

TimeTagger

2.10.6.0

Generated by Doxygen 1.8.17

1 TimeTagger	1
2 Module Index	3
2.1 Modules	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	11
5.1 File List	11
6 Module Documentation	13
6.1 Implementations with a Time Tagger interface	13
6.1.1 Detailed Description	13
6.2 All measurements and virtual channels	14
6.2.1 Detailed Description	14
6.3 Event counting	15
6.3.1 Detailed Description	15
6.4 Time histograms	16
6.4.1 Detailed Description	16
6.5 Fluorescence-lifetime imaging (FLIM)	17
6.5.1 Detailed Description	17
6.6 Time-tag-streaming	18
6.6.1 Detailed Description	18
6.7 Helper classes	19
6.7.1 Detailed Description	19
6.8 Virtual Channels	20
6.8.1 Detailed Description	20
7 Class Documentation	21
7.1 Coincidence Class Reference	21
7.1.1 Detailed Description	22
7.1.2 Constructor & Destructor Documentation	22
7.1.2.1 Coincidence()	22
7.1.3 Member Function Documentation	22
7.1.3.1 getChannel()	23
7.2 Coincidences Class Reference	23
7.2.1 Detailed Description	24
7.2.2 Constructor & Destructor Documentation	24
7.2.2.1 Coincidences()	24
7.2.2.2 ~Coincidences()	24

7.2.3 Member Function Documentation	24
7.2.3.1 getChannels()	24
7.2.3.2 next_impl()	25
7.2.3.3 setCoincidenceWindow()	25
7.3 Combiner Class Reference	25
7.3.1 Detailed Description	26
7.3.2 Constructor & Destructor Documentation	26
7.3.2.1 Combiner()	26
7.3.2.2 ~Combiner()	27
7.3.3 Member Function Documentation	27
7.3.3.1 clear_impl()	27
7.3.3.2 getChannel()	27
7.3.3.3 getChannelCounts()	27
7.3.3.4 getData()	28
7.3.3.5 next_impl()	28
7.4 ConstantFractionDiscriminator Class Reference	28
7.4.1 Detailed Description	29
7.4.2 Constructor & Destructor Documentation	30
7.4.2.1 ConstantFractionDiscriminator()	30
7.4.2.2 ~ConstantFractionDiscriminator()	30
7.4.3 Member Function Documentation	30
7.4.3.1 getChannels()	30
7.4.3.2 next_impl()	30
7.4.3.3 on_start()	31
7.5 Correlation Class Reference	31
7.5.1 Detailed Description	32
7.5.2 Constructor & Destructor Documentation	33
7.5.2.1 Correlation()	33
7.5.2.2 ~Correlation()	33
7.5.3 Member Function Documentation	33
7.5.3.1 clear_impl()	33
7.5.3.2 getData()	34
7.5.3.3 getDataNormalized()	34
7.5.3.4 getIndex()	34
7.5.3.5 next_impl()	34
7.6 CountBetweenMarkers Class Reference	35
7.6.1 Detailed Description	36
7.6.2 Constructor & Destructor Documentation	37
7.6.2.1 CountBetweenMarkers()	37
7.6.2.2 ~CountBetweenMarkers()	37
7.6.3 Member Function Documentation	37
7.6.3.1 clear_impl()	37

7.6.3.2 getBinWidths()	38
7.6.3.3 getData()	38
7.6.3.4 getIndex()	38
7.6.3.5 next_impl()	38
7.6.3.6 ready()	39
7.7 Counter Class Reference	39
7.7.1 Detailed Description	40
7.7.2 Constructor & Destructor Documentation	40
7.7.2.1 Counter()	40
7.7.2.2 ~Counter()	41
7.7.3 Member Function Documentation	41
7.7.3.1 clear_impl()	41
7.7.3.2 getData()	41
7.7.3.3 getDataNormalized()	42
7.7.3.4 getDataObject()	42
7.7.3.5 getDataTotalCounts()	42
7.7.3.6 getIndex()	42
7.7.3.7 next_impl()	43
7.7.3.8 on_start()	43
7.8 CounterData Class Reference	43
7.8.1 Detailed Description	44
7.8.2 Constructor & Destructor Documentation	44
7.8.2.1 ~CounterData()	44
7.8.3 Member Function Documentation	44
7.8.3.1 getChannels()	45
7.8.3.2 getData()	45
7.8.3.3 getDataNormalized()	45
7.8.3.4 getDataTotalCounts()	45
7.8.3.5 getIndex()	45
7.8.3.6 getOverflowMask()	45
7.8.3.7 getTime()	46
7.8.4 Member Data Documentation	46
7.8.4.1 dropped_bins	46
7.8.4.2 overflow	46
7.8.4.3 size	46
7.9 Countrate Class Reference	46
7.9.1 Detailed Description	47
7.9.2 Constructor & Destructor Documentation	47
7.9.2.1 Countrate()	47
7.9.2.2 ~Countrate()	48
7.9.3 Member Function Documentation	48
7.9.3.1 clear_impl()	48

7.9.3.2	getCountsTotal()	48
7.9.3.3	getData()	48
7.9.3.4	next_impl()	48
7.9.3.5	on_start()	49
7.10	CustomLogger Class Reference	49
7.10.1	Detailed Description	49
7.10.2	Constructor & Destructor Documentation	50
7.10.2.1	CustomLogger()	50
7.10.2.2	~CustomLogger()	50
7.10.3	Member Function Documentation	50
7.10.3.1	disable()	50
7.10.3.2	enable()	50
7.10.3.3	Log()	50
7.11	CustomMeasurementBase Class Reference	51
7.11.1	Detailed Description	52
7.11.2	Constructor & Destructor Documentation	52
7.11.2.1	CustomMeasurementBase()	52
7.11.2.2	~CustomMeasurementBase()	52
7.11.3	Member Function Documentation	52
7.11.3.1	_lock()	52
7.11.3.2	_unlock()	52
7.11.3.3	clear_impl()	52
7.11.3.4	finalize_init()	53
7.11.3.5	is_running()	53
7.11.3.6	next_impl()	53
7.11.3.7	next_impl_cs()	53
7.11.3.8	on_start()	54
7.11.3.9	on_stop()	54
7.11.3.10	register_channel()	54
7.11.3.11	stop_all_custom_measurements()	54
7.11.3.12	unregister_channel()	54
7.12	DelayedChannel Class Reference	55
7.12.1	Detailed Description	56
7.12.2	Constructor & Destructor Documentation	56
7.12.2.1	DelayedChannel() [1/2]	56
7.12.2.2	DelayedChannel() [2/2]	56
7.12.2.3	~DelayedChannel()	57
7.12.3	Member Function Documentation	57
7.12.3.1	getChannel()	57
7.12.3.2	getChannels()	57
7.12.3.3	next_impl()	57
7.12.3.4	on_start()	58

7.12.3.5 setDelay()	58
7.13 Dump Class Reference	59
7.13.1 Detailed Description	59
7.13.2 Constructor & Destructor Documentation	60
7.13.2.1 Dump()	60
7.13.2.2 ~Dump()	60
7.13.3 Member Function Documentation	60
7.13.3.1 clear_impl()	60
7.13.3.2 next_impl()	60
7.13.3.3 on_start()	61
7.13.3.4 on_stop()	61
7.14 Event Struct Reference	61
7.14.1 Detailed Description	62
7.14.2 Member Data Documentation	62
7.14.2.1 state	62
7.14.2.2 time	62
7.15 EventGenerator Class Reference	62
7.15.1 Detailed Description	63
7.15.2 Constructor & Destructor Documentation	63
7.15.2.1 EventGenerator()	64
7.15.2.2 ~EventGenerator()	64
7.15.3 Member Function Documentation	64
7.15.3.1 clear_impl()	64
7.15.3.2 getChannel()	64
7.15.3.3 next_impl()	65
7.15.3.4 on_start()	65
7.16 FastBinning Class Reference	65
7.16.1 Detailed Description	66
7.16.2 Member Enumeration Documentation	66
7.16.2.1 Mode	66
7.16.3 Constructor & Destructor Documentation	66
7.16.3.1 FastBinning() [1/2]	66
7.16.3.2 FastBinning() [2/2]	67
7.16.4 Member Function Documentation	67
7.16.4.1 divide()	67
7.16.4.2 getMode()	67
7.17 FileReader Class Reference	67
7.17.1 Detailed Description	68
7.17.2 Constructor & Destructor Documentation	68
7.17.2.1 FileReader() [1/2]	68
7.17.2.2 FileReader() [2/2]	68
7.17.2.3 ~FileReader()	68

7.17.3 Member Function Documentation	68
7.17.3.1 getConfiguration()	69
7.17.3.2 getData()	69
7.17.3.3 getDataRaw()	69
7.17.3.4 getLastMarker()	70
7.17.3.5 hasData()	70
7.18 FileWriter Class Reference	70
7.18.1 Detailed Description	71
7.18.2 Constructor & Destructor Documentation	71
7.18.2.1 FileWriter()	71
7.18.2.2 ~FileWriter()	72
7.18.3 Member Function Documentation	72
7.18.3.1 clear_impl()	72
7.18.3.2 getMaxFileSize()	72
7.18.3.3 getTotalEvents()	72
7.18.3.4 getTotalSize()	73
7.18.3.5 next_impl()	73
7.18.3.6 on_start()	73
7.18.3.7 on_stop()	74
7.18.3.8 setMarker()	74
7.18.3.9 setMaxFileSize()	74
7.18.3.10 split()	74
7.19 Flim Class Reference	75
7.19.1 Detailed Description	77
7.19.2 Constructor & Destructor Documentation	77
7.19.2.1 Flim()	77
7.19.2.2 ~Flim()	78
7.19.3 Member Function Documentation	78
7.19.3.1 clear_impl()	78
7.19.3.2 frameReady()	78
7.19.3.3 get_ready_index()	79
7.19.3.4 getCurrentFrame()	79
7.19.3.5 getCurrentFrameEx()	79
7.19.3.6 getCurrentFrameIntensity()	79
7.19.3.7 getFramesAcquired()	79
7.19.3.8 getIndex()	79
7.19.3.9 getReadyFrame()	80
7.19.3.10 getReadyFrameEx()	80
7.19.3.11 getReadyFrameIntensity()	80
7.19.3.12 getSummedFrames()	81
7.19.3.13 getSummedFramesEx()	81
7.19.3.14 getSummedFramesIntensity()	81

7.19.3.15 initialize()	82
7.19.3.16 on_frame_end()	82
7.19.4 Member Data Documentation	82
7.19.4.1 accum_diffs	82
7.19.4.2 back_frames	82
7.19.4.3 captured_frames	83
7.19.4.4 frame_begins	83
7.19.4.5 frame_ends	83
7.19.4.6 last_frame	83
7.19.4.7 pixels_completed	83
7.19.4.8 summed_frames	83
7.19.4.9 swap_chain_lock	83
7.19.4.10 total_frames	84
7.20 FlimAbstract Class Reference	84
7.20.1 Detailed Description	85
7.20.2 Constructor & Destructor Documentation	85
7.20.2.1 FlimAbstract()	86
7.20.2.2 ~FlimAbstract()	86
7.20.3 Member Function Documentation	86
7.20.3.1 clear_impl()	87
7.20.3.2 isAcquiring()	87
7.20.3.3 next_impl()	87
7.20.3.4 on_frame_end()	88
7.20.3.5 on_start()	88
7.20.3.6 process_tags()	88
7.20.4 Member Data Documentation	88
7.20.4.1 acquiring	88
7.20.4.2 acquisition_lock	88
7.20.4.3 binner	89
7.20.4.4 binwidth	89
7.20.4.5 click_channel	89
7.20.4.6 current_frame_begin	89
7.20.4.7 current_frame_end	89
7.20.4.8 data_base	89
7.20.4.9 finish_after_outputframe	89
7.20.4.10 frame	89
7.20.4.11 frame_acquisition	90
7.20.4.12 frame_begin_channel	90
7.20.4.13 frames_completed	90
7.20.4.14 initialized	90
7.20.4.15 n_bins	90
7.20.4.16 n_frame_average	90

7.20.4.17 n_pixels	90
7.20.4.18 pixel_acquisition	90
7.20.4.19 pixel_begin_channel	91
7.20.4.20 pixel_begins	91
7.20.4.21 pixel_end_channel	91
7.20.4.22 pixel_ends	91
7.20.4.23 pixels_processed	91
7.20.4.24 previous_starts	91
7.20.4.25 start_channel	91
7.20.4.26 ticks	91
7.20.4.27 time_window	92
7.21 FlimBase Class Reference	92
7.21.1 Detailed Description	93
7.21.2 Constructor & Destructor Documentation	93
7.21.2.1 FlimBase()	93
7.21.2.2 ~FlimBase()	94
7.21.3 Member Function Documentation	94
7.21.3.1 frameReady()	94
7.21.3.2 initialize()	94
7.21.3.3 on_frame_end()	94
7.21.4 Member Data Documentation	94
7.21.4.1 total_frames	94
7.22 FlimFrameInfo Class Reference	95
7.22.1 Detailed Description	95
7.22.2 Constructor & Destructor Documentation	95
7.22.2.1 ~FlimFrameInfo()	95
7.22.3 Member Function Documentation	95
7.22.3.1 getFrameNumber()	96
7.22.3.2 getHistograms()	96
7.22.3.3 getIntensities()	96
7.22.3.4 getPixelBegins()	96
7.22.3.5 getPixelEnds()	96
7.22.3.6 getPixelPosition()	96
7.22.3.7 getSummedCounts()	97
7.22.3.8 isValid()	97
7.22.4 Member Data Documentation	97
7.22.4.1 bins	97
7.22.4.2 frame_number	97
7.22.4.3 pixel_position	97
7.22.4.4 pixels	97
7.22.4.5 valid	98
7.23 FrequencyMultiplier Class Reference	98

7.23.1 Detailed Description	99
7.23.2 Constructor & Destructor Documentation	99
7.23.2.1 FrequencyMultiplier()	99
7.23.2.2 ~FrequencyMultiplier()	100
7.23.3 Member Function Documentation	100
7.23.3.1 getChannel()	100
7.23.3.2 getMultiplier()	100
7.23.3.3 next_impl()	100
7.24 FrequencyStability Class Reference	101
7.24.1 Detailed Description	101
7.24.2 Constructor & Destructor Documentation	102
7.24.2.1 FrequencyStability()	102
7.24.2.2 ~FrequencyStability()	103
7.24.3 Member Function Documentation	103
7.24.3.1 clear_impl()	103
7.24.3.2 getDataObject()	103
7.24.3.3 next_impl()	103
7.24.3.4 on_start()	104
7.25 FrequencyStabilityData Class Reference	104
7.25.1 Detailed Description	105
7.25.2 Constructor & Destructor Documentation	105
7.25.2.1 ~FrequencyStabilityData()	105
7.25.3 Member Function Documentation	105
7.25.3.1 getADEV()	105
7.25.3.2 getADEVScaled()	105
7.25.3.3 getHDEV()	106
7.25.3.4 getHDEVScaled()	106
7.25.3.5 getMDEV()	106
7.25.3.6 getSTDD()	106
7.25.3.7 getTau()	106
7.25.3.8 getTDEV()	106
7.25.3.9 getTraceFrequency()	107
7.25.3.10 getTraceIndex()	107
7.25.3.11 getTracePhase()	107
7.26 GatedChannel Class Reference	107
7.26.1 Detailed Description	108
7.26.2 Constructor & Destructor Documentation	108
7.26.2.1 GatedChannel()	108
7.26.2.2 ~GatedChannel()	109
7.26.3 Member Function Documentation	109
7.26.3.1 getChannel()	109
7.26.3.2 next_impl()	109

7.27 Histogram Class Reference	110
7.27.1 Detailed Description	111
7.27.2 Constructor & Destructor Documentation	111
7.27.2.1 Histogram()	111
7.27.2.2 ~Histogram()	112
7.27.3 Member Function Documentation	112
7.27.3.1 clear_impl()	112
7.27.3.2 getData()	112
7.27.3.3 getIndex()	112
7.27.3.4 next_impl()	112
7.27.3.5 on_start()	113
7.28 Histogram2D Class Reference	113
7.28.1 Detailed Description	114
7.28.2 Constructor & Destructor Documentation	115
7.28.2.1 Histogram2D()	115
7.28.2.2 ~Histogram2D()	115
7.28.3 Member Function Documentation	115
7.28.3.1 clear_impl()	115
7.28.3.2 getData()	116
7.28.3.3 getIndex()	116
7.28.3.4 getIndex_1()	116
7.28.3.5 getIndex_2()	116
7.28.3.6 next_impl()	116
7.29 HistogramLogBins Class Reference	117
7.29.1 Detailed Description	118
7.29.2 Constructor & Destructor Documentation	118
7.29.2.1 HistogramLogBins()	119
7.29.2.2 ~HistogramLogBins()	119
7.29.3 Member Function Documentation	119
7.29.3.1 clear_impl()	119
7.29.3.2 getBinEdges()	120
7.29.3.3 getData()	120
7.29.3.4 getDataNormalizedCountsPerPs()	120
7.29.3.5 getDataNormalizedG2()	120
7.29.3.6 next_impl()	120
7.30 HistogramND Class Reference	121
7.30.1 Detailed Description	122
7.30.2 Constructor & Destructor Documentation	122
7.30.2.1 HistogramND()	122
7.30.2.2 ~HistogramND()	122
7.30.3 Member Function Documentation	123
7.30.3.1 clear_impl()	123

7.30.3.2	getData()	123
7.30.3.3	getIndex()	123
7.30.3.4	next_impl()	123
7.31	Iterator Class Reference	124
7.31.1	Detailed Description	125
7.31.2	Constructor & Destructor Documentation	125
7.31.2.1	Iterator()	125
7.31.2.2	~Iterator()	125
7.31.3	Member Function Documentation	125
7.31.3.1	clear_impl()	125
7.31.3.2	next()	126
7.31.3.3	next_impl()	126
7.31.3.4	size()	126
7.32	IteratorBase Class Reference	127
7.32.1	Detailed Description	129
7.32.2	Constructor & Destructor Documentation	129
7.32.2.1	IteratorBase()	129
7.32.2.2	~IteratorBase()	129
7.32.3	Member Function Documentation	129
7.32.3.1	clear()	130
7.32.3.2	clear_impl()	130
7.32.3.3	finish_running()	130
7.32.3.4	finishInitialization()	130
7.32.3.5	getCaptureDuration()	130
7.32.3.6	getConfiguration()	131
7.32.3.7	getLock()	131
7.32.3.8	getNewVirtualChannel()	131
7.32.3.9	isRunning()	131
7.32.3.10	lock()	132
7.32.3.11	next_impl()	132
7.32.3.12	on_start()	132
7.32.3.13	on_stop()	133
7.32.3.14	parallelize()	133
7.32.3.15	registerChannel()	133
7.32.3.16	start()	133
7.32.3.17	startFor()	134
7.32.3.18	stop()	134
7.32.3.19	unlock()	134
7.32.3.20	unregisterChannel()	134
7.32.3.21	waitUntilFinished()	135
7.32.4	Member Data Documentation	135
7.32.4.1	autostart	135

7.32.4.2 capture_duration	135
7.32.4.3 channels_registered	135
7.32.4.4 running	136
7.32.4.5 tagger	136
7.33 OrderedBarrier Class Reference	136
7.33.1 Detailed Description	136
7.33.2 Constructor & Destructor Documentation	136
7.33.2.1 OrderedBarrier()	137
7.33.2.2 ~OrderedBarrier()	137
7.33.3 Member Function Documentation	137
7.33.3.1 queue()	137
7.33.3.2 waitUntilFinished()	137
7.34 OrderedPipeline Class Reference	137
7.34.1 Detailed Description	137
7.34.2 Constructor & Destructor Documentation	138
7.34.2.1 OrderedPipeline()	138
7.34.2.2 ~OrderedPipeline()	138
7.35 OrderedBarrier::OrderInstance Class Reference	138
7.35.1 Detailed Description	138
7.35.2 Constructor & Destructor Documentation	138
7.35.2.1 OrderInstance() [1/2]	138
7.35.2.2 OrderInstance() [2/2]	139
7.35.2.3 ~OrderInstance()	139
7.35.3 Member Function Documentation	139
7.35.3.1 release()	139
7.35.3.2 sync()	139
7.36 Sampler Class Reference	139
7.36.1 Detailed Description	140
7.36.2 Constructor & Destructor Documentation	140
7.36.2.1 Sampler()	140
7.36.2.2 ~Sampler()	141
7.36.3 Member Function Documentation	141
7.36.3.1 clear_impl()	141
7.36.3.2 getData()	141
7.36.3.3 getDataAsMask()	141
7.36.3.4 next_impl()	142
7.36.3.5 on_start()	142
7.37 Scope Class Reference	143
7.37.1 Detailed Description	143
7.37.2 Constructor & Destructor Documentation	144
7.37.2.1 Scope()	144
7.37.2.2 ~Scope()	145

7.37.3 Member Function Documentation	145
7.37.3.1 clear_impl()	145
7.37.3.2 getData()	145
7.37.3.3 getWindowSize()	145
7.37.3.4 next_impl()	145
7.37.3.5 ready()	146
7.37.3.6 triggered()	146
7.38 SoftwareClockState Struct Reference	146
7.38.1 Member Data Documentation	147
7.38.1.1 averaging_periods	147
7.38.1.2 clock_period	147
7.38.1.3 enabled	147
7.38.1.4 error_counter	147
7.38.1.5 ideal_clock_channel	147
7.38.1.6 input_channel	147
7.38.1.7 is_locked	147
7.38.1.8 last_ideal_clock_event	148
7.38.1.9 period_error	148
7.38.1.10 phase_error_estimation	148
7.39 StartStop Class Reference	148
7.39.1 Detailed Description	149
7.39.2 Constructor & Destructor Documentation	149
7.39.2.1 StartStop()	149
7.39.2.2 ~StartStop()	150
7.39.3 Member Function Documentation	150
7.39.3.1 clear_impl()	150
7.39.3.2 getData()	150
7.39.3.3 next_impl()	150
7.39.3.4 on_start()	151
7.40 SynchronizedMeasurements Class Reference	151
7.40.1 Detailed Description	152
7.40.2 Constructor & Destructor Documentation	152
7.40.2.1 SynchronizedMeasurements()	152
7.40.2.2 ~SynchronizedMeasurements()	152
7.40.3 Member Function Documentation	152
7.40.3.1 clear()	152
7.40.3.2 getTagger()	153
7.40.3.3 isRunning()	153
7.40.3.4 registerMeasurement()	153
7.40.3.5 runCallback()	153
7.40.3.6 start()	153
7.40.3.7 startFor()	154

7.40.3.8 stop()	154
7.40.3.9 unregisterMeasurement()	154
7.40.3.10 waitUntilFinished()	154
7.41 SyntheticSingleTag Class Reference	154
7.41.1 Detailed Description	155
7.41.2 Constructor & Destructor Documentation	155
7.41.2.1 SyntheticSingleTag()	155
7.41.2.2 ~SyntheticSingleTag()	156
7.41.3 Member Function Documentation	156
7.41.3.1 getChannel()	156
7.41.3.2 next_impl()	156
7.41.3.3 trigger()	157
7.42 Tag Struct Reference	157
7.42.1 Detailed Description	157
7.42.2 Member Enumeration Documentation	157
7.42.2.1 Type	158
7.42.3 Member Data Documentation	158
7.42.3.1 channel	158
7.42.3.2 missed_events	158
7.42.3.3 reserved	159
7.42.3.4 time	159
7.42.3.5 type	159
7.43 TimeDifferences Class Reference	159
7.43.1 Detailed Description	160
7.43.2 Constructor & Destructor Documentation	161
7.43.2.1 TimeDifferences()	161
7.43.2.2 ~TimeDifferences()	162
7.43.3 Member Function Documentation	162
7.43.3.1 clear_impl()	162
7.43.3.2 getCounts()	162
7.43.3.3 getData()	162
7.43.3.4 getIndex()	162
7.43.3.5 next_impl()	162
7.43.3.6 on_start()	163
7.43.3.7 ready()	163
7.43.3.8 setMaxCounts()	163
7.44 TimeDifferencesImpl< T > Class Template Reference	164
7.45 TimeDifferencesND Class Reference	164
7.45.1 Detailed Description	165
7.45.2 Constructor & Destructor Documentation	165
7.45.2.1 TimeDifferencesND()	166
7.45.2.2 ~TimeDifferencesND()	166

7.45.3 Member Function Documentation	166
7.45.3.1 clear_impl()	166
7.45.3.2 getData()	167
7.45.3.3 getIndex()	167
7.45.3.4 next_impl()	167
7.45.3.5 on_start()	167
7.46 TimeTagger Class Reference	168
7.46.1 Detailed Description	170
7.46.2 Member Function Documentation	170
7.46.2.1 autoCalibration()	170
7.46.2.2 clearConditionalFilter()	170
7.46.2.3 factoryAccess()	171
7.46.2.4 getChannelList()	171
7.46.2.5 getChannelNumberScheme()	171
7.46.2.6 getConditionalFilterFiltered()	171
7.46.2.7 getConditionalFilterTrigger()	171
7.46.2.8 getDACRange()	172
7.46.2.9 getDistributionCount()	172
7.46.2.10 getDistributionPSecs()	172
7.46.2.11 getEventDivider()	172
7.46.2.12 getFirmwareVersion()	172
7.46.2.13 getHardwareBufferSize()	173
7.46.2.14 getHardwareDelayCompensation()	173
7.46.2.15 getInputMux()	174
7.46.2.16 getLicenseInfo()	174
7.46.2.17 getModel()	174
7.46.2.18 getNormalization()	174
7.46.2.19 getPcbVersion()	175
7.46.2.20 getPsPerClock()	175
7.46.2.21 getSensorData()	175
7.46.2.22 getSerial()	175
7.46.2.23 getStreamBlockSizeEvents()	175
7.46.2.24 getStreamBlockSizeLatency()	175
7.46.2.25 getTestSignalDivider()	176
7.46.2.26 getTriggerLevel()	176
7.46.2.27 isChannelRegistered()	176
7.46.2.28 isServerRunning()	176
7.46.2.29 reset()	176
7.46.2.30 setConditionalFilter()	177
7.46.2.31 setEventDivider()	177
7.46.2.32 setHardwareBufferSize()	177
7.46.2.33 setInputMux()	178

7.46.2.34	setLED()	178
7.46.2.35	setNormalization()	178
7.46.2.36	setSoundFrequency()	179
7.46.2.37	setStreamBlockSize()	179
7.46.2.38	setTestSignalDivider()	179
7.46.2.39	setTriggerLevel()	180
7.46.2.40	startServer()	180
7.46.2.41	stopServer()	180
7.47	TimeTaggerBase Class Reference	181
7.47.1	Detailed Description	182
7.47.2	Member Typedef Documentation	183
7.47.2.1	IteratorCallback	183
7.47.2.2	IteratorCallbackMap	183
7.47.3	Constructor & Destructor Documentation	183
7.47.3.1	TimeTaggerBase() [1/2]	183
7.47.3.2	~TimeTaggerBase()	183
7.47.3.3	TimeTaggerBase() [2/2]	183
7.47.4	Member Function Documentation	183
7.47.4.1	addChild()	184
7.47.4.2	addIterator()	184
7.47.4.3	clearOverflows()	184
7.47.4.4	disableSoftwareClock()	184
7.47.4.5	freeIterator()	184
7.47.4.6	freeVirtualChannel()	184
7.47.4.7	getConfiguration()	185
7.47.4.8	getDeadtime()	185
7.47.4.9	getDelayHardware()	185
7.47.4.10	getDelaySoftware()	186
7.47.4.11	getFence()	186
7.47.4.12	getInputDelay()	186
7.47.4.13	getInvertedChannel()	187
7.47.4.14	getNewVirtualChannel()	187
7.47.4.15	getOverflows()	187
7.47.4.16	getOverflowsAndClear()	187
7.47.4.17	getSoftwareClockState()	188
7.47.4.18	getTestSignal()	188
7.47.4.19	isUnusedChannel()	188
7.47.4.20	operator=()	188
7.47.4.21	registerChannel()	188
7.47.4.22	release()	189
7.47.4.23	removeChild()	189
7.47.4.24	runSynchronized()	189

7.47.4.25 setDeadtime()	189
7.47.4.26 setDelayHardware()	190
7.47.4.27 setDelaySoftware()	190
7.47.4.28 setInputDelay()	191
7.47.4.29 setSoftwareClock()	191
7.47.4.30 setTestSignal() [1/2]	192
7.47.4.31 setTestSignal() [2/2]	192
7.47.4.32 sync()	192
7.47.4.33 unregisterChannel()	193
7.47.4.34 waitForFence()	193
7.48 TimeTaggerNetwork Class Reference	193
7.48.1 Detailed Description	195
7.48.2 Member Function Documentation	196
7.48.2.1 clearConditionalFilter()	196
7.48.2.2 clearOverflowsClient()	196
7.48.2.3 getChannelList()	196
7.48.2.4 getChannelNumberScheme()	196
7.48.2.5 getConditionalFilterFiltered()	196
7.48.2.6 getConditionalFilterTrigger()	197
7.48.2.7 getDACRange()	197
7.48.2.8 getDelayClient()	197
7.48.2.9 getEventDivider()	197
7.48.2.10 getFirmwareVersion()	198
7.48.2.11 getHardwareBufferSize()	198
7.48.2.12 getHardwareDelayCompensation()	198
7.48.2.13 getLicenseInfo()	199
7.48.2.14 getModel()	199
7.48.2.15 getNormalization()	199
7.48.2.16 getOverflowsAndClearClient()	200
7.48.2.17 getOverflowsClient()	200
7.48.2.18 getPcbVersion()	200
7.48.2.19 getPsPerClock()	200
7.48.2.20 getSensorData()	200
7.48.2.21 getSerial()	200
7.48.2.22 getStreamBlockSizeEvents()	201
7.48.2.23 getStreamBlockSizeLatency()	201
7.48.2.24 getTestSignal()	201
7.48.2.25 getTestSignalDivider()	202
7.48.2.26 getTriggerLevel()	202
7.48.2.27 isConnected()	202
7.48.2.28 setConditionalFilter()	202
7.48.2.29 setDelayClient()	204

7.48.2.30 setEventDivider()	204
7.48.2.31 setHardwareBufferSize()	205
7.48.2.32 setLED()	205
7.48.2.33 setNormalization()	205
7.48.2.34 setSoundFrequency()	205
7.48.2.35 setStreamBlockSize()	206
7.48.2.36 setTestSignalDivider()	206
7.48.2.37 setTriggerLevel()	206
7.49 TimeTaggerVirtual Class Reference	207
7.49.1 Detailed Description	208
7.49.2 Member Function Documentation	208
7.49.2.1 clearConditionalFilter()	208
7.49.2.2 getConditionalFilterFiltered()	208
7.49.2.3 getConditionalFilterTrigger()	208
7.49.2.4 getReplaySpeed()	208
7.49.2.5 replay()	209
7.49.2.6 reset()	209
7.49.2.7 setConditionalFilter()	209
7.49.2.8 setReplaySpeed()	210
7.49.2.9 stop()	210
7.49.2.10 waitForCompletion()	210
7.50 TimeTagStream Class Reference	211
7.50.1 Detailed Description	211
7.50.2 Constructor & Destructor Documentation	211
7.50.2.1 TimeTagStream()	212
7.50.2.2 ~TimeTagStream()	212
7.50.3 Member Function Documentation	212
7.50.3.1 clear_impl()	212
7.50.3.2 getCounts()	212
7.50.3.3 getData()	213
7.50.3.4 next_impl()	213
7.51 TimeTagStreamBuffer Class Reference	213
7.51.1 Detailed Description	214
7.51.2 Constructor & Destructor Documentation	214
7.51.2.1 ~TimeTagStreamBuffer()	214
7.51.3 Member Function Documentation	214
7.51.3.1 getChannels()	214
7.51.3.2 getEventTypes()	214
7.51.3.3 getMissedEvents()	214
7.51.3.4 getOverflows()	215
7.51.3.5 getTimestamps()	215
7.51.4 Member Data Documentation	215

7.51.4.1 hasOverflows	215
7.51.4.2 size	215
7.51.4.3 tGetData	215
7.51.4.4 tStart	215
7.52 TriggerOnCountrate Class Reference	216
7.52.1 Detailed Description	217
7.52.2 Constructor & Destructor Documentation	217
7.52.2.1 TriggerOnCountrate()	217
7.52.2.2 ~TriggerOnCountrate()	218
7.52.3 Member Function Documentation	218
7.52.3.1 getChannelAbove()	218
7.52.3.2 getChannelBelow()	218
7.52.3.3 getChannels()	218
7.52.3.4 getCurrentCountrate()	219
7.52.3.5 injectCurrentState()	219
7.52.3.6 isAbove()	219
7.52.3.7 isBelow()	219
7.52.3.8 next_impl()	219
7.52.3.9 on_start()	220
8 File Documentation	221
8.1 Iterators.h File Reference	221
8.1.1 Macro Definition Documentation	223
8.1.1.1 BINNING_TEMPLATE_HELPER	224
8.1.2 Enumeration Type Documentation	224
8.1.2.1 CoincidenceTimestamp	224
8.1.2.2 State	225
8.2 TimeTagger.h File Reference	225
8.2.1 Macro Definition Documentation	228
8.2.1.1 channel_t	228
8.2.1.2 ErrorLog	228
8.2.1.3 ErrorLogSuppressed	229
8.2.1.4 GET_DATA_1D	229
8.2.1.5 GET_DATA_1D_OP1	229
8.2.1.6 GET_DATA_1D_OP2	229
8.2.1.7 GET_DATA_2D	230
8.2.1.8 GET_DATA_2D_OP1	230
8.2.1.9 GET_DATA_2D_OP2	230
8.2.1.10 GET_DATA_3D	231
8.2.1.11 InfoLog	231
8.2.1.12 InfoLogSuppressed	231
8.2.1.13 LogMessage	231

8.2.1.14 LogMessageSuppressed	231
8.2.1.15 timestamp_t	231
8.2.1.16 TIMETAGGER_VERSION	232
8.2.1.17 TT_API	232
8.2.1.18 WarningLog	232
8.2.1.19 WarningLogSuppressed	232
8.2.2 Typedef Documentation	232
8.2.2.1 _Iterator	232
8.2.2.2 logger_callback	232
8.2.3 Enumeration Type Documentation	232
8.2.3.1 AccessMode	232
8.2.3.2 ChannelEdge	233
8.2.3.3 FrontendType	233
8.2.3.4 LanguageUsed	234
8.2.3.5 LogLevel	234
8.2.3.6 Resolution	234
8.2.3.7 UsageStatisticsStatus	235
8.2.4 Function Documentation	235
8.2.4.1 createTimeTagger()	235
8.2.4.2 createTimeTaggerNetwork()	235
8.2.4.3 createTimeTaggerVirtual()	236
8.2.4.4 extractLicenseInfo()	236
8.2.4.5 flashLicense()	236
8.2.4.6 freeTimeTagger()	236
8.2.4.7 getTimeTaggerChannelNumberScheme()	237
8.2.4.8 getTimeTaggerModel()	237
8.2.4.9 getTimeTaggerServerInfo()	237
8.2.4.10 getUsageStatisticsReport()	237
8.2.4.11 getUsageStatisticsStatus()	238
8.2.4.12 getVersion()	238
8.2.4.13 hasTimeTaggerVirtualLicense()	238
8.2.4.14 LogBase()	238
8.2.4.15 scanTimeTagger()	239
8.2.4.16 scanTimeTaggerServers()	239
8.2.4.17 setCustomBitFileName()	239
8.2.4.18 setFrontend()	239
8.2.4.19 setLanguageInfo()	240
8.2.4.20 setLogger()	240
8.2.4.21 setTimeTaggerChannelNumberScheme()	240
8.2.4.22 setUsageStatisticsStatus()	241
8.2.5 Variable Documentation	241
8.2.5.1 CHANNEL_UNUSED	241

8.2.5.2 CHANNEL_UNUSED_OLD	241
8.2.5.3 TT_CHANNEL_FALLING_EDGES	241
8.2.5.4 TT_CHANNEL_NUMBER_SCHEME_AUTO	242
8.2.5.5 TT_CHANNEL_NUMBER_SCHEME_ONE	242
8.2.5.6 TT_CHANNEL_NUMBER_SCHEME_ZERO	242
8.2.5.7 TT_CHANNEL_RISING_AND_FALLING_EDGES	242
8.2.5.8 TT_CHANNEL_RISING_EDGES	242

Chapter 1

TimeTagger

backend for [TimeTagger](#), an OpalKelly based single photon counting library

Author

Markus Wick markus@swabianinstruments.com

Helmut Fedder helmut@swabianinstruments.com

Michael Schlagmüller michael@swabianinstruments.com

[TimeTagger](#) provides an easy to use and cost effective hardware solution for time-resolved single photon counting applications.

