TimeTagger

2.9.0.0

Generated by Doxygen 1.8.11

Contents

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

Deprecated List

Class Dump
: use FileWriter

Class Iterator
: use TimeTagStream

Member IteratorBase::lock ()
: use getLock

Member IteratorBase::unlock ()
: use getLock

Member TimeTagger::getDistributionPSecs (std::function< long long *(size_t, size_t)> array_out)=0

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 base iterators

base iterators for photon counting applications

Classes

- 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 Counter

  a simple counter on one or more channels
- class Coincidences

  a coincidence monitor for one or more channel groups
- class Coincidence

  a coincidence monitor for one or more channel groups
- 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 simple event queue
- class TimeTagStream

  access the time tag stream
- class Dump

  dump all time tags to a file
- 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 spectrocopy.
- 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

  cross-correlation between two channels
- class Scope
- 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 EventGenerator

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

  Fluorescence lifetime imaging.
- class SyntheticSingleTag

  synthetic trigger timetag generator.

7.1.1 Detailed Description

base iterators for photon counting applications

Chapter 8

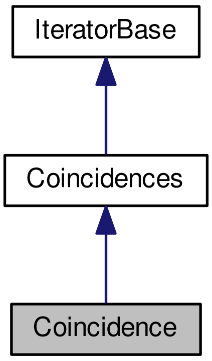
Class Documentation

8.1 Coincidence Class Reference

a coincidence monitor for one or more channel groups

```
#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

8.1.1 Detailed Description

a coincidence monitor for one or more channel groups

Monitor coincidences for a given channel groups passed by the constructor. A coincidence is event is detected when all slected 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 outperformes multiple instances of Conincdence.

8.1.2 Constructor & Destructor Documentation

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

construct a coincidence

Parameters

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

8.1.3 Member Function Documentation

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

virtual channel which contains the coincidences

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

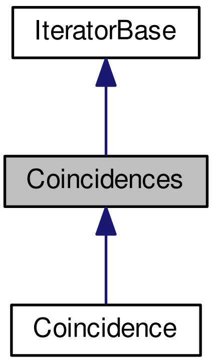
- Iterators.h

8.2 Coincidences Class Reference

a coincidence monitor for one or more 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

Friends

- class CoincidencesImpl

Additional Inherited Members

8.2.1 Detailed Description

a coincidence monitor for one or more 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 coincidence← Window [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.

8.2.2 Constructor & Destructor Documentation

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

construct a Coincidences

Parameters

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

8.2.2.2 Coincidences::∼Coincidences ( )

8.2.3 Member Function Documentation

8.2.3.1 std::vector$<$**channel_t**$>$ Coincidences::getChannels ( )

fetches the block of virtual channels for those coincidence groups

8.2.3.2 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.2.3.3 void Coincidences::setCoincidenceWindow (**timestamp_t** *coincidenceWindow*)

8.2.4 Friends And Related Function Documentation

8.2.4.1 friend class CoincidencesImpl [friend]

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

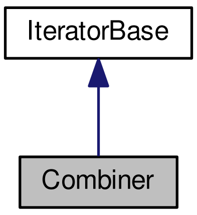
- Iterators.h

8.3 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 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.

Friends

- class CombinerImpl

Additional Inherited Members

8.3.1 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.

8.3.2 Constructor & Destructor Documentation

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

construct a combiner

Parameters

tagger	reference to a TimeTagger
channels	vector of channels to combine

8.3.2.2 Combiner::∼Combiner ( )

8.3.3 Member Function Documentation

8.3.3.1 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.

8.3.3.2 **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.

8.3.3.3 void Combiner::getData (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.

8.3.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.3.4 Friends And Related Function Documentation

8.3.4.1 friend class CombinerImpl `[friend]`

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

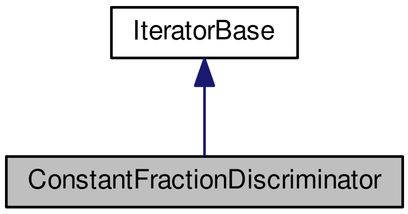
- Iterators.h

8.4 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

Friends

- class ConstantFractionDiscriminatorImpl

Additional Inherited Members

8.4.1 Detailed Description

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

8.4.2 Constructor & Destructor Documentation

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

constructor of a ConstantFractionDiscriminator

Parameters

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

8.4.2.2 ConstantFractionDiscriminator::∼ConstantFractionDiscriminator ( )

8.4.3 Member Function Documentation

8.4.3.1 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.

8.4.3.2 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.4.3.3 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.

8.4.4 Friends And Related Function Documentation

8.4.4.1 friend class ConstantFractionDiscriminatorImpl [friend]

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

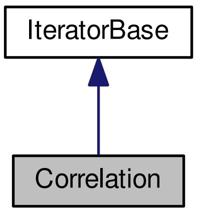
- Iterators.h

8.5 Correlation Class Reference

cross-correlation between two channels

`#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 ()
- 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 histogram - normalized such that a perfectly uncorrelated signals would be flat at a height of one
- 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.

Friends

- class CorrelationImpl

Additional Inherited Members

8.5.1 Detailed Description

cross-correlation between two channels

Accumulates time differences between clicks on two channels into a histogram, where all ticks are considered both as start and stop clicks and both positive and negative time differences are considered. The histogram is determined by the number of total bins and the binwidth.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 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

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

Parameters

tagger	reference to a TimeTagger
channel↩_1	first click channel
channel↩_2	second click channel
binwidth	width of one histogram bin in ps
n_bins	the number of bins in the resulting histogram

8.5.2.2 Correlation::∼Correlation ( )

8.5.3 Member Function Documentation

8.5.3.1 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.

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

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

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

get the histogram - normalized such that a perfectly uncorrelated signals would be flat at a height of one

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

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

8.5.3.5 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.5.4 Friends And Related Function Documentation

8.5.4.1 friend class CorrelationImpl [friend]

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

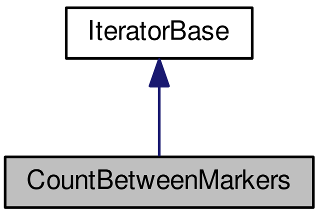
- Iterators.h

8.6 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 ()

  tbd
- void getData (std::function< int32_t ∗(size_t)> array_out)

  tbd
- 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.

Friends

- class CountBetweenMarkersImpl

Additional Inherited Members

8.6.1 Detailed Description

a simple counter where external marker signals determine the bins

Counter with external signals that trigger beginning and end of each counter accumulation. This can be used to implement counting triggered by a pixel clock and gated counting. The thread waits for the first time tag on the 'begin_channel', then begins counting time tags on the 'click_channel'. It ends counting when a tag on the 'end_↩ channel' is detected.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 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

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

8.6.2.2 CountBetweenMarkers::∼CountBetweenMarkers ( )

8.6.3 Member Function Documentation

8.6.3.1 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.

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

fetches the widths of each bins

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

tbd

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

fetches the starting time of each bin

8.6.3.5 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.6.3.6 bool CountBetweenMarkers::ready ( )

tbd

8.6.4 Friends And Related Function Documentation

8.6.4.1 friend class CountBetweenMarkersImpl [friend]

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

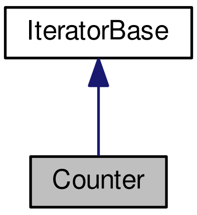
- Iterators.h

8.7 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)

    get counts
- 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

Friends

- class CounterImpl

Additional Inherited Members

8.7.1 Detailed Description

a simple counter on one or more channels

Counter with fixed binwidth and circular buffer output. This class is suitable to generate a time trace of the count rate on one or more channels. The thread repeatedly counts clicks on a single channel over a given time interval and stores the results in a two-dimensional array. The array is treated as a circular buffer. I.e., once the array is full, each new value shifts all previous values one element to the left.

8.7.2 Constructor & Destructor Documentation

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

construct a counter

Parameters

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

8.7.2.2 Counter::∼Counter ( )

8.7.3 Member Function Documentation

8.7.3.1 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.

8.7.3.2 void Counter::getData (std::function< int32_t ∗(size_t, size_t)> *array_out*)

get counts

the counts are copied to a newly allocated allocated memory, an the pointer to this location is returned.

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

8.7.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.7.3.5 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.

8.7.4 Friends And Related Function Documentation

8.7.4.1 friend class CounterImpl `[friend]`

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

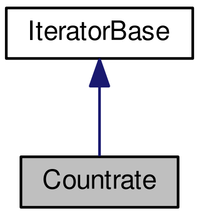
- Iterators.h

8.8 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

Friends

- class CountrateImpl

Additional Inherited Members

8.8.1 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.

8.8.2 Constructor & Destructor Documentation

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

constructor of Countrate

Parameters

tagger	reference to a TimeTagger
channels	the channels to count on

8.8.2.2 Countrate::∼Countrate ( )

8.8.3 Member Function Documentation

8.8.3.1 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.

8.8.3.2 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.

8.8.3.3 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.

8.8.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.8.3.5 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.

8.8.4 Friends And Related Function Documentation

8.8.4.1 friend class CountrateImpl [friend]

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

- Iterators.h

8.9 CustomLogger Class Reference

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

Public Member Functions

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

8.9.1 Constructor & Destructor Documentation

8.9.1.1 CustomLogger::CustomLogger ( )

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

8.9.2 Member Function Documentation

8.9.2.1 void CustomLogger::disable ( )

8.9.2.2 void CustomLogger::enable ( )

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

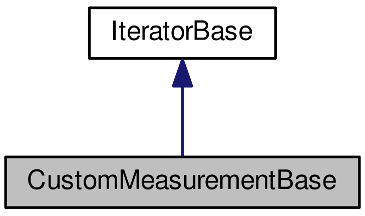
The documentation for this class was generated from the following file:

- TimeTagger.h

8.10 CustomMeasurementBase Class Reference

```
#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

8.10.1 Constructor & Destructor Documentation

8.10.1.1 CustomMeasurementBase::CustomMeasurementBase (TimeTaggerBase ∗ *tagger*) [protected]

8.10.1.2 CustomMeasurementBase::∼CustomMeasurementBase ( ) [override]

8.10.2 Member Function Documentation

8.10.2.1 void CustomMeasurementBase::_lock ( )

8.10.2.2 void CustomMeasurementBase::_unlock ( )

8.10.2.3 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.

