# TimeTagger

2.17.4.0

Generated by Doxygen 1.10.0

# Chapter 1

# TimeTagger

backend for TimeTagger, an OpalKelly based single photon counting library

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

# Topic Index

## 2.1 Topics

Here is a list of all topics with brief descriptions:

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# 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

# Topic Documentation

## 7.1 Implementations with a Time Tagger interface

**Classes**

- class TimeTaggerBase

    *Basis interface for all Time Tagger classes.*
- class TimeTaggerVirtual

    *virtual TimeTagger based on dump files*
- class TimeTagger

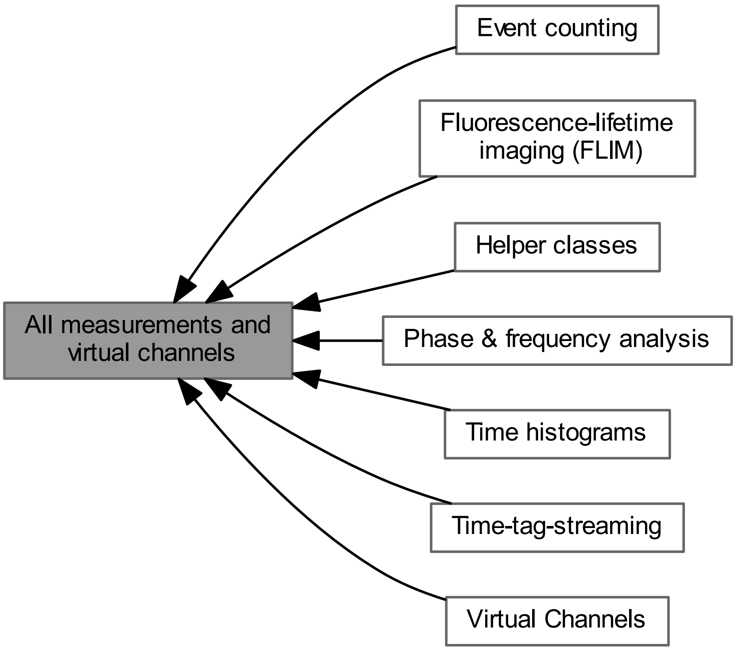
    *backend for the TimeTagger.*

### 7.1.1 Detailed Description

## 7.2 All measurements and virtual channels

Base iterators for photon counting applications.

Collaboration diagram for All measurements and virtual channels:




### Topics

- Event counting
- Time histograms

  *This section describes various measurements that calculate time differences between events and accumulate the results into a histogram.*
- Fluorescence-lifetime imaging (FLIM)

  *This section describes the Flim related measurements classes of the Time Tagger API.*
- Phase & frequency analysis

  *This section describes measurement classes that expect periodic signals.*
- Time-tag-streaming

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

### Classes

- class HistogramND

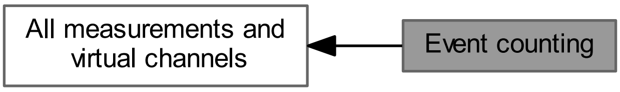
  *A N-dimensional histogram of time differences. This can be used in measurements similar to 2D NRM spectroscopy.*

## 7.2.1 Detailed Description

Base iterators for photon counting applications.

### 7.2.2 Event counting

Collaboration diagram for Event counting:




**Classes**

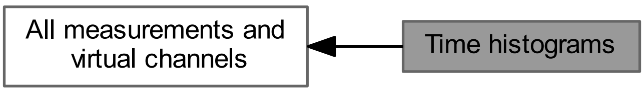
- class CountBetweenMarkers
  *a simple counter where external marker signals determine the bins*
- class Counter
  *a simple counter on one or more channels*
- class Countrate
  *count rate on one or more channels*

#### 7.2.2.1 Detailed Description

### 7.2.3 Time histograms

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

Collaboration diagram for Time histograms:



### Classes

- class StartStop

  *simple start-stop measurement*
- class TimeDifferences

  *Accumulates the time differences between clicks on two channels in one or more histograms.*
- class Histogram2D

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

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

  *Accumulate time differences into a histogram.*
- class HistogramLogBins

  *Accumulate time differences into a histogram with logarithmic increasing bin sizes.*
- class Correlation

  *Auto- and Cross-correlation measurement.*

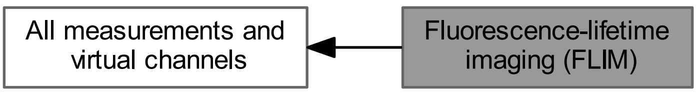
#### 7.2.3.1 Detailed Description

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

### 7.2.4 Fluorescence-lifetime imaging (FLIM)

This section describes the Flim related measurements classes of the Time Tagger API.

Collaboration diagram for Fluorescence-lifetime imaging (FLIM):




### Classes

- class FlimBase

  *basic measurement, containing a minimal set of features for efficiency purposes*
- class Flim

  *Fluorescence lifetime imaging.*

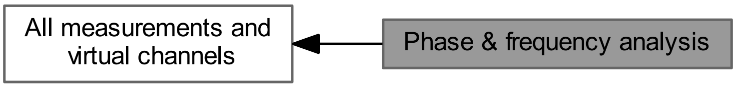
#### 7.2.4.1 Detailed Description

This section describes the Flim related measurements classes of the Time Tagger API.

### 7.2.5 Phase & frequency analysis

This section describes measurement classes that expect periodic signals.

Collaboration diagram for Phase & frequency analysis:




**Classes**

- class FrequencyCounter

  *Calculate the phase of multiple channels at equidistant sampling points.*
- class FrequencyStability

  *Allan deviation (and related metrics) calculator.*
- class Experimental::PulsePerSecondMonitor

  *Monitors the synchronicity of 1 pulse per second (PPS) signals.*

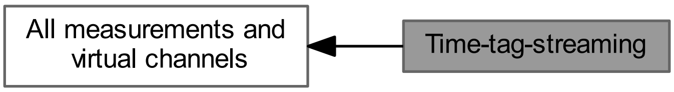
#### 7.2.5.1 Detailed Description

This section describes measurement classes that expect periodic signals.

### 7.2.6 Time-tag-streaming

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

Collaboration diagram for Time-tag-streaming:



**Classes**

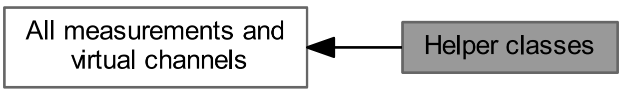
- class Iterator
  *a deprecated simple event queue*
- class TimeTagStream
  *access the time tag stream*
- class Dump
  *dump all time tags to a file*
- class Scope
  *a scope measurement*
- class FileWriter
  *compresses and stores all time tags to a file*
- class FileReader
  *Reads tags from the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

#### 7.2.6.1 Detailed Description

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

### 7.2.7 Helper classes

Collaboration diagram for Helper classes:



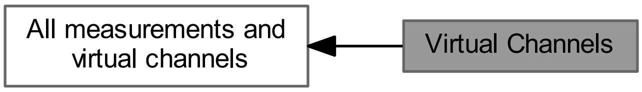

**Classes**

- class SynchronizedMeasurements
  *start, stop and clear several measurements synchronized*
- class CustomMeasurementBase
  *Helper class for custom measurements in Python and C#.*
- class SyntheticSingleTag
  *synthetic trigger timetag generator.*

#### 7.2.7.1 Detailed Description

### 7.2.8 Virtual Channels

Collaboration diagram for Virtual Channels:




**Classes**

- class Combiner

  *Combine some channels in a virtual channel which has a tick for each tick in the input channels.*
- class Coincidences

  *a coincidence monitor for many channel groups*
- class Coincidence

  *a coincidence monitor for one channel group*
- class DelayedChannel

  *a simple delayed queue*
- class TriggerOnCountrate

  *Inject trigger events when exceeding or falling below a given count rate within a rolling time window.*
- class GatedChannel

  *An input channel is gated by a gate channel.*
- class FrequencyMultiplier

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

  *a virtual CFD implementation which returns the mean time between a rising and a falling pair of edges*
- class EventGenerator

  *Generate predefined events in a virtual channel relative to a trigger event.*
- class Combinations
- class Experimental::PhotonNumber

  *Photon number resolution.*

#### 7.2.8.1 Detailed Description

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

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

# Chapter 8

# Namespace Documentation

## 8.1 Experimental Namespace Reference

Namespace for features, which are still in development and are likely to change.

### Classes

- class DlsSignalGenerator
- class ExponentialSignalGenerator
- class FcsSignalGenerator
- class GammaSignalGenerator
- class GaussianSignalGenerator
- class MarkovProcessGenerator
- class OscillatorSimulation
- class PatternSignalGenerator
- class PhotonGenerator
- class PhotonNumber

  *Photon number resolution.*
- class PulsePerSecondData

  *Helper object as return value for PulsePerSecondMonitor::getDataObject.*
- class PulsePerSecondMonitor

  *Monitors the synchronicity of 1 pulse per second (PPS) signals.*
- class SignalGeneratorBase
- class SimDetector
- class SimLifetime
- class SimSignalSplitter
- class TransformCrosstalk
- class TransformDeadtime
- class TransformEfficiency
- class TransformGaussianBroadening
- class TwoStateExponentialSignalGenerator
- class UniformSignalGenerator

### 8.1.1 Detailed Description

Namespace for features, which are still in development and are likely to change.

# Chapter 9

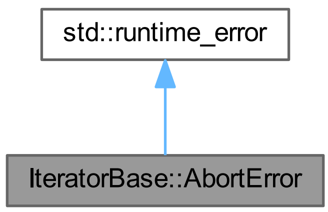
# Class Documentation

## 9.1 IteratorBase::AbortError Class Reference

A custom runtime error thrown by the `abort` call. This can be caught and handled by measurement classes, including `CustomMeasurement`, to perform actions within the abortion process.

`#include <TimeTagger.h>`

Inheritance diagram for IteratorBase::AbortError:




**Public Member Functions**

- AbortError (const std::string &what_arg)
- ∼AbortError ()

### 9.1.1 Detailed Description

A custom runtime error thrown by the `abort` call. This can be caught and handled by measurement classes, including `CustomMeasurement`, to perform actions within the abortion process.

### 9.1.2 Constructor & Destructor Documentation

#### 9.1.2.1 AbortError()

```
IteratorBase::AbortError::AbortError (
            const std::string & what_arg )  [inline]
```

#### 9.1.2.2 ∼AbortError()

```
IteratorBase::AbortError::~AbortError ( )  [inline]
```

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

- TimeTagger.h

## 9.2 ChannelGate Struct Reference

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

### Public Member Functions

- ChannelGate (channel_t gate_open_channel, channel_t gate_close_channel, GatedChannelInitial initial=GatedChannelInitial::Open)

### Public Attributes

- const channel_t gate_open_channel
- const channel_t gate_close_channel
- const GatedChannelInitial initial

### 9.2.1 Constructor & Destructor Documentation

#### 9.2.1.1 ChannelGate()

```
ChannelGate::ChannelGate (
            channel_t gate_open_channel,
            channel_t gate_close_channel,
            GatedChannelInitial initial = GatedChannelInitial::Open )  [inline]
```

### 9.2.2 Member Data Documentation

#### 9.2.2.1 gate_close_channel

```
const channel_t ChannelGate::gate_close_channel
```

#### 9.2.2.2 gate_open_channel

`const channel_t ChannelGate::gate_open_channel`

#### 9.2.2.3 initial

`const GatedChannelInitial ChannelGate::initial`

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

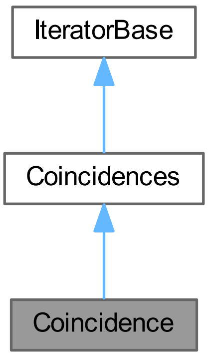
- Iterators.h

## 9.3 Coincidence Class Reference

a coincidence monitor for one channel group

`#include <Iterators.h>`

Inheritance diagram for Coincidence:




**Public Member Functions**

- Coincidence (TimeTaggerBase *tagger, std::vector< channel_t > channels, timestamp_t coincidence↩ Window=1000, CoincidenceTimestamp timestamp=CoincidenceTimestamp::Last)

  *construct a coincidence*
- channel_t getChannel ()

  *virtual channel which contains the coincidences*

## Public Member Functions inherited from Coincidences

- 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)

## Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Additional Inherited Members

## Protected Member Functions inherited from Coincidences

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

  *update iterator state*

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

    *clear Iterator state.*
- virtual void on_start ()

    *callback when the measurement class is started*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

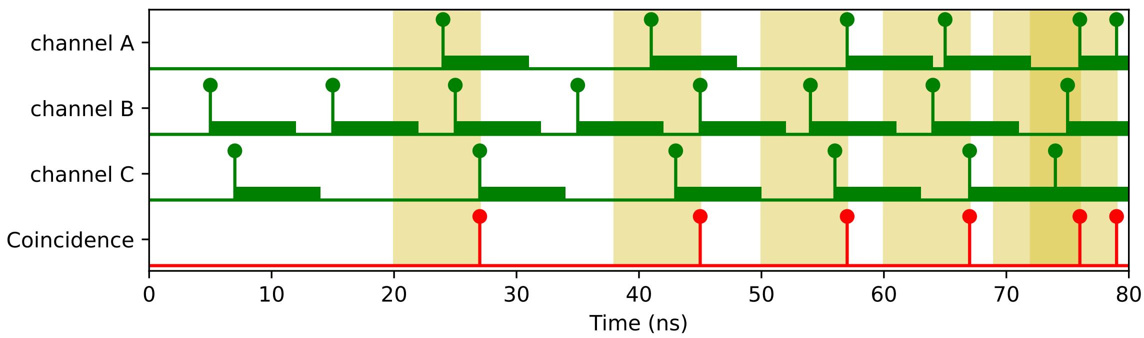
    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

### 9.3.1 Detailed Description

a coincidence monitor for one channel group




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

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 Coincidence()

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

construct a coincidence

**Parameters**

| | |
|---|---|
| *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) |

### 9.3.3 Member Function Documentation

#### 9.3.3.1 getChannel()

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

virtual channel which contains the coincidences

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

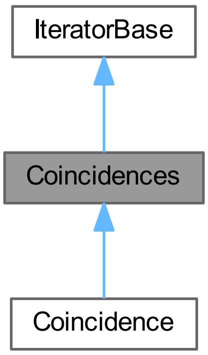
- Iterators.h

## 9.4 Coincidences Class Reference

a coincidence monitor for many channel groups

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

Inheritance diagram for Coincidences:




### Public Member Functions

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

    *construct a Coincidences*
- ∼Coincidences ()
- std::vector< channel_t > getChannels ()

    *fetches the block of virtual channels for those coincidence groups*
- void setCoincidenceWindow (timestamp_t coincidenceWindow)

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

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

- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

#### Protected Member Functions

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

  *update iterator state*

### Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.4.1 Detailed Description

a coincidence monitor for many channel groups

Monitor coincidences for given coincidence groups passed by the constructor. A coincidence is hereby defined as for a given coincidence group a) the incoming is part of this group b) at least tag arrived within the 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.

## 9.4.2 Constructor & Destructor Documentation

### 9.4.2.1 Coincidences()

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

construct a Coincidences

**Parameters**

| | |
|---|---|
| *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) |

#### 9.4.2.2 ∼Coincidences()

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

### 9.4.3 Member Function Documentation

#### 9.4.3.1 getChannels()

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

fetches the block of virtual channels for those coincidence groups

#### 9.4.3.2 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.4.3.3 setCoincidenceWindow()

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

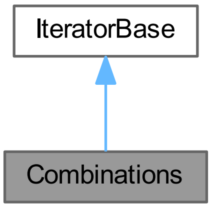
The documentation for this class was generated from the following file:

- Iterators.h

## 9.5 Combinations Class Reference

`#include <Iterators.h>`

Inheritance diagram for Combinations:




### Public Member Functions

- Combinations (TimeTaggerBase *tagger, std::vector< channel_t > const &channels, timestamp_t window_size)

  *construct a Combinations*
- ∼Combinations ()
- channel_t getChannel (std::vector< channel_t > const &input_channels) const

  *Return the virtual channel ID corresponding to an exclusive coincidence on the given input_channels. The channel gets implicitly enabled.*
- channel_t getSumChannel (int n_channels) const

  *return the ID of the virtual channel corresponding to an n_channel-fold combination of input channels*
- std::vector< channel_t > getCombination (channel_t virtual_channel) const

  *Return the set of input channels that emit a coincidence event on the given virtual channel virtual_channel.*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

*Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*

- bool isRunning ()

  *Returns True if the measurement is collecting the data.*

- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*

- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

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

- void registerChannel (channel_t channel)

  *register a channel*

- void unregisterChannel (channel_t channel)

  *unregister a channel*

- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*

- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*

- virtual void on_start ()

  *callback when the measurement class is started*

- virtual void on_stop ()

  *callback when the measurement class is stopped*

- void lock ()

  *acquire update lock*

- void unlock ()

  *release update lock*

- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*

- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*

- void finish_running ()

  *Callback for the measurement to stop itself.*

- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.5.1 Detailed Description

A combination is defined as one or more events occurring on distinct channels within a given time window, preceded and followed by two guard windows of the same duration without any events on these channels. The guard window starts at the time of the last event during the coincidence window.

This iterator emits events on virtual channels whenever such a combination is detected on the monitored channels. Given $N$ input channels $c_1, \ldots, c_N$, there are $2^N - 1$ possible combination, each having a corresponding virtual channel.

The individual virtual channels one is interested in have to be enabled by calling getChannel() before clicks on them are actually generated.

Additionally, there are N "sum channels". The n-th sum channel generates a clicks on each n-fold combination (irrespective of the specific contributing input channel).

Note that multiple events on the same channel during the coincidence window are counted as one.

### 9.5.2 Constructor & Destructor Documentation

#### 9.5.2.1 Combinations()

```
Combinations::Combinations (
            TimeTaggerBase * tagger,
            std::vector< channel_t > const & channels,
            timestamp_t window_size )
```

construct a Combinations

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *channels* | the set of channels to monitor. Elements must be distinct. |
| *window_size* | duration of the coincidence window |

#### 9.5.2.2 ∼Combinations()

```
Combinations::∼Combinations ( )
```

### 9.5.3 Member Function Documentation

#### 9.5.3.1 clear_impl()

```
void Combinations::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.

#### 9.5.3.2 getChannel()

```
channel_t Combinations::getChannel (
            std::vector< channel_t > const & input_channels ) const
```

Return the virtual channel ID corresponding to an exclusive coincidence on the given *input_channels*. The channel gets implicitly enabled.

#### 9.5.3.3 getCombination()

```
std::vector< channel_t > Combinations::getCombination (
            channel_t virtual_channel ) const
```

Return the set of input channels that emit a coincidence event on the given virtual channel *virtual_channel*.

#### 9.5.3.4 getSumChannel()

```
channel_t Combinations::getSumChannel (
            int n_channels ) const
```

return the ID of the virtual channel corresponding to an n_channel-fold combination of input channels

#### 9.5.3.5 next_impl()

```
bool Combinations::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.

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

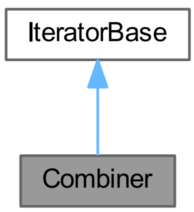
- Iterators.h

## 9.6 Combiner Class Reference

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

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

Inheritance diagram for Combiner:




### Public Member Functions

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

  *construct a combiner*
- ~Combiner ()
- void getChannelCounts (std::function< int64_t *(size_t)> array_out)

  *get sum of counts*
- void getData (std::function< int64_t *(size_t)> array_out)

  *get sum of counts*
- channel_t getChannel ()

  *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

    *callback when the measurement class is started*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

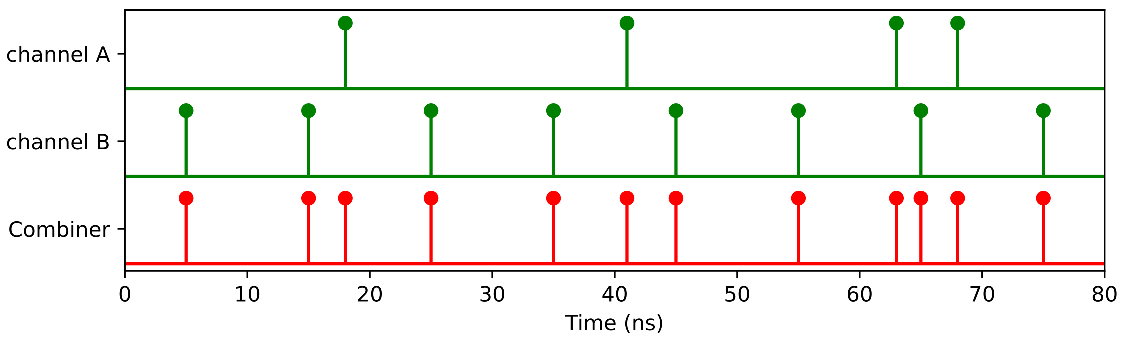
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.6.1 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.

## 9.6.2 Constructor & Destructor Documentation

### 9.6.2.1 Combiner()

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

construct a combiner

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *channels* | vector of channels to combine |

### 9.6.2.2 ∼Combiner()

```
Combiner::∼Combiner ( )
```

## 9.6.3 Member Function Documentation

### 9.6.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.6.3.2 getChannel()

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

the new virtual channel

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

### 9.6.3.3 getChannelCounts()

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

get sum of counts

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

#### 9.6.3.4 getData()

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

get sum of counts

deprecated, use getChannelCounts instead.