This document describes the C++ native interface to the [TimeTagger](#) device.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Implementations with a Time Tagger interface	13
All measurements and virtual channels	14
Event counting	15
Time histograms	16
Fluorescence-lifetime imaging (FLIM)	17
Time-tag-streaming	18
Helper classes	19
Virtual Channels	20

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

CounterData	43
CustomLogger	49
Event	61
FastBinning	65
FileReader	67
FlimFrameInfo	95
FrequencyStabilityData	104
IteratorBase	127
Coincidences	23
Coincidence	21
Combiner	25
ConstantFractionDiscriminator	28
Correlation	31
CountBetweenMarkers	35
Counter	39
Countrate	46
CustomMeasurementBase	51
DelayedChannel	55
Dump	59
EventGenerator	62
FileWriter	70
FlimAbstract	84
Flim	75
FlimBase	92
FrequencyMultiplier	98
FrequencyStability	101
GatedChannel	107
Histogram	110
Histogram2D	113
HistogramLogBins	117
HistogramND	121
Iterator	124
Sampler	139
Scope	143
StartStop	148

SyntheticSingleTag	154
TimeDifferences	159
TimeDifferencesND	164
TimeTagStream	211
TriggerOnCountrate	216
OrderedBarrier	136
OrderedPipeline	137
OrderedBarrier::OrderInstance	138
SoftwareClockState	146
SynchronizedMeasurements	151
Tag	157
TimeDifferencesImpl< T >	164
TimeTaggerBase	181
TimeTagger	168
TimeTaggerNetwork	193
TimeTaggerVirtual	207
TimeTagStreamBuffer	213

Chapter 4

Class Index

4.1 Class List

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

Coincidence	
Coincidence monitor for one channel group	21
Coincidences	
Coincidence monitor for many channel groups	23
Combiner	
Combine some channels in a virtual channel which has a tick for each tick in the input channels	25
ConstantFractionDiscriminator	
Virtual CFD implementation which returns the mean time between a raising and a falling pair of edges	28
Correlation	
Auto- and Cross-correlation measurement	31
CountBetweenMarkers	
Simple counter where external marker signals determine the bins	35
Counter	
Simple counter on one or more channels	39
CounterData	
Helper object as return value for Counter::getDataObject	43
Countrate	
Count rate on one or more channels	46
CustomLogger	
Helper class for setLogger	49
CustomMeasurementBase	
Helper class for custom measurements in Python and C#	51
DelayedChannel	
Simple delayed queue	55
Dump	
Dump all time tags to a file	59
Event	
Object for the return value of Scope::getData	61
EventGenerator	
Generate predefined events in a virtual channel relative to a trigger event	62
FastBinning	
Helper class for fast division with a constant divisor	65
FileReader	
Reads tags from the disk files, which has been created by FileWriter	67

FileWriter	Compresses and stores all time tags to a file	70
Flim	Fluorescence lifetime imaging	75
FlimAbstract	Interface for FLIM measurements, Flim and FlimBase classes inherit from it	84
FlimBase	Basic measurement, containing a minimal set of features for efficiency purposes	92
FlimFrameInfo	Object for storing the state of Flim::getCurrentFrameEx	95
FrequencyMultiplier	The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter	98
FrequencyStability	Allan deviation (and related metrics) calculator	101
FrequencyStabilityData	Return data object for FrequencyStability::getData	104
GatedChannel	An input channel is gated by a gate channel	107
Histogram	Accumulate time differences into a histogram	110
Histogram2D	A 2-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy	113
HistogramLogBins	Accumulate time differences into a histogram with logarithmic increasing bin sizes	117
HistogramND	A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy	121
Iterator	Deprecated simple event queue	124
IteratorBase	Base class for all iterators	127
OrderedBarrier	Helper for implementing parallel measurements	136
OrderedPipeline	Helper for implementing parallel measurements	137
OrderedBarrier::OrderInstance	Internal object for serialization	138
Sampler	Triggered sampling measurement	139
Scope	Scope measurement	143
SoftwareClockState	146
StartStop	Simple start-stop measurement	148
SynchronizedMeasurements	Start, stop and clear several measurements synchronized	151
SyntheticSingleTag	Synthetic trigger timetag generator	154
Tag	Single event on a channel	157
TimeDifferences	Accumulates the time differences between clicks on two channels in one or more histograms	159
TimeDifferencesImpl< T >	164
TimeDifferencesND	Accumulates the time differences between clicks on two channels in a multi-dimensional histogram	164

TimeTagger	
Backend for the TimeTagger	168
TimeTaggerBase	
Basis interface for all Time Tagger classes	181
TimeTaggerNetwork	
Network TimeTagger client	193
TimeTaggerVirtual	
Virtual TimeTagger based on dump files	207
TimeTagStream	
Access the time tag stream	211
TimeTagStreamBuffer	
Return object for TimeTagStream::getData	213
TriggerOnCountrate	
Inject trigger events when exceeding or falling below a given count rate within a rolling time window	216

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Iterators.h	221
TimeTagger.h	225

Chapter 6

Module Documentation

6.1 Implementations with a Time Tagger interface

Classes

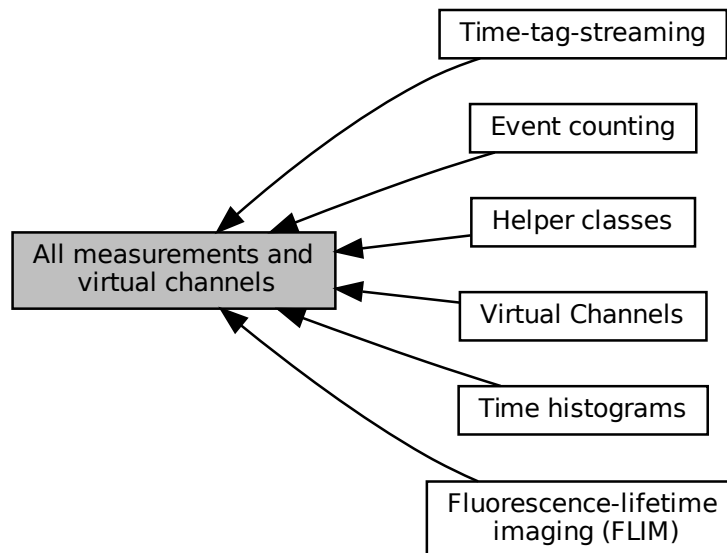
- class [TimeTaggerBase](#)
Basis interface for all Time Tagger classes.
- class [TimeTaggerVirtual](#)
virtual [TimeTagger](#) based on dump files
- class [TimeTagger](#)
backend for the [TimeTagger](#).

6.1.1 Detailed Description

6.2 All measurements and virtual channels

Base iterators for photon counting applications.

Collaboration diagram for All measurements and virtual channels:



Modules

- [Event counting](#)
- [Time histograms](#)

This section describes various measurements that calculate time differences between events and accumulate the results into a histogram.

- [Fluorescence-lifetime imaging \(FLIM\)](#)

This section describes the [Flim](#) related measurements classes of the Time Tagger API.

- [Time-tag-streaming](#)

Measurement classes described in this section provide direct access to the time tag stream with minimal or no pre-processing.

- [Helper classes](#)
- [Virtual Channels](#)

Classes

- class [HistogramND](#)

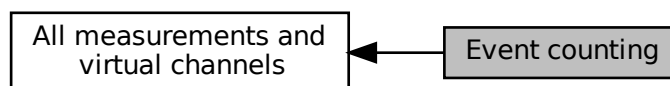
A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.

6.2.1 Detailed Description

Base iterators for photon counting applications.

6.3 Event counting

Collaboration diagram for Event counting:



Classes

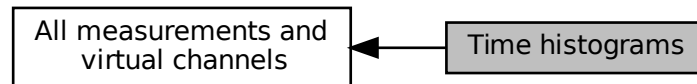
- class [CountBetweenMarkers](#)
a simple counter where external marker signals determine the bins
- class [Counter](#)
a simple counter on one or more channels
- class [Countrate](#)
count rate on one or more channels

6.3.1 Detailed Description

6.4 Time histograms

This section describes various measurements that calculate time differences between events and accumulate the results into a histogram.

Collaboration diagram for Time histograms:



Classes

- class [StartStop](#)
simple start-stop measurement
- class [TimeDifferences](#)
Accumulates the time differences between clicks on two channels in one or more histograms.
- class [Histogram2D](#)
A 2-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.
- class [TimeDifferencesND](#)
Accumulates the time differences between clicks on two channels in a multi-dimensional histogram.
- class [Histogram](#)
Accumulate time differences into a histogram.
- class [HistogramLogBins](#)
Accumulate time differences into a histogram with logarithmic increasing bin sizes.
- class [Correlation](#)
Auto- and Cross-correlation measurement.

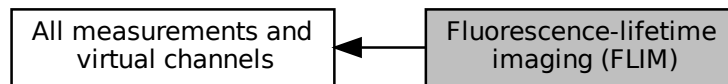
6.4.1 Detailed Description

This section describes various measurements that calculate time differences between events and accumulate the results into a histogram.

6.5 Fluorescence-lifetime imaging (FLIM)

This section describes the [Flim](#) related measurements classes of the Time Tagger API.

Collaboration diagram for Fluorescence-lifetime imaging (FLIM):



Classes

- class [FlimBase](#)
basic measurement, containing a minimal set of features for efficiency purposes
- class [Flim](#)
Fluorescence lifetime imaging.

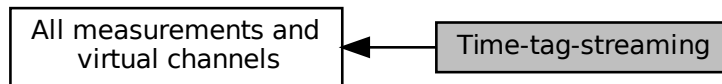
6.5.1 Detailed Description

This section describes the [Flim](#) related measurements classes of the Time Tagger API.

6.6 Time-tag-streaming

Measurement classes described in this section provide direct access to the time tag stream with minimal or no pre-processing.

Collaboration diagram for Time-tag-streaming:



Classes

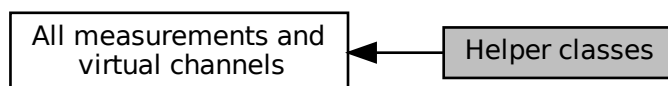
- class [Iterator](#)
a deprecated simple event queue
- class [TimeTagStream](#)
access the time tag stream
- class [Dump](#)
dump all time tags to a file
- class [Scope](#)
a scope measurement
- class [FileWriter](#)
compresses and stores all time tags to a file
- class [FileReader](#)
Reads tags from the disk files, which has been created by [FileWriter](#).
- class [Sampler](#)
a triggered sampling measurement

6.6.1 Detailed Description

Measurement classes described in this section provide direct access to the time tag stream with minimal or no pre-processing.

6.7 Helper classes

Collaboration diagram for Helper classes:



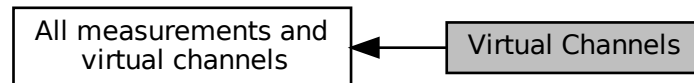
Classes

- class [SynchronizedMeasurements](#)
start, stop and clear several measurements synchronized
- class [CustomMeasurementBase](#)
Helper class for custom measurements in Python and C#.
- class [SyntheticSingleTag](#)
synthetic trigger timetag generator.
- class [FrequencyStability](#)
Allan deviation (and related metrics) calculator.

6.7.1 Detailed Description

6.8 Virtual Channels

Collaboration diagram for Virtual Channels:



Classes

- class [Combiner](#)
Combine some channels in a virtual channel which has a tick for each tick in the input channels.
- class [Coincidences](#)
a coincidence monitor for many channel groups
- class [Coincidence](#)
a coincidence monitor for one channel group
- class [DelayedChannel](#)
a simple delayed queue
- class [TriggerOnCountrate](#)
Inject trigger events when exceeding or falling below a given count rate within a rolling time window.
- class [GatedChannel](#)
An input channel is gated by a gate channel.
- class [FrequencyMultiplier](#)
The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.
- class [ConstantFractionDiscriminator](#)
a virtual CFD implementation which returns the mean time between a raising and a falling pair of edges
- class [EventGenerator](#)
Generate predefined events in a virtual channel relative to a trigger event.

6.8.1 Detailed Description

Virtual channels are software-defined channels as compared to the real input channels. Virtual channels can be understood as a stream flow processing units. They have an input through which they receive time-tags from a real or another virtual channel and output to which they send processed time-tags.

Virtual channels are used as input channels to the measurement classes the same way as real channels. Since the virtual channels are created during run-time, the corresponding channel number(s) are assigned dynamically and can be retrieved using `getChannel()` or `getChannels()` methods of virtual channel object.

Chapter 7

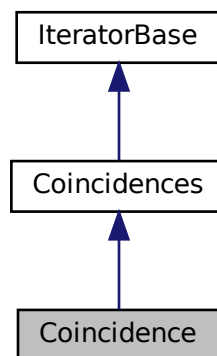
Class Documentation

7.1 Coincidence Class Reference

a coincidence monitor for one channel group

```
#include <Iterators.h>
```

Inheritance diagram for Coincidence:



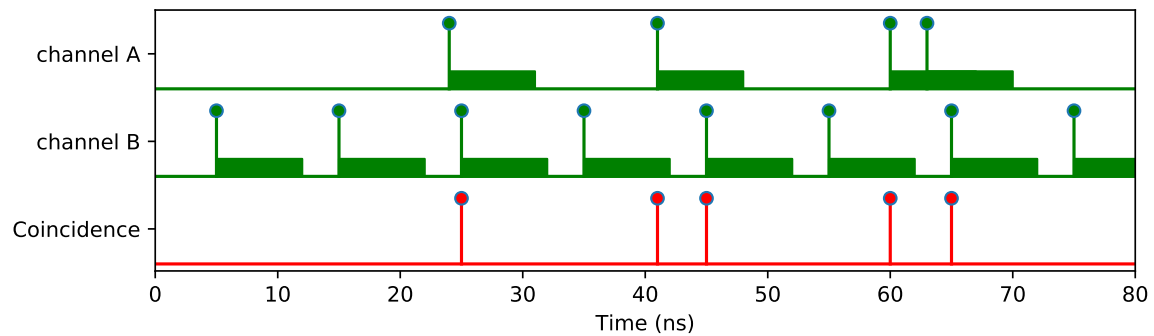
Public Member Functions

- `Coincidence` (`TimeTaggerBase *tagger`, `std::vector< channel_t > channels`, `timestamp_t coincidence`↔
`Window=1000`, `CoincidenceTimestamp timestamp=CoincidenceTimestamp::Last`)
construct a coincidence
- `channel_t getChannel ()`
virtual channel which contains the coincidences

Additional Inherited Members

7.1.1 Detailed Description

a coincidence monitor for one channel group



Monitor coincidences for a given channel groups passed by the constructor. A coincidence is event is detected when all selected channels have a click within the given coincidenceWindow [ps] The coincidence will create a virtual events on a virtual channel with the channel number provided by [getChannel\(\)](#). For multiple coincidence channel combinations use the class [Coincidences](#) which outperforms multiple instances of [Coincidence](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Coincidence()

```
Coincidence::Coincidence (
    TimeTaggerBase * tagger,
    std::vector< channel_t > channels,
    timestamp_t coincidenceWindow = 1000,
    CoincidenceTimestamp timestamp = CoincidenceTimestamp::Last ) [inline]
```

construct a coincidence

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>channels</i>	vector of channels to match
<i>coincidenceWindow</i>	max distance between all clicks for a coincidence [ps]
<i>timestamp</i>	type of timestamp for virtual channel (Last, Average, First, ListedFirst)

7.1.3 Member Function Documentation

7.1.3.1 getChannel()

```
channel_t Coincidence::getChannel ( ) [inline]
```

virtual channel which contains the coincidences

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

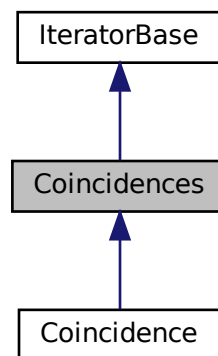
- [Iterators.h](#)

7.2 Coincidences Class Reference

a coincidence monitor for many channel groups

```
#include <Iterators.h>
```

Inheritance diagram for Coincidences:



Public Member Functions

- [Coincidences](#) ([TimeTaggerBase](#) *tagger, std::vector< std::vector< [channel_t](#) >> coincidenceGroups, [timestamp_t](#) coincidenceWindow, [CoincidenceTimestamp](#) timestamp=[CoincidenceTimestamp::Last](#))
construct a [Coincidences](#)
- [~Coincidences](#) ()
- std::vector< [channel_t](#) > [getChannels](#) ()
fetches the block of virtual channels for those coincidence groups
- void [setCoincidenceWindow](#) ([timestamp_t](#) coincidenceWindow)

Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state

Additional Inherited Members

7.2.1 Detailed Description

a coincidence monitor for many channel groups

Monitor coincidences for given coincidence groups passed by the constructor. A coincidence is hereby defined as for a given coincidence group a) the incoming is part of this group b) at least tag arrived within the coincidenceWindow [ps] for all other channels of this coincidence group Each coincidence will create a virtual event. The block of event IDs for those coincidence group can be fetched.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Coincidences()

```
Coincidences::Coincidences (
    TimeTaggerBase * tagger,
    std::vector< std::vector< channel_t >> coincidenceGroups,
    timestamp_t coincidenceWindow,
    CoincidenceTimestamp timestamp = CoincidenceTimestamp::Last )
```

construct a [Coincidences](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>coincidenceGroups</i>	a vector of channels defining the coincidences
<i>coincidenceWindow</i>	the size of the coincidence window in picoseconds
<i>timestamp</i>	type of timestamp for virtual channel (Last, Average, First, ListedFirst)

7.2.2.2 ~Coincidences()

```
Coincidences::~Coincidences ( )
```

7.2.3 Member Function Documentation

7.2.3.1 getChannels()

```
std::vector<channel_t> Coincidences::getChannels ( )
```

fetches the block of virtual channels for those coincidence groups

7.2.3.2 next_impl()

```
bool Coincidences::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.2.3.3 setCoincidenceWindow()

```
void Coincidences::setCoincidenceWindow (
    timestamp_t coincidenceWindow )
```

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

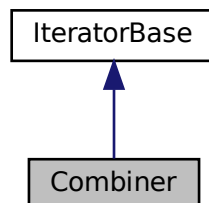
- [Iterators.h](#)

7.3 Combiner Class Reference

Combine some channels in a virtual channel which has a tick for each tick in the input channels.

```
#include <Iterators.h>
```

Inheritance diagram for Combiner:



Public Member Functions

- `Combiner` (`TimeTaggerBase *tagger`, `std::vector< channel_t > channels`)
construct a combiner
- `~Combiner` ()
- `void getChannelCounts` (`std::function< int64_t *(size_t)> array_out`)
get sum of counts
- `void getData` (`std::function< int64_t *(size_t)> array_out`)
get sum of counts
- `channel_t getChannel` ()
the new virtual channel

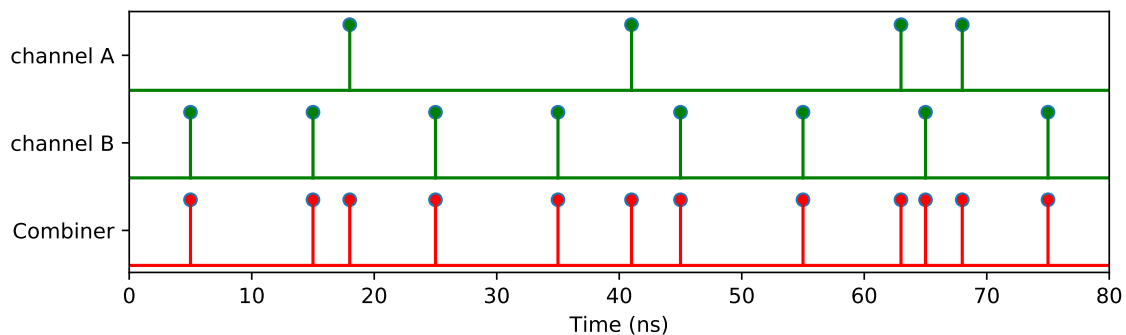
Protected Member Functions

- `bool next_impl` (`std::vector< Tag > &incoming_tags`, `timestamp_t begin_time`, `timestamp_t end_time`) override
update iterator state
- `void clear_impl` () override
*clear *iterator* state.*

Additional Inherited Members

7.3.1 Detailed Description

Combine some channels in a virtual channel which has a tick for each tick in the input channels.



This iterator can be used to get aggregation channels, eg if you want to monitor the countrate of the sum of two channels.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Combiner()

```
Combiner::Combiner (
    TimeTaggerBase * tagger,
    std::vector< channel_t > channels )
```

construct a combiner

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>channels</i>	vector of channels to combine

7.3.2.2 ~Combiner()

```
Combiner::~~Combiner ( )
```

7.3.3 Member Function Documentation**7.3.3.1 clear_impl()**

```
void Combiner::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.3.3.2 getChannel()

```
channel_t Combiner::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

7.3.3.3 getChannelCounts()

```
void Combiner::getChannelCounts (
    std::function< int64_t *(size_t)> array_out )
```

get sum of counts

For reference, this iterators sums up how much ticks are generated because of which input channel. So this functions returns an array with one value per input channel.

7.3.3.4 getData()

```
void Combiner::getData (
    std::function< int64_t *(size_t)> array_out )
```

get sum of counts

deprecated, use getChannelCounts instead.

7.3.3.5 next_impl()

```
bool Combiner::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

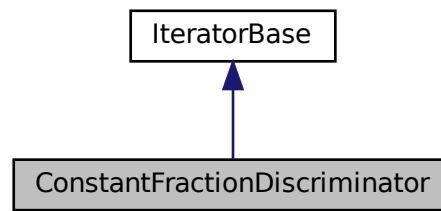
- [Iterators.h](#)

7.4 ConstantFractionDiscriminator Class Reference

a virtual CFD implementation which returns the mean time between a raising and a falling pair of edges

```
#include <Iterators.h>
```

Inheritance diagram for ConstantFractionDiscriminator:



Public Member Functions

- [ConstantFractionDiscriminator](#) ([TimeTaggerBase](#) *tagger, [std::vector](#)< [channel_t](#) > channels, [timestamp_t](#) search_window)
constructor of a [ConstantFractionDiscriminator](#)
- [~ConstantFractionDiscriminator](#) ()
- [std::vector](#)< [channel_t](#) > [getChannels](#) ()
the list of new virtual channels

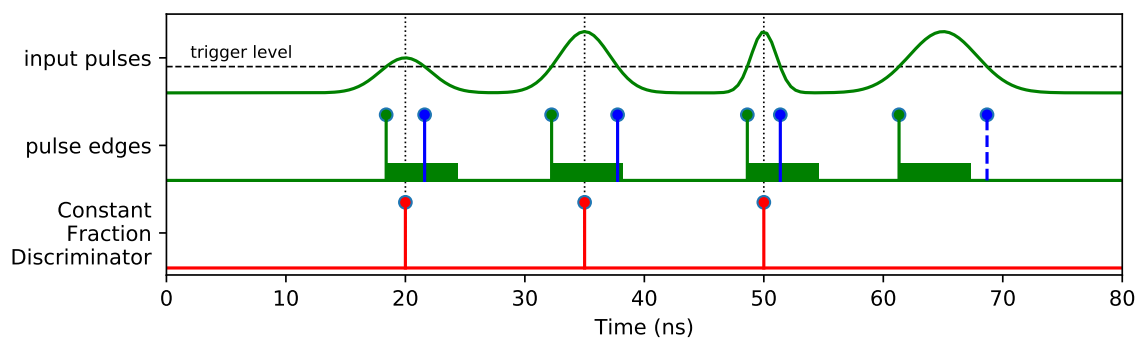
Protected Member Functions

- [bool](#) [next_impl](#) ([std::vector](#)< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- [void](#) [on_start](#) () override
callback when the measurement class is started

Additional Inherited Members

7.4.1 Detailed Description

a virtual CFD implementation which returns the mean time between a raising and a falling pair of edges



7.4.2 Constructor & Destructor Documentation

7.4.2.1 ConstantFractionDiscriminator()

```
ConstantFractionDiscriminator::ConstantFractionDiscriminator (
    TimeTaggerBase * tagger,
    std::vector< channel_t > channels,
    timestamp_t search_window )
```

constructor of a [ConstantFractionDiscriminator](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>channels</i>	list of channels for the CFD, the formers of the raising+falling pairs must be given
<i>search_window</i>	interval for the CFD window, must be positive

7.4.2.2 ~ConstantFractionDiscriminator()

```
ConstantFractionDiscriminator::~~ConstantFractionDiscriminator ( )
```

7.4.3 Member Function Documentation

7.4.3.1 getChannels()

```
std::vector<channel_t> ConstantFractionDiscriminator::getChannels ( )
```

the list of new virtual channels

This function returns the list of new allocated virtual channels. It can be used now in any new measurement class.

7.4.3.2 next_impl()

```
bool ConstantFractionDiscriminator::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.4.3.3 on_start()

```
void ConstantFractionDiscriminator::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

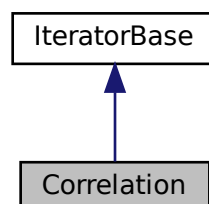
- [Iterators.h](#)

7.5 Correlation Class Reference

Auto- and Cross-correlation measurement.

```
#include <Iterators.h>
```

Inheritance diagram for Correlation:



Public Member Functions

- [Correlation](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) channel_1, [channel_t](#) channel_2=[CHANNEL_UNUSED](#), [timestamp_t](#) binwidth=1000, int n_bins=1000)
constructor of a correlation measurement
- [~Correlation](#) ()
destructor of the [Correlation](#) measurement
- void [getData](#) (std::function< int32_t *(size_t)> array_out)
returns a one-dimensional array of size n_bins containing the histogram
- void [getDataNormalized](#) (std::function< double *(size_t)> array_out)
get the g(2) normalized histogram
- void [getIndex](#) (std::function< long long *(size_t)> array_out)
returns a vector of size n_bins containing the time bins in ps

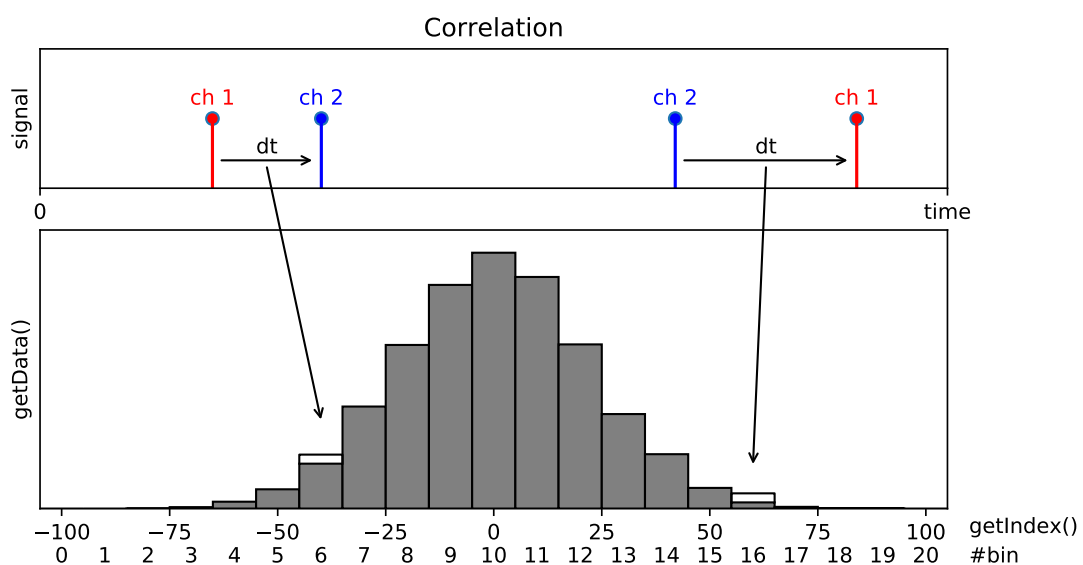
Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.

Additional Inherited Members

7.5.1 Detailed Description

Auto- and Cross-correlation measurement.



Accumulates time differences between clicks on two channels into a histogram, where all clicks are considered both as “start” and “stop” clicks and both positive and negative time differences are calculated.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Correlation()

```
Correlation::Correlation (
    TimeTaggerBase * tagger,
    channel_t channel_1,
    channel_t channel_2 = CHANNEL_UNUSED,
    timestamp_t binwidth = 1000,
    int n_bins = 1000 )
```

constructor of a correlation measurement

Note

When channel_1 is left empty or set to CHANNEL_UNUSED -> an auto-correlation measurement is performed, which is the same as setting channel_1 = channel_2.

Parameters

<i>tagger</i>	time tagger object
<i>channel_1</i>	channel on which (stop) clicks are received
<i>channel_2</i>	channel on which reference clicks (start) are received
<i>binwidth</i>	bin width in ps
<i>n_bins</i>	the number of bins in the resulting histogram

7.5.2.2 ~Correlation()

```
Correlation::~~Correlation ( )
```

destructor of the [Correlation](#) measurement

7.5.3 Member Function Documentation

7.5.3.1 clear_impl()

```
void Correlation::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.5.3.2 getData()

```
void Correlation::getData (
    std::function< int32_t *(size_t)> array_out )
```

returns a one-dimensional array of size n_bins containing the histogram

Parameters

<i>array_out</i>	allocator callback for managed return values
------------------	--

7.5.3.3 getDataNormalized()

```
void Correlation::getDataNormalized (
    std::function< double *(size_t)> array_out )
```

get the g(2) normalized histogram

Return the data normalized as: $g^{(2)}(\tau) = \frac{\Delta t}{\text{binwidth}(\tau) \cdot N_1 \cdot N_2} \cdot \text{histogram}(\tau)$

This is normalized in such a way that a perfectly uncorrelated signals would result in a histogram with a mean value of bins equal to one.

Parameters

<i>array_out</i>	allocator callback for managed return values
------------------	--

7.5.3.4 getIndex()

```
void Correlation::getIndex (
    std::function< long long *(size_t)> array_out )
```

returns a vector of size n_bins containing the time bins in ps

Parameters

<i>array_out</i>	allocator callback for managed return values
------------------	--

7.5.3.5 next_impl()

```
bool Correlation::next_impl (
    std::vector< Tag > & incoming_tags,
```

```
timestamp_t begin_time,
timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

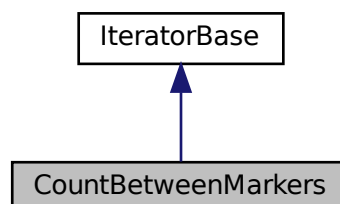
- [Iterators.h](#)

7.6 CountBetweenMarkers Class Reference

a simple counter where external marker signals determine the bins

```
#include <Iterators.h>
```

Inheritance diagram for CountBetweenMarkers:



Public Member Functions

- `CountBetweenMarkers` (`TimeTaggerBase *tagger`, `channel_t click_channel`, `channel_t begin_channel`, `channel_t end_channel=CHANNEL_UNUSED`, `int32_t n_values=1000`)
constructor of `CountBetweenMarkers`
- `~CountBetweenMarkers` ()
- `bool ready` ()
Returns true when the entire array is filled.
- `void getData` (`std::function< int32_t *(size_t)> array_out`)
Returns array of size `n_values` containing the acquired counter values.
- `void getBinWidths` (`std::function< long long *(size_t)> array_out`)
fetches the widths of each bins
- `void getIndex` (`std::function< long long *(size_t)> array_out`)
fetches the starting time of each bin

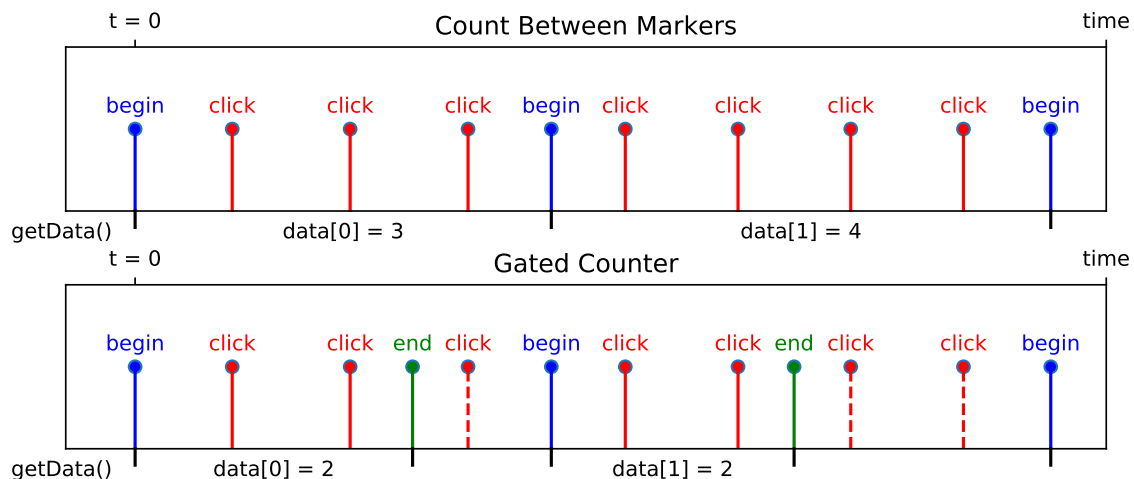
Protected Member Functions

- `bool next_impl` (`std::vector< Tag > &incoming_tags`, `timestamp_t begin_time`, `timestamp_t end_time`) override
update iterator state
- `void clear_impl` () override
clear `Iterator` state.

Additional Inherited Members

7.6.1 Detailed Description

a simple counter where external marker signals determine the bins



Counts events on a single channel within the time indicated by a “start” and “stop” signals. The bin edges between which counts are accumulated are determined by one or more hardware triggers. Specifically, the measurement records data into a vector of length `n_values` (initially filled with zeros). It waits for tags on the `begin_channel`. When a tag is detected on the `begin_channel` it starts counting tags on the `click_channel`. When the next tag is detected on the `begin_channel` it stores the current counter value as the next entry in the data vector, resets the counter to zero and starts accumulating counts again. If an `end_channel` is specified, the measurement stores the current counter value and resets the counter when a tag is detected on the `end_channel` rather than the `begin_channel`. You can use this, e.g., to accumulate counts within a gate by using rising edges on one channel as the `begin_channel` and falling edges on the same channel as the `end_channel`. The accumulation time for each value can be accessed via `getBinWidths()`. The measurement stops when all entries in the data vector are filled.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 CountBetweenMarkers()

```
CountBetweenMarkers::CountBetweenMarkers (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t begin_channel,
    channel_t end_channel = CHANNEL_UNUSED,
    int32_t n_values = 1000 )
```

constructor of [CountBetweenMarkers](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel that increases the count
<i>begin_channel</i>	channel that triggers beginning of counting and stepping to the next value
<i>end_channel</i>	channel that triggers end of counting
<i>n_values</i>	the number of counter values to be stored

7.6.2.2 ~CountBetweenMarkers()

```
CountBetweenMarkers::~~CountBetweenMarkers ( )
```

7.6.3 Member Function Documentation

7.6.3.1 clear_impl()

```
void CountBetweenMarkers::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.6.3.2 getBinWidths()

```
void CountBetweenMarkers::getBinWidths (
    std::function< long long *(size_t)> array_out )
```

fetches the widths of each bins

7.6.3.3 getData()

```
void CountBetweenMarkers::getData (
    std::function< int32_t *(size_t)> array_out )
```

Returns array of size `n_values` containing the acquired counter values.

7.6.3.4 getIndex()

```
void CountBetweenMarkers::getIndex (
    std::function< long long *(size_t)> array_out )
```

fetches the starting time of each bin

7.6.3.5 next_impl()

```
bool CountBetweenMarkers::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the `next_impl()` method. The `next_impl()` function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.6.3.6 ready()

```
bool CountBetweenMarkers::ready ( )
```

Returns true when the entire array is filled.

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

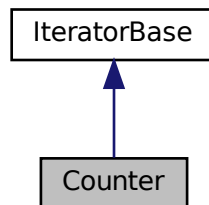
- [Iterators.h](#)

7.7 Counter Class Reference

a simple counter on one or more channels

```
#include <Iterators.h>
```

Inheritance diagram for Counter:



Public Member Functions

- [Counter](#) ([TimeTaggerBase](#) *tagger, std::vector< [channel_t](#) > channels, [timestamp_t](#) binwidth=1000000000, int32_t n_values=1)
construct a counter
- [~Counter](#) ()
- void [getData](#) (std::function< int32_t *(size_t, size_t)> array_out, bool rolling=true)
An array of size 'number of channels' by n_values containing the current values of the circular buffer (counts in each bin).
- void [getDataNormalized](#) (std::function< double *(size_t, size_t)> array_out, bool rolling=true)
get countrate in Hz
- void [getDataTotalCounts](#) (std::function< uint64_t *(size_t)> array_out)
get the total amount of clicks per channel since the last clear including the currently integrating bin
- void [getIndex](#) (std::function< long long *(size_t)> array_out)
A vector of size n_values containing the time bins in ps.
- [CounterData](#) [getDataObject](#) (bool remove=false)
Fetch the most recent up to n_values bins, which have not been removed before.

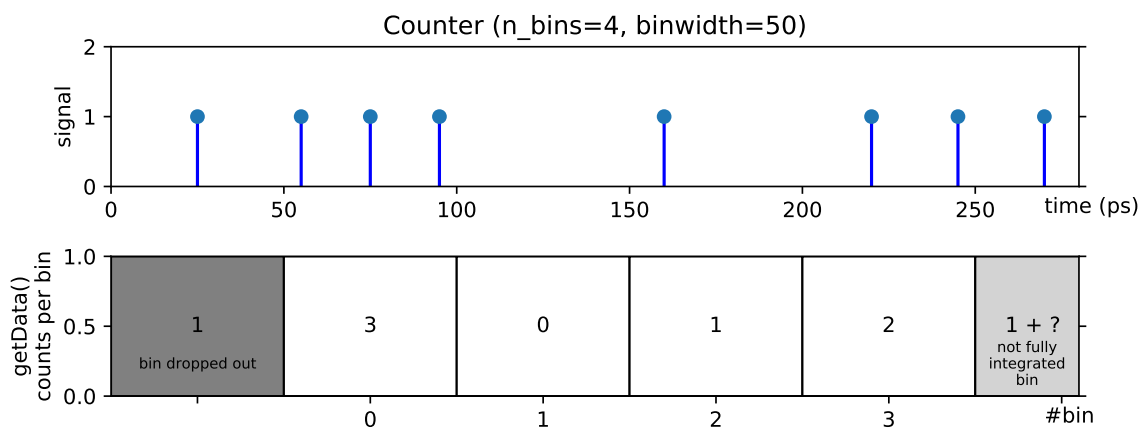
Protected Member Functions

- bool `next_impl` (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override
update iterator state
- void `clear_impl` () override
clear iterator state.
- void `on_start` () override
callback when the measurement class is started

Additional Inherited Members

7.7.1 Detailed Description

a simple counter on one or more channels



Time trace of the count rate on one or more channels. Specifically, this measurement repeatedly counts tags on one or more channels within a time interval binwidth and stores the results in a two-dimensional array of size 'number of channels' by 'n_values'. The array is treated as a circular buffer, which means all values in the array are shifted by one position when a new value is generated. The last entry in the array is always the most recent value.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Counter()

```
Counter::Counter (
    TimeTaggerBase * tagger,
    std::vector< channel_t > channels,
    timestamp_t binwidth = 1000000000,
    int32_t n_values = 1 )
```

construct a counter

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>channels</i>	channels to count on
<i>binwidth</i>	counts are accumulated for binwidth picoseconds
<i>n_values</i>	number of counter values stored (for each channel)

7.7.2.2 ~Counter()

```
Counter::~Counter ( )
```

7.7.3 Member Function Documentation

7.7.3.1 clear_impl()

```
void Counter::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.7.3.2 getData()

```
void Counter::getData (
    std::function< int32_t *(size_t, size_t)> array_out,
    bool rolling = true )
```

An array of size 'number of channels' by *n_values* containing the current values of the circular buffer (counts in each bin).

Parameters

<i>array_out</i>	allocator callback for managed return values
<i>rolling</i>	if true, the returning array starts with the oldest data and goes up to the newest data

7.7.3.3 getDataNormalized()

```
void Counter::getDataNormalized (
    std::function< double *(size_t, size_t)> array_out,
    bool rolling = true )
```

get countrate in Hz

the counts are normalized are copied to a newly allocated allocated memory, an the pointer to this location is returned. Invalid bins are replaced with NaNs.

Parameters

<i>array_out</i>	allocator callback for managed return values
<i>rolling</i>	if true, the returning array starts with the oldest data and goes up to the newest data

7.7.3.4 getDataObject()

```
CounterData Counter::getDataObject (
    bool remove = false )
```

Fetch the most recent up to `n_values` bins, which have not been removed before.

This method allows atomic polling of bins, so each bin is guaranteed to be returned exactly once.

Parameters

<i>remove</i>	remove all fetched bins
---------------	-------------------------

Returns

a [CounterData](#) object, which contains all data of the fetches bins

7.7.3.5 getDataTotalCounts()

```
void Counter::getDataTotalCounts (
    std::function< uint64_t *(size_t)> array_out )
```

get the total amount of clicks per channel since the last clear including the currently integrating bin

7.7.3.6 getIndex()

```
void Counter::getIndex (
    std::function< long long *(size_t)> array_out )
```

A vector of size `n_values` containing the time bins in ps.

7.7.3.7 next_impl()

```
bool Counter::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.7.3.8 on_start()

```
void Counter::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.8 CounterData Class Reference

Helper object as return value for [Counter::getDataObject](#).

