- struct LowShelf
- struct HighShelf
- struct BandShelf

6.4.1 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.

6.5 Iir::Custom Namespace Reference

Classes

- struct OnePole
- struct TwoPole
- struct SOSCascade

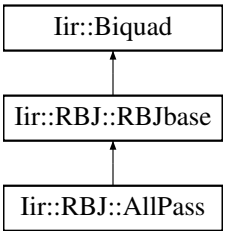
6.5.1 Detailed Description

Single pole, Biquad and cascade of Biquads with parameters allowing for directly setting the parameters.

7 Class Documentation

7.1 Iir::RBJ::AllPass Struct Reference

`#include <RBJ.h>`
Inheritance diagram for Iir::RBJ::AllPass:



Public Member Functions

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

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.1.1 Detailed Description

Allpass filter

7.1.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
phaseFrequency	Frequency where the phase flips
q	Q-factor

setupN()

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

Calculates the coefficients

Parameters

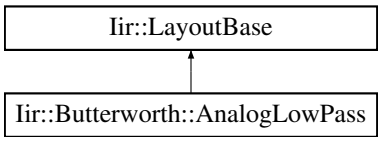
phaseFrequency	Normalised frequency where the phase flips
q	Q-factor

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

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

7.2 Iir::Butterworth::AnalogLowPass Class Reference

`#include <Butterworth.h>`
Inheritance diagram for Iir::Butterworth::AnalogLowPass:



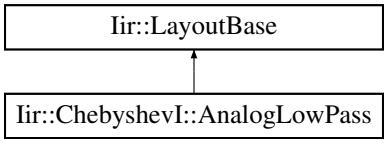
7.2.1 Detailed Description

Analogue lowpass prototypes (s-plane)
The documentation for this class was generated from the following files:

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

7.3 Iir::ChebyshevI::AnalogLowPass Class Reference

`#include <ChebyshevI.h>`
Inheritance diagram for Iir::ChebyshevI::AnalogLowPass:



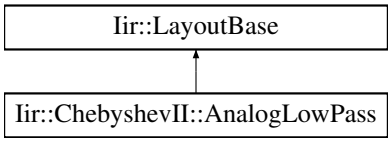
7.3.1 Detailed Description

Analog lowpass prototypes (s-plane)
The documentation for this class was generated from the following files:

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

7.4 Iir::ChebyshevII::AnalogLowPass Class Reference

`#include <ChebyshevII.h>`
Inheritance diagram for Iir::ChebyshevII::AnalogLowPass:



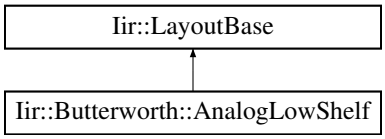
7.4.1 Detailed Description

Analogue lowpass prototype (s-plane)
The documentation for this class was generated from the following files:

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

7.5 Iir::Butterworth::AnalogLowShelf Class Reference

`#include <Butterworth.h>`
Inheritance diagram for Iir::Butterworth::AnalogLowShelf:



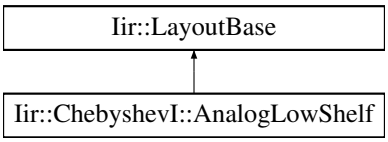
7.5.1 Detailed Description

Analogue low shelf prototypes (s-plane)
The documentation for this class was generated from the following files:

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

7.6 Iir::ChebyshevI::AnalogLowShelf Class Reference

`#include <ChebyshevI.h>`
Inheritance diagram for Iir::ChebyshevI::AnalogLowShelf:



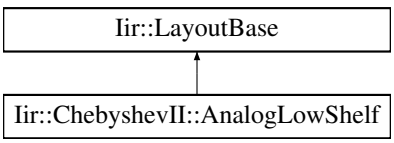
7.6.1 Detailed Description

Analog lowpass shelf prototype (s-plane)
The documentation for this class was generated from the following files:

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

7.7 Iir::ChebyshevII::AnalogLowShelf Class Reference

`#include <ChebyshevII.h>`
Inheritance diagram for Iir::ChebyshevII::AnalogLowShelf:



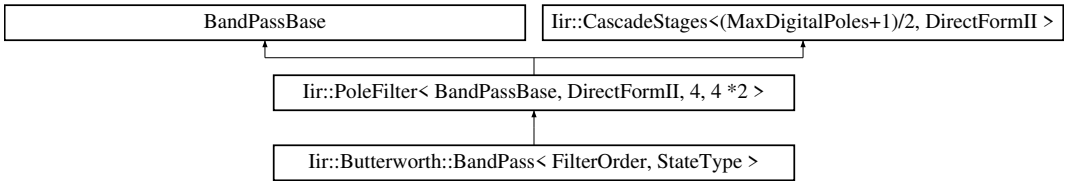
7.7.1 Detailed Description

Analogue shelf lowpass prototype (s-plane)
The documentation for this class was generated from the following files:

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

7.8 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.8.1 Detailed Description

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

Butterworth Bandpass filter.

Parameters

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

7.8.2 Member Function Documentation

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

sampleRate	Sampling rate
centerFrequency	Centre frequency of the bandpass
widthFrequency	Width of the bandpass

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

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

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

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

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

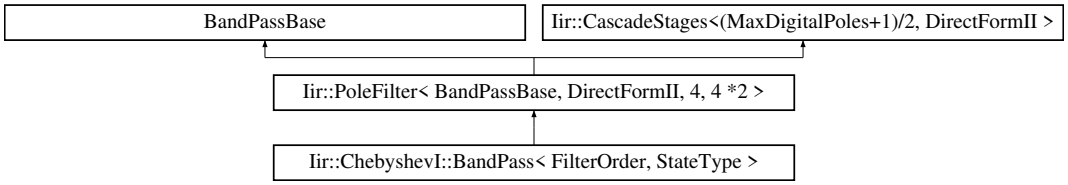
reqOrder	The actual order which can be less than the instantiated one
centerFrequency	Normalised centre frequency (0..1/2) of the bandpass
widthFrequency	Width of the bandpass in normalised freq

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

- iir/Butterworth.h

7.9 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.9.1 Detailed Description

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

ChebyshevI bandpass filter

Parameters

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

7.9.2 Member Function Documentation

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

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

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

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

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

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

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

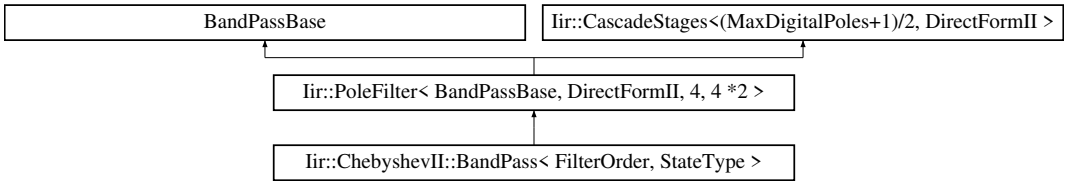
reqOrder	Actual order for the filter calculations
centerFrequency	Normalised center frequency (0..1/2) of the bandpass
widthFrequency	Frequency with of the passband
rippleDb	Permitted ripples in dB in the passband

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

- iir/ChebyshevI.h

7.10 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 stop↩ BandDb)
- void setupN (double centerFrequency, double widthFrequency, double stopBandDb)
- void setupN (int reqOrder, double centerFrequency, double widthFrequency, double stopBandDb)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.10.1 Detailed Description

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

ChebyshevII bandpass filter

Parameters

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

7.10.2 Member Function Documentation

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

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

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

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

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

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

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

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

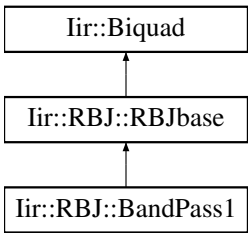
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

7.11 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.11.1 Detailed Description

Bandpass with constant skirt gain

7.11.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
centerFrequency	Center frequency of the bandpass
bandWidth	Bandwidth in octaves

setupN()

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

Calculates the coefficients

Parameters

centerFrequency	Center frequency of the bandpass
bandWidth	Bandwidth in octaves

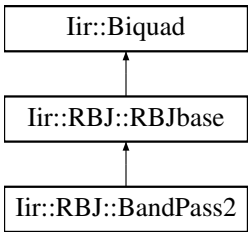
The documentation for this struct was generated from the following files:

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

7.12 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.12.1 Detailed Description

Bandpass with constant 0 dB peak gain

7.12.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
centerFrequency	Center frequency of the bandpass
bandWidth	Bandwidth in octaves

setupN()

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

Calculates the coefficients

Parameters

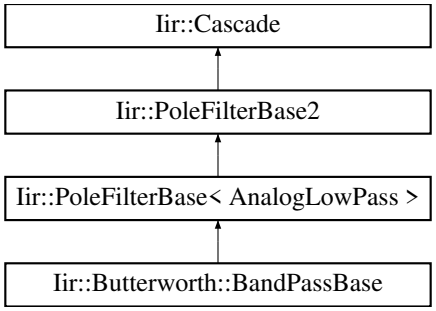
centerFrequency	Normalised centre frequency of the bandpass
bandWidth	Bandwidth in octaves

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

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

7.13 Iir::Butterworth::BandPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

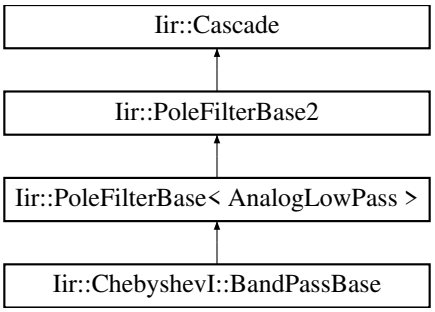
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.14 Iir::ChebyshevI::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

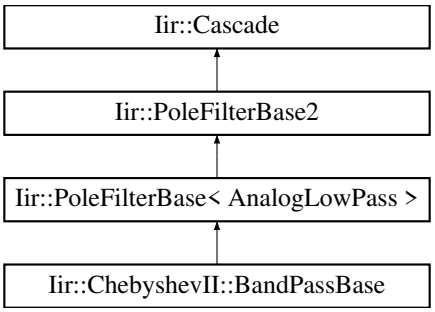
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.15 Iir::ChebyshevII::BandPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

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

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

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

7.16 Iir::BandPassTransform Class Reference

`#include <PoleFilter.h>`

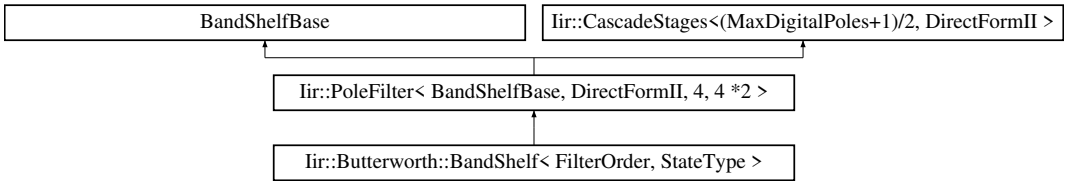
7.16.1 Detailed Description

low pass to band pass transform
The documentation for this class was generated from the following files:

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

7.17 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.17.1 Detailed Description

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

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

Parameters

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

7.17.2 Member Function Documentation

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

sampleRate	Sampling rate
centerFrequency	Centre frequency of the passband
widthFrequency	Width of the passband
gainDb	The gain in the passband

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

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

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

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

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

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

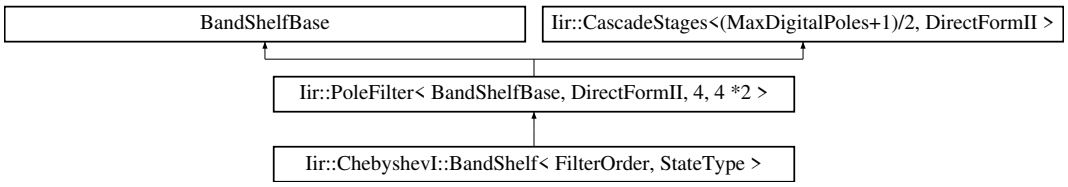
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

7.18 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.18.1 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

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

7.18.2 Member Function Documentation

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

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

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

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

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

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

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

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

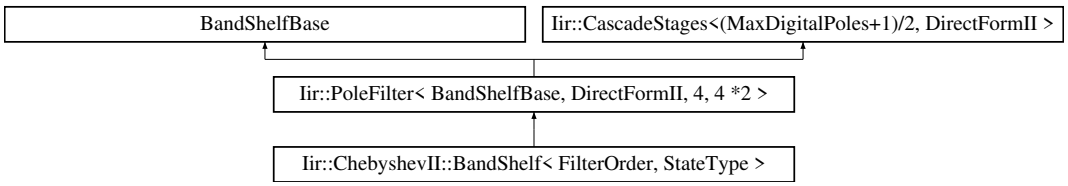
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

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

`#include <ChebyshevII.h>`

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



Public Member Functions

- void setup (double sampleRate, double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void setup (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency, double gain↩ Db, double stopBandDb)
- void setupN (double centerFrequency, double widthFrequency, double gainDb, double stopBandDb)
- void setupN (int reqOrder, double centerFrequency, double widthFrequency, double gainDb, double stop↩ BandDb)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.19.1 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

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

7.19.2 Member Function Documentation

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

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

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

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

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

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

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

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

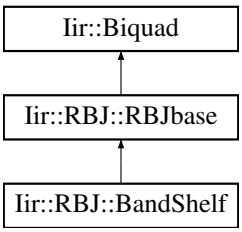
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

7.20 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.20.1 Detailed Description

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

7.20.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
centerFrequency	frequency
gainDb	Gain in the passband
bandWidth	Bandwidth in octaves

setupN()

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

Calculates the coefficients

Parameters

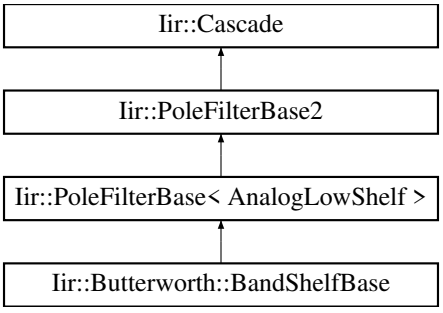
centerFrequency	Normalised centre frequency
gainDb	Gain in the passband
bandWidth	Bandwidth in octaves

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

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

7.21 Iir::Butterworth::BandShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

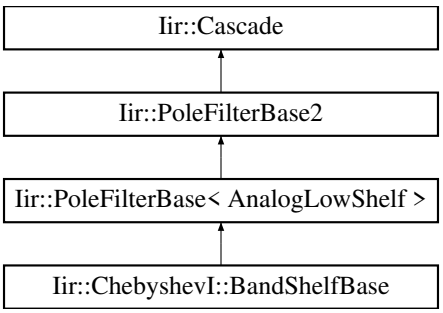
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.22 Iir::ChebyshevI::BandShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

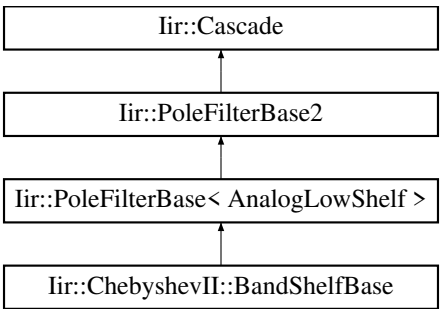
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.23 Iir::ChebyshevII::BandShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

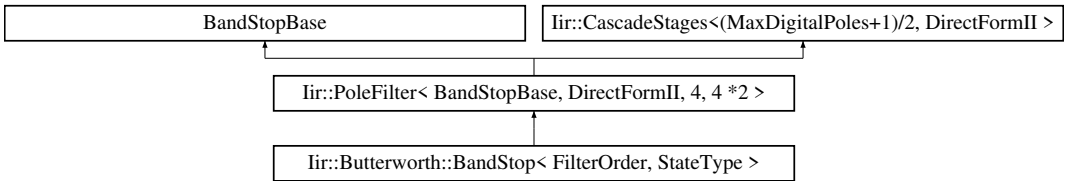
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.24 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 setup (int reqOrder, double sampleRate, double centerFrequency, double widthFrequency)
- void setupN (double centerFrequency, double widthFrequency)
- void setupN (int reqOrder, double centerFrequency, double widthFrequency)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.24.1 Detailed Description

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

Butterworth Bandstop filter.