#### 9.6.3.5 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

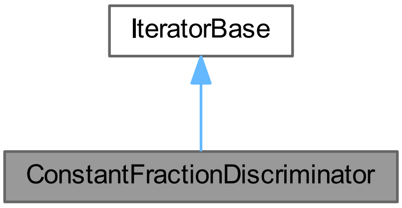
- Iterators.h

## 9.7 ConstantFractionDiscriminator Class Reference

a virtual CFD implementation which returns the mean time between a rising 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*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

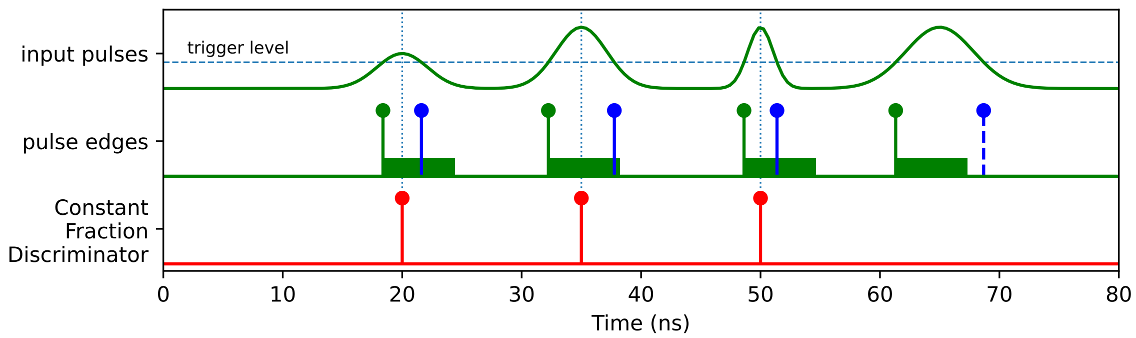
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.7.1 Detailed Description

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




## 9.7.2 Constructor & Destructor Documentation

### 9.7.2.1 ConstantFractionDiscriminator()

```
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 rising+falling pairs must be given |
| *search_window* | interval for the CFD window, must be positive |

### 9.7.2.2 ∼ConstantFractionDiscriminator()

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

## 9.7.3 Member Function Documentation

### 9.7.3.1 getChannels()

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

the list of new virtual channels

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

#### 9.7.3.2 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.7.3.3 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

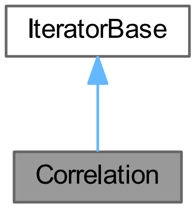
- Iterators.h

## 9.8 Correlation Class Reference

Auto- and Cross-correlation measurement.

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

Inheritance diagram for Correlation:




### Public Member Functions

- Correlation (TimeTaggerBase ∗tagger, channel_t channel_1, channel_t channel_2=CHANNEL_UNUSED, timestamp_t binwidth=1000, int n_bins=1000)

    *constructor of a correlation measurement*
- ∼Correlation ()

    *destructor of the Correlation measurement*
- void getData (std::function< int32_t ∗(size_t)> array_out)

    *returns a one-dimensional array of size n_bins containing the histogram*
- void getDataNormalized (std::function< double ∗(size_t)> array_out)

    *get the g(2) normalized histogram*
- void getIndex (std::function< timestamp_t ∗(size_t)> array_out)

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

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

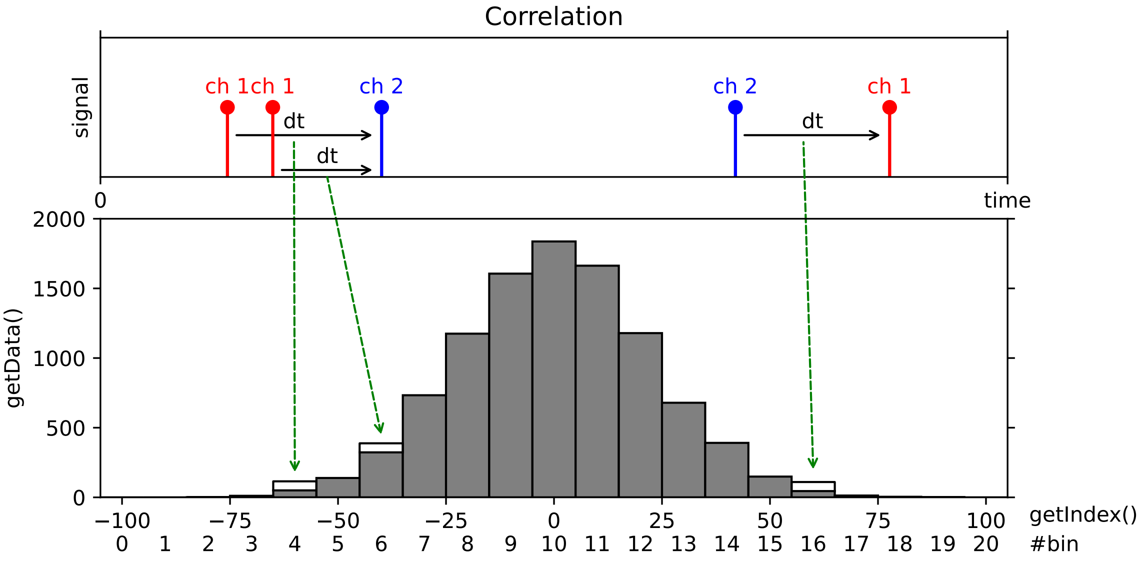
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.8.1 Detailed Description

Auto- and Cross-correlation measurement.




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

### 9.8.2 Constructor & Destructor Documentation

#### 9.8.2.1 Correlation()

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

constructor of a correlation measurement

**Note**

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

**Parameters**

| | |
|---|---|
| *tagger* | time tagger object |
| *channel←_1* | channel on which (stop) clicks are received |
| *channel←_2* | channel on which reference clicks (start) are received |
| *binwidth* | bin width in ps |
| *n_bins* | the number of bins in the resulting histogram |

#### 9.8.2.2 ∼Correlation()

```
Correlation::∼Correlation ( )
```

destructor of the Correlation measurement

### 9.8.3 Member Function Documentation

#### 9.8.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.8.3.2 getData()

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

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

**Parameters**

| | |
|---|---|
| *array_out* | allocator callback for managed return values |

#### 9.8.3.3 getDataNormalized()

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

get the g(2) normalized histogram

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

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

**Parameters**

| | |
|---|---|
| *array_out* | allocator callback for managed return values |

### 9.8.3.4 getIndex()

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

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

**Parameters**

| | |
|---|---|
| *array_out* | allocator callback for managed return values |

### 9.8.3.5 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

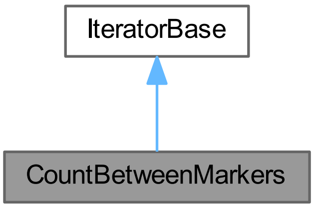
- Iterators.h

## 9.9 CountBetweenMarkers Class Reference

a simple counter where external marker signals determine the bins

`#include <Iterators.h>`

Inheritance diagram for CountBetweenMarkers:




### Public Member Functions

- CountBetweenMarkers (TimeTaggerBase ∗tagger, channel_t click_channel, channel_t begin_channel, channel_t end_channel=CHANNEL_UNUSED, int32_t n_values=1000)

    *constructor of CountBetweenMarkers*
- ∼CountBetweenMarkers ()
- bool ready ()

    *Returns true when the entire array is filled.*
- void getData (std::function< int32_t ∗(size_t)> array_out)

    *Returns array of size n_values containing the acquired counter values.*
- void getBinWidths (std::function< timestamp_t ∗(size_t)> array_out)

    *fetches the widths of each bins*
- void getIndex (std::function< timestamp_t ∗(size_t)> array_out)

    *fetches the starting time of each bin*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

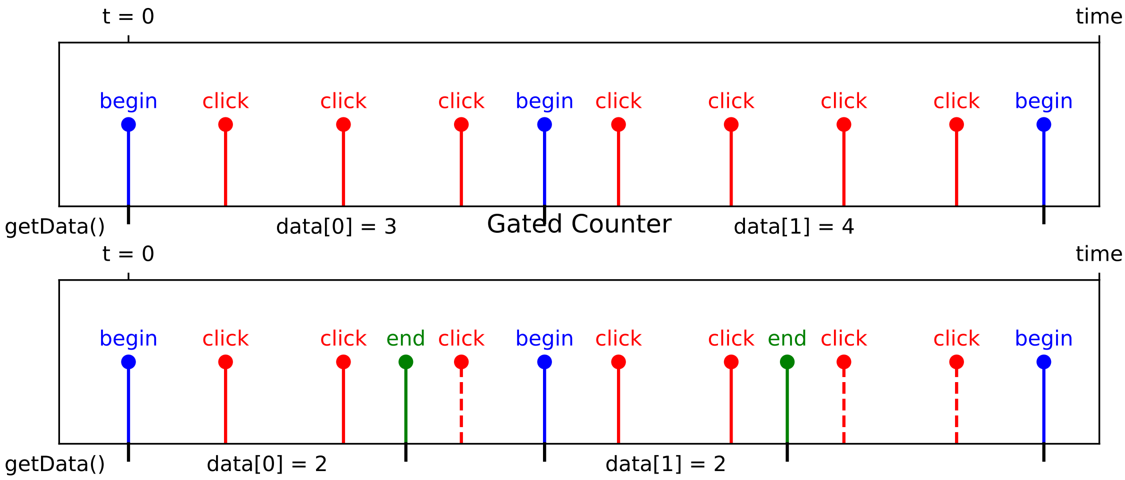
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.9.1 Detailed Description

a simple counter where external marker signals determine the bins




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

### 9.9.2 Constructor & Destructor Documentation

#### 9.9.2.1 CountBetweenMarkers()

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

constructor of CountBetweenMarkers

**Parameters**

| | |
|---|---|
| *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 |

#### 9.9.2.2 ∼CountBetweenMarkers()

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

### 9.9.3 Member Function Documentation

#### 9.9.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.9.3.2 getBinWidths()

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

fetches the widths of each bins

#### 9.9.3.3 getData()

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

Returns array of size n_values containing the acquired counter values.

#### 9.9.3.4 getIndex()

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

fetches the starting time of each bin

#### 9.9.3.5 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.9.3.6 ready()

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

Returns true when the entire array is filled.

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

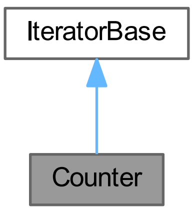
- Iterators.h

## 9.10 Counter Class Reference

a simple counter on one or more channels

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

Inheritance diagram for Counter:



### Public Member Functions

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

  *construct a counter*
- ∼Counter ()
- void getData (std::function< int32_t *(size_t, size_t)> array_out, bool rolling=true)

  *An array of size 'number of channels' by n_values containing the current values of the circular buffer (counts in each bin).*
- void getDataNormalized (std::function< double *(size_t, size_t)> array_out, bool rolling=true)

  *get countrate in Hz*
- void getDataTotalCounts (std::function< uint64_t *(size_t)> array_out)

  *get the total amount of clicks per channel since the last clear including the currently integrating bin*
- void getIndex (std::function< timestamp_t *(size_t)> array_out)

  *A vector of size n_values containing the time bins in ps.*
- CounterData getDataObject (bool remove=false)

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

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

- IteratorBase (TimeTaggerBase ∗tagger, std::string base_type_="IteratorBase", std::string extra_info_="")
  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)
  *register a channel*
- void unregisterChannel (channel_t channel)
  *unregister a channel*
- channel_t getNewVirtualChannel ()
  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()
  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()
  *callback when the measurement class is stopped*
- void lock ()
  *acquire update lock*
- void unlock ()
  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)
  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()
  *acquire update lock*
- void finish_running ()
  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

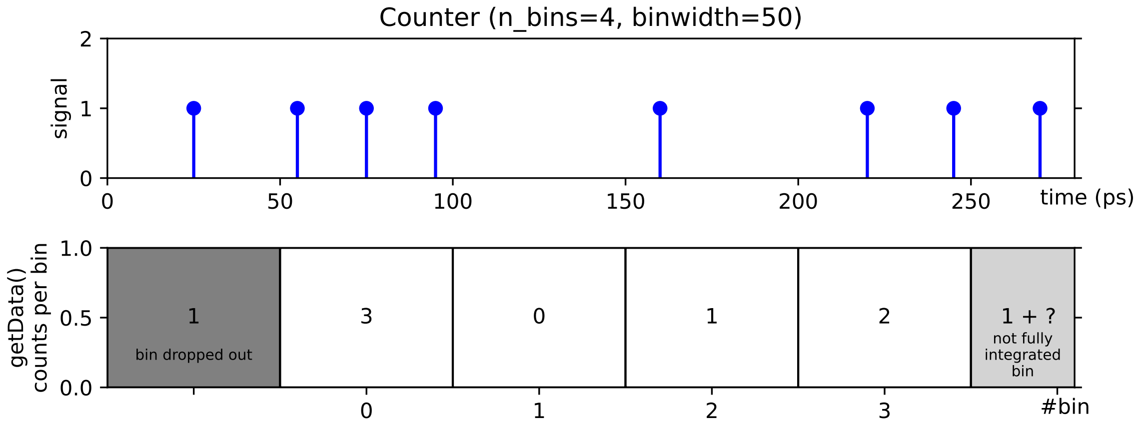
### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

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

### 9.10.1 Detailed Description

a simple counter on one or more channels




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

### 9.10.2 Constructor & Destructor Documentation

#### 9.10.2.1 Counter()

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

construct a counter

**Parameters**

| | |
|---|---|
| *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) |

#### 9.10.2.2 ∼Counter()

```
Counter::∼Counter ( )
```

### 9.10.3 Member Function Documentation

#### 9.10.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.10.3.2 getData()

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

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

**Parameters**

| | |
|---|---|
| *array_out* | allocator callback for managed return values |
| *rolling* | if true, the returning array starts with the oldest data and goes up to the newest data |

#### 9.10.3.3 getDataNormalized()

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

get countrate in Hz

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

**Parameters**

| | |
|---|---|
| *array_out* | allocator callback for managed return values |
| *rolling* | if true, the returning array starts with the oldest data and goes up to the newest data |

#### 9.10.3.4 getDataObject()

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

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

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

**Parameters**

| | |
|---|---|
| *remove* | remove all fetched bins |

**Returns**

a CounterData object, which contains all data of the fetches bins

#### 9.10.3.5 getDataTotalCounts()

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

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

#### 9.10.3.6 getIndex()

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

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

#### 9.10.3.7 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.10.3.8 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.11 CounterData Class Reference

Helper object as return value for Counter::getDataObject.

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

### Public Member Functions

- ∼CounterData ()
- void getData (std::function< int32_t ∗(size_t, size_t)> array_out)

    *get the amount of clicks (or 0 if overflow occurs) per bin and per channel*
- void getFrequency (std::function< double ∗(size_t, size_t)> array_out, timestamp_t time_scale=1000000000000)

    *get the counts normalized to the specified time scale*
- void getDataNormalized (std::function< double ∗(size_t, size_t)> array_out)

    *get the average rate of clicks per bin and per channel*
- void getDataTotalCounts (std::function< uint64_t ∗(size_t)> array_out)

    *get the total amount of clicks per channel since the last clear up to the most rececnt bin*
- void getIndex (std::function< timestamp_t ∗(size_t)> array_out)

    *get an index which corresponds to the timestamp of these bins*
- void getTime (std::function< timestamp_t ∗(size_t)> array_out)

    *get the timestamp of the bins since the last clear*
- void getOverflowMask (std::function< signed char ∗(size_t)> array_out)

    *get if the bins were in overflow*
- void getChannels (std::function< channel_t ∗(size_t)> array_out)

    *get the configured list of channels*

### Public Attributes

- const uint32_t size

    *number of returned bins*
- const uint32_t dropped_bins

    *number of bins which have been dropped because n_bins has been exceeded, usually 0*
- const bool overflow

    *has anything been in overflow mode*

## 9.11.1 Detailed Description

Helper object as return value for Counter::getDataObject.

This object stores the result of up to n_values bins.

## 9.11.2 Constructor & Destructor Documentation

### 9.11.2.1 ∼CounterData()

```
CounterData::∼CounterData ( )
```

## 9.11.3 Member Function Documentation

### 9.11.3.1 getChannels()

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

get the configured list of channels

### 9.11.3.2 getData()

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

get the amount of clicks (or 0 if overflow occurs) per bin and per channel

Consider using getFrequency for explicit overflows. Alternatively, you may check overflow field and/or call get←
OverflowMask function.

### 9.11.3.3 getDataNormalized()

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

get the average rate of clicks per bin and per channel

### 9.11.3.4 getDataTotalCounts()

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

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

#### 9.11.3.5 getFrequency()

```
void CounterData::getFrequency (
            std::function< double *(size_t, size_t)> array_out,
            timestamp_t time_scale = 1000000000000 )
```

get the counts normalized to the specified time scale

Bins in overflow mode are marked as `NaN`. The parameter time_scale scales the return value to this time interval. Default is 1 s, so the return value is in Hz. For negative values, the time scale is set to `binwidth`.

#### 9.11.3.6 getIndex()

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

get an index which corresponds to the timestamp of these bins

#### 9.11.3.7 getOverflowMask()

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

get if the bins were in overflow

#### 9.11.3.8 getTime()

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

get the timestamp of the bins since the last clear

### 9.11.4 Member Data Documentation

#### 9.11.4.1 dropped_bins

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

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

#### 9.11.4.2 overflow

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

has anything been in overflow mode

#### 9.11.4.3 size

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

number of returned bins

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

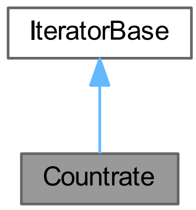
- Iterators.h

## 9.12 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*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

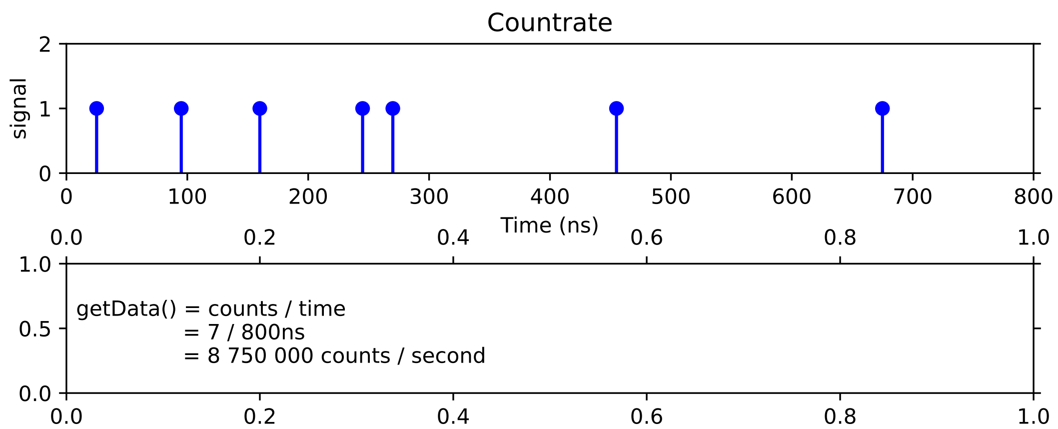
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.12.1 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.

## 9.12.2 Constructor & Destructor Documentation

### 9.12.2.1 Countrate()

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

constructor of Countrate

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *channels* | the channels to count on |

### 9.12.2.2 ∼Countrate()

```
Countrate::∼Countrate ( )
```

## 9.12.3 Member Function Documentation

### 9.12.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.12.3.2 getCountsTotal()

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

get the total amount of events

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

### 9.12.3.3 getData()

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

get the count rates

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

#### 9.12.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.12.3.5 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.13 CustomLogger Class Reference

Helper class for setLogger.

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

### Public Member Functions

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

### 9.13.1 Detailed Description

Helper class for setLogger.

### 9.13.2 Constructor & Destructor Documentation

#### 9.13.2.1 CustomLogger()

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

#### 9.13.2.2 ∼CustomLogger()

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

### 9.13.3 Member Function Documentation

#### 9.13.3.1 disable()

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

#### 9.13.3.2 enable()

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

#### 9.13.3.3 Log()

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

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

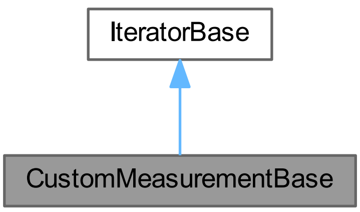
- TimeTagger.h

## 9.14 CustomMeasurementBase Class Reference

Helper class for custom measurements in Python and C#.

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

Inheritance diagram for CustomMeasurementBase:



### Public Member Functions

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

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.14.1 Detailed Description

Helper class for custom measurements in Python and C#.

## 9.14.2 Constructor & Destructor Documentation

### 9.14.2.1 CustomMeasurementBase()

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

### 9.14.2.2 ∼CustomMeasurementBase()

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

## 9.14.3 Member Function Documentation

### 9.14.3.1 _lock()

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

### 9.14.3.2 _unlock()

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

### 9.14.3.3 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.14.3.4 finalize_init()

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

### 9.14.3.5 is_running()

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

### 9.14.3.6 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.14.3.7 next_impl_cs()

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

#### 9.14.3.8 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.14.3.9 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.14.3.10 register_channel()

```
void CustomMeasurementBase::register_channel (
            channel_t channel )
```

#### 9.14.3.11 stop_all_custom_measurements()

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

#### 9.14.3.12 unregister_channel()

```
void CustomMeasurementBase::unregister_channel (
            channel_t channel )
```

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

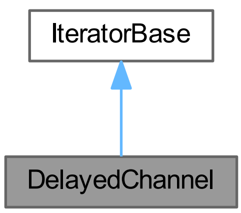
- Iterators.h

## 9.15 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.*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

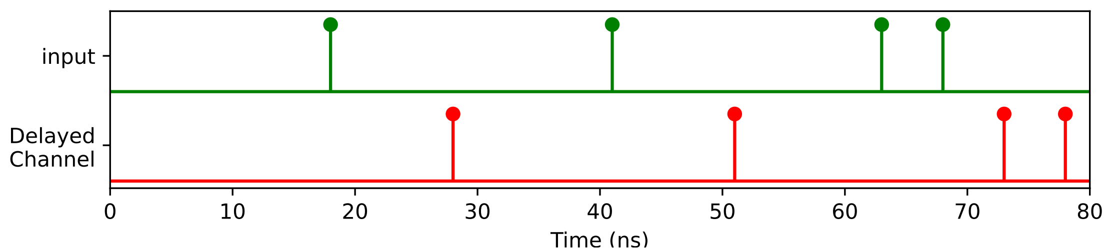
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.15.1 Detailed Description

a simple delayed queue




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

## 9.15.2 Constructor & Destructor Documentation

### 9.15.2.1 DelayedChannel() [1/2]

```
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 |

#### 9.15.2.2 DelayedChannel() [2/2]

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

constructor of a DelayedChannel for delaying many channels at once

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

**Parameters**

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

#### 9.15.2.3 ~DelayedChannel()

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

### 9.15.3 Member Function Documentation

#### 9.15.3.1 getChannel()

```
channel_t DelayedChannel::getChannel ( )
```

the first new virtual channel

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

#### 9.15.3.2 getChannels()

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

the new virtual channels

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

#### 9.15.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.15.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.15.3.5 setDelay()

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

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

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

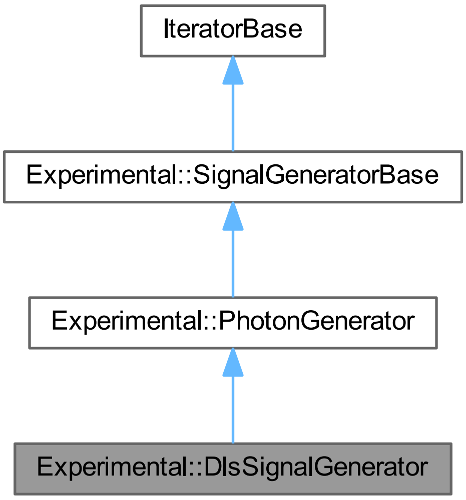
The documentation for this class was generated from the following file:

- Iterators.h

## 9.16 Experimental::DlsSignalGenerator Class Reference

`#include <Iterators.h>`

Inheritance diagram for Experimental::DlsSignalGenerator:




### Public Member Functions

- DlsSignalGenerator (TimeTaggerBase ∗tagger, double decay_time, double countrate, channel_t output_↩ channel=CHANNEL_UNUSED, int32_t seed=-1)

    *Construct a DLS event channel.*
- DlsSignalGenerator (TimeTaggerBase ∗tagger, std::vector< double > decay_times, double countrate, channel_t output_channel=CHANNEL_UNUSED, int32_t seed=-1)
- ∼DlsSignalGenerator ()
- unsigned int get_N ()

### Public Member Functions inherited from Experimental::PhotonGenerator

- PhotonGenerator (TimeTaggerBase ∗tagger, double countrate, channel_t base_channel, int32_t seed=-1)

    *A generator for TimeTags arising from a laser driven process. `PhotonGenerator` should be used as the base class of a virtual class with a dedicated `get_intensity` function which models the relevant physical processes.*
- ∼PhotonGenerator ()
- void finalize_init ()
- void set_T_PERIOD (timestamp_t new_T)
- timestamp_t get_T_PERIOD ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

    *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- double get_intensity () override

## Protected Member Functions inherited from Experimental::PhotonGenerator

- void initialize (timestamp_t initial_time) override
- void on_restart (timestamp_t restart_time) override
- timestamp_t get_next () override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

    *update iterator state*
- void on_stop () override

    *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

### Additional Inherited Members

## Protected Attributes inherited from Experimental::PhotonGenerator

- timestamp_t T_PERIOD

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

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

## 9.16.1 Constructor & Destructor Documentation

### 9.16.1.1 DlsSignalGenerator() [1/2]

```
Experimental::DlsSignalGenerator::DlsSignalGenerator (
            TimeTaggerBase * tagger,
            double decay_time,
            double countrate,
            channel_t output_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a DLS event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger. |
| *decay_time* | characteristic decay times (in seconds) for the g2 curve. |
| *countrate* | rate (in Hz) of Time Tags to be generated. |
| *output_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.16.1.2 DlsSignalGenerator() [2/2]

```
Experimental::DlsSignalGenerator::DlsSignalGenerator (
            TimeTaggerBase * tagger,
            std::vector< double > decay_times,
            double countrate,
            channel_t output_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

### 9.16.1.3 ∼DlsSignalGenerator()

```
Experimental::DlsSignalGenerator::∼DlsSignalGenerator ( )
```

## 9.16.2 Member Function Documentation

### 9.16.2.1 get_intensity()

```
double Experimental::DlsSignalGenerator::get_intensity ( )  [override], [protected], [virtual]
```

Implements Experimental::PhotonGenerator.