8.10.2.4 void CustomMeasurementBase::finalize_init ( )

8.10.2.5 bool CustomMeasurementBase::is_running ( ) const

8.10.2.6 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

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

8.10.2.8 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.

8.10.2.9 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.

8.10.2.10 void CustomMeasurementBase::register_channel (channel_t *channel*)

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

8.10.2.12 void CustomMeasurementBase::unregister_channel (channel_t *channel*)

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

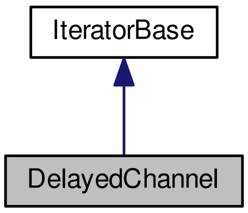
- Iterators.h

8.11 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

Friends

- class DelayedChannelImpl

Additional Inherited Members

8.11.1 Detailed Description

a simple delayed queue

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

8.11.2 Constructor & Destructor Documentation

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

constructor of a DelayedChannel

Parameters

tagger	reference to a TimeTagger
input_channel	channel which is delayed
delay	amount of time to delay

8.11.2.2 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

tagger	reference to a TimeTagger
input_channels	channels which will be delayed
delay	amount of time to delay

8.11.2.3 DelayedChannel::$\sim$DelayedChannel ( )

8.11.3 Member Function Documentation

8.11.3.1 **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.

8.11.3.2 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.

8.11.3.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.11.3.4 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.

8.11.3.5 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.

8.11.4 Friends And Related Function Documentation

8.11.4.1 friend class DelayedChannelImpl [friend]

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

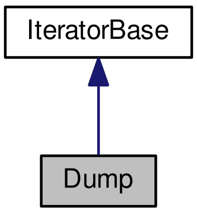
- Iterators.h

8.12 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 ()

  tbd

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

Friends

- class DumpImpl

Additional Inherited Members

8.12.1 Detailed Description

dump all time tags to a file

Deprecated use FileWriter

8.12.2 Constructor & Destructor Documentation

8.12.2.1 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

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

8.12.2.2 Dump::∼Dump ( )

tbd

8.12.3 Member Function Documentation

8.12.3.1 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.

8.12.3.2 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.12.3.3 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.

8.12.3.4 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.

8.12.4 Friends And Related Function Documentation

8.12.4.1 friend class DumpImpl `[friend]`

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

- Iterators.h

8.13 Event Struct Reference

`#include <Iterators.h>`

Public Attributes

- timestamp_t time
- State state

8.13.1 Member Data Documentation

8.13.1.1 State Event::state

8.13.1.2 timestamp_t Event::time

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

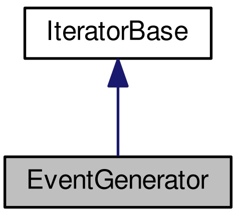
- Iterators.h

8.14 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

Friends

- class EventGeneratorImpl

Additional Inherited Members

8.14.1 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↩
_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.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 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

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

8.14.2.2 EventGenerator::∼EventGenerator ( )

8.14.3 Member Function Documentation

8.14.3.1 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.

8.14.3.2 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.

8.14.3.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.14.3.4 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.

8.14.4 Friends And Related Function Documentation

8.14.4.1 friend class EventGeneratorImpl `[friend]`

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

- Iterators.h

8.15 FastBinning Class Reference

`#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

8.15.1 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.

8.15.2 Member Enumeration Documentation

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

Enumerator

ConstZero

Dividend

PowerOfTwo

FixedPoint_32

FixedPoint_64

Divide_32

Divide_64

8.15.3 Constructor & Destructor Documentation

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

8.15.3.2 FastBinning::FastBinning (uint64_t *divisor,* uint64_t *max_duration_*)

8.15.4 Member Function Documentation

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

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

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

- Iterators.h

8.16 FileReader Class Reference

`#include <Iterators.h>`

Public Member Functions

- FileReader (std::vector< std::string > filenames)
- FileReader (const std::string &filename)
- ~FileReader ()
- bool hasData ()
- TimeTagStreamBuffer getData (uint64_t n_events)
- bool getDataRaw (std::vector< Tag > &tag_buffer)
- std::string getConfiguration ()
- std::string getLastMarker ()

Friends

- class FileReaderImpl

8.16.1 Detailed Description

Reads tags from the disk files, which has been created by FileWriter. Its usage is compatible with the TimeTag←Stream.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 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

filenames	list of files to read

8.16.2.2 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

filename	file to read

8.16.2.3 FileReader::∼FileReader ( )

8.16.3 Member Function Documentation

8.16.3.1 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.

8.16.3.2 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

n_events	maximum amount of elements to fetch

Returns

a TimeTagStreamBuffer with up to n_events events

8.16.3.3 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

tag_buffer	a buffer, which will be filled with the new events

Returns

true if fetching the data was successfully

8.16.3.4 std::string FileReader::getLastMarker ( )

return the last processed marker from the file.

Returns

the last marker from the file

8.16.3.5 bool FileReader::hasData ( )

Checks if there are still events in the FileReader

Returns

false if no more events can be read from this FileReader

8.16.4 Friends And Related Function Documentation

8.16.4.1 friend class FileReaderImpl `[friend]`

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

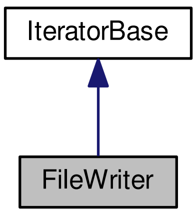
- Iterators.h

8.17 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="")
- void setMaxFileSize (uint64_t max_file_size)
- uint64_t getMaxFileSize ()
- uint64_t getTotalEvents ()
- uint64_t getTotalSize ()
- void setMarker (const std::string &marker)

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

Friends

- class FileWriterImpl

Additional Inherited Members

8.17.1 Detailed Description

compresses and stores all time tags to a file

8.17.2 Constructor & Destructor Documentation

8.17.2.1 FileWriter::FileWriter (TimeTaggerBase ∗ *tagger,* const std::string & *filename,* std::vector< channel_t > *channels*)

constructor of a FileWriter

Parameters

tagger	reference to a TimeTagger
filename	name of the file to store to
channels	channels which are stored to the file

8.17.2.2 FileWriter::∼FileWriter ( )

8.17.3 Member Function Documentation

8.17.3.1 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.

8.17.3.2 uint64_t FileWriter::getMaxFileSize ( )

fetches the maximum file size. Please see setMaxFileSize for more details.

Returns

the maximum file size in bytes

8.17.3.3 uint64_t FileWriter::getTotalEvents ( )

queries the total amount of events stored in all files

Returns

the total amount of events stored

8.17.3.4 uint64_t FileWriter::getTotalSize ( )

queries the total amount of bytes stored in all files

Returns

the total amount of bytes stored

8.17.3.5 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.17.3.6 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.

8.17.3.7 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.

8.17.3.8 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

marker	the marker to write into the file

8.17.3.9 void FileWriter::setMaxFileSize (uint64_t *max_file_size*)

Set the maximum file size on disk and so when the automatical split happens. Note: This is a rough limit, the actual file might be larger by one block.

Parameters

max_file_size	new maximum file size in bytes

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

Close the current file and create a new one

Parameters

new_filename	filename of the new file. If empty, the old one will be used.

8.17.4 Friends And Related Function Documentation

8.17.4.1 friend class FileWriterImpl `[friend]`

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

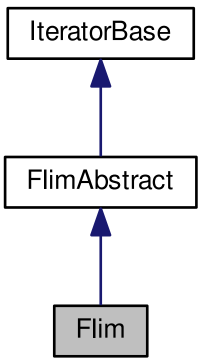
- Iterators.h

8.18 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=CHANN↩ EL_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 variaty 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 for.
- 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

8.18.1 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.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 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 variaty of high-level functionality

Parameters

tagger	reference to a TimeTagger
start_channel	channel on which start clicks are received for the time differences histogramming
click_channel	channel on which clicks are received for the time differences histogramming
pixel_begin_channel	start of a pixel (histogram)
n_pixels	number of pixels (histograms) of one frame
n_bins	number of histogram bins for each pixel
binwidth	bin size in picoseconds
pixel_end_channel	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
frame_begin_channel	(optional) start the frame, or reset the pixel index
finish_after_outputframe	(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.
n_frame_average	(optional) average multiple input frames into one output frame, default: 1
pre_initialize	(optional) initializes the measurement on constructing.

8.18.2.2 Flim::∼Flim ( )

8.18.3 Member Function Documentation

8.18.3.1 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.

8.18.3.2 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]

8.18.3.3 uint32_t Flim::get_ready_index (int32_t *index*) [protected]

8.18.3.4 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

8.18.3.5 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

8.18.3.6 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.

8.18.3.7 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.

8.18.3.8 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.

8.18.3.9 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 >= finish_after_outputframe, it will throw an error.

Parameters

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

8.18.3.10 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 >= finish_after_outputframe, it will throw an error.

Parameters

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

8.18.3.11 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 >= 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

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

8.18.3.12 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

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

8.18.3.13 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 for.

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

Parameters

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

8.18.3.14 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

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

8.18.3.15 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.

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

Implements FlimAbstract.

8.18.4 Member Data Documentation

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

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

8.18.4.3 uint32_t Flim::captured_frames `[protected]`

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

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

8.18.4.6 int32_t Flim::last_frame [protected]

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

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

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

8.18.4.10 uint32_t Flim::total_frames [protected]

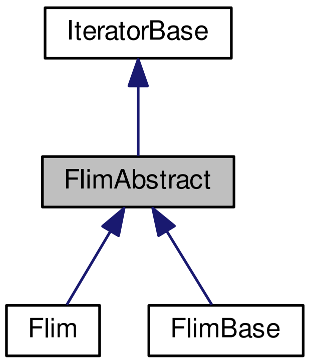
The documentation for this class was generated from the following file:

- Iterators.h

8.19 FlimAbstract Class Reference

`#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 aquisition 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

8.19.1 Constructor & Destructor Documentation

8.19.1.1 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

tagger	reference to a TimeTagger
start_channel	channel on which start clicks are received for the time differences histogramming
click_channel	channel on which clicks are received for the time differences histogramming
pixel_begin_channel	start of a pixel (histogram)
n_pixels	number of pixels (histograms) of one frame
n_bins	number of histogram bins for each pixel
binwidth	bin size in picoseconds
pixel_end_channel	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
frame_begin_channel	(optional) start the frame, or reset the pixel index
finish_after_outputframe	(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.
n_frame_average	(optional) average multiple input frames into one output frame, default: 1
pre_initialize	(optional) initializes the measurement on constructing.