Parameters

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

7.24.2 Member Function Documentation

setup() [1/2]

```
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

sampleRate	Sampling rate
centerFrequency	Centre frequency of the bandstop
widthFrequency	Width of the bandstop

setup() [2/2]

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

Calculates the coefficients

Parameters

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

setupN() [1/2]

```
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

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

setupN() [2/2]

```
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

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

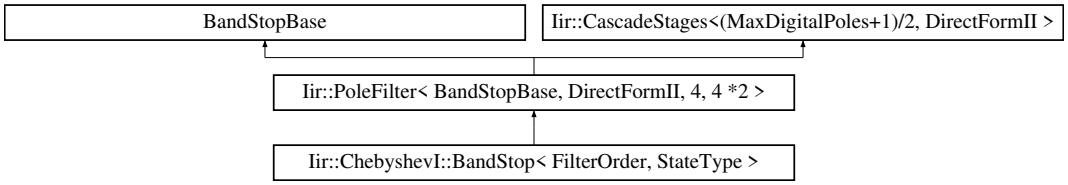
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

7.25 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.25.1 Detailed Description

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

ChebyshevI bandstop filter

Parameters

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

7.25.2 Member Function Documentation

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

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

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

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

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

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

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

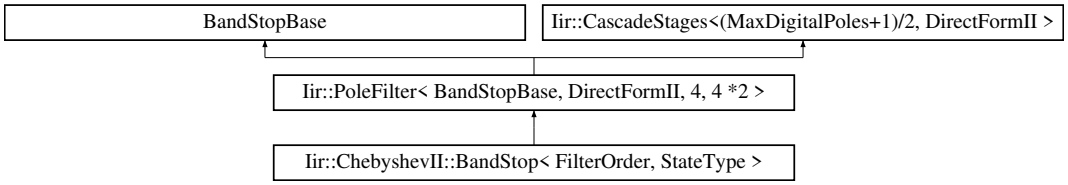
reqOrder	Actual order for the filter calculations
centerFrequency	Normalised centre frequency (0..1/2) of the notch
widthFrequency	Frequency width of the notch
rippleDb	Permitted ripples in dB in the passband

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

- iir/ChebyshevI.h

7.26 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxDigitalPoles+1)/2, DirectFormII >

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

7.26.1 Detailed Description

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

ChebyshevII bandstop filter.

Parameters

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

7.26.2 Member Function Documentation

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

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

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

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

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

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

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

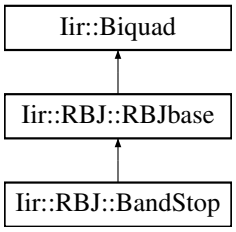
reqOrder	Requested order which can be less than the instantiated one
centerFrequency	Normalised centre frequency (0..1/2) of the bandstop
widthFrequency	Width of the bandstop
stopBandDb	Permitted ripples in dB in the stopband

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

- iir/ChebyshevII.h

7.27 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.27.1 Detailed Description

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

7.27.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
centerFrequency	Center frequency of the bandstop
bandWidth	Bandwidth in octaves

setupN()

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

Calculates the coefficients

Parameters

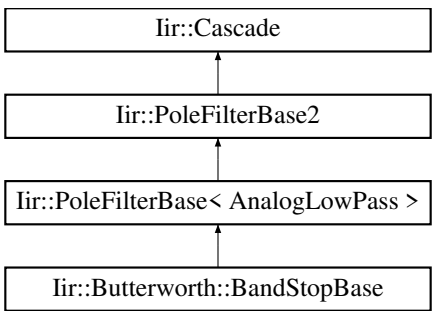
centerFrequency	Normalised Centre frequency of the bandstop
bandWidth	Bandwidth in octaves

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

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

7.28 Iir::Butterworth::BandStopBase Struct Reference

Inheritance diagram for Iir::Butterworth::BandStopBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

- int getNumStages () const

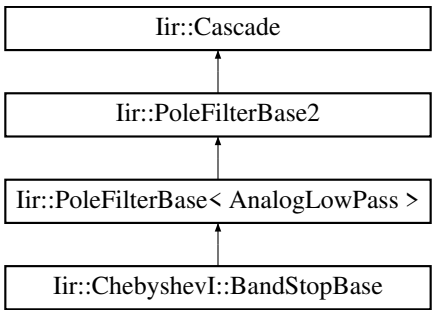
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.29 Iir::ChebyshevI::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::BandStopBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

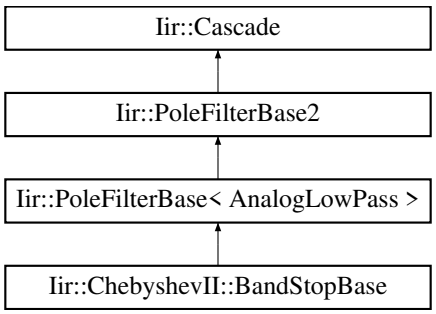
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.30 Iir::ChebyshevII::BandStopBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::BandStopBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

- int getNumStages () const
- const Biquad & operator[ ] (int index)

- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.31 Iir::BandStopTransform Class Reference

`#include <PoleFilter.h>`

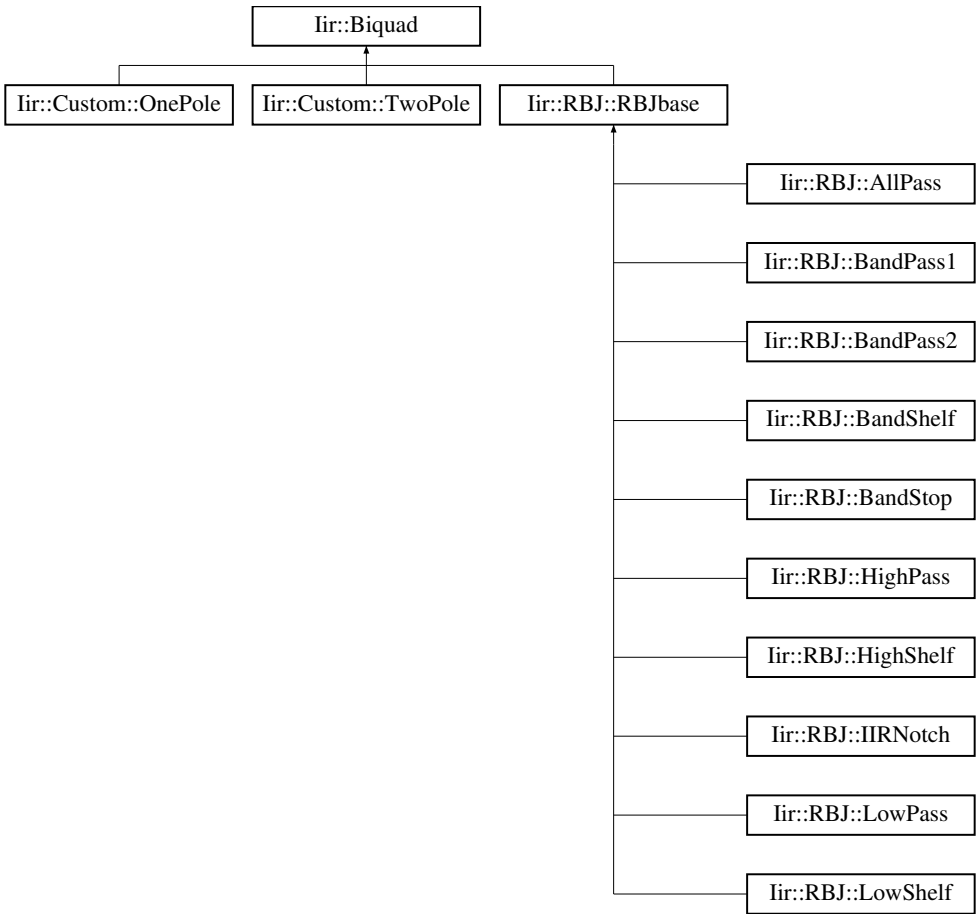
7.31.1 Detailed Description

low pass to band stop transform
The documentation for this class was generated from the following files:

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

7.32 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)

7.32.1 Member Function Documentation

applyScale()

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

Performs scaling operation on the FIR coefficients

Parameters

scale	Mulitplies the coefficients b0,b1,b2 with the scaling factor scale.

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

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

Returns

The filtered sample.

getA0()

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

Returns 1st IIR coefficient (usually one)

getA1()

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

Returns 2nd IIR coefficient

getA2()

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

getB0()

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

getB1()

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

getB2()

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

getPoleZeros()

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

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

normalizedFrequency	Normalised frequency (0 to 0.5)

setCoefficients()

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

Parameters

a0	1st IIR coefficient
a1	2nd IIR coefficient
a2	3rd IIR coefficient
b0	1st FIR coefficient
b1	2nd FIR coefficient
b2	3rd FIR coefficient

setIdentity()

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

setOnePole()

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

setPoleZeroPair()

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

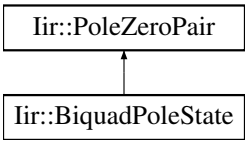
setTwoPole()