### 9.16.2.2 get_N()

```
unsigned int Experimental::DlsSignalGenerator::get_N ( )
```

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

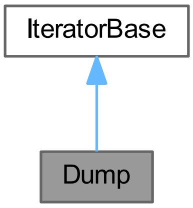
- Iterators.h

## 9.17 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 ()

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.17.1 Detailed Description

dump all time tags to a file

## 9.17.2 Constructor & Destructor Documentation

### 9.17.2.1 Dump()

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

constructor of a Dump thread

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *filename* | name of the file to dump to, must be encoded as UTF-8 |
| *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) |

### 9.17.2.2 ∼Dump()

```
Dump::∼Dump ( )
```

## 9.17.3 Member Function Documentation

### 9.17.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.17.3.2 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.17.3.3 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.17.3.4 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.18 Event Struct Reference

Object for the return value of Scope::getData.

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

### Public Attributes

- timestamp_t time
- State state

### 9.18.1 Detailed Description

Object for the return value of Scope::getData.

### 9.18.2 Member Data Documentation

#### 9.18.2.1 state

```
State Event::state
```

#### 9.18.2.2 time

```
timestamp_t Event::time
```

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

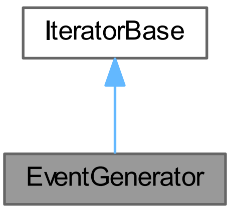
- Iterators.h

## 9.19 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*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

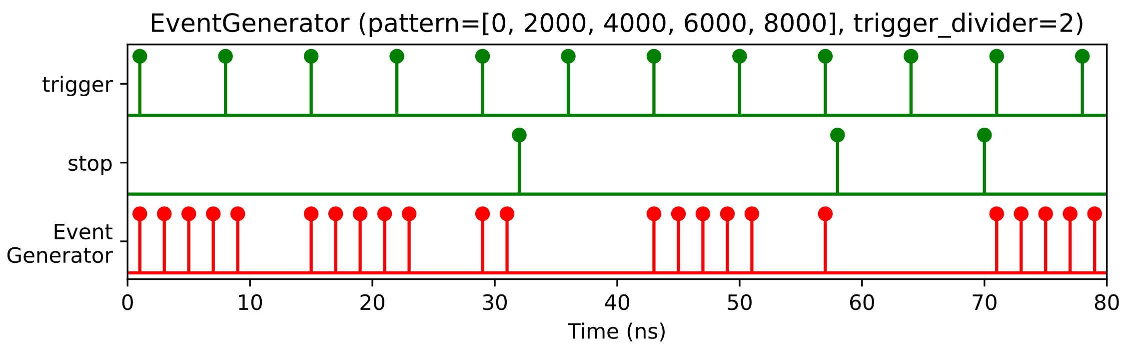
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.19.1 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.

## 9.19.2 Constructor & Destructor Documentation

### 9.19.2.1 EventGenerator()

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

construct a event generator

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *trigger_channel* | trigger for generating the pattern |
| *pattern* | vector of time stamp generated relative 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 |

### 9.19.2.2 ∼EventGenerator()

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

## 9.19.3 Member Function Documentation

### 9.19.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.19.3.2 getChannel()

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

the new virtual channel

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

#### 9.19.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.19.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

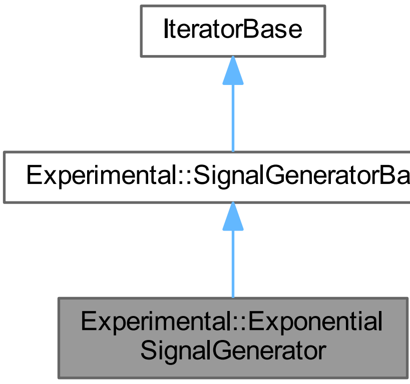
The documentation for this class was generated from the following file:

- Iterators.h

## 9.20 Experimental::ExponentialSignalGenerator Class Reference

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

Inheritance diagram for Experimental::ExponentialSignalGenerator:




**Public Member Functions**

- ExponentialSignalGenerator (TimeTaggerBase ∗tagger, double rate, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

  *Construct a exponential event channel.*
- ∼ExponentialSignalGenerator ()

## Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

*Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.20.1 Constructor & Destructor Documentation

### 9.20.1.1 ExponentialSignalGenerator()

```
Experimental::ExponentialSignalGenerator::ExponentialSignalGenerator (
            TimeTaggerBase * tagger,
            double rate,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a exponential event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *rate* | event rate in herz |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.20.1.2 ∼ExponentialSignalGenerator()

```
Experimental::ExponentialSignalGenerator::~ExponentialSignalGenerator ( )
```

## 9.20.2 Member Function Documentation

### 9.20.2.1 get_next()

```
timestamp_t Experimental::ExponentialSignalGenerator::get_next ( ) [override], [protected],
[virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.20.2.2 initialize()

```
void Experimental::ExponentialSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.20.2.3 on_restart()

```
void Experimental::ExponentialSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

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

- Iterators.h

# 9.21 FastBinning Class Reference

Helper class for fast division with a constant divisor.

`#include <Iterators.h>`

#### Public Types

- enum class Mode {
  ConstZero , Dividend , PowerOfTwo , FixedPoint_32 ,
  FixedPoint_64 , Divide_32 , 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

## 9.21.1 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.

## 9.21.2 Member Enumeration Documentation

### 9.21.2.1 Mode

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

**Enumerator**

| | |
|---|---|
| ConstZero | |
| Dividend | |
| PowerOfTwo | |
| FixedPoint_32 | |
| FixedPoint_64 | |
| Divide_32 | |
| Divide_64 | |

## 9.21.3 Constructor & Destructor Documentation

### 9.21.3.1 FastBinning() [1/2]

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

### 9.21.3.2 FastBinning() [2/2]

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

## 9.21.4 Member Function Documentation

### 9.21.4.1 divide()

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

### 9.21.4.2 getMode()

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

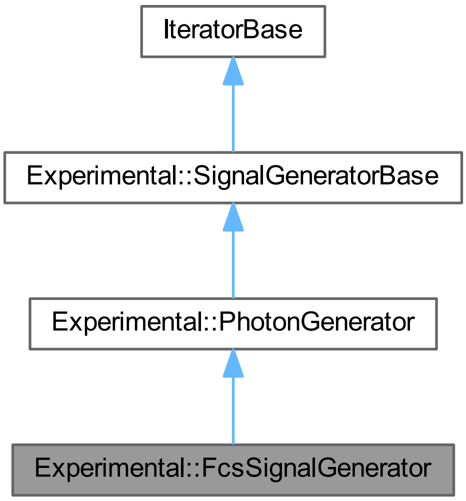
The documentation for this class was generated from the following file:

- Iterators.h

## 9.22 Experimental::FcsSignalGenerator Class Reference

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

Inheritance diagram for Experimental::FcsSignalGenerator:




#### Public Member Functions

- FcsSignalGenerator (TimeTaggerBase *tagger, double correlation_time, double N_focus, double countrate, channel_t output_channel=CHANNEL_UNUSED, int32_t seed=-1)

    *Construct an FCS event channel.*
- ~FcsSignalGenerator ()
- unsigned int get_N ()
- void set_boundary_limit (double new_boundary)

### Public Member Functions inherited from Experimental::PhotonGenerator

- PhotonGenerator (TimeTaggerBase *tagger, double countrate, channel_t base_channel, int32_t seed=-1)

    *A generator for TimeTags arising from a laser driven process.* `PhotonGenerator` *should be used as the base class of a virtual class with a dedicated* `get_intensity` *function which models the relevant physical processes.*
- ~PhotonGenerator ()
- void finalize_init ()
- void set_T_PERIOD (timestamp_t new_T)
- timestamp_t get_T_PERIOD ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase *tagger, channel_t base_channel=CHANNEL_UNUSED)
- ~SignalGeneratorBase ()
- channel_t getChannel ()

    *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- double get_intensity () override

## Protected Member Functions inherited from Experimental::PhotonGenerator

- void initialize (timestamp_t initial_time) override
- void on_restart (timestamp_t restart_time) override
- timestamp_t get_next () override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

    *update iterator state*
- void on_stop () override

    *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

    *clear Iterator state.*
- virtual void on_start ()

    *callback when the measurement class is started*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from Experimental::PhotonGenerator

- timestamp_t T_PERIOD

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.22.1 Constructor & Destructor Documentation

### 9.22.1.1 FcsSignalGenerator()

```
Experimental::FcsSignalGenerator::FcsSignalGenerator (
            TimeTaggerBase * tagger,
            double correlation_time,
            double N_focus,
            double countrate,
            channel_t output_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct an FCS event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger. |
| *correlation_time* | characteristic correlation time in the exponential g2 curve. |
| *countrate* | rate (in Hz) of Time Tags to be generated. |
| *N_focus* | the average number of particles in the laser focus. |
| *output_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.22.1.2 ∼FcsSignalGenerator()

```
Experimental::FcsSignalGenerator::~FcsSignalGenerator ( )
```

## 9.22.2 Member Function Documentation

### 9.22.2.1 get_intensity()

```
double Experimental::FcsSignalGenerator::get_intensity ( )  [override], [protected], [virtual]
```

Implements Experimental::PhotonGenerator.

### 9.22.2.2 get_N()

```
unsigned int Experimental::FcsSignalGenerator::get_N ( )
```

### 9.22.2.3 set_boundary_limit()

```
void Experimental::FcsSignalGenerator::set_boundary_limit (
            double new_boundary )
```

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

- Iterators.h

## 9.23 FileReader Class Reference

Reads tags from the disk files, which has been created by FileWriter.

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

### Public Member Functions

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

  *Creates a file reader with the given filename.*
- FileReader (const std::string &filename)

  *Creates a file reader with the given filename.*
- ∼FileReader ()
- bool hasData ()

  *Checks if there are still events in the FileReader.*
- TimeTagStreamBuffer getData (uint64_t n_events)

  *Fetches and delete the next tags from the internal buffer.*
- bool getDataRaw (std::vector< Tag > &tag_buffer)

  *Low level file reading.*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the Time Tagger object, which was serialized in the current file.*
- std::vector< channel_t > getChannelList ()

  *Fetches channels from the input file.*
- std::string getLastMarker ()

  *return the last processed marker from the file.*

### 9.23.1 Detailed Description

Reads tags from the disk files, which has been created by FileWriter.

Its usage is compatible with the TimeTagStream.

### 9.23.2 Constructor & Destructor Documentation

#### 9.23.2.1 FileReader() [1/2]

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

Creates a file reader with the given filename.

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

**Parameters**

| | |
|---|---|
| *filenames* | list of files to read, must be encoded as UTF-8 |

#### 9.23.2.2 FileReader() [2/2]

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

Creates a file reader with the given filename.

The file reader automatically continues to read split FileWriter Streams

**Parameters**

| | |
|---|---|
| *filename* | file to read, must be encoded as UTF-8 |

#### 9.23.2.3 ~FileReader()

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

### 9.23.3 Member Function Documentation

#### 9.23.3.1 getChannelList()

```
std::vector< channel_t > FileReader::getChannelList ( )
```

Fetches channels from the input file.

**Returns**

a vector of channels from the input file.

#### 9.23.3.2 getConfiguration()

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

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

**Returns**

a JSON serialized string with all configuration and status flags.

#### 9.23.3.3 getData()

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

Fetches and delete the next tags from the internal buffer.

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

**Parameters**

| | |
|---|---|
| *n_events* | maximum amount of elements to fetch |

**Returns**

a TimeTagStreamBuffer with up to n_events events

#### 9.23.3.4 getDataRaw()

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

Low level file reading.

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

**Parameters**

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

**Returns**

true if fetching the data was successfully

#### 9.23.3.5 getLastMarker()

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

return the last processed marker from the file.

**Returns**

the last marker from the file

#### 9.23.3.6 hasData()

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

Checks if there are still events in the FileReader.

**Returns**

false if no more events can be read from this FileReader

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

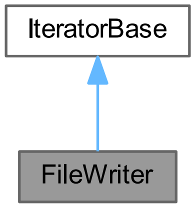
- Iterators.h

## 9.24 FileWriter Class Reference

compresses and stores all time tags to a file

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

Inheritance diagram for FileWriter:




### Public Member Functions

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

    *constructor of a FileWriter*
- ~FileWriter ()
- void split (const std::string &new_filename="")

    *Close the current file and create a new one.*
- void setMaxFileSize (uint64_t max_file_size)

    *Set the maximum file size on disk when the automatic split shall happen.*
- uint64_t getMaxFileSize ()

    *fetches the maximum file size. Please see setMaxFileSize for more details.*
- uint64_t getTotalEvents ()

    *queries the total amount of events stored in all files*
- uint64_t getTotalSize ()

    *queries the total amount of bytes stored in all files*
- void setMarker (const std::string &marker)

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

### Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

*After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.24.1 Detailed Description

compresses and stores all time tags to a file

## 9.24.2 Constructor & Destructor Documentation

### 9.24.2.1 FileWriter()

```
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, must be encoded as UTF-8 |
| *channels* | channels which are stored to the file |

### 9.24.2.2 ∼FileWriter()

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

## 9.24.3 Member Function Documentation

### 9.24.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.24.3.2 getMaxFileSize()

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

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

**Returns**

the maximum file size in bytes

#### 9.24.3.3 getTotalEvents()

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

queries the total amount of events stored in all files

**Returns**

the total amount of events stored

#### 9.24.3.4 getTotalSize()

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

queries the total amount of bytes stored in all files

**Returns**

the total amount of bytes stored

#### 9.24.3.5 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.24.3.6 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.24.3.7 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.24.3.8 setMarker()

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

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

**Parameters**

| | |
|---|---|
| *marker* | the marker to write into the file |

#### 9.24.3.9 setMaxFileSize()

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

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

**Note**

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

**Parameters**

| | |
|---|---|
| *max_file_size* | new maximum file size in bytes |

#### 9.24.3.10 split()

```
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. |

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

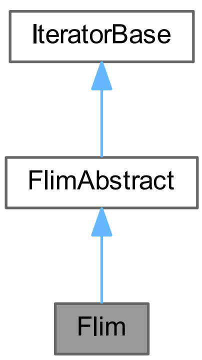
- Iterators.h

## 9.25 Flim Class Reference

Fluorescence lifetime imaging.

`#include <Iterators.h>`

Inheritance diagram for Flim:



### Public Member Functions

- Flim (TimeTaggerBase *tagger, channel_t start_channel, channel_t click_channel, channel_t pixel_begin_channel, uint32_t n_pixels, uint32_t n_bins, timestamp_t binwidth, channel_t pixel_end_channel=CHANNEL_UNUSED, channel_t frame_begin_channel=CHANNEL_UNUSED, uint32_t finish_after_outputframe=0, uint32_↩ t n_frame_average=1, bool pre_initialize=true)

  *construct a Flim measurement with a variety of high-level functionality*
- ∼Flim ()
- void initialize ()

  *initializes and starts measuring this Flim measurement*
- void getReadyFrame (std::function< uint32_t *(size_t, size_t)> array_out, int32_t index=-1)

  *obtain for each pixel the histogram for the given frame index*
- void getReadyFrameIntensity (std::function< float *(size_t)> array_out, int32_t index=-1)

  *obtain an array of the pixel intensity of the given frame index*
- void getCurrentFrame (std::function< uint32_t *(size_t, size_t)> array_out)

  *obtain for each pixel the histogram for the frame currently active*
- void getCurrentFrameIntensity (std::function< float *(size_t)> array_out)

  *obtain the array of the pixel intensities of the frame currently active*
- void getSummedFrames (std::function< uint32_t *(size_t, size_t)> array_out, bool only_ready_frames=true, bool clear_summed=false)

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

  *obtain the array of the pixel intensities from all frames acquired so far*
- FlimFrameInfo getReadyFrameEx (int32_t index=-1)

  *obtain a frame information object, for the given frame index*
- FlimFrameInfo getCurrentFrameEx ()

  *obtain a frame information object, for the currently active frame*
- FlimFrameInfo getSummedFramesEx (bool only_ready_frames=true, bool clear_summed=false)

  *obtain a frame information object, that represents the sum of all frames acquired so for.*
- uint32_t getFramesAcquired () const

  *total number of frames completed so far*
- void getIndex (std::function< timestamp_t *(size_t)> array_out)

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

## Public Member Functions inherited from 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_UNUS channel_t frame_begin_channel=CHANNEL_UNUSED, uint32_t finish_after_outputframe=0, uint32_↩ t n_frame_average=1, bool pre_initialize=true)

  *construct a FlimAbstract object, Flim and FlimBase classes inherit from it*
- ∼FlimAbstract ()
- bool isAcquiring () const

  *tells if the data acquisition has finished reaching finish_after_outputframe*

## Public Member Functions inherited from IteratorBase

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

### Protected Member Functions

- void on_frame_end () override final
- 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 Member Functions inherited from FlimAbstract

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### 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

## Protected Attributes inherited from FlimAbstract

- 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

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

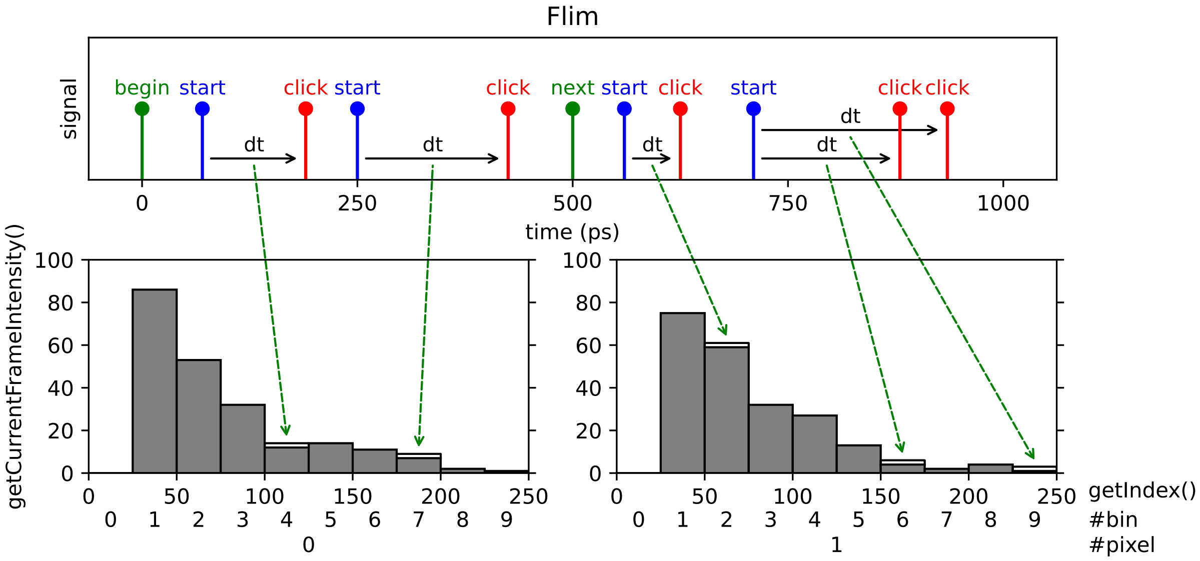
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.25.1 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.

## 9.25.2 Constructor & Destructor Documentation

### 9.25.2.1 Flim()

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

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

**Parameters**

| | |
|---|---|
| *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) |

**Parameters**

| | |
|---|---|
| *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. |

#### 9.25.2.2 ∼Flim()

```
Flim::∼Flim ( )
```

### 9.25.3 Member Function Documentation

#### 9.25.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.25.3.2 frameReady()

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

#### 9.25.3.3 get_ready_index()

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

#### 9.25.3.4 getCurrentFrame()

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

obtain for each pixel the histogram for the frame currently active

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

#### 9.25.3.5 getCurrentFrameEx()

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

obtain a frame information object, for the currently active frame

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

#### 9.25.3.6 getCurrentFrameIntensity()

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

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

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

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

#### 9.25.3.7 getFramesAcquired()

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

total number of frames completed so far

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

#### 9.25.3.8 getIndex()

```
void Flim::getIndex (
            std::function< timestamp_t *(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.

#### 9.25.3.9 getReadyFrame()

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

obtain for each pixel the histogram for the given frame index

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

#### 9.25.3.10 getReadyFrameEx()

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

obtain a frame information object, for the given frame index

This function returns a frame information object according to the index given. If the index is -1, it will return the last completed frame. When finish_after_outputframe is 0, index must be -1. If index >= 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 |

#### 9.25.3.11 getReadyFrameIntensity()

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

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

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

#### 9.25.3.12 getSummedFrames()

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

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

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

**Parameters**

| | |
|---|---|
| *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. |

#### 9.25.3.13 getSummedFramesEx()

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

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

#### 9.25.3.14 getSummedFramesIntensity()

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

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

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

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

**Parameters**

| | |
|---|---|
| *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. |

#### 9.25.3.15 initialize()

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

initializes and starts measuring this Flim measurement

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

#### 9.25.3.16 on_frame_end()

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

Implements FlimAbstract.

## 9.25.4 Member Data Documentation

#### 9.25.4.1 accum_diffs

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

#### 9.25.4.2 back_frames

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

#### 9.25.4.3 captured_frames

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

#### 9.25.4.4 frame_begins

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

#### 9.25.4.5 frame_ends

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

#### 9.25.4.6 last_frame

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

#### 9.25.4.7 pixels_completed

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

#### 9.25.4.8 summed_frames

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

#### 9.25.4.9 swap_chain_lock

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

#### 9.25.4.10 total_frames

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

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

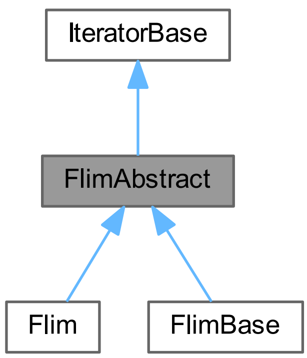
- Iterators.h

## 9.26 FlimAbstract Class Reference

Interface for FLIM measurements, Flim and FlimBase classes inherit from it.

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

Inheritance diagram for FlimAbstract:




### Public Member Functions

- FlimAbstract (TimeTaggerBase *tagger, channel_t start_channel, channel_t click_channel, channel_t pixel_begin_channel, uint32_t n_pixels, uint32_t n_bins, timestamp_t binwidth, channel_t pixel_end_channel=CHANNEL_UNUS channel_t frame_begin_channel=CHANNEL_UNUSED, uint32_t finish_after_outputframe=0, uint32_↩ t n_frame_average=1, bool pre_initialize=true)

    *construct a FlimAbstract object, Flim and FlimBase classes inherit from it*
- ∼FlimAbstract ()
- bool isAcquiring () const

    *tells if the data acquisition has finished reaching finish_after_outputframe*

## Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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 Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*

- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### 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

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.26.1 Detailed Description

Interface for FLIM measurements, Flim and FlimBase classes inherit from it.