8.19.1.2 FlimAbstract::∼FlimAbstract ( )

8.19.2 Member Function Documentation

8.19.2.1 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.

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

tells if the data aquisition has finished reaching finish_after_outputframe

This function returns a boolean which tells the user if the class is still aquiring 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.

8.19.2.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

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

Implemented in Flim, and FlimBase.

8.19.2.5 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.

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

8.19.3 Member Data Documentation

8.19.3.1 bool FlimAbstract::acquiring {} [protected]

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

8.19.3.3 FastBinning FlimAbstract::binner [protected]

8.19.3.4 const timestamp_t FlimAbstract::binwidth [protected]

8.19.3.5 const channel_t FlimAbstract::click_channel [protected]

8.19.3.6 timestamp_t FlimAbstract::current_frame_begin [protected]

8.19.3.7 timestamp_t FlimAbstract::current_frame_end [protected]

8.19.3.8 size_t FlimAbstract::data_base {} [protected]

8.19.3.9 const uint32_t FlimAbstract::finish_after_outputframe [protected]

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

8.19.3.11 bool FlimAbstract::frame_acquisition {} [protected]

8.19.3.12 const channel_t FlimAbstract::frame_begin_channel [protected]

8.19.3.13 uint32_t FlimAbstract::frames_completed {} [protected]

8.19.3.14 bool FlimAbstract::initialized [protected]

8.19.3.15 const uint32_t FlimAbstract::n_bins [protected]

8.19.3.16 const uint32_t FlimAbstract::n_frame_average [protected]

8.19.3.17 const uint32_t FlimAbstract::n_pixels [protected]

8.19.3.18 bool FlimAbstract::pixel_acquisition {} [protected]

8.19.3.19 const channel_t FlimAbstract::pixel_begin_channel [protected]

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

8.19.3.21 const channel_t FlimAbstract::pixel_end_channel [protected]

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

8.19.3.23 uint32_t FlimAbstract::pixels_processed {} [protected]

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

8.19.3.25 const channel_t FlimAbstract::start_channel [protected]

8.19.3.26 uint32_t FlimAbstract::ticks {} [protected]

8.19.3.27 const timestamp_t FlimAbstract::time_window [protected]

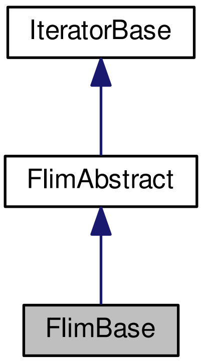
The documentation for this class was generated from the following file:

- Iterators.h

8.20 FlimBase Class Reference

`#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 efficienty 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

8.20.1 Constructor & Destructor Documentation

8.20.1.1 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 efficienty purposes

Parameters

tagger	reference to a TimeTagger
start_channel	channel on which start clicks are received for the time differences histogramming
click_channel	channel on which clicks are received for the time differences histogramming
pixel_begin_channel	start of a pixel (histogram)
n_pixels	number of pixels (histograms) of one frame
n_bins	number of histogram bins for each pixel
binwidth	bin size in picoseconds
pixel_end_channel	end marker of a pixel - incoming clicks on the click_channel will be ignored afterwards
frame_begin_channel	(optional) start the frame, or reset the pixel index
finish_after_outputframe	(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.
n_frame_average	(optional) average multiple input frames into one output frame, default: 1
pre_initialize	(optional) initializes the measurement on constructing.

8.20.1.2 FlimBase::∼FlimBase ( )

8.20.2 Member Function Documentation

8.20.2.1 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]

8.20.2.2 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.

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

Implements FlimAbstract.

8.20.3 Member Data Documentation

8.20.3.1 uint32_t FlimBase::total_frames [protected]

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

- Iterators.h

8.21 FlimFrameInfo Class Reference

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

Public Member Functions

- int32_t getFrameNumber () const

    index of this frame
- bool isValid () const

    tells if this frame is valid
- uint32_t getPixelPosition () const

    number of pixels aquired 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

Friends

- class Flim

8.21.1 Member Function Documentation

8.21.1.1 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.

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

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

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

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

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

number of pixels aquired on this frame

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

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

8.21.1.8 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.

8.21.2 Friends And Related Function Documentation

8.21.2.1 friend class Flim [friend]

8.21.3 Member Data Documentation

8.21.3.1 uint32_t FlimFrameInfo::bins

8.21.3.2 int32_t FlimFrameInfo::frame_number

8.21.3.3 uint32_t FlimFrameInfo::pixel_position

8.21.3.4 uint32_t FlimFrameInfo::pixels

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

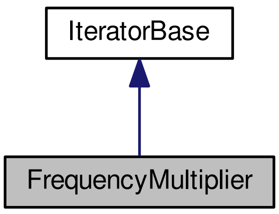
- Iterators.h

8.22 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

Friends

- class FrequencyMultiplierImpl

Additional Inherited Members

8.22.1 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 reduces 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.

8.22.2 Constructor & Destructor Documentation

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

constructor of a FrequencyMultiplier

Parameters

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

8.22.2.2 FrequencyMultiplier::∼FrequencyMultiplier ( )

8.22.3 Member Function Documentation

8.22.3.1 channel_t FrequencyMultiplier::getChannel ( )

8.22.3.2 int32_t FrequencyMultiplier::getMultiplier ( )

8.22.3.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.22.4 Friends And Related Function Documentation

8.22.4.1 friend class FrequencyMultiplierImpl `[friend]`

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

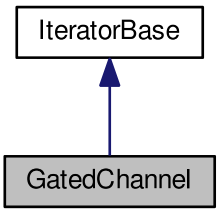
- Iterators.h

8.23 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

Friends

- class GatedChannelImpl

Additional Inherited Members

8.23.1 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.

8.23.2 Constructor & Destructor Documentation

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

constructor of a GatedChannel

Parameters

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

8.23.2.2 GatedChannel::∼GatedChannel ( )

8.23.3 Member Function Documentation

8.23.3.1 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.

8.23.3.2 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.23.4 Friends And Related Function Documentation

8.23.4.1 friend class GatedChannelImpl [friend]

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

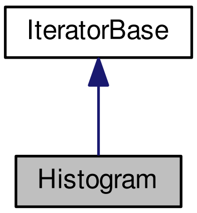
- Iterators.h

8.24 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_UNU↩SED, 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

Friends

- class TimeDifferencesImpl< Histogram >

Additional Inherited Members

8.24.1 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 'Time↩ Differences' 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.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 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

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

8.24.2.2 Histogram::∼Histogram ( )

8.24.3 Member Function Documentation

8.24.3.1 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.

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

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

8.24.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.24.3.5 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.

8.24.4 Friends And Related Function Documentation

8.24.4.1 friend class **TimeDifferencesImpl**< **Histogram** > `[friend]`

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

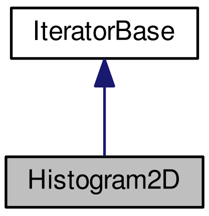
- Iterators.h

8.25 Histogram2D Class Reference

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

`#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.

Friends

- class Histogram2DImpl

Additional Inherited Members

8.25.1 Detailed Description

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

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.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 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

tagger	time tagger object
start_channel	channel on which start clicks are received
stop_channel↩ _1	channel on which stop clicks for the time axis 1 are received
stop_channel↩ _2	channel on which stop clicks for the time axis 2 are received
binwidth_1	bin width in ps for the time axis 1
binwidth_2	bin width in ps for the time axis 2
n_bins_1	the number of bins along the time axis 1
n_bins_2	the number of bins along the time axis 2

8.25.2.2 Histogram2D::∼Histogram2D ( )

8.25.3 Member Function Documentation

8.25.3.1 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.

8.25.3.2 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.

8.25.3.3 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

8.25.3.4 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.

8.25.3.5 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.

8.25.3.6 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.25.4 Friends And Related Function Documentation

8.25.4.1 friend class Histogram2DImpl `[friend]`

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

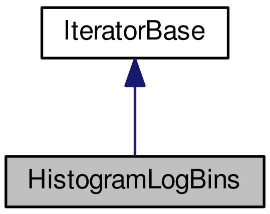
- Iterators.h

8.26 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)
- void getDataNormalizedCountsPerPs (std::function< double ∗(size_t)> array_out)
- void getDataNormalizedG2 (std::function< double ∗(size_t)> array_out)
- void getBinEdges (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.

Friends

- class HistogramLogBinsImpl

Additional Inherited Members

8.26.1 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.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 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

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

8.26.2.2 HistogramLogBins::∼HistogramLogBins ( )

8.26.3 Member Function Documentation

8.26.3.1 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.

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

returns the edges of the bins in ps

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

returns the absolute counts for the bins

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

returns the counts normalized by the binwidth of each bin

8.26.3.5 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

8.26.3.6 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.26.4 Friends And Related Function Documentation

8.26.4.1 friend class HistogramLogBinsImpl `[friend]`

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

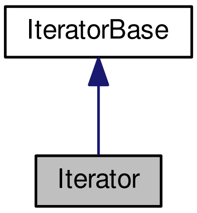
- Iterators.h

8.27 Iterator Class Reference

a 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.

Friends

- class IteratorImpl

Additional Inherited Members

8.27.1 Detailed Description

a simple event queue

A simple Iterator, just keeping a first-in first-out queue of event timestamps.

Deprecated use TimeTagStream

8.27.2 Constructor & Destructor Documentation

8.27.2.1 Iterator::Iterator (TimeTaggerBase ∗ *tagger*, channel_t *channel*)

standard constructor

Parameters

tagger	the backend
channel	the channel to get events from

8.27.2.2 Iterator::∼Iterator ( )

8.27.3 Member Function Documentation

8.27.3.1 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.

8.27.3.2 timestamp_t Iterator::next ( )

get next timestamp

get the next timestamp from the queue.