```
#include <Iterators.h>
```

Public Member Functions

- [~CounterData](#) ()
- void [getData](#) (std::function< int32_t *(size_t, size_t)> array_out)
get the amount of clicks per bin and per channel
- void [getDataNormalized](#) (std::function< double *(size_t, size_t)> array_out)
get the average rate of clicks per bin and per channel
- void [getDataTotalCounts](#) (std::function< uint64_t *(size_t)> array_out)
get the total amount of clicks per channel since the last clear up to the most rececnt bin
- void [getIndex](#) (std::function< long long *(size_t)> array_out)
get an index which corresponds to the timestamp of these bins
- void [getTime](#) (std::function< long long *(size_t)> array_out)
get the timestamp of the bins since the last clear
- void [getOverflowMask](#) (std::function< signed char *(size_t)> array_out)
get if the bins were in overflow
- void [getChannels](#) (std::function< int *(size_t)> array_out)
get the configured list of channels

Public Attributes

- const uint32_t [size](#)
number of returned bins
- const uint32_t [dropped_bins](#)
number of bins which have been dropped because n_bins has been exceeded, usually 0
- const bool [overflow](#)
has anything been in overflow mode

7.8.1 Detailed Description

Helper object as return value for [Counter::getDataObject](#).

This object stores the result of up to n_values bins.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 ~CounterData()

```
CounterData::~CounterData ( )
```

7.8.3 Member Function Documentation

7.8.3.1 getChannels()

```
void CounterData::getChannels (
    std::function< int *(size_t)> array_out )
```

get the configured list of channels

7.8.3.2 getData()

```
void CounterData::getData (
    std::function< int32_t *(size_t, size_t)> array_out )
```

get the amount of clicks per bin and per channel

7.8.3.3 getDataNormalized()

```
void CounterData::getDataNormalized (
    std::function< double *(size_t, size_t)> array_out )
```

get the average rate of clicks per bin and per channel

7.8.3.4 getDataTotalCounts()

```
void CounterData::getDataTotalCounts (
    std::function< uint64_t *(size_t)> array_out )
```

get the total amount of clicks per channel since the last clear up to the most recent bin

7.8.3.5 getIndex()

```
void CounterData::getIndex (
    std::function< long long *(size_t)> array_out )
```

get an index which corresponds to the timestamp of these bins

7.8.3.6 getOverflowMask()

```
void CounterData::getOverflowMask (
    std::function< signed char *(size_t)> array_out )
```

get if the bins were in overflow

7.8.3.7 getTime()

```
void CounterData::getTime (
    std::function< long long *(size_t)> array_out )
```

get the timestamp of the bins since the last clear

7.8.4 Member Data Documentation

7.8.4.1 dropped_bins

```
const uint32_t CounterData::dropped_bins
```

number of bins which have been dropped because n_bins has been exceeded, usually 0

7.8.4.2 overflow

```
const bool CounterData::overflow
```

has anything been in overflow mode

7.8.4.3 size

```
const uint32_t CounterData::size
```

number of returned bins

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

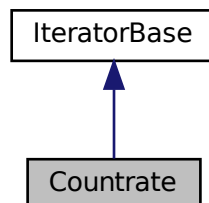
- [Iterators.h](#)

7.9 Countrate Class Reference

count rate on one or more channels

```
#include <Iterators.h>
```

Inheritance diagram for Countrate:



Public Member Functions

- `Countrate` (`TimeTaggerBase *tagger`, `std::vector< channel_t > channels`)
constructor of `Countrate`
- `~Countrate` ()
- `void getData` (`std::function< double *(size_t)> array_out`)
get the count rates
- `void getCountsTotal` (`std::function< int64_t *(size_t)> array_out`)
get the total amount of events

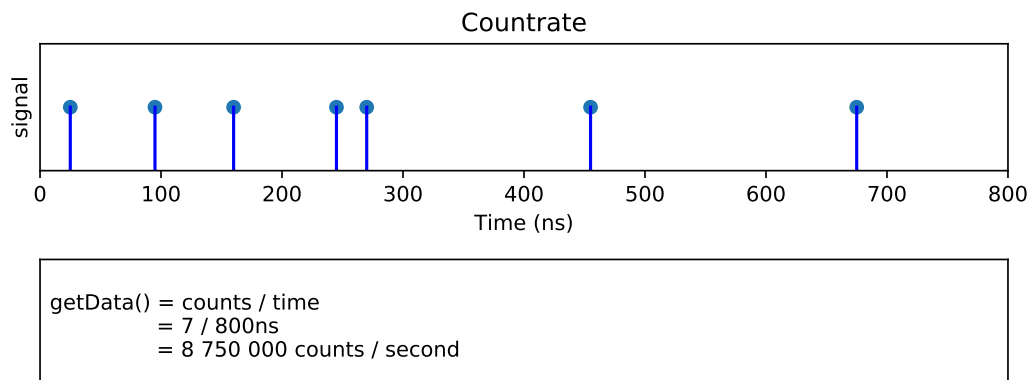
Protected Member Functions

- `bool next_impl` (`std::vector< Tag > &incoming_tags`, `timestamp_t begin_time`, `timestamp_t end_time`) override
update iterator state
- `void clear_impl` () override
*clear *Iterator* state.*
- `void on_start` () override
callback when the measurement class is started

Additional Inherited Members

7.9.1 Detailed Description

count rate on one or more channels



Measures the average count rate on one or more channels. Specifically, it counts incoming clicks and determines the time between the initial click and the latest click. The number of clicks divided by the time corresponds to the average countrate since the initial click.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Countrate()

```

Countrate::Countrate (
    TimeTaggerBase * tagger,
    std::vector< channel_t > channels )

```

constructor of `Countrate`

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>channels</i>	the channels to count on

7.9.2.2 ~Countrate()

```
Countrate::~~Countrate ( )
```

7.9.3 Member Function Documentation**7.9.3.1 clear_impl()**

```
void Countrate::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.9.3.2 getCountsTotal()

```
void Countrate::getCountsTotal (
    std::function< int64_t *(size_t)> array_out )
```

get the total amount of events

Returns the total amount of events per channel as an array.

7.9.3.3 getData()

```
void Countrate::getData (
    std::function< double *(size_t)> array_out )
```

get the count rates

Returns the average rate of events per second per channel as an array.

7.9.3.4 next_impl()

```
bool Countrate::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.9.3.5 on_start()

```
void Countrate::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.10 CustomLogger Class Reference

Helper class for setLogger.

```
#include <TimeTagger.h>
```

Public Member Functions

- [CustomLogger](#) ()
- virtual [~CustomLogger](#) ()
- void [enable](#) ()
- void [disable](#) ()
- virtual void [Log](#) (int level, const std::string &msg)=0

7.10.1 Detailed Description

Helper class for setLogger.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 CustomLogger()

```
CustomLogger::CustomLogger ( )
```

7.10.2.2 ~CustomLogger()

```
virtual CustomLogger::~~CustomLogger ( ) [virtual]
```

7.10.3 Member Function Documentation

7.10.3.1 disable()

```
void CustomLogger::disable ( )
```

7.10.3.2 enable()

```
void CustomLogger::enable ( )
```

7.10.3.3 Log()

```
virtual void CustomLogger::Log (
    int level,
    const std::string & msg ) [pure virtual]
```

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

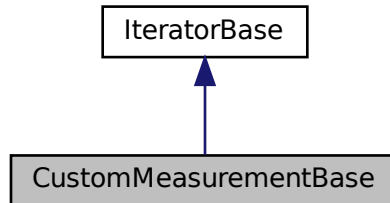
- [TimeTagger.h](#)

7.11 CustomMeasurementBase Class Reference

Helper class for custom measurements in Python and C#.

```
#include <Iterators.h>
```

Inheritance diagram for CustomMeasurementBase:



Public Member Functions

- `~CustomMeasurementBase ()` override
- void `register_channel (channel_t channel)`
- void `unregister_channel (channel_t channel)`
- void `finalize_init ()`
- bool `is_running ()` const
- void `_lock ()`
- void `_unlock ()`

Static Public Member Functions

- static void `stop_all_custom_measurements ()`

Protected Member Functions

- `CustomMeasurementBase (TimeTaggerBase *tagger)`
- virtual bool `next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)` override
update iterator state
- virtual void `next_impl_cs (void *tags_ptr, uint64_t num_tags, timestamp_t begin_time, timestamp_t end_time)`
- virtual void `clear_impl ()` override
clear iterator state.
- virtual void `on_start ()` override
callback when the measurement class is started
- virtual void `on_stop ()` override
callback when the measurement class is stopped

Additional Inherited Members

7.11.1 Detailed Description

Helper class for custom measurements in Python and C#.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 CustomMeasurementBase()

```
CustomMeasurementBase::CustomMeasurementBase (
    TimeTaggerBase * tagger ) [protected]
```

7.11.2.2 ~CustomMeasurementBase()

```
CustomMeasurementBase::~~CustomMeasurementBase ( ) [override]
```

7.11.3 Member Function Documentation

7.11.3.1 _lock()

```
void CustomMeasurementBase::_lock ( )
```

7.11.3.2 _unlock()

```
void CustomMeasurementBase::_unlock ( )
```

7.11.3.3 clear_impl()

```
virtual void CustomMeasurementBase::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.11.3.4 finalize_init()

```
void CustomMeasurementBase::finalize_init ( )
```

7.11.3.5 is_running()

```
bool CustomMeasurementBase::is_running ( ) const
```

7.11.3.6 next_impl()

```
virtual bool CustomMeasurementBase::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.11.3.7 next_impl_cs()

```
virtual void CustomMeasurementBase::next_impl_cs (
    void * tags_ptr,
    uint64_t num_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [protected], [virtual]
```

7.11.3.8 on_start()

```
virtual void CustomMeasurementBase::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.11.3.9 on_stop()

```
virtual void CustomMeasurementBase::on_stop ( ) [override], [protected], [virtual]
```

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.11.3.10 register_channel()

```
void CustomMeasurementBase::register_channel (
    channel\_t channel )
```

7.11.3.11 stop_all_custom_measurements()

```
static void CustomMeasurementBase::stop_all_custom_measurements ( ) [static]
```

7.11.3.12 unregister_channel()

```
void CustomMeasurementBase::unregister_channel (
    channel\_t channel )
```

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

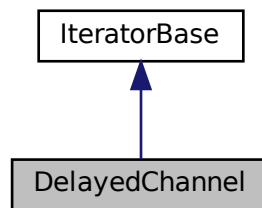
- [Iterators.h](#)

7.12 DelayedChannel Class Reference

a simple delayed queue

```
#include <Iterators.h>
```

Inheritance diagram for DelayedChannel:



Public Member Functions

- `DelayedChannel (TimeTaggerBase *tagger, channel_t input_channel, timestamp_t delay)`
constructor of a `DelayedChannel`
- `DelayedChannel (TimeTaggerBase *tagger, std::vector< channel_t > input_channels, timestamp_t delay)`
constructor of a `DelayedChannel` for delaying many channels at once
- `~DelayedChannel ()`
- `channel_t getChannel ()`
the first new virtual channel
- `std::vector< channel_t > getChannels ()`
the new virtual channels
- `void setDelay (timestamp_t delay)`
set the delay time delay for the cloned tags in the virtual channels. A negative delay will delay all other events.

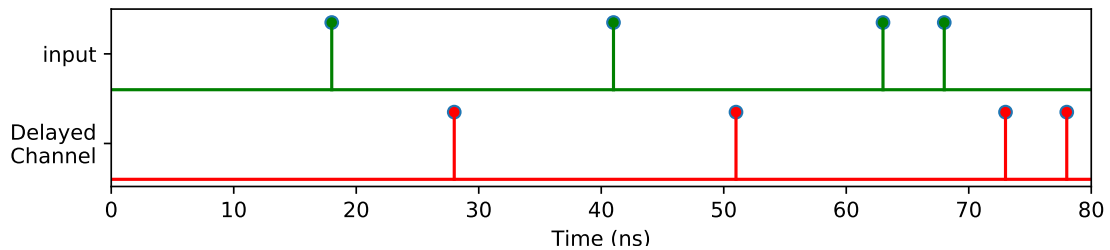
Protected Member Functions

- `bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)` override
update iterator state
- `void on_start ()` override
callback when the measurement class is started

Additional Inherited Members

7.12.1 Detailed Description

a simple delayed queue



A simple first-in first-out queue of delayed event timestamps.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 DelayedChannel() [1/2]

```
DelayedChannel::DelayedChannel (
    TimeTaggerBase * tagger,
    channel_t input_channel,
    timestamp_t delay )
```

constructor of a [DelayedChannel](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>input_channel</i>	channel which is delayed
<i>delay</i>	amount of time to delay

7.12.2.2 DelayedChannel() [2/2]

```
DelayedChannel::DelayedChannel (
    TimeTaggerBase * tagger,
    std::vector< channel_t > input_channels,
    timestamp_t delay )
```

constructor of a [DelayedChannel](#) for delaying many channels at once

This function is not exposed to Python/C#/Matlab/Labview

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>input_channels</i>	channels which will be delayed
<i>delay</i>	amount of time to delay

7.12.2.3 ~DelayedChannel()

```
DelayedChannel::~~DelayedChannel ( )
```

7.12.3 Member Function Documentation**7.12.3.1 getChannel()**

```
channel\_t DelayedChannel::getChannel ( )
```

the first new virtual channel

This function returns the first of the new allocated virtual channels. It can be used now in any new iterator.

7.12.3.2 getChannels()

```
std::vector<channel\_t> DelayedChannel::getChannels ( )
```

the new virtual channels

This function returns the new allocated virtual channels. It can be used now in any new iterator.

7.12.3.3 next_impl()

```
bool DelayedChannel::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.12.3.4 on_start()

```
void DelayedChannel::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.12.3.5 setDelay()

```
void DelayedChannel::setDelay (
    timestamp_t delay )
```

set the delay time delay for the cloned tags in the virtual channels. A negative delay will delay all other events.

Note: When the delay is the same or greater than the previous value all incoming tags will be visible at virtual channel. By applying a shorter delay time, the tags stored in the local buffer will be flushed and won't be visible in the virtual channel.

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

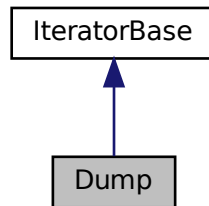
- [Iterators.h](#)

7.13 Dump Class Reference

dump all time tags to a file

```
#include <Iterators.h>
```

Inheritance diagram for Dump:



Public Member Functions

- `Dump` (`TimeTaggerBase` *tagger, `std::string` filename, `int64_t` max_tags, `std::vector`< `channel_t` > channels=`std::vector`< `channel_t` >())
constructor of a `Dump` thread
- `~Dump` ()

Protected Member Functions

- `bool` `next_impl` (`std::vector`< `Tag` > &incoming_tags, `timestamp_t` begin_time, `timestamp_t` end_time) override
update iterator state
- `void` `clear_impl` () override
clear `Iterator` state.
- `void` `on_start` () override
callback when the measurement class is started
- `void` `on_stop` () override
callback when the measurement class is stopped

Additional Inherited Members

7.13.1 Detailed Description

dump all time tags to a file

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Dump()

```
Dump::Dump (
    TimeTaggerBase * tagger,
    std::string filename,
    int64_t max_tags,
    std::vector< channel_t > channels = std::vector< channel_t >() )
```

constructor of a [Dump](#) thread

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>filename</i>	name of the file to dump to
<i>max_tags</i>	stop after this number of tags has been dumped. Negative values will dump forever
<i>channels</i>	channels which are dumped to the file (when empty or not passed all active channels are dumped)

7.13.2.2 ~Dump()

```
Dump::~Dump ( )
```

7.13.3 Member Function Documentation

7.13.3.1 clear_impl()

```
void Dump::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.13.3.2 next_impl()

```
bool Dump::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.13.3.3 on_start()

```
void Dump::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.13.3.4 on_stop()

```
void Dump::on_stop ( ) [override], [protected], [virtual]
```

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.14 Event Struct Reference

Object for the return value of [Scope::getData](#).

```
#include <Iterators.h>
```

Public Attributes

- [timestamp_t](#) time
- [State](#) state

7.14.1 Detailed Description

Object for the return value of [Scope::getData](#).

7.14.2 Member Data Documentation

7.14.2.1 state

[State](#) Event::state

7.14.2.2 time

[timestamp_t](#) Event::time

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

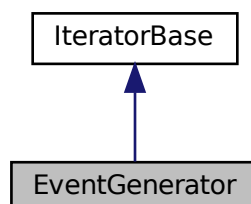
- [Iterators.h](#)

7.15 EventGenerator Class Reference

Generate predefined events in a virtual channel relative to a trigger event.

```
#include <Iterators.h>
```

Inheritance diagram for EventGenerator:



Public Member Functions

- [EventGenerator](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) trigger_channel, [std::vector](#)< [timestamp_t](#) > pattern, [uint64_t](#) trigger_divider=1, [uint64_t](#) divider_offset=0, [channel_t](#) stop_channel=[CHANNEL_UNUSED](#))
construct a event generator
- [~EventGenerator](#) ()
- [channel_t](#) getChannel ()
the new virtual channel

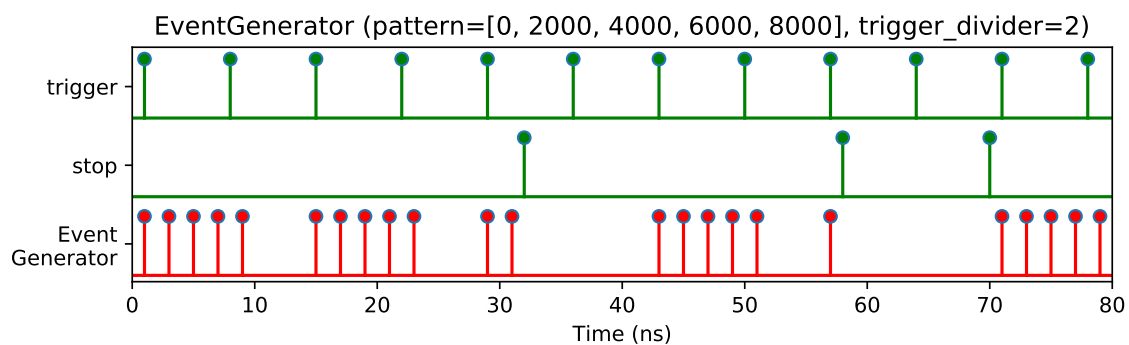
Protected Member Functions

- [bool](#) next_impl ([std::vector](#)< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- [void](#) clear_impl () override
clear iterator state.
- [void](#) on_start () override
callback when the measurement class is started

Additional Inherited Members

7.15.1 Detailed Description

Generate predefined events in a virtual channel relative to a trigger event.



This iterator can be used to generate a predefined series of events, the pattern, relative to a trigger event on a defined channel. A trigger_divider can be used to fire the pattern not on every, but on every n'th trigger received. The trigger_offset can be used to select on which of the triggers the pattern will be generated when trigger trigger_offset_divider is greater than 1. To abort the pattern being generated, a stop_channel can be defined. In case it is the very same as the trigger_channel, the subsequent generated patterns will not overlap.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 EventGenerator()

```
EventGenerator::EventGenerator (
    TimeTaggerBase * tagger,
    channel_t trigger_channel,
    std::vector< timestamp_t > pattern,
    uint64_t trigger_divider = 1,
    uint64_t divider_offset = 0,
    channel_t stop_channel = CHANNEL_UNUSED )
```

construct a event generator

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>trigger_channel</i>	trigger for generating the pattern
<i>pattern</i>	vector of time stamp generated relative to the trigger event
<i>trigger_divider</i>	establishes every how many trigger events a pattern is generated
<i>divider_offset</i>	the offset of the divided trigger when the pattern shall be emitted
<i>stop_channel</i>	channel on which a received event will stop all pending patterns from being generated

7.15.2.2 ~EventGenerator()

```
EventGenerator::~~EventGenerator ( )
```

7.15.3 Member Function Documentation

7.15.3.1 clear_impl()

```
void EventGenerator::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.15.3.2 getChannel()

```
channel_t EventGenerator::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

7.15.3.3 next_impl()

```
bool EventGenerator::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.15.3.4 on_start()

```
void EventGenerator::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.16 FastBinning Class Reference

Helper class for fast division with a constant divisor.

```
#include <Iterators.h>
```

Public Types

- enum [Mode](#) {
[Mode::ConstZero](#), [Mode::Dividend](#), [Mode::PowerOfTwo](#), [Mode::FixedPoint_32](#),
[Mode::FixedPoint_64](#), [Mode::Divide_32](#), [Mode::Divide_64](#) }

Public Member Functions

- [FastBinning](#) ()
- [FastBinning](#) (uint64_t divisor, uint64_t max_duration_)
- template<Mode mode>
uint64_t [divide](#) (uint64_t duration) const
- [Mode](#) [getMode](#) () const

7.16.1 Detailed Description

Helper class for fast division with a constant divisor.

It chooses the method on initialization time and precompile the evaluation functions for all methods.

7.16.2 Member Enumeration Documentation

7.16.2.1 Mode

```
enum FastBinning::Mode [strong]
```

Enumerator

ConstZero	
Dividend	
PowerOfTwo	
FixedPoint_32	
FixedPoint_64	
Divide_32	
Divide_64	

7.16.3 Constructor & Destructor Documentation

7.16.3.1 FastBinning() [1/2]

```
FastBinning::FastBinning ( ) [inline]
```

7.16.3.2 FastBinning() [2/2]

```
FastBinning::FastBinning (
    uint64_t divisor,
    uint64_t max_duration_ )
```

7.16.4 Member Function Documentation

7.16.4.1 divide()

```
template<Mode mode>
uint64_t FastBinning::divide (
    uint64_t duration ) const [inline]
```

7.16.4.2 getMode()

```
Mode FastBinning::getMode ( ) const [inline]
```

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

- [Iterators.h](#)

7.17 FileReader Class Reference

Reads tags from the disk files, which has been created by [FileWriter](#).

```
#include <Iterators.h>
```

Public Member Functions

- [FileReader](#) (std::vector< std::string > filenames)
Creates a file reader with the given filename.
- [FileReader](#) (const std::string &filename)
Creates a file reader with the given filename.
- [~FileReader](#) ()
- bool [hasData](#) ()
Checks if there are still events in the [FileReader](#).
- [TimeTagStreamBuffer getData](#) (uint64_t n_events)
Fetches and delete the next tags from the internal buffer.
- bool [getDataRaw](#) (std::vector< [Tag](#) > &tag_buffer)
Low level file reading.
- std::string [getConfiguration](#) ()
Fetches the overall configuration status of the Time Tagger object, which was serialized in the current file.
- std::string [getLastMarker](#) ()
return the last processed marker from the file.

7.17.1 Detailed Description

Reads tags from the disk files, which has been created by [FileWriter](#).

Its usage is compatible with the [TimeTagStream](#).

7.17.2 Constructor & Destructor Documentation

7.17.2.1 FileReader() [1/2]

```
FileReader::FileReader (
    std::vector< std::string > filenames )
```

Creates a file reader with the given filename.

The file reader automatically continues to read split [FileWriter](#) Streams In case multiple filenames are given, the files will be read in successively.

Parameters

<i>filenames</i>	list of files to read
------------------	-----------------------

7.17.2.2 FileReader() [2/2]

```
FileReader::FileReader (
    const std::string & filename )
```

Creates a file reader with the given filename.

The file reader automatically continues to read split [FileWriter](#) Streams

Parameters

<i>filename</i>	file to read
-----------------	--------------

7.17.2.3 ~FileReader()

```
FileReader::~FileReader ( )
```

7.17.3 Member Function Documentation

7.17.3.1 getConfiguration()

```
std::string FileReader::getConfiguration ( )
```

Fetches the overall configuration status of the Time Tagger object, which was serialized in the current file.

Returns

a JSON serialized string with all configuration and status flags.

7.17.3.2 getData()

```
TimeTagStreamBuffer FileReader::getData (
    uint64_t n_events )
```

Fetches and delete the next tags from the internal buffer.

Every tag is returned exactly once. If less than `n_events` are returned, the reader is at the end-of-files.

Parameters

<code>n_events</code>	maximum amount of elements to fetch
-----------------------	-------------------------------------

Returns

a [TimeTagStreamBuffer](#) with up to `n_events` events

7.17.3.3 getDataRaw()

```
bool FileReader::getDataRaw (
    std::vector< Tag > & tag_buffer )
```

Low level file reading.

This function will return the next non-empty buffer in a raw format.

Parameters

<code>tag_buffer</code>	a buffer, which will be filled with the new events
-------------------------	--

Returns

true if fetching the data was successfully

7.17.3.4 getLastMarker()

```
std::string FileReader::getLastMarker ( )
```

return the last processed marker from the file.

Returns

the last marker from the file

7.17.3.5 hasData()

```
bool FileReader::hasData ( )
```

Checks if there are still events in the [FileReader](#).

Returns

false if no more events can be read from this [FileReader](#)

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

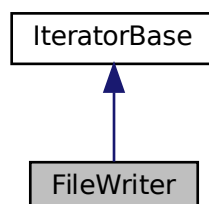
- [Iterators.h](#)

7.18 FileWriter Class Reference

compresses and stores all time tags to a file

```
#include <Iterators.h>
```

Inheritance diagram for FileWriter:



Public Member Functions

- [FileWriter](#) ([TimeTaggerBase](#) *[tagger](#), const std::string &filename, std::vector< [channel_t](#) > channels)
constructor of a [FileWriter](#)
- [~FileWriter](#) ()
- void [split](#) (const std::string &new_filename="")
Close the current file and create a new one.
- void [setMaxFileSize](#) (uint64_t max_file_size)
Set the maximum file size on disk when the automatic split shall happen.
- uint64_t [getMaxFileSize](#) ()
fetches the maximum file size. Please see setMaxFileSize for more details.
- uint64_t [getTotalEvents](#) ()
queries the total amount of events stored in all files
- uint64_t [getTotalSize](#) ()
queries the total amount of bytes stored in all files
- void [setMarker](#) (const std::string &marker)
writes a marker in the file. While parsing the file, the last marker can be extracted again.

Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.
- void [on_start](#) () override
callback when the measurement class is started
- void [on_stop](#) () override
callback when the measurement class is stopped

Additional Inherited Members

7.18.1 Detailed Description

compresses and stores all time tags to a file

7.18.2 Constructor & Destructor Documentation

7.18.2.1 [FileWriter](#)()

```
FileWriter::FileWriter (
    TimeTaggerBase * tagger,
    const std::string & filename,
    std::vector< channel\_t > channels )
```

constructor of a [FileWriter](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>filename</i>	name of the file to store to
<i>channels</i>	channels which are stored to the file

7.18.2.2 ~FileWriter()

```
FileWriter::~FileWriter ( )
```

7.18.3 Member Function Documentation**7.18.3.1 clear_impl()**

```
void FileWriter::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.18.3.2 getMaxFileSize()

```
uint64_t FileWriter::getMaxFileSize ( )
```

fetches the maximum file size. Please see [setMaxFileSize](#) for more details.

Returns

the maximum file size in bytes

7.18.3.3 getTotalEvents()

```
uint64_t FileWriter::getTotalEvents ( )
```

queries the total amount of events stored in all files

Returns

the total amount of events stored

7.18.3.4 getTotalSize()

```
uint64_t FileWriter::getTotalSize ( )
```

queries the total amount of bytes stored in all files

Returns

the total amount of bytes stored

7.18.3.5 next_impl()

```
bool FileWriter::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.18.3.6 on_start()

```
void FileWriter::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.18.3.7 on_stop()

```
void FileWriter::on_stop ( ) [override], [protected], [virtual]
```

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.18.3.8 setMarker()

```
void FileWriter::setMarker (
    const std::string & marker )
```

writes a marker in the file. While parsing the file, the last marker can be extracted again.

Parameters

<i>marker</i>	the marker to write into the file
---------------	-----------------------------------

7.18.3.9 setMaxFileSize()

```
void FileWriter::setMaxFileSize (
    uint64_t max_file_size )
```

Set the maximum file size on disk when the automatic split shall happen.

Note

This is a rough limit, the actual file might be larger by one block.

Parameters

<i>max_file_size</i>	new maximum file size in bytes
----------------------	--------------------------------

7.18.3.10 split()

```
void FileWriter::split (
    const std::string & new_filename = "" )
```

Close the current file and create a new one.

Parameters

<code>new_filename</code>	filename of the new file. If empty, the old one will be used.
---------------------------	---

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

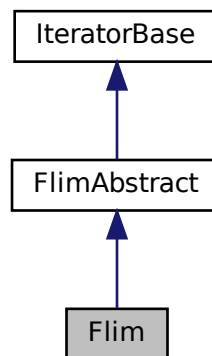
- [Iterators.h](#)

7.19 Flim Class Reference

Fluorescence lifetime imaging.

```
#include <Iterators.h>
```

Inheritance diagram for Flim:



Public Member Functions

- [Flim](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) start_channel, [channel_t](#) click_channel, [channel_t](#) pixel_begin_channel, [uint32_t](#) n_pixels, [uint32_t](#) n_bins, [timestamp_t](#) binwidth, [channel_t](#) pixel_end_channel=CHANNEL_UNUSED, [channel_t](#) frame_begin_channel=CHANNEL_UNUSED, [uint32_t](#) finish_after_outputframe=0, [uint32_t](#) n_frame_average=1, [bool](#) pre_initialize=true)
construct a [Flim](#) measurement with a variety of high-level functionality
- [~Flim](#) ()
- void [initialize](#) ()
initializes and starts measuring this [Flim](#) measurement
- void [getReadyFrame](#) (std::function< [uint32_t](#) *([size_t](#), [size_t](#))> array_out, [int32_t](#) index=-1)
obtain for each pixel the histogram for the given frame index
- void [getReadyFrameIntensity](#) (std::function< [float](#) *([size_t](#))> array_out, [int32_t](#) index=-1)
obtain an array of the pixel intensity of the given frame index
- void [getCurrentFrame](#) (std::function< [uint32_t](#) *([size_t](#), [size_t](#))> array_out)

- obtain for each pixel the histogram for the frame currently active*

 - void [getCurrentFrameIntensity](#) (std::function< float *(size_t)> array_out)
- obtain the array of the pixel intensities of the frame currently active*

 - void [getSummedFrames](#) (std::function< uint32_t *(size_t, size_t)> array_out, bool only_ready_frames=true, bool clear_summed=false)
- obtain for each pixel the histogram from all frames acquired so far*

 - void [getSummedFramesIntensity](#) (std::function< float *(size_t)> array_out, bool only_ready_frames=true, bool clear_summed=false)
- obtain the array of the pixel intensities from all frames acquired so far*

 - [FlimFrameInfo](#) [getReadyFrameEx](#) (int32_t index=-1)
- obtain a frame information object, for the given frame index*

 - [FlimFrameInfo](#) [getCurrentFrameEx](#) ()
- obtain a frame information object, for the currently active frame*

 - [FlimFrameInfo](#) [getSummedFramesEx](#) (bool only_ready_frames=true, bool clear_summed=false)
- obtain a frame information object, that represents the sum of all frames acquired so far.*

 - uint32_t [getFramesAcquired](#) () const
- total number of frames completed so far*

 - void [getIndex](#) (std::function< long long *(size_t)> array_out)
- a vector of size n_bins containing the time bins in ps*

Protected Member Functions

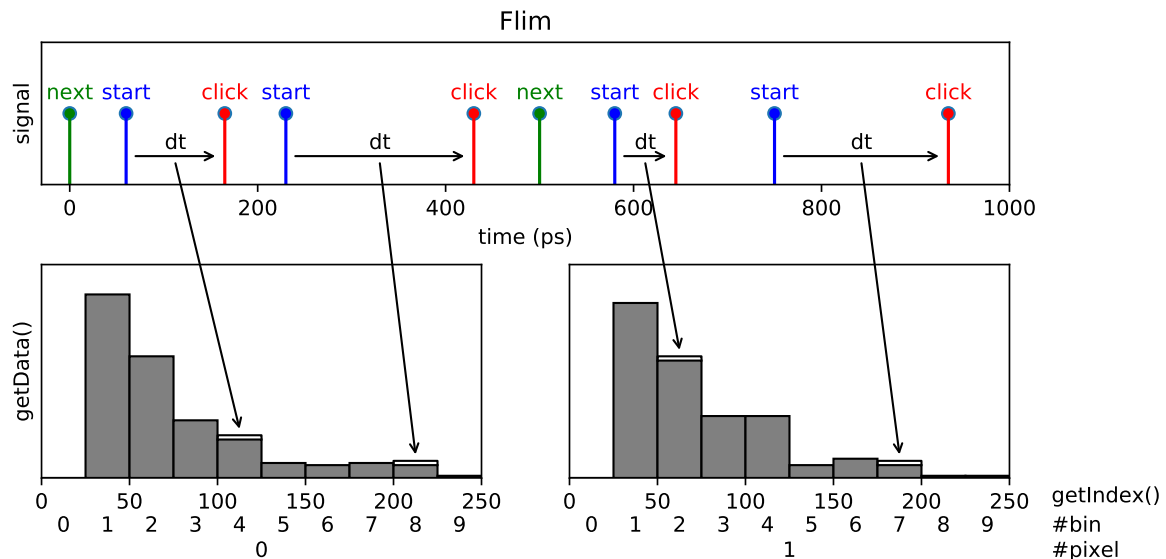
- void [on_frame_end](#) () override
- void [clear_impl](#) () override
- clear [Iterator](#) state.*
- uint32_t [get_ready_index](#) (int32_t index)
- virtual void [frameReady](#) (uint32_t frame_number, std::vector< uint32_t > &data, std::vector< [timestamp_t](#) > &pixel_begin_times, std::vector< [timestamp_t](#) > &pixel_end_times, [timestamp_t](#) frame_begin_time, [timestamp_t](#) frame_end_time)

Protected Attributes

- std::vector< std::vector< uint32_t > > [back_frames](#)
- std::vector< std::vector< [timestamp_t](#) > > [frame_begins](#)
- std::vector< std::vector< [timestamp_t](#) > > [frame_ends](#)
- std::vector< uint32_t > [pixels_completed](#)
- std::vector< uint32_t > [summed_frames](#)
- std::vector< [timestamp_t](#) > [accum_diffs](#)
- uint32_t [captured_frames](#)
- uint32_t [total_frames](#)
- int32_t [last_frame](#)
- std::mutex [swap_chain_lock](#)

7.19.1 Detailed Description

Fluorescence lifetime imaging.



Successively acquires n histograms (one for each pixel in the image), where each histogram is determined by the number of bins and the binwidth. Clicks that fall outside the histogram range are ignored.

Fluorescence-lifetime imaging microscopy or **Flim** is an imaging technique for producing an image based on the differences in the exponential decay rate of the fluorescence from a fluorescent sample.

Fluorescence lifetimes can be determined in the time domain by using a pulsed source. When a population of fluorophores is excited by an ultrashort or delta pulse of light, the time-resolved fluorescence will decay exponentially.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Flim()

```
Flim::Flim (
    TimeTaggerBase * tagger,
    channel_t start_channel,
    channel_t click_channel,
    channel_t pixel_begin_channel,
    uint32_t n_pixels,
    uint32_t n_bins,
    timestamp_t binwidth,
    channel_t pixel_end_channel = CHANNEL_UNUSED,
    channel_t frame_begin_channel = CHANNEL_UNUSED,
    uint32_t finish_after_outputframe = 0,
    uint32_t n_frame_average = 1,
    bool pre_initialize = true )
```

construct a **Flim** measurement with a variety of high-level functionality

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>start_channel</i>	channel on which start clicks are received for the time differences histogramming
<i>click_channel</i>	channel on which clicks are received for the time differences histogramming
<i>pixel_begin_channel</i>	start of a pixel (histogram)
<i>n_pixels</i>	number of pixels (histograms) of one frame
<i>n_bins</i>	number of histogram bins for each pixel
<i>binwidth</i>	bin size in picoseconds
<i>pixel_end_channel</i>	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
<i>frame_begin_channel</i>	(optional) start the frame, or reset the pixel index
<i>finish_after_outputframe</i>	(optional) sets the number of frames stored within the measurement class. After reaching the number, the measurement will stop. If the number is 0 (default value), one frame is stored and the measurement runs continuously.
<i>n_frame_average</i>	(optional) average multiple input frames into one output frame, default: 1
<i>pre_initialize</i>	(optional) initializes the measurement on constructing.

7.19.2.2 ~Flim()

```
Flim::~~Flim ( )
```

7.19.3 Member Function Documentation

7.19.3.1 clear_impl()

```
void Flim::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [FlimAbstract](#).

7.19.3.2 frameReady()

```
virtual void Flim::frameReady (
    uint32_t frame_number,
    std::vector< uint32_t > & data,
    std::vector< timestamp_t > & pixel_begin_times,
    std::vector< timestamp_t > & pixel_end_times,
    timestamp_t frame_begin_time,
    timestamp_t frame_end_time ) [protected], [virtual]
```

7.19.3.3 get_ready_index()

```
uint32_t Flim::get_ready_index (
    int32_t index ) [protected]
```

7.19.3.4 getCurrentFrame()

```
void Flim::getCurrentFrame (
    std::function< uint32_t *(size_t, size_t)> array_out )
```

obtain for each pixel the histogram for the frame currently active

This function returns the histograms for all pixels of the currently active frame

7.19.3.5 getCurrentFrameEx()

```
FlimFrameInfo Flim::getCurrentFrameEx ( )
```

obtain a frame information object, for the currently active frame

This function returns the frame information object for the currently active frame

7.19.3.6 getCurrentFrameIntensity()

```
void Flim::getCurrentFrameIntensity (
    std::function< float *(size_t)> array_out )
```

obtain the array of the pixel intensities of the frame currently active

This function returns the intensities of all pixels of the currently active frame

The pixel intensity is defined by the number of counts acquired within the pixel divided by the respective integration time.

7.19.3.7 getFramesAcquired()

```
uint32_t Flim::getFramesAcquired ( ) const [inline]
```

total number of frames completed so far

This function returns the amount of frames that have been completed so far, since the creation / last clear of the object.

7.19.3.8 getIndex()

```
void Flim::getIndex (
    std::function< long long *(size_t)> array_out )
```

a vector of size n_bins containing the time bins in ps

This function returns a vector of size n_bins containing the time bins in ps.

7.19.3.9 getReadyFrame()

```
void Flim::getReadyFrame (
    std::function< uint32_t *(size_t, size_t)> array_out,
    int32_t index = -1 )
```

obtain for each pixel the histogram for the given frame index

This function returns the histograms for all pixels according to the frame index given. If the index is -1, it will return the last frame, which has been completed. When finish_after_outputframe is 0, the index value must be -1. If index \geq finish_after_outputframe, it will throw an error.

Parameters

<i>array_out</i>	callback for the array output allocation
<i>index</i>	index of the frame to be obtained. if -1, the last frame which has been completed is returned

7.19.3.10 getReadyFrameEx()

```
FlimFrameInfo Flim::getReadyFrameEx (
    int32_t index = -1 )
```

obtain a frame information object, for the given frame index

This function returns a frame information object according to the index given. If the index is -1, it will return the last completed frame. When finish_after_outputframe is 0, index must be -1. If index \geq finish_after_outputframe, it will throw an error.

Parameters

<i>index</i>	index of the frame to be obtained. if -1, last completed frame will be returned
--------------	---

7.19.3.11 getReadyFrameIntensity()

```
void Flim::getReadyFrameIntensity (
    std::function< float *(size_t)> array_out,
    int32_t index = -1 )
```

obtain an array of the pixel intensity of the given frame index

This function returns the intensities according to the frame index given. If the index is -1, it will return the intensity of the last frame, which has been completed. When finish_after_outputframe is 0, the index value must be -1. If index \geq finish_after_outputframe, it will throw an error.

The pixel intensity is defined by the number of counts acquired within the pixel divided by the respective integration time.

Parameters

<i>array_out</i>	callback for the array output allocation
<i>index</i>	index of the frame to be obtained. if -1, the last frame which has been completed is returned

7.19.3.12 getSummedFrames()

```
void Flim::getSummedFrames (
    std::function< uint32_t *(size_t, size_t)> array_out,
    bool only_ready_frames = true,
    bool clear_summed = false )
```

obtain for each pixel the histogram from all frames acquired so far

This function returns the histograms for all pixels. The counts within the histograms are integrated since the start or the last clear of the measurement.

Parameters

<i>array_out</i>	callback for the array output allocation
<i>only_ready_frames</i>	if true, only the finished frames are added. On false, the currently active frame is aggregated.
<i>clear_summed</i>	if true, the summed frames memory will be cleared.

7.19.3.13 getSummedFramesEx()

```
FlimFrameInfo Flim::getSummedFramesEx (
    bool only_ready_frames = true,
    bool clear_summed = false )
```

obtain a frame information object, that represents the sum of all frames acquired so far.

This function returns the frame information object that represents the sum of all acquired frames.

Parameters

<i>only_ready_frames</i>	if true only the finished frames are added. On false, the currently active is aggregated.
<i>clear_summed</i>	if true, the summed frames memory will be reset and all frames stored prior will be unaccounted in the future.

7.19.3.14 getSummedFramesIntensity()

```
void Flim::getSummedFramesIntensity (
    std::function< float *(size_t)> array_out,
```

```
bool only_ready_frames = true,
bool clear_summed = false )
```

obtain the array of the pixel intensities from all frames acquired so far

The pixel intensity is the number of counts within the pixel divided by the integration time.

This function returns the intensities of all pixels summed over all acquired frames.

Parameters

<i>array_out</i>	callback for the array output allocation
<i>only_ready_frames</i>	if true only the finished frames are added. On false, the currently active frame is aggregated.
<i>clear_summed</i>	if true, the summed frames memory will be cleared.