```
void Iir::Biquad::setTwoPole (
            complex_t pole1,
            complex_t zero1,
            complex_t pole2,
            complex_t zero2)
```
Sets two poles/zoes as a pair. Needs to be complex conjugate.
The documentation for this class was generated from the following files:

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

7.33 Iir::BiquadPoleState Struct Reference

`#include <Biquad.h>`
Inheritance diagram for Iir::BiquadPoleState:



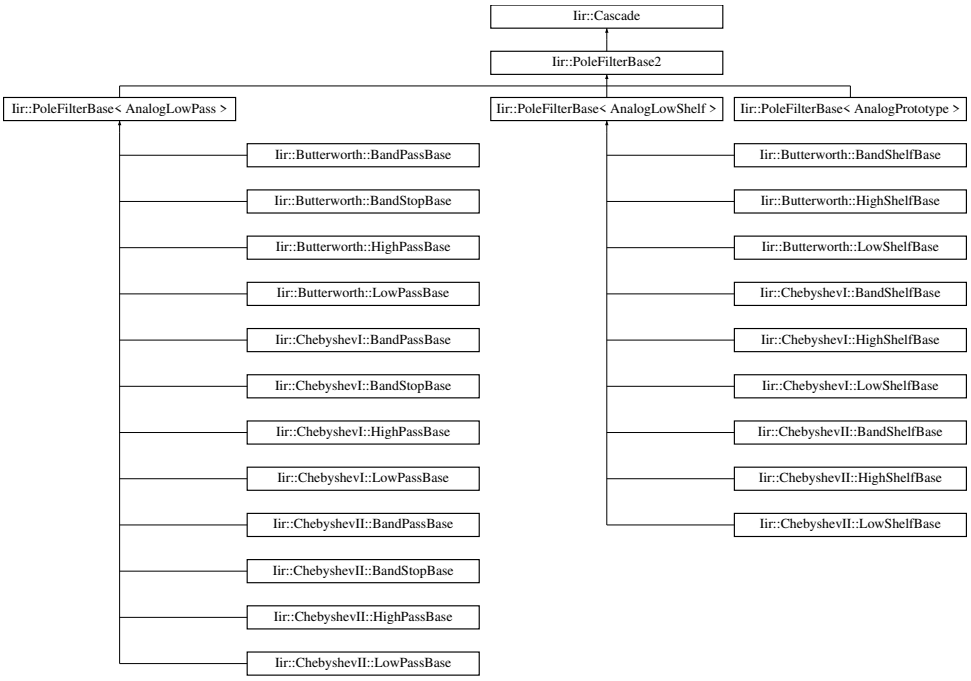
7.33.1 Detailed Description

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

7.34 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

7.34.1 Detailed Description

Holds coefficients for a cascade of second order sections.

7.34.2 Member Function Documentation

getNumStages()

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

Returns the number of Biquads kept here

getPoleZeros()

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

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

operator[]()

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

Returns a reference to a biquad

response()

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

Calculate filter response at the given normalized frequency

Parameters

normalizedFrequency	Frequency from 0 to 0.5 (Nyquist)

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

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

7.35 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 ()

7.35.1 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.

7.35.2 Member Function Documentation

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

in	Sample to be filtered

Returns

filtered sample

getCascadeStorage()

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

Returns the coefficients of the entire Biquad chain

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)

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

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

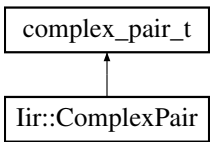
The documentation for this class was generated from the following file:

- iir/Cascade.h

7.36 Iir::ComplexPair Struct Reference

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

Inheritance diagram for Iir::ComplexPair:



Public Member Functions

- bool isMatchedPair () const

7.36.1 Detailed Description

A conjugate or real pair

7.36.2 Member Function Documentation

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

7.37 Iir::DirectFormI Class Reference

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

7.37.1 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

7.38 Iir::DirectFormII Class Reference

`#include <State.h>`

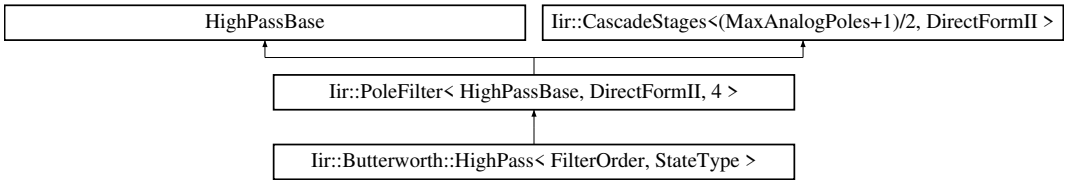
7.38.1 Detailed Description

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] 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

7.39 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.39.1 Detailed Description

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

Butterworth Highpass filter.

Parameters

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

7.39.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency

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

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

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

cutoffFrequency	Normalised cutoff frequency (0..1/2)

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

reqOrder	The actual order which can be less than the instantiated one

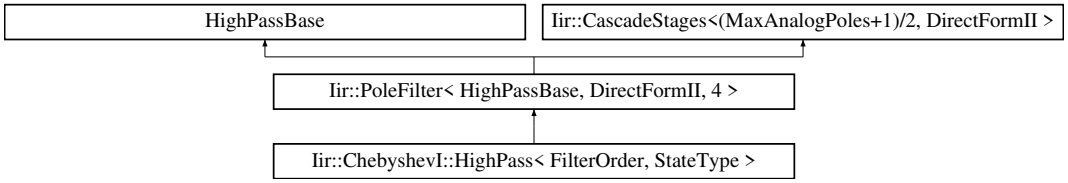
cutoffFrequency	Normalised cutoff frequency (0..1/2)

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

- iir/Butterworth.h

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

`#include <ChebyshevI.h>`
Inheritance diagram for Iir::ChebyshevI::HighPass< FilterOrder, StateType >:



Public Member Functions

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

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.40.1 Detailed Description

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

ChebyshevI highpass filter

Parameters

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

7.40.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency.
rippleDb	Permitted ripples in dB in the passband

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

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

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

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

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

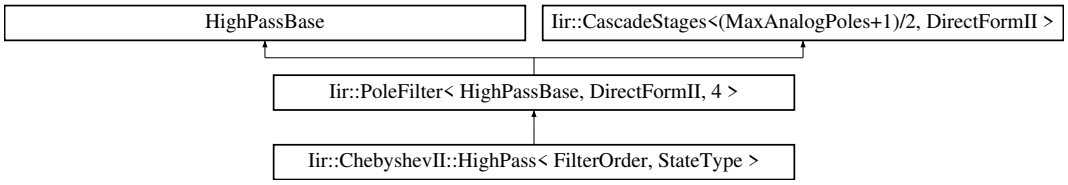
reqOrder	Actual order for the filter calculations
cutoffFrequency	Normalised cutoff frequency (0..1/2)
rippleDb	Permitted ripples in dB in the passband

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

- iir/ChebyshevI.h

7.41 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.41.1 Detailed Description

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

ChebyshevII highpass filter

Parameters

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

7.41.2 Member Function Documentation

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

sampleRate	Sampling rate

cutoffFrequency	Cutoff frequency.
stopBandDb	Permitted ripples in dB in the stopband

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

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

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

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

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

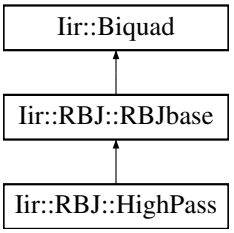
reqOrder	Requested order which can be less than the instantiated one
cutoffFrequency	Normalised cutoff frequency (0..1/2)
stopBandDb	Permitted ripples in dB in the stopband

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

- iir/ChebyshevII.h

7.42 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)))

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.42.1 Detailed Description

Highpass.

7.42.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

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

setupN()

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

Calculates the coefficients

Parameters

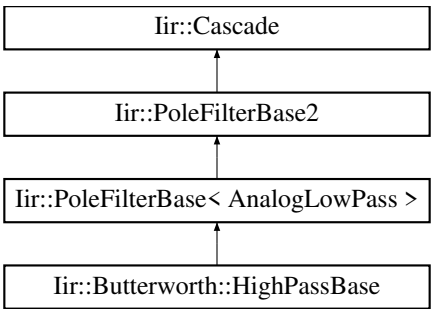
cutoffFrequency	Normalised cutoff frequency (0..1/2)
q	Q factor determines the resonance peak at the cutoff.

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

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

7.43 Iir::Butterworth::HighPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

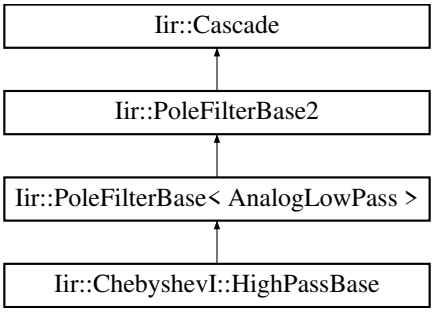
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.44 Iir::ChebyshevI::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

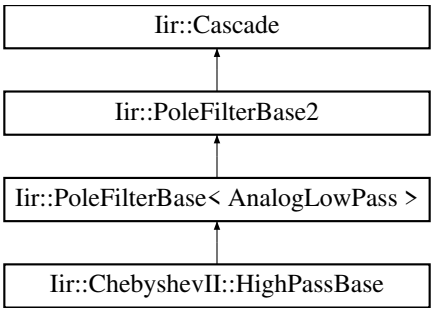
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.45 Iir::ChebyshevII::HighPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

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

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

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

7.46 Iir::HighPassTransform Class Reference

`#include <PoleFilter.h>`

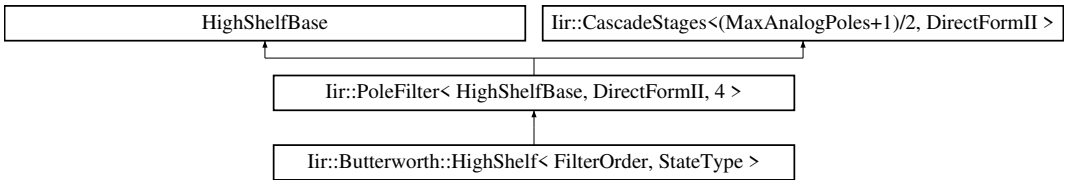
7.46.1 Detailed Description

low pass to high pass
The documentation for this class was generated from the following files:

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

7.47 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.47.1 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

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

7.47.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff
gainDb	Gain in dB of the filter in the passband

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

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

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

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

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

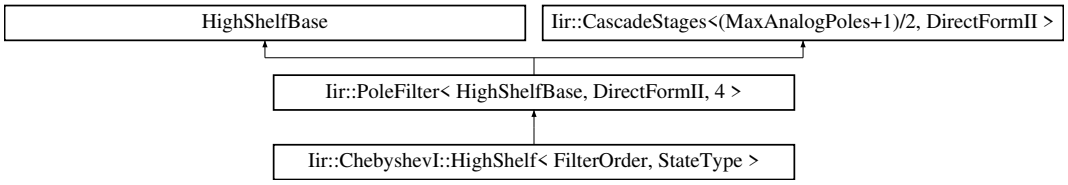
reqOrder	The actual order which can be less than the instantiated one
cutoffFrequency	Normalised cutoff frequency (0..1/2)
gainDb	Gain in dB of the filter in the passband

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

- iir/Butterworth.h

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

`#include <ChebyshevI.h>`
Inheritance diagram for Iir::ChebyshevI::HighShelf< FilterOrder, StateType >:



Public Member Functions

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

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.48.1 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

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

7.48.2 Member Function Documentation

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

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

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

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

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

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

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

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

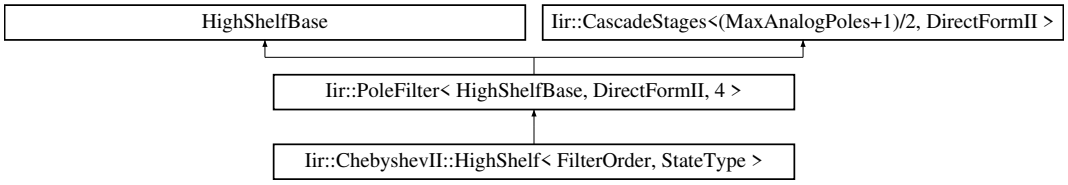
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

7.49 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.49.1 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

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

7.49.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency.
gainDb	Gain the passbard. The stopband has 0 dB gain.
stopBandDb	Permitted ripples in dB in the stopband

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

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

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

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

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

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

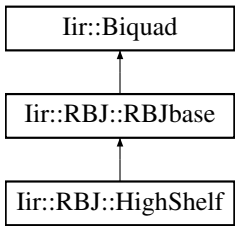
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

7.50 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.50.1 Detailed Description

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

7.50.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

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

setupN()

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

Calculates the coefficients

Parameters

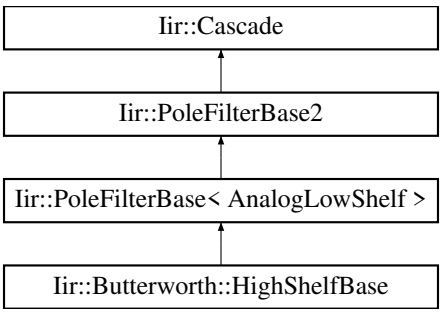
cutoffFrequency	Normalised cutoff frequency
gainDb	Gain in the passband
shelfSlope	Slope between stop/passband. 1 = as steep as it can.

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

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

7.51 Iir::Butterworth::HighShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::HighShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

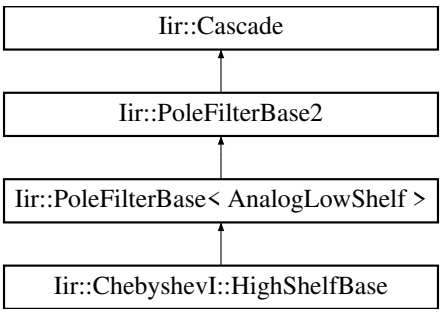
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.52 Iir::ChebyshevI::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::HighShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

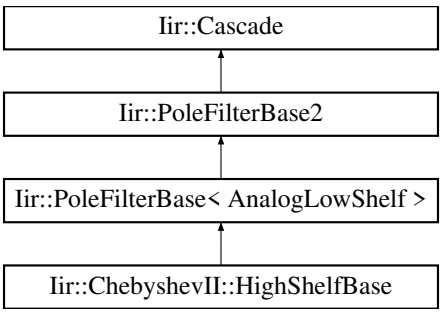
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.53 Iir::ChebyshevII::HighShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::HighShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

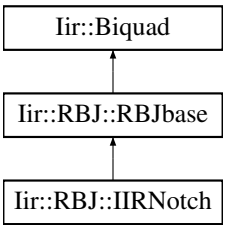
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.54 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.54.1 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∗(centerFrequency/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.

7.54.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
centerFrequency	Center frequency of the notch
q_factor	Q factor of the notch (1 to ∼20)

setupN()

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

Calculates the coefficients

Parameters

centerFrequency	Normalised centre frequency of the notch
q_factor	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

7.55 Iir::Layout< MaxPoles > Class Template Reference

`#include <Layout.h>`

7.55.1 Detailed Description

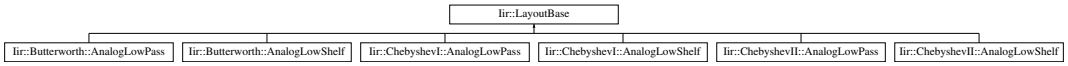
template<int MaxPoles>
class Iir::Layout< MaxPoles >

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

- iir/Layout.h

7.56 Iir::LayoutBase Class Reference

`#include <Layout.h>`
Inheritance diagram for Iir::LayoutBase:



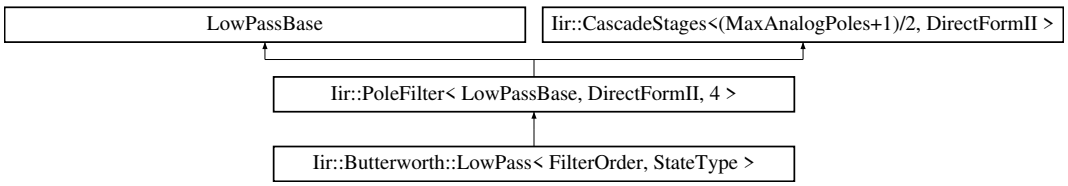
7.56.1 Detailed Description

Base uses pointers to reduce template instantiations
The documentation for this class was generated from the following file:

- iir/Layout.h

7.57 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.57.1 Detailed Description

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

Butterworth Lowpass filter.

Parameters

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

7.57.2 Member Function Documentation

setup() [1/2]

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

Calculates the coefficients

Parameters

sampleRate	Sampling rate
cutoffFrequency	Cutoff

setup() [2/2]

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

Calculates the coefficients

Parameters

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

setupN() [1/2]

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

Calculates the coefficients

Parameters

cutoffFrequency	Normalised cutoff frequency (0..1/2)

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

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

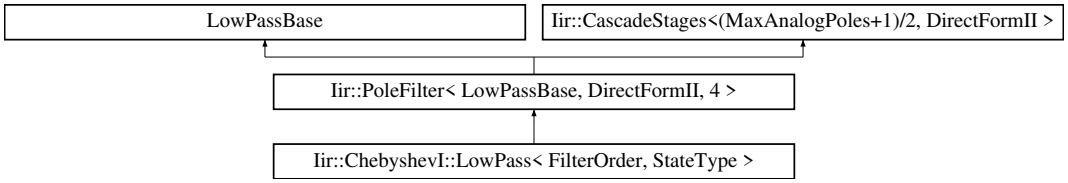
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

7.58 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.58.1 Detailed Description

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

ChebyshevI lowpass filter

Parameters

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

7.58.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency
rippleDb	Permitted ripples in dB in the passband

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

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

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

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

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

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

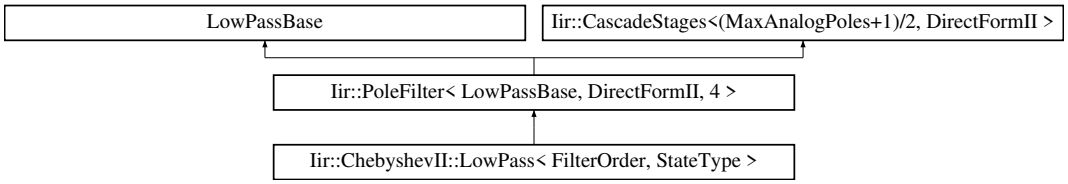
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

7.59 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.59.1 Detailed Description

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

ChebyshevII lowpass filter

Parameters

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

7.59.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency.
stopBandDb	Permitted ripples in dB in the stopband

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

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

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

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

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

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

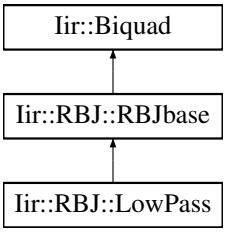
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

7.60 Iir::RBJ::LowPass Struct Reference

`#include <RBJ.h>`

Inheritance diagram for Iir::RBJ::LowPass:



Public Member Functions

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

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.60.1 Detailed Description

Lowpass.

7.60.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

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

setupN()

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

Calculates the coefficients

Parameters

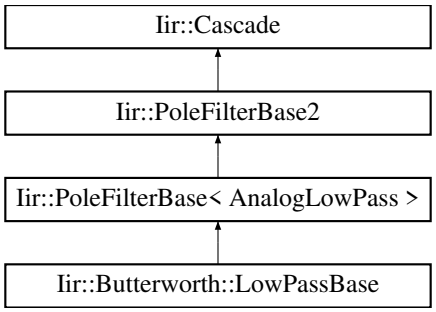
cutoffFrequency	Normalised cutoff frequency
q	Q factor determines the resonance peak at the cutoff.

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

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

7.61 Iir::Butterworth::LowPassBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

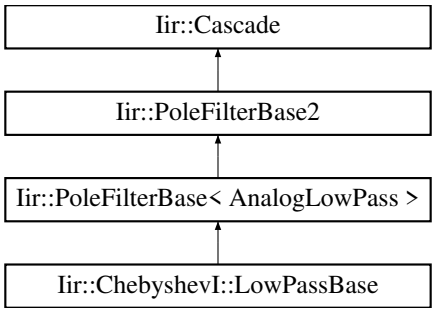
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.62 Iir::ChebyshevI::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

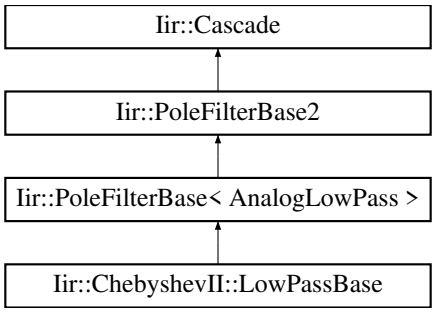
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.63 Iir::ChebyshevII::LowPassBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowPassBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

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

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

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

7.64 Iir::LowPassTransform Class Reference

`#include <PoleFilter.h>`

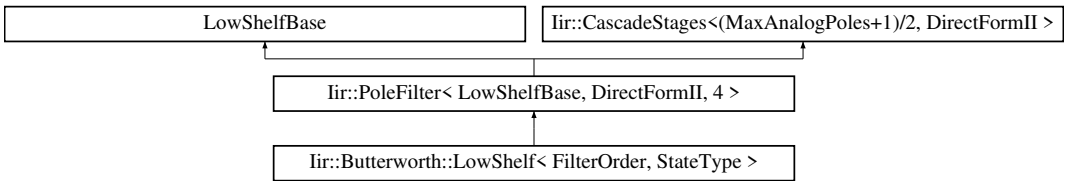
7.64.1 Detailed Description

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

7.65 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.65.1 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

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

7.65.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff
gainDb	Gain in dB of the filter in the passband

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

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

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

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

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

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

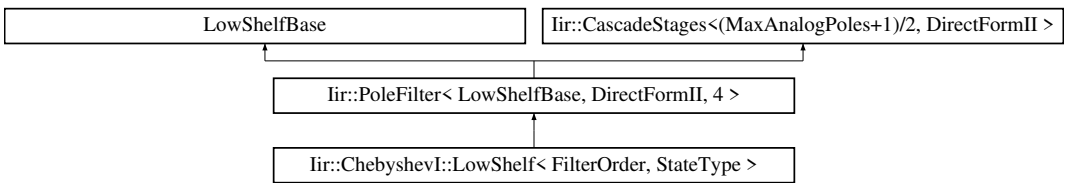
The documentation for this struct was generated from the following file:

- iir/Butterworth.h

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

`#include <ChebyshevI.h>`

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



Public Member Functions

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

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

- void reset ()
- void setup (const double(&sosCoefficients)[MaxStages][6])

- Sample filter (const Sample in)
- const Cascade::Storage getCascadeStorage ()

7.66.1 Detailed Description

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

ChebyshevI low shelf filter. Specified gain in the passband. Otherwise 0 dB.

Parameters

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

7.66.2 Member Function Documentation

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

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

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

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

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

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

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

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

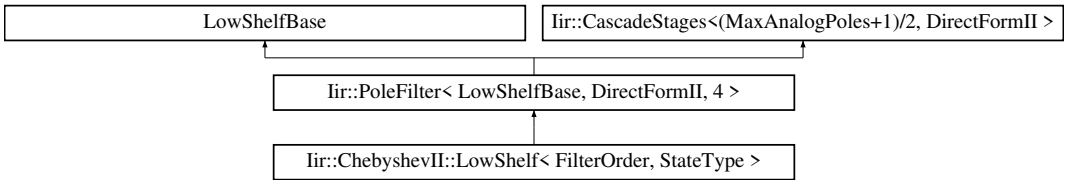
The documentation for this struct was generated from the following file:

- iir/ChebyshevI.h

7.67 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)

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, DirectFormII >

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

7.67.1 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

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

7.67.2 Member Function Documentation

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

sampleRate	Sampling rate
cutoffFrequency	Cutoff frequency.
gainDb	Gain of the passbard. The stopband has 0 dB gain.
stopBandDb	Permitted ripples in dB in the stopband

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

reqOrder	Requested order which can be less than the instantiated one
sampleRate	Sampling rate

cutoffFrequency	Cutoff frequency
gainDb	Gain of the passbard. The stopband has 0 dB gain.
stopBandDb	Permitted ripples in dB in the stopband

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

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

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

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

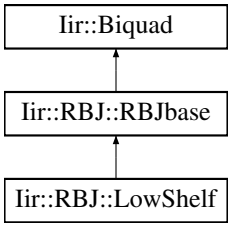
The documentation for this struct was generated from the following file:

- iir/ChebyshevII.h

7.68 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)

Public Member Functions inherited from Iir::RBJ::RBJbase

- 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

Public Member Functions inherited from Iir::Biquad

- 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)

7.68.1 Detailed Description

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

7.68.2 Member Function Documentation

setup()

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

Calculates the coefficients

Parameters

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

setupN()

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

Calculates the coefficients

Parameters

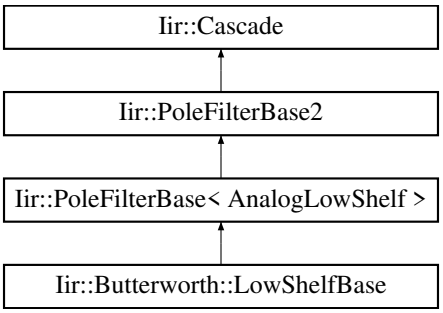
cutoffFrequency	Normalised cutoff frequency
gainDb	Gain in the passband
shelfSlope	Slope between stop/passband. 1 = as steep as it can.

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

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

7.69 Iir::Butterworth::LowShelfBase Struct Reference

Inheritance diagram for Iir::Butterworth::LowShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

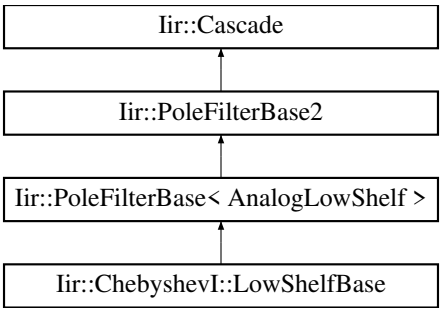
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.70 Iir::ChebyshevI::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevI::LowShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

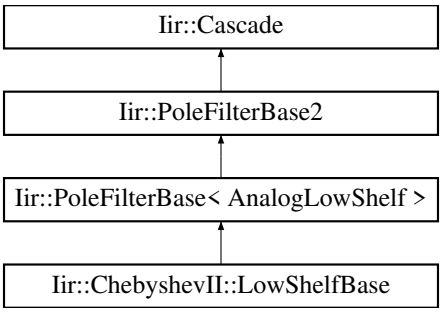
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.71 Iir::ChebyshevII::LowShelfBase Struct Reference

Inheritance diagram for Iir::ChebyshevII::LowShelfBase:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

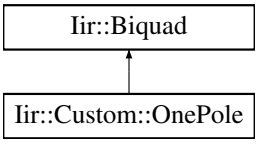
- int getNumStages () const
- const Biquad & operator[ ] (int index)
- complex_t response (double normalizedFrequency) const
- std::vector< PoleZeroPair > getPoleZeros () const

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

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

7.72 Iir::Custom::OnePole Struct Reference

`#include <Custom.h>`
Inheritance diagram for Iir::Custom::OnePole:



Additional Inherited Members

Public Member Functions inherited from Iir::Biquad

- 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)

7.72.1 Detailed Description

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

Parameters

scale	Scale the FIR coefficients by this factor
pole	Position of the pole on the real axis
zero	Position of the zero on the real axis

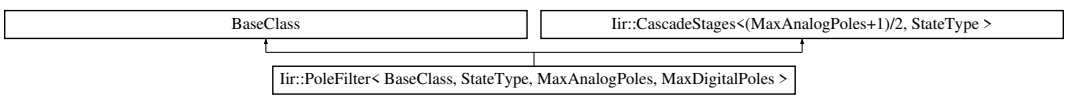
The documentation for this struct was generated from the following files:

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

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

`#include <PoleFilter.h>`

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



Additional Inherited Members

Public Member Functions inherited from Iir::CascadeStages<(MaxAnalogPoles+1)/2, StateType >

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

7.73.1 Detailed Description

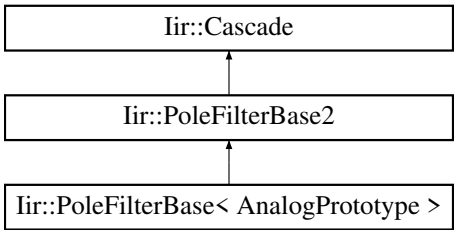
template<class BaseClass, class StateType, int MaxAnalogPoles, int MaxDigitalPoles = MaxAnalogPoles>
struct Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >

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

- iir/PoleFilter.h

7.74 Iir::PoleFilterBase< AnalogPrototype > Class Template Reference

`#include <PoleFilter.h>`
Inheritance diagram for Iir::PoleFilterBase< AnalogPrototype >:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

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

7.74.1 Detailed Description

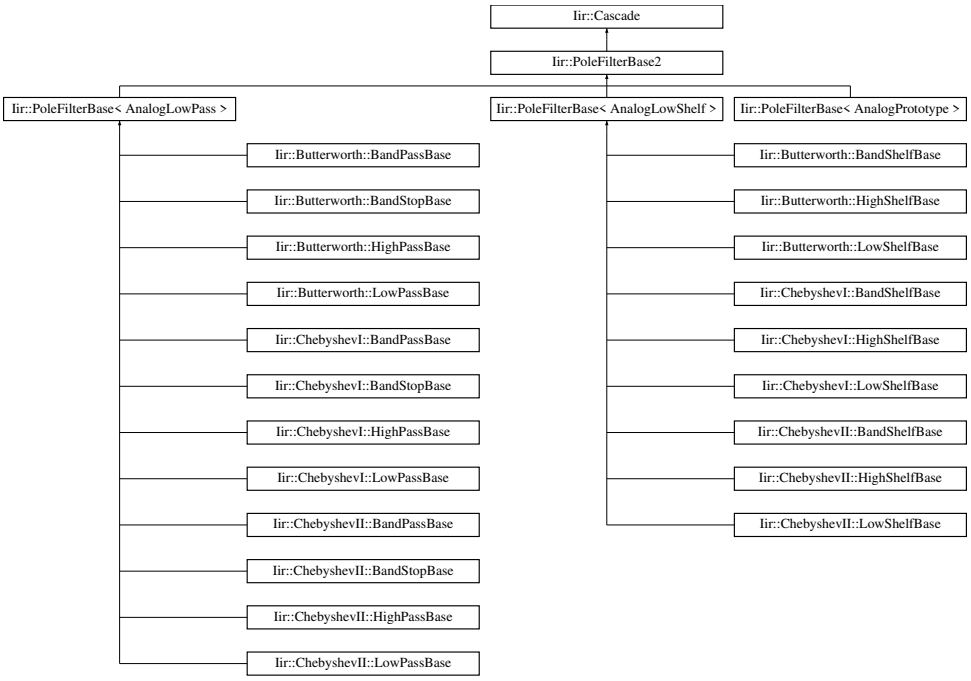
template<class AnalogPrototype>
class Iir::PoleFilterBase< AnalogPrototype >

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

7.75 Iir::PoleFilterBase2 Class Reference

`#include <PoleFilter.h>`
Inheritance diagram for Iir::PoleFilterBase2:



Additional Inherited Members

Public Member Functions inherited from Iir::Cascade

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

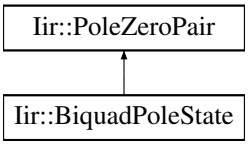
7.75.1 Detailed Description

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

- iir/PoleFilter.h

7.76 Iir::PoleZeroPair Struct Reference

`#include <Types.h>`
Inheritance diagram for Iir::PoleZeroPair:



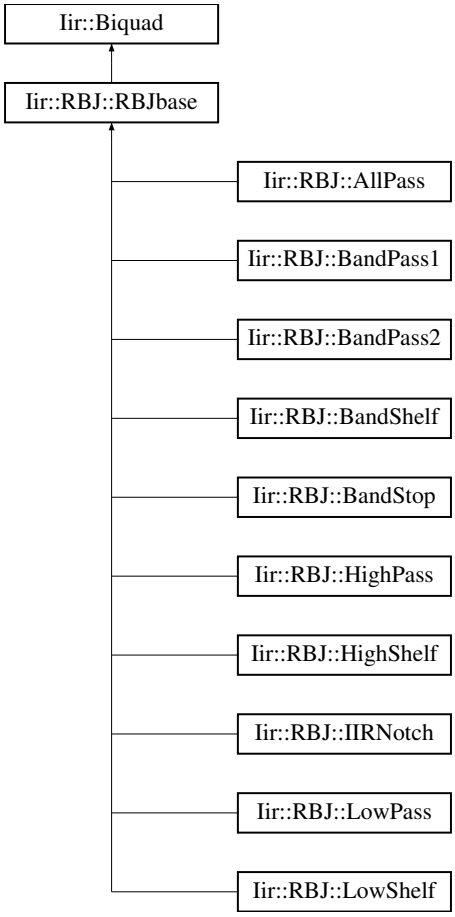
7.76.1 Detailed Description

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

7.77 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

Public Member Functions inherited from Iir::Biquad

- 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)

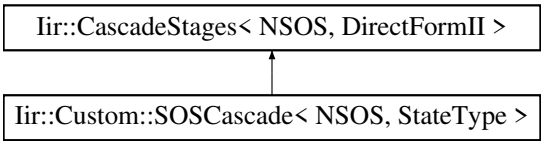
7.77.1 Detailed Description

The base class of all RBJ filters
The documentation for this struct was generated from the following file:

- iir/RBJ.h

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

`#include <Custom.h>`
Inheritance diagram for Iir::Custom::SOSCascade< NSOS, StateType >:



Public Member Functions

- SOSCascade ()=default
- SOSCascade (const double(&sosCoefficients)[NSOS][6])
- SOSCascade (const std::array< std::array< double, 6 >, NSOS > sosCoefficients)
- void setup (const double(&sosCoefficients)[NSOS][6])
- void setup (const std::array< std::array< double, 6 >, NSOS > sosCoefficients)

Public Member Functions inherited from Iir::CascadeStages< NSOS, DirectFormII >

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

7.78.1 Detailed Description

template<std::size_t NSOS, class StateType = DirectFormII>
struct Iir::Custom::SOSCascade< NSOS, StateType >

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

Parameters

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

7.78.2 Constructor & Destructor Documentation

SOSCascade() [1/3]