8.27.3.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.27.3.4 uint64_t Iterator::size ( )

get queue size

8.27.4 Friends And Related Function Documentation

8.27.4.1 friend class IteratorImpl `[friend]`

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

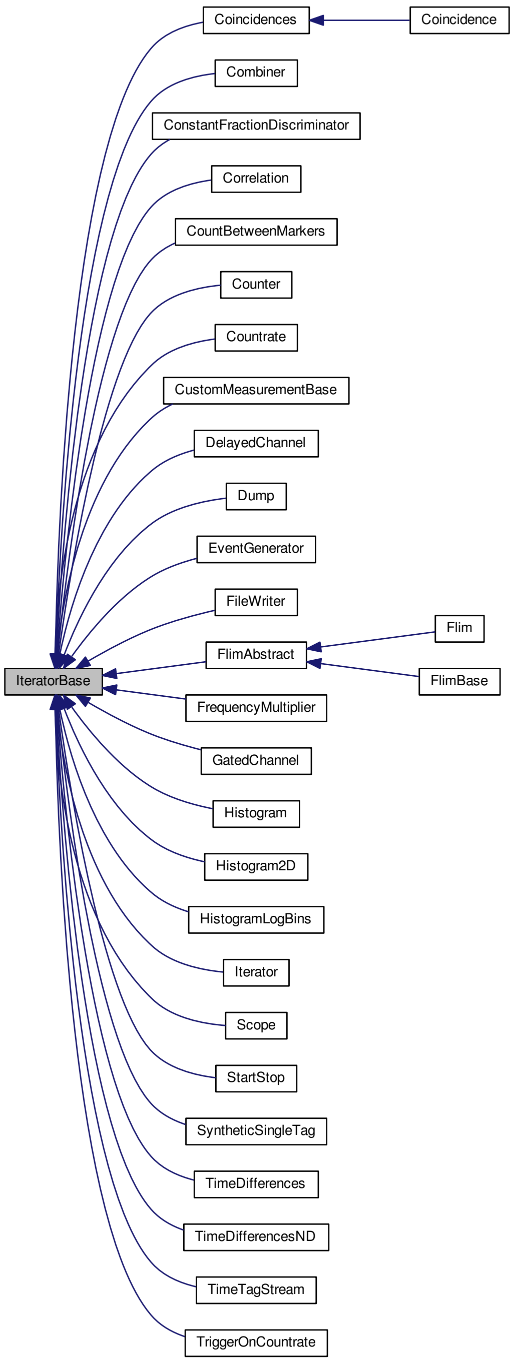
- Iterators.h

8.28 IteratorBase Class Reference

Base class for all iterators.

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

Inheritance diagram for IteratorBase:



Public Member Functions

- virtual ∼IteratorBase ()

  destructor
- void start ()

  start the iterator
- void startFor (timestamp_t capture_duration, bool clear=true)

start the iterator, and stops it after the capture_duration
- bool waitUntilFinished (int64_t timeout=-1)

  wait until the iterator has finished running.
- void stop ()

  stop the iterator
- void clear ()

  clear Iterator state.
- bool isRunning ()

  query the Iterator state.
- timestamp_t getCaptureDuration ()

  query the evaluation time

Protected Member Functions

- IteratorBase (TimeTaggerBase ∗tagger, std::string base_type_="IteratorBase", std::string extra_info_="")

  standard constructor
- 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 ()

  aquire 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 ()

  aquire 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 ()

Protected Attributes

- std::set< channel_t > channels_registered

  list of channels used by the iterator
- bool running

  running state of the iterator
- bool autostart
- TimeTaggerBase ∗ tagger
- timestamp_t capture_duration

Friends

- class TimeTaggerRunner
- class TimeTaggerProxy
- class SynchronizedMeasurements

8.28.1 Detailed Description

Base class for all iterators.

8.28.2 Constructor & Destructor Documentation

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

standard constructor

will register with the TimeTagger backend.

8.28.2.2 virtual IteratorBase::∼IteratorBase ( ) [virtual]

destructor

will stop and unregister prior finalization.

8.28.3 Member Function Documentation

8.28.3.1 void IteratorBase::clear ( )

clear Iterator state.

8.28.3.2 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 Flim, FlimAbstract, CustomMeasurementBase, EventGenerator, FileWriter, Scope, Correlation, HistogramLogBins, Histogram, TimeDifferencesND, Histogram2D, TimeDifferences, StartStop, Dump, TimeTag↩ Stream, Iterator, Countrate, Counter, CountBetweenMarkers, and Combiner.

8.28.3.3 void IteratorBase::finish_running ( ) [protected]

8.28.3.4 void IteratorBase::finishInitialization ( ) [protected]

method to call after finishing the initialization of the measurement

8.28.3.5 timestamp_t IteratorBase::getCaptureDuration ( )

query the evaluation time

Query the total capture duration since the last call to clear. This might have a wrong amount of time if there were some overflows within this range.

Returns

capture duration of the data

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

aquire update lock

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

Returns

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

8.28.3.7 channel_t IteratorBase::getNewVirtualChannel ( ) [protected]

allocate a new virtual output channel for this iterator

8.28.3.8 bool IteratorBase::isRunning ( )

query the Iterator state.

Fetches if this iterator is running.

8.28.3.9 void IteratorBase::lock ( ) [protected]

aquire update lock

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

Deprecated use getLock

8.28.3.10 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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 SyntheticSingleTag, FlimAbstract, CustomMeasurementBase, EventGenerator, FileWriter, ConstantFractionDiscriminator, Scope, Correlation, HistogramLogBins, Histogram, TimeDifferencesND, Histogram2D, TimeDifferences, StartStop, Dump, TimeTagStream, Iterator, FrequencyMultiplier, GatedChannel, TriggerOnCountrate, DelayedChannel, Countrate, Coincidences, Counter, CountBetweenMarkers, and Combiner.

8.28.3.11 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 FlimAbstract, CustomMeasurementBase, EventGenerator, FileWriter, ConstantFractionDiscriminator, Histogram, TimeDifferencesND, TimeDifferences, StartStop, Dump, TriggerOnCountrate, DelayedChannel, Countrate, and Counter.

8.28.3.12 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.

8.28.3.13 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.

8.28.3.14 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

channel	the channel

8.28.3.15 void IteratorBase::start ( )

start the iterator

The default behavior for iterators is to start automatically on creation.

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

start the iterator, and stops it after the capture_duration

Parameters

capture_duration	capture duration until the meassurement is stopped
clear	resets the data aquired

When the startFor is called before the previous measurement has ended and the clear parameter is set to false, then the passed capture_duration will be added on top to the current max_capture_duration

8.28.3.17 void IteratorBase::stop ( )

stop the iterator

The iterator is put into the STOPPED state, but will still be registered with the backend.

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

release update lock

see lock()

Deprecated use getLock

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

unregister a channel

Parameters

channel	the channel

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

wait until the iterator has finished running.

Parameters

timeout	time in milliseconds to wait for the measurements. If negative, wait until finished.

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 indefinetly, it will log an error and return inmediatly.

8.28.4 Friends And Related Function Documentation

8.28.4.1 friend class SynchronizedMeasurements `[friend]`

8.28.4.2 friend class TimeTaggerProxy `[friend]`

8.28.4.3 friend class TimeTaggerRunner `[friend]`

8.28.5 Member Data Documentation

8.28.5.1 bool IteratorBase::autostart `[protected]`

8.28.5.2 timestamp_t IteratorBase::capture_duration `[protected]`

8.28.5.3 std::set<channel_t> IteratorBase::channels_registered `[protected]`

list of channels used by the iterator

8.28.5.4 bool IteratorBase::running `[protected]`

running state of the iterator

8.28.5.5 TimeTaggerBase* IteratorBase::tagger `[protected]`

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

- TimeTagger.h

8.29 OrderedBarrier Class Reference

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

Classes

- class OrderInstance

Public Member Functions

- OrderedBarrier ()
- ∼OrderedBarrier ()
- OrderInstance queue ()
- void waitUntilFinished ()

Friends

- class OrderInstance

8.29.1 Constructor & Destructor Documentation

8.29.1.1 OrderedBarrier::OrderedBarrier ( )

8.29.1.2 OrderedBarrier::∼OrderedBarrier ( )

8.29.2 Member Function Documentation

8.29.2.1 OrderInstance OrderedBarrier::queue ( )

8.29.2.2 void OrderedBarrier::waitUntilFinished ( )

8.29.3 Friends And Related Function Documentation

8.29.3.1 friend class OrderInstance `[friend]`

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

- TimeTagger.h

8.30 OrderedPipeline Class Reference

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

Public Member Functions

- OrderedPipeline ()
- ∼OrderedPipeline ()

Friends

- class IteratorBase

8.30.1 Constructor & Destructor Documentation

8.30.1.1 OrderedPipeline::OrderedPipeline ( )

8.30.1.2 OrderedPipeline::∼OrderedPipeline ( )

8.30.2 Friends And Related Function Documentation

8.30.2.1 friend class **IteratorBase** `[friend]`

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

- TimeTagger.h

8.31 OrderedBarrier::OrderInstance Class Reference

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

Public Member Functions

- OrderInstance ()
- OrderInstance (OrderedBarrier ∗parent, uint64_t instance_id)
- ∼OrderInstance ()
- void sync ()
- void release ()

Friends

- class OrderedBarrier

8.31.1 Constructor & Destructor Documentation

8.31.1.1 OrderedBarrier::OrderInstance::OrderInstance ( )

8.31.1.2 OrderedBarrier::OrderInstance::OrderInstance (**OrderedBarrier** ∗ *parent*, uint64_t *instance_id*)

8.31.1.3 OrderedBarrier::OrderInstance::∼OrderInstance ( )

8.31.2 Member Function Documentation

8.31.2.1 void OrderedBarrier::OrderInstance::release ( )

8.31.2.2 void OrderedBarrier::OrderInstance::sync ( )

8.31.3 Friends And Related Function Documentation

8.31.3.1 friend class **OrderedBarrier** `[friend]`

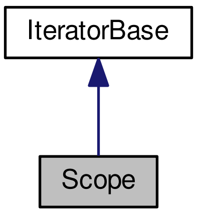
The documentation for this class was generated from the following file:

- TimeTagger.h

8.32 Scope Class Reference

```
#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.

Friends

- class ScopeImpl

Additional Inherited Members

8.32.1 Constructor & Destructor Documentation

8.32.1.1 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

tagger	reference to a TimeTagger
event_channels	channels which are captured
trigger_channel	channel that starts a new trace
window_size	window time of each trace
n_traces	amount of traces (n_traces $<$ 1, automatic retrigger)
n_max_events	maximum number of tags in each trace

8.32.1.2 Scope::∼Scope ( )

8.32.2 Member Function Documentation

8.32.2.1 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.

8.32.2.2 std::vector<std::vector<Event> > Scope::getData ( )

8.32.2.3 timestamp_t Scope::getWindowSize ( )

8.32.2.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.32.2.5 bool Scope::ready ( )

8.32.2.6 int32_t Scope::triggered ( )

8.32.3 Friends And Related Function Documentation

8.32.3.1 friend class ScopeImpl [friend]

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

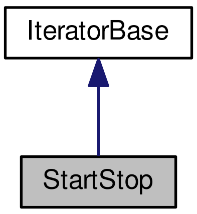
- Iterators.h

8.33 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_UNU↩ SED, 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

Friends

- class StartStopImpl

Additional Inherited Members

8.33.1 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.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 StartStop::StartStop (**TimeTaggerBase** ∗ *tagger,* **channel_t** *click_channel,* **channel_t** *start_channel =* **CHANNEL_UNUSED***,* **timestamp_t** *binwidth =* `1000`)

constructor of StartStop

Parameters

tagger	reference to a TimeTagger
click_channel	channel for stop clicks
start_channel	channel for start clicks
binwidth	width of one histogram bin in ps

8.33.2.2 StartStop::∼StartStop ( )

8.33.3 Member Function Documentation

8.33.3.1 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.

8.33.3.2 void StartStop::getData (std::function< long long *(size_t, size_t)> *array_out*)

8.33.3.3 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.33.3.4 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.

8.33.4 Friends And Related Function Documentation

8.33.4.1 friend class StartStopImpl `[friend]`

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

- Iterators.h

8.34 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 ()

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

Friends

- class TimeTaggerProxy

8.34.1 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 Synchronized↩ Measurements 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.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 SynchronizedMeasurements::SynchronizedMeasurements (TimeTaggerBase ∗ *tagger*)

construct a SynchronizedMeasurements object

Parameters

tagger	reference to a TimeTagger

8.34.2.2 SynchronizedMeasurements::∼SynchronizedMeasurements ( )

8.34.3 Member Function Documentation

8.34.3.1 void SynchronizedMeasurements::clear ( )

clear all registered measurements synchronously

8.34.3.2 TimeTaggerBase∗ SynchronizedMeasurements::getTagger ( )

Returns a proxy tagger object, which shall be used to create immediately registered measurements. Those measurements will not start automatically.

8.34.3.3 bool SynchronizedMeasurements::isRunning ( )

check if any iterator is running

8.34.3.4 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.

8.34.3.5 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.

8.34.3.6 void SynchronizedMeasurements::start ( )

start all registered measurements synchronously

8.34.3.7 void SynchronizedMeasurements::startFor (timestamp_t *capture_duration*, bool *clear* = `true`)

start all registered measurements synchronously, and stops them after the capture_duration

8.34.3.8 void SynchronizedMeasurements::stop ( )

stop all registered measurements synchronously

8.34.3.9 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.

8.34.3.10 bool SynchronizedMeasurements::waitUntilFinished (int64_t *timeout* = `-1`)

wait until all registered measurements have finished running.

Parameters

timeout	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 indefinetly, it will log an error and return inmediatly.

8.34.4 Friends And Related Function Documentation

8.34.4.1 friend class TimeTaggerProxy `[friend]`

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

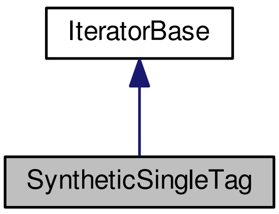
- Iterators.h

8.35 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

Friends

- class SyntheticSingleTagImpl

Additional Inherited Members

8.35.1 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.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 SyntheticSingleTag::SyntheticSingleTag (**TimeTaggerBase** ∗ *tagger,* **channel_t** *base_channel =* **CHANNEL_UNUSED**)

Construct a pulse event generator.

Parameters

tagger	reference to a TimeTagger
base_channel	base channel to which this signal will be added. If unused, a new channel will be created.

8.35.2.2 SyntheticSingleTag::∼SyntheticSingleTag ( )

8.35.3 Member Function Documentation

8.35.3.1 channel_t SyntheticSingleTag::getChannel ( ) const

8.35.3.2 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.35.3.3 void SyntheticSingleTag::trigger ( )

Generate a timetag for each call of this method.

8.35.4 Friends And Related Function Documentation

8.35.4.1 friend class SyntheticSingleTagImpl [friend]

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

- Iterators.h

8.36 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 }

Public Attributes

- enum Tag::Type type
- char reserved
- unsigned short missed_events
- channel_t channel
- timestamp_t time

8.36.1 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.

8.36.2 Member Enumeration Documentation

8.36.2.1 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 Overflow↵ End: this marks the end of the interval. All events, which were lost in this interval, have been handled Missed↵ Events: 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

8.36.3 Member Data Documentation

8.36.3.1 channel_t Tag::channel

8.36.3.2 unsigned short Tag::missed_events

8.36.3.3 char Tag::reserved

8.36.3.4 timestamp_t Tag::time

8.36.3.5 enum Tag::Type Tag::type

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

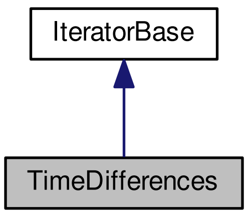
- TimeTagger.h

8.37 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

Friends

- class TimeDifferencesImpl< TimeDifferences >

Additional Inherited Members

8.37.1 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.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 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

tagger	reference to a TimeTagger
click_channel	channel that increments the count in a bin
start_channel	channel that sets start times relative to which clicks on the click channel are measured
next_channel	channel that increments the histogram index
sync_channel	channel that resets the histogram index to zero
binwidth	width of one histogram bin in ps
n_bins	number of bins in each histogram
n_histograms	number of histograms

8.37.2.2 TimeDifferences::∼TimeDifferences ( )

8.37.3 Member Function Documentation

8.37.3.1 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.

8.37.3.2 uint64_t TimeDifferences::getCounts ( )

returns the number of rollovers (histogram index resets)

8.37.3.3 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

8.37.3.4 void TimeDifferences::getIndex (std::function< long long ∗(size_t)> *array_out*)

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

8.37.3.5 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.37.3.6 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.

8.37.3.7 bool TimeDifferences::ready ( )

returns 'true' when the required number of rollovers set by 'setMaxCounts' has been reached

8.37.3.8 void TimeDifferences::setMaxCounts (uint64_t *max_counts*)

set the number of rollovers at which the measurement stops integrating

Parameters

max_counts	maximum number of sync/next clicks

8.37.4 Friends And Related Function Documentation

8.37.4.1 friend class TimeDifferencesImpl< TimeDifferences > [friend]

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

- Iterators.h

8.38 TimeDifferencesImpl< T > Class Template Reference

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

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

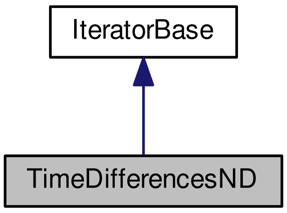
- Iterators.h

8.39 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↩ _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_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

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

Friends

- class TimeDifferencesNDImpl

Additional Inherited Members

8.39.1 Detailed Description

Accumulates the time differences between clicks on two channels in a multi-dimensional histogram.

This is a multidimensional implementation of the TimeDifferences measurement class. Please read their documentation first.

This measurement class extends the TimeDifferences interface for a multidimensional amount of histograms. It captures many multiple start - multiple stop histograms, but with many asynchronous next_channel triggers. After each tag on each next_channel, the histogram index of the associated dimension is incremented by one and reset to zero after reaching the last valid index. The elements of the parameter n_histograms specifies the number of histograms per dimension. The accumulation starts when next_channel has been triggered on all dimensions.

You should provide a synchronization trigger by specifying a sync_channel per dimension. It will stop the accumulation when an associated histogram index rollover occurs. A sync event will also stop the accumulation, reset the histogram index of the associated dimension, and a subsequent event on the corresponding next_channel starts the accumulation again. The synchronization is done asynchronous, so an event on the next_channel increases the histogram index even if the accumulation is stopped. The accumulation starts when a tag on the sync_channel arrives with a subsequent tag on next_channel for all dimensions.

Please use setInputDelay to adjust the latency of all channels. In general, the order of the provided triggers including maximum jitter should be: old start trigger – all sync triggers – all next triggers – new start trigger

8.39.2 Constructor & Destructor Documentation

8.39.2.1 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

tagger	reference to a TimeTagger
click_channel	channel that increments the count in a bin
start_channel	channel that sets start times relative to which clicks on the click channel are measured
next_channels	vector of channels that increments the histogram index
sync_channels	vector of channels that resets the histogram index to zero
n_histograms	vector of numbers of histograms per dimension.
binwidth	width of one histogram bin in ps
n_bins	number of bins in each histogram

8.39.2.2 TimeDifferencesND::∼TimeDifferencesND ( )

8.39.3 Member Function Documentation

8.39.3.1 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.

8.39.3.2 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

8.39.3.3 void TimeDifferencesND::getIndex (std::function< long long *(size_t)> *array_out*)

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

8.39.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.39.3.5 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.

8.39.4 Friends And Related Function Documentation

8.39.4.1 friend class TimeDifferencesNDImpl `[friend]`

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

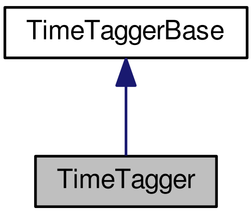
- Iterators.h

8.40 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 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 minumum 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 This method is not supported any more on the Time Tagger Ultra series
- 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 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)

Additional Inherited Members

8.40.1 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.

8.40.2 Member Function Documentation

8.40.2.1 virtual void TimeTagger::autoCalibration (std::function< double ∗(size_t)> *array_out*) [pure virtual]

runs a calibrations based on the on-chip uncorrelated signal generator.

8.40.2.2 virtual void TimeTagger::clearConditionalFilter ( ) [pure virtual]

deactivates the conditional filter

equivilent to setConditionalFilter({},{})

8.40.2.3 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.

8.40.2.4 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_CHA↩ NNEL_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_CH↩ ANNEL_FALLING_EDGES resturn only the falling edges SCHEME_ONE: {-1,-2,-3,-4,-5,-6,-7,-8} which are the invertedChannels of the rising edges.

8.40.2.5 virtual int TimeTagger::getChannelNumberScheme ( ) [pure virtual]

Fetch the configured numbering scheme for this TimeTagger object.

Please see setTimeTaggerChannelNumberScheme() for details.

8.40.2.6 virtual std::vector<channel_t> TimeTagger::getConditionalFilterFiltered ( ) [pure virtual]

fetches the configuration of the conditional filter

see setConditionalFilter

8.40.2.7 virtual std::vector<channel_t> TimeTagger::getConditionalFilterTrigger ( ) [pure virtual]

fetches the configuration of the conditional filter

see setConditionalFilter

8.40.2.8 virtual std::vector<double> TimeTagger::getDACRange ( ) [pure virtual]

returns the minumum and the maximum voltage of the DACs as a trigger reference

8.40.2.9 virtual void TimeTagger::getDistributionCount (std::function< uint64_t *(size_t, size_t)> *array_out*) [pure virtual]

get internal calibration data

8.40.2.10 virtual void TimeTagger::getDistributionPSecs (std::function< long long *(size_t, size_t)> *array_out*) [pure virtual]

get internal calibration data This method is not supported any more on the Time Tagger Ultra series

Deprecated

8.40.2.11 virtual unsigned int TimeTagger::getEventDivider (channel_t *channel*) [pure virtual]

Returns the factor of the dividing filter.

See setEventDivider for further details.

Parameters

channel	channel to be queried

Returns

the configured divider

8.40.2.12 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 indentifier of the USB chip: OK 1.30 eg "TT-Ultra, FW 3, TS 2018-11-13 22:57:32, OK 1.30"

8.40.2.13 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

8.40.2.14 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

channel	the channel

Returns

the hardware delay compensation in picoseconds

8.40.2.15 virtual int TimeTagger::getInputMux (channel_t *channel*) [pure virtual]

fetches the configuration of the input multiplexer

Parameters

channel	the phyiscal channel of the input multiplexer

Returns

the configuration mode of the input multiplexer

8.40.2.16 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

8.40.2.17 virtual std::string TimeTagger::getModel ( ) [pure virtual]

identifies the hardware by Time Tagger Model

8.40.2.18 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

channel	the channel to query

Returns

if the normalization is enabled

8.40.2.19 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.

8.40.2.20 virtual timestamp_t TimeTagger::getPsPerClock ( ) [pure virtual]

fetch the duration of each clock cycle in picoseconds

8.40.2.21 virtual std::string TimeTagger::getSensorData ( ) [pure virtual]

Show the status of the sensor data from the FPGA and peripherals on the console.

8.40.2.22 virtual std::string TimeTagger::getSerial ( ) [pure virtual]

identifies the hardware by serial number

8.40.2.23 virtual int TimeTagger::getStreamBlockSizeEvents ( ) [pure virtual]

8.40.2.24 virtual int TimeTagger::getStreamBlockSizeLatency ( ) [pure virtual]

8.40.2.25 virtual int TimeTagger::getTestSignalDivider ( ) [pure virtual]

get the divider for the frequency of the test signal

8.40.2.26 virtual double TimeTagger::getTriggerLevel (channel_t *channel*) [pure virtual]

get the trigger voltage threshold of a channel

Parameters

channel	the channel

8.40.2.27 virtual void TimeTagger::reset ( ) [pure virtual]

reset the TimeTagger object to default settings and detach all iterators

8.40.2.28 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 supressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

Parameters

trigger	the channels that sets the condition
filtered	the channels that are filtered by the condition
hardwareDelayCompensation	if false, the physical hardware delay will not be compensated

8.40.2.29 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 nth 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

channel	channel to be configured
divider	new divider, must be smaller than 65536

8.40.2.30 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

size	the maximum buffer size in events

8.40.2.31 virtual void TimeTagger::setInputMux (channel_t *channel,* int *mux_mode*) [pure virtual]

configures the input multiplexer

Every phyiscal 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

channel	the phyiscal channel of the input multiplexer
mux_mode	the configuration mode of the input multiplexer

8.40.2.32 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.

8.40.2.33 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

channel	list of channels to modify
state	the new state

8.40.2.34 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

freq_hz	the generated audio frequency

8.40.2.35 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

max_events	maximum number of events
max_latency	maximum latency in ms

8.40.2.36 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

divider	frequency divisor of the oscillator

8.40.2.37 virtual void TimeTagger::setTriggerLevel (channel_t *channel,* double *voltage*) [pure virtual]

set the trigger voltage threshold of a channel

Parameters

channel	the channel to set
voltage	voltage level.. [0..1]

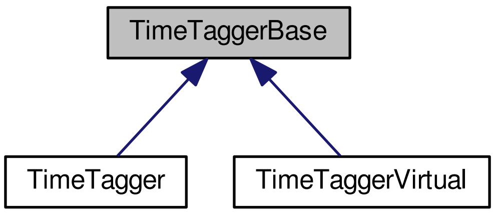
The documentation for this class was generated from the following file:

- TimeTagger.h

8.41 TimeTaggerBase Class Reference