7.19.3.15 initialize()

```
void Flim::initialize ( )
```

initializes and starts measuring this [Flim](#) measurement

This function initializes the [Flim](#) measurement and starts executing it. It does nothing if preinitialized in the constructor is set to true.

7.19.3.16 on_frame_end()

```
void Flim::on_frame_end ( ) [override], [protected], [virtual]
```

Implements [FlimAbstract](#).

7.19.4 Member Data Documentation

7.19.4.1 accum_diffs

```
std::vector<timestamp_t> Flim::accum_diffs [protected]
```

7.19.4.2 back_frames

```
std::vector<std::vector<uint32_t> > Flim::back_frames [protected]
```

7.19.4.3 captured_frames

```
uint32_t Flim::captured_frames [protected]
```

7.19.4.4 frame_begins

```
std::vector<std::vector<timestamp_t> > Flim::frame_begins [protected]
```

7.19.4.5 frame_ends

```
std::vector<std::vector<timestamp_t> > Flim::frame_ends [protected]
```

7.19.4.6 last_frame

```
int32_t Flim::last_frame [protected]
```

7.19.4.7 pixels_completed

```
std::vector<uint32_t> Flim::pixels_completed [protected]
```

7.19.4.8 summed_frames

```
std::vector<uint32_t> Flim::summed_frames [protected]
```

7.19.4.9 swap_chain_lock

```
std::mutex Flim::swap_chain_lock [protected]
```

7.19.4.10 total_frames

```
uint32_t Flim::total_frames [protected]
```

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

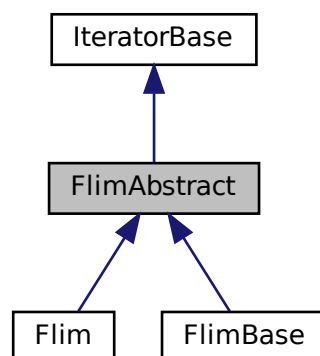
- [Iterators.h](#)

7.20 FlimAbstract Class Reference

Interface for FLIM measurements, [Flim](#) and [FlimBase](#) classes inherit from it.

```
#include <Iterators.h>
```

Inheritance diagram for FlimAbstract:



Public Member Functions

- [FlimAbstract](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) start_channel, [channel_t](#) click_channel, [channel_t](#) pixel_begin_channel, [uint32_t](#) n_pixels, [uint32_t](#) n_bins, [timestamp_t](#) binwidth, [channel_t](#) pixel_end_channel=[CHANNEL_UNUSED](#), [channel_t](#) frame_begin_channel=[CHANNEL_UNUSED](#), [uint32_t](#) finish_after_outputframe=0, [uint32_t](#) n_frame_average=1, [bool](#) pre_initialize=true)
construct a [FlimAbstract](#) object, [Flim](#) and [FlimBase](#) classes inherit from it
- [~FlimAbstract](#) ()
- [bool](#) [isAcquiring](#) () const
tells if the data acquisition has finished reaching finish_after_outputframe

Protected Member Functions

- template<FastBinning::Mode bin_mode>
void [process_tags](#) (const std::vector< [Tag](#) > &incoming_tags)
- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.
- void [on_start](#) () override
callback when the measurement class is started
- virtual void [on_frame_end](#) ()=0

Protected Attributes

- const [channel_t](#) [start_channel](#)
- const [channel_t](#) [click_channel](#)
- const [channel_t](#) [pixel_begin_channel](#)
- const uint32_t [n_pixels](#)
- const uint32_t [n_bins](#)
- const [timestamp_t](#) [binwidth](#)
- const [channel_t](#) [pixel_end_channel](#)
- const [channel_t](#) [frame_begin_channel](#)
- const uint32_t [finish_after_outputframe](#)
- const uint32_t [n_frame_average](#)
- const [timestamp_t](#) [time_window](#)
- [timestamp_t](#) [current_frame_begin](#)
- [timestamp_t](#) [current_frame_end](#)
- bool [acquiring](#) {}
- bool [frame_acquisition](#) {}
- bool [pixel_acquisition](#) {}
- uint32_t [pixels_processed](#) {}
- uint32_t [frames_completed](#) {}
- uint32_t [ticks](#) {}
- size_t [data_base](#) {}
- std::vector< uint32_t > [frame](#)
- std::vector< [timestamp_t](#) > [pixel_begins](#)
- std::vector< [timestamp_t](#) > [pixel_ends](#)
- std::deque< [timestamp_t](#) > [previous_starts](#)
- [FastBinning](#) [binner](#)
- std::recursive_mutex [acquisition_lock](#)
- bool [initialized](#)

7.20.1 Detailed Description

Interface for FLIM measurements, [Flim](#) and [FlimBase](#) classes inherit from it.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 FlimAbstract()

```

FlimAbstract::FlimAbstract (
    TimeTaggerBase * tagger,
    channel_t start_channel,
    channel_t click_channel,
    channel_t pixel_begin_channel,
    uint32_t n_pixels,
    uint32_t n_bins,
    timestamp_t binwidth,
    channel_t pixel_end_channel = CHANNEL_UNUSED,
    channel_t frame_begin_channel = CHANNEL_UNUSED,
    uint32_t finish_after_outputframe = 0,
    uint32_t n_frame_average = 1,
    bool pre_initialize = true )

```

construct a [FlimAbstract](#) object, [Flim](#) and [FlimBase](#) classes inherit from it

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>start_channel</i>	channel on which start clicks are received for the time differences histogramming
<i>click_channel</i>	channel on which clicks are received for the time differences histogramming
<i>pixel_begin_channel</i>	start of a pixel (histogram)
<i>n_pixels</i>	number of pixels (histograms) of one frame
<i>n_bins</i>	number of histogram bins for each pixel
<i>binwidth</i>	bin size in picoseconds
<i>pixel_end_channel</i>	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
<i>frame_begin_channel</i>	(optional) start the frame, or reset the pixel index
<i>finish_after_outputframe</i>	(optional) sets the number of frames stored within the measurement class. After reaching the number, the measurement will stop. If the number is 0 (default value), one frame is stored and the measurement runs continuously.
<i>n_frame_average</i>	(optional) average multiple input frames into one output frame, default: 1
<i>pre_initialize</i>	(optional) initializes the measurement on constructing.

7.20.2.2 ~FlimAbstract()

```

FlimAbstract::~~FlimAbstract ( )

```

7.20.3 Member Function Documentation

7.20.3.1 clear_impl()

```
void FlimAbstract::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

Reimplemented in [Flim](#).

7.20.3.2 isAcquiring()

```
bool FlimAbstract::isAcquiring ( ) const [inline]
```

tells if the data acquisition has finished reaching `finish_after_outputframe`

This function returns a boolean which tells the user if the class is still acquiring data. It can only reach the false state for `finish_after_outputframe > 0`.

Note

This can differ from `isRunning`. The return value of `isRunning` state depends only on `start/startFor/stop`.

7.20.3.3 next_impl()

```
bool FlimAbstract::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	<code>begin_time</code> of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.20.3.4 on_frame_end()

```
virtual void FlimAbstract::on_frame_end ( ) [protected], [pure virtual]
```

Implemented in [Flim](#), and [FlimBase](#).

7.20.3.5 on_start()

```
void FlimAbstract::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.20.3.6 process_tags()

```
template<FastBinning::Mode bin_mode>
void FlimAbstract::process_tags (
    const std::vector< Tag > & incoming_tags ) [protected]
```

7.20.4 Member Data Documentation**7.20.4.1 acquiring**

```
bool FlimAbstract::acquiring {} [protected]
```

7.20.4.2 acquisition_lock

```
std::recursive_mutex FlimAbstract::acquisition_lock [protected]
```


7.20.4.3 binner

```
FastBinning FlimAbstract::binner [protected]
```

7.20.4.4 binwidth

```
const timestamp_t FlimAbstract::binwidth [protected]
```

7.20.4.5 click_channel

```
const channel_t FlimAbstract::click_channel [protected]
```

7.20.4.6 current_frame_begin

```
timestamp_t FlimAbstract::current_frame_begin [protected]
```

7.20.4.7 current_frame_end

```
timestamp_t FlimAbstract::current_frame_end [protected]
```

7.20.4.8 data_base

```
size_t FlimAbstract::data_base {} [protected]
```

7.20.4.9 finish_after_outputframe

```
const uint32_t FlimAbstract::finish_after_outputframe [protected]
```

7.20.4.10 frame

```
std::vector<uint32_t> FlimAbstract::frame [protected]
```

7.20.4.11 frame_acquisition

```
bool FlimAbstract::frame_acquisition {} [protected]
```

7.20.4.12 frame_begin_channel

```
const channel_t FlimAbstract::frame_begin_channel [protected]
```

7.20.4.13 frames_completed

```
uint32_t FlimAbstract::frames_completed {} [protected]
```

7.20.4.14 initialized

```
bool FlimAbstract::initialized [protected]
```

7.20.4.15 n_bins

```
const uint32_t FlimAbstract::n_bins [protected]
```

7.20.4.16 n_frame_average

```
const uint32_t FlimAbstract::n_frame_average [protected]
```

7.20.4.17 n_pixels

```
const uint32_t FlimAbstract::n_pixels [protected]
```

7.20.4.18 pixel_acquisition

```
bool FlimAbstract::pixel_acquisition {} [protected]
```

7.20.4.19 pixel_begin_channel

```
const channel_t FlimAbstract::pixel_begin_channel [protected]
```

7.20.4.20 pixel_begins

```
std::vector<timestamp_t> FlimAbstract::pixel_begins [protected]
```

7.20.4.21 pixel_end_channel

```
const channel_t FlimAbstract::pixel_end_channel [protected]
```

7.20.4.22 pixel_ends

```
std::vector<timestamp_t> FlimAbstract::pixel_ends [protected]
```

7.20.4.23 pixels_processed

```
uint32_t FlimAbstract::pixels_processed {} [protected]
```

7.20.4.24 previous_starts

```
std::deque<timestamp_t> FlimAbstract::previous_starts [protected]
```

7.20.4.25 start_channel

```
const channel_t FlimAbstract::start_channel [protected]
```

7.20.4.26 ticks

```
uint32_t FlimAbstract::ticks {} [protected]
```

7.20.4.27 time_window

```
const timestamp_t FlimAbstract::time_window [protected]
```

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

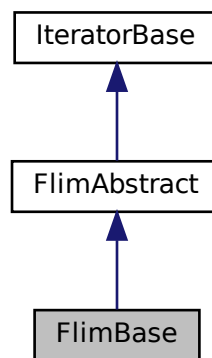
- [Iterators.h](#)

7.21 FlimBase Class Reference

basic measurement, containing a minimal set of features for efficiency purposes

```
#include <Iterators.h>
```

Inheritance diagram for FlimBase:



Public Member Functions

- [FlimBase](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) start_channel, [channel_t](#) click_channel, [channel_t](#) pixel_begin_channel, [uint32_t](#) n_pixels, [uint32_t](#) n_bins, [timestamp_t](#) binwidth, [channel_t](#) pixel_end_channel=CHANNEL_UNUSED, [channel_t](#) frame_begin_channel=CHANNEL_UNUSED, [uint32_t](#) finish_after_outputframe=0, [uint32_t](#) n_frame_average=1, [bool](#) pre_initialize=true)
construct a basic [Flim](#) measurement, containing a minimum featureset for efficiency purposes
- [~FlimBase](#) ()
- void [initialize](#) ()
initializes and starts measuring this [Flim](#) measurement

Protected Member Functions

- void [on_frame_end](#) () override
- virtual void [frameReady](#) ([uint32_t](#) frame_number, [std::vector](#)< [uint32_t](#) > &data, [std::vector](#)< [timestamp_t](#) > &pixel_begin_times, [std::vector](#)< [timestamp_t](#) > &pixel_end_times, [timestamp_t](#) frame_begin_time, [timestamp_t](#) frame_end_time)

Protected Attributes

- uint32_t [total_frames](#)

7.21.1 Detailed Description

basic measurement, containing a minimal set of features for efficiency purposes

The [FlimBase](#) provides only the most essential functionality for FLIM tasks. The benefit from the reduced functionality is that it is very memory and CPU efficient. The class provides the [frameReady\(\)](#) callback, which must be used to analyze the data.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 FlimBase()

```
FlimBase::FlimBase (
    TimeTaggerBase * tagger,
    channel_t start_channel,
    channel_t click_channel,
    channel_t pixel_begin_channel,
    uint32_t n_pixels,
    uint32_t n_bins,
    timestamp_t binwidth,
    channel_t pixel_end_channel = CHANNEL_UNUSED,
    channel_t frame_begin_channel = CHANNEL_UNUSED,
    uint32_t finish_after_outputframe = 0,
    uint32_t n_frame_average = 1,
    bool pre_initialize = true )
```

construct a basic [Flim](#) measurement, containing a minimum featureset for efficiency purposes

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>start_channel</i>	channel on which start clicks are received for the time differences histogramming
<i>click_channel</i>	channel on which clicks are received for the time differences histogramming
<i>pixel_begin_channel</i>	start of a pixel (histogram)
<i>n_pixels</i>	number of pixels (histograms) of one frame
<i>n_bins</i>	number of histogram bins for each pixel
<i>binwidth</i>	bin size in picoseconds
<i>pixel_end_channel</i>	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
<i>frame_begin_channel</i>	(optional) start the frame, or reset the pixel index
<i>finish_after_outputframe</i>	(optional) sets the number of frames stored within the measurement class. After reaching the number, the measurement will stop. If the number is 0 (default value), one frame is stored and the measurement runs continuously.
<i>n_frame_average</i>	(optional) average multiple input frames into one output frame, default: 1
<i>pre_initialize</i>	(optional) initializes the measurement on constructing.

7.21.2.2 ~FlimBase()

```
FlimBase::~~FlimBase ( )
```

7.21.3 Member Function Documentation

7.21.3.1 frameReady()

```
virtual void FlimBase::frameReady (
    uint32_t frame_number,
    std::vector< uint32_t > & data,
    std::vector< timestamp_t > & pixel_begin_times,
    std::vector< timestamp_t > & pixel_end_times,
    timestamp_t frame_begin_time,
    timestamp_t frame_end_time ) [protected], [virtual]
```

7.21.3.2 initialize()

```
void FlimBase::initialize ( )
```

initializes and starts measuring this [Flim](#) measurement

This function initializes the [Flim](#) measurement and starts executing it. It does nothing if preinitialized in the constructor is set to true.

7.21.3.3 on_frame_end()

```
void FlimBase::on_frame_end ( ) [override], [protected], [virtual]
```

Implements [FlimAbstract](#).

7.21.4 Member Data Documentation

7.21.4.1 total_frames

```
uint32_t FlimBase::total_frames [protected]
```

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

- [Iterators.h](#)

7.22 FlimFrameInfo Class Reference

object for storing the state of [Flim::getCurrentFrameEx](#)

```
#include <Iterators.h>
```

Public Member Functions

- [~FlimFrameInfo](#) ()
- [int32_t getFrameNumber](#) () const
index of this frame
- [bool isValid](#) () const
tells if this frame is valid
- [uint32_t getPixelPosition](#) () const
number of pixels acquired on this frame
- [void getHistograms](#) (std::function< [uint32_t](#) *([size_t](#), [size_t](#))> array_out)
- [void getIntensities](#) (std::function< [float](#) *([size_t](#))> array_out)
- [void getSummedCounts](#) (std::function< [uint64_t](#) *([size_t](#))> array_out)
- [void getPixelBegins](#) (std::function< [long long](#) *([size_t](#))> array_out)
- [void getPixelEnds](#) (std::function< [long long](#) *([size_t](#))> array_out)

Public Attributes

- [uint32_t pixels](#)
- [uint32_t bins](#)
- [int32_t frame_number](#)
- [uint32_t pixel_position](#)
- [bool valid](#)

7.22.1 Detailed Description

object for storing the state of [Flim::getCurrentFrameEx](#)

7.22.2 Constructor & Destructor Documentation

7.22.2.1 ~FlimFrameInfo()

```
FlimFrameInfo::~FlimFrameInfo ( )
```

7.22.3 Member Function Documentation

7.22.3.1 getFrameNumber()

```
int32_t FlimFrameInfo::getFrameNumber ( ) const [inline]
```

index of this frame

This function returns the frame number, starting from 0 for the very first frame acquired. If the index is -1, it is an invalid frame which is returned on error

deprecated, use frame_number instead..

7.22.3.2 getHistograms()

```
void FlimFrameInfo::getHistograms (
    std::function< uint32_t *(size_t, size_t)> array_out )
```

7.22.3.3 getIntensities()

```
void FlimFrameInfo::getIntensities (
    std::function< float *(size_t)> array_out )
```

7.22.3.4 getPixelBegins()

```
void FlimFrameInfo::getPixelBegins (
    std::function< long long *(size_t)> array_out )
```

7.22.3.5 getPixelEnds()

```
void FlimFrameInfo::getPixelEnds (
    std::function< long long *(size_t)> array_out )
```

7.22.3.6 getPixelPosition()

```
uint32_t FlimFrameInfo::getPixelPosition ( ) const [inline]
```

number of pixels acquired on this frame

This function returns a value which tells how many pixels were processed for this frame.

7.22.3.7 getSummedCounts()

```
void FlimFrameInfo::getSummedCounts (
    std::function< uint64_t *(size_t)> array_out )
```

7.22.3.8 isValid()

```
bool FlimFrameInfo::isValid ( ) const [inline]
```

tells if this frame is valid

This function returns a boolean which tells if this frame is valid or not. Invalid frames are possible on errors, such as asking for the last completed frame when no frame has been completed so far.

deprecated, use isValid instead.

7.22.4 Member Data Documentation

7.22.4.1 bins

```
uint32_t FlimFrameInfo::bins
```

7.22.4.2 frame_number

```
int32_t FlimFrameInfo::frame_number
```

7.22.4.3 pixel_position

```
uint32_t FlimFrameInfo::pixel_position
```

7.22.4.4 pixels

```
uint32_t FlimFrameInfo::pixels
```

7.22.4.5 valid

```
bool FlimFrameInfo::valid
```

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

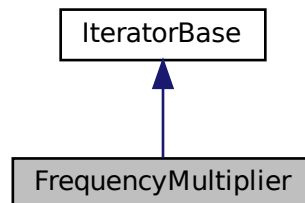
- [Iterators.h](#)

7.23 FrequencyMultiplier Class Reference

The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.

```
#include <Iterators.h>
```

Inheritance diagram for FrequencyMultiplier:



Public Member Functions

- [FrequencyMultiplier](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) input_channel, [int32_t](#) multiplier)
constructor of a [FrequencyMultiplier](#)
- [~FrequencyMultiplier](#) ()
- [channel_t](#) getChannel ()
- [int32_t](#) getMultiplier ()

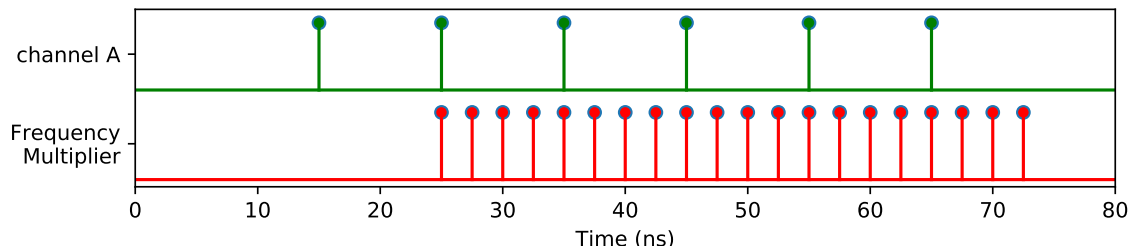
Protected Member Functions

- [bool](#) [next_impl](#) ([std::vector](#)< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state

Additional Inherited Members

7.23.1 Detailed Description

The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.



The [FrequencyMultiplier](#) inserts copies the original input events from the `input_channel` and adds additional events to match the upscaling factor. The algorithm used assumes a constant frequency and calculates out of the last two incoming events linearly the intermediate timestamps to match the upscaled frequency given by the multiplier parameter.

The [FrequencyMultiplier](#) can be used to restore the actual frequency applied to an `input_channel` which was reduced via the `EventDivider` to lower the effective data rate. For example a 80 MHz laser sync signal can be scaled down via `setEventDivider(..., 80)` to 1 MHz (hardware side) and an 80 MHz signal can be restored via [FrequencyMultiplier](#)(..., 80) on the software side with some loss in precision. The [FrequencyMultiplier](#) is an alternative way to reduce the data rate in comparison to the `EventFilter`, which has a higher precision but can be more difficult to use.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 FrequencyMultiplier()

```
FrequencyMultiplier::FrequencyMultiplier (
    TimeTaggerBase * tagger,
    channel_t input_channel,
    int32_t multiplier )
```

constructor of a [FrequencyMultiplier](#)

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>input_channel</i>	channel on which the upscaling of the frequency is based on
<i>multiplier</i>	frequency upscaling factor

7.23.2.2 ~FrequencyMultiplier()

```
FrequencyMultiplier::~~FrequencyMultiplier ( )
```

7.23.3 Member Function Documentation

7.23.3.1 getChannel()

```
channel_t FrequencyMultiplier::getChannel ( )
```

7.23.3.2 getMultiplier()

```
int32_t FrequencyMultiplier::getMultiplier ( )
```

7.23.3.3 next_impl()

```
bool FrequencyMultiplier::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

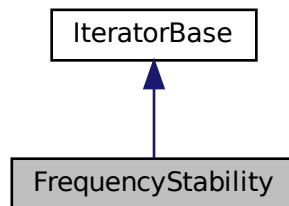
- [Iterators.h](#)

7.24 FrequencyStability Class Reference

Allan deviation (and related metrics) calculator.

```
#include <Iterators.h>
```

Inheritance diagram for FrequencyStability:



Public Member Functions

- `FrequencyStability (TimeTaggerBase *tagger, channel_t channel, std::vector< uint64_t > steps, timestamp_t average=1000, uint64_t trace_len=1000)`
constructor of a `FrequencyStability` measurement
- `~FrequencyStability ()`
- `FrequencyStabilityData getDataObject ()`
get a return object with all data in a synchronized way

Protected Member Functions

- `bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)` override
update iterator state
- `void clear_impl ()` override
clear `Iterator` state.
- `void on_start ()` override
callback when the measurement class is started

Additional Inherited Members

7.24.1 Detailed Description

Allan deviation (and related metrics) calculator.

It shall analyse the stability of a clock by computing deviations of $\text{phase}[i] - \text{phase}[i + n]$. The list of all n values needs to be declared in the beginning.

Reference: <https://www.nist.gov/publications/handbook-frequency-stability-analysis>

It calculates the STDD, ADEV, MDEV and HDEV on the fly:

- STDD: Standard derivation of each period pair. This is not a stable analysis with frequency drifts and only calculated for reference.
- ADEV: Overlapping Allan deviation, the most common analysis framework. Square mean value of the second derivate $\text{phase}[i] - 2*\text{phase}[i + n] + \text{phase}[i + 2*n]$. In a loglog plot, the slope allows to identify the source of noise:
 - -1: white or flicker phase noise, like discretization or analog noisy delay
 - -0.5: white period noise
 - 0: flicker period noise, like electric noisy oscillator
 - 0.5: integrated white period noise (random walk period)
 - 1: frequency drift, e.g. thermal

As this tool is most likely used to analyse timings, a scaled ADEV is implemented. It adds 1.0 to each slope and normalize the return value to picoseconds for phase noise.

- MDEV: Modified overlapping Allan deviation. It averages the second derivate of ADEV before calculating the MSE. This splits the slope of white and flicker phase noise:
 - -1.5: white phase noise, like discretization
 - -1.0: flicker phase noise, like an electric noisy delay

The scaled approach (+1 on each slope yielding picoseconds as return value) is called TDEV and more commonly used than MDEV.

- HDEV: The overlapping Hadamard deviation uses the third derivate of the phase. This cancels the effect of a constant phase drift.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 FrequencyStability()

```
FrequencyStability::FrequencyStability (
    TimeTaggerBase * tagger,
    channel_t channel,
    std::vector< uint64_t > steps,
    timestamp_t average = 1000,
    uint64_t trace_len = 1000 )
```

constructor of a [FrequencyStability](#) measurement

Parameters

<i>tagger</i>	time tagger object
<i>channel</i>	the clock input channel used for the analysis
<i>steps</i>	a vector or integer tau values for all deviations
<i>average</i>	an averaging down sampler to reduce noise and memory requirements
<i>trace_len</i>	length of the phase and frequency trace capture of the averaged data

Note

This measurements needs 24 times the largest value in steps bytes of main memory

7.24.2.2 ~FrequencyStability()

```
FrequencyStability::~~FrequencyStability ( )
```

7.24.3 Member Function Documentation**7.24.3.1 clear_impl()**

```
void FrequencyStability::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.24.3.2 getDataObject()

```
FrequencyStabilityData FrequencyStability::getDataObject ( )
```

get a return object with all data in a synchronized way

7.24.3.3 next_impl()

```
bool FrequencyStability::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.24.3.4 on_start()

```
void FrequencyStability::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.25 FrequencyStabilityData Class Reference

return data object for FrequencyStability::getData.

```
#include <Iterators.h>
```

Public Member Functions

- [~FrequencyStabilityData](#) ()
- void [getSTDD](#) (std::function< double *(size_t)> array_out)
returns the standard derivation of each period pair
- void [getADEV](#) (std::function< double *(size_t)> array_out)
returns the overlapping Allan deviation
- void [getMDEV](#) (std::function< double *(size_t)> array_out)
returns the modified overlapping Allan deviation
- void [getTDEV](#) (std::function< double *(size_t)> array_out)
returns the overlapping time deviation
- void [getHDEV](#) (std::function< double *(size_t)> array_out)
returns the overlapping Hadamard deviation
- void [getADEVScaled](#) (std::function< double *(size_t)> array_out)

- returns the scaled version of the overlapping Allan deviation*
- void [getHDEVScaled](#) (std::function< double *(size_t)> array_out)
returns the scaled version of the overlapping Hadamard deviation
- void [getTau](#) (std::function< double *(size_t)> array_out)
returns the analysis position of all deviations
- void [getTracePhase](#) (std::function< double *(size_t)> array_out)
returns a trace of the last phase samples in seconds
- void [getTraceFrequency](#) (std::function< double *(size_t)> array_out)
returns a trace of the last normalized frequency error samples in pp1
- void [getTraceIndex](#) (std::function< double *(size_t)> array_out)
returns the timestamps of the traces in seconds

7.25.1 Detailed Description

return data object for FrequencyStability::getData.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 ~FrequencyStabilityData()

```
FrequencyStabilityData::~FrequencyStabilityData ( )
```

7.25.3 Member Function Documentation

7.25.3.1 getADEV()

```
void FrequencyStabilityData::getADEV (
    std::function< double *(size_t)> array_out )
```

returns the overlapping Allan deviation

7.25.3.2 getADEVScaled()

```
void FrequencyStabilityData::getADEVScaled (
    std::function< double *(size_t)> array_out )
```

returns the scaled version of the overlapping Allan deviation

7.25.3.3 getHDEV()

```
void FrequencyStabilityData::getHDEV (
    std::function< double *(size_t)> array_out )
```

returns the overlapping Hadamard deviation

7.25.3.4 getHDEVScaled()

```
void FrequencyStabilityData::getHDEVScaled (
    std::function< double *(size_t)> array_out )
```

returns the scaled version of the overlapping Hadamard deviation

7.25.3.5 getMDEV()

```
void FrequencyStabilityData::getMDEV (
    std::function< double *(size_t)> array_out )
```

returns the modified overlapping Allan deviation

7.25.3.6 getSTDD()

```
void FrequencyStabilityData::getSTDD (
    std::function< double *(size_t)> array_out )
```

returns the standard derivation of each period pair

7.25.3.7 getTau()

```
void FrequencyStabilityData::getTau (
    std::function< double *(size_t)> array_out )
```

returns the analysis position of all deviations

7.25.3.8 getTDEV()

```
void FrequencyStabilityData::getTDEV (
    std::function< double *(size_t)> array_out )
```

returns the overlapping time deviation

This is the scaled version of the modified overlapping Allan deviation.

7.25.3.9 getTraceFrequency()

```
void FrequencyStabilityData::getTraceFrequency (
    std::function< double *(size_t)> array_out )
```

returns a trace of the last normalized frequency error samples in pp1

7.25.3.10 getTraceIndex()

```
void FrequencyStabilityData::getTraceIndex (
    std::function< double *(size_t)> array_out )
```

returns the timestamps of the traces in seconds

7.25.3.11 getTracePhase()

```
void FrequencyStabilityData::getTracePhase (
    std::function< double *(size_t)> array_out )
```

returns a trace of the last phase samples in seconds

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

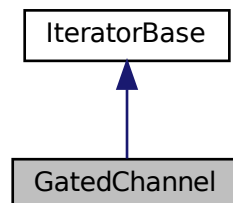
- [Iterators.h](#)

7.26 GatedChannel Class Reference

An input channel is gated by a gate channel.

```
#include <Iterators.h>
```

Inheritance diagram for GatedChannel:



Public Member Functions

- `GatedChannel` (`TimeTaggerBase` *`tagger`, `channel_t` `input_channel`, `channel_t` `gate_start_channel`, `channel_t` `gate_stop_channel`)
constructor of a `GatedChannel`
- `~GatedChannel` ()
- `channel_t` `getChannel` ()
the new virtual channel

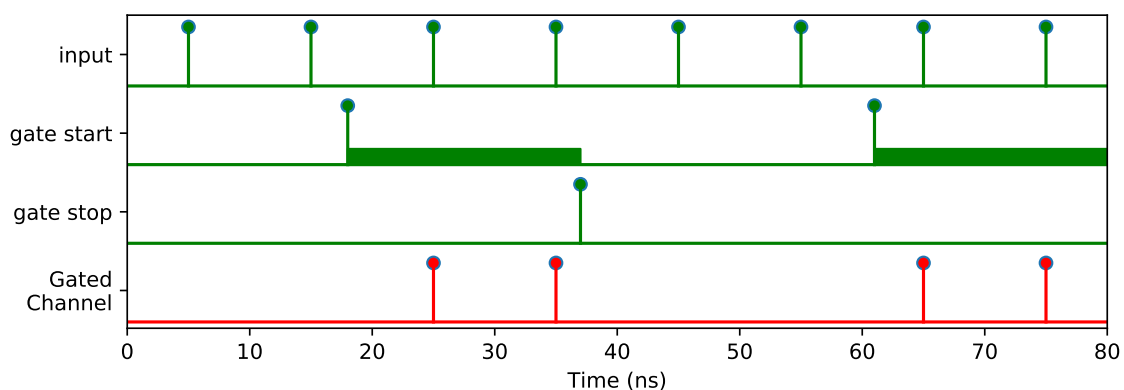
Protected Member Functions

- `bool` `next_impl` (`std::vector`< `Tag` > &`incoming_tags`, `timestamp_t` `begin_time`, `timestamp_t` `end_time`) override
update iterator state

Additional Inherited Members

7.26.1 Detailed Description

An input channel is gated by a gate channel.



Note: The gate is edge sensitive and not level sensitive. That means that the gate will transfer data only when an appropriate level change is detected on the `gate_start_channel`.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 GatedChannel()

```
GatedChannel::GatedChannel (
    TimeTaggerBase * tagger,
    channel_t input_channel,
    channel_t gate_start_channel,
    channel_t gate_stop_channel )
```

constructor of a `GatedChannel`

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>input_channel</i>	channel which is gated
<i>gate_start_channel</i>	channel on which a signal detected will start the transmission of the input_channel through the gate
<i>gate_stop_channel</i>	channel on which a signal detected will stop the transmission of the input_channel through the gate

7.26.2.2 ~GatedChannel()

```
GatedChannel::~GatedChannel ( )
```

7.26.3 Member Function Documentation

7.26.3.1 getChannel()

```
channel_t GatedChannel::getChannel ( )
```

the new virtual channel

This function returns the new allocated virtual channel. It can be used now in any new iterator.

7.26.3.2 next_impl()

```
bool GatedChannel::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

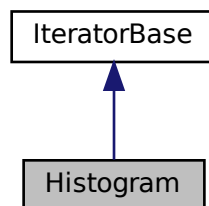
- [Iterators.h](#)

7.27 Histogram Class Reference

Accumulate time differences into a histogram.

```
#include <Iterators.h>
```

Inheritance diagram for Histogram:



Public Member Functions

- [Histogram](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) click_channel, [channel_t](#) start_channel=[CHANNEL_UNUSED](#), [timestamp_t](#) binwidth=1000, [int32_t](#) n_bins=1000)
constructor of a [Histogram](#) measurement
- [~Histogram](#) ()
- void [getData](#) (std::function< [int32_t](#) *([size_t](#))> array_out)
- void [getIndex](#) (std::function< long long *([size_t](#))> array_out)

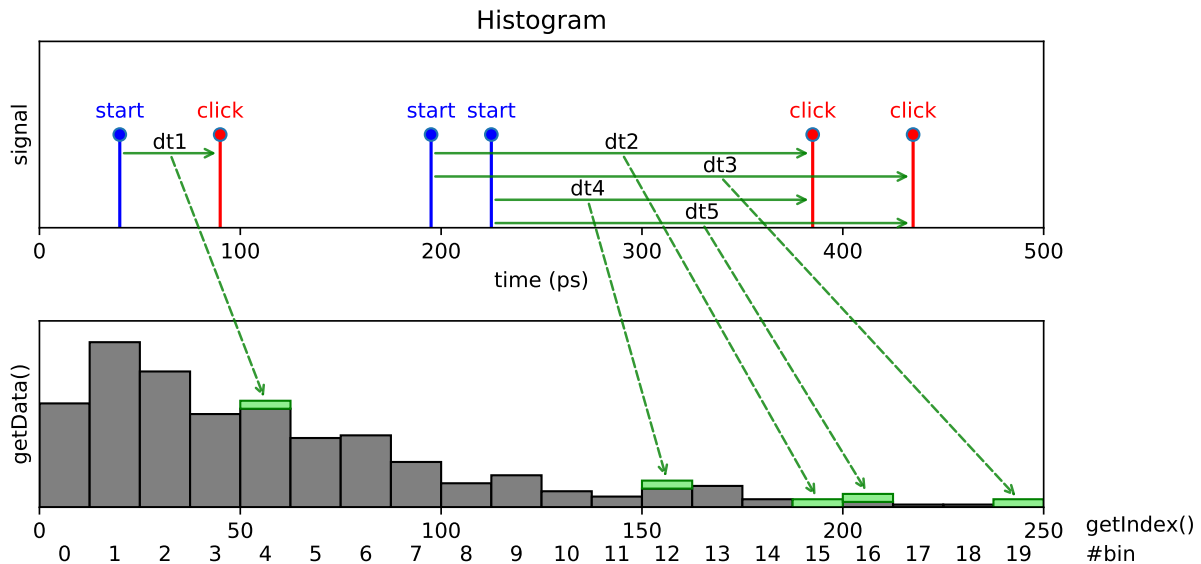
Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.
- void [on_start](#) () override
callback when the measurement class is started

Additional Inherited Members

7.27.1 Detailed Description

Accumulate time differences into a histogram.



This is a simple multiple start, multiple stop measurement. This is a special case of the more general 'TimeDifferences' measurement. Specifically, the thread waits for clicks on a first channel, the 'start channel', then measures the time difference between the last start click and all subsequent clicks on a second channel, the 'click channel', and stores them in a histogram. The histogram range and resolution is specified by the number of bins and the binwidth. Clicks that fall outside the histogram range are ignored. Data accumulation is performed independently for all start clicks. This type of measurement is frequently referred to as 'multiple start, multiple stop' measurement and corresponds to a full auto- or cross-correlation measurement.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 Histogram()

```
Histogram::Histogram (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t start_channel = CHANNEL_UNUSED,
    timestamp_t binwidth = 1000,
    int32_t n_bins = 1000 )
```

constructor of a [Histogram](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel that increments the count in a bin
<i>start_channel</i>	channel that sets start times relative to which clicks on the click channel are measured
<i>binwidth</i>	width of one histogram bin in ps
<i>n_bins</i>	number of bins in the histogram

7.27.2.2 ~Histogram()

```
Histogram::~~Histogram ( )
```

7.27.3 Member Function Documentation**7.27.3.1 clear_impl()**

```
void Histogram::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.27.3.2 getData()

```
void Histogram::getData (
    std::function< int32_t *(size_t)> array_out )
```

7.27.3.3 getIndex()

```
void Histogram::getIndex (
    std::function< long long *(size_t)> array_out )
```

7.27.3.4 next_impl()

```
bool Histogram::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.27.3.5 on_start()

```
void Histogram::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

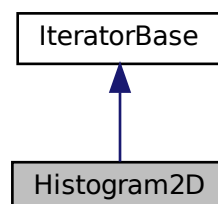
- [Iterators.h](#)

7.28 Histogram2D Class Reference

A 2-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.