```
template<std::size_t NSOS, class StateType = DirectFormII>
Iir::Custom::SOSCascade< NSOS, StateType >::SOSCascade ()  [default]
```

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

SOSCascade() [2/3]

```
template<std::size_t 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

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

SOSCascade() [3/3]

```
template<std::size_t NSOS, class StateType = DirectFormII>
Iir::Custom::SOSCascade< NSOS, StateType >::SOSCascade (
            const std::array< std::array< double, 6 >, NSOS > sosCoefficients)  [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

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

7.78.3 Member Function Documentation

setup() [1/2]

```
template<std::size_t 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

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

setup() [2/2]

```
template<std::size_t NSOS, class StateType = DirectFormII>
void Iir::Custom::SOSCascade< NSOS, StateType >::setup (
            const std::array< std::array< double, 6 >, NSOS > sosCoefficients)  [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

sosCoefficients	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

7.79 Iir::Cascade::Storage Struct Reference

`#include <Cascade.h>`

7.79.1 Detailed Description

To return the array from a function and to set it. Transmits number of stages and the pointer to the array.
The documentation for this struct was generated from the following file:

- iir/Cascade.h

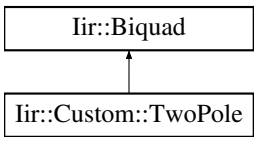
7.80 Iir::TransposedDirectFormII Class Reference

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

- iir/State.h

7.81 Iir::Custom::TwoPole Struct Reference

`#include <Custom.h>`
Inheritance diagram for Iir::Custom::TwoPole:



Additional Inherited Members

Public Member Functions inherited from Iir::Biquad

- 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)

7.81.1 Detailed Description

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

Parameters

poleRho	Radius of the pole
poleTheta	Angle of the pole
zeroRho	Radius of the zero
zeroTheta	Angle of the zero

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

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

8 File Documentation

8.1 Iir.h

```
00001
00034
00035 #pragma once
00036
00037 //
00038 // Include this file in your application to get everything
00039 //
00040
00041 #include "iir/Common.h"
00042
00043 #include "iir/Biquad.h"
00044 #include "iir/Cascade.h"
00045 #include "iir/PoleFilter.h"
00046 #include "iir/State.h"
00047
00048 #include "iir/Butterworth.h"
00049 #include "iir/ChebyshevI.h"
00050 #include "iir/ChebyshevII.h"
00051 #include "iir/Custom.h"
00052 #include "iir/RBJ.h"
```

8.2 Biquad.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "MathSupplement.h"
00040 #include "Types.h"
00041
00042 namespace Iir {
00043
00044         struct IIR_EXPORT BiquadPoleState;
00045
00046 /*
00047  * Holds coefficients for a second order Infinite Impulse Response
00048  * digital filter. This is the building block for all IIR filters.
```

```
00049  *
00050  */
00051         class IIR_EXPORT Biquad {
00052         public:
00053
00054         Biquad() = default;
00055
00060         complex_t response (double normalizedFrequency) const;
00061
00065         std::vector<PoleZeroPair> getPoleZeros () const;
00066
00070         double getA0 () const { return m_a0; }
00071
00075         double getA1 () const { return m_a1*m_a0; }
00076
00080         double getA2 () const { return m_a2*m_a0; }
00081
00085         double getB0 () const { return m_b0*m_a0; }
00086
00090         double getB1 () const { return m_b1*m_a0; }
00091
00095         double getB2 () const { return m_b2*m_a0; }
00096
00103         template <class StateType>
00104         inline double filter(double s, StateType& state) const
00105         {
00106                 return state.filter(s, *this);
00107         }
00108
00109         public:
00119         void setCoefficients (double a0, double a1, double a2,
00120                               double b0, double b1, double b2);
00121
00125         void setOnePole (complex_t pole, complex_t zero);
00126
00130         void setTwoPole (complex_t pole1, complex_t zero1,
00131                          complex_t pole2, complex_t zero2);
00132
00136         void setPoleZeroPair (const PoleZeroPair& pair)
00137         {
00138                 if (pair.isSinglePole ())
00139                         setOnePole (pair.poles.first, pair.zeros.first);
00140                 else
00141                         setTwoPole (pair.poles.first, pair.zeros.first,
00142                                     pair.poles.second, pair.zeros.second);
00143         }
00144
00145         void setPoleZeroForm (const BiquadPoleState& bps);
00146
00150         void setIdentity ();
00151
00156         void applyScale (double scale);
00157
00158         public:
00159         double m_a0 = 1.0;
00160         double m_a1 = 0.0;
00161         double m_a2 = 0.0;
00162         double m_b1 = 0.0;
00163         double m_b2 = 0.0;
00164         double m_b0 = 1.0;
00165         };
00166
00167 //-----------------------------------------------------------------------------
00168
00169
00174         struct IIR_EXPORT BiquadPoleState : PoleZeroPair
00175         {
00176                 BiquadPoleState () = default;
00177
00178                 explicit BiquadPoleState (const Biquad& s);
00179
00180                 double gain = 1.0;
00181         };
00182
00183 }
```

8.3 Butterworth.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Cascade.h"
00040 #include "PoleFilter.h"
```

```
00041 #include "State.h"
00042
00043 namespace Iir {
00044
00051 namespace Butterworth {
00052
00056 class IIR_EXPORT AnalogLowPass : public LayoutBase
00057 {
00058 public:
00059         AnalogLowPass ();
00060
00061         void design (const int numPoles);
00062
00063 private:
00064         int m_numPoles = 0;
00065 };
00066
00067 //------------------------------------------------------------------------------
00068
00072 class IIR_EXPORT AnalogLowShelf : public LayoutBase
00073 {
00074 public:
00075         AnalogLowShelf ();
00076
00077         void design (int numPoles, double gainDb);
00078
00079 private:
00080         int m_numPoles = 0;
00081         double m_gainDb = 0.0;
00082 };
00083
00084 //------------------------------------------------------------------------------
00085
00086 struct IIR_EXPORT LowPassBase : PoleFilterBase <AnalogLowPass>
00087 {
00088         void setup (int order,
00089                     double cutoffFrequency);
00090 };
00091
00092 struct IIR_EXPORT HighPassBase : PoleFilterBase <AnalogLowPass>
00093 {
00094         void setup (int order,
00095                     double cutoffFrequency);
00096 };
00097
00098 struct IIR_EXPORT BandPassBase : PoleFilterBase <AnalogLowPass>
00099 {
00100         void setup (int order,
00101                     double centerFrequency,
00102                     double widthFrequency);
00103 };
00104
00105 struct IIR_EXPORT BandStopBase : PoleFilterBase <AnalogLowPass>
00106 {
00107         void setup (int order,
00108                     double centerFrequency,
00109                     double widthFrequency);
00110 };
00111
00112 struct IIR_EXPORT LowShelfBase : PoleFilterBase <AnalogLowShelf>
00113 {
00114         void setup (int order,
00115                     double cutoffFrequency,
00116                     double gainDb);
00117 };
00118
00119 struct IIR_EXPORT HighShelfBase : PoleFilterBase <AnalogLowShelf>
00120 {
00121         void setup (int order,
00122                     double cutoffFrequency,
00123                     double gainDb);
00124 };
00125
00126 struct IIR_EXPORT BandShelfBase : PoleFilterBase <AnalogLowShelf>
00127 {
00128         void setup (int order,
00129                     double centerFrequency,
00130                     double widthFrequency,
00131                     double gainDb);
00132 };
00133
00134 //------------------------------------------------------------------------------
00135
00136 //
00137 // Filters for the user
00138 //
00139
```

```
00145 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00146 struct LowPass : PoleFilter <LowPassBase, StateType, FilterOrder>
00147 {
00153         void setup (double sampleRate,
00154                     double cutoffFrequency) {
00155                 LowPassBase::setup (FilterOrder,
00156                                     cutoffFrequency / sampleRate);
00157         }
00158
00165         void setup (int reqOrder,
00166                     double sampleRate,
00167                     double cutoffFrequency) {
00168                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00169                 LowPassBase::setup (reqOrder,
00170                                     cutoffFrequency / sampleRate);
00171         }
00172
00173
00178         void setupN(double cutoffFrequency) {
00179                 LowPassBase::setup (FilterOrder,
00180                                     cutoffFrequency);
00181         }
00182
00188         void setupN(int reqOrder,
00189                     double cutoffFrequency) {
00190                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00191                 LowPassBase::setup (reqOrder,
00192                                     cutoffFrequency);
00193         }
00194 };
00195
00201 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00202 struct HighPass : PoleFilter <HighPassBase, StateType, FilterOrder>
00203 {
00204
00210         void setup (double sampleRate,
00211                     double cutoffFrequency) {
00212                 HighPassBase::setup (FilterOrder,
00213                                      cutoffFrequency / sampleRate);
00214         }
00221         void setup (int reqOrder,
00222                     double sampleRate,
00223                     double cutoffFrequency) {
00224                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00225                 HighPassBase::setup (reqOrder,
00226                                      cutoffFrequency / sampleRate);
00227         }
00228
00229
00234         void setupN(double cutoffFrequency) {
00235                 HighPassBase::setup (FilterOrder,
00236                                      cutoffFrequency);
00237         }
00243         void setupN(int reqOrder,
00244                     double cutoffFrequency) {
00245                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00246                 HighPassBase::setup (reqOrder,
00247                                      cutoffFrequency);
00248         }
00249 };
00250
00256 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00257 struct BandPass : PoleFilter <BandPassBase, StateType, FilterOrder, FilterOrder*2>
00258 {
00265         void setup (double sampleRate,
00266                     double centerFrequency,
00267                     double widthFrequency) {
00268                 BandPassBase::setup(FilterOrder,
00269                                     centerFrequency / sampleRate,
00270                                     widthFrequency / sampleRate);
00271         }
00272
00280         void setup (int reqOrder,
00281                     double sampleRate,
00282                     double centerFrequency,
00283                     double widthFrequency) {
00284                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00285                 BandPassBase::setup(reqOrder,
00286                                     centerFrequency / sampleRate,
00287                                     widthFrequency / sampleRate);
00288         }
00289
00290
00291
00297         void setupN(double centerFrequency,
00298                     double widthFrequency) {
00299                 BandPassBase::setup(FilterOrder,
```