```
#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
- virtual bool waitForFence (unsigned int fence, int64_t timeout=-1)=0
- virtual bool sync (int64_t timeout=-1)=0
- 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
- 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 the calibration on a channel.
- virtual void setTestSignal (std::vector< channel_t > channel, bool enabled)=0
- virtual bool getTestSignal (channel_t channel)=0

  fetch the status of the test signal generator
- 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 ()
- 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

Friends

- class IteratorBase
- class TimeTaggerProxy
- class TimeTaggerRunner

8.41.1 Member Typedef Documentation

8.41.1.1 typedef std::function<void(**IteratorBase** ∗)> **TimeTaggerBase::IteratorCallback**

8.41.1.2 typedef std::map<**IteratorBase** ∗, **IteratorCallback**> **TimeTaggerBase::IteratorCallbackMap**

8.41.2 Constructor & Destructor Documentation

8.41.2.1 TimeTaggerBase::TimeTaggerBase ( ) `[inline]`, `[protected]`

abstract interface class

8.41.2.2 virtual TimeTaggerBase::∼TimeTaggerBase ( ) `[inline]`, `[protected]`, `[virtual]`

destructor

8.41.2.3 TimeTaggerBase::TimeTaggerBase (const **TimeTaggerBase** &) `[protected]`, `[delete]`

8.41.3 Member Function Documentation

8.41.3.1 virtual void TimeTaggerBase::addChild (**TimeTaggerBase** ∗ *child*) `[protected]`, `[pure virtual]`

8.41.3.2 virtual std::shared_ptr<IteratorBaseListNode> TimeTaggerBase::addIterator (**IteratorBase** ∗ *it*) `[protected]`, `[pure virtual]`

8.41.3.3 virtual void TimeTaggerBase::clearOverflows ( ) `[pure virtual]`

clear overflow counter

Sets the overflow counter to zero

8.41.3.4 virtual void TimeTaggerBase::freeIterator (**IteratorBase** ∗ *it*) `[protected]`, `[pure virtual]`

8.41.3.5 virtual void TimeTaggerBase::freeVirtualChannel (**channel_t** *channel*) `[protected]`, `[pure virtual]`

8.41.3.6 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.

8.41.3.7 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 configureable deadtime.

Parameters

channel	channel to be queried

Returns

the real configured deadtime

8.41.3.8 virtual timestamp_t TimeTaggerBase::getDelayHardware (channel_t *channel*) [pure virtual]

get time delay of a channel

see setDelayHardware

Parameters

channel	the channel

8.41.3.9 virtual timestamp_t TimeTaggerBase::getDelaySoftware (channel_t *channel*) [pure virtual]

get time delay of a channel

see setDelaySoftware

Parameters

channel	the channel

8.41.3.10 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

alloc_fence	if false, a reference to the most recently created fence will be returned instead

Returns

the allocated fence

8.41.3.11 virtual timestamp_t TimeTaggerBase::getInputDelay (channel_t *channel*) [pure virtual]

get time delay of a channel

see setInputDelay

Parameters

channel	the channel

8.41.3.12 virtual channel_t TimeTaggerBase::getInvertedChannel (channel_t *channel*) [pure virtual]

get the falling channel id for a raising channel and vice versa

8.41.3.13 virtual channel_t TimeTaggerBase::getNewVirtualChannel ( ) [protected], [pure virtual]

8.41.3.14 virtual long long TimeTaggerBase::getOverflows ( ) [pure virtual]

get overflow count

Get the number of communication overflows occured

8.41.3.15 virtual long long TimeTaggerBase::getOverflowsAndClear ( ) [pure virtual]

get and clear overflow counter

Get the number of communication overflows occured and sets them to zero

8.41.3.16 virtual bool TimeTaggerBase::getTestSignal (channel_t *channel*) [pure virtual]

fetch the status of the test signal generator

Parameters

channel	the channel

8.41.3.17 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

8.41.3.18 TimeTaggerBase& TimeTaggerBase::operator= (const TimeTaggerBase &) [protected], [delete]

8.41.3.19 virtual void TimeTaggerBase::registerChannel (channel_t *channel*) [protected], [pure virtual]

register a FPGA channel.

Only events on previously registered channels will be transfered over the communication channel.

Parameters

channel	the channel

8.41.3.20 virtual void TimeTaggerBase::release ( ) [protected], [pure virtual]

8.41.3.21 virtual void TimeTaggerBase::removeChild (TimeTaggerBase ∗ *child*) [protected], [pure virtual]

8.41.3.22 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

callbacks	Map of callbacks per iterator
block	Shall this method block until all callbacks are finished

8.41.3.23 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 configureable 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

channel	channel to be configured
deadtime	new deadtime

Returns

the real configured deadtime

8.41.3.24 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

channel	the channel to set
delay	the delay in picoseconds

8.41.3.25 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 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

channel	the channel to set
delay	the delay in picoseconds

8.41.3.26 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

channel	the channel to set
delay	the delay in picoseconds

8.41.3.27 virtual void TimeTaggerBase::setTestSignal (channel_t *channel*, bool *enabled*) [pure virtual]

enable the calibration on a channel.