```
#include <Iterators.h>
```

Inheritance diagram for Histogram2D:



Public Member Functions

- [Histogram2D](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) start_channel, [channel_t](#) stop_channel_1, [channel_t](#) stop_channel_2, [timestamp_t](#) binwidth_1, [timestamp_t](#) binwidth_2, [int32_t](#) n_bins_1, [int32_t](#) n_bins_2)
constructor of a [Histogram2D](#) measurement
- [~Histogram2D](#) ()
- void [getData](#) (std::function< [int32_t](#) *([size_t](#), [size_t](#))> array_out)
- void [getIndex](#) (std::function< long long *([size_t](#), [size_t](#), [size_t](#))> array_out)
- void [getIndex_1](#) (std::function< long long *([size_t](#))> array_out)
- void [getIndex_2](#) (std::function< long long *([size_t](#))> array_out)

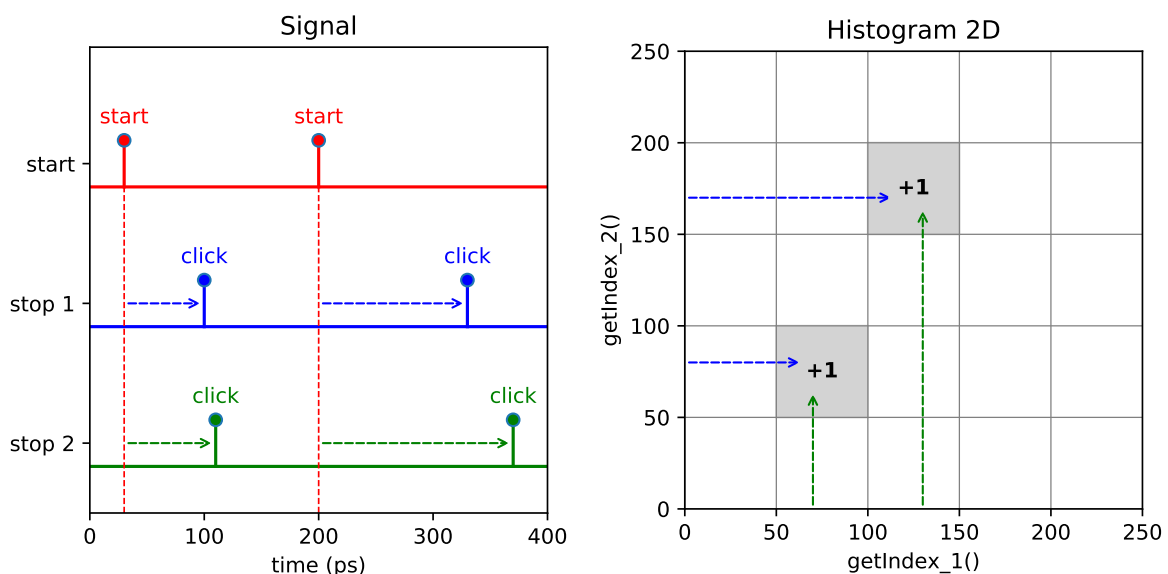
Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.

Additional Inherited Members

7.28.1 Detailed Description

A 2-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.



This measurement is a 2-dimensional version of the [Histogram](#) measurement. The measurement accumulates two-dimensional histogram where stop signals from two separate channels define the bin coordinate. For instance, this kind of measurement is similar to that of typical 2D NMR spectroscopy.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 Histogram2D()

```

Histogram2D::Histogram2D (
    TimeTaggerBase * tagger,
    channel_t start_channel,
    channel_t stop_channel_1,
    channel_t stop_channel_2,
    timestamp_t binwidth_1,
    timestamp_t binwidth_2,
    int32_t n_bins_1,
    int32_t n_bins_2 )

```

constructor of a [Histogram2D](#) measurement

Parameters

<i>tagger</i>	time tagger object
<i>start_channel</i>	channel on which start clicks are received
<i>stop_channel_1</i>	channel on which stop clicks for the time axis 1 are received
<i>stop_channel_2</i>	channel on which stop clicks for the time axis 2 are received
<i>binwidth_1</i>	bin width in ps for the time axis 1
<i>binwidth_2</i>	bin width in ps for the time axis 2
<i>n_bins_1</i>	the number of bins along the time axis 1
<i>n_bins_2</i>	the number of bins along the time axis 2

7.28.2.2 ~Histogram2D()

```

Histogram2D::~~Histogram2D ( )

```

7.28.3 Member Function Documentation

7.28.3.1 clear_impl()

```

void Histogram2D::clear_impl ( ) [override], [protected], [virtual]

```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.28.3.2 `getData()`

```
void Histogram2D::getData (
    std::function< int32_t *(size_t, size_t)> array_out )
```

Returns a two-dimensional array of size `n_bins_1` by `n_bins_2` containing the 2D histogram.

7.28.3.3 `getIndex()`

```
void Histogram2D::getIndex (
    std::function< long long *(size_t, size_t, size_t)> array_out )
```

Returns a 3D array containing two coordinate matrices (meshgrid) for time bins in ps for the time axes 1 and 2. For details on meshgrid please take a look at the respective documentation either for Matlab or Python NumPy

7.28.3.4 `getIndex_1()`

```
void Histogram2D::getIndex_1 (
    std::function< long long *(size_t)> array_out )
```

Returns a vector of size `n_bins_1` containing the bin locations in ps for the time axis 1.

7.28.3.5 `getIndex_2()`

```
void Histogram2D::getIndex_2 (
    std::function< long long *(size_t)> array_out )
```

Returns a vector of size `n_bins_2` containing the bin locations in ps for the time axis 2.

7.28.3.6 `next_impl()`

```
bool Histogram2D::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

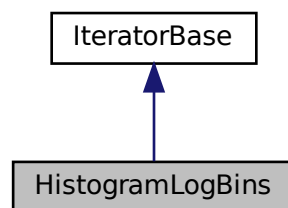
- [Iterators.h](#)

7.29 HistogramLogBins Class Reference

Accumulate time differences into a histogram with logarithmic increasing bin sizes.

```
#include <Iterators.h>
```

Inheritance diagram for HistogramLogBins:



Public Member Functions

- [HistogramLogBins](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) click_channel, [channel_t](#) start_channel, double exp_start, double exp_stop, int32_t n_bins)
constructor of a [HistogramLogBins](#) measurement
- [~HistogramLogBins](#) ()
- void [getData](#) (std::function< uint64_t *(size_t)> array_out)
returns the absolute counts for the bins
- void [getDataNormalizedCountsPerPs](#) (std::function< double *(size_t)> array_out)
returns the counts normalized by the binwidth of each bin
- void [getDataNormalizedG2](#) (std::function< double *(size_t)> array_out)
returns the counts normalized by the binwidth and the average count rate.
- void [getBinEdges](#) (std::function< long long *(size_t)> array_out)
returns the edges of the bins in ps

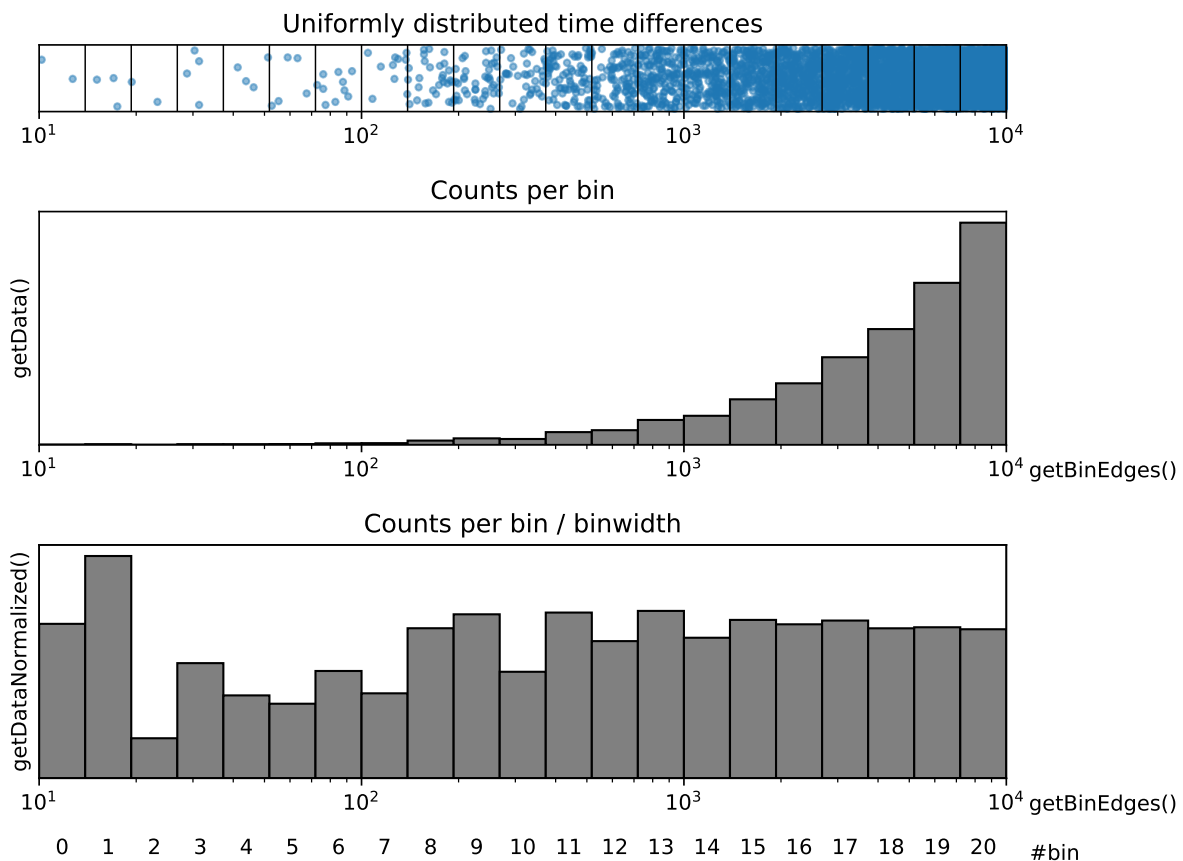
Protected Member Functions

- bool `next_impl` (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override
update iterator state
- void `clear_impl` () override
clear iterator state.

Additional Inherited Members

7.29.1 Detailed Description

Accumulate time differences into a histogram with logarithmic increasing bin sizes.



This is a multiple start, multiple stop measurement, and works the very same way as the histogram measurement but with logarithmic increasing bin widths. After initializing the measurement (or after an overflow) no data is accumulated in the histogram until the full histogram duration has passed to ensure a balanced count accumulation over the full histogram.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 HistogramLogBins()

```
HistogramLogBins::HistogramLogBins (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t start_channel,
    double exp_start,
    double exp_stop,
    int32_t n_bins )
```

constructor of a [HistogramLogBins](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel that increments the count in a bin
<i>start_channel</i>	channel that sets start times relative to which clicks on the click channel are measured
<i>exp_start</i>	exponent for the lowest time differences in the histogram: $10^{\text{exp_start}}$ s, lowest exp_start: -12 => 1ps
<i>exp_stop</i>	exponent for the highest time differences in the histogram: $10^{\text{exp_stop}}$ s
<i>n_bins</i>	total number of bins in the histogram

7.29.2.2 ~HistogramLogBins()

```
HistogramLogBins::~~HistogramLogBins ( )
```

7.29.3 Member Function Documentation

7.29.3.1 clear_impl()

```
void HistogramLogBins::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.29.3.2 getBinEdges()

```
void HistogramLogBins::getBinEdges (
    std::function< long long *(size_t)> array_out )
```

returns the edges of the bins in ps

7.29.3.3 getData()

```
void HistogramLogBins::getData (
    std::function< uint64_t *(size_t)> array_out )
```

returns the absolute counts for the bins

7.29.3.4 getDataNormalizedCountsPerPs()

```
void HistogramLogBins::getDataNormalizedCountsPerPs (
    std::function< double *(size_t)> array_out )
```

returns the counts normalized by the binwidth of each bin

7.29.3.5 getDataNormalizedG2()

```
void HistogramLogBins::getDataNormalizedG2 (
    std::function< double *(size_t)> array_out )
```

returns the counts normalized by the binwidth and the average count rate.

This matches the implementation of [Correlation::getDataNormalized](#)

7.29.3.6 next_impl()

```
bool HistogramLogBins::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

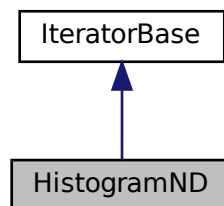
- [Iterators.h](#)

7.30 HistogramND Class Reference

A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.

```
#include <Iterators.h>
```

Inheritance diagram for HistogramND:



Public Member Functions

- [HistogramND](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) start_channel, std::vector< [channel_t](#) > stop_channels, std::vector< [timestamp_t](#) > binwidths, std::vector< int32_t > n_bins)
constructor of a [Histogram2D](#) measurement
- [~HistogramND](#) ()
- void [getData](#) (std::function< int32_t *(size_t)> array_out)
- void [getIndex](#) (std::function< long long *(size_t)> array_out, int32_t dim=0)

Protected Member Functions

- bool `next_impl` (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void `clear_impl` () override
clear [Iterator](#) state.

Additional Inherited Members

7.30.1 Detailed Description

A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.

This measurement is a N-dimensional version of the [Histogram](#) measurement. The measurement accumulates N-dimensional histogram where stop signals from N separate channels define the bin coordinate. For instance, this kind of measurement is similar to that of typical 2D NMR spectroscopy.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 HistogramND()

```
HistogramND::HistogramND (
    TimeTaggerBase * tagger,
    channel\_t start_channel,
    std::vector< channel\_t > stop_channels,
    std::vector< timestamp\_t > binwidths,
    std::vector< int32\_t > n_bins )
```

constructor of a [Histogram2D](#) measurement

Parameters

<i>tagger</i>	time tagger object
<i>start_channel</i>	channel on which start clicks are received
<i>stop_channels</i>	channels on which stop clicks for each time axis are received
<i>binwidths</i>	bin widths in ps for each time axis
<i>n_bins</i>	the number of bins along each time axis

7.30.2.2 ~HistogramND()

```
HistogramND::~~HistogramND ( )
```

7.30.3 Member Function Documentation

7.30.3.1 clear_impl()

```
void HistogramND::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.30.3.2 getData()

```
void HistogramND::getData (
    std::function< int32_t *(size_t)> array_out )
```

Returns a one-dimensional array of size of the product of `n_bins` containing the N-dimensional histogram. The 1D return value is in row-major ordering like on C, Python, C#. This conflicts with Fortran or Matlab. Please reshape the result to get the N-dimensional array.

7.30.3.3 getIndex()

```
void HistogramND::getIndex (
    std::function< long long *(size_t)> array_out,
    int32_t dim = 0 )
```

Returns a vector of size `n_bins[dim]` containing the bin locations in ps for the corresponding time axis.

7.30.3.4 next_impl()

```
bool HistogramND::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

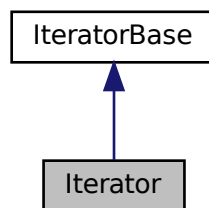
- [Iterators.h](#)

7.31 Iterator Class Reference

a deprecated simple event queue

```
#include <Iterators.h>
```

Inheritance diagram for Iterator:



Public Member Functions

- [Iterator](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) channel)
standard constructor
- [~Iterator](#) ()
- [timestamp_t next](#) ()
get next timestamp
- [uint64_t size](#) ()
get queue size

Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.

Additional Inherited Members

7.31.1 Detailed Description

a deprecated simple event queue

A simple [Iterator](#), just keeping a first-in first-out queue of event timestamps.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 Iterator()

```
Iterator::Iterator (
    TimeTaggerBase * tagger,
    channel_t channel )
```

standard constructor

Parameters

<i>tagger</i>	the backend
<i>channel</i>	the channel to get events from

7.31.2.2 ~Iterator()

```
Iterator::~~Iterator ( )
```

7.31.3 Member Function Documentation

7.31.3.1 clear_impl()

```
void Iterator::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.31.3.2 next()

```
timestamp_t Iterator::next ( )
```

get next timestamp

get the next timestamp from the queue.

7.31.3.3 next_impl()

```
bool Iterator::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.31.3.4 size()

```
uint64_t Iterator::size ( )
```

get queue size

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

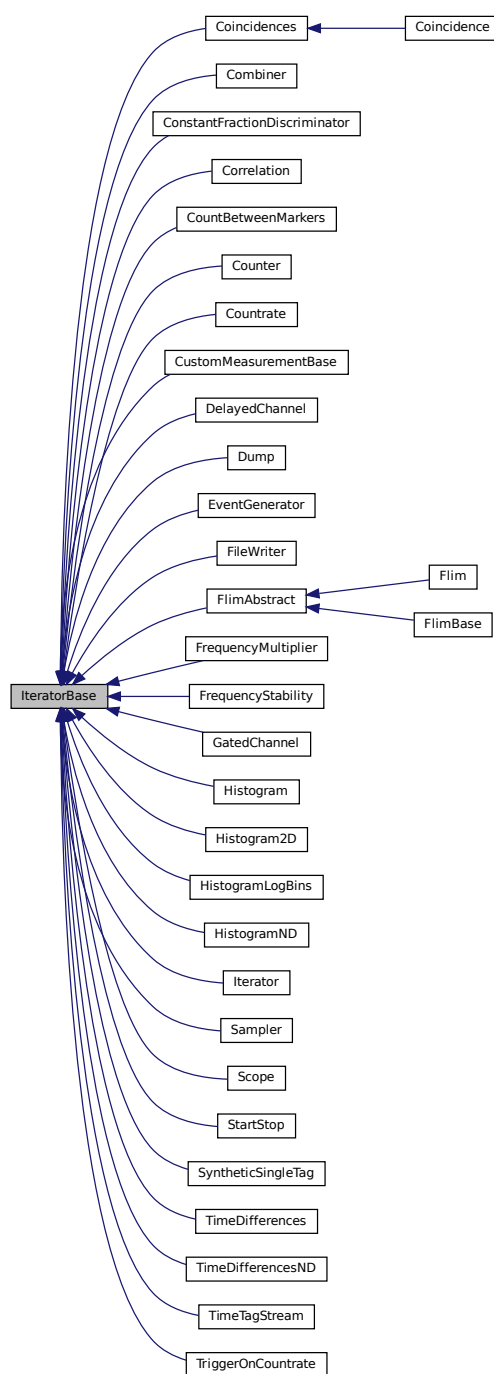
- [Iterators.h](#)

7.32 IteratorBase Class Reference

Base class for all iterators.

```
#include <TimeTagger.h>
```

Inheritance diagram for IteratorBase:



Public Member Functions

- virtual [~IteratorBase](#) ()
destructor, will unregister from the Time Tagger prior finalization.
- void [start](#) ()
Starts or continues data acquisition.
- void [startFor](#) (timestamp_t capture_duration, bool clear=true)
Starts or continues the data acquisition for the given duration.
- bool [waitUntilFinished](#) (int64_t timeout=-1)
Blocks the execution until the measurement has finished. Can be used with [startFor\(\)](#).
- void [stop](#) ()
After calling this method, the measurement will stop processing incoming tags.
- void [clear](#) ()
Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.
- bool [isRunning](#) ()
Returns True if the measurement is collecting the data.
- timestamp_t [getCaptureDuration](#) ()
Total capture duration since the measurement creation or last call to [clear\(\)](#).
- std::string [getConfiguration](#) ()
Fetches the overall configuration status of the measurement.

Protected Member Functions

- [IteratorBase](#) (TimeTaggerBase *tagger, std::string base_type_="IteratorBase", std::string extra_info_="")
Standard constructor, which will register with the Time Tagger backend.
- void [registerChannel](#) (channel_t channel)
register a channel
- void [unregisterChannel](#) (channel_t channel)
unregister a channel
- channel_t [getNewVirtualChannel](#) ()
allocate a new virtual output channel for this iterator
- void [finishInitialization](#) ()
method to call after finishing the initialization of the measurement
- virtual void [clear_impl](#) ()
*clear *Iterator* state.*
- virtual void [on_start](#) ()
callback when the measurement class is started
- virtual void [on_stop](#) ()
callback when the measurement class is stopped
- void [lock](#) ()
acquire update lock
- void [unlock](#) ()
release update lock
- OrderedBarrier::OrderInstance [parallelize](#) (OrderedPipeline &pipeline)
release lock and continue work in parallel
- std::unique_lock< std::mutex > [getLock](#) ()
acquire update lock
- virtual bool [next_impl](#) (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)=0
update iterator state
- void [finish_running](#) ()
Callback for the measurement to stop itself.

Protected Attributes

- `std::set< channel_t > channels_registered`
list of channels used by the iterator
- `bool running`
running state of the iterator
- `bool autostart`
Condition if this measurement shall be started by the finishInitialization callback.
- `TimeTaggerBase * tagger`
Pointer to the corresponding Time Tagger object.
- `timestamp_t capture_duration`
Duration the iterator has already processed data.

7.32.1 Detailed Description

Base class for all iterators.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 IteratorBase()

```
IteratorBase::IteratorBase (
    TimeTaggerBase * tagger,
    std::string base_type_ = "IteratorBase",
    std::string extra_info_ = "" ) [protected]
```

Standard constructor, which will register with the Time Tagger backend.

7.32.2.2 ~IteratorBase()

```
virtual IteratorBase::~~IteratorBase ( ) [virtual]
```

destructor, will unregister from the Time Tagger prior finalization.

7.32.3 Member Function Documentation

7.32.3.1 clear()

```
void IteratorBase::clear ( )
```

Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.

7.32.3.2 clear_impl()

```
virtual void IteratorBase::clear_impl ( ) [inline], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented in [FrequencyStability](#), [Sampler](#), [Flim](#), [FlimAbstract](#), [CustomMeasurementBase](#), [EventGenerator](#), [FileWriter](#), [Scope](#), [Correlation](#), [HistogramLogBins](#), [Histogram](#), [TimeDifferencesND](#), [HistogramND](#), [Histogram2D](#), [TimeDifferences](#), [StartStop](#), [Dump](#), [TimeTagStream](#), [Iterator](#), [Countrate](#), [Counter](#), [CountBetweenMarkers](#), and [Combiner](#).

7.32.3.3 finish_running()

```
void IteratorBase::finish_running ( ) [protected]
```

Callback for the measurement to stop itself.

It shall only be called while the measurement mutex is locked. It will make sure that no new data is passed to this measurement. The caller has to call `on_stop` itself if needed.

7.32.3.4 finishInitialization()

```
void IteratorBase::finishInitialization ( ) [protected]
```

method to call after finishing the initialization of the measurement

7.32.3.5 getCaptureDuration()

```
timestamp_t IteratorBase::getCaptureDuration ( )
```

Total capture duration since the measurement creation or last call to [clear\(\)](#).

Returns

Capture duration in ps

7.32.3.6 getConfiguration()

```
std::string IteratorBase::getConfiguration ( )
```

Fetches the overall configuration status of the measurement.

Returns

a JSON serialized string with all configuration and status flags.

7.32.3.7 getLock()

```
std::unique_lock<std::mutex> IteratorBase::getLock ( ) [protected]
```

acquire update lock

All mutable operations on a iterator are guarded with an update mutex. Implementers are advised to lock an iterator, whenever internal state is queried or changed.

Returns

a lock object, which releases the lock when this instance is freed

7.32.3.8 getNewVirtualChannel()

```
channel_t IteratorBase::getNewVirtualChannel ( ) [protected]
```

allocate a new virtual output channel for this iterator

7.32.3.9 isRunning()

```
bool IteratorBase::isRunning ( )
```

Returns True if the measurement is collecting the data.

This method will returns False if the measurement was stopped manually by calling [stop\(\)](#) or automatically after calling [startFor\(\)](#) and the duration has passed.

Note

All measurements start accumulating data immediately after their creation.

Returns

True if the measurement is still running

7.32.3.10 lock()

```
void IteratorBase::lock ( ) [protected]
```

acquire update lock

All mutable operations on a iterator are guarded with an update mutex. Implementers are advised to [lock\(\)](#) an iterator, whenever internal state is queried or changed.

7.32.3.11 next_impl()

```
virtual bool IteratorBase::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [protected], [pure virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implemented in [FrequencyStability](#), [SyntheticSingleTag](#), [Sampler](#), [FlimAbstract](#), [CustomMeasurementBase](#), [EventGenerator](#), [FileWriter](#), [ConstantFractionDiscriminator](#), [Scope](#), [Correlation](#), [HistogramLogBins](#), [Histogram](#), [TimeDifferencesND](#), [HistogramND](#), [Histogram2D](#), [TimeDifferences](#), [StartStop](#), [Dump](#), [TimeTagStream](#), [Iterator](#), [FrequencyMultiplier](#), [GatedChannel](#), [TriggerOnCountrate](#), [DelayedChannel](#), [Countrate](#), [Coincidences](#), [Counter](#), [CountBetweenMarkers](#), and [Combiner](#).

7.32.3.12 on_start()

```
virtual void IteratorBase::on_start ( ) [inline], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented in [FrequencyStability](#), [Sampler](#), [FlimAbstract](#), [CustomMeasurementBase](#), [EventGenerator](#), [FileWriter](#), [ConstantFractionDiscriminator](#), [Histogram](#), [TimeDifferencesND](#), [TimeDifferences](#), [StartStop](#), [Dump](#), [TriggerOnCountrate](#), [DelayedChannel](#), [Countrate](#), and [Counter](#).

7.32.3.13 on_stop()

```
virtual void IteratorBase::on_stop ( ) [inline], [protected], [virtual]
```

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented in [CustomMeasurementBase](#), [FileWriter](#), and [Dump](#).

7.32.3.14 parallelize()

```
OrderedBarrier::OrderInstance IteratorBase::parallelize (
    OrderedPipeline & pipeline ) [protected]
```

release lock and continue work in parallel

The measurement's lock is released, allowing this measurement to continue, while still executing work in parallel.

Returns

a ordered barrier instance that can be synced afterwards.

7.32.3.15 registerChannel()

```
void IteratorBase::registerChannel (
    channel_t channel ) [protected]
```

register a channel

Only channels registered by any iterator attached to a backend are delivered over the usb.

Parameters

<i>channel</i>	the channel
----------------	-------------

7.32.3.16 start()

```
void IteratorBase::start ( )
```

Starts or continues data acquisition.

This method is implicitly called when a measurement object is created.

7.32.3.17 startFor()

```
void IteratorBase::startFor (
    timestamp_t capture_duration,
    bool clear = true )
```

Starts or continues the data acquisition for the given duration.

After the duration time, the method [stop\(\)](#) is called and [isRunning\(\)](#) will return False. Whether the accumulated data is cleared at the beginning of [startFor\(\)](#) is controlled with the second parameter `clear`, which is True by default.

Parameters

<i>capture_duration</i>	capture duration in picoseconds until the measurement is stopped
<i>clear</i>	resets the data acquired

7.32.3.18 stop()

```
void IteratorBase::stop ( )
```

After calling this method, the measurement will stop processing incoming tags.

Use [start\(\)](#) or [startFor\(\)](#) to continue or restart the measurement.

7.32.3.19 unlock()

```
void IteratorBase::unlock ( ) [protected]
```

release update lock

see [lock\(\)](#)

7.32.3.20 unregisterChannel()

```
void IteratorBase::unregisterChannel (
    channel_t channel ) [protected]
```

unregister a channel

Parameters

<i>channel</i>	the channel
----------------	-------------

7.32.3.21 waitUntilFinished()

```
bool IteratorBase::waitUntilFinished (
    int64_t timeout = -1 )
```

Blocks the execution until the measurement has finished. Can be used with [startFor\(\)](#).

waitUntilFinished will wait according to the timeout and return true if the iterator finished or false if not. Furthermore, when waitUntilFinished is called on a iterator running indefinitely, it will log an error and return immediately.

Parameters

<i>timeout</i>	time in milliseconds to wait for the measurements. If negative, wait until finished.
----------------	--

Returns

True if the measurement has finished, false on timeout

7.32.4 Member Data Documentation

7.32.4.1 autostart

```
bool IteratorBase::autostart [protected]
```

Condition if this measurement shall be started by the finishInitialization callback.

7.32.4.2 capture_duration

```
timestamp_t IteratorBase::capture_duration [protected]
```

Duration the iterator has already processed data.

7.32.4.3 channels_registered

```
std::set<channel_t> IteratorBase::channels_registered [protected]
```

list of channels used by the iterator

7.32.4.4 running

```
bool IteratorBase::running [protected]
```

running state of the iterator

7.32.4.5 tagger

```
TimeTaggerBase* IteratorBase::tagger [protected]
```

Pointer to the corresponding Time Tagger object.

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

- [TimeTagger.h](#)

7.33 OrderedBarrier Class Reference

Helper for implementing parallel measurements.

```
#include <TimeTagger.h>
```

Classes

- class [OrderInstance](#)
Internal object for serialization.

Public Member Functions

- [OrderedBarrier](#) ()
- [~OrderedBarrier](#) ()
- [OrderInstance queue](#) ()
- void [waitUntilFinished](#) ()

7.33.1 Detailed Description

Helper for implementing parallel measurements.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 OrderedBarrier()

```
OrderedBarrier::OrderedBarrier ( )
```

7.33.2.2 ~OrderedBarrier()

```
OrderedBarrier::~~OrderedBarrier ( )
```

7.33.3 Member Function Documentation

7.33.3.1 queue()

```
OrderInstance OrderedBarrier::queue ( )
```

7.33.3.2 waitUntilFinished()

```
void OrderedBarrier::waitUntilFinished ( )
```

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

- [TimeTagger.h](#)

7.34 OrderedPipeline Class Reference

Helper for implementing parallel measurements.

```
#include <TimeTagger.h>
```

Public Member Functions

- [OrderedPipeline](#) ()
- [~OrderedPipeline](#) ()

7.34.1 Detailed Description

Helper for implementing parallel measurements.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 OrderedPipeline()

```
OrderedPipeline::OrderedPipeline ( )
```

7.34.2.2 ~OrderedPipeline()

```
OrderedPipeline::~~OrderedPipeline ( )
```

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

- [TimeTagger.h](#)

7.35 OrderedBarrier::OrderInstance Class Reference

Internal object for serialization.

```
#include <TimeTagger.h>
```

Public Member Functions

- [OrderInstance](#) ()
- [OrderInstance](#) ([OrderedBarrier](#) *parent, uint64_t instance_id)
- [~OrderInstance](#) ()
- void [sync](#) ()
- void [release](#) ()

7.35.1 Detailed Description

Internal object for serialization.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 OrderInstance() [1/2]

```
OrderedBarrier::OrderInstance::OrderInstance ( )
```

7.35.2.2 OrderInstance() [2/2]

```
OrderedBarrier::OrderInstance::OrderInstance (
    OrderedBarrier * parent,
    uint64_t instance_id )
```

7.35.2.3 ~OrderInstance()

```
OrderedBarrier::OrderInstance::~~OrderInstance ( )
```

7.35.3 Member Function Documentation

7.35.3.1 release()

```
void OrderedBarrier::OrderInstance::release ( )
```

7.35.3.2 sync()

```
void OrderedBarrier::OrderInstance::sync ( )
```

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

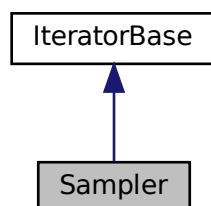
- [TimeTagger.h](#)

7.36 Sampler Class Reference

a triggered sampling measurement

```
#include <Iterators.h>
```

Inheritance diagram for Sampler:



Public Member Functions

- [Sampler](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) trigger, std::vector< [channel_t](#) > channels, size_t max_triggers)
constructor of a [Sampler](#) measurement
- [~Sampler](#) ()
- void [getData](#) (std::function< long long *(size_t, size_t)> array_out)
fetches the internal data as 2D array.
- void [getDataAsMask](#) (std::function< long long *(size_t, size_t)> array_out)
fetches the internal data as 2D array with a channel mask.

Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.
- void [on_start](#) () override
callback when the measurement class is started

Additional Inherited Members

7.36.1 Detailed Description

a triggered sampling measurement

This measurement class will perform a triggered sampling measurement. So for every event on the trigger input, the current state (low : 0, high : 1, unknown : 2) will be written to an internal buffer. Fetching the data of the internal buffer will clear its internal state without any deadtime. So every event will be recorded exactly once.

The unknown state might happen after an overflow without an event on the input channel. This processing assumes that no event was filtered by the deadtime. Else invalid data will be reported till the next event on this input channel.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 Sampler()

```
Sampler::Sampler (
    TimeTaggerBase * tagger,
    channel_t trigger,
    std::vector< channel_t > channels,
    size_t max_triggers )
```

constructor of a [Sampler](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>trigger</i>	the channel which shall trigger the measurement
<i>channels</i>	a list of channels which will be recorded for every trigger
<i>max_triggers</i>	the maximum amount of triggers without <code>getData*</code> call till this measurement will stop itself

7.36.2.2 ~Sampler()

```
Sampler::~Sampler ( )
```

7.36.3 Member Function Documentation

7.36.3.1 clear_impl()

```
void Sampler::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.36.3.2 getData()

```
void Sampler::getData (
    std::function< long long *(size_t, size_t)> array_out )
```

fetches the internal data as 2D array.

Its layout is roughly: [[timestamp of first trigger, state of channel 0, state of channel 1, ...], [timestamp of second trigger, state of channel 0, state of channel 1, ...], ...] Where state means: 0 – low 1 – high 2 – undefined (after overflow)

7.36.3.3 getDataAsMask()

```
void Sampler::getDataAsMask (
    std::function< long long *(size_t, size_t)> array_out )
```

fetches the internal data as 2D array with a channel mask.

Its layout is roughly: [[timestamp of first trigger, (state of channel 0) << 0 | (state of channel 1) << 1 | ... | undefined << 63], [timestamp of second trigger, (state of channel 0) << 0 | (state of channel 1) << 1 | ... | undefined << 63], ...] Where state means: 0 – low or undefined (after overflow) 1 – high

7.36.3.4 next_impl()

```
bool Sampler::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.36.3.5 on_start()

```
void Sampler::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

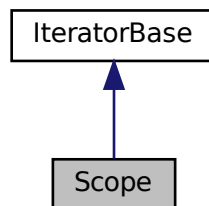
- [Iterators.h](#)

7.37 Scope Class Reference

a scope measurement

```
#include <Iterators.h>
```

Inheritance diagram for Scope:



Public Member Functions

- `Scope (TimeTaggerBase *tagger, std::vector< channel_t > event_channels, channel_t trigger_channel, timestamp_t window_size=1000000000, int32_t n_traces=1, int32_t n_max_events=1000)`
constructor of a [Scope](#) measurement
- `~Scope ()`
- `bool ready ()`
- `int32_t triggered ()`
- `std::vector< std::vector< Event > > getData ()`
- `timestamp_t getWindowSize ()`

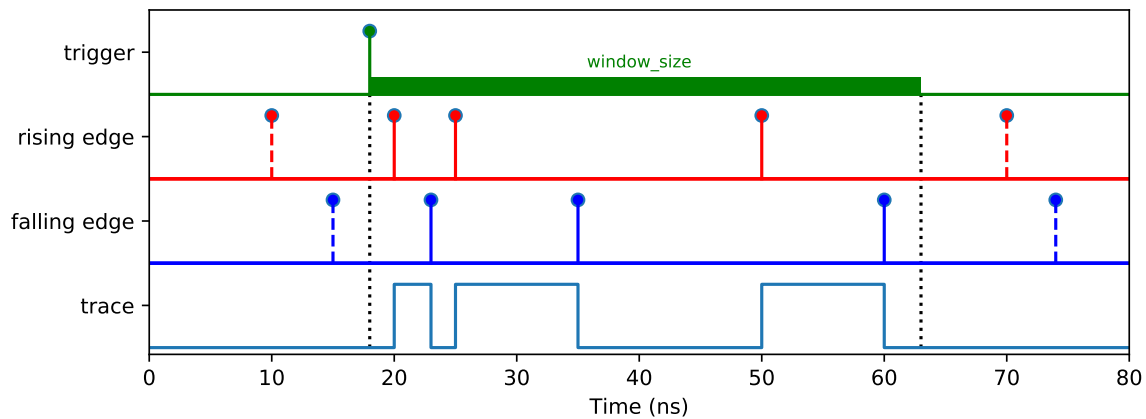
Protected Member Functions

- `bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)` override
update iterator state
- `void clear_impl ()` override
clear [Iterator](#) state.

Additional Inherited Members

7.37.1 Detailed Description

a scope measurement



The [Scope](#) class allows to visualize time tags for rising and falling edges in a time trace diagram similarly to an ultrafast logic analyzer. The trace recording is synchronized to a trigger signal which can be any physical or virtual channel. However, only physical channels can be specified to the `event_channels` parameter. Additionally, one has to specify the `window_size` which is the timetrace duration to be recorded, the number of traces to be recorded and the maximum number of events to be detected. If `n_traces < 1` then retriggering will occur infinitely, which is similar to the “normal” mode of an oscilloscope.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 Scope()

```
Scope::Scope (
    TimeTaggerBase * tagger,
    std::vector< channel_t > event_channels,
    channel_t trigger_channel,
    timestamp_t window_size = 1000000000,
    int32_t n_traces = 1,
    int32_t n_max_events = 1000 )
```

constructor of a [Scope](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>event_channels</i>	channels which are captured
<i>trigger_channel</i>	channel that starts a new trace
<i>window_size</i>	window time of each trace
<i>n_traces</i>	amount of traces (<code>n_traces < 1</code> , automatic retrigger)
<i>n_max_events</i>	maximum number of tags in each trace

7.37.2.2 ~Scope()

```
Scope::~~Scope ( )
```

7.37.3 Member Function Documentation

7.37.3.1 clear_impl()

```
void Scope::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.37.3.2 getData()

```
std::vector<std::vector<Event> > Scope::getData ( )
```

7.37.3.3 getWindowSize()

```
timestamp\_t Scope::getWindowSize ( )
```

7.37.3.4 next_impl()

```
bool Scope::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.37.3.5 ready()

```
bool Scope::ready ( )
```

7.37.3.6 triggered()

```
int32_t Scope::triggered ( )
```

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

- [Iterators.h](#)

7.38 SoftwareClockState Struct Reference

```
#include <TimeTagger.h>
```

Public Attributes

- [timestamp_t](#) clock_period
- [channel_t](#) input_channel
- [channel_t](#) ideal_clock_channel
- double averaging_periods
- bool enabled
- bool is_locked
- [uint32_t](#) error_counter
- [timestamp_t](#) last_ideal_clock_event
- double period_error
- double phase_error_estimation

7.38.1 Member Data Documentation

7.38.1.1 averaging_periods

`double SoftwareClockState::averaging_periods`

7.38.1.2 clock_period

`timestamp_t SoftwareClockState::clock_period`

7.38.1.3 enabled

`bool SoftwareClockState::enabled`

7.38.1.4 error_counter

`uint32_t SoftwareClockState::error_counter`

7.38.1.5 ideal_clock_channel

`channel_t SoftwareClockState::ideal_clock_channel`

7.38.1.6 input_channel

`channel_t SoftwareClockState::input_channel`

7.38.1.7 is_locked

`bool SoftwareClockState::is_locked`

7.38.1.8 last_ideal_clock_event

```
timestamp_t SoftwareClockState::last_ideal_clock_event
```

7.38.1.9 period_error

```
double SoftwareClockState::period_error
```

7.38.1.10 phase_error_estimation

```
double SoftwareClockState::phase_error_estimation
```

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

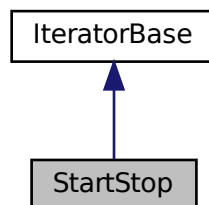
- [TimeTagger.h](#)

7.39 StartStop Class Reference

simple start-stop measurement

```
#include <Iterators.h>
```

Inheritance diagram for StartStop:



Public Member Functions

- [StartStop](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) click_channel, [channel_t](#) start_channel=[CHANNEL_UNUSED](#), [timestamp_t](#) binwidth=1000)
constructor of [StartStop](#)
- [~StartStop](#) ()
- void [getData](#) (std::function< long long *(size_t, size_t)> array_out)

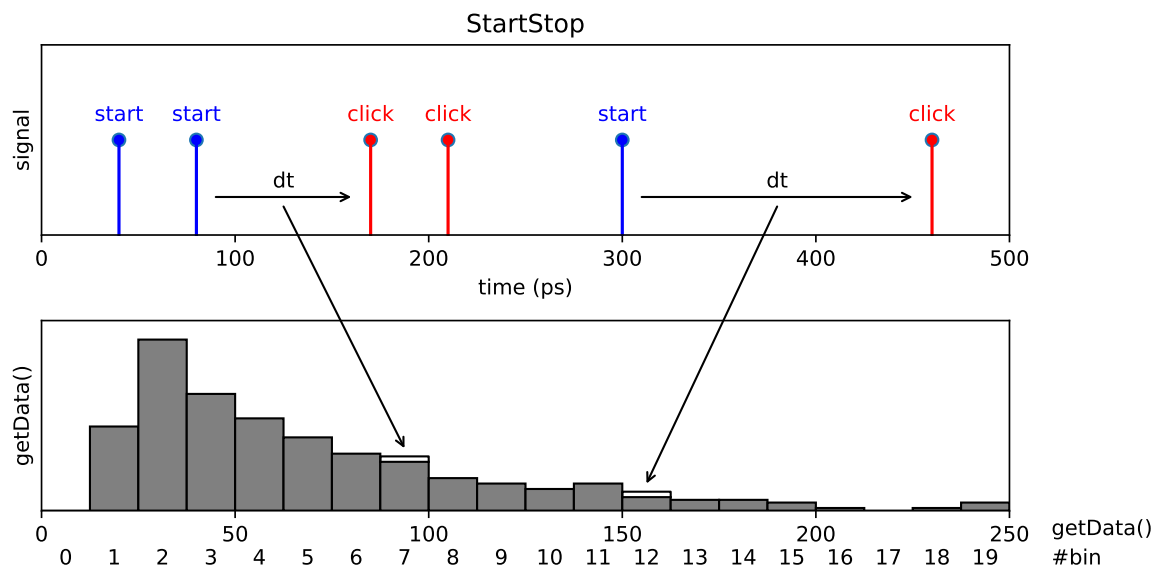
Protected Member Functions

- bool `next_impl` (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override
update iterator state
- void `clear_impl` () override
clear iterator state.
- void `on_start` () override
callback when the measurement class is started

Additional Inherited Members

7.39.1 Detailed Description

simple start-stop measurement



This class performs a start-stop measurement between two channels and stores the time differences in a histogram. The histogram resolution is specified beforehand (binwidth) but the histogram range is unlimited. It is adapted to the largest time difference that was detected. Thus all pairs of subsequent clicks are registered.

Be aware, on long-running measurements this may considerably slow down system performance and even crash the system entirely when attached to an unsuitable signal source.