## 9.26.2 Constructor & Destructor Documentation

### 9.26.2.1 FlimAbstract()

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

construct a FlimAbstract object, Flim and FlimBase classes inherit from it

**Parameters**

| | |
|---|---|
| *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 |

**Parameters**

| | |
|---|---|
| *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. |

#### 9.26.2.2 ∼FlimAbstract()

```
FlimAbstract::∼FlimAbstract ( )
```

### 9.26.3 Member Function Documentation

#### 9.26.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.26.3.2 isAcquiring()

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

tells if the data acquisition has finished reaching finish_after_outputframe

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

**Note**

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

#### 9.26.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.26.3.4 on_frame_end()

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

Implemented in FlimBase, and Flim.

#### 9.26.3.5 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.26.3.6 process_tags()

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

### 9.26.4 Member Data Documentation

#### 9.26.4.1 acquiring

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

#### 9.26.4.2 acquisition_lock

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

#### 9.26.4.3 binner

`FastBinning FlimAbstract::binner [protected]`

#### 9.26.4.4 binwidth

`const timestamp_t FlimAbstract::binwidth [protected]`

#### 9.26.4.5 click_channel

`const channel_t FlimAbstract::click_channel [protected]`

#### 9.26.4.6 current_frame_begin

`timestamp_t FlimAbstract::current_frame_begin [protected]`

#### 9.26.4.7 current_frame_end

`timestamp_t FlimAbstract::current_frame_end [protected]`

#### 9.26.4.8 data_base

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

#### 9.26.4.9 finish_after_outputframe

`const uint32_t FlimAbstract::finish_after_outputframe [protected]`

#### 9.26.4.10 frame

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

#### 9.26.4.11 frame_acquisition

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

#### 9.26.4.12 frame_begin_channel

`const channel_t FlimAbstract::frame_begin_channel [protected]`

#### 9.26.4.13 frames_completed

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

#### 9.26.4.14 initialized

`bool FlimAbstract::initialized [protected]`

#### 9.26.4.15 n_bins

`const uint32_t FlimAbstract::n_bins [protected]`

#### 9.26.4.16 n_frame_average

`const uint32_t FlimAbstract::n_frame_average [protected]`

#### 9.26.4.17 n_pixels

`const uint32_t FlimAbstract::n_pixels [protected]`

#### 9.26.4.18 pixel_acquisition

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

#### 9.26.4.19 pixel_begin_channel

`const channel_t FlimAbstract::pixel_begin_channel [protected]`

#### 9.26.4.20 pixel_begins

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

#### 9.26.4.21 pixel_end_channel

`const channel_t FlimAbstract::pixel_end_channel [protected]`

#### 9.26.4.22 pixel_ends

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

#### 9.26.4.23 pixels_processed

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

#### 9.26.4.24 previous_starts

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

#### 9.26.4.25 start_channel

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

#### 9.26.4.26 ticks

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

#### 9.26.4.27 time_window

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

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

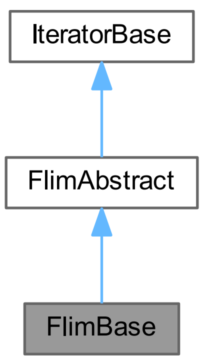
- Iterators.h

## 9.27 FlimBase Class Reference

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

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

Inheritance diagram for FlimBase:



### Public Member Functions

- FlimBase (TimeTaggerBase *tagger, channel_t start_channel, channel_t click_channel, channel_t pixel_begin_channel, uint32_t n_pixels, uint32_t n_bins, timestamp_t binwidth, channel_t pixel_end_channel=CHANNEL_UNUSED, channel_t frame_begin_channel=CHANNEL_UNUSED, uint32_t finish_after_outputframe=0, uint32_↩ t n_frame_average=1, bool pre_initialize=true)

  *construct a basic Flim measurement, containing a minimum featureset for efficiency purposes*
- ∼FlimBase ()
- void initialize ()

  *initializes and starts measuring this Flim measurement*

## Public Member Functions inherited from 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*
- ∼FlimAbstract ()
- bool isAcquiring () const

  *tells if the data acquisition has finished reaching finish_after_outputframe*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void on_frame_end () override final
- 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 Member Functions inherited from FlimAbstract

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Protected Attributes

- uint32_t total_frames

## Protected Attributes inherited from FlimAbstract

- 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

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.27.1 Detailed Description

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

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

## 9.27.2 Constructor & Destructor Documentation

### 9.27.2.1 FlimBase()

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

construct a basic Flim measurement, containing a minimum featureset for efficiency purposes

**Parameters**

| | |
|---|---|
| *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. |

### 9.27.2.2 ∼FlimBase()

```
FlimBase::∼FlimBase ( )
```

## 9.27.3 Member Function Documentation

### 9.27.3.1 frameReady()

```
virtual void FlimBase::frameReady (
            uint32_t frame_number,
            std::vector< uint32_t > & data,
```

```
std::vector< timestamp_t > & pixel_begin_times,
std::vector< timestamp_t > & pixel_end_times,
timestamp_t frame_begin_time,
timestamp_t frame_end_time )  [protected], [virtual]
```

#### 9.27.3.2 initialize()

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

initializes and starts measuring this Flim measurement

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

#### 9.27.3.3 on_frame_end()

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

Implements FlimAbstract.

### 9.27.4 Member Data Documentation

#### 9.27.4.1 total_frames

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

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

- Iterators.h

## 9.28 FlimFrameInfo Class Reference

object for storing the state of Flim::getCurrentFrameEx

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

**Public Member Functions**

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

**Public Attributes**

- uint32_t pixels
- uint32_t bins
- int32_t frame_number
- uint32_t pixel_position
- bool valid

## 9.28.1 Detailed Description

object for storing the state of Flim::getCurrentFrameEx

## 9.28.2 Constructor & Destructor Documentation

### 9.28.2.1 ∼FlimFrameInfo()

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

## 9.28.3 Member Function Documentation

### 9.28.3.1 getFrameNumber()

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

index of this frame

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

deprecated, use frame_number instead..

### 9.28.3.2 getHistograms()

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

### 9.28.3.3 getIntensities()

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

### 9.28.3.4 getPixelBegins()

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

#### 9.28.3.5 getPixelEnds()

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

#### 9.28.3.6 getPixelPosition()

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

number of pixels acquired on this frame

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

#### 9.28.3.7 getSummedCounts()

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

#### 9.28.3.8 isValid()

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

tells if this frame is valid

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

deprecated, use isValid instead.

### 9.28.4 Member Data Documentation

#### 9.28.4.1 bins

```
uint32_t FlimFrameInfo::bins
```

#### 9.28.4.2 frame_number

```
int32_t FlimFrameInfo::frame_number
```

#### 9.28.4.3 pixel_position

```
uint32_t FlimFrameInfo::pixel_position
```

#### 9.28.4.4 pixels

```
uint32_t FlimFrameInfo::pixels
```

#### 9.28.4.5 valid

```
bool FlimFrameInfo::valid
```

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

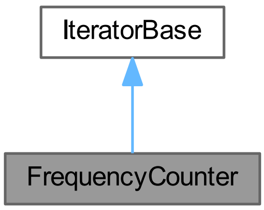
- Iterators.h

## 9.29 FrequencyCounter Class Reference

Calculate the phase of multiple channels at equidistant sampling points.

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

Inheritance diagram for FrequencyCounter:




**Public Member Functions**

- FrequencyCounter (TimeTaggerBase *tagger, std::vector< channel_t > channels, timestamp_t sampling_interval, timestamp_t fitting_window, int32_t n_values=0)
- ~FrequencyCounter ()
- FrequencyCounterData getDataObject (uint16_t event_divider=1, bool remove=false, bool channels_last_dim=false)

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.29.1 Detailed Description

Calculate the phase of multiple channels at equidistant sampling points.

This measurement calculates the phase of a periodic signal at evenly spaced sampling times. Multiple channels can be analyzed in parallel to compare the phase evolution in time. Around every sampling time, the time tags within an adjustable `fitting_window` are used to fit the phase.

## 9.29.2 Constructor & Destructor Documentation

### 9.29.2.1 FrequencyCounter()

```
FrequencyCounter::FrequencyCounter (
            TimeTaggerBase * tagger,
            std::vector< channel_t > channels,
            timestamp_t sampling_interval,
            timestamp_t fitting_window,
            int32_t n_values = 0 )
```

#### 9.29.2.2 ∼FrequencyCounter()

```
FrequencyCounter::~FrequencyCounter ( )
```

### 9.29.3 Member Function Documentation

#### 9.29.3.1 clear_impl()

```
void FrequencyCounter::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.

#### 9.29.3.2 getDataObject()

```
FrequencyCounterData FrequencyCounter::getDataObject (
            uint16_t event_divider = 1,
            bool remove = false,
            bool channels_last_dim = false )
```

#### 9.29.3.3 next_impl()

```
bool FrequencyCounter::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.

#### 9.29.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.30 FrequencyCounterData Class Reference

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

### Public Member Functions

- ∼FrequencyCounterData ()
- void getIndex (std::function< timestamp_t ∗(size_t)> array_out)

  *Returns the index of each sampling point.*
- void getTime (std::function< timestamp_t ∗(size_t)> array_out)

  *get the timestamp of the bins since the last clear*
- void getOverflowMask (std::function< signed char ∗(size_t, size_t)> array_out)

  *Returns an overflow mask with 1 = "has overflow" and 0 = "is valid".*
- void getPeriodsCount (std::function< timestamp_t ∗(size_t, size_t)> array_out)

  *Integer part of the absolute phase.*
- void getPeriodsFraction (std::function< double ∗(size_t, size_t)> array_out)

  *Fraction of the current cycle as a value from [0, 1).*
- void getFrequency (std::function< double ∗(size_t, size_t)> array_out, timestamp_t time_scale=1000000000000)

  *Frequency of the previous sampling period calculated from the phase difference between the current and the previous sample.*
- void getFrequencyInstantaneous (std::function< double ∗(size_t, size_t)> array_out)

  *Instantaneous frequency within the fitting window obtained from the fit slope.*
- void getPhase (std::function< double ∗(size_t, size_t)> array_out, double reference_frequency=0)

  *Phase with respect to an expected reference frequency.*

### Public Attributes

- const timestamp_t overflow_samples

  *Number of sampling points affected by an overflow range since the start of the measurement.*
- const unsigned int size

  *Number of sampling points represented by the object.*
- const bool align_to_reference

  *Indicates if the sampling grid has been aligned to the SoftwareClock.*
- const timestamp_t sampling_interval

  *The sampling interval in picoseconds.*
- const timestamp_t sample_offset

  *Index offset of the first index.*
- const bool channels_last_dim

  *Channels as last dimension.*

## 9.30.1 Constructor & Destructor Documentation

### 9.30.1.1 ∼FrequencyCounterData()

```
FrequencyCounterData::~FrequencyCounterData ( )
```

## 9.30.2 Member Function Documentation

### 9.30.2.1 getFrequency()

```
void FrequencyCounterData::getFrequency (
            std::function< double *(size_t, size_t)> array_out,
            timestamp_t time_scale = 1000000000000 )
```

Frequency of the previous sampling period calculated from the phase difference between the current and the previous sample.

### 9.30.2.2 getFrequencyInstantaneous()

```
void FrequencyCounterData::getFrequencyInstantaneous (
            std::function< double *(size_t, size_t)> array_out )
```

Instantaneous frequency within the fitting window obtained from the fit slope.

### 9.30.2.3 getIndex()

```
void FrequencyCounterData::getIndex (
            std::function< timestamp_t *(size_t)> array_out )
```

Returns the index of each sampling point.

### 9.30.2.4 getOverflowMask()

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

Returns an overflow mask with 1 = "has overflow" and 0 = "is valid".

### 9.30.2.5 getPeriodsCount()

```
void FrequencyCounterData::getPeriodsCount (
            std::function< timestamp_t *(size_t, size_t)> array_out )
```

Integer part of the absolute phase.

#### 9.30.2.6 getPeriodsFraction()

```
void FrequencyCounterData::getPeriodsFraction (
            std::function< double *(size_t, size_t)> array_out )
```

Fraction of the current cycle as a value from [0, 1).

#### 9.30.2.7 getPhase()

```
void FrequencyCounterData::getPhase (
            std::function< double *(size_t, size_t)> array_out,
            double reference_frequency = 0 )
```

Phase with respect to an expected reference frequency.

#### 9.30.2.8 getTime()

```
void FrequencyCounterData::getTime (
            std::function< timestamp_t *(size_t)> array_out )
```

get the timestamp of the bins since the last clear

### 9.30.3 Member Data Documentation

#### 9.30.3.1 align_to_reference

```
const bool FrequencyCounterData::align_to_reference
```

Indicates if the sampling grid has been aligned to the SoftwareClock.

#### 9.30.3.2 channels_last_dim

```
const bool FrequencyCounterData::channels_last_dim
```

Channels as last dimension.

#### 9.30.3.3 overflow_samples

```
const timestamp_t FrequencyCounterData::overflow_samples
```

Number of sampling points affected by an overflow range since the start of the measurement.

#### 9.30.3.4 sample_offset

```
const timestamp_t FrequencyCounterData::sample_offset
```

Index offset of the first index.

#### 9.30.3.5 sampling_interval

```
const timestamp_t FrequencyCounterData::sampling_interval
```

The sampling interval in picoseconds.

#### 9.30.3.6 size

```
const unsigned int FrequencyCounterData::size
```

Number of sampling points represented by the object.

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

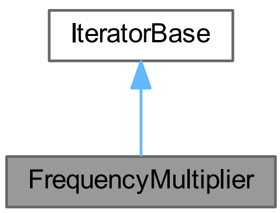
- Iterators.h

## 9.31 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 ()

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

    *callback when the measurement class is started*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

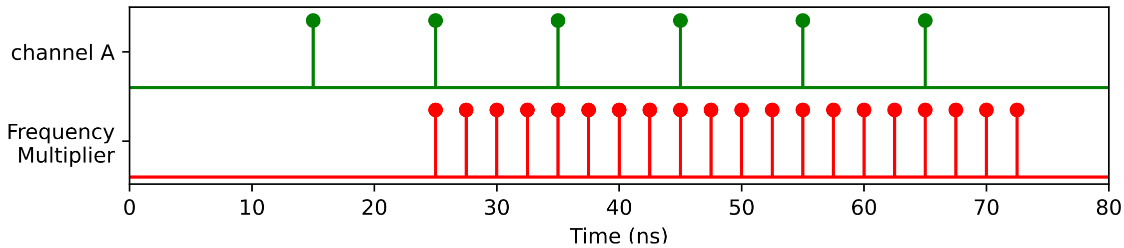
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.31.1 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.

## 9.31.2 Constructor & Destructor Documentation

### 9.31.2.1 FrequencyMultiplier()

```
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 |

### 9.31.2.2 ∼FrequencyMultiplier()

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

## 9.31.3 Member Function Documentation

### 9.31.3.1 clear_impl()

```
void FrequencyMultiplier::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.

### 9.31.3.2 getChannel()

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

### 9.31.3.3 getMultiplier()

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

### 9.31.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

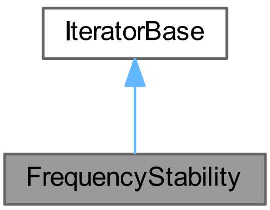
- Iterators.h

## 9.32 FrequencyStability Class Reference

Allan deviation (and related metrics) calculator.

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

Inheritance diagram for FrequencyStability:




### Public Member Functions

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

  *constructor of a FrequencyStability measurement*
- ∼FrequencyStability ()
- FrequencyStabilityData getDataObject ()

  *get a return object with all data in a synchronized way*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.32.1 Detailed Description

Allan deviation (and related metrics) calculator.

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

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

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

- STDD: Standard derivation of each period pair. This is not a stable analysis with frequency drifts and only calculated for reference.
- ADEV: Overlapping Allan deviation, the most common analysis framework. Square mean value of the second derivate `phase[i] - 2*phase[i + n] + phase[i + 2*n]`. In a loglog plot, the slope allows to identify the source of noise:
  - -1: white or flicker phase noise, like discretization or analog noisy delay
  - -0.5: white period noise

  - 0: flicker period noise, like electric noisy oscillator
  - 0.5: integrated white period noise (random walk period)
  - 1: frequency drift, e.g. thermal

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

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

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

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

## 9.32.2 Constructor & Destructor Documentation

### 9.32.2.1 FrequencyStability()

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

constructor of a FrequencyStability measurement

**Parameters**

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

**Note**

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

### 9.32.2.2 ∼FrequencyStability()

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

### 9.32.3 Member Function Documentation

#### 9.32.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.32.3.2 getDataObject()

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

get a return object with all data in a synchronized way

#### 9.32.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.32.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.33 FrequencyStabilityData Class Reference

return data object for FrequencyStability::getData.

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

### Public Member Functions

- ∼FrequencyStabilityData ()
- void getSTDD (std::function< double ∗(size_t)> array_out)
  *returns the standard derivation of each period pair*
- void getADEV (std::function< double ∗(size_t)> array_out)
  *returns the overlapping Allan deviation*
- void getMDEV (std::function< double ∗(size_t)> array_out)
  *returns the modified overlapping Allan deviation*
- void getTDEV (std::function< double ∗(size_t)> array_out)
  *returns the overlapping time deviation*
- void getHDEV (std::function< double ∗(size_t)> array_out)
  *returns the overlapping Hadamard deviation*
- void getADEVScaled (std::function< double ∗(size_t)> array_out)
  *returns the scaled version of the overlapping Allan deviation*
- void getHDEVScaled (std::function< double ∗(size_t)> array_out)
  *returns the scaled version of the overlapping Hadamard deviation*
- void getTau (std::function< double ∗(size_t)> array_out)
  *returns the analysis position of all deviations*
- void getTracePhase (std::function< double ∗(size_t)> array_out)
  *returns a trace of the last phase samples in seconds*
- void getTraceFrequency (std::function< double ∗(size_t)> array_out)
  *returns a trace of the last normalized frequency error samples in pp1*
- void getTraceFrequencyAbsolute (std::function< double ∗(size_t)> array_out, double input_frequency=0.0)
  *returns a trace of the last absolute frequency samples in Hz*
- void getTraceIndex (std::function< double ∗(size_t)> array_out)
  *returns the timestamps of the traces in seconds*

### 9.33.1 Detailed Description

return data object for FrequencyStability::getData.

## 9.33.2 Constructor & Destructor Documentation

### 9.33.2.1 ∼FrequencyStabilityData()

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

## 9.33.3 Member Function Documentation

### 9.33.3.1 getADEV()

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

returns the overlapping Allan deviation

### 9.33.3.2 getADEVScaled()

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

returns the scaled version of the overlapping Allan deviation

### 9.33.3.3 getHDEV()

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

returns the overlapping Hadamard deviation

### 9.33.3.4 getHDEVScaled()

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

returns the scaled version of the overlapping Hadamard deviation

### 9.33.3.5 getMDEV()

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

returns the modified overlapping Allan deviation

#### 9.33.3.6 getSTDD()

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

returns the standard derivation of each period pair

#### 9.33.3.7 getTau()

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

returns the analysis position of all deviations

#### 9.33.3.8 getTDEV()

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

returns the overlapping time deviation

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

#### 9.33.3.9 getTraceFrequency()

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

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

#### 9.33.3.10 getTraceFrequencyAbsolute()

```
void FrequencyStabilityData::getTraceFrequencyAbsolute (
            std::function< double *(size_t)> array_out,
            double input_frequency = 0.0 )
```

returns a trace of the last absolute frequency samples in Hz

**Parameters**

| | |
|---|---|
| *array_out* | allocator for return array |
| *input_frequency* | reference frequency in Hz |

**Note**

The precision of the parameter input_frequency and so the mean value of the return values are limited to 15 digits. However the relative errors within the return values have a higher precision.

#### 9.33.3.11 getTraceIndex()

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

returns the timestamps of the traces in seconds

#### 9.33.3.12 getTracePhase()

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

returns a trace of the last phase samples in seconds

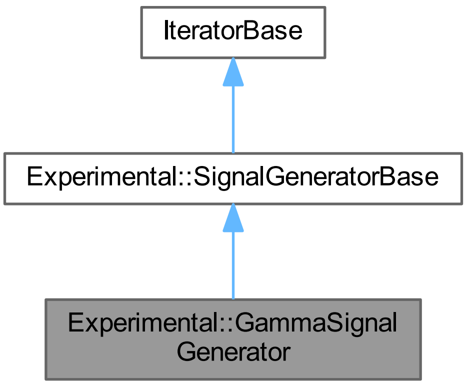
The documentation for this class was generated from the following file:

- Iterators.h

## 9.34 Experimental::GammaSignalGenerator Class Reference

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

Inheritance diagram for Experimental::GammaSignalGenerator:




### Public Member Functions

- GammaSignalGenerator (TimeTaggerBase *tagger, double alpha, double beta, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

  *Construct a gamma event channel.*
- ~GammaSignalGenerator ()

## Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase *tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.34.1 Constructor & Destructor Documentation

### 9.34.1.1 GammaSignalGenerator()

```
Experimental::GammaSignalGenerator::GammaSignalGenerator (
            TimeTaggerBase * tagger,
            double alpha,
            double beta,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a gamma event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *alpha* | alpha value of the gamma distribution |
| *beta* | beta value of the gamma distribution |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.34.1.2 ∼GammaSignalGenerator()

```
Experimental::GammaSignalGenerator::~GammaSignalGenerator ( )
```

## 9.34.2 Member Function Documentation

### 9.34.2.1 get_next()

```
timestamp_t Experimental::GammaSignalGenerator::get_next ( ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.34.2.2 initialize()

```
void Experimental::GammaSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.34.2.3 on_restart()

```
void Experimental::GammaSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

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

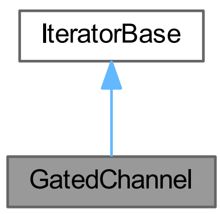
- Iterators.h

## 9.35 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, GatedChannelInitial initial=GatedChannelInitial::Closed)

    *constructor of a GatedChannel*
- ∼GatedChannel ()
- channel_t getChannel ()

    *the new virtual channel*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

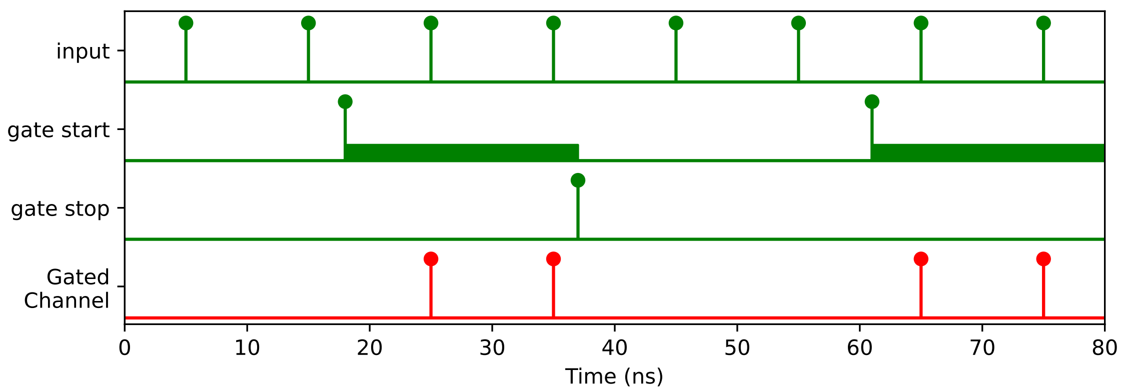
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.35.1 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.