This will connect or disconnect the channel with the on-chip uncorrelated signal generator.

Parameters

channel	the channel
enabled	enabled / disabled flag

8.41.3.28 virtual void TimeTaggerBase::setTestSignal (std::vector< channel_t > *channel*, bool *enabled*) [pure virtual]

8.41.3.29 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

timeout	timeout in milliseconds. Negative means no timeout, zero returns immediately.

Returns

true on success, false on timeout

8.41.3.30 virtual void TimeTaggerBase::unregisterChannel (channel_t *channel*) [protected], [pure virtual]

release a previously registered channel.

Parameters

channel	the channel

8.41.3.31 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

fence	fence object, which shall be waited on
timeout	timeout in milliseconds. Negative means no timeout, zero returns immediately.

Returns

true if the fence has passed, false on timeout

8.41.4 Friends And Related Function Documentation

8.41.4.1 friend class IteratorBase `[friend]`

8.41.4.2 friend class TimeTaggerProxy `[friend]`

8.41.4.3 friend class TimeTaggerRunner `[friend]`

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

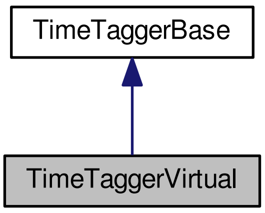
- TimeTagger.h

8.42 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

8.42.1 Detailed Description

virtual TimeTagger based on dump files

The TimeTaggerVirtual class represents a virtual Time Tagger. But instead of connecting to Swabians hardware, it replays all tags from a recorded file.

8.42.2 Member Function Documentation

8.42.2.1 virtual void TimeTaggerVirtual::clearConditionalFilter ( ) [pure virtual]

deactivates the conditional filter

equivilent to setConditionalFilter({},{})

8.42.2.2 virtual std::vector<channel_t> TimeTaggerVirtual::getConditionalFilterFiltered ( ) [pure virtual]

fetches the configuration of the conditional filter

see setConditionalFilter

8.42.2.3 virtual std::vector<**channel_t**> TimeTaggerVirtual::getConditionalFilterTrigger ( ) `[pure virtual]`

fetches the configuration of the conditional filter

see setConditionalFilter

8.42.2.4 virtual double TimeTaggerVirtual::getReplaySpeed ( ) `[pure virtual]`

fetches the speed factor

Please see setReplaySpeed for more details.

Returns

the speed factor

8.42.2.5 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 immediatelly.

Parameters

file	the file to be replayed
begin	amount of ps to skip at the begin of the file. A negativ time will generate a pause in the replay
duration	time period in ps of the file. -1 replays till the last tag
queue	flag if this file shall be queued

Returns

ID of the queued file

8.42.2.6 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.

8.42.2.7 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 supressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

Parameters

trigger	the channels that sets the condition
filtered	the channels that are filtered by the condition

8.42.2.8 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

speed	ratio of the replay speed and the real time

8.42.2.9 virtual void TimeTaggerVirtual::stop ( ) `[pure virtual]`

stops the current and all queued files.

This method stops the current file and clears the replay queue.

8.42.2.10 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

ID	selects which file to wait for
timeout	timeout in milliseconds

Returns

true if the file is complete, false on timeout

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

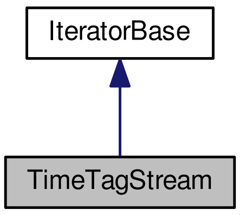
- TimeTagger.h

8.43 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=std↩ ::vector< channel_t >())

    constructor of a TimeTagStream thread
- ∼TimeTagStream ()

    tbd
- uint64_t getCounts ()

    get incoming time 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.

Friends

- class TimeTagStreamImpl

Additional Inherited Members

8.43.1 Detailed Description

access the time tag stream

8.43.2 Constructor & Destructor Documentation

8.43.2.1 TimeTagStream::TimeTagStream (**TimeTaggerBase** ∗ *tagger,* uint64_t *n_max_events,* std::vector< **channel_t** > *channels* = `std::vector<` **channel_t** `>()`)

constructor of a TimeTagStream thread

Gives access to the time tag stream

Parameters

tagger	reference to a TimeTagger
n_max_events	maximum number of tags stored
channels	channels which are dumped to the file (when empty or not passed all active channels are dumped)

8.43.2.2 TimeTagStream::∼TimeTagStream ( )

tbd

8.43.3 Member Function Documentation

8.43.3.1 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.

8.43.3.2 uint64_t TimeTagStream::getCounts ( )

get incoming time tags

All incoming time tags are stored in a buffer (max size: max_tags). The buffer is cleared after retrieving the data with getData() return the number of stored tags

8.43.3.3 **TimeTagStreamBuffer** TimeTagStream::getData ( )

fetches all stored tags and clears the internal state

8.43.3.4 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.43.4 Friends And Related Function Documentation

8.43.4.1 friend class TimeTagStreamImpl `[friend]`

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

- Iterators.h

8.44 TimeTagStreamBuffer Class Reference

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

Public Member Functions

- 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

Friends

- class TimeTagStreamImpl
- class FileReaderImpl

8.44.1 Member Function Documentation

8.44.1.1 void TimeTagStreamBuffer::getChannels (std::function< int *(size_t)> *array_out*)

8.44.1.2 void TimeTagStreamBuffer::getEventTypes (std::function< unsigned char *(size_t)> *array_out*)

8.44.1.3 void TimeTagStreamBuffer::getMissedEvents (std::function< unsigned short *(size_t)> *array_out*)

8.44.1.4 void TimeTagStreamBuffer::getOverflows (std::function< unsigned char *(size_t)> *array_out*)

8.44.1.5 void TimeTagStreamBuffer::getTimestamps (std::function< long long *(size_t)> *array_out*)

8.44.2 Friends And Related Function Documentation

8.44.2.1 friend class FileReaderImpl `[friend]`

8.44.2.2 friend class TimeTagStreamImpl `[friend]`

8.44.3 Member Data Documentation

8.44.3.1 bool TimeTagStreamBuffer::hasOverflows

8.44.3.2 uint64_t TimeTagStreamBuffer::size

8.44.3.3 timestamp_t TimeTagStreamBuffer::tGetData

8.44.3.4 timestamp_t TimeTagStreamBuffer::tStart

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

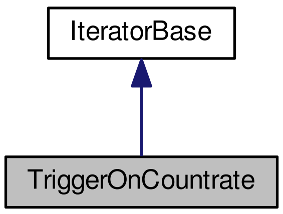
- Iterators.h

8.45 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

Friends

- class TriggerOnCountrateImpl

Additional Inherited Members

8.45.1 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`.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 TriggerOnCountrate::TriggerOnCountrate (**TimeTaggerBase** $*$ *tagger*, **channel_t** *input_channel*, double *reference_countrate*, double *hysteresis*, **timestamp_t** *time_window*)

constructor of a TriggerOnCountrate

Parameters

tagger	Reference to a TimeTagger object.
input_channel	Channel number of the channel whose count rate will control the trigger channels.
reference_countrate	The reference count rate in Hz that separates the `above` range from the `below` range.
hysteresis	The threshold count rate in Hz for transitioning to the `above` threshold state is `countrate >= reference_countrate + hysteresis`, whereas it is `countrate <= reference_countrate - hysteresis` for transitioning to the `below` threshold state. The hysteresis avoids the emission of multiple trigger tags upon a single transition.
time_window	Rolling time window size in ps. The count rate is analyzed within this time window and compared to the threshold count rate.

8.45.2.2 TriggerOnCountrate::$\sim$TriggerOnCountrate ( )

8.45.3 Member Function Documentation

8.45.3.1 channel_t TriggerOnCountrate::getChannelAbove ( )

Get the channel number of the `above` channel.

8.45.3.2 channel_t TriggerOnCountrate::getChannelBelow ( )

Get the channel number of the `below` channel.

8.45.3.3 std::vector<channel_t> TriggerOnCountrate::getChannels ( )

Get both virtual channel numbers: [`getChannelAbove()`, `getChannelBelow()`].

8.45.3.4 double TriggerOnCountrate::getCurrentCountrate ( )

Get the current count rate averaged within the `time_window`.

8.45.3.5 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.

8.45.3.6 bool TriggerOnCountrate::isAbove ( )

Returns whether the Virtual Channel is currently in the `above` state.

8.45.3.7 bool TriggerOnCountrate::isBelow ( )

Returns whether the Virtual Channel is currently in the `below` state.

8.45.3.8 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

incoming_tags	block of events
begin_time	earliest event in the block
end_time	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.

8.45.3.9 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.

8.45.4 Friends And Related Function Documentation

8.45.4.1 friend class TriggerOnCountrateImpl [friend]

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

- Iterators.h

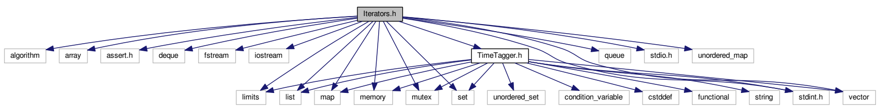
Chapter 9

File Documentation

9.1 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
- 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 Counter

a simple counter on one or more channels
- class Coincidences

  a coincidence monitor for one or more channel groups
- class Coincidence

  a coincidence monitor for one or more channel groups
- 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 simple event queue
- class TimeTagStreamBuffer
- 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 spectrocopy.
- 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

  cross-correlation between two channels
- struct Event
- class Scope
- 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
- class EventGenerator

  Generate predefined events in a virtual channel relative to a trigger event.
- class CustomMeasurementBase
- class FlimAbstract
- class FlimBase

- class FlimFrameInfo
- class Flim

  Fluorescence lifetime imaging.
- class SyntheticSingleTag

  synthetic trigger timetag generator.

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 }
- enum State { UNKNOWN, HIGH, LOW }

9.1.1 Macro Definition Documentation

9.1.1.1 #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.