7.39.2 Constructor & Destructor Documentation

7.39.2.1 StartStop()

```
StartStop::StartStop (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t start_channel = CHANNEL_UNUSED,
    timestamp_t binwidth = 1000 )
```

constructor of `StartStop`

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel for stop clicks
<i>start_channel</i>	channel for start clicks
<i>binwidth</i>	width of one histogram bin in ps

7.39.2.2 ~StartStop()

```
StartStop::~~StartStop ( )
```

7.39.3 Member Function Documentation**7.39.3.1 clear_impl()**

```
void StartStop::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.39.3.2 getData()

```
void StartStop::getData (
    std::function< long long *(size_t, size_t)> array_out )
```

7.39.3.3 next_impl()

```
bool StartStop::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.39.3.4 on_start()

```
void StartStop::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

7.40 SynchronizedMeasurements Class Reference

start, stop and clear several measurements synchronized

```
#include <Iterators.h>
```

Public Member Functions

- [SynchronizedMeasurements](#) ([TimeTaggerBase](#) *tagger)
construct a [SynchronizedMeasurements](#) object
- [~SynchronizedMeasurements](#) ()
- void [registerMeasurement](#) ([IteratorBase](#) *measurement)
register a measurement (iterator) to the SynchronizedMeasurements-group.
- void [unregisterMeasurement](#) ([IteratorBase](#) *measurement)
unregister a measurement (iterator) from the SynchronizedMeasurements-group.
- void [clear](#) ()
clear all registered measurements synchronously
- void [start](#) ()
start all registered measurements synchronously
- void [stop](#) ()
stop all registered measurements synchronously
- void [startFor](#) ([timestamp_t](#) capture_duration, bool [clear](#)=true)
start all registered measurements synchronously, and stops them after the capture_duration
- bool [waitUntilFinished](#) (int64_t timeout=-1)
wait until all registered measurements have finished running.
- bool [isRunning](#) ()
check if any iterator is running
- [TimeTaggerBase](#) * [getTagger](#) ()
Returns a proxy tagger object, which shall be used to create immediately registered measurements.

Protected Member Functions

- void `runCallback` (`TimeTaggerBase::IteratorCallback` callback, `std::unique_lock< std::mutex > &lk`, bool block=true)

run a callback on all registered measurements synchronously

7.40.1 Detailed Description

start, stop and clear several measurements synchronized

For the case that several measurements should be started, stopped or cleared at the very same time, a `SynchronizedMeasurements` object can be create to which all the measurements (also called iterators) can be registered with `.registerMeasurement(measurement)`. Calling `.stop()`, `.start()` or `.clear()` on the `SynchronizedMeasurements` object will call the respective method on each of the registered measurements at the very same time. That means that all measurements taking part will have processed the very same time tags.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 SynchronizedMeasurements()

```
SynchronizedMeasurements::SynchronizedMeasurements (
    TimeTaggerBase * tagger )
```

construct a `SynchronizedMeasurements` object

Parameters

<code>tagger</code>	reference to a <code>TimeTagger</code>
---------------------	--

7.40.2.2 ~SynchronizedMeasurements()

```
SynchronizedMeasurements::~~SynchronizedMeasurements ( )
```

7.40.3 Member Function Documentation

7.40.3.1 clear()

```
void SynchronizedMeasurements::clear ( )
```

clear all registered measurements synchronously

7.40.3.2 getTagger()

```
TimeTaggerBase* SynchronizedMeasurements::getTagger ( )
```

Returns a proxy tagger object, which shall be used to create immediately registered measurements.

Those measurements will not start automatically.

7.40.3.3 isRunning()

```
bool SynchronizedMeasurements::isRunning ( )
```

check if any iterator is running

7.40.3.4 registerMeasurement()

```
void SynchronizedMeasurements::registerMeasurement (
    IteratorBase * measurement )
```

register a measurement (iterator) to the SynchronizedMeasurements-group.

All available methods called on the [SynchronizedMeasurements](#) will happen at the very same time for all the registered measurements.

7.40.3.5 runCallback()

```
void SynchronizedMeasurements::runCallback (
    TimeTaggerBase::IteratorCallback callback,
    std::unique_lock< std::mutex > & lk,
    bool block = true ) [protected]
```

run a callback on all registered measurements synchronously

Please keep in mind that the callback is copied for each measurement. So please avoid big captures.

7.40.3.6 start()

```
void SynchronizedMeasurements::start ( )
```

start all registered measurements synchronously

7.40.3.7 startFor()

```
void SynchronizedMeasurements::startFor (
    timestamp_t capture_duration,
    bool clear = true )
```

start all registered measurements synchronously, and stops them after the capture_duration

7.40.3.8 stop()

```
void SynchronizedMeasurements::stop ( )
```

stop all registered measurements synchronously

7.40.3.9 unregisterMeasurement()

```
void SynchronizedMeasurements::unregisterMeasurement (
    IteratorBase * measurement )
```

unregister a measurement (iterator) from the SynchronizedMeasurements-group.

Stops synchronizing calls on the selected measurement, if the measurement is not within this synchronized group, the method does nothing.

7.40.3.10 waitUntilFinished()

```
bool SynchronizedMeasurements::waitUntilFinished (
    int64_t timeout = -1 )
```

wait until all registered measurements have finished running.

Parameters

<i>timeout</i>	time in milliseconds to wait for the measurements. If negative, wait until finished.
----------------	--

waitUntilFinished will wait according to the timeout and return true if all measurements finished or false if not. Furthermore, when waitUntilFinished is called on a set running indefinitely, it will log an error and return immediately.

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

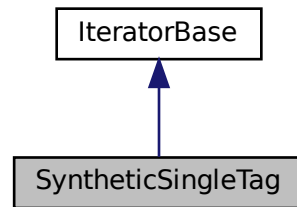
- [Iterators.h](#)

7.41 SyntheticSingleTag Class Reference

synthetic trigger timetag generator.

```
#include <Iterators.h>
```

Inheritance diagram for SyntheticSingleTag:



Public Member Functions

- [SyntheticSingleTag](#) ([TimeTaggerBase](#) *[tagger](#), [channel_t](#) [base_channel](#)=[CHANNEL_UNUSED](#))
Construct a pulse event generator.
- [~SyntheticSingleTag](#) ()
- void [trigger](#) ()
Generate a timetag for each call of this method.
- [channel_t](#) [getChannel](#) () const

Protected Member Functions

- bool [next_impl](#) (std::vector< [Tag](#) > &[incoming_tags](#), [timestamp_t](#) [begin_time](#), [timestamp_t](#) [end_time](#)) override
update iterator state

Additional Inherited Members

7.41.1 Detailed Description

synthetic trigger timetag generator.

Creates timetags based on a trigger method. Whenever the user calls the 'trigger' method, a timetag will be added to the [base_channel](#).

This synthetic channel can inject timetags into an existing channel or create a new virtual channel.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 SyntheticSingleTag()

```
SyntheticSingleTag::SyntheticSingleTag (
    TimeTaggerBase * tagger,
    channel_t base\_channel = CHANNEL\_UNUSED )
```

Construct a pulse event generator.

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>base_channel</i>	base channel to which this signal will be added. If unused, a new channel will be created.

7.41.2.2 ~SyntheticSingleTag()

```
SyntheticSingleTag::~SyntheticSingleTag ( )
```

7.41.3 Member Function Documentation**7.41.3.1 getChannel()**

```
channel_t SyntheticSingleTag::getChannel ( ) const
```

7.41.3.2 next_impl()

```
bool SyntheticSingleTag::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.41.3.3 trigger()

```
void SyntheticSingleTag::trigger ( )
```

Generate a timetag for each call of this method.

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

- [Iterators.h](#)

7.42 Tag Struct Reference

a single event on a channel

```
#include <TimeTagger.h>
```

Public Types

- enum [Type](#) : unsigned char {
[Type::TimeTag](#) = 0, [Type::Error](#) = 1, [Type::OverflowBegin](#) = 2, [Type::OverflowEnd](#) = 3,
[Type::MissedEvents](#) = 4 }

This enum marks what kind of event this object represents.

Public Attributes

- enum [Tag::Type](#) type
- char [reserved](#)
8 bit padding
- unsigned short [missed_events](#)
Amount of missed events in overflow mode.
- [channel_t](#) channel
the channel number
- [timestamp_t](#) time
the timestamp of the event in picoseconds

7.42.1 Detailed Description

a single event on a channel

Channel events are passed from the backend to registered iterators by the `IteratorBase::next()` callback function.

A [Tag](#) describes a single event on a channel.

7.42.2 Member Enumeration Documentation

7.42.2.1 Type

```
enum Tag::Type : unsigned char [strong]
```

This enum marks what kind of event this object represents.

- TimeTag: a normal event from any input channel
- Error: an error in the internal data processing, e.g. on plugging the external clock. This invalidates the global time
- OverflowBegin: this marks the begin of an interval with incomplete data because of too high data rates
- OverflowEnd: this marks the end of the interval. All events, which were lost in this interval, have been handled
- MissedEvents: this virtual event signals the amount of lost events per channel within an overflow interval. Repeated usage for higher amounts of events

Enumerator

TimeTag	
Error	
OverflowBegin	
OverflowEnd	
MissedEvents	

7.42.3 Member Data Documentation

7.42.3.1 channel

```
channel_t Tag::channel
```

the channel number

7.42.3.2 missed_events

```
unsigned short Tag::missed_events
```

Amount of missed events in overflow mode.

Within overflow intervals, the timing of all events is skipped. However, the total amount of events is still recorded. For events with type = MissedEvents, this indicates that a given amount of tags for this channel have been skipped in the interval. Note: There might be many missed events tags per overflow interval and channel. The accumulated amount represents the total skipped events.

7.42.3.3 reserved

```
char Tag::reserved
```

8 bit padding

Reserved for future use. Set it to zero.

7.42.3.4 time

```
timestamp_t Tag::time
```

the timestamp of the event in picoseconds

7.42.3.5 type

```
enum Tag::Type Tag::type
```

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

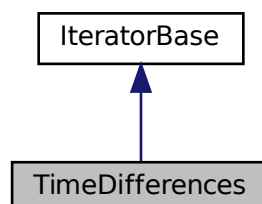
- [TimeTagger.h](#)

7.43 TimeDifferences Class Reference

Accumulates the time differences between clicks on two channels in one or more histograms.

```
#include <Iterators.h>
```

Inheritance diagram for TimeDifferences:



Public Member Functions

- `TimeDifferences` (`TimeTaggerBase *tagger`, `channel_t click_channel`, `channel_t start_channel=CHANNEL_UNUSED`, `channel_t next_channel=CHANNEL_UNUSED`, `channel_t sync_channel=CHANNEL_UNUSED`, `timestamp_t binwidth=1000`, `int32_t n_bins=1000`, `int32_t n_histograms=1`)
constructor of a `TimeDifferences` measurement
- `~TimeDifferences` ()
- void `getData` (`std::function< int32_t *(size_t, size_t)> array_out`)
returns a two-dimensional array of size 'n_bins' by 'n_histograms' containing the histograms
- void `getIndex` (`std::function< long long *(size_t)> array_out`)
returns a vector of size 'n_bins' containing the time bins in ps
- void `setMaxCounts` (`uint64_t max_counts`)
set the number of rollovers at which the measurement stops integrating
- `uint64_t` `getCounts` ()
returns the number of rollovers (histogram index resets)
- bool `ready` ()
returns 'true' when the required number of rollovers set by 'setMaxCounts' has been reached

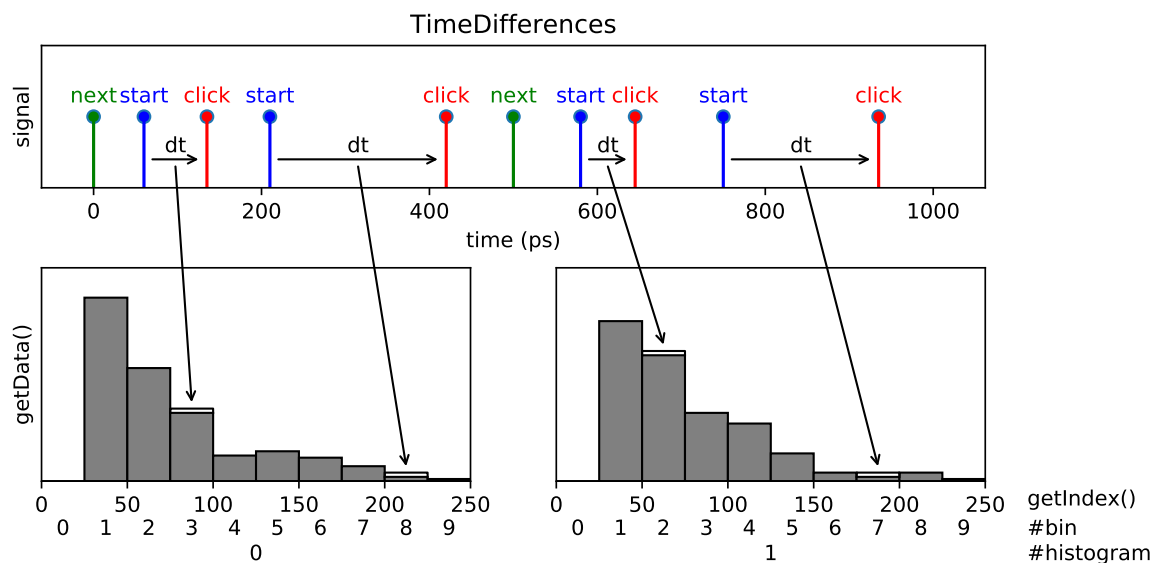
Protected Member Functions

- bool `next_impl` (`std::vector< Tag > &incoming_tags`, `timestamp_t begin_time`, `timestamp_t end_time`) override
update iterator state
- void `clear_impl` () override
clear `Iterator` state.
- void `on_start` () override
callback when the measurement class is started

Additional Inherited Members

7.43.1 Detailed Description

Accumulates the time differences between clicks on two channels in one or more histograms.



A multidimensional histogram measurement with the option up to include three additional channels that control how to step through the indices of the histogram array. This is a very powerful and generic measurement. You can use it to record cross-correlation, lifetime measurements, fluorescence lifetime imaging and many more measurements based on pulsed excitation. Specifically, the measurement waits for a tag on the 'start_channel', then measures the time difference between the start tag and all subsequent tags on the 'click_channel' and stores them in a histogram. If no 'start_channel' is specified, the 'click_channel' is used as 'start_channel' corresponding to an auto-correlation measurement. The histogram has a number 'n_bins' of bins of bin width 'binwidth'. Clicks that fall outside the histogram range are discarded. Data accumulation is performed independently for all start tags. This type of measurement is frequently referred to as 'multiple start, multiple stop' measurement and corresponds to a full auto- or cross-correlation measurement.

The data obtained from subsequent start tags can be accumulated into the same histogram (one-dimensional measurement) or into different histograms (two-dimensional measurement). In this way, you can perform more general two-dimensional time-difference measurements. The parameter 'n_histograms' specifies the number of histograms. After each tag on the 'next_channel', the histogram index is incremented by one and reset to zero after reaching the last valid index. The measurement starts with the first tag on the 'next_channel'.

You can also provide a synchronization trigger that resets the histogram index by specifying a 'sync_channel'. The measurement starts when a tag on the 'sync_channel' arrives with a subsequent tag on 'next_channel'. When a rollover occurs, the accumulation is stopped until the next sync and subsequent next signal. A sync signal before a rollover will stop the accumulation, reset the histogram index and a subsequent signal on the 'next_channel' starts the accumulation again.

Typically, you will run the measurement indefinitely until stopped by the user. However, it is also possible to specify the maximum number of rollovers of the histogram index. In this case the measurement stops when the number of rollovers has reached the specified value. This means that for both a one-dimensional and for a two-dimensional measurement, it will measure until the measurement went through the specified number of rollovers / sync tags.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 TimeDifferences()

```
TimeDifferences::TimeDifferences (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t start_channel = CHANNEL_UNUSED,
    channel_t next_channel = CHANNEL_UNUSED,
    channel_t sync_channel = CHANNEL_UNUSED,
    timestamp_t binwidth = 1000,
    int32_t n_bins = 1000,
    int32_t n_histograms = 1 )
```

constructor of a [TimeDifferences](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel that increments the count in a bin
<i>start_channel</i>	channel that sets start times relative to which clicks on the click channel are measured
<i>next_channel</i>	channel that increments the histogram index
<i>sync_channel</i>	channel that resets the histogram index to zero
<i>binwidth</i>	width of one histogram bin in ps
<i>n_bins</i>	number of bins in each histogram
<i>n_histograms</i>	number of histograms

7.43.2.2 ~TimeDifferences()

```
TimeDifferences::~~TimeDifferences ( )
```

7.43.3 Member Function Documentation

7.43.3.1 clear_impl()

```
void TimeDifferences::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.43.3.2 getCounts()

```
uint64_t TimeDifferences::getCounts ( )
```

returns the number of rollovers (histogram index resets)

7.43.3.3 getData()

```
void TimeDifferences::getData (
    std::function< int32_t *(size_t, size_t)> array_out )
```

returns a two-dimensional array of size 'n_bins' by 'n_histograms' containing the histograms

7.43.3.4 getIndex()

```
void TimeDifferences::getIndex (
    std::function< long long *(size_t)> array_out )
```

returns a vector of size 'n_bins' containing the time bins in ps

7.43.3.5 next_impl()

```
bool TimeDifferences::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp\_t begin_time,
    timestamp\_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.43.3.6 on_start()

```
void TimeDifferences::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.43.3.7 ready()

```
bool TimeDifferences::ready ( )
```

returns 'true' when the required number of rollovers set by 'setMaxCounts' has been reached

7.43.3.8 setMaxCounts()

```
void TimeDifferences::setMaxCounts (
    uint64_t max_counts )
```

set the number of rollovers at which the measurement stops integrating

Parameters

<i>max_counts</i>	maximum number of sync/next clicks
-------------------	------------------------------------

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

- [Iterators.h](#)

7.44 TimeDifferencesImpl< T > Class Template Reference

```
#include <Iterators.h>
```

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

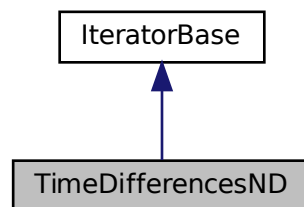
- [Iterators.h](#)

7.45 TimeDifferencesND Class Reference

Accumulates the time differences between clicks on two channels in a multi-dimensional histogram.

```
#include <Iterators.h>
```

Inheritance diagram for TimeDifferencesND:



Public Member Functions

- [TimeDifferencesND](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) click_channel, [channel_t](#) start_channel, [std::vector](#)< [channel_t](#) > next_channels, [std::vector](#)< [channel_t](#) > sync_channels, [std::vector](#)< [int32_t](#) > n_h<→_histograms, [timestamp_t](#) binwidth, [int32_t](#) n_bins)
constructor of a [TimeDifferencesND](#) measurement
- [~TimeDifferencesND](#) ()
- void [getData](#) ([std::function](#)< [int32_t](#) *([size_t](#), [size_t](#))> array_out)
returns a two-dimensional array of size n_bins by all n_histgrams containing the histograms
- void [getIndex](#) ([std::function](#)< [long long](#) *([size_t](#))> array_out)
returns a vector of size n_bins containing the time bins in ps

Protected Member Functions

- bool [next_impl](#) ([std::vector](#)< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) over-
ride
update iterator state
- void [clear_impl](#) () override
clear [Iterator](#) state.
- void [on_start](#) () override
callback when the measurement class is started

7.45.2.1 TimeDifferencesND()

```
TimeDifferencesND::TimeDifferencesND (
    TimeTaggerBase * tagger,
    channel_t click_channel,
    channel_t start_channel,
    std::vector< channel_t > next_channels,
    std::vector< channel_t > sync_channels,
    std::vector< int32_t > n_histograms,
    timestamp_t binwidth,
    int32_t n_bins )
```

constructor of a [TimeDifferencesND](#) measurement

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>click_channel</i>	channel that increments the count in a bin
<i>start_channel</i>	channel that sets start times relative to which clicks on the click channel are measured
<i>next_channels</i>	vector of channels that increments the histogram index
<i>sync_channels</i>	vector of channels that resets the histogram index to zero
<i>n_histograms</i>	vector of numbers of histograms per dimension.
<i>binwidth</i>	width of one histogram bin in ps
<i>n_bins</i>	number of bins in each histogram

7.45.2.2 ~TimeDifferencesND()

```
TimeDifferencesND::~TimeDifferencesND ( )
```

7.45.3 Member Function Documentation

7.45.3.1 clear_impl()

```
void TimeDifferencesND::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.45.3.2 getData()

```
void TimeDifferencesND::getData (
    std::function< int32_t *(size_t, size_t)> array_out )
```

returns a two-dimensional array of size `n_bins` by all `n_histograms` containing the histograms

7.45.3.3 getIndex()

```
void TimeDifferencesND::getIndex (
    std::function< long long *(size_t)> array_out )
```

returns a vector of size `n_bins` containing the time bins in ps

7.45.3.4 next_impl()

```
bool TimeDifferencesND::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the `next_impl()` method. The `next_impl()` function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.45.3.5 on_start()

```
void TimeDifferencesND::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

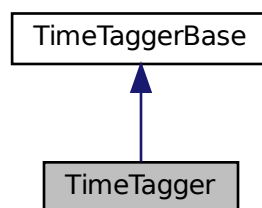
- [Iterators.h](#)

7.46 TimeTagger Class Reference

backend for the [TimeTagger](#).

```
#include <TimeTagger.h>
```

Inheritance diagram for TimeTagger:



Public Member Functions

- virtual void [reset](#) ()=0
reset the [TimeTagger](#) object to default settings and detach all iterators
- virtual bool [isChannelRegistered](#) ([channel_t](#) chan)=0
- virtual void [setTestSignalDivider](#) (int divider)=0
set the divider for the frequency of the test signal
- virtual int [getTestSignalDivider](#) ()=0
get the divider for the frequency of the test signal
- virtual void [setTriggerLevel](#) ([channel_t](#) channel, double voltage)=0
set the trigger voltage threshold of a channel
- virtual double [getTriggerLevel](#) ([channel_t](#) channel)=0
get the trigger voltage threshold of a channel
- virtual [timestamp_t](#) [getHardwareDelayCompensation](#) ([channel_t](#) channel)=0
get hardware delay compensation of a channel
- virtual void [setInputMux](#) ([channel_t](#) channel, int mux_mode)=0
configures the input multiplexer
- virtual int [getInputMux](#) ([channel_t](#) channel)=0

- fetches the configuration of the input multiplexer*
- virtual void [setConditionalFilter](#) (std::vector< [channel_t](#) > trigger, std::vector< [channel_t](#) > filtered, bool hardwareDelayCompensation=true)=0
- configures the conditional filter*
- virtual void [clearConditionalFilter](#) ()=0
- deactivates the conditional filter*
- virtual std::vector< [channel_t](#) > [getConditionalFilterTrigger](#) ()=0
- fetches the configuration of the conditional filter*
- virtual std::vector< [channel_t](#) > [getConditionalFilterFiltered](#) ()=0
- fetches the configuration of the conditional filter*
- virtual void [setNormalization](#) (std::vector< [channel_t](#) > channel, bool state)=0
- enables or disables the normalization of the distribution.*
- virtual bool [getNormalization](#) ([channel_t](#) channel)=0
- returns the the normalization of the distribution.*
- virtual void [setHardwareBufferSize](#) (int size)=0
- sets the maximum USB buffer size*
- virtual int [getHardwareBufferSize](#) ()=0
- queries the size of the USB queue*
- virtual void [setStreamBlockSize](#) (int max_events, int max_latency)=0
- sets the maximum events and latency for the stream block size*
- virtual int [getStreamBlockSizeEvents](#) ()=0
- virtual int [getStreamBlockSizeLatency](#) ()=0
- virtual void [setEventDivider](#) ([channel_t](#) channel, unsigned int divider)=0
- Divides the amount of transmitted edge per channel.*
- virtual unsigned int [getEventDivider](#) ([channel_t](#) channel)=0
- Returns the factor of the dividing filter.*
- virtual void [autoCalibration](#) (std::function< double *(size_t)> array_out)=0
- runs a calibrations based on the on-chip uncorrelated signal generator.*
- virtual std::string [getSerial](#) ()=0
- identifies the hardware by serial number*
- virtual std::string [getModel](#) ()=0
- identifies the hardware by Time Tagger Model*
- virtual int [getChannelNumberScheme](#) ()=0
- Fetch the configured numbering scheme for this [TimeTagger](#) object.*
- virtual std::vector< double > [getDACRange](#) ()=0
- returns the minimum and the maximum voltage of the DACs as a trigger reference*
- virtual void [getDistributionCount](#) (std::function< uint64_t *(size_t, size_t)> array_out)=0
- get internal calibration data*
- virtual void [getDistributionPSecs](#) (std::function< long long *(size_t, size_t)> array_out)=0
- get internal calibration data*
- virtual std::vector< [channel_t](#) > [getChannelList](#) ([ChannelEdge](#) type=[ChannelEdge::All](#))=0
- fetch a vector of all physical input channel ids*
- virtual [timestamp_t](#) [getPsPerClock](#) ()=0
- fetch the duration of each clock cycle in picoseconds*
- virtual std::string [getPcbVersion](#) ()=0
- Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.*
- virtual std::string [getFirmwareVersion](#) ()=0
- Return an unique identifier for the applied firmware.*
- virtual std::string [getSensorData](#) ()=0
- Show the status of the sensor data from the FPGA and peripherals on the console.*

- virtual void [setLED](#) (uint32_t bitmask)=0
Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.
- virtual std::string [getLicenseInfo](#) ()=0
Fetches and parses the current installed license on this device.
- virtual uint32_t [factoryAccess](#) (uint32_t pw, uint32_t addr, uint32_t data, uint32_t mask)=0
Direct read/write access to WireIn/WireOuts in FPGA (mask==0 for readonly)
- virtual void [setSoundFrequency](#) (uint32_t freq_hz)=0
Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)
- virtual void [startServer](#) ([AccessMode](#) access_mode, std::vector< [channel_t](#) > channels=std::vector< [channel_t](#) >(), uint32_t port=41101)=0
starts the Time Tagger server that will stream the time tags to the client.
- virtual bool [isServerRunning](#) ()=0
check if the server is still running.
- virtual void [stopServer](#) ()=0
stops the time tagger server if currently running, otherwise does nothing.

Additional Inherited Members

7.46.1 Detailed Description

backend for the [TimeTagger](#).

The [TimeTagger](#) class connects to the hardware, and handles the communication over the usb. There may be only one instance of the backend per physical device.

7.46.2 Member Function Documentation

7.46.2.1 [autoCalibration\(\)](#)

```
virtual void TimeTagger::autoCalibration (
    std::function< double *(size_t)> array_out ) [pure virtual]
```

runs a calibrations based on the on-chip uncorrelated signal generator.

7.46.2.2 [clearConditionalFilter\(\)](#)

```
virtual void TimeTagger::clearConditionalFilter ( ) [pure virtual]
```

deactivates the conditional filter

equivalent to [setConditionalFilter](#)({},{})

7.46.2.3 factoryAccess()

```
virtual uint32_t TimeTagger::factoryAccess (
    uint32_t pw,
    uint32_t addr,
    uint32_t data,
    uint32_t mask ) [pure virtual]
```

Direct read/write access to WireIn/WireOuts in FPGA (mask==0 for readonly)

DO NOT USE. Only for internal debug purposes.

7.46.2.4 getChannelList()

```
virtual std::vector<channel_t> TimeTagger::getChannelList (
    ChannelEdge type = ChannelEdge::All ) [pure virtual]
```

fetch a vector of all physical input channel ids

The function returns the channel of all rising and falling edges. For example for the Time Tagger 20 (8 input channels) TT_CHANNEL_NUMBER_SCHEME_ZERO: {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} and for TT_CHANNEL_NUMBER_SCHEME_ONE: {-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8}

TT_CHANNEL_RISING_EDGES returns only the rising edges SCHEME_ONE: {1,2,3,4,5,6,7,8} and TT_CHANNEL_FALLING_EDGES return only the falling edges SCHEME_ONE: {-1,-2,-3,-4,-5,-6,-7,-8} which are the inverted Channels of the rising edges.

7.46.2.5 getChannelNumberScheme()

```
virtual int TimeTagger::getChannelNumberScheme ( ) [pure virtual]
```

Fetch the configured numbering scheme for this [TimeTagger](#) object.

Please see [setTimeTaggerChannelNumberScheme\(\)](#) for details.

7.46.2.6 getConditionalFilterFiltered()

```
virtual std::vector<channel_t> TimeTagger::getConditionalFilterFiltered ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see [setConditionalFilter](#)

7.46.2.7 getConditionalFilterTrigger()

```
virtual std::vector<channel_t> TimeTagger::getConditionalFilterTrigger ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see [setConditionalFilter](#)

7.46.2.8 getDACRange()

```
virtual std::vector<double> TimeTagger::getDACRange ( ) [pure virtual]
```

returns the minimum and the maximum voltage of the DACs as a trigger reference

7.46.2.9 getDistributionCount()

```
virtual void TimeTagger::getDistributionCount (
    std::function< uint64_t *(size_t, size_t)> array_out ) [pure virtual]
```

get internal calibration data

7.46.2.10 getDistributionPSEcs()

```
virtual void TimeTagger::getDistributionPSEcs (
    std::function< long long *(size_t, size_t)> array_out ) [pure virtual]
```

get internal calibration data

7.46.2.11 getEventDivider()

```
virtual unsigned int TimeTagger::getEventDivider (
    channel_t channel ) [pure virtual]
```

Returns the factor of the dividing filter.

See setEventDivider for further details.

Parameters

<i>channel</i>	channel to be queried
----------------	-----------------------

Returns

the configured divider

7.46.2.12 getFirmwareVersion()

```
virtual std::string TimeTagger::getFirmwareVersion ( ) [pure virtual]
```

Return an unique identifier for the applied firmware.

This function returns a comma separated list of the firmware version with

- the device identifier: TT-20 or TT-Ultra
- the firmware identifier: FW 3
- optional the timestamp of the assembling of the firmware
- the firmware identifier of the USB chip: OK 1.30 eg "TT-Ultra, FW 3, TS 2018-11-13 22:57:32, OK 1.30"

7.46.2.13 getHardwareBufferSize()

```
virtual int TimeTagger::getHardwareBufferSize ( ) [pure virtual]
```

queries the size of the USB queue

See setHardwareBufferSize for more information.

Returns

the actual size of the USB queue in events

7.46.2.14 getHardwareDelayCompensation()

```
virtual timestamp_t TimeTagger::getHardwareDelayCompensation (
    channel_t channel ) [pure virtual]
```

get hardware delay compensation of a channel

The physical input delays are calibrated and compensated. However this compensation is implemented after the conditional filter and so affects its result. This function queries the effective input delay, which compensates the hardware delay.

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the hardware delay compensation in picoseconds

7.46.2.15 getInputMux()

```
virtual int TimeTagger::getInputMux (
    channel_t channel ) [pure virtual]
```

fetches the configuration of the input multiplexer

Parameters

<i>channel</i>	the physical channel of the input multiplexer
----------------	---

Returns

the configuration mode of the input multiplexer

7.46.2.16 getLicenseInfo()

```
virtual std::string TimeTagger::getLicenseInfo ( ) [pure virtual]
```

Fetches and parses the current installed license on this device.

Returns

a human readable string containing all information about the license on this device

7.46.2.17 getModel()

```
virtual std::string TimeTagger::getModel ( ) [pure virtual]
```

identifies the hardware by Time Tagger Model

7.46.2.18 getNormalization()

```
virtual bool TimeTagger::getNormalization (
    channel_t channel ) [pure virtual]
```

returns the the normalization of the distribution.

Refer the Manual for a description of this function.

Parameters

<i>channel</i>	the channel to query
----------------	----------------------

Returns

if the normalization is enabled

7.46.2.19 getPcbVersion()

```
virtual std::string TimeTagger::getPcbVersion ( ) [pure virtual]
```

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version ≥ 1 is channel configuration ONE.

7.46.2.20 getPsPerClock()

```
virtual timestamp_t TimeTagger::getPsPerClock ( ) [pure virtual]
```

fetch the duration of each clock cycle in picoseconds

7.46.2.21 getSensorData()

```
virtual std::string TimeTagger::getSensorData ( ) [pure virtual]
```

Show the status of the sensor data from the FPGA and peripherals on the console.

7.46.2.22 getSerial()

```
virtual std::string TimeTagger::getSerial ( ) [pure virtual]
```

identifies the hardware by serial number

7.46.2.23 getStreamBlockSizeEvents()

```
virtual int TimeTagger::getStreamBlockSizeEvents ( ) [pure virtual]
```

7.46.2.24 getStreamBlockSizeLatency()

```
virtual int TimeTagger::getStreamBlockSizeLatency ( ) [pure virtual]
```

7.46.2.25 `getTestSignalDivider()`

```
virtual int TimeTagger::getTestSignalDivider ( ) [pure virtual]
```

get the divider for the frequency of the test signal

7.46.2.26 `getTriggerLevel()`

```
virtual double TimeTagger::getTriggerLevel (
    channel_t channel ) [pure virtual]
```

get the trigger voltage threshold of a channel

Parameters

<i>channel</i>	the channel
----------------	-------------

7.46.2.27 `isChannelRegistered()`

```
virtual bool TimeTagger::isChannelRegistered (
    channel_t chan ) [pure virtual]
```

7.46.2.28 `isServerRunning()`

```
virtual bool TimeTagger::isServerRunning ( ) [pure virtual]
```

check if the server is still running.

Returns

returns true if running; false, if not running

7.46.2.29 `reset()`

```
virtual void TimeTagger::reset ( ) [pure virtual]
```

reset the [TimeTagger](#) object to default settings and detach all iterators

7.46.2.30 setConditionalFilter()

```
virtual void TimeTagger::setConditionalFilter (
    std::vector< channel_t > trigger,
    std::vector< channel_t > filtered,
    bool hardwareDelayCompensation = true ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

Parameters

<i>trigger</i>	the channels that sets the condition
<i>filtered</i>	the channels that are filtered by the condition
<i>hardwareDelayCompensation</i>	if false, the physical hardware delay will not be compensated

7.46.2.31 setEventDivider()

```
virtual void TimeTagger::setEventDivider (
    channel_t channel,
    unsigned int divider ) [pure virtual]
```

Divides the amount of transmitted edge per channel.

This filter decimates the events on a given channel by a specified factor. So for a divider n , every n th event is transmitted through the filter and $n-1$ events are skipped between consecutive transmitted events. If a conditional filter is also active, the event divider is applied after the conditional filter, so the conditional is applied to the complete event stream and only events which pass the conditional filter are forwarded to the divider.

As it is a hardware filter, it reduces the required USB bandwidth and CPU processing power, but it cannot be configured for virtual channels.

Parameters

<i>channel</i>	channel to be configured
<i>divider</i>	new divider, must be smaller than 65536

7.46.2.32 setHardwareBufferSize()

```
virtual void TimeTagger::setHardwareBufferSize (
    int size ) [pure virtual]
```

sets the maximum USB buffer size

This option controls the maximum buffer size of the USB connection. This can be used to balance low input latency vs high (peak) throughput.

Parameters

<i>size</i>	the maximum buffer size in events
-------------	-----------------------------------

7.46.2.33 setInputMux()

```
virtual void TimeTagger::setInputMux (
    channel_t channel,
    int mux_mode ) [pure virtual]
```

configures the input multiplexer

Every physical input channel has an input multiplexer with 4 modes: 0: normal input mode 1: use the input from channel -1 (left) 2: use the input from channel +1 (right) 3: use the reference oscillator

Mode 1 and 2 cascades, so many inputs can be configured to get the same input events.

Parameters

<i>channel</i>	the physical channel of the input multiplexer
<i>mux_mode</i>	the configuration mode of the input multiplexer

7.46.2.34 setLED()

```
virtual void TimeTagger::setLED (
    uint32_t bitmask ) [pure virtual]
```

Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.

7.46.2.35 setNormalization()

```
virtual void TimeTagger::setNormalization (
    std::vector< channel_t > channel,
    bool state ) [pure virtual]
```

enables or disables the normalization of the distribution.

Refer the Manual for a description of this function.

Parameters

<i>channel</i>	list of channels to modify
<i>state</i>	the new state

7.46.2.36 setSoundFrequency()

```
virtual void TimeTagger::setSoundFrequency (
    uint32_t freq_hz ) [pure virtual]
```

Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)

Parameters

<i>freq_hz</i>	the generated audio frequency
----------------	-------------------------------

7.46.2.37 setStreamBlockSize()

```
virtual void TimeTagger::setStreamBlockSize (
    int max_events,
    int max_latency ) [pure virtual]
```

sets the maximum events and latency for the stream block size

This option controls the latency and the block size of the data stream. The default values are max_events = 131072 events and max_latency = 20 ms. Depending on which of the two parameters is exceeded first, the block stream size is adjusted accordingly. The block size will be reduced automatically for blocks when no signal is arriving for 512 ns on the Time Tagger Ultra and 1536 ns for the Time Tagger 20. *

Parameters

<i>max_events</i>	maximum number of events
<i>max_latency</i>	maximum latency in ms

7.46.2.38 setTestSignalDivider()

```
virtual void TimeTagger::setTestSignalDivider (
    int divider ) [pure virtual]
```

set the divider for the frequency of the test signal

The base clock of the test signal oscillator for the Time Tagger Ultra is running at 100.8 MHz sampled down by an factor of 2 to have a similar base clock as the Time Tagger 20 (~50 MHz). The default divider is 63 -> ~800 kEvents/s

Parameters

<i>divider</i>	frequency divisor of the oscillator
----------------	-------------------------------------

7.46.2.39 setTriggerLevel()

```
virtual void TimeTagger::setTriggerLevel (
    channel_t channel,
    double voltage ) [pure virtual]
```

set the trigger voltage threshold of a channel

Parameters

<i>channel</i>	the channel to set
<i>voltage</i>	voltage level.. [0..1]

7.46.2.40 startServer()

```
virtual void TimeTagger::startServer (
    AccessMode access_mode,
    std::vector< channel_t > channels = std::vector< channel_t >(),
    uint32_t port = 41101 ) [pure virtual]
```

starts the Time Tagger server that will stream the time tags to the client.

Parameters

<i>access_mode</i>	set the type of access a user can have.
<i>port</i>	port at which this time tagger server will be listening on.
<i>channels</i>	channels to be streamed, if empty, all the channels will be exposed.

7.46.2.41 stopServer()

```
virtual void TimeTagger::stopServer ( ) [pure virtual]
```

stops the time tagger server if currently running, otherwise does nothing.

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

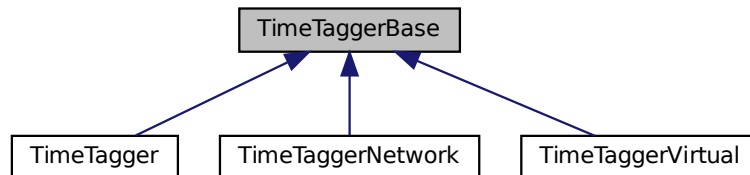
- [TimeTagger.h](#)

7.47 TimeTaggerBase Class Reference

Basis interface for all Time Tagger classes.