## 9.35.2 Constructor & Destructor Documentation

### 9.35.2.1 GatedChannel()

```
GatedChannel::GatedChannel (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            channel_t gate_start_channel,
            channel_t gate_stop_channel,
            GatedChannelInitial initial = GatedChannelInitial::Closed )
```

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 |
| *initial* | initial state of the gate |

### 9.35.2.2 ∼GatedChannel()

```
GatedChannel::∼GatedChannel ( )
```

### 9.35.3 Member Function Documentation

#### 9.35.3.1 clear_impl()

```
void GatedChannel::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.

#### 9.35.3.2 getChannel()

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

the new virtual channel

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

#### 9.35.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

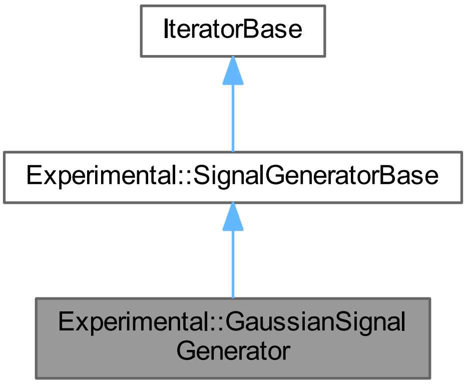
The documentation for this class was generated from the following file:

- Iterators.h

## 9.36 Experimental::GaussianSignalGenerator Class Reference

`#include <Iterators.h>`

Inheritance diagram for Experimental::GaussianSignalGenerator:




### Public Member Functions

- GaussianSignalGenerator (TimeTaggerBase ∗tagger, double mean, double standard_deviation, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

  *Construct a gaussian event channel.*
- ∼GaussianSignalGenerator ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

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

- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

#### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*

- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.36.1 Constructor & Destructor Documentation

### 9.36.1.1 GaussianSignalGenerator()

```
Experimental::GaussianSignalGenerator::GaussianSignalGenerator (
            TimeTaggerBase * tagger,
            double mean,
            double standard_deviation,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a gaussian event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *mean* | mean time each event is generated. |
| *standard_deviation* | standard deviation of the normal distribution. |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

#### 9.36.1.2 ∼GaussianSignalGenerator()

```
Experimental::GaussianSignalGenerator::~GaussianSignalGenerator ( )
```

### 9.36.2 Member Function Documentation

#### 9.36.2.1 get_next()

```
timestamp_t Experimental::GaussianSignalGenerator::get_next ( ) [override], [protected],
[virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.36.2.2 initialize()

```
void Experimental::GaussianSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.36.2.3 on_restart()

```
void Experimental::GaussianSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

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

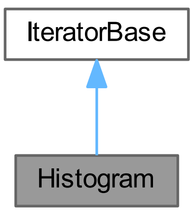
- Iterators.h

## 9.37 Histogram Class Reference

Accumulate time differences into a histogram.

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

Inheritance diagram for Histogram:



### Public Member Functions

- Histogram (TimeTaggerBase ∗tagger, channel_t click_channel, channel_t start_channel=CHANNEL_UNUSED, timestamp_t binwidth=1000, int32_t n_bins=1000)

    *constructor of a Histogram measurement*
- ∼Histogram ()
- void getData (std::function< int32_t ∗(size_t)> array_out)
- void getIndex (std::function< timestamp_t ∗(size_t)> array_out)

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

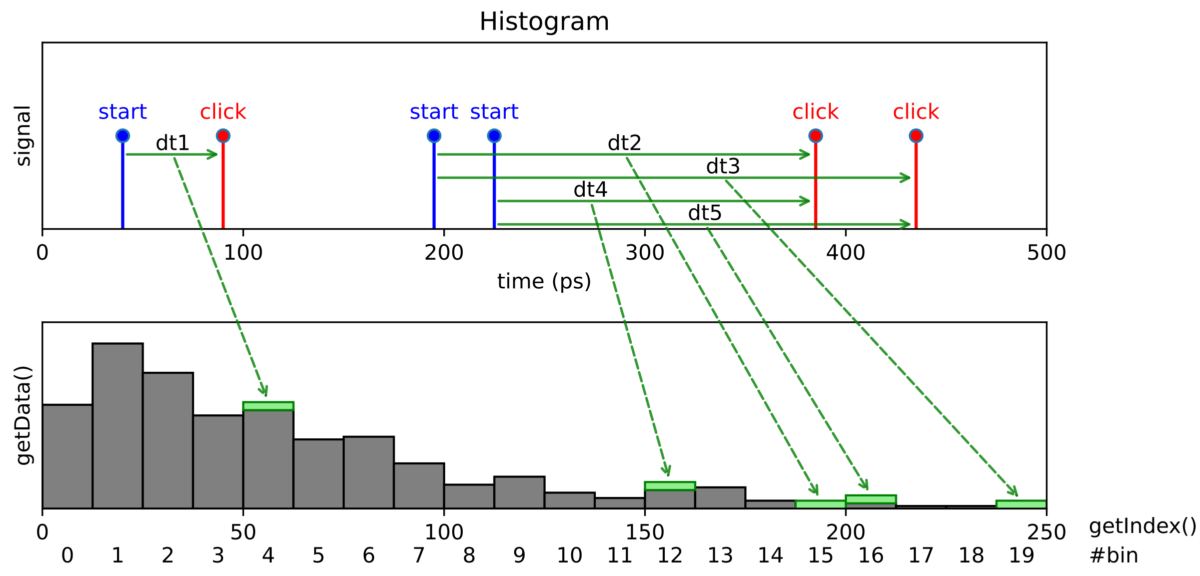
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.37.1 Detailed Description

Accumulate time differences into a histogram.




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

### 9.37.2 Constructor & Destructor Documentation

#### 9.37.2.1 Histogram()

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

constructor of a Histogram measurement

**Parameters**

| | |
|---|---|
| *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 |

#### 9.37.2.2 ∼Histogram()

```
Histogram::∼Histogram ( )
```

### 9.37.3 Member Function Documentation

#### 9.37.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.37.3.2 getData()

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

#### 9.37.3.3 getIndex()

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

#### 9.37.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.37.3.5 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

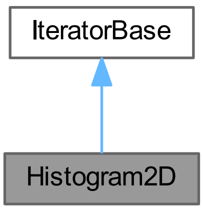
- Iterators.h

## 9.38 Histogram2D Class Reference

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

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

Inheritance diagram for Histogram2D:




### Public Member Functions

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

    *constructor of a Histogram2D measurement*
- ~Histogram2D ()
- void getData (std::function< int32_t *(size_t, size_t)> array_out)
- void getIndex (std::function< timestamp_t *(size_t, size_t, size_t)> array_out)
- void getIndex_1 (std::function< timestamp_t *(size_t)> array_out)
- void getIndex_2 (std::function< timestamp_t *(size_t)> array_out)

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

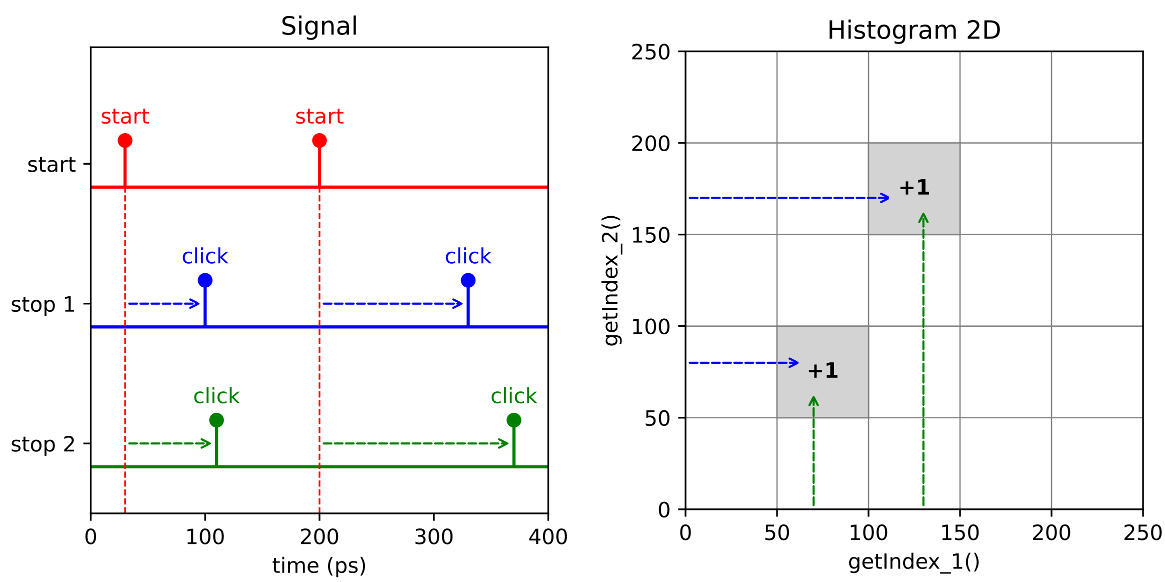
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.38.1 Detailed Description

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



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

## 9.38.2 Constructor & Destructor Documentation

### 9.38.2.1 Histogram2D()

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

constructor of a Histogram2D measurement

**Parameters**

| | |
|---|---|
| *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 |

### 9.38.2.2 ∼Histogram2D()

```
Histogram2D::~Histogram2D ( )
```

## 9.38.3 Member Function Documentation

### 9.38.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.38.3.2 getData()

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

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

#### 9.38.3.3 getIndex()

```
void Histogram2D::getIndex (
            std::function< timestamp_t *(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

#### 9.38.3.4 getIndex_1()

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

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

#### 9.38.3.5 getIndex_2()

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

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

#### 9.38.3.6 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

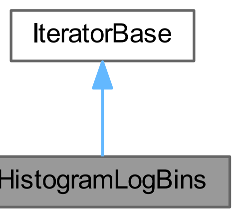
- Iterators.h

## 9.39 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, const ChannelGate *click_gate=nullptr, const ChannelGate *start_gate=nullptr)

    *constructor of a HistogramLogBins measurement*
- ~HistogramLogBins ()
- HistogramLogBinsData getDataObject ()
- void getData (std::function< uint64_t *(size_t)> array_out)

    *returns the absolute counts for the bins*
- void getDataNormalizedCountsPerPs (std::function< double *(size_t)> array_out)

    *returns the counts normalized by the binwidth of each bin*
- void getDataNormalizedG2 (std::function< double *(size_t)> array_out)

    *returns the counts normalized by the binwidth and the average count rate.*
- void getBinEdges (std::function< timestamp_t *(size_t)> array_out)

    *returns the edges of the bins in ps*

## Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

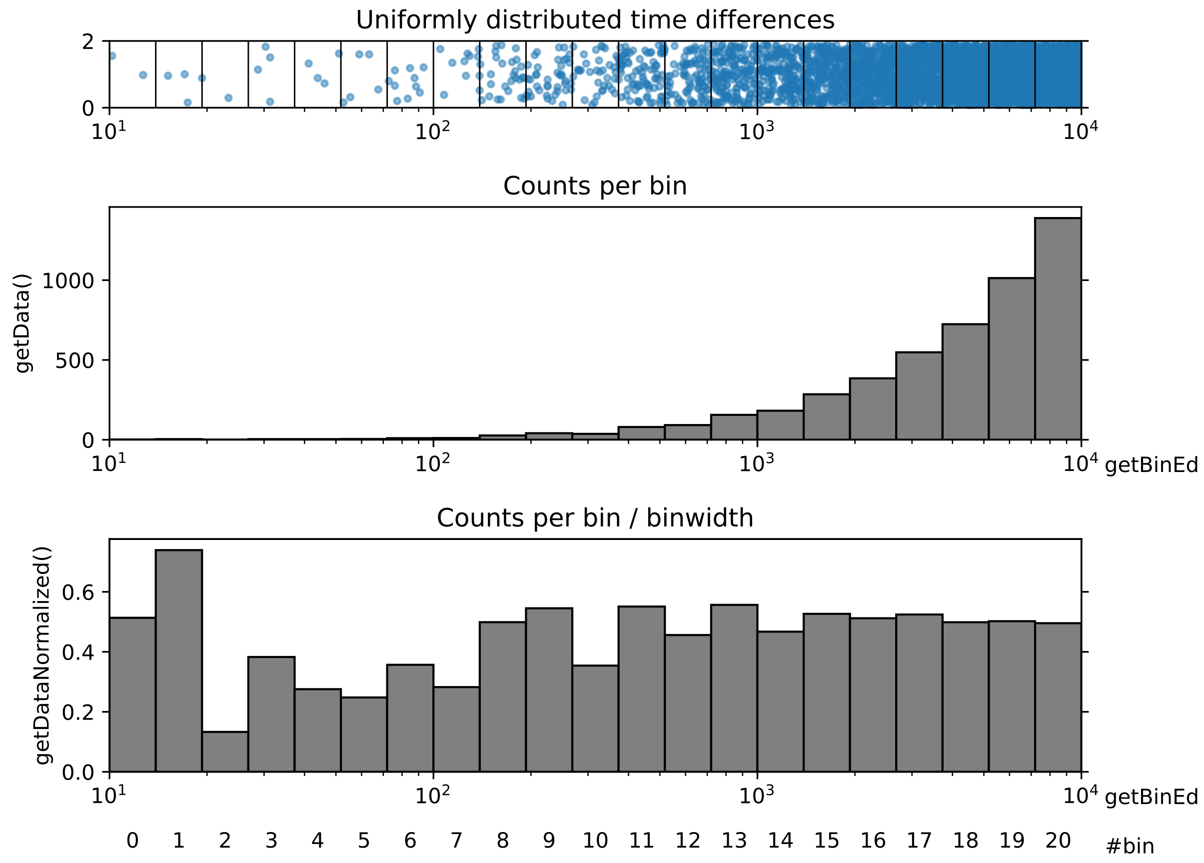
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.39.1 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.

## 9.39.2 Constructor & Destructor Documentation

### 9.39.2.1 HistogramLogBins()

```
HistogramLogBins::HistogramLogBins (
            TimeTaggerBase * tagger,
            channel_t click_channel,
            channel_t start_channel,
            double exp_start,
            double exp_stop,
            int32_t n_bins,
            const ChannelGate * click_gate = nullptr,
            const ChannelGate * start_gate = nullptr )
```

constructor of a HistogramLogBins measurement

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *click_channel* | channel that increments the count in a bin |

**Parameters**

| | |
|---|---|
| *start_channel* | channel that sets start times relative to which clicks on the click channel are measured |
| *exp_start* | exponent for the lowest time differences in the histogram: 10^exp_start s, lowest exp_start: -12 => 1ps |
| *exp_stop* | exponent for the highest time differences in the histogram: 10^exp_stop s |
| *n_bins* | total number of bins in the histogram |
| *click_gate* | ChannelGate object for toggling the click_channel, nullptr if unused |
| *start_gate* | ChannelGate object for toggling the start_channel, nullptr if unused |

#### 9.39.2.2 ∼HistogramLogBins()

```
HistogramLogBins::∼HistogramLogBins ( )
```

### 9.39.3 Member Function Documentation

#### 9.39.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.39.3.2 getBinEdges()

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

returns the edges of the bins in ps

#### 9.39.3.3 getData()

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

returns the absolute counts for the bins

#### 9.39.3.4 getDataNormalizedCountsPerPs()

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

returns the counts normalized by the binwidth of each bin

#### 9.39.3.5 getDataNormalizedG2()

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

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

This matches the implementation of Correlation::getDataNormalized

#### 9.39.3.6 getDataObject()

```
HistogramLogBinsData HistogramLogBins::getDataObject ( )
```

#### 9.39.3.7 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

- Iterators.h

## 9.40 HistogramLogBinsData Class Reference

Helper object as return value for HistogramLogBins::getDataObject.

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

**Public Member Functions**

- ∼HistogramLogBinsData ()
- void getCounts (std::function< uint64_t ∗(size_t)> array_out)

    *Get the amount of clicks per bin and per channel.*
- void getG2Normalization (std::function< double ∗(size_t)> array_out)

    *Get the calculated normalization for each bin.*
- void getG2 (std::function< double ∗(size_t)> array_out)

    *Get the normalized histogram.*

**Public Attributes**

- const timestamp_t accumulation_time_start
- const timestamp_t accumulation_time_click

## 9.40.1 Detailed Description

Helper object as return value for HistogramLogBins::getDataObject.

This object stores counts and normalization of the logarithmic bins.

## 9.40.2 Constructor & Destructor Documentation

### 9.40.2.1 ∼HistogramLogBinsData()

```
HistogramLogBinsData::~HistogramLogBinsData ( )
```

## 9.40.3 Member Function Documentation

### 9.40.3.1 getCounts()

```
void HistogramLogBinsData::getCounts (
            std::function< uint64_t *(size_t)> array_out )
```

Get the amount of clicks per bin and per channel.

### 9.40.3.2 getG2()

```
void HistogramLogBinsData::getG2 (
            std::function< double *(size_t)> array_out )
```

Get the normalized histogram.

#### 9.40.3.3 getG2Normalization()

```
void HistogramLogBinsData::getG2Normalization (
            std::function< double *(size_t)> array_out )
```

Get the calculated normalization for each bin.

### 9.40.4 Member Data Documentation

#### 9.40.4.1 accumulation_time_click

```
const timestamp_t HistogramLogBinsData::accumulation_time_click
```

#### 9.40.4.2 accumulation_time_start

```
const timestamp_t HistogramLogBinsData::accumulation_time_start
```

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

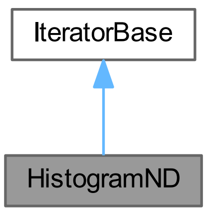
- Iterators.h

## 9.41 HistogramND Class Reference

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

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

Inheritance diagram for HistogramND:




**Public Member Functions**

- HistogramND (TimeTaggerBase *tagger, channel_t start_channel, std::vector< channel_t > stop_channels, std::vector< timestamp_t > binwidths, std::vector< int32_t > n_bins)

    *constructor of a Histogram2D measurement*
- ∼HistogramND ()
- void getData (std::function< int32_t *(size_t)> array_out)
- void getIndex (std::function< timestamp_t *(size_t)> array_out, int32_t dim=0)

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.41.1 Detailed Description

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

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

## 9.41.2 Constructor & Destructor Documentation

### 9.41.2.1 HistogramND()

```
HistogramND::HistogramND (
            TimeTaggerBase * tagger,
            channel_t start_channel,
            std::vector< channel_t > stop_channels,
            std::vector< timestamp_t > binwidths,
            std::vector< int32_t > n_bins )
```

constructor of a Histogram2D measurement

**Parameters**

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

#### 9.41.2.2 ∼HistogramND()

```
HistogramND::∼HistogramND ( )
```

### 9.41.3 Member Function Documentation

#### 9.41.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.41.3.2 getData()

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

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

#### 9.41.3.3 getIndex()

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

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

#### 9.41.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

- Iterators.h

## 9.42 HistogramNDImpl< T > Class Template Reference

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

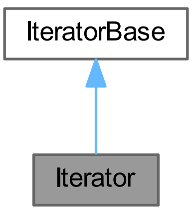
- Iterators.h

## 9.43 Iterator Class Reference

a deprecated simple event queue

`#include <Iterators.h>`

Inheritance diagram for Iterator:



### Public Member Functions

- Iterator (TimeTaggerBase *tagger, channel_t channel)
  *standard constructor*
- ∼Iterator ()
- timestamp_t next ()
  *get next timestamp*
- uint64_t size ()
  *get queue size*

## Public Member Functions inherited from IteratorBase

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

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.43.1 Detailed Description

a deprecated simple event queue

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

## 9.43.2 Constructor & Destructor Documentation

### 9.43.2.1 Iterator()

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

standard constructor

**Parameters**

| | |
|---|---|
| *tagger* | the backend |
| *channel* | the channel to get events from |

### 9.43.2.2 ∼Iterator()

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

## 9.43.3 Member Function Documentation

### 9.43.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.43.3.2 next()

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

get next timestamp

get the next timestamp from the queue.

#### 9.43.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.43.3.4 size()

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

get queue size

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

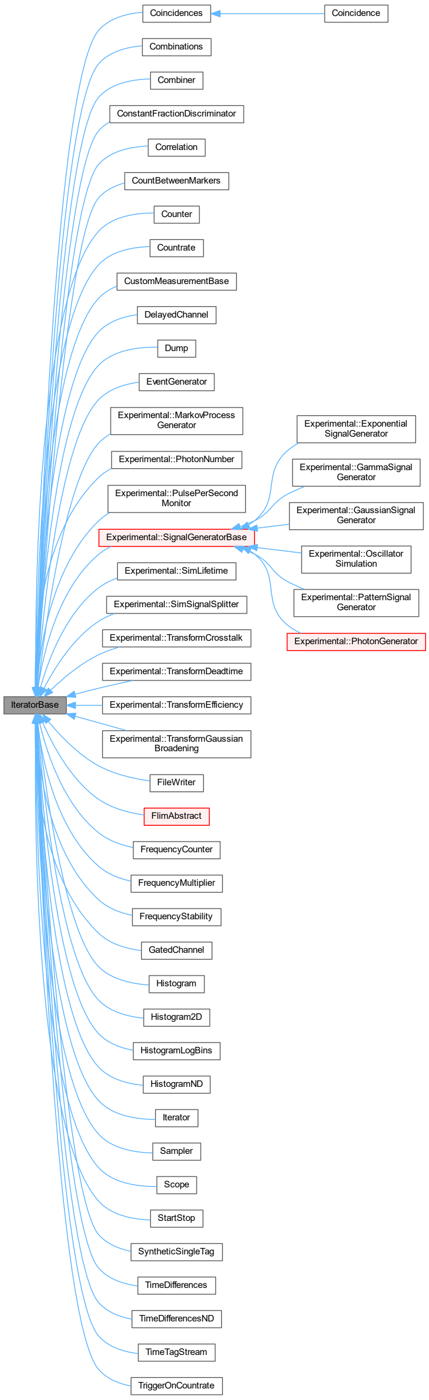
- Iterators.h

## 9.44 IteratorBase Class Reference

Base class for all iterators.

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

Inheritance diagram for IteratorBase:




## Classes

- class AbortError

  *A custom runtime error thrown by the `abort` call. This can be caught and handled by measurement classes, including `CustomMeasurement`, to perform actions within the abortion process.*

### Public Member Functions

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

### Protected Member Functions

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

- virtual bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_↩ time)=0

  *update iterator state*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

#### Protected Attributes

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.44.1 Detailed Description

Base class for all iterators.

## 9.44.2 Constructor & Destructor Documentation

### 9.44.2.1 IteratorBase()

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

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

### 9.44.2.2 ∼IteratorBase()

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

destructor, will unregister from the Time Tagger prior finalization.

### 9.44.3 Member Function Documentation

#### 9.44.3.1 abort()

```
void IteratorBase::abort ( )
```

Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.

**Warning**

After calling `abort`, the last block of data might become irreversibly corrupted. Please always use `stop` to end a measurement.

#### 9.44.3.2 checkForAbort() [1/2]

```
void IteratorBase::checkForAbort ( )  [inline], [protected]
```

#### 9.44.3.3 checkForAbort() [2/2]

```
template<typename T >
void IteratorBase::checkForAbort (
            T callback )  [inline], [protected]
```

#### 9.44.3.4 clear()

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

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

#### 9.44.3.5 clear_impl()

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

clear Iterator state.

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

Reimplemented in Combiner, CountBetweenMarkers, Counter, Countrate, TriggerOnCountrate, GatedChannel, FrequencyMultiplier, Iterator, TimeTagStream, Dump, StartStop, TimeDifferences, Histogram2D, HistogramND, TimeDifferencesND, Histogram, FrequencyCounter, HistogramLogBins, Correlation, Scope, FileWriter, EventGenerator, Combinations, CustomMeasurementBase, FlimAbstract, Flim, Sampler, FrequencyStability, Experimental::PulsePerSecondMonitor, and Experimental::PhotonNumber.