9.1.2 Enumeration Type Documentation

9.1.2.1 enum **CoincidenceTimestamp** : uint32_t `[strong]`

type of timestamp for the Coincidence virtual channel (Last, Average, First, ListedFirst)

Enumerator

Last

Average

First

ListedFirst

9.1.2.2 enum **State**

Enumerator

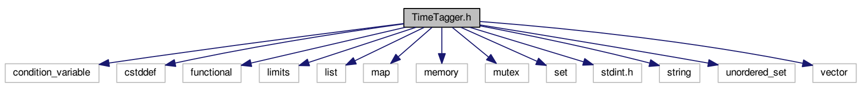
UNKNOWN

HIGH

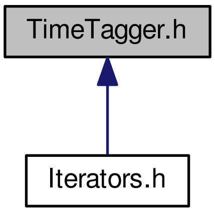
LOW

9.2 TimeTagger.h File Reference

```
#include <condition_variable>
#include <cstddef>
#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:



This graph shows which files directly or indirectly include this file:



Classes

- class CustomLogger
- class TimeTaggerBase
- class TimeTaggerVirtual

  virtual TimeTagger based on dump files
- class TimeTagger

  backend for the TimeTagger.
- struct Tag

  a single event on a channel
- class OrderedBarrier
- class OrderedBarrier::OrderInstance
- class OrderedPipeline
- class IteratorBase

  Base class for all iterators.

Macros

- #define TT_API __declspec(dllimport)
- #define timestamp_t long long
- #define channel_t int
- #define TIMETAGGER_VERSION "2.9.0"
- #define GET_DATA_1D(function_name, type, argout, attribute) attribute void function_name(std::function<type ∗(size_t)> argout)
- #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 LogMessageCensored(level, ...) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);
- #define ErrorLogCensored(...) LogMessage(LOGGER_ERROR, __VA_ARGS__);
- #define WarningLogCensored(...) LogMessage(LOGGER_WARNING, __VA_ARGS__);
- #define InfoLogCensored(...) LogMessage(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 }
- 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 LogLevel { LOGGER_ERROR = 40, LOGGER_WARNING = 30, LOGGER_INFO = 10 }
- 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, Usage↩ StatisticsStatus::CollectingAndUploading }

Functions

- TT_API std::string getVersion ()
- 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 void setCustomBitFileName (const std::string &bitFileName)

  set path and filename of the bitfile to be loeaded 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 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)
- TT_API std::string extractLicenseInfo (const std::string &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 censored, 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. Magic channel_t value to indicate an unused channel. So the iterators either have to disable this channel, or to choose a default one.
- constexpr channel_t CHANNEL_UNUSED_OLD = -1
- constexpr int TT_CHANNEL_NUMBER_SCHEME_AUTO = 0
- 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

9.2.1 Macro Definition Documentation

9.2.1.1 #define channel_t int

9.2.1.2 #define ErrorLog(...) LogMessage(LOGGER_ERROR, __VA_ARGS__);

9.2.1.3 #define ErrorLogCensored(...) LogMessage(LOGGER_ERROR, __VA_ARGS__);

9.2.1.4 #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.

9.2.1.5 #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)

9.2.1.6 #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)
```

9.2.1.7 #define GET_DATA_2D(*function_name, type, argout, attribute*) attribute void function_name(std::function<type ∗(size_t, size_t)> argout)

9.2.1.8 #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)
```

9.2.1.9 #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)
```

9.2.1.10 #define GET_DATA_3D(*function_name, type, argout, attribute*) attribute void function_name(std::function<type ∗(size_t, size_t, size_t)> argout)

9.2.1.11 #define InfoLog(*...*) LogMessage(LOGGER_INFO, __VA_ARGS__);

9.2.1.12 #define InfoLogCensored(*...*) LogMessage(LOGGER_INFO, __VA_ARGS__);

9.2.1.13 #define LogMessage(*level, ...*) LogBase(level, __FILE__, __LINE__, false, __VA_ARGS__);

9.2.1.14 #define LogMessageCensored(*level, ...*) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);

9.2.1.15 #define timestamp_t long long

9.2.1.16 #define TIMETAGGER_VERSION "2.9.0"

9.2.1.17 #define TT_API __declspec(dllimport)

9.2.1.18 #define WarningLog(...) LogMessage(LOGGER_WARNING, __VA_ARGS__);

9.2.1.19 #define WarningLogCensored(...) LogMessage(LOGGER_WARNING, __VA_ARGS__);

9.2.2 Typedef Documentation

9.2.2.1 using _Iterator = IteratorBase

9.2.2.2 typedef void(∗ logger_callback) (LogLevel level, std::string msg)

9.2.3 Enumeration Type Documentation

9.2.3.1 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

9.2.3.2 enum FrontendType : std::uint32_t [strong]

Enumerator

Undefined
WebApp
Firefly
Pyro5RPC
UserFrontend

9.2.3.3 enum LanguageUsed : std::uint32_t `[strong]`

Enumerator

Cpp
Python
Csharp
Matlab
Labview
Mathematica
Unknown

9.2.3.4 enum LogLevel

Enumerator

LOGGER_ERROR
LOGGER_WARNING
LOGGER_INFO

9.2.3.5 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

9.2.3.6 enum UsageStatisticsStatus `[strong]`

Enumerator

Disabled
Collecting
CollectingAndUploading

9.2.4 Function Documentation

9.2.4.1 TT_API TimeTagger∗ createTimeTagger (std::string *serial* = "", Resolution *resolution* = Resolution::Standard)

default constructor factory.

Parameters

serial	serial number of FPGA board to use. if empty, the first board found is used.
resolution	enum for how many channels shall be grouped.

See also

Resolution for details

9.2.4.2 TT_API TimeTaggerVirtual∗ createTimeTaggerVirtual ( )

default constructor factory for the createTimeTaggerVirtual class.

9.2.4.3 TT_API std::string extractLicenseInfo (const std::string & *license*)

Parses the binary license and return a human readable information about this license.

Parameters

license	the binary license, encoded as a hexadecimal string

Returns

a human readable string containing all information about this license

9.2.4.4 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 instancated while updating the license.

Parameters

serial	the serial of the device to update the license. Must not be empty
license	the binary license, encoded as a hexadecimal string

9.2.4.5 TT_API bool freeTimeTagger (TimeTaggerBase ∗ *tagger*)

free a copy of a TimeTagger reference.

Parameters

tagger	the TimeTagger reference to free

9.2.4.6 TT_API int getTimeTaggerChannelNumberScheme ( )

Fetch the currently configured global numbering scheme.

Please see setTimeTaggerChannelNumberScheme() for details. Please use TimeTagger::getChannelNumber↩Scheme() to query the actual used numbering scheme, this function here will just return the scheme a newly created TimeTagger object will use.

9.2.4.7 TT_API std::string getTimeTaggerModel (const std::string & *serial*)

9.2.4.8 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 pseudonymize.

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.

9.2.4.9 TT_API UsageStatisticsStatus getUsageStatisticsStatus ( )

gets the status of the usage statistics system.

Returns

the current status of the usage statistics system.

9.2.4.10 TT_API std::string getVersion ( )

9.2.4.11 TT_API bool hasTimeTaggerVirtualLicense ( )

Check if a license for the TimeTaggerVirtual is available.

9.2.4.12 TT_API void LogBase (LogLevel *level,* const char ∗ *file,* int *line,* bool *censored,* const char ∗ *fmt,* *...*)

Raise a new log message. Please use the XXXLog macro instead.

9.2.4.13 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.

9.2.4.14 TT_API void setCustomBitFileName (const std::string & *bitFileName*)

set path and filename of the bitfile to be loeaded 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

bitFileName	custom bitfile to use for the FPGA.

9.2.4.15 TT_API void setFrontend (FrontendType *frontend*)

sets the frontend being used currently for usage statistics system.

Parameters

frontend	the frontend currently being used.

9.2.4.16 TT_API void setLanguageInfo (std::uint32_t *pw*, LanguageUsed *language*, std::string *version*)

sets the language being used currently for usage statistics system.

Parameters

pw	password for authporization to change the language.
language	programming language being used.
version	version of the programming langugae being used.

9.2.4.17 TT_API logger_callback setLogger (logger_callback *callback*)

Sets the notifier callback which is called for each log message.

Returns

The old callback

If this function is called with nullptr, the default callback will be used.

9.2.4.18 TT_API void setTimeTaggerChannelNumberScheme (int *scheme*)

Configure the numbering scheme for new TimeTagger objects.

Parameters

scheme	new numbering scheme, must be TT_CHANNEL_NUMBER_SCHEME_AUTO, TT_CHANNEL_NUMBER_SCHEME_ZERO or TT_CHANNEL_NUMBER_SCHEME_ONE

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 createTime↵ Tagger().

_ZERO will typically allocate the channel numbers 0 to 7 for the 8 input channels. 8 to 15 will be allocated for the coresponding 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 coresponding falling events.

_AUTO will choose the scheme based on the hardware revision and so based on the printed label.

9.2.4.19 TT_API void setUsageStatisticsStatus (UsageStatisticsStatus *new_status*)

sets the status of the usage statistics system.

This fuinctionality allows configuring the usage statistics system.

Parameters

new_status	new status of the usage statistics system.

9.2.5 Variable Documentation

9.2.5.1 constexpr channel_t CHANNEL_UNUSED = -134217728

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.

9.2.5.2 constexpr channel_t CHANNEL_UNUSED_OLD = -1

9.2.5.3 constexpr ChannelEdge TT_CHANNEL_FALLING_EDGES = ChannelEdge::Falling

9.2.5.4 constexpr int TT_CHANNEL_NUMBER_SCHEME_AUTO = 0

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 coresponding 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 coresponding falling events.

_AUTO will choose the scheme based on the hardware revision and so based on the printed label.

9.2.5.5 constexpr int TT_CHANNEL_NUMBER_SCHEME_ONE = 2

9.2.5.6 constexpr int TT_CHANNEL_NUMBER_SCHEME_ZERO = 1

9.2.5.7 constexpr ChannelEdge TT_CHANNEL_RISING_AND_FALLING_EDGES = ChannelEdge::All

9.2.5.8 constexpr ChannelEdge TT_CHANNEL_RISING_EDGES = ChannelEdge::Rising