```
#include <TimeTagger.h>
```

Inheritance diagram for TimeTaggerBase:



Public Types

- typedef std::function< void([IteratorBase](#) *)> [IteratorCallback](#)
- typedef std::map< [IteratorBase](#) *, [IteratorCallback](#) > [IteratorCallbackMap](#)

Public Member Functions

- virtual unsigned int [getFence](#) (bool alloc_fence=true)=0
Generate a new fence object, which validates the current configuration and the current time.
- virtual bool [waitForFence](#) (unsigned int fence, int64_t timeout=-1)=0
Wait for a fence in the data stream.
- virtual bool [sync](#) (int64_t timeout=-1)=0
Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.
- virtual [channel_t](#) [getInvertedChannel](#) ([channel_t](#) channel)=0
get the falling channel id for a raising channel and vice versa
- virtual bool [isUnusedChannel](#) ([channel_t](#) channel)=0
compares the provided channel with CHANNEL_UNUSED
- virtual void [runSynchronized](#) (const [IteratorCallbackMap](#) &callbacks, bool block=true)=0
Run synchronized callbacks for a list of iterators.
- virtual std::string [getConfiguration](#) ()=0
Fetches the overall configuration status of the Time Tagger object.
- virtual void [setInputDelay](#) ([channel_t](#) channel, [timestamp_t](#) delay)=0
set time delay on a channel
- virtual void [setDelayHardware](#) ([channel_t](#) channel, [timestamp_t](#) delay)=0
set time delay on a channel
- virtual void [setDelaySoftware](#) ([channel_t](#) channel, [timestamp_t](#) delay)=0
set time delay on a channel
- virtual [timestamp_t](#) [getInputDelay](#) ([channel_t](#) channel)=0
get time delay of a channel
- virtual [timestamp_t](#) [getDelaySoftware](#) ([channel_t](#) channel)=0

- get time delay of a channel*
- virtual [timestamp_t getDelayHardware](#) ([channel_t](#) channel)=0
- get time delay of a channel*
- virtual [timestamp_t setDeadtime](#) ([channel_t](#) channel, [timestamp_t](#) deadtime)=0
- set the deadtime between two edges on the same channel.*
- virtual [timestamp_t getDeadtime](#) ([channel_t](#) channel)=0
- get the deadtime between two edges on the same channel.*
- virtual void [setTestSignal](#) ([channel_t](#) channel, bool enabled)=0
- enable/disable internal test signal on a channel.*
- virtual void [setTestSignal](#) (std::vector< [channel_t](#) > channel, bool enabled)=0
- enable/disable internal test signal on multiple channels.*
- virtual bool [getTestSignal](#) ([channel_t](#) channel)=0
- fetch the status of the test signal generator*
- virtual void [setSoftwareClock](#) ([channel_t](#) input_channel, double input_frequency=10e6, double averaging_↔ periods=1000, bool wait_until_locked=true)=0
- enables a software PLL to lock the time to an external clock*
- virtual void [disableSoftwareClock](#) ()=0
- disabled the software PLL*
- virtual [SoftwareClockState](#) [getSoftwareClockState](#) ()=0
- queries all state information of the software clock*
- virtual long long [getOverflows](#) ()=0
- get overflow count*
- virtual void [clearOverflows](#) ()=0
- clear overflow counter*
- virtual long long [getOverflowsAndClear](#) ()=0
- get and clear overflow counter*

Protected Member Functions

- [TimeTaggerBase](#) ()
- abstract interface class*
- virtual [~TimeTaggerBase](#) ()
- destructor*
- [TimeTaggerBase](#) (const [TimeTaggerBase](#) &)=delete
- [TimeTaggerBase](#) & operator= (const [TimeTaggerBase](#) &)=delete
- virtual std::shared_ptr< [IteratorBaseListNode](#) > [addIterator](#) ([IteratorBase](#) *it)=0
- virtual void [freeIterator](#) ([IteratorBase](#) *it)=0
- virtual [channel_t](#) [getNewVirtualChannel](#) ()=0
- virtual void [freeVirtualChannel](#) ([channel_t](#) channel)=0
- virtual void [registerChannel](#) ([channel_t](#) channel)=0
- register a FPGA channel.*
- virtual void [unregisterChannel](#) ([channel_t](#) channel)=0
- release a previously registered channel.*
- virtual void [addChild](#) ([TimeTaggerBase](#) *child)=0
- virtual void [removeChild](#) ([TimeTaggerBase](#) *child)=0
- virtual void [release](#) ()=0

7.47.1 Detailed Description

Basis interface for all Time Tagger classes.

This basis interface represents all common methods to add, remove, and run measurements.

7.47.2 Member Typedef Documentation

7.47.2.1 IteratorCallback

```
typedef std::function<void(IteratorBase *)> TimeTaggerBase::IteratorCallback
```

7.47.2.2 IteratorCallbackMap

```
typedef std::map<IteratorBase *, IteratorCallback> TimeTaggerBase::IteratorCallbackMap
```

7.47.3 Constructor & Destructor Documentation

7.47.3.1 TimeTaggerBase() [1/2]

```
TimeTaggerBase::TimeTaggerBase ( ) [inline], [protected]
```

abstract interface class

7.47.3.2 ~TimeTaggerBase()

```
virtual TimeTaggerBase::~~TimeTaggerBase ( ) [inline], [protected], [virtual]
```

destructor

7.47.3.3 TimeTaggerBase() [2/2]

```
TimeTaggerBase::TimeTaggerBase (
    const TimeTaggerBase & ) [protected], [delete]
```

7.47.4 Member Function Documentation

7.47.4.1 addChild()

```
virtual void TimeTaggerBase::addChild (
    TimeTaggerBase * child ) [protected], [pure virtual]
```

7.47.4.2 addIterator()

```
virtual std::shared_ptr<IteratorBaseListNode> TimeTaggerBase::addIterator (
    IteratorBase * it ) [protected], [pure virtual]
```

7.47.4.3 clearOverflows()

```
virtual void TimeTaggerBase::clearOverflows ( ) [pure virtual]
```

clear overflow counter

Sets the overflow counter to zero

7.47.4.4 disableSoftwareClock()

```
virtual void TimeTaggerBase::disableSoftwareClock ( ) [pure virtual]
```

disabled the software PLL

See setSoftwareClock for further details.

7.47.4.5 freeIterator()

```
virtual void TimeTaggerBase::freeIterator (
    IteratorBase * it ) [protected], [pure virtual]
```

7.47.4.6 freeVirtualChannel()

```
virtual void TimeTaggerBase::freeVirtualChannel (
    channel_t channel ) [protected], [pure virtual]
```


7.47.4.7 getConfiguration()

```
virtual std::string TimeTaggerBase::getConfiguration ( ) [pure virtual]
```

Fetches the overall configuration status of the Time Tagger object.

Returns

a JSON serialized string with all configuration and status flags.

7.47.4.8 getDeadtime()

```
virtual timestamp_t TimeTaggerBase::getDeadtime (
    channel_t channel ) [pure virtual]
```

get the deadtime between two edges on the same channel.

This function gets the user configurable deadtime.

Parameters

<i>channel</i>	channel to be queried
----------------	-----------------------

Returns

the real configured deadtime in picoseconds

7.47.4.9 getDelayHardware()

```
virtual timestamp_t TimeTaggerBase::getDelayHardware (
    channel_t channel ) [pure virtual]
```

get time delay of a channel

see setDelayHardware

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the hardware delay in picoseconds

7.47.4.10 getDelaySoftware()

```
virtual timestamp_t TimeTaggerBase::getDelaySoftware (
    channel_t channel ) [pure virtual]
```

get time delay of a channel

see setDelaySoftware

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the software delay in picoseconds

7.47.4.11 getFence()

```
virtual unsigned int TimeTaggerBase::getFence (
    bool alloc_fence = true ) [pure virtual]
```

Generate a new fence object, which validates the current configuration and the current time.

This fence is uploaded to the earliest pipeline stage of the Time Tagger. Waiting on this fence ensures that all hardware settings such as trigger levels, channel registrations, etc., have propagated to the FPGA and are physically active. Synchronizes the Time Tagger internal memory, so that all tags arriving after the waitForFence call were actually produced after the getFence call. The waitForFence function waits until all tags, which are present at the time of the function call within the internal memory of the Time Tagger, are processed. This call might block to limit the amount of active fences.

Parameters

<i>alloc_fence</i>	if false, a reference to the most recently created fence will be returned instead
--------------------	---

Returns

the allocated fence

7.47.4.12 getInputDelay()

```
virtual timestamp_t TimeTaggerBase::getInputDelay (
    channel_t channel ) [pure virtual]
```

get time delay of a channel

see setInputDelay

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the software delay in picoseconds

7.47.4.13 getInvertedChannel()

```
virtual channel_t TimeTaggerBase::getInvertedChannel (
    channel_t channel ) [pure virtual]
```

get the falling channel id for a raising channel and vice versa

If this channel has no inverted channel, UNUSED_CHANNEL is returned. This is the case for most virtual channels.

Parameters

<i>channel</i>	The channel id to query
----------------	-------------------------

Returns

the inverted channel id

7.47.4.14 getNewVirtualChannel()

```
virtual channel_t TimeTaggerBase::getNewVirtualChannel ( ) [protected], [pure virtual]
```

7.47.4.15 getOverflows()

```
virtual long long TimeTaggerBase::getOverflows ( ) [pure virtual]
```

get overflow count

Get the number of communication overflows occurred

7.47.4.16 getOverflowsAndClear()

```
virtual long long TimeTaggerBase::getOverflowsAndClear ( ) [pure virtual]
```

get and clear overflow counter

Get the number of communication overflows occurred and sets them to zero

7.47.4.17 getSoftwareClockState()

```
virtual SoftwareClockState TimeTaggerBase::getSoftwareClockState ( ) [pure virtual]
```

queries all state information of the software clock

See setSoftwareClock for further details.

7.47.4.18 getTestSignal()

```
virtual bool TimeTaggerBase::getTestSignal (
    channel_t channel ) [pure virtual]
```

fetch the status of the test signal generator

Parameters

<i>channel</i>	the channel
----------------	-------------

Implemented in [TimeTaggerNetwork](#).

7.47.4.19 isUnusedChannel()

```
virtual bool TimeTaggerBase::isUnusedChannel (
    channel_t channel ) [pure virtual]
```

compares the provided channel with CHANNEL_UNUSED

But also keeps care about the channel number scheme and selects either CHANNEL_UNUSED or CHANNEL_↔UNUSED_OLD

7.47.4.20 operator=()

```
TimeTaggerBase& TimeTaggerBase::operator= (
    const TimeTaggerBase & ) [protected], [delete]
```

7.47.4.21 registerChannel()

```
virtual void TimeTaggerBase::registerChannel (
    channel_t channel ) [protected], [pure virtual]
```

register a FPGA channel.

Only events on previously registered channels will be transferred over the communication channel.

Parameters

<i>channel</i>	the channel
----------------	-------------

7.47.4.22 release()

```
virtual void TimeTaggerBase::release ( ) [protected], [pure virtual]
```

7.47.4.23 removeChild()

```
virtual void TimeTaggerBase::removeChild (
    TimeTaggerBase * child ) [protected], [pure virtual]
```

7.47.4.24 runSynchronized()

```
virtual void TimeTaggerBase::runSynchronized (
    const IteratorCallbackMap & callbacks,
    bool block = true ) [pure virtual]
```

Run synchronized callbacks for a list of iterators.

This method has a list of callbacks for a list of iterators. Those callbacks are called for a synchronized data set, but in parallel. They are called from an internal worker thread. As the data set is synchronized, this creates a bottleneck for one worker thread, so only fast and non-blocking callbacks are allowed.

Parameters

<i>callbacks</i>	Map of callbacks per iterator
<i>block</i>	Shall this method block until all callbacks are finished

7.47.4.25 setDeadtime()

```
virtual timestamp_t TimeTaggerBase::setDeadtime (
    channel_t channel,
    timestamp_t deadtime ) [pure virtual]
```

set the deadtime between two edges on the same channel.

This function sets the user configurable deadtime. The requested time will be rounded to the nearest multiple of the clock time. The deadtime will also be clamped to device specific limitations.

As the actual deadtime will be altered, the real value will be returned.

Parameters

<i>channel</i>	channel to be configured
<i>deadtime</i>	new deadtime in picoseconds

Returns

the real configured deadtime in picoseconds

7.47.4.26 setDelayHardware()

```
virtual void TimeTaggerBase::setDelayHardware (
    channel_t channel,
    timestamp_t delay ) [pure virtual]
```

set time delay on a channel

When set, every event on this physical input channel is delayed by the given delay in picoseconds. This delay is implemented on the hardware before any filter with no performance overhead. The maximum delay on the Time Tagger Ultra series is 2 us. This affects both the rising and the falling event at the same time.

Parameters

<i>channel</i>	the channel to set
<i>delay</i>	the hardware delay in picoseconds

7.47.4.27 setDelaySoftware()

```
virtual void TimeTaggerBase::setDelaySoftware (
    channel_t channel,
    timestamp_t delay ) [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds. This happens on the computer and so after the on-device filters. Please use setDelayHardware instead for better performance. This affects either the rising or the falling event only.

This method has the best performance with "small delays". The delay is considered "small" when less than 100 events arrive within the time of the largest delay set. For example, if the total event-rate over all channels used is 10 Mevent/s, the signal can be delayed efficiently up to 10 microseconds. For large delays, please use [DelayedChannel](#) instead.

Parameters

<i>channel</i>	the channel to set
<i>delay</i>	the software delay in picoseconds

7.47.4.28 setInputDelay()

```
virtual void TimeTaggerBase::setInputDelay (
    channel_t channel,
    timestamp_t delay ) [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds.

This method has the best performance with "small delays". The delay is considered "small" when less than 100 events arrive within the time of the largest delay set. For example, if the total event-rate over all channels used is 10 Mevent/s, the signal can be delayed efficiently up to 10 microseconds. For large delays, please use [DelayedChannel](#) instead.

Parameters

<i>channel</i>	the channel to set
<i>delay</i>	the delay in picoseconds

7.47.4.29 setSoftwareClock()

```
virtual void TimeTaggerBase::setSoftwareClock (
    channel_t input_channel,
    double input_frequency = 10e6,
    double averaging_periods = 1000,
    bool wait_until_locked = true ) [pure virtual]
```

enables a software PLL to lock the time to an external clock

This feature implements a software PLL on the CPU. This can replace external clocks with no restrictions on correlated data to other inputs. It uses a first-order loop filter to ignore the discretization noise of the input and to provide some kind of cutoff frequency when to apply the extern clock.

Note

Within the first $100 * \text{averaging_factor} * \text{clock_period}$, a frequency locking approach is applied. The phase gets locked afterwards.

Parameters

<i>input_channel</i>	The physical input channel
<i>input_frequency</i>	Frequency of the configured external clock. Slight variations will be canceled out. Defaults to 10e6 for 10 MHz
<i>averaging_periods</i>	Times clock_period is the cutoff period for the filter. Shorter periods are evaluated with the Time Tagger's internal clock, longer periods are evaluated with the here configured external clock
<i>wait_until_locked</i>	Blocks the execution until the software clock is locked. Throws an exception on locking errors. All locking log messages are filtered while this call is executed.

7.47.4.30 setTestSignal() [1/2]

```
virtual void TimeTaggerBase::setTestSignal (
    channel_t channel,
    bool enabled ) [pure virtual]
```

enable/disable internal test signal on a channel.

This will connect or disconnect the channel with the on-chip uncorrelated signal generator.

Parameters

<i>channel</i>	the channel
<i>enabled</i>	enabled / disabled flag

7.47.4.31 setTestSignal() [2/2]

```
virtual void TimeTaggerBase::setTestSignal (
    std::vector< channel_t > channel,
    bool enabled ) [pure virtual]
```

enable/disable internal test signal on multiple channels.

This will connect or disconnect the channels with the on-chip uncorrelated signal generator.

Parameters

<i>channel</i>	a vector of channels
<i>enabled</i>	enabled / disabled flag

7.47.4.32 sync()

```
virtual bool TimeTaggerBase::sync (
    int64_t timeout = -1 ) [pure virtual]
```

Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.

This is a shortcut for calling getFence and waitForFence at once. See getFence for more details.

Parameters

<i>timeout</i>	timeout in milliseconds. Negative means no timeout, zero returns immediately.
----------------	---

Returns

true on success, false on timeout

7.47.4.33 unregisterChannel()

```
virtual void TimeTaggerBase::unregisterChannel (
    channel_t channel ) [protected], [pure virtual]
```

release a previously registered channel.

Parameters

<i>channel</i>	the channel
----------------	-------------

7.47.4.34 waitForFence()

```
virtual bool TimeTaggerBase::waitForFence (
    unsigned int fence,
    int64_t timeout = -1 ) [pure virtual]
```

Wait for a fence in the data stream.

See `getFence` for more details.

Parameters

<i>fence</i>	fence object, which shall be waited on
<i>timeout</i>	timeout in milliseconds. Negative means no timeout, zero returns immediately.

Returns

true if the fence has passed, false on timeout

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

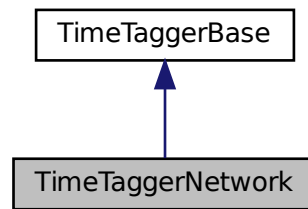
- [TimeTagger.h](#)

7.48 TimeTaggerNetwork Class Reference

network [TimeTagger](#) client.

```
#include <TimeTagger.h>
```

Inheritance diagram for TimeTaggerNetwork:



Public Member Functions

- virtual bool `isConnected` ()=0
check if the network time tagger is currently connected to a server
- virtual void `setTriggerLevel` (`channel_t` channel, double voltage)=0
set the trigger voltage threshold of a channel
- virtual double `getTriggerLevel` (`channel_t` channel)=0
get the trigger voltage threshold of a channel
- virtual void `setConditionalFilter` (std::vector< `channel_t` > trigger, std::vector< `channel_t` > filtered, bool hardwareDelayCompensation=true)=0
configures the conditional filter
- virtual void `clearConditionalFilter` ()=0
deactivates the conditional filter
- virtual std::vector< `channel_t` > `getConditionalFilterTrigger` ()=0
fetches the configuration of the conditional filter
- virtual std::vector< `channel_t` > `getConditionalFilterFiltered` ()=0
fetches the configuration of the conditional filter
- virtual void `setTestSignalDivider` (int divider)=0
set the divider for the frequency of the test signal
- virtual int `getTestSignalDivider` ()=0
get the divider for the frequency of the test signal
- virtual bool `getTestSignal` (`channel_t` channel)=0
fetch the status of the test signal generator
- virtual void `setDelayClient` (`channel_t` channel, `timestamp_t` time)=0
set time delay on a channel
- virtual `timestamp_t` `getDelayClient` (`channel_t` channel)=0
get the time delay of a channel
- virtual `timestamp_t` `getHardwareDelayCompensation` (`channel_t` channel)=0
get hardware delay compensation of a channel
- virtual void `setNormalization` (std::vector< `channel_t` > channel, bool state)=0
enables or disables the normalization of the distribution.
- virtual bool `getNormalization` (`channel_t` channel)=0
returns the the normalization of the distribution.
- virtual void `setHardwareBufferSize` (int size)=0
sets the maximum USB buffer size

- virtual int [getHardwareBufferSize](#) ()=0
queries the size of the USB queue
- virtual void [setStreamBlockSize](#) (int max_events, int max_latency)=0
sets the maximum events and latency for the stream block size
- virtual int [getStreamBlockSizeEvents](#) ()=0
- virtual int [getStreamBlockSizeLatency](#) ()=0
- virtual void [setEventDivider](#) (channel_t channel, unsigned int divider)=0
Divides the amount of transmitted edge per channel.
- virtual unsigned int [getEventDivider](#) (channel_t channel)=0
Returns the factor of the dividing filter.
- virtual std::string [getSerial](#) ()=0
identifies the hardware by serial number
- virtual std::string [getModel](#) ()=0
identifies the hardware by Time Tagger Model
- virtual int [getChannelNumberScheme](#) ()=0
Fetch the configured numbering scheme for this [TimeTagger](#) object.
- virtual std::vector< double > [getDACRange](#) ()=0
returns the minimum and the maximum voltage of the DACs as a trigger reference
- virtual std::vector< channel_t > [getChannelList](#) (ChannelEdge type=ChannelEdge::All)=0
fetch a vector of all physical input channel ids
- virtual timestamp_t [getPsPerClock](#) ()=0
fetch the duration of each clock cycle in picoseconds
- virtual std::string [getPcbVersion](#) ()=0
Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.
- virtual std::string [getFirmwareVersion](#) ()=0
Return an unique identifier for the applied firmware.
- virtual std::string [getSensorData](#) ()=0
Show the status of the sensor data from the FPGA and peripherals on the console.
- virtual void [setLED](#) (uint32_t bitmask)=0
Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.
- virtual std::string [getLicenseInfo](#) ()=0
- virtual void [setSoundFrequency](#) (uint32_t freq_hz)=0
Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)
- virtual long long [getOverflowsClient](#) ()=0
- virtual void [clearOverflowsClient](#) ()=0
- virtual long long [getOverflowsAndClearClient](#) ()=0

Additional Inherited Members

7.48.1 Detailed Description

network [TimeTagger](#) client.

The [TimeTaggerNetwork](#) class is a client that implements access to the Time Tagger server. [TimeTaggerNetwork](#) receives the time-tag stream from the Time Tagger server over the network and provides an interface for controlling connection and the Time Tagger hardware. Instance of this class can be transparently used to create measurements, virtual channels and other [Iterator](#) instances.

7.48.2 Member Function Documentation

7.48.2.1 clearConditionalFilter()

```
virtual void TimeTaggerNetwork::clearConditionalFilter ( ) [pure virtual]
```

deactivates the conditional filter

equivalent to `setConditionalFilter({},{})`

7.48.2.2 clearOverflowsClient()

```
virtual void TimeTaggerNetwork::clearOverflowsClient ( ) [pure virtual]
```

7.48.2.3 getChannelList()

```
virtual std::vector<channel_t> TimeTaggerNetwork::getChannelList (
    ChannelEdge type = ChannelEdge::All ) [pure virtual]
```

fetch a vector of all physical input channel ids

The function returns the channel of all rising and falling edges. For example for the Time Tagger 20 (8 input channels) `TT_CHANNEL_NUMBER_SCHEME_ZERO`: {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} and for `TT_CHANNEL_NUMBER_SCHEME_ONE`: {-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8}

`TT_CHANNEL_RISING_EDGES` returns only the rising edges `SCHEME_ONE`: {1,2,3,4,5,6,7,8} and `TT_CHANNEL_FALLING_EDGES` return only the falling edges `SCHEME_ONE`: {-1,-2,-3,-4,-5,-6,-7,-8} which are the inverted Channels of the rising edges.

7.48.2.4 getChannelNumberScheme()

```
virtual int TimeTaggerNetwork::getChannelNumberScheme ( ) [pure virtual]
```

Fetch the configured numbering scheme for this [TimeTagger](#) object.

Please see [setTimeTaggerChannelNumberScheme\(\)](#) for details.

7.48.2.5 getConditionalFilterFiltered()

```
virtual std::vector<channel_t> TimeTaggerNetwork::getConditionalFilterFiltered ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see `setConditionalFilter`

7.48.2.6 getConditionalFilterTrigger()

```
virtual std::vector<channel_t> TimeTaggerNetwork::getConditionalFilterTrigger ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see setConditionalFilter

7.48.2.7 getDACRange()

```
virtual std::vector<double> TimeTaggerNetwork::getDACRange ( ) [pure virtual]
```

returns the minimum and the maximum voltage of the DACs as a trigger reference

7.48.2.8 getDelayClient()

```
virtual timestamp_t TimeTaggerNetwork::getDelayClient (
    channel_t channel ) [pure virtual]
```

get the time delay of a channel

see setDelayClient

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the software delay in picoseconds

7.48.2.9 getEventDivider()

```
virtual unsigned int TimeTaggerNetwork::getEventDivider (
    channel_t channel ) [pure virtual]
```

Returns the factor of the dividing filter.

See setEventDivider for further details.

Parameters

<i>channel</i>	channel to be queried
----------------	-----------------------

Returns

the configured divider

7.48.2.10 getFirmwareVersion()

```
virtual std::string TimeTaggerNetwork::getFirmwareVersion ( ) [pure virtual]
```

Return an unique identifier for the applied firmware.

This function returns a comma separated list of the firmware version with

- the device identifier: TT-20 or TT-Ultra
- the firmware identifier: FW 3
- optional the timestamp of the assembling of the firmware
- the firmware identifier of the USB chip: OK 1.30 eg "TT-Ultra, FW 3, TS 2018-11-13 22:57:32, OK 1.30"

7.48.2.11 getHardwareBufferSize()

```
virtual int TimeTaggerNetwork::getHardwareBufferSize ( ) [pure virtual]
```

queries the size of the USB queue

See setHardwareBufferSize for more information.

Returns

the actual size of the USB queue in events

7.48.2.12 getHardwareDelayCompensation()

```
virtual timestamp_t TimeTaggerNetwork::getHardwareDelayCompensation (
    channel_t channel ) [pure virtual]
```

get hardware delay compensation of a channel

The physical input delays are calibrated and compensated. However this compensation is implemented after the conditional filter and so affects its result. This function queries the effective input delay, which compensates the hardware delay.

Parameters

<i>channel</i>	the channel
----------------	-------------

Returns

the hardware delay compensation in picoseconds

7.48.2.13 getLicenseInfo()

```
virtual std::string TimeTaggerNetwork::getLicenseInfo ( ) [pure virtual]
```

Fetches and parses the current installed license on this device

Returns

a human readable string containing all information about the license on this device

7.48.2.14 getModel()

```
virtual std::string TimeTaggerNetwork::getModel ( ) [pure virtual]
```

identifies the hardware by Time Tagger Model

7.48.2.15 getNormalization()

```
virtual bool TimeTaggerNetwork::getNormalization (
    channel_t channel ) [pure virtual]
```

returns the the normalization of the distribution.

Refer the Manual for a description of this function.

Parameters

<i>channel</i>	the channel to query
----------------	----------------------

Returns

if the normalization is enabled

7.48.2.16 getOverflowsAndClearClient()

```
virtual long long TimeTaggerNetwork::getOverflowsAndClearClient ( ) [pure virtual]
```

7.48.2.17 getOverflowsClient()

```
virtual long long TimeTaggerNetwork::getOverflowsClient ( ) [pure virtual]
```

7.48.2.18 getPcbVersion()

```
virtual std::string TimeTaggerNetwork::getPcbVersion ( ) [pure virtual]
```

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version ≥ 1 is channel configuration ONE.

7.48.2.19 getPsPerClock()

```
virtual timestamp_t TimeTaggerNetwork::getPsPerClock ( ) [pure virtual]
```

fetch the duration of each clock cycle in picoseconds

7.48.2.20 getSensorData()

```
virtual std::string TimeTaggerNetwork::getSensorData ( ) [pure virtual]
```

Show the status of the sensor data from the FPGA and peripherals on the console.

7.48.2.21 getSerial()

```
virtual std::string TimeTaggerNetwork::getSerial ( ) [pure virtual]
```

identifies the hardware by serial number

7.48.2.22 getStreamBlockSizeEvents()

```
virtual int TimeTaggerNetwork::getStreamBlockSizeEvents ( ) [pure virtual]
```

7.48.2.23 getStreamBlockSizeLatency()

```
virtual int TimeTaggerNetwork::getStreamBlockSizeLatency ( ) [pure virtual]
```

7.48.2.24 getTestSignal()

```
virtual bool TimeTaggerNetwork::getTestSignal (
    channel_t channel ) [pure virtual]
```

fetch the status of the test signal generator

Parameters

<i>channel</i>	the channel
----------------	-------------

Implements [TimeTaggerBase](#).

7.48.2.25 getTestSignalDivider()

```
virtual int TimeTaggerNetwork::getTestSignalDivider ( ) [pure virtual]
```

get the divider for the frequency of the test signal

7.48.2.26 getTriggerLevel()

```
virtual double TimeTaggerNetwork::getTriggerLevel (
    channel\_t channel ) [pure virtual]
```

get the trigger voltage threshold of a channel

Parameters

<i>channel</i>	the channel
----------------	-------------

7.48.2.27 isConnected()

```
virtual bool TimeTaggerNetwork::isConnected ( ) [pure virtual]
```

check if the network time tagger is currently connected to a server

Returns

returns true if it's currently connected to a server; false, otherwise.

7.48.2.28 setConditionalFilter()

```
virtual void TimeTaggerNetwork::setConditionalFilter (
    std::vector< channel\_t > trigger,
    std::vector< channel\_t > filtered,
    bool hardwareDelayCompensation = true ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

Parameters

<i>trigger</i>	the channels that sets the condition
<i>filtered</i>	the channels that are filtered by the condition
<i>hardwareDelayCompensation</i>	if false, the physical hardware delay will not be compensated

7.48.2.29 setDelayClient()

```
virtual void TimeTaggerNetwork::setDelayClient (
    channel_t channel,
    timestamp_t time ) [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds.

This delay is implemented on the client and does not affect the server nor requires the Control flag.

Parameters

<i>channel</i>	the channel to set
<i>time</i>	the delay in picoseconds

7.48.2.30 setEventDivider()

```
virtual void TimeTaggerNetwork::setEventDivider (
    channel_t channel,
    unsigned int divider ) [pure virtual]
```

Divides the amount of transmitted edge per channel.

This filter decimates the events on a given channel by a specified factor. So for a divider n , every n th event is transmitted through the filter and $n-1$ events are skipped between consecutive transmitted events. If a conditional filter is also active, the event divider is applied after the conditional filter, so the conditional is applied to the complete event stream and only events which pass the conditional filter are forwarded to the divider.

As it is a hardware filter, it reduces the required USB bandwidth and CPU processing power, but it cannot be configured for virtual channels.

Parameters

<i>channel</i>	channel to be configured
<i>divider</i>	new divider, must be smaller than 65536

7.48.2.31 setHardwareBufferSize()

```
virtual void TimeTaggerNetwork::setHardwareBufferSize (
    int size ) [pure virtual]
```

sets the maximum USB buffer size

This option controls the maximum buffer size of the USB connection. This can be used to balance low input latency vs high (peak) throughput.

Parameters

<i>size</i>	the maximum buffer size in events
-------------	-----------------------------------

7.48.2.32 setLED()

```
virtual void TimeTaggerNetwork::setLED (
    uint32_t bitmask ) [pure virtual]
```

Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.

7.48.2.33 setNormalization()

```
virtual void TimeTaggerNetwork::setNormalization (
    std::vector< channel_t > channel,
    bool state ) [pure virtual]
```

enables or disables the normalization of the distribution.

Refer the Manual for a description of this function.

Parameters

<i>channel</i>	list of channels to modify
<i>state</i>	the new state

7.48.2.34 setSoundFrequency()

```
virtual void TimeTaggerNetwork::setSoundFrequency (
    uint32_t freq_hz ) [pure virtual]
```

Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)

Parameters

<i>freq_hz</i>	the generated audio frequency
----------------	-------------------------------

7.48.2.35 setStreamBlockSize()

```
virtual void TimeTaggerNetwork::setStreamBlockSize (
    int max_events,
    int max_latency ) [pure virtual]
```

sets the maximum events and latency for the stream block size

This option controls the latency and the block size of the data stream. The default values are max_events = 131072 events and max_latency = 20 ms. Depending on which of the two parameters is exceeded first, the block stream size is adjusted accordingly. The block size will be reduced automatically for blocks when no signal is arriving for 512 ns on the Time Tagger Ultra and 1536 ns for the Time Tagger 20. *

Parameters

<i>max_events</i>	maximum number of events
<i>max_latency</i>	maximum latency in ms

7.48.2.36 setTestSignalDivider()

```
virtual void TimeTaggerNetwork::setTestSignalDivider (
    int divider ) [pure virtual]
```

set the divider for the frequency of the test signal

The base clock of the test signal oscillator for the Time Tagger Ultra is running at 100.8 MHz sampled down by an factor of 2 to have a similar base clock as the Time Tagger 20 (~50 MHz). The default divider is 63 -> ~800 kEvents/s

Parameters

<i>divider</i>	frequency divisor of the oscillator
----------------	-------------------------------------

7.48.2.37 setTriggerLevel()

```
virtual void TimeTaggerNetwork::setTriggerLevel (
    channel_t channel,
    double voltage ) [pure virtual]
```

set the trigger voltage threshold of a channel

Parameters

<i>channel</i>	the channel to set
<i>voltage</i>	voltage level.. [0..1]

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

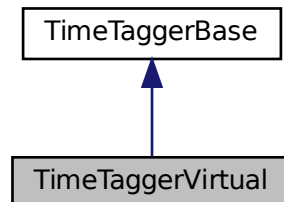
- [TimeTagger.h](#)

7.49 TimeTaggerVirtual Class Reference

virtual [TimeTagger](#) based on dump files

```
#include <TimeTagger.h>
```

Inheritance diagram for TimeTaggerVirtual:



Public Member Functions

- virtual uint64_t [replay](#) (const std::string &file, [timestamp_t](#) begin=0, [timestamp_t](#) duration=-1, bool queue=true)=0
replay a given dump file on the disc
- virtual void [stop](#) ()=0
stops the current and all queued files.
- virtual void [reset](#) ()=0
stops the all queued files and resets the [TimeTaggerVirtual](#) to its default settings
- virtual bool [waitForCompletion](#) (uint64_t ID=0, int64_t timeout=-1)=0
block the current thread until the replay finish
- virtual void [setReplaySpeed](#) (double speed)=0
configures the speed factor for the virtual tagger.
- virtual double [getReplaySpeed](#) ()=0
fetches the speed factor
- virtual void [setConditionalFilter](#) (std::vector< [channel_t](#) > trigger, std::vector< [channel_t](#) > filtered)=0
configures the conditional filter
- virtual void [clearConditionalFilter](#) ()=0
deactivates the conditional filter
- virtual std::vector< [channel_t](#) > [getConditionalFilterTrigger](#) ()=0
fetches the configuration of the conditional filter
- virtual std::vector< [channel_t](#) > [getConditionalFilterFiltered](#) ()=0
fetches the configuration of the conditional filter

Additional Inherited Members

7.49.1 Detailed Description

virtual [TimeTagger](#) based on dump files

The [TimeTaggerVirtual](#) class represents a virtual Time Tagger. But instead of connecting to Swabian hardware, it replays all tags from a recorded file.

7.49.2 Member Function Documentation

7.49.2.1 clearConditionalFilter()

```
virtual void TimeTaggerVirtual::clearConditionalFilter ( ) [pure virtual]
```

deactivates the conditional filter

equivalent to `setConditionalFilter({},{})`

7.49.2.2 getConditionalFilterFiltered()

```
virtual std::vector<channel_t> TimeTaggerVirtual::getConditionalFilterFiltered ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see `setConditionalFilter`

7.49.2.3 getConditionalFilterTrigger()

```
virtual std::vector<channel_t> TimeTaggerVirtual::getConditionalFilterTrigger ( ) [pure virtual]
```

fetches the configuration of the conditional filter

see `setConditionalFilter`

7.49.2.4 getReplaySpeed()

```
virtual double TimeTaggerVirtual::getReplaySpeed ( ) [pure virtual]
```

fetches the speed factor

Please see `setReplaySpeed` for more details.

Returns

the speed factor

7.49.2.5 replay()

```
virtual uint64_t TimeTaggerVirtual::replay (
    const std::string & file,
    timestamp_t begin = 0,
    timestamp_t duration = -1,
    bool queue = true ) [pure virtual]
```

replay a given dump file on the disc

This method adds the file to the replay queue. If the flag 'queue' is false, the current queue will be flushed and this file will be replayed immediately.

Parameters

<i>file</i>	the file to be replayed
<i>begin</i>	amount of ps to skip at the begin of the file. A negative time will generate a pause in the replay
<i>duration</i>	time period in ps of the file. -1 replays till the last tag
<i>queue</i>	flag if this file shall be queued

Returns

ID of the queued file

7.49.2.6 reset()

```
virtual void TimeTaggerVirtual::reset ( ) [pure virtual]
```

stops the all queued files and resets the [TimeTaggerVirtual](#) to its default settings

This method stops the current file, clears the replay queue and resets the [TimeTaggerVirtual](#) to its default settings.

7.49.2.7 setConditionalFilter()

```
virtual void TimeTaggerVirtual::setConditionalFilter (
    std::vector< channel_t > trigger,
    std::vector< channel_t > filtered ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

Parameters

<i>trigger</i>	the channels that sets the condition
<i>filtered</i>	the channels that are filtered by the condition

7.49.2.8 setReplaySpeed()

```
virtual void TimeTaggerVirtual::setReplaySpeed (
    double speed ) [pure virtual]
```

configures the speed factor for the virtual tagger.

This method configures the speed factor of this virtual Time Tagger. A value of 1.0 will replay in real time. All values < 0.0 will replay the data as fast as possible, but stops at the end of all data. This is the default value.

Parameters

<i>speed</i>	ratio of the replay speed and the real time
--------------	---

7.49.2.9 stop()

```
virtual void TimeTaggerVirtual::stop ( ) [pure virtual]
```

stops the current and all queued files.

This method stops the current file and clears the replay queue.

7.49.2.10 waitForCompletion()

```
virtual bool TimeTaggerVirtual::waitForCompletion (
    uint64_t ID = 0,
    int64_t timeout = -1 ) [pure virtual]
```

block the current thread until the replay finish

This method blocks the current execution and waits till the given file has finished its replay. If no ID is provided, it waits until all queued files are replayed.

This function does not block on a zero timeout. Negative timeouts are interpreted as infinite timeouts.

Parameters

<i>ID</i>	selects which file to wait for
<i>timeout</i>	timeout in milliseconds

Returns

true if the file is complete, false on timeout

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

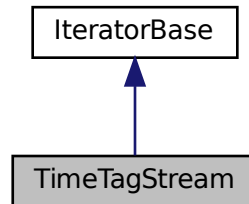
- [TimeTagger.h](#)

7.50 TimeTagStream Class Reference

access the time tag stream

```
#include <Iterators.h>
```

Inheritance diagram for TimeTagStream:



Public Member Functions

- `TimeTagStream (TimeTaggerBase *tagger, uint64_t n_max_events, std::vector< channel_t > channels)`
constructor of a `TimeTagStream` thread
- `~TimeTagStream ()`
- `uint64_t getCounts ()`
return the number of stored tags
- `TimeTagStreamBuffer getData ()`
fetches all stored tags and clears the internal state

Protected Member Functions

- `bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time)` override
update iterator state
- `void clear_impl ()` override
clear `Iterator` state.

Additional Inherited Members

7.50.1 Detailed Description

access the time tag stream

7.50.2 Constructor & Destructor Documentation

7.50.2.1 TimeTagStream()

```
TimeTagStream::TimeTagStream (
    TimeTaggerBase * tagger,
    uint64_t n_max_events,
    std::vector< channel_t > channels )
```

constructor of a [TimeTagStream](#) thread

Gives access to the time tag stream

Parameters

<i>tagger</i>	reference to a TimeTagger
<i>n_max_events</i>	maximum number of tags stored
<i>channels</i>	channels which are dumped to the file

7.50.2.2 ~TimeTagStream()

```
TimeTagStream::~~TimeTagStream ( )
```

7.50.3 Member Function Documentation

7.50.3.1 clear_impl()

```
void TimeTagStream::clear_impl ( ) [override], [protected], [virtual]
```

clear [Iterator](#) state.

Each [Iterator](#) should implement the [clear_impl\(\)](#) method to reset its internal state. The [clear_impl\(\)](#) function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

7.50.3.2 getCounts()

```
uint64_t TimeTagStream::getCounts ( )
```

return the number of stored tags

7.50.3.3 getData()

```
TimeTagStreamBuffer TimeTagStream::getData ( )
```

fetches all stored tags and clears the internal state

7.50.3.4 next_impl()

```
bool TimeTagStream::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each [Iterator](#) must implement the [next_impl\(\)](#) method. The [next_impl\(\)](#) function is guarded by the update lock.