```
00300                                   centerFrequency,
00301                                   widthFrequency);
00302         }
00303
00310         void setupN(int reqOrder,
00311                     double centerFrequency,
00312                     double widthFrequency) {
00313                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00314                 BandPassBase::setup(reqOrder,
00315                                     centerFrequency,
00316                                     widthFrequency);
00317         }
00318 };
00319
00320
00326 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00327 struct BandStop : PoleFilter <BandStopBase, StateType, FilterOrder, FilterOrder*2>
00328 {
00335         void setup (double sampleRate,
00336                     double centerFrequency,
00337                     double widthFrequency) {
00338                 BandStopBase::setup (FilterOrder,
00339                                      centerFrequency / sampleRate,
00340                                      widthFrequency / sampleRate);
00341         }
00342
00350         void setup (int reqOrder,
00351                     double sampleRate,
00352                     double centerFrequency,
00353                     double widthFrequency) {
00354                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00355                 BandStopBase::setup (reqOrder,
00356                                      centerFrequency / sampleRate,
00357                                      widthFrequency / sampleRate);
00358         }
00359
00360
00361
00367         void setupN(double centerFrequency,
00368                     double widthFrequency) {
00369                 BandStopBase::setup (FilterOrder,
00370                                      centerFrequency,
00371                                      widthFrequency);
00372         }
00373
00380         void setupN(int reqOrder,
00381                     double centerFrequency,
00382                     double widthFrequency) {
00383                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00384                 BandStopBase::setup (reqOrder,
00385                                      centerFrequency,
00386                                      widthFrequency);
00387         }
00388
00389 };
00390
00397 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00398 struct LowShelf : PoleFilter <LowShelfBase, StateType, FilterOrder>
00399 {
00406         void setup (double sampleRate,
00407                     double cutoffFrequency,
00408                     double gainDb) {
00409                 LowShelfBase::setup (FilterOrder,
00410                                      cutoffFrequency / sampleRate,
00411                                      gainDb);
00412         }
00413
00421         void setup (int reqOrder,
00422                     double sampleRate,
00423                     double cutoffFrequency,
00424                     double gainDb) {
00425                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00426                 LowShelfBase::setup (reqOrder,
00427                                      cutoffFrequency / sampleRate,
00428                                      gainDb);
00429         }
00430
00431
00432
00433
00439         void setupN(double cutoffFrequency,
00440                     double gainDb) {
00441                 LowShelfBase::setup (FilterOrder,
00442                                      cutoffFrequency,
00443                                      gainDb);
00444         }
00445
```

```
00452         void setupN(int reqOrder,
00453                     double cutoffFrequency,
00454                     double gainDb) {
00455                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00456                 LowShelfBase::setup (reqOrder,
00457                                      cutoffFrequency,
00458                                      gainDb);
00459         }
00460
00461 };
00462
00463
00470 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00471 struct HighShelf : PoleFilter <HighShelfBase, StateType, FilterOrder>
00472 {
00479         void setup (double sampleRate,
00480                     double cutoffFrequency,
00481                     double gainDb) {
00482                 HighShelfBase::setup (FilterOrder,
00483                                       cutoffFrequency / sampleRate,
00484                                       gainDb);
00485         }
00486
00494         void setup (int reqOrder,
00495                     double sampleRate,
00496                     double cutoffFrequency,
00497                     double gainDb) {
00498                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00499                 HighShelfBase::setup (reqOrder,
00500                                       cutoffFrequency / sampleRate,
00501                                       gainDb);
00502         }
00503
00504
00505
00511         void setupN(double cutoffFrequency,
00512                     double gainDb) {
00513                 HighShelfBase::setup (FilterOrder,
00514                                       cutoffFrequency,
00515                                       gainDb);
00516         }
00517
00524         void setupN(int reqOrder,
00525                     double cutoffFrequency,
00526                     double gainDb) {
00527                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00528                 HighShelfBase::setup (reqOrder,
00529                                       cutoffFrequency,
00530                                       gainDb);
00531         }
00532 };
00533
00534
00541 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00542 struct BandShelf : PoleFilter <BandShelfBase, StateType, FilterOrder, FilterOrder*2>
00543 {
00551         void setup (double sampleRate,
00552                     double centerFrequency,
00553                     double widthFrequency,
00554                     double gainDb) {
00555                 BandShelfBase::setup (FilterOrder,
00556                                       centerFrequency / sampleRate,
00557                                       widthFrequency / sampleRate,
00558                                       gainDb);
00559         }
00560
00569         void setup (int reqOrder,
00570                     double sampleRate,
00571                     double centerFrequency,
00572                     double widthFrequency,
00573                     double gainDb) {
00574                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00575                 BandShelfBase::setup (reqOrder,
00576                                       centerFrequency / sampleRate,
00577                                       widthFrequency / sampleRate,
00578                                       gainDb);
00579         }
00580
00581
00582
00589         void setupN(double centerFrequency,
00590                     double widthFrequency,
00591                     double gainDb) {
00592                 BandShelfBase::setup (FilterOrder,
00593                                       centerFrequency,
00594                                       widthFrequency,
00595                                       gainDb);
```

```
00596         }
00597
00605         void setupN(int reqOrder,
00606                     double centerFrequency,
00607                     double widthFrequency,
00608                     double gainDb) {
00609                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00610                 BandShelfBase::setup (reqOrder,
00611                                       centerFrequency,
00612                                       widthFrequency,
00613                                       gainDb);
00614         }
00615 };
00616
00617 }
00618
00619 }
```

8.4 Cascade.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Biquad.h"
00040 #include "Layout.h"
00041 #include "MathSupplement.h"
00042
00043 namespace Iir {
00044
00048         class IIR_EXPORT Cascade
00049         {
00050         public:
00051
00052         Cascade () = default;
00053
00058         struct IIR_EXPORT Storage
00059         {
00060             Storage() = delete;
00061             Storage(const int maxNumBiquads, Biquad* const biquadArray) : maxStages(maxNumBiquads),
      stageArray(biquadArray) {}
00062             const int maxStages = 0;
00063             Biquad* const stageArray = nullptr;
00064         };
00065
00069         int getNumStages () const
00070         {
00071                 return m_numStages;
00072         }
00073
00077         const Biquad& operator[] (int index)
00078         {
00079                 if ((index < 0) || (index >= m_numStages))
00080                         throw_invalid_argument("Index out of bounds.");
00081                 return m_stageArray[index];
00082         }
00083
00088         complex_t response (double normalizedFrequency) const;
00089
00093         std::vector<PoleZeroPair> getPoleZeros () const;
00094
00095         void setCascadeStorage (const Storage& storage);
00096
00097         void applyScale (double scale);
00098
00099         void setLayout (const LayoutBase& proto);
00100
00101         private:
00102         int m_numStages = 0;
00103         int m_maxStages = 0;
00104         Biquad* m_stageArray = nullptr;
00105         };
00106
00107
00108 //---------------------------------------------------------------------------
00109
00114         template <int MaxStages,class StateType>
00115         class CascadeStages {
00116
00117         public:
00118         CascadeStages() = default;
00119
00120
00121         public:
```

```
00125         void reset ()
00126         {
00127                 for (auto &state: m_states)
00128                         state.reset();
00129         }
00130
00131         public:
00137         void setup (const double (&sosCoefficients)[MaxStages][6]) {
00138                 for (int i = 0; i < MaxStages; i++) {
00139                         m_stages[i].setCoefficients(
00140                                 sosCoefficients[i][3],
00141                                 sosCoefficients[i][4],
00142                                 sosCoefficients[i][5],
00143                                 sosCoefficients[i][0],
00144                                 sosCoefficients[i][1],
00145                                 sosCoefficients[i][2]);
00146                 }
00147         }
00148
00149         void setup (const std::array<std::array<double,6>,MaxStages> sosCoefficients) {
00150                 for (int i = 0; i < MaxStages; i++) {
00151                         m_stages[i].setCoefficients(
00152                                 sosCoefficients[i][3],
00153                                 sosCoefficients[i][4],
00154                                 sosCoefficients[i][5],
00155                                 sosCoefficients[i][0],
00156                                 sosCoefficients[i][1],
00157                                 sosCoefficients[i][2]);
00158                 }
00159         }
00160
00161         public:
00167         template <typename Sample>
00168         inline Sample filter(const Sample in)
00169         {
00170                 double out = in;
00171                 StateType* state = m_states;
00172                 for (const auto &stage: m_stages)
00173                         out = (state++)->filter(out, stage);
00174                 return static_cast<Sample> (out);
00175         }
00176
00180         const Cascade::Storage getCascadeStorage()
00181         {
00182             const Cascade::Storage s(MaxStages, m_stages);
00183             return s;
00184         }
00185
00186         private:
00187         Biquad m_stages[MaxStages] = {};
00188         StateType m_states[MaxStages] = {};
00189         };
00190
00191 }
```

8.5 ChebyshevI.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Cascade.h"
00040 #include "PoleFilter.h"
00041 #include "State.h"
00042
00043 namespace Iir {
00044
00049 namespace ChebyshevI {
00050
00054 class IIR_EXPORT AnalogLowPass : public LayoutBase
00055 {
00056 public:
00057         AnalogLowPass ();
00058
00059         void design (const int numPoles,
00060                      double rippleDb);
00061
00062 private:
00063         int m_numPoles = 0;
00064         double m_rippleDb = 0.0;
00065 };
00066
00070 class IIR_EXPORT AnalogLowShelf : public LayoutBase
00071 {
```

```
00072 public:
00073         AnalogLowShelf ();
00074
00075         void design (int numPoles,
00076                      double gainDb,
00077                      double rippleDb);
00078
00079 private:
00080         int m_numPoles = 0;
00081         double m_rippleDb = 0.0;
00082         double m_gainDb = 0.0;
00083 };
00084
00085 //------------------------------------------------------------------------------
00086
00087 struct IIR_EXPORT LowPassBase : PoleFilterBase <AnalogLowPass>
00088 {
00089   void setup (int order,
00090               double cutoffFrequency,
00091               double rippleDb);
00092 };
00093
00094 struct IIR_EXPORT HighPassBase : PoleFilterBase <AnalogLowPass>
00095 {
00096   void setup (int order,
00097               double cutoffFrequency,
00098               double rippleDb);
00099 };
00100
00101 struct IIR_EXPORT BandPassBase : PoleFilterBase <AnalogLowPass>
00102 {
00103   void setup (int order,
00104               double centerFrequency,
00105               double widthFrequency,
00106               double rippleDb);
00107 };
00108
00109 struct IIR_EXPORT BandStopBase : PoleFilterBase <AnalogLowPass>
00110 {
00111   void setup (int order,
00112               double centerFrequency,
00113               double widthFrequency,
00114               double rippleDb);
00115 };
00116
00117 struct IIR_EXPORT LowShelfBase : PoleFilterBase <AnalogLowShelf>
00118 {
00119   void setup (int order,
00120               double cutoffFrequency,
00121               double gainDb,
00122               double rippleDb);
00123 };
00124
00125 struct IIR_EXPORT HighShelfBase : PoleFilterBase <AnalogLowShelf>
00126 {
00127   void setup (int order,
00128               double cutoffFrequency,
00129               double gainDb,
00130               double rippleDb);
00131 };
00132
00133 struct IIR_EXPORT BandShelfBase : PoleFilterBase <AnalogLowShelf>
00134 {
00135   void setup (int order,
00136               double centerFrequency,
00137               double widthFrequency,
00138               double gainDb,
00139               double rippleDb);
00140 };
00141
00142 //------------------------------------------------------------------------------
00143
00144 //
00145 // Userland filters
00146 //
00147
00153 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00154         struct LowPass : PoleFilter <LowPassBase, StateType, FilterOrder>
00155         {
00162                 void setup (double sampleRate,
00163                             double cutoffFrequency,
00164                             double rippleDb) {
00165                         LowPassBase::setup (FilterOrder,
00166                                             cutoffFrequency / sampleRate,
00167                                             rippleDb);
00168                 }
00169
```

```
00177                 void setup (int reqOrder,
00178                             double sampleRate,
00179                             double cutoffFrequency,
00180                             double rippleDb) {
00181                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00182                         LowPassBase::setup (reqOrder,
00183                                             cutoffFrequency / sampleRate,
00184                                             rippleDb);
00185                 }
00186
00187
00188
00194                 void setupN(double cutoffFrequency,
00195                             double rippleDb) {
00196                         LowPassBase::setup (FilterOrder,
00197                                             cutoffFrequency,
00198                                             rippleDb);
00199                 }
00200
00207                 void setupN(int reqOrder,
00208                             double cutoffFrequency,
00209                             double rippleDb) {
00210                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00211                         LowPassBase::setup (reqOrder,
00212                                             cutoffFrequency,
00213                                             rippleDb);
00214                 }
00215 };
00216
00222 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00223         struct HighPass : PoleFilter <HighPassBase, StateType, FilterOrder>
00224         {
00231                 void setup (double sampleRate,
00232                             double cutoffFrequency,
00233                             double rippleDb) {
00234                         HighPassBase::setup (FilterOrder,
00235                                              cutoffFrequency / sampleRate,
00236                                              rippleDb);
00237                 }
00238
00246                 void setup (int reqOrder,
00247                             double sampleRate,
00248                             double cutoffFrequency,
00249                             double rippleDb) {
00250                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00251                         HighPassBase::setup (reqOrder,
00252                                              cutoffFrequency / sampleRate,
00253                                              rippleDb);
00254                 }
00255
00256
00257
00263                 void setupN(double cutoffFrequency,
00264                             double rippleDb) {
00265                         HighPassBase::setup (FilterOrder,
00266                                              cutoffFrequency,
00267                                              rippleDb);
00268                 }
00269
00276                 void setupN(int reqOrder,
00277                             double cutoffFrequency,
00278                             double rippleDb) {
00279                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00280                         HighPassBase::setup (reqOrder,
00281                                              cutoffFrequency,
00282                                              rippleDb);
00283                 }
00284 };
00285
00291 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00292         struct BandPass : PoleFilter <BandPassBase, StateType, FilterOrder, FilterOrder*2>
00293         {
00301                 void setup (double sampleRate,
00302                             double centerFrequency,
00303                             double widthFrequency,
00304                             double rippleDb) {
00305                         BandPassBase::setup (FilterOrder,
00306                                centerFrequency / sampleRate,
00307                                widthFrequency / sampleRate,
00308                                rippleDb);
00309                 }
00310
00319                 void setup (int reqOrder,
00320                             double sampleRate,
00321                             double centerFrequency,
00322                             double widthFrequency,
00323                             double rippleDb) {
```

```
00324                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00325                         BandPassBase::setup (reqOrder,
00326                                centerFrequency / sampleRate,
00327                                widthFrequency / sampleRate,
00328                                rippleDb);
00329                 }
00330
00331
00332
00339                 void setupN(double centerFrequency,
00340                             double widthFrequency,
00341                             double rippleDb) {
00342                         BandPassBase::setup (FilterOrder,
00343                                centerFrequency,
00344                                widthFrequency,
00345                                rippleDb);
00346                 }
00347
00355                 void setupN(int reqOrder,
00356                             double centerFrequency,
00357                             double widthFrequency,
00358                             double rippleDb) {
00359                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00360                         BandPassBase::setup (reqOrder,
00361                                centerFrequency,
00362                                widthFrequency,
00363                                rippleDb);
00364                 }
00365 };
00366
00372 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00373         struct BandStop : PoleFilter <BandStopBase, StateType, FilterOrder, FilterOrder*2>
00374         {
00382                 void setup (double sampleRate,
00383                             double centerFrequency,
00384                             double widthFrequency,
00385                             double rippleDb) {
00386                         BandStopBase::setup (FilterOrder,
00387                                              centerFrequency / sampleRate,
00388                                              widthFrequency / sampleRate,
00389                                              rippleDb);
00390                 }
00391
00400                 void setup (int reqOrder,
00401                             double sampleRate,
00402                             double centerFrequency,
00403                             double widthFrequency,
00404                             double rippleDb) {
00405                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00406                         BandStopBase::setup (reqOrder,
00407                                              centerFrequency / sampleRate,
00408                                              widthFrequency / sampleRate,
00409                                              rippleDb);
00410                 }
00411
00412
00413
00420                 void setupN(double centerFrequency,
00421                             double widthFrequency,
00422                             double rippleDb) {
00423                         BandStopBase::setup (FilterOrder,
00424                                              centerFrequency,
00425                                              widthFrequency,
00426                                              rippleDb);
00427                 }
00428
00436                 void setupN(int reqOrder,
00437                             double centerFrequency,
00438                             double widthFrequency,
00439                             double rippleDb) {
00440                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00441                         BandStopBase::setup (reqOrder,
00442                                              centerFrequency,
00443                                              widthFrequency,
00444                                              rippleDb);
00445                 }
00446
00447 };
00448
00454 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00455         struct LowShelf : PoleFilter <LowShelfBase, StateType, FilterOrder>
00456         {
00464                 void setup (double sampleRate,
00465                             double cutoffFrequency,
00466                             double gainDb,
00467                             double rippleDb) {
00468                         LowShelfBase::setup (FilterOrder,
```

```
00469                                           cutoffFrequency / sampleRate,
00470                                           gainDb,
00471                                           rippleDb);
00472                   }
00473
00482                   void setup (int reqOrder,
00483                               double sampleRate,
00484                               double cutoffFrequency,
00485                               double gainDb,
00486                               double rippleDb) {
00487                           if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00488                           LowShelfBase::setup (reqOrder,
00489                                           cutoffFrequency / sampleRate,
00490                                           gainDb,
00491                                           rippleDb);
00492                   }
00493
00494
00495
00502                   void setupN(double cutoffFrequency,
00503                               double gainDb,
00504                               double rippleDb) {
00505                           LowShelfBase::setup (FilterOrder,
00506                                           cutoffFrequency,
00507                                           gainDb,
00508                                           rippleDb);
00509                   }
00510
00518                   void setupN(int reqOrder,
00519                               double cutoffFrequency,
00520                               double gainDb,
00521                               double rippleDb) {
00522                           if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00523                           LowShelfBase::setup (reqOrder,
00524                                           cutoffFrequency,
00525                                           gainDb,
00526                                           rippleDb);
00527                   }
00528 };
00529
00535 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00536         struct HighShelf : PoleFilter <HighShelfBase, StateType, FilterOrder>
00537         {
00545                   void setup (double sampleRate,
00546                               double cutoffFrequency,
00547                               double gainDb,
00548                               double rippleDb) {
00549                           HighShelfBase::setup (FilterOrder,
00550                                   cutoffFrequency / sampleRate,
00551                                   gainDb,
00552                                   rippleDb);
00553                   }
00554
00563                   void setup (int reqOrder,
00564                               double sampleRate,
00565                               double cutoffFrequency,
00566                               double gainDb,
00567                               double rippleDb) {
00568                           if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00569                           HighShelfBase::setup (reqOrder,
00570                                   cutoffFrequency / sampleRate,
00571                                   gainDb,
00572                                   rippleDb);
00573                   }
00574
00575
00576
00577
00584                   void setupN(double cutoffFrequency,
00585                               double gainDb,
00586                               double rippleDb) {
00587                           HighShelfBase::setup (FilterOrder,
00588                                   cutoffFrequency,
00589                                   gainDb,
00590                                   rippleDb);
00591                   }
00592
00600                   void setupN(int reqOrder,
00601                               double cutoffFrequency,
00602                               double gainDb,
00603                               double rippleDb) {
00604                           if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00605                           HighShelfBase::setup (reqOrder,
00606                                   cutoffFrequency,
00607                                   gainDb,
00608                                   rippleDb);
00609                   }
```