#### 9.44.3.6 finish_running()

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

Callback for the measurement to stop itself.

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

#### 9.44.3.7 finishInitialization()

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

method to call after finishing the initialization of the measurement

#### 9.44.3.8 getCaptureDuration()

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

Total capture duration since the measurement creation or last call to clear().

**Returns**

Capture duration in ps

#### 9.44.3.9 getConfiguration()

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

Fetches the overall configuration status of the measurement.

**Returns**

a JSON serialized string with all configuration and status flags.

#### 9.44.3.10 getLock()

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

acquire update lock

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

**Returns**

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

#### 9.44.3.11 getNewVirtualChannel()

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

allocate a new virtual output channel for this iterator

#### 9.44.3.12 isRunning()

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

Returns True if the measurement is collecting the data.

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

**Note**

All measurements start accumulating data immediately after their creation.

**Returns**

True if the measurement is still running

#### 9.44.3.13 lock()

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

acquire update lock

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

#### 9.44.3.14 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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 Combiner, CountBetweenMarkers, Counter, Coincidences, Countrate, DelayedChannel, TriggerOnCountrate, GatedChannel, FrequencyMultiplier, Iterator, TimeTagStream, Dump, StartStop, TimeDifferences, Histogram2D, HistogramND, TimeDifferencesND, Histogram, FrequencyCounter, HistogramLogBins, Correlation, Scope, ConstantFractionDiscriminator, FileWriter, EventGenerator, Combinations, CustomMeasurementBase, FlimAbstract, Sampler, SyntheticSingleTag, FrequencyStability, Experimental::PulsePerSecondMonitor, Experimental::SignalGenerat Experimental::MarkovProcessGenerator, Experimental::SimSignalSplitter, Experimental::TransformEfficiency, Experimental::TransformGaussianBroadening, Experimental::TransformDeadtime, Experimental::TransformCrosstalk, Experimental::SimLifetime, and Experimental::PhotonNumber.

#### 9.44.3.15 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented in Counter, Countrate, DelayedChannel, TriggerOnCountrate, Dump, StartStop, TimeDifferences, TimeDifferencesND, Histogram, FrequencyCounter, ConstantFractionDiscriminator, FileWriter, EventGenerator, CustomMeasurementBase, FlimAbstract, Sampler, FrequencyStability, and Experimental::PulsePerSecondMonitor.

#### 9.44.3.16 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented in Dump, FileWriter, CustomMeasurementBase, Experimental::SignalGeneratorBase, and Experimental::MarkovProcessGenerator.

#### 9.44.3.17 parallelize()

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

release lock and continue work in parallel

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

**Returns**

a ordered barrier instance that can be synced afterwards.

#### 9.44.3.18 registerChannel()

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

register a channel

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

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.44.3.19 start()

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

Starts or continues data acquisition.

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

#### 9.44.3.20 startFor()

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

Starts or continues the data acquisition for the given duration.

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

**Parameters**

| | |
|---|---|
| *capture_duration* | capture duration in picoseconds until the measurement is stopped |
| *clear* | resets the data acquired |

#### 9.44.3.21 stop()

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

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

Use start() or startFor() to continue or restart the measurement.

#### 9.44.3.22 unlock()

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

release update lock

see lock()

#### 9.44.3.23 unregisterChannel()

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

unregister a channel

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.44.3.24 waitUntilFinished()

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

Blocks the execution until the measurement has finished. Can be used with startFor().

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

**Parameters**

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

**Returns**

True if the measurement has finished, false on timeout

### 9.44.4 Member Data Documentation

#### 9.44.4.1 aborting

```
std::atomic<bool> IteratorBase::aborting [protected]
```

#### 9.44.4.2 autostart

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

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

#### 9.44.4.3 capture_duration

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

Duration the iterator has already processed data.

#### 9.44.4.4 channels_registered

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

list of channels used by the iterator

#### 9.44.4.5 pre_capture_duration

```
timestamp_t IteratorBase::pre_capture_duration [protected]
```

For internal use.

#### 9.44.4.6 running

```
bool IteratorBase::running [protected]
```

running state of the iterator

#### 9.44.4.7 tagger

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

Pointer to the corresponding Time Tagger object.

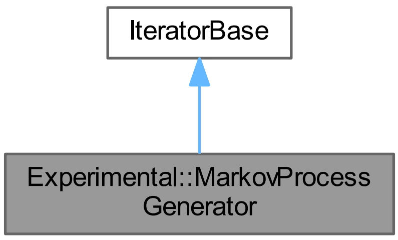
The documentation for this class was generated from the following file:

- TimeTagger.h

## 9.45 Experimental::MarkovProcessGenerator Class Reference

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

Inheritance diagram for Experimental::MarkovProcessGenerator:




**Public Member Functions**

- MarkovProcessGenerator (TimeTaggerBase *tagger, uint64_t num_states, std::vector< double > frequencies, std::vector< channel_t > ref_channels, std::vector< channel_t > base_channels=std::vector< channel_t >(), int32_t seed=-1)

  *Construct a continuous-time Markov chain process.*
- ∼MarkovProcessGenerator ()
- channel_t getChannel ()
- std::vector< channel_t > getChannels ()

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.45.1 Constructor & Destructor Documentation

### 9.45.1.1 MarkovProcessGenerator()

```
Experimental::MarkovProcessGenerator::MarkovProcessGenerator (
            TimeTaggerBase * tagger,
            uint64_t num_states,
            std::vector< double > frequencies,
            std::vector< channel_t > ref_channels,
            std::vector< channel_t > base_channels = std::vector< channel_t >(),
            int32_t seed = -1 )
```

Construct a continuous-time Markov chain process.

https://en.wikipedia.org/wiki/Continuous-time_Markov_chain

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *num_states* | Number of exponential states. |
| *frequencies* | frequencies of each state transition, it's size is num_states ∗ num_states. |
| *ref_channels* | tells the net channel to look at on a state transition. its size is num_states ∗ num_states. |
| *base_channels* | channels in which to generate or add the new timetags if CHANNEL_UNUSED or empty, generate a new virtual channel |

#### 9.45.1.2 ∼MarkovProcessGenerator()

```
Experimental::MarkovProcessGenerator::~MarkovProcessGenerator ( )
```

### 9.45.2 Member Function Documentation

#### 9.45.2.1 getChannel()

```
channel_t Experimental::MarkovProcessGenerator::getChannel ( )
```

#### 9.45.2.2 getChannels()

```
std::vector< channel_t > Experimental::MarkovProcessGenerator::getChannels ( )
```

#### 9.45.2.3 next_impl()

```
bool Experimental::MarkovProcessGenerator::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.

#### 9.45.2.4 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

# 9.46 OrderedBarrier Class Reference

Helper for implementing parallel measurements.

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

### Classes

- class OrderInstance

    *Internal object for serialization.*

### Public Member Functions

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

## 9.46.1 Detailed Description

Helper for implementing parallel measurements.

## 9.46.2 Constructor & Destructor Documentation

### 9.46.2.1 OrderedBarrier()

```
OrderedBarrier::OrderedBarrier ( )
```

### 9.46.2.2 ∼OrderedBarrier()

```
OrderedBarrier::~OrderedBarrier ( )
```

## 9.46.3 Member Function Documentation

### 9.46.3.1 queue()

```
OrderInstance OrderedBarrier::queue ( )
```

### 9.46.3.2 waitUntilFinished()

```
void OrderedBarrier::waitUntilFinished ( )
```

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

- TimeTagger.h

## 9.47 OrderedPipeline Class Reference

Helper for implementing parallel measurements.

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

#### Public Member Functions

- OrderedPipeline ()
- ~OrderedPipeline ()

### 9.47.1 Detailed Description

Helper for implementing parallel measurements.

### 9.47.2 Constructor & Destructor Documentation

#### 9.47.2.1 OrderedPipeline()

```
OrderedPipeline::OrderedPipeline ( )
```

#### 9.47.2.2 ~OrderedPipeline()

```
OrderedPipeline::~OrderedPipeline ( )
```

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

- TimeTagger.h

## 9.48 OrderedBarrier::OrderInstance Class Reference

Internal object for serialization.

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

#### Public Member Functions

- OrderInstance ()
- OrderInstance (OrderedBarrier *parent, uint64_t instance_id)
- ~OrderInstance ()
- void sync ()
- void release ()

## 9.48.1 Detailed Description

Internal object for serialization.

## 9.48.2 Constructor & Destructor Documentation

### 9.48.2.1 OrderInstance() [1/2]

```
OrderedBarrier::OrderInstance::OrderInstance ( )
```

### 9.48.2.2 OrderInstance() [2/2]

```
OrderedBarrier::OrderInstance::OrderInstance (
            OrderedBarrier * parent,
            uint64_t instance_id )
```

### 9.48.2.3 ∼OrderInstance()

```
OrderedBarrier::OrderInstance::∼OrderInstance ( )
```

## 9.48.3 Member Function Documentation

### 9.48.3.1 release()

```
void OrderedBarrier::OrderInstance::release ( )
```

### 9.48.3.2 sync()

```
void OrderedBarrier::OrderInstance::sync ( )
```

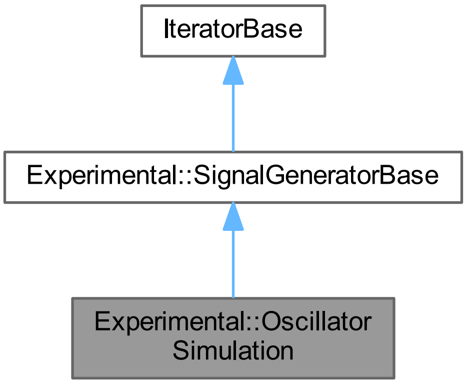
The documentation for this class was generated from the following file:

- TimeTagger.h

## 9.49 Experimental::OscillatorSimulation Class Reference

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

Inheritance diagram for Experimental::OscillatorSimulation:




#### Public Member Functions

- OscillatorSimulation (TimeTaggerBase *tagger, double nominal_frequency, double coeff_phase_white=0.0, double coeff_phase_flicker=0.0, double coeff_freq_white=0.0, double coeff_freq_flicker=0.0, double coeff_random_drift=0.0, double coeff_linear_drift=0.0, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

    *Construct a simulated oscillator event channel.*
- ∼OscillatorSimulation ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase *tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

    *the new virtual channel*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

*Blocks the execution until the measurement has finished. Can be used with startFor().*

- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

**Protected Member Functions**

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*

- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

### Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.49.1 Constructor & Destructor Documentation

### 9.49.1.1 OscillatorSimulation()

```
Experimental::OscillatorSimulation::OscillatorSimulation (
            TimeTaggerBase * tagger,
            double nominal_frequency,
            double coeff_phase_white = 0.0,
            double coeff_phase_flicker = 0.0,
            double coeff_freq_white = 0.0,
            double coeff_freq_flicker = 0.0,
            double coeff_random_drift = 0.0,
            double coeff_linear_drift = 0.0,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a simulated oscillator event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *nominal_frequency* | Normal frequency of the oscillator in Hz |
| *coeff_phase_white* | RMS value of the white phase noise in seconds. |
| *coeff_phase_flicker* | RMS value of the flicker phase noise in seconds per octave. |
| *coeff_freq_white* | Scaling parameter for the white frequency modulated noise in sqrt(s), use `10e-12 * sqrt(1e-3)` for 10 ppt RMS error at 1 kHz cutoff frequency. |
| *coeff_freq_flicker* | Scaling parameter for the relative flicker frequency modulated noise, use `10e-12` for 10 ppt error per octave. |
| *coeff_random_drift* | Scaling parameter for the random walk drift in sqrt(Hz), use `10e-9 / sqrt(60*60*24)` for 10 ppb / sqrt(day). |
| *coeff_linear_drift* | Scaling parameter for the relative linear frequency drift in Hz, use `1e-6 / (60*60*24*365)` for 1 ppm / year. |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

#### 9.49.1.2 ∼OscillatorSimulation()

```
Experimental::OscillatorSimulation::~OscillatorSimulation ( )
```

### 9.49.2 Member Function Documentation

#### 9.49.2.1 get_next()

```
timestamp_t Experimental::OscillatorSimulation::get_next ( ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.49.2.2 initialize()

```
void Experimental::OscillatorSimulation::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.49.2.3 on_restart()

```
void Experimental::OscillatorSimulation::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

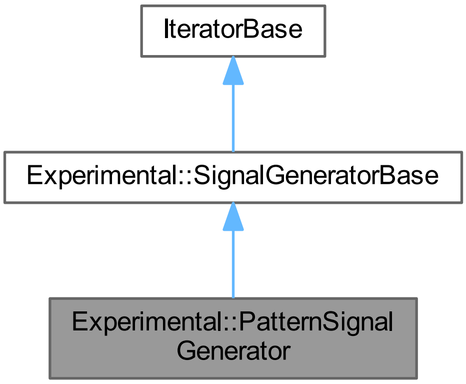
The documentation for this class was generated from the following file:

- Iterators.h

## 9.50 Experimental::PatternSignalGenerator Class Reference

`#include <Iterators.h>`

Inheritance diagram for Experimental::PatternSignalGenerator:




### Public Member Functions

- PatternSignalGenerator (TimeTaggerBase ∗tagger, std::vector< timestamp_t > sequence, bool repeat=false, timestamp_t start_delay=0, timestamp_t spacing=0, channel_t base_channel=CHANNEL_UNUSED)

  *Construct a pattern event generator.*
- ∼PatternSignalGenerator ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

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

- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*

- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

### Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.50.1 Constructor & Destructor Documentation

### 9.50.1.1 PatternSignalGenerator()

```
Experimental::PatternSignalGenerator::PatternSignalGenerator (
            TimeTaggerBase * tagger,
            std::vector< timestamp_t > sequence,
            bool repeat = false,
            timestamp_t start_delay = 0,
            timestamp_t spacing = 0,
            channel_t base_channel = CHANNEL_UNUSED )
```

Construct a pattern event generator.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *sequence* | sequence of offsets pattern to be used continuously. |
| *repeat* | tells if to repeat the pattern or only generate it once. |
| *start_delay* | initial delay before the first pattern is applied. |
| *spacing* | delay between pattern repetitions. |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |

#### 9.50.1.2 ∼PatternSignalGenerator()

```
Experimental::PatternSignalGenerator::~PatternSignalGenerator ( )
```

### 9.50.2 Member Function Documentation

#### 9.50.2.1 get_next()

```
timestamp_t Experimental::PatternSignalGenerator::get_next ( ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.50.2.2 initialize()

```
void Experimental::PatternSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.50.2.3 on_restart()

```
void Experimental::PatternSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

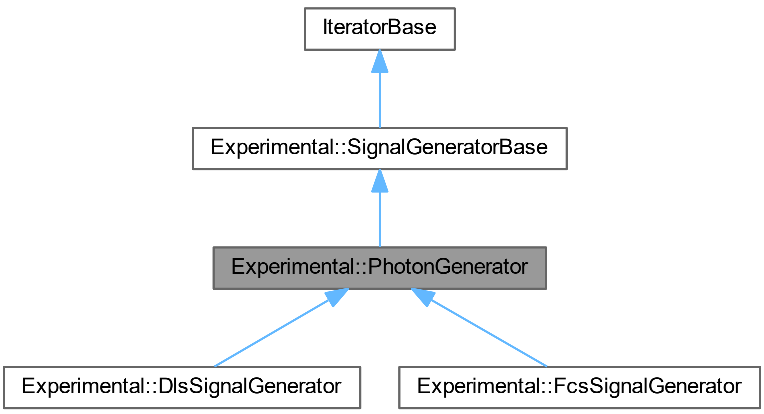
The documentation for this class was generated from the following file:

- Iterators.h

## 9.51 Experimental::PhotonGenerator Class Reference

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

Inheritance diagram for Experimental::PhotonGenerator:



### Public Member Functions

- PhotonGenerator (TimeTaggerBase *tagger, double countrate, channel_t base_channel, int32_t seed=-1)

  *A generator for TimeTags arising from a laser driven process.* `PhotonGenerator` *should be used as the base class of a virtual class with a dedicated* `get_intensity` *function which models the relevant physical processes.*
- ∼PhotonGenerator ()
- void finalize_init ()
- void set_T_PERIOD (timestamp_t new_T)
- timestamp_t get_T_PERIOD ()

## Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase *tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- void on_restart (timestamp_t restart_time) override
- timestamp_t get_next () override
- virtual double get_intensity ()=0

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Protected Attributes

- timestamp_t T_PERIOD

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.51.1 Constructor & Destructor Documentation

### 9.51.1.1 PhotonGenerator()

```
Experimental::PhotonGenerator::PhotonGenerator (
            TimeTaggerBase * tagger,
            double countrate,
            channel_t base_channel,
            int32_t seed = -1 )
```

A generator for TimeTags arising from a laser driven process. `PhotonGenerator` should be used as the base class of a virtual class with a dedicated `get_intensity` function which models the relevant physical processes.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger. |
| *countrate* | rate (in Hz) of Time Tags to be generated. |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.51.1.2 ∼PhotonGenerator()

```
Experimental::PhotonGenerator::~PhotonGenerator ( )
```

## 9.51.2 Member Function Documentation

### 9.51.2.1 finalize_init()

```
void Experimental::PhotonGenerator::finalize_init ( )
```

#### 9.51.2.2 get_intensity()

```
virtual double Experimental::PhotonGenerator::get_intensity ( )  [protected], [pure virtual]
```

Implemented in Experimental::DlsSignalGenerator, and Experimental::FcsSignalGenerator.

#### 9.51.2.3 get_next()

```
timestamp_t Experimental::PhotonGenerator::get_next ( )  [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.51.2.4 get_T_PERIOD()

```
timestamp_t Experimental::PhotonGenerator::get_T_PERIOD ( )
```

#### 9.51.2.5 initialize()

```
void Experimental::PhotonGenerator::initialize (
            timestamp_t initial_time )  [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.51.2.6 on_restart()

```
void Experimental::PhotonGenerator::on_restart (
            timestamp_t restart_time )  [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

#### 9.51.2.7 set_T_PERIOD()

```
void Experimental::PhotonGenerator::set_T_PERIOD (
            timestamp_t new_T )
```

### 9.51.3 Member Data Documentation

#### 9.51.3.1 T_PERIOD

```
timestamp_t Experimental::PhotonGenerator::T_PERIOD  [protected]
```

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

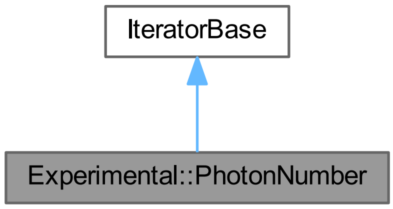
- Iterators.h

## 9.52 Experimental::PhotonNumber Class Reference

Photon number resolution.

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

Inheritance diagram for Experimental::PhotonNumber:




### Public Member Functions

- PhotonNumber (TimeTaggerBase *tagger, channel_t trigger_ch, channel_t signal_start_ch, channel_t signal_stop_ch, double slope, std::vector< double > x_intercepts, timestamp_t dead_time)

    *construct a PhotonNumber*
- ~PhotonNumber ()
- std::vector< channel_t > const & getChannels () const

    *the new virtual channels*

### Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.52.1 Detailed Description

Photon number resolution.

## 9.52.2 Constructor & Destructor Documentation

### 9.52.2.1 PhotonNumber()

```
Experimental::PhotonNumber::PhotonNumber (
            TimeTaggerBase * tagger,
            channel_t trigger_ch,
            channel_t signal_start_ch,
            channel_t signal_stop_ch,
            double slope,
            std::vector< double > x_intercepts,
            timestamp_t dead_time )
```

construct a PhotonNumber

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *trigger_ch* | trigger channel |
| *signal_start_ch* | start-of-signal channel (likely rising edge) |
| *signal_stop_ch* | end-of-signal channel (likely falling edge) |
| *slope* | common slope of decision boundary lines |
| *x_intercepts* | x-intercepts of decision boundary lines. Has to be in descending order |
| *dead_time* | the dead time of the detector |

### 9.52.2.2 ∼PhotonNumber()

```
Experimental::PhotonNumber::~PhotonNumber ( )
```

## 9.52.3 Member Function Documentation

### 9.52.3.1 clear_impl()

```
void Experimental::PhotonNumber::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.

#### 9.52.3.2 getChannels()

```
std::vector< channel_t > const & Experimental::PhotonNumber::getChannels ( ) const
```

the new virtual channels

This function returns the IDs of the allocated virtual channels, corresponding to photon counts of $1, \ldots, N, \geq N+1$ for given $N$ decision boundary lines.