The backend delivers each [Tag](#) on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

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

- [Iterators.h](#)

7.51 TimeTagStreamBuffer Class Reference

return object for [TimeTagStream::getData](#)

```
#include <Iterators.h>
```

Public Member Functions

- [~TimeTagStreamBuffer](#) ()
- void [getOverflows](#) (std::function< unsigned char *(size_t)> array_out)
- void [getChannels](#) (std::function< int *(size_t)> array_out)
- void [getTimestamps](#) (std::function< long long *(size_t)> array_out)
- void [getMissedEvents](#) (std::function< unsigned short *(size_t)> array_out)
- void [getEventTypes](#) (std::function< unsigned char *(size_t)> array_out)

Public Attributes

- uint64_t [size](#)
- bool [hasOverflows](#)
- timestamp_t [tStart](#)
- timestamp_t [tGetData](#)

7.51.1 Detailed Description

return object for [TimeTagStream::getData](#)

7.51.2 Constructor & Destructor Documentation

7.51.2.1 ~TimeTagStreamBuffer()

```
TimeTagStreamBuffer::~TimeTagStreamBuffer ( )
```

7.51.3 Member Function Documentation

7.51.3.1 getChannels()

```
void TimeTagStreamBuffer::getChannels (
    std::function< int *(size_t)> array_out )
```

7.51.3.2 getEventTypes()

```
void TimeTagStreamBuffer::getEventTypes (
    std::function< unsigned char *(size_t)> array_out )
```

7.51.3.3 getMissedEvents()

```
void TimeTagStreamBuffer::getMissedEvents (
    std::function< unsigned short *(size_t)> array_out )
```

7.51.3.4 getOverflows()

```
void TimeTagStreamBuffer::getOverflows (
    std::function< unsigned char *(size_t)> array_out )
```

7.51.3.5 getTimestamps()

```
void TimeTagStreamBuffer::getTimestamps (
    std::function< long long *(size_t)> array_out )
```

7.51.4 Member Data Documentation

7.51.4.1 hasOverflows

```
bool TimeTagStreamBuffer::hasOverflows
```

7.51.4.2 size

```
uint64_t TimeTagStreamBuffer::size
```

7.51.4.3 tGetData

```
timestamp_t TimeTagStreamBuffer::tGetData
```

7.51.4.4 tStart

```
timestamp_t TimeTagStreamBuffer::tStart
```

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

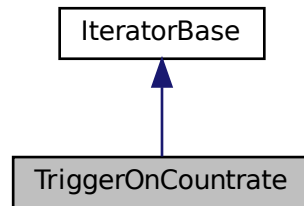
- [Iterators.h](#)

7.52 TriggerOnCountrate Class Reference

Inject trigger events when exceeding or falling below a given count rate within a rolling time window.

```
#include <Iterators.h>
```

Inheritance diagram for TriggerOnCountrate:



Public Member Functions

- [TriggerOnCountrate](#) ([TimeTaggerBase](#) *tagger, [channel_t](#) input_channel, double reference_countrate, double hysteresis, [timestamp_t](#) time_window)
constructor of a [TriggerOnCountrate](#)
- [~TriggerOnCountrate](#) ()
- [channel_t](#) [getChannelAbove](#) ()
*Get the channel number of the *above* channel.*
- [channel_t](#) [getChannelBelow](#) ()
*Get the channel number of the *below* channel.*
- [std::vector](#)< [channel_t](#) > [getChannels](#) ()
Get both virtual channel numbers: [[getChannelAbove](#) () , [getChannelBelow](#) ()].
- [bool](#) [isAbove](#) ()
*Returns whether the Virtual Channel is currently in the *above* state.*
- [bool](#) [isBelow](#) ()
*Returns whether the Virtual Channel is currently in the *below* state.*
- [double](#) [getCurrentCountrate](#) ()
*Get the current count rate averaged within the *time_window*.*
- [bool](#) [injectCurrentState](#) ()
Emit a time-tag into the respective channel according to the current state.

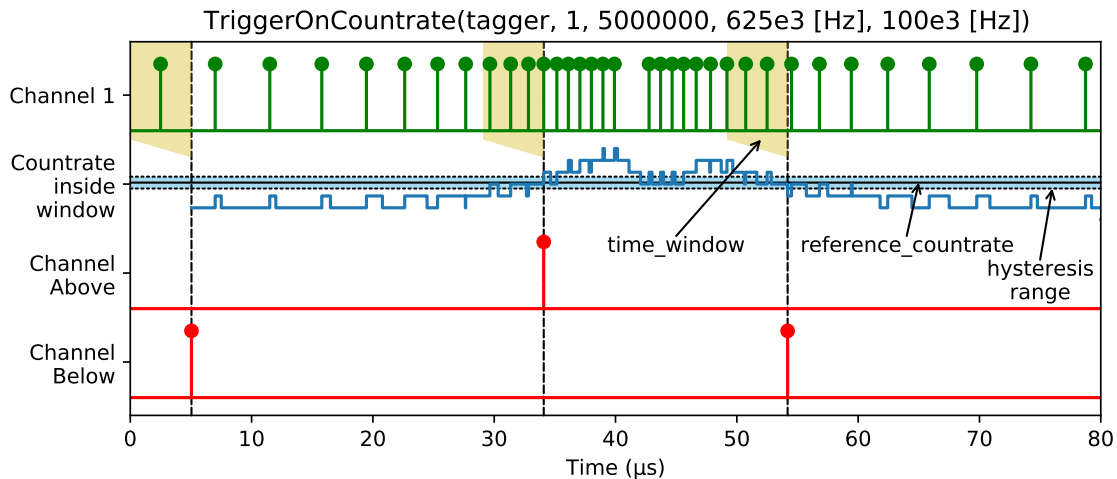
Protected Member Functions

- [bool](#) [next_impl](#) ([std::vector](#)< [Tag](#) > &incoming_tags, [timestamp_t](#) begin_time, [timestamp_t](#) end_time) override
update iterator state
- [void](#) [on_start](#) () override
callback when the measurement class is started

Additional Inherited Members

7.52.1 Detailed Description

Inject trigger events when exceeding or falling below a given count rate within a rolling time window.



Measures the count rate inside a rolling time window and emits tags when a given `reference_countrate` is crossed. A `TriggerOnCountrate` object provides two virtual channels: The `above` channel is triggered when the count rate exceeds the threshold (transition from `below` to `above`). The `below` channel is triggered when the count rate falls below the threshold (transition from `above` to `below`). To avoid the emission of multiple trigger tags in the transition area, the `hysteresis` count rate modifies the threshold with respect to the transition direction: An event in the `above` channel will be triggered when the channel is in the `below` state and rises to `reference_countrate + hysteresis` or above. Vice versa, the `below` channel fires when the channel is in the `above` state and falls to the limit of `reference_countrate - hysteresis` or below.

The time-tags are always injected at the end of the integration window. You can use the `DelayedChannel` to adjust the temporal position of the trigger tags with respect to the integration time window.

The very first tag of the virtual channel will be emitted `time_window` after the instantiation of the object and will reflect the current state, so either `above` or `below`.

7.52.2 Constructor & Destructor Documentation

7.52.2.1 TriggerOnCountrate()

```
TriggerOnCountrate::TriggerOnCountrate (
    TimeTaggerBase * tagger,
    channel_t input_channel,
    double reference_countrate,
    double hysteresis,
    timestamp_t time_window )
```

constructor of a `TriggerOnCountrate`

Parameters

<i>tagger</i>	Reference to a TimeTagger object.
<i>input_channel</i>	Channel number of the channel whose count rate will control the trigger channels.
<i>reference_countrate</i>	The reference count rate in Hz that separates the <code>above</code> range from the <code>below</code> range.
<i>hysteresis</i>	The threshold count rate in Hz for transitioning to the <code>above</code> threshold state is <code>countrate >= reference_countrate + hysteresis</code> , whereas it is <code>countrate <= reference_countrate - hysteresis</code> for transitioning to the <code>below</code> threshold state. The hysteresis avoids the emission of multiple trigger tags upon a single transition.
<i>time_window</i>	Rolling time window size in ps. The count rate is analyzed within this time window and compared to the threshold count rate.

7.52.2.2 ~TriggerOnCountrate()

```
TriggerOnCountrate::~~TriggerOnCountrate ( )
```

7.52.3 Member Function Documentation

7.52.3.1 getChannelAbove()

```
channel_t TriggerOnCountrate::getChannelAbove ( )
```

Get the channel number of the `above` channel.

7.52.3.2 getChannelBelow()

```
channel_t TriggerOnCountrate::getChannelBelow ( )
```

Get the channel number of the `below` channel.

7.52.3.3 getChannels()

```
std::vector<channel_t> TriggerOnCountrate::getChannels ( )
```

Get both virtual channel numbers: [`getChannelAbove()`, `getChannelBelow()`].

7.52.3.4 getCurrentCountrate()

```
double TriggerOnCountrate::getCurrentCountrate ( )
```

Get the current count rate averaged within the `time_window`.

7.52.3.5 injectCurrentState()

```
bool TriggerOnCountrate::injectCurrentState ( )
```

Emit a time-tag into the respective channel according to the current state.

Emit a time-tag into the respective channel according to the current state. This is useful if you start a new measurement that requires the information. The function returns whether it was possible to inject the event. The injection is not possible if the Time Tagger is in overflow mode or the time window has not passed yet. The function call is non-blocking.

7.52.3.6 isAbove()

```
bool TriggerOnCountrate::isAbove ( )
```

Returns whether the Virtual Channel is currently in the `above` state.

7.52.3.7 isBelow()

```
bool TriggerOnCountrate::isBelow ( )
```

Returns whether the Virtual Channel is currently in the `below` state.

7.52.3.8 next_impl()

```
bool TriggerOnCountrate::next_impl (
    std::vector< Tag > & incoming_tags,
    timestamp_t begin_time,
    timestamp_t end_time ) [override], [protected], [virtual]
```

update iterator state

Each `Iterator` must implement the `next_impl()` method. The `next_impl()` function is guarded by the update lock.

The backend delivers each `Tag` on each registered channel to this callback function.

Parameters

<i>incoming_tags</i>	block of events
<i>begin_time</i>	earliest event in the block
<i>end_time</i>	begin_time of the next block, not including in this block

Returns

true if the content of this block was modified, false otherwise

Implements [IteratorBase](#).

7.52.3.9 on_start()

```
void TriggerOnCountrate::on_start ( ) [override], [protected], [virtual]
```

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from [IteratorBase](#).

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

- [Iterators.h](#)

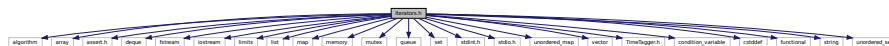
Chapter 8

File Documentation

8.1 Iterators.h File Reference

```
#include <algorithm>
#include <array>
#include <assert.h>
#include <deque>
#include <fstream>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <queue>
#include <set>
#include <stdint.h>
#include <stdio.h>
#include <unordered_map>
#include <vector>
#include "TimeTagger.h"
```

Include dependency graph for Iterators.h:



Classes

- class [FastBinning](#)
Helper class for fast division with a constant divisor.
- class [Combiner](#)
Combine some channels in a virtual channel which has a tick for each tick in the input channels.
- class [CountBetweenMarkers](#)
a simple counter where external marker signals determine the bins
- class [CounterData](#)
Helper object as return value for [Counter::getDataObject](#).

- class [Counter](#)
a simple counter on one or more channels
- class [Coincidences](#)
a coincidence monitor for many channel groups
- class [Coincidence](#)
a coincidence monitor for one channel group
- class [Countrate](#)
count rate on one or more channels
- class [DelayedChannel](#)
a simple delayed queue
- class [TriggerOnCountrate](#)
Inject trigger events when exceeding or falling below a given count rate within a rolling time window.
- class [GatedChannel](#)
An input channel is gated by a gate channel.
- class [FrequencyMultiplier](#)
The signal of an input channel is scaled up to a higher frequency according to the multiplier passed as a parameter.
- class [Iterator](#)
a deprecated simple event queue
- class [TimeTagStreamBuffer](#)
return object for [TimeTagStream::getData](#)
- class [TimeTagStream](#)
access the time tag stream
- class [Dump](#)
dump all time tags to a file
- class [StartStop](#)
simple start-stop measurement
- class [TimeDifferencesImpl< T >](#)
- class [TimeDifferences](#)
Accumulates the time differences between clicks on two channels in one or more histograms.
- class [Histogram2D](#)
A 2-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.
- class [HistogramND](#)
A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.
- class [TimeDifferencesND](#)
Accumulates the time differences between clicks on two channels in a multi-dimensional histogram.
- class [Histogram](#)
Accumulate time differences into a histogram.
- class [HistogramLogBins](#)
Accumulate time differences into a histogram with logarithmic increasing bin sizes.
- class [Correlation](#)
Auto- and Cross-correlation measurement.
- struct [Event](#)
Object for the return value of [Scope::getData](#).
- class [Scope](#)
a scope measurement
- class [SynchronizedMeasurements](#)
start, stop and clear several measurements synchronized
- class [ConstantFractionDiscriminator](#)
a virtual CFD implementation which returns the mean time between a raising and a falling pair of edges
- class [FileWriter](#)
compresses and stores all time tags to a file

- class [FileReader](#)
Reads tags from the disk files, which has been created by [FileWriter](#).
- class [EventGenerator](#)
Generate predefined events in a virtual channel relative to a trigger event.
- class [CustomMeasurementBase](#)
Helper class for custom measurements in Python and C#.
- class [FlimAbstract](#)
Interface for FLIM measurements, [Flim](#) and [FlimBase](#) classes inherit from it.
- class [FlimBase](#)
basic measurement, containing a minimal set of features for efficiency purposes
- class [FlimFrameInfo](#)
object for storing the state of [Flim::getCurrentFrameEx](#)
- class [Flim](#)
Fluorescence lifetime imaging.
- class [Sampler](#)
a triggered sampling measurement
- class [SyntheticSingleTag](#)
synthetic trigger timetag generator.
- class [FrequencyStabilityData](#)
return data object for [FrequencyStability::getData](#).
- class [FrequencyStability](#)
Allan deviation (and related metrics) calculator.

Macros

- #define [BINNING_TEMPLATE_HELPER](#)(fun_name, binner, ...)
[FastBinning](#) caller helper.

Enumerations

- enum [CoincidenceTimestamp](#) : uint32_t { [CoincidenceTimestamp::Last](#) = 0, [CoincidenceTimestamp::Average](#) = 1, [CoincidenceTimestamp::First](#) = 2, [CoincidenceTimestamp::ListedFirst](#) = 3 }
type of timestamp for the [Coincidence](#) virtual channel (Last, Average, First, ListedFirst)
- enum [State](#) { [UNKNOWN](#), [HIGH](#), [LOW](#) }
Input state in the return object of [Scope](#).

8.1.1 Macro Definition Documentation

8.1.1.1 BINNING_TEMPLATE_HELPER

```
#define BINNING_TEMPLATE_HELPER(
    fun_name,
    binner,
    ... )
```

Value:

```
switch (binner.getMode()) {
case FastBinning::Mode::ConstZero:
    fun_name<FastBinning::Mode::ConstZero>(__VA_ARGS__);
    break;
case FastBinning::Mode::Dividend:
    fun_name<FastBinning::Mode::Dividend>(__VA_ARGS__);
    break;
case FastBinning::Mode::PowerOfTwo:
    fun_name<FastBinning::Mode::PowerOfTwo>(__VA_ARGS__);
    break;
case FastBinning::Mode::FixedPoint_32:
    fun_name<FastBinning::Mode::FixedPoint_32>(__VA_ARGS__);
    break;
case FastBinning::Mode::FixedPoint_64:
    fun_name<FastBinning::Mode::FixedPoint_64>(__VA_ARGS__);
    break;
case FastBinning::Mode::Divide_32:
    fun_name<FastBinning::Mode::Divide_32>(__VA_ARGS__);
    break;
case FastBinning::Mode::Divide_64:
    fun_name<FastBinning::Mode::Divide_64>(__VA_ARGS__);
    break;
}
```

[FastBinning](#) caller helper.

8.1.2 Enumeration Type Documentation

8.1.2.1 CoincidenceTimestamp

```
enum CoincidenceTimestamp : uint32_t [strong]
```

type of timestamp for the [Coincidence](#) virtual channel (Last, Average, First, ListedFirst)

Enumerator

Last	time of the last event completing the coincidence (fastest option - default)
Average	average time of all tags completing the coincidence
First	time of the first event received of the coincidence
ListedFirst	time of the first channel of the list with which the Coincidence was initialized

8.1.2.2 State

enum [State](#)

Input state in the return object of [Scope](#).

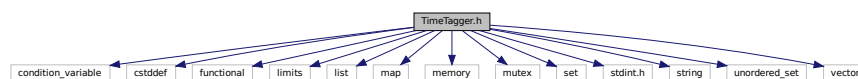
Enumerator

UNKNOWN	
HIGH	
LOW	

8.2 TimeTagger.h File Reference

```
#include <condition_variable>
#include <cstdint>
#include <functional>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <set>
#include <stdint.h>
#include <string>
#include <unordered_set>
#include <vector>
```

Include dependency graph for TimeTagger.h:



Classes

- struct [SoftwareClockState](#)
- class [CustomLogger](#)
Helper class for setLogger.
- class [TimeTaggerBase](#)
Basis interface for all Time Tagger classes.
- class [TimeTaggerVirtual](#)
virtual TimeTagger based on dump files
- class [TimeTaggerNetwork](#)
network TimeTagger client.
- class [TimeTagger](#)

- backend for the [TimeTagger](#).
- struct [Tag](#)
a single event on a channel
- class [OrderedBarrier](#)
Helper for implementing parallel measurements.
- class [OrderedBarrier::OrderInstance](#)
Internal object for serialization.
- class [OrderedPipeline](#)
Helper for implementing parallel measurements.
- class [IteratorBase](#)
Base class for all iterators.

Macros

- #define [TT_API](#) __declspec(dllimport)
- #define [timestamp_t](#) long long
The type for all timestamps used in the Time Tagger suite, always in picoseconds.
- #define [channel_t](#) int
The type for storing a channel identifier.
- #define [TIMETAGGER_VERSION](#) "2.10.6"
The version of this software suite.
- #define [GET_DATA_1D](#)(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t)> argout)
This are the default wrapper functions without any overloadings.
- #define [GET_DATA_1D_OP1](#)(function_name, type, argout, optional_type, optional_name, optional_default, attribute) attribute void function_name(std::function<type *(size_t)> argout, optional_type optional_name = optional_default)
- #define [GET_DATA_1D_OP2](#)(function_name, type, argout, optional_type, optional_name, optional_default, optional_type2, optional_name2, optional_default2, attribute)
- #define [GET_DATA_2D](#)(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t, size_t)> argout)
- #define [GET_DATA_2D_OP1](#)(function_name, type, argout, optional_type, optional_name, optional_default, attribute)
- #define [GET_DATA_2D_OP2](#)(function_name, type, argout, optional_type, optional_name, optional_default, optional_type2, optional_name2, optional_default2, attribute)
- #define [GET_DATA_3D](#)(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t, size_t, size_t)> argout)
- #define [LogMessage](#)(level, ...) [LogBase](#)(level, __FILE__, __LINE__, false, __VA_ARGS__);
- #define [ErrorLog](#)(...) [LogMessage](#)(LOGGER_ERROR, __VA_ARGS__);
- #define [WarningLog](#)(...) [LogMessage](#)(LOGGER_WARNING, __VA_ARGS__);
- #define [InfoLog](#)(...) [LogMessage](#)(LOGGER_INFO, __VA_ARGS__);
- #define [LogMessageSuppressed](#)(level, ...) [LogBase](#)(level, __FILE__, __LINE__, true, __VA_ARGS__);
- #define [ErrorLogSuppressed](#)(...) [LogMessageSuppressed](#)(LOGGER_ERROR, __VA_ARGS__);
- #define [WarningLogSuppressed](#)(...) [LogMessageSuppressed](#)(LOGGER_WARNING, __VA_ARGS__);
- #define [InfoLogSuppressed](#)(...) [LogMessageSuppressed](#)(LOGGER_INFO, __VA_ARGS__);

Typedefs

- typedef void(* [logger_callback](#)) ([LogLevel](#) level, std::string msg)
- using [_Iterator](#) = [IteratorBase](#)

Enumerations

- enum `Resolution` { `Resolution::Standard` = 0, `Resolution::HighResA` = 1, `Resolution::HighResB` = 2, `Resolution::HighResC` = 3 }

This enum selects the high resolution mode of the Time Tagger series.

- enum `ChannelEdge` : `int32_t` {
`ChannelEdge::NoFalling` = 1 << 0, `ChannelEdge::NoRising` = 1 << 1, `ChannelEdge::NoStandard` = 1 << 2,
`ChannelEdge::NoHighRes` = 1 << 3,
`ChannelEdge::All` = 0, `ChannelEdge::Rising` = 1, `ChannelEdge::Falling` = 2, `ChannelEdge::HighResAll` = 4,
`ChannelEdge::HighResRising` = 4 | 1, `ChannelEdge::HighResFalling` = 4 | 2, `ChannelEdge::StandardAll` = 8,
`ChannelEdge::StandardRising` = 8 | 1,
`ChannelEdge::StandardFalling` = 8 | 2 }

Enum for filtering the channel list returned by `getChannelList`.

- enum `LogLevel` { `LOGGER_ERROR` = 40, `LOGGER_WARNING` = 30, `LOGGER_INFO` = 10 }
- enum `AccessMode` { `AccessMode::Listen` = 0, `AccessMode::Control` = 2, `AccessMode::SynchronousControl` = 3 }
- enum `LanguageUsed` : `std::uint32_t` {
`LanguageUsed::Cpp` = 0, `LanguageUsed::Python`, `LanguageUsed::Csharp`, `LanguageUsed::Matlab`,
`LanguageUsed::Labview`, `LanguageUsed::Mathematica`, `LanguageUsed::Unknown` = 255 }
- enum `FrontendType` : `std::uint32_t` {
`FrontendType::Undefined` = 0, `FrontendType::WebApp`, `FrontendType::Firefly`, `FrontendType::Pyro5RPC`,
`FrontendType::UserFrontend` }
- enum `UsageStatisticsStatus` { `UsageStatisticsStatus::Disabled`, `UsageStatisticsStatus::Collecting`, `UsageStatisticsStatus::Collected` }

Functions

- `TT_API std::string getVersion ()`
Get the version of the TimeTagger cxx backend.
- `TT_API TimeTagger * createTimeTagger (std::string serial="", Resolution resolution=Resolution::Standard)`
default constructor factory.
- `TT_API TimeTaggerVirtual * createTimeTaggerVirtual ()`
default constructor factory for the createTimeTaggerVirtual class.
- `TT_API TimeTaggerNetwork * createTimeTaggerNetwork (std::string address="localhost:41101")`
default constructor factory for the TimeTaggerNetwork class.
- `TT_API void setCustomBitFileName (const std::string &bitFileName)`
set path and filename of the bitfile to be loaded into the FPGA
- `TT_API bool freeTimeTagger (TimeTaggerBase *tagger)`
free a copy of a TimeTagger reference.
- `TT_API std::vector< std::string > scanTimeTagger ()`
fetches a list of all available TimeTagger serials.
- `TT_API std::string getTimeTaggerServerInfo (std::string address="localhost:41101")`
connect to a Time Tagger server.
- `TT_API std::vector< std::string > scanTimeTaggerServers ()`
scan the local network for running time tagger servers.
- `TT_API std::string getTimeTaggerModel (const std::string &serial)`
- `TT_API void setTimeTaggerChannelNumberScheme (int scheme)`
Configure the numbering scheme for new TimeTagger objects.
- `TT_API int getTimeTaggerChannelNumberScheme ()`
Fetch the currently configured global numbering scheme.
- `TT_API bool hasTimeTaggerVirtualLicense ()`
Check if a license for the TimeTaggerVirtual is available.

- **TT_API** void [flashLicense](#) (const std::string &serial, const std::string &license)
Update the license on the device.
- **TT_API** std::string [extractLicenseInfo](#) (const std::string &license)
Parses the binary license and return a human readable information about this license.
- **TT_API** logger_callback [setLogger](#) (logger_callback callback)
Sets the notifier callback which is called for each log message.
- **TT_API** void [LogBase](#) (LogLevel level, const char *file, int line, bool suppressed, const char *fmt,...)
Raise a new log message. Please use the XXXLog macro instead.
- **TT_API** void [setLanguageInfo](#) (std::uint32_t pw, LanguageUsed language, std::string version)
sets the language being used currently for usage statistics system.
- **TT_API** void [setFrontend](#) (FrontendType frontend)
sets the frontend being used currently for usage statistics system.
- **TT_API** void [setUsageStatisticsStatus](#) (UsageStatisticsStatus new_status)
sets the status of the usage statistics system.
- **TT_API** UsageStatisticsStatus [getUsageStatisticsStatus](#) ()
gets the status of the usage statistics system.
- **TT_API** std::string [getUsageStatisticsReport](#) ()
gets the current recorded data by the usage statistics system.

Variables

- constexpr [channel_t](#) [CHANNEL_UNUSED](#) = -134217728
Constant for unused channel.
- constexpr [channel_t](#) [CHANNEL_UNUSED_OLD](#) = -1
- constexpr int [TT_CHANNEL_NUMBER_SCHEME_AUTO](#) = 0
Allowed values for [setTimeTaggerChannelNumberScheme\(\)](#).
- constexpr int [TT_CHANNEL_NUMBER_SCHEME_ZERO](#) = 1
- constexpr int [TT_CHANNEL_NUMBER_SCHEME_ONE](#) = 2
- constexpr [ChannelEdge](#) [TT_CHANNEL_RISING_AND_FALLING_EDGES](#) = [ChannelEdge::All](#)
- constexpr [ChannelEdge](#) [TT_CHANNEL_RISING_EDGES](#) = [ChannelEdge::Rising](#)
- constexpr [ChannelEdge](#) [TT_CHANNEL_FALLING_EDGES](#) = [ChannelEdge::Falling](#)

8.2.1 Macro Definition Documentation

8.2.1.1 [channel_t](#)

```
#define channel_t int
```

The type for storing a channel identifier.

8.2.1.2 [ErrorLog](#)

```
#define ErrorLog(  
    ... ) LogMessage (LOGGER_ERROR, __VA_ARGS__);
```

8.2.1.3 ErrorLogSuppressed

```
#define ErrorLogSuppressed(
    ... ) LogMessageSuppressed(LOGGER_ERROR, __VA_ARGS__);
```

8.2.1.4 GET_DATA_1D

```
#define GET_DATA_1D(
    function_name,
    type,
    argout,
    attribute ) attribute void function_name(std::function<type *(size_t)> argout)
```

This are the default wrapper functions without any overloadings.

8.2.1.5 GET_DATA_1D_OP1

```
#define GET_DATA_1D_OP1(
    function_name,
    type,
    argout,
    optional_type,
    optional_name,
    optional_default,
    attribute ) attribute void function_name(std::function<type *(size_t)> argout,
optional_type optional_name = optional_default)
```

8.2.1.6 GET_DATA_1D_OP2

```
#define GET_DATA_1D_OP2(
    function_name,
    type,
    argout,
    optional_type,
    optional_name,
    optional_default,
    optional_type2,
    optional_name2,
    optional_default2,
    attribute )
```

Value:

```
attribute void function_name(std::function<type *(size_t)> argout, optional_type optional_name =
    optional_default, \
    optional_type2 optional_name2 = optional_default2)
```

8.2.1.7 GET_DATA_2D

```
#define GET_DATA_2D(
    function_name,
    type,
    argout,
    attribute ) attribute void function_name(std::function<type *(size_t, size_t)>
argout)
```

8.2.1.8 GET_DATA_2D_OP1

```
#define GET_DATA_2D_OP1(
    function_name,
    type,
    argout,
    optional_type,
    optional_name,
    optional_default,
    attribute )
```

Value:

```
attribute void function_name(std::function<type *(size_t, size_t)> argout,
\
    optional_type optional_name = optional_default)
```

8.2.1.9 GET_DATA_2D_OP2

```
#define GET_DATA_2D_OP2(
    function_name,
    type,
    argout,
    optional_type,
    optional_name,
    optional_default,
    optional_type2,
    optional_name2,
    optional_default2,
    attribute )
```

Value:

```
attribute void function_name(std::function<type *(size_t, size_t)> argout,
\
\
    optional_type optional_name = optional_default,
    optional_type2 optional_name2 = optional_default2)
```

8.2.1.10 GET_DATA_3D

```
#define GET_DATA_3D(  
    function_name,  
    type,  
    argout,  
    attribute ) attribute void function_name(std::function<type *(size_t, size_t,  
size_t)> argout)
```

8.2.1.11 InfoLog

```
#define InfoLog(  
    ... ) LogMessage(LOGGER_INFO, __VA_ARGS__);
```

8.2.1.12 InfoLogSuppressed

```
#define InfoLogSuppressed(  
    ... ) LogMessageSuppressed(LOGGER_INFO, __VA_ARGS__);
```

8.2.1.13 LogMessage

```
#define LogMessage(  
    level,  
    ... ) LogBase(level, __FILE__, __LINE__, false, __VA_ARGS__);
```

8.2.1.14 LogMessageSuppressed

```
#define LogMessageSuppressed(  
    level,  
    ... ) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);
```

8.2.1.15 timestamp_t

```
#define timestamp_t long long
```

The type for all timestamps used in the Time Tagger suite, always in picoseconds.

8.2.1.16 TIMETAGGER_VERSION

```
#define TIMETAGGER_VERSION "2.10.6"
```

The version of this software suite.

8.2.1.17 TT_API

```
#define TT_API __declspec(dllimport)
```

8.2.1.18 WarningLog

```
#define WarningLog(  
    ... ) LogMessage(LOGGER\_WARNING, __VA_ARGS__);
```

8.2.1.19 WarningLogSuppressed

```
#define WarningLogSuppressed(  
    ... ) LogMessageSuppressed(LOGGER\_WARNING, __VA_ARGS__);
```

8.2.2 Typedef Documentation

8.2.2.1 _Iterator

```
using \_Iterator = IteratorBase
```

8.2.2.2 logger_callback

```
typedef void(* logger_callback) (LogLevel level, std::string msg)
```

8.2.3 Enumeration Type Documentation

8.2.3.1 AccessMode

```
enum AccessMode [strong]
```


Enumerator

Listen	
Control	
SynchronousControl	

8.2.3.2 ChannelEdge

```
enum ChannelEdge : int32_t [strong]
```

Enum for filtering the channel list returned by getChannellist.

Enumerator

NoFalling	
NoRising	
NoStandard	
NoHighRes	
All	
Rising	
Falling	
HighResAll	
HighResRising	
HighResFalling	
StandardAll	
StandardRising	
StandardFalling	

8.2.3.3 FrontendType

```
enum FrontendType : std::uint32_t [strong]
```

Enumerator

Undefined	
WebApp	
Firefly	
Pyro5RPC	
UserFrontend	

8.2.3.4 LanguageUsed

```
enum LanguageUsed : std::uint32_t [strong]
```

Enumerator

Cpp	
Python	
Csharp	
Matlab	
Labview	
Mathematica	
Unknown	

8.2.3.5 LogLevel

```
enum LogLevel
```

Enumerator

LOGGER_ERROR	
LOGGER_WARNING	
LOGGER_INFO	

8.2.3.6 Resolution

```
enum Resolution [strong]
```

This enum selects the high resolution mode of the Time Tagger series.

If any high resolution mode is selected, the hardware will combine 2, 4 or even 8 input channels and average their timestamps. This results in a discretization jitter improvement of factor \sqrt{N} for N combined channels. The averaging is implemented before any filter, buffer or USB transmission. So all of those features are available with the averaged timestamps. Because of hardware limitations, only fixed combinations of channels are supported:

- HighResA: 1 : [1,2], 3 : [3,4], 5 : [5,6], 7 : [7,8], 10 : [10,11], 12 : [12,13], 14 : [14,15], 16 : [16,17], 9, 18
- HighResB: 1 : [1,2,3,4], 5 : [5,6,7,8], 10 : [10,11,12,13], 14 : [14,15,16,17], 9, 18
- HighResC: 5 : [1,2,3,4,5,6,7,8], 14 : [10,11,12,13,14,15,16,17], 9, 18 The inputs 9 and 18 are always available without averaging. The number of channels available will be limited to the number of channels licensed.

Enumerator

Standard	
HighResA	
HighResB	
HighResC	

8.2.3.7 UsageStatisticsStatus

```
enum UsageStatisticsStatus [strong]
```

Enumerator

Disabled	
Collecting	
CollectingAndUploading	

8.2.4 Function Documentation

8.2.4.1 createTimeTagger()

```
TT_API TimeTagger* createTimeTagger (
    std::string serial = "",
    Resolution resolution = Resolution::Standard )
```

default constructor factory.

Parameters

<i>serial</i>	serial number of FPGA board to use. if empty, the first board found is used.
<i>resolution</i>	enum for how many channels shall be grouped.

See also

[Resolution](#) for details

8.2.4.2 createTimeTaggerNetwork()

```
TT_API TimeTaggerNetwork* createTimeTaggerNetwork (
    std::string address = "localhost:41101" )
```

default constructor factory for the [TimeTaggerNetwork](#) class.

Parameters

<i>address</i>	IP address of the server. Use hostname:port.
----------------	--

8.2.4.3 createTimeTaggerVirtual()

```
TT_API TimeTaggerVirtual* createTimeTaggerVirtual ( )
```

default constructor factory for the createTimeTaggerVirtual class.

8.2.4.4 extractLicenseInfo()

```
TT_API std::string extractLicenseInfo (
    const std::string & license )
```

Parses the binary license and return a human readable information about this license.

Parameters

<i>license</i>	the binary license, encoded as a hexadecimal string
----------------	---

Returns

a human readable string containing all information about this license

8.2.4.5 flashLicense()

```
TT_API void flashLicense (
    const std::string & serial,
    const std::string & license )
```

Update the license on the device.

Updated license may be fetched by getRemoteLicense. The Time Tagger must not be instantiated while updating the license.

Parameters

<i>serial</i>	the serial of the device to update the license. Must not be empty
<i>license</i>	the binary license, encoded as a hexadecimal string

8.2.4.6 freeTimeTagger()

```
TT_API bool freeTimeTagger (
    TimeTaggerBase * tagger )
```

free a copy of a [TimeTagger](#) reference.

Parameters

<i>tagger</i>	the TimeTagger reference to free
---------------	--

8.2.4.7 getTimeTaggerChannelNumberScheme()

```
TT_API int getTimeTaggerChannelNumberScheme ( )
```

Fetch the currently configured global numbering scheme.

Please see [setTimeTaggerChannelNumberScheme\(\)](#) for details. Please use [TimeTagger::getChannelNumberScheme\(\)](#) to query the actual used numbering scheme, this function here will just return the scheme a newly created [TimeTagger](#) object will use.

8.2.4.8 getTimeTaggerModel()

```
TT_API std::string getTimeTaggerModel (
    const std::string & serial )
```

8.2.4.9 getTimeTaggerServerInfo()

```
TT_API std::string getTimeTaggerServerInfo (
    std::string address = "localhost:41101" )
```

connect to a Time Tagger server.

Parameters

<i>address</i>	ip address or domain and port of the server hosting time tagger. Use hostname:port.
----------------	---

8.2.4.10 getUsageStatisticsReport()

```
TT_API std::string getUsageStatisticsReport ( )
```

gets the current recorded data by the usage statistics system.

Use this function to see what data has been collected so far and what will be sent to Swabian Instruments if 'CollectingAndUploading' is enabled. All data is pseudonymous.

Note

if no data has been collected or due to a system error, the database was corrupted, it will return an error. else it will be a database in json format.

Returns

the current recorded data by the usage statistics system.

8.2.4.11 `getUsageStatisticsStatus()`

```
TT_API UsageStatisticsStatus getUsageStatisticsStatus ( )
```

gets the status of the usage statistics system.

Returns

the current status of the usage statistics system.

8.2.4.12 `getVersion()`

```
TT_API std::string getVersion ( )
```

Get the version of the [TimeTagger](#) cxx backend.

8.2.4.13 `hasTimeTaggerVirtualLicense()`

```
TT_API bool hasTimeTaggerVirtualLicense ( )
```

Check if a license for the [TimeTaggerVirtual](#) is available.

8.2.4.14 `LogBase()`

```
TT_API void LogBase (
    LogLevel level,
    const char * file,
    int line,
    bool suppressed,
    const char * fmt,
    ... )
```

Raise a new log message. Please use the XXXLog macro instead.

8.2.4.15 scanTimeTagger()

```
TT_API std::vector<std::string> scanTimeTagger ( )
```

fetches a list of all available [TimeTagger](#) serials.

This function may return serials blocked by other processes or already disconnected some milliseconds later.

8.2.4.16 scanTimeTaggerServers()

```
TT_API std::vector<std::string> scanTimeTaggerServers ( )
```

scan the local network for running time tagger servers.

Returns

a vector of strings of "ip_address:port" for each active server in local network.

8.2.4.17 setCustomBitFileName()

```
TT_API void setCustomBitFileName (
    const std::string & bitFileName )
```

set path and filename of the bitfile to be loaded into the FPGA

For debugging/development purposes the firmware loaded into the FPGA can be set manually with this function. To load the default bitfile set bitFileName = ""

Parameters

<i>bitFileName</i>	custom bitfile to use for the FPGA.
--------------------	-------------------------------------

8.2.4.18 setFrontend()

```
TT_API void setFrontend (
    FrontendType frontend )
```

sets the frontend being used currently for usage statistics system.

Parameters

<i>frontend</i>	the frontend currently being used.
-----------------	------------------------------------

8.2.4.19 setLanguageInfo()

```
TT_API void setLanguageInfo (
    std::uint32_t pw,
    LanguageUsed language,
    std::string version )
```

sets the language being used currently for usage statistics system.

Parameters

<i>pw</i>	password for authorization to change the language.
<i>language</i>	programming language being used.
<i>version</i>	version of the programming language being used.

8.2.4.20 setLogger()

```
TT_API logger_callback setLogger (
    logger_callback callback )
```

Sets the notifier callback which is called for each log message.

If this function is called with nullptr, the default callback will be used.

Returns

The old callback

8.2.4.21 setTimeTaggerChannelNumberScheme()

```
TT_API void setTimeTaggerChannelNumberScheme (
    int scheme )
```

Configure the numbering scheme for new [TimeTagger](#) objects.

This function sets the numbering scheme for newly created [TimeTagger](#) objects. The default value is `_AUTO`.

Note: [TimeTagger](#) objects are cached internally, so the scheme should be set before the first call of [createTimeTagger\(\)](#).

`_ZERO` will typically allocate the channel numbers 0 to 7 for the 8 input channels. 8 to 15 will be allocated for the corresponding falling events.

`_ONE` will typically allocate the channel numbers 1 to 8 for the 8 input channels. -1 to -8 will be allocated for the corresponding falling events.

`_AUTO` will choose the scheme based on the hardware revision and so based on the printed label.

Parameters

<i>scheme</i>	new numbering scheme, must be TT_CHANNEL_NUMBER_SCHEME_AUTO, TT_CHANNEL_NUMBER_SCHEME_ZERO or TT_CHANNEL_NUMBER_SCHEME_ONE
---------------	--

8.2.4.22 setUsageStatisticsStatus()

```
TT_API void setUsageStatisticsStatus (
    UsageStatisticsStatus new_status )
```

sets the status of the usage statistics system.

This functionality allows configuring the usage statistics system.

Parameters

<i>new_status</i>	new status of the usage statistics system.
-------------------	--

8.2.5 Variable Documentation**8.2.5.1 CHANNEL_UNUSED**

```
constexpr channel_t CHANNEL_UNUSED = -134217728 [constexpr]
```

Constant for unused channel.

Magic channel_t value to indicate an unused channel. So the iterators either have to disable this channel, or to choose a default one.

This value changed in version 2.1. The old value -1 aliases with falling events. The old value will still be accepted for now if the old numbering scheme is active.

8.2.5.2 CHANNEL_UNUSED_OLD

```
constexpr channel_t CHANNEL_UNUSED_OLD = -1 [constexpr]
```

8.2.5.3 TT_CHANNEL_FALLING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_FALLING_EDGES = ChannelEdge::Falling [constexpr]
```

8.2.5.4 TT_CHANNEL_NUMBER_SCHEME_AUTO

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_AUTO = 0 [constexpr]
```

Allowed values for [setTimeTaggerChannelNumberScheme\(\)](#).

`_ZERO` will typically allocate the channel numbers 0 to 7 for the 8 input channels. 8 to 15 will be allocated for the corresponding falling events.

`_ONE` will typically allocate the channel numbers 1 to 8 for the 8 input channels. -1 to -8 will be allocated for the corresponding falling events.

`_AUTO` will choose the scheme based on the hardware revision and so based on the printed label.

8.2.5.5 TT_CHANNEL_NUMBER_SCHEME_ONE

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_ONE = 2 [constexpr]
```

8.2.5.6 TT_CHANNEL_NUMBER_SCHEME_ZERO

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_ZERO = 1 [constexpr]
```

8.2.5.7 TT_CHANNEL_RISING_AND_FALLING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_RISING_AND_FALLING_EDGES = ChannelEdge::All [constexpr]
```

8.2.5.8 TT_CHANNEL_RISING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_RISING_EDGES = ChannelEdge::Rising [constexpr]
```