```
00610
00611 };
00612
00618 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00619         struct BandShelf : PoleFilter <BandShelfBase, StateType, FilterOrder, FilterOrder*2>
00620         {
00629                 void setup (double sampleRate,
00630                             double centerFrequency,
00631                             double widthFrequency,
00632                             double gainDb,
00633                             double rippleDb) {
00634                         BandShelfBase::setup (FilterOrder,
00635                                               centerFrequency / sampleRate,
00636                                               widthFrequency / sampleRate,
00637                                               gainDb,
00638                                               rippleDb);
00639
00640                 }
00641
00651                 void setup (int reqOrder,
00652                             double sampleRate,
00653                             double centerFrequency,
00654                             double widthFrequency,
00655                             double gainDb,
00656                             double rippleDb) {
00657                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00658                         BandShelfBase::setup (reqOrder,
00659                                               centerFrequency / sampleRate,
00660                                               widthFrequency / sampleRate,
00661                                               gainDb,
00662                                               rippleDb);
00663
00664                 }
00665
00666
00667
00668
00676                 void setupN(double centerFrequency,
00677                             double widthFrequency,
00678                             double gainDb,
00679                             double rippleDb) {
00680                         BandShelfBase::setup (FilterOrder,
00681                                               centerFrequency,
00682                                               widthFrequency,
00683                                               gainDb,
00684                                               rippleDb);
00685
00686                 }
00687
00696                 void setupN(int reqOrder,
00697                             double centerFrequency,
00698                             double widthFrequency,
00699                             double gainDb,
00700                             double rippleDb) {
00701                         if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00702                         BandShelfBase::setup (reqOrder,
00703                                               centerFrequency,
00704                                               widthFrequency,
00705                                               gainDb,
00706                                               rippleDb);
00707
00708                 }
00709
00710         };
00711
00712 }
00713
00714 }
```

8.6 ChebyshevII.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Cascade.h"
00040 #include "PoleFilter.h"
00041 #include "State.h"
00042
00043 namespace Iir {
00044
00052 namespace ChebyshevII {
00053
00057 class IIR_EXPORT AnalogLowPass : public LayoutBase
```

```
00058 {
00059 public:
00060         AnalogLowPass ();
00061 
00062         void design (const int numPoles,
00063                      double stopBandDb);
00064 
00065 private:
00066         int m_numPoles = 0;
00067         double m_stopBandDb = 0.0;
00068 };
00069 
00070 
00074 class IIR_EXPORT AnalogLowShelf : public LayoutBase
00075 {
00076 public:
00077         AnalogLowShelf ();
00078 
00079         void design (int numPoles,
00080                      double gainDb,
00081                      double stopBandDb);
00082 
00083 private:
00084         int m_numPoles = 0;
00085         double m_stopBandDb = 0.0;
00086         double m_gainDb = 0.0;
00087 };
00088 
00089 //------------------------------------------------------------------------------
00090 
00091 struct IIR_EXPORT LowPassBase : PoleFilterBase <AnalogLowPass>
00092 {
00093         void setup (int order,
00094                     double cutoffFrequency,
00095                     double stopBandDb);
00096 };
00097 
00098 struct IIR_EXPORT HighPassBase : PoleFilterBase <AnalogLowPass>
00099 {
00100         void setup (int order,
00101                     double cutoffFrequency,
00102                     double stopBandDb);
00103 };
00104 
00105 struct IIR_EXPORT BandPassBase : PoleFilterBase <AnalogLowPass>
00106 {
00107         void setup (int order,
00108                     double centerFrequency,
00109                     double widthFrequency,
00110                     double stopBandDb);
00111 };
00112 
00113 struct IIR_EXPORT BandStopBase : PoleFilterBase <AnalogLowPass>
00114 {
00115         void setup (int order,
00116                     double centerFrequency,
00117                     double widthFrequency,
00118                     double stopBandDb);
00119 };
00120 
00121 struct IIR_EXPORT LowShelfBase : PoleFilterBase <AnalogLowShelf>
00122 {
00123         void setup (int order,
00124                     double cutoffFrequency,
00125                     double gainDb,
00126                     double stopBandDb);
00127 };
00128 
00129 struct IIR_EXPORT HighShelfBase : PoleFilterBase <AnalogLowShelf>
00130 {
00131         void setup (int order,
00132                     double cutoffFrequency,
00133                     double gainDb,
00134                     double stopBandDb);
00135 };
00136 
00137 struct IIR_EXPORT BandShelfBase : PoleFilterBase <AnalogLowShelf>
00138 {
00139         void setup (int order,
00140                     double centerFrequency,
00141                     double widthFrequency,
00142                     double gainDb,
00143                     double stopBandDb);
00144 };
00145 
00146 //------------------------------------------------------------------------------
00147 
```

```
00148 //
00149 // Userland filters
00150 //
00151
00157 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00158 struct LowPass : PoleFilter <LowPassBase, StateType, FilterOrder>
00159 {
00166         void setup (double sampleRate,
00167                     double cutoffFrequency,
00168                     double stopBandDb) {
00169                 LowPassBase::setup (FilterOrder,
00170                                     cutoffFrequency / sampleRate,
00171                                     stopBandDb);
00172         }
00173
00181         void setup (int reqOrder,
00182                     double sampleRate,
00183                     double cutoffFrequency,
00184                     double stopBandDb) {
00185                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00186                 LowPassBase::setup (reqOrder,
00187                                     cutoffFrequency / sampleRate,
00188                                     stopBandDb);
00189         }
00190
00191
00192
00193
00194
00200         void setupN(double cutoffFrequency,
00201                     double stopBandDb) {
00202                 LowPassBase::setup (FilterOrder,
00203                                     cutoffFrequency,
00204                                     stopBandDb);
00205         }
00206
00213         void setupN(int reqOrder,
00214                     double cutoffFrequency,
00215                     double stopBandDb) {
00216                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00217                 LowPassBase::setup (reqOrder,
00218                                     cutoffFrequency,
00219                                     stopBandDb);
00220         }
00221
00222 };
00223
00229 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00230 struct HighPass : PoleFilter <HighPassBase, StateType, FilterOrder>
00231 {
00238         void setup (double sampleRate,
00239                     double cutoffFrequency,
00240                     double stopBandDb) {
00241                 HighPassBase::setup (FilterOrder,
00242                                      cutoffFrequency / sampleRate,
00243                                      stopBandDb);
00244         }
00245
00253         void setup (int reqOrder,
00254                     double sampleRate,
00255                     double cutoffFrequency,
00256                     double stopBandDb) {
00257                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00258                 HighPassBase::setup (reqOrder,
00259                                      cutoffFrequency / sampleRate,
00260                                      stopBandDb);
00261         }
00262
00263
00264
00265
00271         void setupN(double cutoffFrequency,
00272                     double stopBandDb) {
00273                 HighPassBase::setup (FilterOrder,
00274                                      cutoffFrequency,
00275                                      stopBandDb);
00276         }
00277
00284         void setupN(int reqOrder,
00285                     double cutoffFrequency,
00286                     double stopBandDb) {
00287                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00288                 HighPassBase::setup (reqOrder,
00289                                      cutoffFrequency,
00290                                      stopBandDb);
00291         }
00292
```

```
00293 };
00294
00300 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00301 struct BandPass : PoleFilter <BandPassBase, StateType, FilterOrder, FilterOrder*2>
00302 {
00310         void setup (double sampleRate,
00311                     double centerFrequency,
00312                     double widthFrequency,
00313                     double stopBandDb) {
00314                 BandPassBase::setup (FilterOrder,
00315                                      centerFrequency / sampleRate,
00316                                      widthFrequency / sampleRate,
00317                                      stopBandDb);
00318         }
00319
00328         void setup (int reqOrder,
00329                     double sampleRate,
00330                     double centerFrequency,
00331                     double widthFrequency,
00332                     double stopBandDb) {
00333                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00334                 BandPassBase::setup (reqOrder,
00335                                      centerFrequency / sampleRate,
00336                                      widthFrequency / sampleRate,
00337                                      stopBandDb);
00338         }
00339
00340
00341
00342
00349         void setupN(double centerFrequency,
00350                     double widthFrequency,
00351                     double stopBandDb) {
00352                 BandPassBase::setup (FilterOrder,
00353                                      centerFrequency,
00354                                      widthFrequency,
00355                                      stopBandDb);
00356         }
00357
00365         void setupN(int reqOrder,
00366                     double centerFrequency,
00367                     double widthFrequency,
00368                     double stopBandDb) {
00369                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00370                 BandPassBase::setup (reqOrder,
00371                                      centerFrequency,
00372                                      widthFrequency,
00373                                      stopBandDb);
00374         }
00375 };
00376
00382 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00383 struct BandStop : PoleFilter <BandStopBase, StateType, FilterOrder, FilterOrder*2>
00384 {
00392         void setup (double sampleRate,
00393                     double centerFrequency,
00394                     double widthFrequency,
00395                     double stopBandDb) {
00396                 BandStopBase::setup (FilterOrder,
00397                                      centerFrequency / sampleRate,
00398                                      widthFrequency / sampleRate,
00399                                      stopBandDb);
00400         }
00401
00410         void setup (int reqOrder,
00411                     double sampleRate,
00412                     double centerFrequency,
00413                     double widthFrequency,
00414                     double stopBandDb) {
00415                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00416                 BandStopBase::setup (reqOrder,
00417                                      centerFrequency / sampleRate,
00418                                      widthFrequency / sampleRate,
00419                                      stopBandDb);
00420         }
00421
00422
00423
00424
00431         void setupN(double centerFrequency,
00432                     double widthFrequency,
00433                     double stopBandDb) {
00434                 BandStopBase::setup (FilterOrder,
00435                                      centerFrequency,
00436                                      widthFrequency,
00437                                      stopBandDb);
00438         }
```

```
00439
00447         void setupN(int reqOrder,
00448                     double centerFrequency,
00449                     double widthFrequency,
00450                     double stopBandDb) {
00451                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00452                 BandStopBase::setup (reqOrder,
00453                                      centerFrequency,
00454                                      widthFrequency,
00455                                      stopBandDb);
00456         }
00457 };
00458
00464 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00465 struct LowShelf : PoleFilter <LowShelfBase, StateType, FilterOrder>
00466 {
00474         void setup (double sampleRate,
00475                     double cutoffFrequency,
00476                     double gainDb,
00477                     double stopBandDb) {
00478                 LowShelfBase::setup (FilterOrder,
00479                                      cutoffFrequency / sampleRate,
00480                                      gainDb,
00481                                      stopBandDb);
00482         }
00483
00492         void setup (int reqOrder,
00493                     double sampleRate,
00494                     double cutoffFrequency,
00495                     double gainDb,
00496                     double stopBandDb) {
00497                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00498                 LowShelfBase::setup (reqOrder,
00499                                      cutoffFrequency / sampleRate,
00500                                      gainDb,
00501                                      stopBandDb);
00502         }
00503
00504
00505
00506
00507
00514         void setupN(double cutoffFrequency,
00515                     double gainDb,
00516                     double stopBandDb) {
00517                 LowShelfBase::setup (FilterOrder,
00518                                      cutoffFrequency,
00519                                      gainDb,
00520                                      stopBandDb);
00521         }
00522
00530         void setupN(int reqOrder,
00531                     double cutoffFrequency,
00532                     double gainDb,
00533                     double stopBandDb) {
00534                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00535                 LowShelfBase::setup (reqOrder,
00536                                      cutoffFrequency,
00537                                      gainDb,
00538                                      stopBandDb);
00539         }
00540
00541 };
00542
00548 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00549 struct HighShelf : PoleFilter <HighShelfBase, StateType, FilterOrder>
00550 {
00558         void setup (double sampleRate,
00559                     double cutoffFrequency,
00560                     double gainDb,
00561                     double stopBandDb) {
00562                 HighShelfBase::setup (FilterOrder,
00563                                       cutoffFrequency / sampleRate,
00564                                       gainDb,
00565                                       stopBandDb);
00566         }
00567
00576         void setup (int reqOrder,
00577                     double sampleRate,
00578                     double cutoffFrequency,
00579                     double gainDb,
00580                     double stopBandDb) {
00581                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00582                 HighShelfBase::setup (reqOrder,
00583                                       cutoffFrequency / sampleRate,
00584                                       gainDb,
00585                                       stopBandDb);
```

```
00586         }
00587
00588
00589
00590
00591
00592
00599         void setupN(double cutoffFrequency,
00600                     double gainDb,
00601                     double stopBandDb) {
00602                 HighShelfBase::setup (FilterOrder,
00603                                       cutoffFrequency,
00604                                       gainDb,
00605                                       stopBandDb);
00606         }
00607
00615         void setupN(int reqOrder,
00616                     double cutoffFrequency,
00617                     double gainDb,
00618                     double stopBandDb) {
00619                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00620                 HighShelfBase::setup (reqOrder,
00621                                       cutoffFrequency,
00622                                       gainDb,
00623                                       stopBandDb);
00624         }
00625
00626 };
00627
00633 template <int FilterOrder = DEFAULT_FILTER_ORDER, class StateType = DEFAULT_STATE>
00634 struct BandShelf : PoleFilter <BandShelfBase, StateType, FilterOrder, FilterOrder*2>
00635 {
00644         void setup (double sampleRate,
00645                     double centerFrequency,
00646                     double widthFrequency,
00647                     double gainDb,
00648                     double stopBandDb) {
00649                 BandShelfBase::setup (FilterOrder,
00650                                       centerFrequency / sampleRate,
00651                                       widthFrequency / sampleRate,
00652                                       gainDb,
00653                                       stopBandDb);
00654         }
00655
00656
00666         void setup (int reqOrder,
00667                     double sampleRate,
00668                     double centerFrequency,
00669                     double widthFrequency,
00670                     double gainDb,
00671                     double stopBandDb) {
00672                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00673                 BandShelfBase::setup (reqOrder,
00674                                       centerFrequency / sampleRate,
00675                                       widthFrequency / sampleRate,
00676                                       gainDb,
00677                                       stopBandDb);
00678         }
00679
00680
00681
00682
00683
00684
00685
00693         void setupN(double centerFrequency,
00694                     double widthFrequency,
00695                     double gainDb,
00696                     double stopBandDb) {
00697                 BandShelfBase::setup (FilterOrder,
00698                                       centerFrequency,
00699                                       widthFrequency,
00700                                       gainDb,
00701                                       stopBandDb);
00702         }
00703
00704
00713         void setupN(int reqOrder,
00714                     double centerFrequency,
00715                     double widthFrequency,
00716                     double gainDb,
00717                     double stopBandDb) {
00718                 if (reqOrder > FilterOrder) throw_invalid_argument(orderTooHigh);
00719                 BandShelfBase::setup (reqOrder,
00720                                       centerFrequency,
00721                                       widthFrequency,
00722                                       gainDb,
```

```
00723                                       stopBandDb);
00724         }
00725
00726
00727 };
00728
00729 }
00730
00731 }
```

8.7 Common.h

```
00001
00035
00036  #pragma once
00037
00038 //
00039 // This must be the first file included in every DspFilters header and source
00040 //
00041
00042 #ifdef _MSC_VER
00043 #  pragma warning (disable: 4100)
00044 #  ifndef _CRT_SECURE_NO_WARNINGS
00045 #    define _CRT_SECURE_NO_WARNINGS
00046 #  endif
00047 #endif
00048
00049 // This exports the classes/structures to the windows DLL
00050 #if defined(_WIN32) && defined(iir_EXPORTS)
00051 #define IIR_EXPORT __declspec( dllexport )
00052 #else
00053 #define IIR_EXPORT
00054 #endif
00055
00056 #include <stdlib.h>
00057
00058 #include <cassert>
00059 #include <cfloat>
00060 #include <cmath>
00061 #include <complex>
00062 #include <cstring>
00063 #include <string>
00064 #include <limits>
00065 #include <vector>
00066 #include <array>
00067 #include <stdexcept> // for invalid_argument
00068
00069 static const char orderTooHigh[] = "Requested order is too high. Provide a higher order for the
      template.";
00070
00071 #define DEFAULT_FILTER_ORDER 4
00072
00078 inline void throw_invalid_argument(const char* msg) {
00079
00080 #ifndef IIR1_NO_EXCEPTIONS
00081     throw std::invalid_argument(msg);
00082 #else
00083     (void) msg; // Discard parameter
00084     abort();
00085 #endif
00086
00087 }
```

8.8 Custom.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Biquad.h"
00039 #include "Cascade.h"
00040 #include "State.h"
00041
00042 namespace Iir
00043 {
00044
00049
00050 namespace Custom
00051 {
00052
00059 struct OnePole : public Biquad
00060 {
00061     void setup (double scale, double pole, double zero);
```