#### 9.52.3.3 next_impl()

```
bool Experimental::PhotonNumber::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.

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

- Iterators.h

## 9.53 Experimental::PulsePerSecondData Class Reference

Helper object as return value for PulsePerSecondMonitor::getDataObject.

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

**Public Member Functions**

- void getIndices (std::function< int64_t *(size_t)> array_out)

    *The indices of each reference pulse in the `PulsePerSecondData` object. In case of overflows in the reference channel, this index will be incremented by the number of missed pulses.*
- void getReferenceOffsets (std::function< double *(size_t)> array_out)

    *A list of offsets of each reference pulse with respective to its predecessor, with the period subtracted.*
- void getSignalOffsets (std::function< double *(size_t, size_t)> array_out)

    *For each reference contained in the `PulsePerSecondData` object a list of offsets for each signal channel is given, in the channel order given by `signal_channels`.*
- void getUtcSeconds (std::function< double *(size_t)> array_out)

    *The number of elapsed seconds from the beginning of the Unix epoch (1st of January 1970) to the time at which each reference pulse is processed, as a floating point number.*
- std::vector< std::string > getUtcDates ()

    *The UTC timestamps for the system time at which each reference pulse is processed, as a string with ISO 8601 formatting.*
- void getStatus (std::function< bool *(size_t)> array_out)

    *A vector of booleans values describing whether all signals, including from the reference source, were detected.*
- ~PulsePerSecondData ()

**Public Attributes**

- const size_t size

    *Number of reference pulses contained in the `PulsePerSecondData` object.*

## 9.53.1 Detailed Description

Helper object as return value for PulsePerSecondMonitor::getDataObject.

This object stores the results of all monitored PPS pulses.

## 9.53.2 Constructor & Destructor Documentation

### 9.53.2.1 ~PulsePerSecondData()

```
Experimental::PulsePerSecondData::~PulsePerSecondData ( )
```

## 9.53.3 Member Function Documentation

### 9.53.3.1 getIndices()

```
void Experimental::PulsePerSecondData::getIndices (
            std::function< int64_t *(size_t)> array_out )
```

The indices of each reference pulse in the `PulsePerSecondData` object. In case of overflows in the reference channel, this index will be incremented by the number of missed pulses.

#### 9.53.3.2 getReferenceOffsets()

```
void Experimental::PulsePerSecondData::getReferenceOffsets (
            std::function< double *(size_t)> array_out )
```

A list of offsets of each reference pulse with respective to its predecessor, with the period subtracted.

#### 9.53.3.3 getSignalOffsets()

```
void Experimental::PulsePerSecondData::getSignalOffsets (
            std::function< double *(size_t, size_t)> array_out )
```

For each reference contained in the `PulsePerSecondData` object a list of offsets for each signal channel is given, in the channel order given by `signal_channels`.

#### 9.53.3.4 getStatus()

```
void Experimental::PulsePerSecondData::getStatus (
            std::function< bool *(size_t)> array_out )
```

A vector of booleans values describing whether all signals, including from the reference source, were detected.

#### 9.53.3.5 getUtcDates()

```
std::vector< std::string > Experimental::PulsePerSecondData::getUtcDates ( )
```

The UTC timestamps for the system time at which each reference pulse is processed, as a string with ISO 8601 formatting.

#### 9.53.3.6 getUtcSeconds()

```
void Experimental::PulsePerSecondData::getUtcSeconds (
            std::function< double *(size_t)> array_out )
```

The number of elapsed seconds from the beginning of the Unix epoch (1st of January 1970) to the time at which each reference pulse is processed, as a floating point number.

### 9.53.4 Member Data Documentation

#### 9.53.4.1 size

```
const size_t Experimental::PulsePerSecondData::size
```

Number of reference pulses contained in the `PulsePerSecondData` object.

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

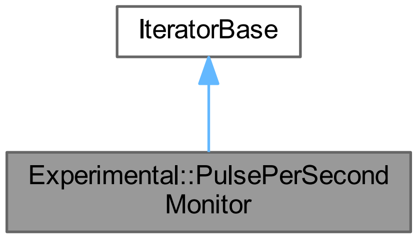
- Iterators.h

## 9.54 Experimental::PulsePerSecondMonitor Class Reference

Monitors the synchronicity of 1 pulse per second (PPS) signals.

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

Inheritance diagram for Experimental::PulsePerSecondMonitor:




### Public Member Functions

- PulsePerSecondMonitor (TimeTaggerBase *tagger, channel_t reference_channel, std::vector< channel_t > signal_channels, std::string filename="", timestamp_t period=1E12)

    *constructor of a PulsePerSecondMonitor measurement*
- ∼PulsePerSecondMonitor ()
- PulsePerSecondData getDataObject (bool remove=false)

    *Fetches the results of all measured PPS pulses.*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.54.1 Detailed Description

Monitors the synchronicity of 1 pulse per second (PPS) signals.

This measurement allows the user to monitor the synchronicity of different sources of 1 pulse per second (PPS) signals with respect to a reference source. For each signal from the reference PPS source, comparative offsets are calculated for the other signal channels. Upon processing, a UTC timestamp from the system time is associated with each reference pulse.

The monitoring starts on the first signal from the reference source and will run uninterrupted until the measurement is stopped. If a signal from a channel is not detected within one and a half periods, its respective offset will not be calculated but the measurement will continue nonetheless.

By specifying an output file name, the monitoring data can be continuously written to a comma-separated value file (.csv).

### 9.54.2 Constructor & Destructor Documentation

#### 9.54.2.1 PulsePerSecondMonitor()

```
Experimental::PulsePerSecondMonitor::PulsePerSecondMonitor (
            TimeTaggerBase * tagger,
            channel_t reference_channel,
            std::vector< channel_t > signal_channels,
            std::string filename = "",
            timestamp_t period = 1E12 )
```

constructor of a PulsePerSecondMonitor measurement

**Parameters**

| | |
|---|---|
| *tagger* | a TimeTagger object. |
| *reference_channel* | the channel whose signal will be the standard against which other signals are compared. |
| *signal_channels* | a list of channel numbers with PPS signals to be compared to the reference. |
| *filename* | the name of the .csv file to store measurement data. By default, no data is written to file. |
| *period* | the assumed period of the reference source, typically one second, in picoseconds. |

#### 9.54.2.2 ∼PulsePerSecondMonitor()

```
Experimental::PulsePerSecondMonitor::~PulsePerSecondMonitor ( )
```

### 9.54.3 Member Function Documentation

#### 9.54.3.1 clear_impl()

```
void Experimental::PulsePerSecondMonitor::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.

#### 9.54.3.2 getDataObject()

```
PulsePerSecondData Experimental::PulsePerSecondMonitor::getDataObject (
            bool remove = false )
```

Fetches the results of all measured PPS pulses.

**Returns**

a `PulsePerSecondData` object, which contains all data of the monitored PPS pulses. To remove the data from the internal memory after each call, set `remove` to `true`.

#### 9.54.3.3 next_impl()

```
bool Experimental::PulsePerSecondMonitor::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.

#### 9.54.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

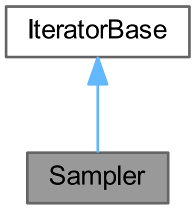
- Iterators.h

## 9.55 Sampler Class Reference

a triggered sampling measurement

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

Inheritance diagram for Sampler:




### Public Member Functions

- Sampler (TimeTaggerBase *tagger, channel_t trigger, std::vector< channel_t > channels, size_t max_triggers)

    *constructor of a Sampler measurement*
- ~Sampler ()
- void getData (std::function< timestamp_t *(size_t, size_t)> array_out)

    *fetches the internal data as 2D array.*
- void getDataAsMask (std::function< timestamp_t *(size_t, size_t)> array_out)

    *fetches the internal data as 2D array with a channel mask.*

### Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*

- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

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

## 9.55.1 Detailed Description

a triggered sampling measurement

This measurement class will perform a triggered sampling measurement. So for every event on the trigger input, the current state (low : 0, high : 1, unknown : 2) will be written to an internal buffer. Fetching the data of the internal buffer will clear its internal state without any deadtime. So every event will recorded exactly once.

The unknown state might happen after an overflow without an event on the input channel. This processing assumes that no event was filtered by the deadtime. Else invalid data will be reported till the next event on this input channel.

## 9.55.2 Constructor & Destructor Documentation

### 9.55.2.1 Sampler()

```
Sampler::Sampler (
            TimeTaggerBase * tagger,
            channel_t trigger,
            std::vector< channel_t > channels,
            size_t max_triggers )
```

constructor of a Sampler measurement

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *trigger* | the channel which shall trigger the measurement |
| *channels* | a list of channels which will be recorded for every trigger |
| *max_triggers* | the maximum amount of triggers without getData* call till this measurement will stop itself |

#### 9.55.2.2 ∼Sampler()

```
Sampler::∼Sampler ( )
```

### 9.55.3 Member Function Documentation

#### 9.55.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.55.3.2 getData()

```
void Sampler::getData (
            std::function< timestamp_t *(size_t, size_t)> array_out )
```

fetches the internal data as 2D array.

Its layout is roughly: [ [timestamp of first trigger, state of channel 0, state of channel 1, ...], [timestamp of second trigger, state of channel 0, state of channel 1, ...], ... ] Where state means: 0 – low 1 – high 2 – undefined (after overflow)

#### 9.55.3.3 getDataAsMask()

```
void Sampler::getDataAsMask (
            std::function< timestamp_t *(size_t, size_t)> array_out )
```

fetches the internal data as 2D array with a channel mask.

Its layout is roughly: [ [timestamp of first trigger, (state of channel 0) << 0 | (state of channel 1) << 1 | ... | undefined << 63], [timestamp of second trigger, (state of channel 0) << 0 | (state of channel 1) << 1 | ... | undefined << 63], ... ] Where state means: 0 – low or undefined (after overflow) 1 – high

#### 9.55.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.55.3.5 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

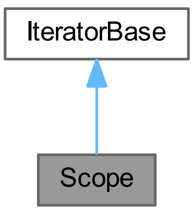
- Iterators.h

## 9.56 Scope Class Reference

a scope measurement

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

Inheritance diagram for Scope:



### Public Member Functions

- Scope (TimeTaggerBase *tagger, std::vector< channel_t > event_channels, channel_t trigger_channel, timestamp_t window_size=1000000000, int32_t n_traces=1, int32_t n_max_events=1000)

  *constructor of a Scope measurement*
- ~Scope ()
- bool ready ()
- int32_t triggered ()
- std::vector< std::vector< Event > > getData ()
- timestamp_t getWindowSize ()

## Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

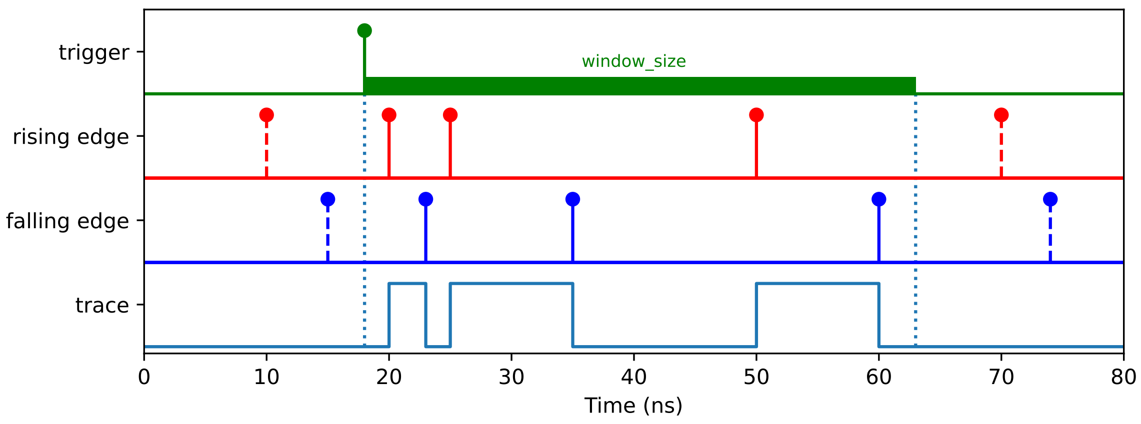
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.56.1 Detailed Description

a scope measurement




The Scope class allows to visualize time tags for rising and falling edges in a time trace diagram similarly to an ultrafast logic analyzer. The trace recording is synchronized to a trigger signal which can be any physical or virtual channel. However, only physical channels can be specified to the event_channels parameter. Additionally, one has to specify the time window_size which is the timetrace duration to be recorded, the number of traces to be recorded and the maximum number of events to be detected. If n_traces $<$ 1 then retriggering will occur infinitely, which is similar to the “normal” mode of an oscilloscope.

### 9.56.2 Constructor & Destructor Documentation

#### 9.56.2.1 Scope()

```
Scope::Scope (
            TimeTaggerBase * tagger,
            std::vector< channel_t > event_channels,
            channel_t trigger_channel,
            timestamp_t window_size = 1000000000,
            int32_t n_traces = 1,
            int32_t n_max_events = 1000 )
```

constructor of a Scope measurement

**Parameters**

| | |
|---|---|
| *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 |

#### 9.56.2.2 ∼Scope()

```
Scope::∼Scope ( )
```

### 9.56.3 Member Function Documentation

#### 9.56.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.56.3.2 getData()

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

#### 9.56.3.3 getWindowSize()

```
timestamp_t Scope::getWindowSize ( )
```

#### 9.56.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.56.3.5 ready()

```
bool Scope::ready ( )
```

#### 9.56.3.6 triggered()

```
int32_t Scope::triggered ( )
```

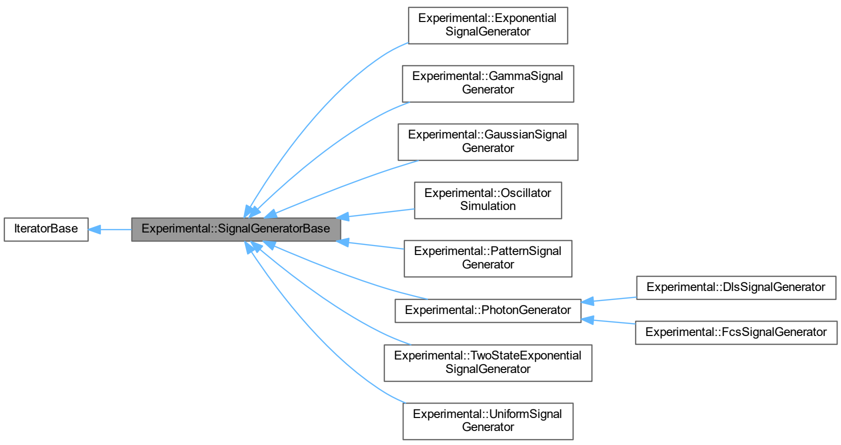
The documentation for this class was generated from the following file:

- Iterators.h

## 9.57 Experimental::SignalGeneratorBase Class Reference

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

Inheritance diagram for Experimental::SignalGeneratorBase:




### Public Member Functions

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

    *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- virtual void initialize (timestamp_t initial_time)=0
- virtual timestamp_t get_next ()=0
- virtual void on_restart (timestamp_t restart_time)
- bool next_impl (std::vector< Tag > &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

*clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Protected Attributes

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.57.1 Constructor & Destructor Documentation

### 9.57.1.1 SignalGeneratorBase()

```
Experimental::SignalGeneratorBase::SignalGeneratorBase (
            TimeTaggerBase * tagger,
            channel_t base_channel = CHANNEL_UNUSED )
```

### 9.57.1.2 ~SignalGeneratorBase()

```
Experimental::SignalGeneratorBase::~SignalGeneratorBase ( )
```

## 9.57.2 Member Function Documentation

### 9.57.2.1 get_next()

```
virtual timestamp_t Experimental::SignalGeneratorBase::get_next ( ) [protected], [pure virtual]
```

Implemented in Experimental::PhotonGenerator, Experimental::UniformSignalGenerator, Experimental::GaussianSignalGenerator, Experimental::OscillatorSimulation, Experimental::TwoStateExponentialSignalGenerator, Experimental::ExponentialSignalGenerator, Experimental::GammaSignalGenerator, and Experimental::PatternSignalGenerator.

### 9.57.2.2 getChannel()

```
channel_t Experimental::SignalGeneratorBase::getChannel ( )
```

the new virtual channel

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

### 9.57.2.3 initialize()

```
virtual void Experimental::SignalGeneratorBase::initialize (
            timestamp_t initial_time ) [protected], [pure virtual]
```

Implemented in Experimental::PhotonGenerator, Experimental::UniformSignalGenerator, Experimental::GaussianSignalGenerator, Experimental::OscillatorSimulation, Experimental::TwoStateExponentialSignalGenerator, Experimental::ExponentialSignalGenerator, Experimental::GammaSignalGenerator, and Experimental::PatternSignalGenerator.

### 9.57.2.4 isProcessingFinished()

```
bool Experimental::SignalGeneratorBase::isProcessingFinished ( ) [protected]
```

### 9.57.2.5 next_impl()

```
bool Experimental::SignalGeneratorBase::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.

#### 9.57.2.6 on_restart()

```
virtual void Experimental::SignalGeneratorBase::on_restart (
            timestamp_t restart_time )  [protected], [virtual]
```

Reimplemented in Experimental::PhotonGenerator, Experimental::UniformSignalGenerator, Experimental::GaussianSignalGenerator, Experimental::OscillatorSimulation, Experimental::TwoStateExponentialSignalGenerator, Experimental::ExponentialSignalGenerator, Experimental::GammaSignalGenerator, and Experimental::PatternSignalGenerator.

#### 9.57.2.7 on_stop()

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

callback when the measurement class is stopped

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.57.2.8 set_processing_finished()

```
void Experimental::SignalGeneratorBase::set_processing_finished (
            bool is_finished )  [protected]
```

### 9.57.3 Member Data Documentation

#### 9.57.3.1 impl

```
std::unique_ptr<SignalGeneratorBaseImpl> Experimental::SignalGeneratorBase::impl  [protected]
```

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

- Iterators.h

## 9.58 Experimental::SimDetector Class Reference

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

**Public Member Functions**

- SimDetector (TimeTaggerBase *tagger, channel_t input_channel, double efficiency=1.0, double darkcount↩
_rate=0.0, double jitter=0, double deadtime=0.0, int32_t seed=-1)

    *Construct a simulation of a physical detector for a given channel/signal.*
- ∼SimDetector ()
- channel_t getChannel ()

## 9.58.1 Constructor & Destructor Documentation

### 9.58.1.1 SimDetector()

```
Experimental::SimDetector::SimDetector (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double efficiency = 1.0,
            double darkcount_rate = 0.0,
            double jitter = 0,
            double deadtime = 0.0,
            int32_t seed = -1 )
```

Construct a simulation of a physical detector for a given channel/signal.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel with the signal passing through the virtual detector |
| *efficiency* | rate of acceptance for inputs. |
| *darkcount_rate* | rate of noise in Herz. |
| *jitter* | standard deviation of the gaussian broadening, in seconds. |
| *deadtime* | deadtime, in seconds. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.58.1.2 ∼SimDetector()

```
Experimental::SimDetector::~SimDetector ( )
```

## 9.58.2 Member Function Documentation

### 9.58.2.1 getChannel()

```
channel_t Experimental::SimDetector::getChannel ( )
```

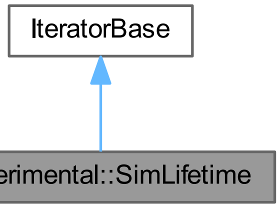
The documentation for this class was generated from the following file:

- Iterators.h

## 9.59 Experimental::SimLifetime Class Reference

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

Inheritance diagram for Experimental::SimLifetime:




### Public Member Functions

- SimLifetime (TimeTaggerBase *tagger, channel_t input_channel, double lifetime, double emission_rate=0.1, int32_t seed=-1)

    *Construct a simulation of a physical exaltation.*
- ~SimLifetime ()
- channel_t getChannel ()
- void registerLifetimeReactor (channel_t trigger_channel, std::vector< double > lifetimes, bool repeat)
- void registerEmissionReactor (channel_t trigger_channel, std::vector< double > emissions, bool repeat)

### Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

    *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

    *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

    *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

    *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

    *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

    *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

    *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

    *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

    *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

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

  *update iterator state*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.59.1 Constructor & Destructor Documentation

### 9.59.1.1 SimLifetime()

```
Experimental::SimLifetime::SimLifetime (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double lifetime,
            double emission_rate = 0.1,
            int32_t seed = -1 )
```

Construct a simulation of a physical exaltation.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel which triggers the exaltation. |
| *lifetime* | lifetime of the exaltation. |
| *emission_rate* | poissonian emission rate for each input event. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.59.1.2 ∼SimLifetime()

```
Experimental::SimLifetime::~SimLifetime ( )
```

## 9.59.2 Member Function Documentation

### 9.59.2.1 getChannel()

```
channel_t Experimental::SimLifetime::getChannel ( )
```

### 9.59.2.2 next_impl()

```
bool Experimental::SimLifetime::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.

#### 9.59.2.3 registerEmissionReactor()

```
void Experimental::SimLifetime::registerEmissionReactor (
            channel_t trigger_channel,
            std::vector< double > emissions,
            bool repeat )
```

#### 9.59.2.4 registerLifetimeReactor()

```
void Experimental::SimLifetime::registerLifetimeReactor (
            channel_t trigger_channel,
            std::vector< double > lifetimes,
            bool repeat )
```

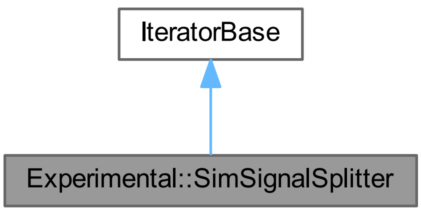
The documentation for this class was generated from the following file:

- Iterators.h

## 9.60 Experimental::SimSignalSplitter Class Reference

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

Inheritance diagram for Experimental::SimSignalSplitter:




**Public Member Functions**

- SimSignalSplitter (TimeTaggerBase *tagger, channel_t input_channel, double ratio=0.5, int32_t seed=-1)
  *Construct a signal splitter which will split events from an input channel into a left and a right virtual channels.*
- ∼SimSignalSplitter ()
- std::vector< channel_t > getChannels ()
- channel_t getLeftChannel ()
- channel_t getRightChannel ()

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

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

  *update iterator state*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.60.1 Constructor & Destructor Documentation

### 9.60.1.1 SimSignalSplitter()

```
Experimental::SimSignalSplitter::SimSignalSplitter (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double ratio = 0.5,
            int32_t seed = -1 )
```

Construct a signal splitter which will split events from an input channel into a left and a right virtual channels.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel to be split. |
| *ratio* | bias towards right or left channel. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

#### 9.60.1.2 ∼SimSignalSplitter()

```
Experimental::SimSignalSplitter::~SimSignalSplitter ( )
```

### 9.60.2 Member Function Documentation

#### 9.60.2.1 getChannels()

```
std::vector< channel_t > Experimental::SimSignalSplitter::getChannels ( )
```

#### 9.60.2.2 getLeftChannel()

```
channel_t Experimental::SimSignalSplitter::getLeftChannel ( )
```

#### 9.60.2.3 getRightChannel()

```
channel_t Experimental::SimSignalSplitter::getRightChannel ( )
```

#### 9.60.2.4 next_impl()

```
bool Experimental::SimSignalSplitter::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.

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

- Iterators.h

# 9.61 SoftwareClockState Struct Reference

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

### Public Attributes

- timestamp_t clock_period
- channel_t input_channel
- channel_t ideal_clock_channel
- double averaging_periods
- bool enabled
- bool is_locked
- uint32_t error_counter
- timestamp_t last_ideal_clock_event
- double period_error
- double phase_error_estimation

## 9.61.1 Member Data Documentation

### 9.61.1.1 averaging_periods

```
double SoftwareClockState::averaging_periods
```

### 9.61.1.2 clock_period

```
timestamp_t SoftwareClockState::clock_period
```

### 9.61.1.3 enabled

```
bool SoftwareClockState::enabled
```

### 9.61.1.4 error_counter

```
uint32_t SoftwareClockState::error_counter
```

### 9.61.1.5 ideal_clock_channel

```
channel_t SoftwareClockState::ideal_clock_channel
```

### 9.61.1.6 input_channel

```
channel_t SoftwareClockState::input_channel
```

#### 9.61.1.7 is_locked

```
bool SoftwareClockState::is_locked
```

#### 9.61.1.8 last_ideal_clock_event

```
timestamp_t SoftwareClockState::last_ideal_clock_event
```

#### 9.61.1.9 period_error

```
double SoftwareClockState::period_error
```

#### 9.61.1.10 phase_error_estimation

```
double SoftwareClockState::phase_error_estimation
```

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

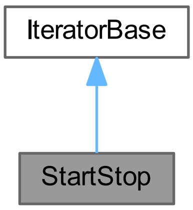
- TimeTagger.h

## 9.62 StartStop Class Reference

simple start-stop measurement

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

Inheritance diagram for StartStop:




### Public Member Functions

- StartStop (TimeTaggerBase ∗tagger, channel_t click_channel, channel_t start_channel=CHANNEL_UNUSED, timestamp_t binwidth=1000)

    *constructor of StartStop*
- ∼StartStop ()
- void getData (std::function< timestamp_t ∗(size_t, size_t)> array_out)

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

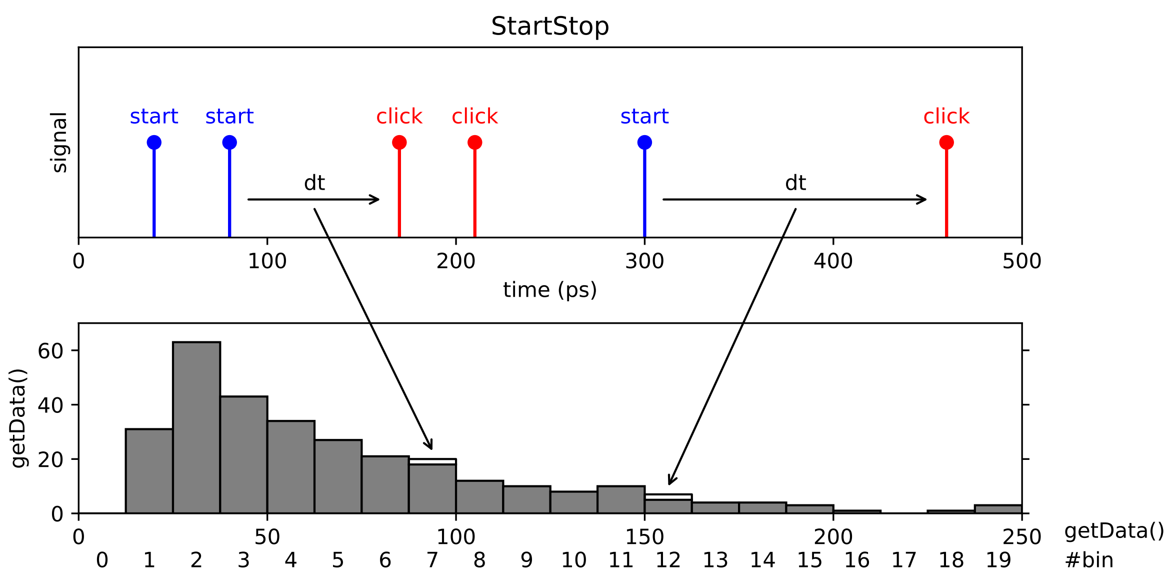
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.62.1 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.