```
00062 };
00063
00071 struct TwoPole : public Biquad
00072 {
00073     void setup (double scale, double poleRho, double poleTheta, double zeroRho,
00074                 double zeroTheta);
00075 };
00076
00082 template <std::size_t NSOS, class StateType = DEFAULT_STATE>
00083 struct SOSCascade : CascadeStages<NSOS, StateType>
00084 {
00089     SOSCascade () = default;
00090
00101     SOSCascade (const double (&sosCoefficients)[NSOS][6])
00102     {
00103         CascadeStages<NSOS, StateType>::setup (sosCoefficients);
00104     }
00105
00116     SOSCascade (const std::array<std::array<double, 6>, NSOS> sosCoefficients)
00117     {
00118         CascadeStages<NSOS, StateType>::setup (sosCoefficients);
00119     }
00120
00131     void setup (const double (&sosCoefficients)[NSOS][6])
00132     {
00133         CascadeStages<NSOS, StateType>::setup (sosCoefficients);
00134     }
00135
00146     void setup (const std::array<std::array<double, 6>, NSOS> sosCoefficients)
00147     {
00148         CascadeStages<NSOS, StateType>::setup (sosCoefficients);
00149     }
00150 };
00151
00152 }
00153
00154 }
```

8.9 Layout.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "MathSupplement.h"
00040
00046 namespace Iir {
00047
00048         static const char errPoleisNaN[] = "Pole to add is NaN.";
00049         static const char errZeroisNaN[] = "Zero to add is NaN.";
00050
00051         static const char errCantAdd2ndOrder[] = "Can't add 2nd order after a 1st order filter.";
00052
00053         static const char errPolesNotComplexConj[] = "Poles not complex conjugate.";
00054         static const char errZerosNotComplexConj[] = "Zeros not complex conjugate.";
00055
00056         static const char pairIndexOutOfBounds[] = "Pair index out of bounds.";
00057
00061         class IIR_EXPORT LayoutBase
00062         {
00063         public:
00064                 LayoutBase ()
00065                         : m_numPoles (0)
00066                         , m_maxPoles (0)
00067                         , m_pair (nullptr)
00068                 {
00069                 }
00070
00071                 LayoutBase (int maxPoles, PoleZeroPair* pairs)
00072                         : m_numPoles (0)
00073                         , m_maxPoles (maxPoles)
00074                         , m_pair (pairs)
00075                 {
00076                 }
00077
00078                 void setStorage (const LayoutBase& other)
00079                 {
00080                         m_numPoles = 0;
00081                         m_maxPoles = other.m_maxPoles;
00082                         m_pair = other.m_pair;
00083                 }
00084
00085                 void reset ()
00086                 {
```

```
00087                           m_numPoles = 0;
00088                   }
00089
00090                   int getNumPoles () const
00091                   {
00092                           return m_numPoles;
00093                   }
00094
00095                   int getMaxPoles () const
00096                   {
00097                           return m_maxPoles;
00098                   }
00099
00100                  void add (const complex_t& pole, const complex_t& zero)
00101                  {
00102                          if (m_numPoles&1)
00103                                  throw_invalid_argument(errCantAdd2ndOrder);
00104                          if (Iir::is_nan(pole))
00105                                  throw_invalid_argument(errPoleisNaN);
00106                          if (Iir::is_nan(zero))
00107                                  throw_invalid_argument(errZeroisNaN);
00108                          m_pair[m_numPoles/2] = PoleZeroPair (pole, zero);
00109                          ++m_numPoles;
00110                  }
00111
00112                  void addPoleZeroConjugatePairs (const complex_t& pole,
00113                                                  const complex_t& zero)
00114                  {
00115                          if (m_numPoles&1)
00116                                  throw_invalid_argument(errCantAdd2ndOrder);
00117                          if (Iir::is_nan(pole))
00118                                  throw_invalid_argument(errPoleisNaN);
00119                          if (Iir::is_nan(zero))
00120                                  throw_invalid_argument(errZeroisNaN);
00121                          m_pair[m_numPoles/2] = PoleZeroPair (
00122                                  pole, zero, std::conj (pole), std::conj (zero));
00123                          m_numPoles += 2;
00124                  }
00125
00126                  void add (const ComplexPair& poles, const ComplexPair& zeros)
00127                  {
00128                          if (m_numPoles&1)
00129                                  throw_invalid_argument(errCantAdd2ndOrder);
00130                          if (!poles.isMatchedPair ())
00131                                  throw_invalid_argument(errPolesNotComplexConj);
00132                          if (!zeros.isMatchedPair ())
00133                                  throw_invalid_argument(errZerosNotComplexConj);
00134                          m_pair[m_numPoles/2] = PoleZeroPair (poles.first, zeros.first,
00135                                                               poles.second, zeros.second);
00136                          m_numPoles += 2;
00137                  }
00138
00139                  const PoleZeroPair& getPair (int pairIndex) const
00140                  {
00141                          if ((pairIndex < 0) || (pairIndex >= (m_numPoles+1)/2))
00142                                  throw_invalid_argument(pairIndexOutOfBounds);
00143                          return m_pair[pairIndex];
00144                  }
00145
00146                  const PoleZeroPair& operator[] (int pairIndex) const
00147                  {
00148                          return getPair (pairIndex);
00149                  }
00150
00151                  double getNormalW () const
00152                  {
00153                          return m_normalW;
00154                  }
00155
00156                  double getNormalGain () const
00157                  {
00158                          return m_normalGain;
00159                  }
00160
00161                  void setNormal (double w, double g)
00162                  {
00163                          m_normalW = w;
00164                          m_normalGain = g;
00165                  }
00166
00167          private:
00168                  int m_numPoles = 0;
00169                  int m_maxPoles = 0;
00170                  PoleZeroPair* m_pair = nullptr;
00171                  double m_normalW = 0;
00172                  double m_normalGain = 1;
00173          };
```

```
00174
00175 //------------------------------------------------------------------------
00176
00180         template <int MaxPoles>
00181                 class Layout
00182         {
00183         public:
00184                 operator LayoutBase ()
00185                 {
00186                         return LayoutBase (MaxPoles, m_pairs);
00187                 }
00188
00189         private:
00190                 PoleZeroPair m_pairs[(MaxPoles+1)/2] = {};
00191         };
00192
00193 }
```

8.10 MathSupplement.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039
00040 #include<complex>
00041
00042 #ifdef _MSC_VER
00043  // Under Unix these have already default instantiations but not under Vis Studio
00044 template class IIR_EXPORT std::complex<double>;
00045 template class IIR_EXPORT std::complex<float>;
00046 #endif
00047
00048 namespace Iir {
00049
00050 const double doublePi   =3.14159265358979323846264338327950288419 71;
00051 const double doublePi_2 =1.57079632679489661923132169163975144209 86;
00052 const double doubleLn2  =0.69314718055994530941723212145818;
00053 const double doubleLn10 =2.30258509299404568401799145468 44;
00054
00055 typedef std::complex<double> complex_t;
00056 typedef std::pair<complex_t, complex_t> complex_pair_t;
00057
00058 inline const complex_t infinity()
00059 {
00060   return complex_t (std::numeric_limits<double>::infinity());
00061 }
00062
00063 template <typename Ty, typename To>
00064 inline std::complex<Ty> addmul (const std::complex<Ty>& c,
00065                                 Ty v,
00066                                 const std::complex<To>& c1)
00067 {
00068   return std::complex <Ty> (
00069     c.real() + v * c1.real(), c.imag() + v * c1.imag());
00070 }
00071
00072 template <typename Ty>
00073 inline Ty asinh (Ty x)
00074 {
00075   return log (x + std::sqrt (x * x + 1 ));
00076 }
00077
00078 template <typename Ty>
00079 inline bool is_nan (Ty v)
00080 {
00081   return !(v == v);
00082 }
00083
00084 template <>
00085 inline bool is_nan<complex_t> (complex_t v)
00086 {
00087   return Iir::is_nan (v.real()) || Iir::is_nan (v.imag());
00088 }
00089
00090 }
```

8.11 PoleFilter.h

```
00001
00035
00036 #pragma once
```

```
00037
00038 #include "Common.h"
00039 #include "MathSupplement.h"
00040 #include "Cascade.h"
00041
00042 namespace Iir {
00043
00044 /***
00045  * Base for filters designed via algorithmic placement of poles and zeros.
00046  *
00047  * Typically, the filter is first designed as a half-band low pass or
00048  * low shelf analog filter (s-plane). Then, using a transformation such
00049  * as the ones from Constantinides, the poles and zeros of the analog filter
00050  * are calculated in the z-plane.
00051  *
00052  ***/
00053
00057         class IIR_EXPORT PoleFilterBase2 : public Cascade
00058         {
00059         public:
00060                 // This gets the poles/zeros directly from the digital
00061                 // prototype. It is used to double check the correctness
00062                 // of the recovery of pole/zeros from biquad coefficients.
00063                 //
00064                 // It can also be used to accelerate the interpolation
00065                 // of pole/zeros for parameter modulation, since a pole
00066                 // filter already has them calculated
00067
00068                 PoleFilterBase2() = default;
00069
00070                 std::vector<PoleZeroPair> getPoleZeros () const
00071                 {
00072                         std::vector<PoleZeroPair> vpz;
00073                         const int pairs = (m_digitalProto.getNumPoles () + 1) / 2;
00074                         for (int i = 0; i < pairs; ++i)
00075                                 vpz.push_back (m_digitalProto[i]);
00076                         return vpz;
00077                 }
00078
00079         protected:
00080                 LayoutBase m_digitalProto = {};
00081         };
00082
00083
00088         template <class AnalogPrototype>
00089         class PoleFilterBase : public PoleFilterBase2
00090         {
00091         protected:
00092                 void setPrototypeStorage (const LayoutBase& analogStorage,
00093                                           const LayoutBase& digitalStorage)
00094                 {
00095                         m_analogProto.setStorage (analogStorage);
00096                         m_digitalProto = digitalStorage;
00097                 }
00098
00099         protected:
00100                 AnalogPrototype m_analogProto = {};
00101         };
00102
00103 //------------------------------------------------------------------------------
00104
00108         template <class BaseClass,
00109                   class StateType,
00110                   int MaxAnalogPoles,
00111                   int MaxDigitalPoles = MaxAnalogPoles>
00112         struct PoleFilter : BaseClass
00113                 , CascadeStages <(MaxDigitalPoles + 1) / 2 , StateType>
00114         {
00115         public:
00116                 PoleFilter ()
00117                         {
00118                                 // This glues together the factored base classes
00119                                 // with the templatized storage classes.
00120                                 BaseClass::setCascadeStorage (this->getCascadeStorage());
00121                                 BaseClass::setPrototypeStorage (m_analogStorage, m_digitalStorage);
00122                                 CascadeStages<(MaxDigitalPoles + 1) / 2 , StateType>::reset();
00123                         }
00124
00125                 PoleFilter(const PoleFilter&) = default;
00126
00127                 PoleFilter& operator=(const PoleFilter&)
00128                         {
00129                                 // Reset the filter state when copied for now
00130                                 CascadeStages<(MaxDigitalPoles + 1) / 2 , StateType>::reset();
00131                                 return *this;
00132                         }
00133
```

```
00134         private:
00135                 Layout <MaxAnalogPoles> m_analogStorage = {};
00136                 Layout <MaxDigitalPoles> m_digitalStorage = {};
00137         };
00138
00139 //------------------------------------------------------------------------------
00140
00151
00155         class IIR_EXPORT LowPassTransform
00156         {
00157         public:
00158         LowPassTransform (double fc,
00159                           LayoutBase& digital,
00160                           LayoutBase const& analog);
00161
00162         private:
00163         complex_t transform (complex_t c);
00164
00165         double f = 0.0;
00166         };
00167
00168 //------------------------------------------------------------------------------
00169
00173         class IIR_EXPORT HighPassTransform
00174         {
00175         public:
00176         HighPassTransform (double fc,
00177                            LayoutBase& digital,
00178                            LayoutBase const& analog);
00179
00180         private:
00181         complex_t transform (complex_t c);
00182
00183         double f = 0.0;
00184         };
00185
00186 //------------------------------------------------------------------------------
00187
00191         class IIR_EXPORT BandPassTransform
00192         {
00193
00194         public:
00195         BandPassTransform (double fc,
00196                            double fw,
00197                            LayoutBase& digital,
00198                            LayoutBase const& analog);
00199
00200         private:
00201         ComplexPair transform (complex_t c);
00202
00203         double wc = 0.0;
00204         double wc2 = 0.0;
00205         double a = 0.0;
00206         double b = 0.0;
00207         double a2 = 0.0;
00208         double b2 = 0.0;
00209         double ab = 0.0;
00210         double ab_2 = 0.0;
00211         };
00212
00213 //------------------------------------------------------------------------------
00214
00218         class IIR_EXPORT BandStopTransform
00219         {
00220         public:
00221         BandStopTransform (double fc,
00222                            double fw,
00223                            LayoutBase& digital,
00224                            LayoutBase const& analog);
00225
00226         private:
00227         ComplexPair transform (complex_t c);
00228
00229         double wc = 0.0;
00230         double wc2 = 0.0;
00231         double a = 0.0;
00232         double b = 0.0;
00233         double a2 = 0.0;
00234         double b2 = 0.0;
00235         };
00236
00237 }
```

8.12 RBJ.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Biquad.h"
00040 #include "State.h"
00041
00042 namespace Iir {
00043
00057
00058 #define ONESQRT2 (1/sqrt(2))
00059
00060 namespace RBJ {
00061
00065         struct IIR_EXPORT RBJbase : Biquad
00066         {
00067         public:
00069                 template <typename Sample>
00070                         inline Sample filter(Sample s) {
00071                         return static_cast<Sample>(state.filter((double)s,*this));
00072                 }
00074                 void reset() {
00075                         state.reset();
00076                 }
00078                 const DirectFormI& getState() {
00079                         return state;
00080                 }
00081         private:
00082                 DirectFormI state = {};
00083         };
00084
00088         struct IIR_EXPORT LowPass : RBJbase
00089         {
00095                 void setupN(double cutoffFrequency,
00096                             double q = ONESQRT2);
00097
00104                 void setup(double sampleRate,
00105                            double cutoffFrequency,
00106                            double q = ONESQRT2) {
00107                         setupN(cutoffFrequency / sampleRate, q);
00108                 }
00109         };
00110
00114         struct IIR_EXPORT HighPass : RBJbase
00115         {
00121                 void setupN(double cutoffFrequency,
00122                             double q = ONESQRT2);
00129                 void setup (double sampleRate,
00130                             double cutoffFrequency,
00131                             double q = ONESQRT2) {
00132                         setupN(cutoffFrequency / sampleRate, q);
00133                 }
00134         };
00135
00139         struct IIR_EXPORT BandPass1 : RBJbase
00140         {
00146                 void setupN(double centerFrequency,
00147                             double bandWidth);
00154                 void setup (double sampleRate,
00155                             double centerFrequency,
00156                             double bandWidth) {
00157                         setupN(centerFrequency / sampleRate, bandWidth);
00158                 }
00159         };
00160
00164         struct IIR_EXPORT BandPass2 : RBJbase
00165         {
00171                 void setupN(double centerFrequency,
00172                             double bandWidth);
00179                 void setup (double sampleRate,
00180                             double centerFrequency,
00181                             double bandWidth) {
00182                         setupN(centerFrequency / sampleRate, bandWidth);
00183                 }
00184         };
00185
00190         struct IIR_EXPORT BandStop : RBJbase
00191         {
00197                 void setupN(double centerFrequency,
00198                             double bandWidth);
00205                 void setup (double sampleRate,
00206                             double centerFrequency,
00207                             double bandWidth) {
00208                         setupN(centerFrequency / sampleRate, bandWidth);
```

```
00209                 }
00210         };
00211
00223         struct IIR_EXPORT IIRNotch : RBJbase
00224         {
00230                 void setupN(double centerFrequency,
00231                             double q_factor = 10);
00238                 void setup (double sampleRate,
00239                             double centerFrequency,
00240                             double q_factor = 10) {
00241                         setupN(centerFrequency / sampleRate, q_factor);
00242                 }
00243         };
00244
00248         struct IIR_EXPORT LowShelf : RBJbase
00249         {
00256                 void setupN(double cutoffFrequency,
00257                             double gainDb,
00258                             double shelfSlope = 1);
00266                 void setup (double sampleRate,
00267                             double cutoffFrequency,
00268                             double gainDb,
00269                             double shelfSlope = 1) {
00270                         setupN( cutoffFrequency / sampleRate, gainDb, shelfSlope);
00271                 }
00272         };
00273
00277         struct IIR_EXPORT HighShelf : RBJbase
00278         {
00285                 void setupN(double cutoffFrequency,
00286                             double gainDb,
00287                             double shelfSlope = 1);
00295                 void setup (double sampleRate,
00296                             double cutoffFrequency,
00297                             double gainDb,
00298                             double shelfSlope = 1) {
00299                         setupN( cutoffFrequency / sampleRate, gainDb, shelfSlope);
00300                 }
00301         };
00302
00306         struct IIR_EXPORT BandShelf : RBJbase
00307         {
00314                 void setupN(double centerFrequency,
00315                             double gainDb,
00316                             double bandWidth);
00324                 void setup (double sampleRate,
00325                             double centerFrequency,
00326                             double gainDb,
00327                             double bandWidth) {
00328                         setupN(centerFrequency / sampleRate, gainDb, bandWidth);
00329                 }
00330         };
00331
00335         struct IIR_EXPORT AllPass : RBJbase
00336         {
00342                 void setupN(double phaseFrequency,
00343                             double q  = ONESQRT2);
00344
00351                 void setup (double sampleRate,
00352                             double phaseFrequency,
00353                             double q  = ONESQRT2) {
00354                         setupN( phaseFrequency / sampleRate, q);
00355                 }
00356         };
00357
00358 }
00359
00360 }
```

8.13 State.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "Biquad.h"
00040
00041
00042 #define DEFAULT_STATE DirectFormII
00043
00044 namespace Iir {
00045
00054         class IIR_EXPORT DirectFormI
00055         {
```

```
00056         public:
00057         DirectFormI () = default;
00058
00059         void reset ()
00060         {
00061                 m_x1 = 0;
00062                 m_x2 = 0;
00063                 m_y1 = 0;
00064                 m_y2 = 0;
00065         }
00066
00067         inline double filter(const double in,
00068                              const Biquad& s)
00069         {
00070                 const double out = s.m_b0*in + s.m_b1*m_x1 + s.m_b2*m_x2
00071                 - s.m_a1*m_y1 - s.m_a2*m_y2;
00072                 m_x2 = m_x1;
00073                 m_y2 = m_y1;
00074                 m_x1 = in;
00075                 m_y1 = out;
00076
00077                 return out;
00078         }
00079
00080         protected:
00081         double m_x2 = 0.0; // x[n-2]
00082         double m_y2 = 0.0; // y[n-2]
00083         double m_x1 = 0.0; // x[n-1]
00084         double m_y1 = 0.0; // y[n-1]
00085         };
00086
00087 //-----------------------------------------------------------------------------
00088
00098         class IIR_EXPORT DirectFormII
00099         {
00100         public:
00101         DirectFormII () = default;
00102
00103         void reset ()
00104         {
00105                 m_v1 = 0.0;
00106                 m_v2 = 0.0;
00107         }
00108
00109         inline double filter(const double in,
00110                              const Biquad& s)
00111         {
00112                 const double w   = in - s.m_a1*m_v1 - s.m_a2*m_v2;
00113                 const double out =      s.m_b0*w    + s.m_b1*m_v1 + s.m_b2*m_v2;
00114
00115                 m_v2 = m_v1;
00116                 m_v1 = w;
00117
00118                 return out;
00119         }
00120
00121         private:
00122         double m_v1 = 0.0; // v[-1]
00123         double m_v2 = 0.0; // v[-2]
00124         };
00125
00126
00127 //-----------------------------------------------------------------------------
00128
00129         class IIR_EXPORT TransposedDirectFormII
00130         {
00131         public:
00132         TransposedDirectFormII() = default;
00133
00134         void reset ()
00135         {
00136                 m_s1 = 0.0;
00137                 m_s1_1 = 0.0;
00138                 m_s2 = 0.0;
00139                 m_s2_1 = 0.0;
00140         }
00141
00142         inline double filter(const double in,
00143                              const Biquad& s)
00144         {
00145                 const double out = m_s1_1 + s.m_b0*in;
00146                 m_s1 = m_s2_1 + s.m_b1*in - s.m_a1*out;
00147                 m_s2 = s.m_b2*in - s.m_a2*out;
00148                 m_s1_1 = m_s1;
00149                 m_s2_1 = m_s2;
00150
00151                 return out;
```

```
00152         }
00153
00154         private:
00155         double m_s1 = 0.0;
00156         double m_s1_1 = 0.0;
00157         double m_s2 = 0.0;
00158         double m_s2_1 = 0.0;
00159         };
00160
00161 }
```

8.14 Types.h

```
00001
00035
00036 #pragma once
00037
00038 #include "Common.h"
00039 #include "MathSupplement.h"
00040
00041 namespace Iir {
00042
00046         struct IIR_EXPORT ComplexPair : complex_pair_t
00047         {
00048                 ComplexPair() = default;
00049
00050                 explicit ComplexPair (const complex_t& c1)
00051                         : complex_pair_t (c1, 0.)
00052                 {
00053                         if (!isReal()) throw_invalid_argument("A single complex number needs to be
      real.");
00054                 }
00055
00056                 ComplexPair (const complex_t& c1,
00057                              const complex_t& c2)
00058                         : complex_pair_t (c1, c2)
00059                 {
00060                 }
00061
00062                 bool isReal () const
00063                 {
00064                         return first.imag() == 0 && second.imag() == 0;
00065                 }
00066
00071                 bool isMatchedPair () const
00072                 {
00073                         if (first.imag() != 0)
00074                                 return second == std::conj (first);
00075                         else
00076                                 return second.imag () == 0 &&
00077                                         second.real () != 0 &&
00078                                         first.real () != 0;
00079                 }
00080
00081                 bool is_nan () const
00082                 {
00083                         return Iir::is_nan (first) || Iir::is_nan (second);
00084                 }
00085         };
00086
00087
00091         struct IIR_EXPORT PoleZeroPair
00092         {
00093                 ComplexPair poles = ComplexPair();
00094                 ComplexPair zeros = ComplexPair();
00095
00096                 PoleZeroPair () = default;
00097
00098                 // single pole/zero
00099                 PoleZeroPair (const complex_t& p, const complex_t& z)
00100                         : poles (p), zeros (z)
00101                 {
00102                 }
00103
00104                 // pole/zero pair
00105                 PoleZeroPair (const complex_t& p1, const complex_t& z1,
00106                               const complex_t& p2, const complex_t& z2)
00107                         : poles (p1, p2)
00108                         , zeros (z1, z2)
00109                 {
00110                 }
00111
00112                 bool isSinglePole () const
00113                 {
00114                         return poles.second == 0. && zeros.second == 0.;
```

```
00115                 }
00116
00117                 bool is_nan () const
00118                 {
00119                         return poles.is_nan() || zeros.is_nan();
00120                 }
00121         };
00122
00123
00124 }
```

Index

applyScale
  Iir::Biquad, 47

DSP IIR Realtime C++ filter library, 1

filter
  Iir::Biquad, 47
  Iir::CascadeStages< MaxStages, StateType >, 51

getA0
  Iir::Biquad, 47
getA1
  Iir::Biquad, 47
getA2
  Iir::Biquad, 47
getB0
  Iir::Biquad, 48
getB1
  Iir::Biquad, 48
getB2
  Iir::Biquad, 48
getCascadeStorage
  Iir::CascadeStages< MaxStages, StateType >, 51
getNumStages
  Iir::Cascade, 50
getPoleZeros
  Iir::Biquad, 48
  Iir::Cascade, 50

Iir, 8
iir/Biquad.h, 99
iir/Butterworth.h, 100
iir/Cascade.h, 105
iir/ChebyshevI.h, 106
iir/ChebyshevII.h, 111
iir/Common.h, 117
iir/Custom.h, 117
iir/Layout.h, 118
iir/MathSupplement.h, 120
iir/PoleFilter.h, 120
iir/RBJ.h, 123
iir/State.h, 124
iir/Types.h, 126
Iir::BandPassTransform, 26
Iir::BandStopTransform, 46
Iir::Biquad, 46
  applyScale, 47
  filter, 47
  getA0, 47
  getA1, 47
  getA2, 47
  getB0, 48
  getB1, 48
  getB2, 48
  getPoleZeros, 48
  response, 48
  setCoefficients, 48
  setIdentity, 49
  setOnePole, 49
  setPoleZeroPair, 49
  setTwoPole, 49
Iir::BiquadPoleState, 49
Iir::Butterworth, 11
Iir::Butterworth::AnalogLowPass, 14
Iir::Butterworth::AnalogLowShelf, 15
Iir::Butterworth::BandPass< FilterOrder, StateType >, 16
  setup, 16, 17
  setupN, 17
Iir::Butterworth::BandPassBase, 25
Iir::Butterworth::BandShelf< FilterOrder, StateType >, 27
  setup, 27, 28
  setupN, 28
Iir::Butterworth::BandShelfBase, 35
Iir::Butterworth::BandStop< FilterOrder, StateType >, 37
  setup, 37
  setupN, 38
Iir::Butterworth::BandStopBase, 44
Iir::Butterworth::HighPass< FilterOrder, StateType >, 53
  setup, 54
  setupN, 54
Iir::Butterworth::HighPassBase, 60
Iir::Butterworth::HighShelf< FilterOrder, StateType >, 62
  setup, 62, 63
  setupN, 63
Iir::Butterworth::HighShelfBase, 70
Iir::Butterworth::LowPass< FilterOrder, StateType >, 73
  setup, 74
  setupN, 74, 75
Iir::Butterworth::LowPassBase, 80
Iir::Butterworth::LowShelf< FilterOrder, StateType >, 82
  setup, 83
  setupN, 83, 84
Iir::Butterworth::LowShelfBase, 90
Iir::Cascade, 49
  getNumStages, 50
  getPoleZeros, 50
  operator[], 50
  response, 50
Iir::Cascade::Storage, 98
Iir::CascadeStages< MaxStages, StateType >, 51
  filter, 51
  getCascadeStorage, 51
  reset, 51
  setup, 52
Iir::ChebyshevI, 11
Iir::ChebyshevI::AnalogLowPass, 14

Iir::ChebyshevI::AnalogLowShelf, 15
Iir::ChebyshevI::BandPass< FilterOrder, StateType >, 18
  setup, 18, 19
  setupN, 19
Iir::ChebyshevI::BandPassBase, 25
Iir::ChebyshevI::BandShelf< FilterOrder, StateType >, 29
  setup, 30
  setupN, 30, 31
Iir::ChebyshevI::BandShelfBase, 36
Iir::ChebyshevI::BandStop< FilterOrder, StateType >, 38
  setup, 39
  setupN, 40
Iir::ChebyshevI::BandStopBase, 45
Iir::ChebyshevI::HighPass< FilterOrder, StateType >, 55
  setup, 55, 56
  setupN, 56
Iir::ChebyshevI::HighPassBase, 61
Iir::ChebyshevI::HighShelf< FilterOrder, StateType >, 64
  setup, 64, 65
  setupN, 65
Iir::ChebyshevI::HighShelfBase, 70
Iir::ChebyshevI::LowPass< FilterOrder, StateType >, 75
  setup, 76
  setupN, 76
Iir::ChebyshevI::LowPassBase, 81
Iir::ChebyshevI::LowShelf< FilterOrder, StateType >, 84
  setup, 85
  setupN, 86
Iir::ChebyshevI::LowShelfBase, 90
Iir::ChebyshevII, 12
Iir::ChebyshevII::AnalogLowPass, 14
Iir::ChebyshevII::AnalogLowShelf, 15
Iir::ChebyshevII::BandPass< FilterOrder, StateType >, 20
  setup, 20, 21
  setupN, 21
Iir::ChebyshevII::BandPassBase, 26
Iir::ChebyshevII::BandShelf< FilterOrder, StateType >, 31
  setup, 32
  setupN, 33
Iir::ChebyshevII::BandShelfBase, 36
Iir::ChebyshevII::BandStop< FilterOrder, StateType >, 41
  setup, 41, 42
  setupN, 42
Iir::ChebyshevII::BandStopBase, 45
Iir::ChebyshevII::HighPass< FilterOrder, StateType >, 57
  setup, 57, 58
  setupN, 58
Iir::ChebyshevII::HighPassBase, 61
Iir::ChebyshevII::HighShelf< FilterOrder, StateType >, 66
  setup, 67
  setupN, 67, 68
Iir::ChebyshevII::HighShelfBase, 71
Iir::ChebyshevII::LowPass< FilterOrder, StateType >, 77
  setup, 78
  setupN, 78
Iir::ChebyshevII::LowPassBase, 81
Iir::ChebyshevII::LowShelf< FilterOrder, StateType >, 86
  setup, 87
  setupN, 88
Iir::ChebyshevII::LowShelfBase, 91
Iir::ComplexPair, 52
  isMatchedPair, 52
Iir::Custom, 12
Iir::Custom::OnePole, 91
Iir::Custom::SOSCascade< NSOS, StateType >, 96
  setup, 97
  SOSCascade, 97
Iir::Custom::TwoPole, 98
Iir::DirectFormI, 52
Iir::DirectFormII, 53
Iir::HighPassTransform, 62
Iir::Layout< MaxPoles >, 73
Iir::LayoutBase, 73
Iir::LowPassTransform, 82
Iir::PoleFilter< BaseClass, StateType, MaxAnalogPoles, MaxDigitalPoles >, 92
Iir::PoleFilterBase< AnalogPrototype >, 93
Iir::PoleFilterBase2, 93
Iir::PoleZeroPair, 94
Iir::RBJ::AllPass, 12
  setup, 13
  setupN, 13
Iir::RBJ::BandPass1, 22
  setup, 23
  setupN, 23
Iir::RBJ::BandPass2, 23
  setup, 24
  setupN, 25
Iir::RBJ::BandShelf, 33
  setup, 34
  setupN, 35
Iir::RBJ::BandStop, 43
  setup, 44
  setupN, 44
Iir::RBJ::HighPass, 59
  setup, 60
  setupN, 60
Iir::RBJ::HighShelf, 68
  setup, 69
  setupN, 69
Iir::RBJ::IIRNotch, 71
  setup, 72
  setupN, 72

Iir::RBJ::LowPass, 79
    setup, 80
    setupN, 80
Iir::RBJ::LowShelf, 88
    setup, 89
    setupN, 89
Iir::RBJ::RBJbase, 95
Iir::TransposedDirectFormII, 98
isMatchedPair
    Iir::ComplexPair, 52

operator[]
    Iir::Cascade, 50

reset
    Iir::CascadeStages< MaxStages, StateType >, 51
response
    Iir::Biquad, 48
    Iir::Cascade, 50

setCoefficients
    Iir::Biquad, 48
setIdentity
    Iir::Biquad, 49
setOnePole
    Iir::Biquad, 49
setPoleZeroPair
    Iir::Biquad, 49
setTwoPole
    Iir::Biquad, 49
setup
    Iir::Butterworth::BandPass< FilterOrder, StateType >, 16, 17
    Iir::Butterworth::BandShelf< FilterOrder, StateType >, 27, 28
    Iir::Butterworth::BandStop< FilterOrder, StateType >, 37
    Iir::Butterworth::HighPass< FilterOrder, StateType >, 54
    Iir::Butterworth::HighShelf< FilterOrder, StateType >, 62, 63
    Iir::Butterworth::LowPass< FilterOrder, StateType >, 74
    Iir::Butterworth::LowShelf< FilterOrder, StateType >, 83
    Iir::CascadeStages< MaxStages, StateType >, 52
    Iir::ChebyshevI::BandPass< FilterOrder, StateType >, 18, 19
    Iir::ChebyshevI::BandShelf< FilterOrder, StateType >, 30
    Iir::ChebyshevI::BandStop< FilterOrder, StateType >, 39
    Iir::ChebyshevI::HighPass< FilterOrder, StateType >, 55, 56
    Iir::ChebyshevI::HighShelf< FilterOrder, StateType >, 64, 65
    Iir::ChebyshevI::LowPass< FilterOrder, StateType >, 76
    Iir::ChebyshevI::LowShelf< FilterOrder, StateType >, 85
    Iir::ChebyshevII::BandPass< FilterOrder, StateType >, 20, 21
    Iir::ChebyshevII::BandShelf< FilterOrder, StateType >, 32
    Iir::ChebyshevII::BandStop< FilterOrder, StateType >, 41, 42
    Iir::ChebyshevII::HighPass< FilterOrder, StateType >, 57, 58
    Iir::ChebyshevII::HighShelf< FilterOrder, StateType >, 67
    Iir::ChebyshevII::LowPass< FilterOrder, StateType >, 78
    Iir::ChebyshevII::LowShelf< FilterOrder, StateType >, 87
    Iir::Custom::SOSCascade< NSOS, StateType >, 97
    Iir::RBJ::AllPass, 13
    Iir::RBJ::BandPass1, 23
    Iir::RBJ::BandPass2, 24
    Iir::RBJ::BandShelf, 34
    Iir::RBJ::BandStop, 44
    Iir::RBJ::HighPass, 60
    Iir::RBJ::HighShelf, 69
    Iir::RBJ::IIRNotch, 72
    Iir::RBJ::LowPass, 80
    Iir::RBJ::LowShelf, 89
setupN
    Iir::Butterworth::BandPass< FilterOrder, StateType >, 17
    Iir::Butterworth::BandShelf< FilterOrder, StateType >, 28
    Iir::Butterworth::BandStop< FilterOrder, StateType >, 38
    Iir::Butterworth::HighPass< FilterOrder, StateType >, 54
    Iir::Butterworth::HighShelf< FilterOrder, StateType >, 63
    Iir::Butterworth::LowPass< FilterOrder, StateType >, 74, 75
    Iir::Butterworth::LowShelf< FilterOrder, StateType >, 83, 84
    Iir::ChebyshevI::BandPass< FilterOrder, StateType >, 19
    Iir::ChebyshevI::BandShelf< FilterOrder, StateType >, 30, 31
    Iir::ChebyshevI::BandStop< FilterOrder, StateType >, 40
    Iir::ChebyshevI::HighPass< FilterOrder, StateType >, 56
    Iir::ChebyshevI::HighShelf< FilterOrder, StateType >, 65
    Iir::ChebyshevI::LowPass< FilterOrder, StateType >, 76
    Iir::ChebyshevI::LowShelf< FilterOrder, StateType >, 86
    Iir::ChebyshevII::BandPass< FilterOrder, State-

Type >, 21
iir::ChebyshevII::BandShelf< FilterOrder, StateType >, 33
iir::ChebyshevII::BandStop< FilterOrder, StateType >, 42
iir::ChebyshevII::HighPass< FilterOrder, StateType >, 58
iir::ChebyshevII::HighShelf< FilterOrder, StateType >, 67, 68
iir::ChebyshevII::LowPass< FilterOrder, StateType >, 78
iir::ChebyshevII::LowShelf< FilterOrder, StateType >, 88
iir::RBJ::AllPass, 13
iir::RBJ::BandPass1, 23
iir::RBJ::BandPass2, 25
iir::RBJ::BandShelf, 35
iir::RBJ::BandStop, 44
iir::RBJ::HighPass, 60
iir::RBJ::HighShelf, 69
iir::RBJ::IIRNotch, 72
iir::RBJ::LowPass, 80
iir::RBJ::LowShelf, 89
SOSCascade
iir::Custom::SOSCascade< NSOS, StateType >, 97