## 9.62.2 Constructor & Destructor Documentation

### 9.62.2.1 StartStop()

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

constructor of StartStop

**Parameters**

| | |
|---|---|
| *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 |

### 9.62.2.2 ∼StartStop()

```
StartStop::∼StartStop ( )
```

## 9.62.3 Member Function Documentation

### 9.62.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

### 9.62.3.2 getData()

```
void StartStop::getData (
            std::function< timestamp_t *(size_t, size_t)> array_out )
```

#### 9.62.3.3 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.62.3.4 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

- Iterators.h

## 9.63 SynchronizedMeasurements Class Reference

start, stop and clear several measurements synchronized

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

#### Public Member Functions

- SynchronizedMeasurements (TimeTaggerBase ∗tagger)

    *construct a SynchronizedMeasurements object*
- ∼SynchronizedMeasurements ()
- void registerMeasurement (IteratorBase ∗measurement)

    *register a measurement (iterator) to the SynchronizedMeasurements-group.*
- void unregisterMeasurement (IteratorBase ∗measurement)

    *unregister a measurement (iterator) from the SynchronizedMeasurements-group.*
- void clear ()

    *clear all registered measurements synchronously*
- void start ()

    *start all registered measurements synchronously*
- void stop ()

    *stop all registered measurements synchronously*
- void startFor (timestamp_t capture_duration, bool clear=true)

    *start all registered measurements synchronously, and stops them after the capture_duration*
- bool waitUntilFinished (int64_t timeout=-1)

    *wait until all registered measurements have finished running.*
- bool isRunning ()

    *check if any iterator is running*
- TimeTaggerBase ∗ getTagger ()

    *Returns a proxy tagger object, which shall be used to create immediately registered measurements.*

#### Protected Member Functions

- void runCallback (TimeTaggerBase::IteratorCallback callback, std::unique_lock< std::mutex > &lk, bool block=true)

    *run a callback on all registered measurements synchronously*

## 9.63.1 Detailed Description

start, stop and clear several measurements synchronized

For the case that several measurements should be started, stopped or cleared at the very same time, a SynchronizedMeasurements object can be create to which all the measurements (also called iterators) can be registered with .registerMeasurement(measurement). Calling .stop(), .start() or .clear() on the SynchronizedMeasurements object will call the respective method on each of the registered measurements at the very same time. That means that all measurements taking part will have processed the very same time tags.

## 9.63.2 Constructor & Destructor Documentation

### 9.63.2.1 SynchronizedMeasurements()

```
SynchronizedMeasurements::SynchronizedMeasurements (
            TimeTaggerBase * tagger )
```

construct a SynchronizedMeasurements object

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |

#### 9.63.2.2 ∼SynchronizedMeasurements()

```
SynchronizedMeasurements::~SynchronizedMeasurements ( )
```

### 9.63.3 Member Function Documentation

#### 9.63.3.1 clear()

```
void SynchronizedMeasurements::clear ( )
```

clear all registered measurements synchronously

#### 9.63.3.2 getTagger()

```
TimeTaggerBase * SynchronizedMeasurements::getTagger ( )
```

Returns a proxy tagger object, which shall be used to create immediately registered measurements.

Those measurements will not start automatically.

#### 9.63.3.3 isRunning()

```
bool SynchronizedMeasurements::isRunning ( )
```

check if any iterator is running

#### 9.63.3.4 registerMeasurement()

```
void SynchronizedMeasurements::registerMeasurement (
            IteratorBase * measurement )
```

register a measurement (iterator) to the SynchronizedMeasurements-group.

All available methods called on the SynchronizedMeasurements will happen at the very same time for all the registered measurements.

#### 9.63.3.5 runCallback()

```
void SynchronizedMeasurements::runCallback (
            TimeTaggerBase::IteratorCallback callback,
            std::unique_lock< std::mutex > & lk,
            bool block = true ) [protected]
```

run a callback on all registered measurements synchronously

Please keep in mind that the callback is copied for each measurement. So please avoid big captures.

### 9.63.3.6 start()

```
void SynchronizedMeasurements::start ( )
```

start all registered measurements synchronously

### 9.63.3.7 startFor()

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

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

### 9.63.3.8 stop()

```
void SynchronizedMeasurements::stop ( )
```

stop all registered measurements synchronously

### 9.63.3.9 unregisterMeasurement()

```
void SynchronizedMeasurements::unregisterMeasurement (
            IteratorBase * measurement )
```

unregister a measurement (iterator) from the SynchronizedMeasurements-group.

Stops synchronizing calls on the selected measurement, if the measurement is not within this synchronized group, the method does nothing.

### 9.63.3.10 waitUntilFinished()

```
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 indefinitely, it will log an error and return immediately.

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

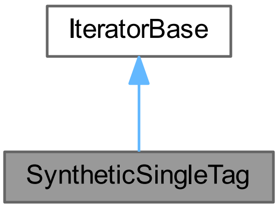
- Iterators.h

## 9.64 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

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

#### Protected Member Functions

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

    *update iterator state*

### Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

    *clear Iterator state.*
- virtual void on_start ()

    *callback when the measurement class is started*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

#### Additional Inherited Members

### Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

### 9.64.1 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.

### 9.64.2 Constructor & Destructor Documentation

#### 9.64.2.1 SyntheticSingleTag()

```
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. |

#### 9.64.2.2 ∼SyntheticSingleTag()

```
SyntheticSingleTag::~SyntheticSingleTag ( )
```

### 9.64.3 Member Function Documentation

#### 9.64.3.1 getChannel()

```
channel_t SyntheticSingleTag::getChannel ( ) const
```

#### 9.64.3.2 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.64.3.3 trigger()

```
void SyntheticSingleTag::trigger ( )
```

Generate a timetag for each call of this method.

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

- Iterators.h

## 9.65 Tag Struct Reference

a single event on a channel

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

### Public Types

- enum class Type : unsigned char {
  TimeTag = 0 , Error = 1 , OverflowBegin = 2 , OverflowEnd = 3 ,
  MissedEvents = 4 }

  *This enum marks what kind of event this object represents.*

### Public Member Functions

- Tag ()
- Tag (timestamp_t ts, channel_t ch, Type type=Type::TimeTag)
- Tag (Type type, char reserved, unsigned short missed_events, channel_t ch, timestamp_t ts)

**Public Attributes**

- enum Tag::Type TimeTag
- char reserved {}

  *8 bit padding*
- unsigned short missed_events {}

  *Amount of missed events in overflow mode.*
- channel_t channel {}

  *the channel number*
- timestamp_t time {}

  *the timestamp of the event in picoseconds*

## 9.65.1 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.

## 9.65.2 Member Enumeration Documentation

### 9.65.2.1 Type

```
enum class Tag::Type : unsigned char [strong]
```

This enum marks what kind of event this object represents.

- TimeTag: a normal event from any input channel
- Error: an error in the internal data processing, e.g. on plugging the external clock. This invalidates the global time
- OverflowBegin: this marks the begin of an interval with incomplete data because of too high data rates
- OverflowEnd: this marks the end of the interval. All events, which were lost in this interval, have been handled
- MissedEvents: this virtual event signals the amount of lost events per channel within an overflow interval. Repeated usage for higher amounts of events

**Enumerator**

| | |
|---|---|
| TimeTag | |
| Error | |
| OverflowBegin | |
| OverflowEnd | |
| MissedEvents | |

### 9.65.3 Constructor & Destructor Documentation

#### 9.65.3.1 Tag() [1/3]

```
Tag::Tag ( ) [inline]
```

#### 9.65.3.2 Tag() [2/3]

```
Tag::Tag (
        timestamp_t ts,
        channel_t ch,
        Type type = Type::TimeTag ) [inline]
```

#### 9.65.3.3 Tag() [3/3]

```
Tag::Tag (
        Type type,
        char reserved,
        unsigned short missed_events,
        channel_t ch,
        timestamp_t ts ) [inline]
```

### 9.65.4 Member Data Documentation

#### 9.65.4.1 channel

```
channel_t Tag::channel {}
```

the channel number

#### 9.65.4.2 missed_events

```
unsigned short Tag::missed_events {}
```

Amount of missed events in overflow mode.

Within overflow intervals, the timing of all events is skipped. However, the total amount of events is still recorded. For events with type = MissedEvents, this indicates that a given amount of tags for this channel have been skipped in the interval. Note: There might be many missed events tags per overflow interval and channel. The accumulated amount represents the total skipped events.

#### 9.65.4.3 reserved

```
char Tag::reserved {}
```

8 bit padding

Reserved for future use. Set it to zero.

#### 9.65.4.4 time

```
timestamp_t Tag::time {}
```

the timestamp of the event in picoseconds

#### 9.65.4.5 TimeTag

```
enum Tag::Type Tag::TimeTag
```

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

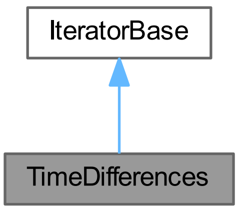
- TimeTagger.h

## 9.66 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< timestamp_t ∗(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)*
- int32_t getHistogramIndex () const
  *The index of the currently processed histogram or the waiting state.*
- bool ready ()
  *returns 'true' when the required number of rollovers set by 'setMaxCounts' has been reached*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

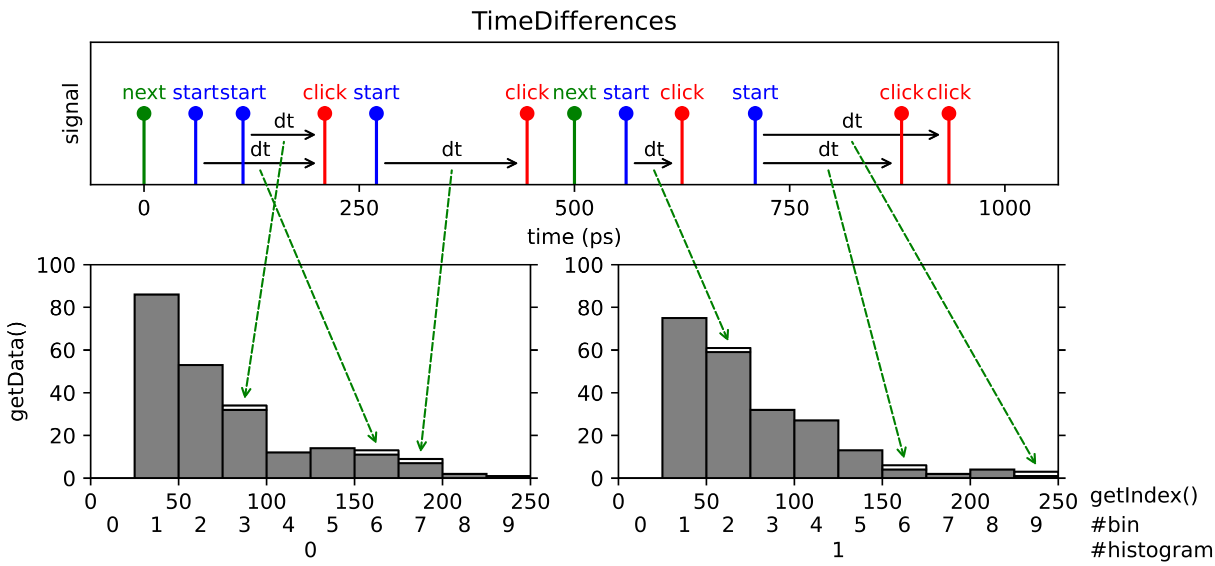
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.66.1 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.

## 9.66.2 Constructor & Destructor Documentation

### 9.66.2.1 TimeDifferences()

```
TimeDifferences::TimeDifferences (
            TimeTaggerBase * tagger,
            channel_t click_channel,
            channel_t start_channel = CHANNEL_UNUSED,
            channel_t next_channel = CHANNEL_UNUSED,
            channel_t sync_channel = CHANNEL_UNUSED,
            timestamp_t binwidth = 1000,
            int32_t n_bins = 1000,
            int32_t n_histograms = 1 )
```

constructor of a TimeDifferences measurement

**Parameters**

| | |
|---|---|
| *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 |

#### 9.66.2.2 ∼TimeDifferences()

```
TimeDifferences::~TimeDifferences ( )
```

### 9.66.3 Member Function Documentation

#### 9.66.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.66.3.2 getCounts()

```
uint64_t TimeDifferences::getCounts ( )
```

returns the number of rollovers (histogram index resets)

#### 9.66.3.3 getData()

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

returns a two-dimensional array of size ‘n_bins’ by ‘n_histograms’ containing the histograms

#### 9.66.3.4 getHistogramIndex()

```
int32_t TimeDifferences::getHistogramIndex ( ) const
```

The index of the currently processed histogram or the waiting state.

Possible return values are: `-2`: Waiting for an event on `sync_channel` (only if `sync_channel` is defined) `-1`: Waiting for an event on `next_channel` (only if `sync_channel` is defined) `0` ... `(n_histograms - 1)`: Index of the currently processed histogram

#### 9.66.3.5 getIndex()

```
void TimeDifferences::getIndex (
            std::function< timestamp_t *(size_t)> array_out )
```

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

#### 9.66.3.6 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.66.3.7 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

#### 9.66.3.8 ready()

```
bool TimeDifferences::ready ( )
```

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

#### 9.66.3.9 setMaxCounts()

```
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 |

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

- Iterators.h

## 9.67 TimeDifferencesImpl< T > Class Template Reference

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

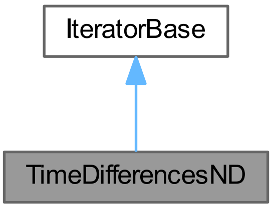
- Iterators.h

## 9.68 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< timestamp_t ∗(size_t)> array_out)

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

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

*Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase ∗ tagger

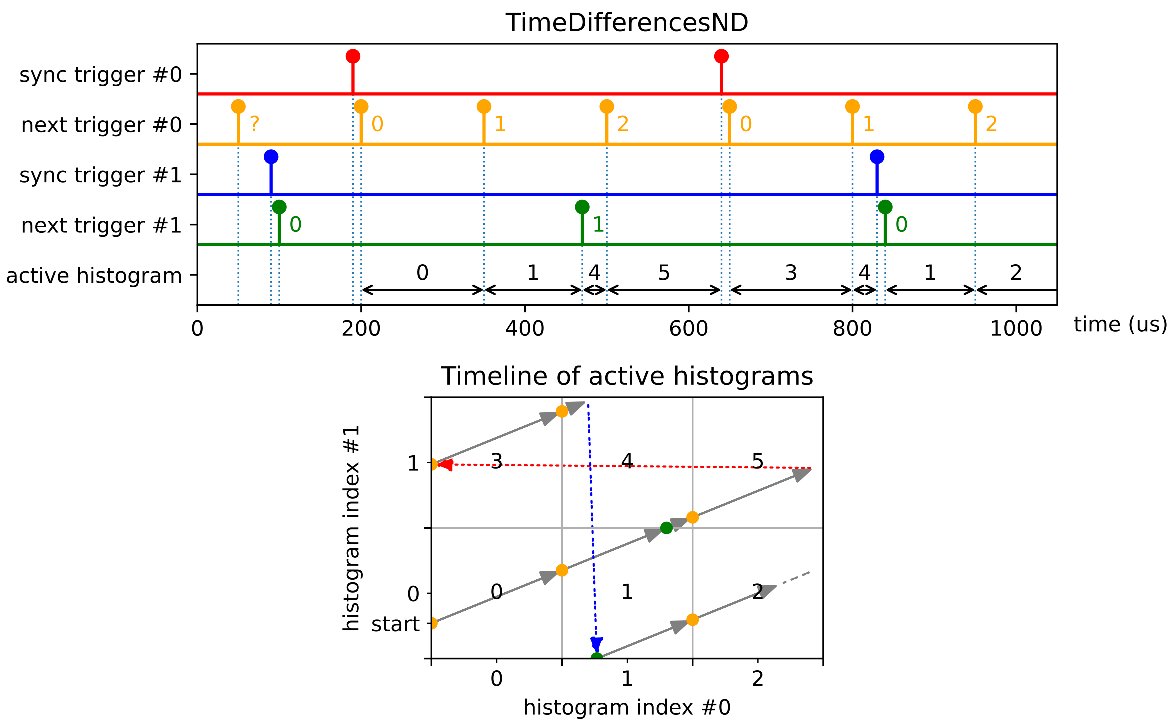
  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

### 9.68.1 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

## 9.68.2 Constructor & Destructor Documentation

### 9.68.2.1 TimeDifferencesND()

```
TimeDifferencesND::TimeDifferencesND (
            TimeTaggerBase * tagger,
            channel_t click_channel,
            channel_t start_channel,
            std::vector< channel_t > next_channels,
            std::vector< channel_t > sync_channels,
            std::vector< int32_t > n_histograms,
            timestamp_t binwidth,
            int32_t n_bins )
```

constructor of a TimeDifferencesND measurement

**Parameters**

| | |
|---|---|
| *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 |

### 9.68.2.2 ∼TimeDifferencesND()

```
TimeDifferencesND::∼TimeDifferencesND ( )
```

## 9.68.3 Member Function Documentation

### 9.68.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.68.3.2 getData()

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

returns a two-dimensional array of size n_bins by all n_histograms containing the histograms

#### 9.68.3.3 getIndex()

```
void TimeDifferencesND::getIndex (
            std::function< timestamp_t *(size_t)> array_out )
```

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

#### 9.68.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.68.3.5 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

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

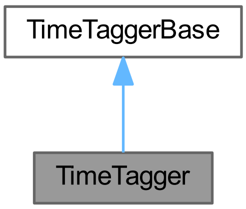
- Iterators.h

## 9.69 TimeTagger Class Reference

backend for the TimeTagger.

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

Inheritance diagram for TimeTagger:




### Public Member Functions

- virtual void reset ()=0

    *reset the TimeTagger object to default settings and detach all iterators*
- virtual bool isChannelRegistered (channel_t chan)=0
- virtual void setTestSignalDivider (int divider)=0

    *set the divider for the frequency of the test signal*
- virtual int getTestSignalDivider ()=0

    *get the divider for the frequency of the test signal*
- virtual void xtra_setAuxOutSignal (int channel, int divider, double duty_cycle=0.5)=0

    *set the divider for the frequency of the aux out signal generator and enable aux out*
- virtual int xtra_getAuxOutSignalDivider (int channel)=0

    *get the divider for the frequency of the aux out signal generator*
- virtual double xtra_getAuxOutSignalDutyCycle (int channel)=0

    *get the dutycycle of the aux out signal generator*
- virtual void xtra_setAuxOut (int channel, bool enabled)=0

    *enable or disable aux out*
- virtual bool xtra_getAuxOut (int channel)=0

    *fetch the status of the aux out signal generator*

- virtual void xtra_setFanSpeed (double percentage=-1)=0

  *configures the FAN speed on TTU HW >= 1.3*
- 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 double xtra_measureTriggerLevel (channel_t channel)=0

  *measures the electrically applied 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 > channels, bool state)=0

  *enables or disables the normalization of the distribution.*
- virtual bool getNormalization (channel_t channel)=0

  *returns the the normalization of the distribution.*
- virtual void setHardwareBufferSize (int size)=0

  *sets the maximum USB buffer size*
- virtual int getHardwareBufferSize ()=0

  *queries the size of the USB queue*
- virtual void setStreamBlockSize (int max_events, int max_latency)=0

  *sets the maximum events and latency for the stream block size*
- virtual int getStreamBlockSizeEvents ()=0
- virtual int getStreamBlockSizeLatency ()=0
- virtual void setEventDivider (channel_t channel, unsigned int divider)=0

  *Divides the amount of transmitted edge per channel.*
- virtual unsigned int getEventDivider (channel_t channel)=0

  *Returns the factor of the dividing filter.*
- virtual void autoCalibration (std::function< double *(size_t)> array_out)=0

  *runs a calibrations based on the on-chip uncorrelated signal generator.*
- virtual std::string getSerial ()=0

  *identifies the hardware by serial number*
- virtual std::string getModel ()=0

  *identifies the hardware by Time Tagger Model*
- virtual int getChannelNumberScheme ()=0

  *Fetch the configured numbering scheme for this TimeTagger object.*
- virtual std::vector< double > getDACRange ()=0

  *returns the minimum and the maximum voltage of the DACs as a trigger reference*
- virtual void getDistributionCount (std::function< uint64_t *(size_t, size_t)> array_out)=0

  *get internal calibration data*

- virtual void getDistributionPSecs (std::function< double ∗(size_t, size_t)> array_out)=0

  *get internal calibration data*
- virtual std::vector< channel_t > getChannelList (ChannelEdge type=ChannelEdge::All)=0

  *fetch a vector of all physical input channel ids*
- virtual timestamp_t getPsPerClock ()=0

  *fetch the duration of each clock cycle in picoseconds*
- virtual std::string getPcbVersion ()=0

  *Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.*
- virtual std::string getFirmwareVersion ()=0

  *Return an unique identifier for the applied firmware.*
- virtual void xtra_setClockSource (int source)=0

  *manually overwrite the reference clock source*
- virtual int xtra_getClockSource ()=0

  *fetch the overwritten reference clock source*
- virtual void xtra_setClockAutoSelect (bool enabled)=0

  *activates auto clocking function*
- virtual bool xtra_getClockAutoSelect ()=0

  *queries if the auto clocking function is enabled*
- virtual void xtra_setClockOut (bool enabled)=0

  *enables the clock output*
- 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 void disableLEDs (bool disabled)=0

  *disables the LEDs on the TT*
- virtual std::string getDeviceLicense ()=0

  *gets the license, installed on this device currently*
- virtual uint32_t factoryAccess (uint32_t pw, uint32_t addr, uint32_t data, uint32_t mask, bool use_wb=false)=0

  *Direct read/write access to WireIn/WireOuts in FPGA (mask==0 for readonly)*
- virtual void setSoundFrequency (uint32_t freq_hz)=0

  *Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)*
- virtual void enableFpgaLink (std::vector< channel_t > channels, std::string destination_mac, FpgaLinkInterface link_interface=FpgaLinkInterface::SFPP_10GE, bool exclusive=false)=0

  *Enable the FPGA link of the Time Tagger X.*
- virtual void disableFpgaLink ()=0

  *Disable the FPGA link of the Time Tagger X.*
- virtual void startServer (AccessMode access_mode, std::vector< channel_t > channels=std::vector< channel_t >(), uint32_t port=41101)=0

  *starts the Time Tagger server that will stream the time tags to the client.*
- virtual bool isServerRunning ()=0

  *check if the server is still running.*
- virtual void stopServer ()=0

  *stops the time tagger server if currently running, otherwise does nothing.*
- virtual void setTimeTaggerNetworkStreamCompression (bool active)=0

  *enable or disable additional compression of the timetag stream as ent over the network.*
- virtual void setInputImpedanceHigh (channel_t channel, bool high_impedance)=0

  *enable high impedance termination mode*

- virtual bool getInputImpedanceHigh (channel_t channel)=0

  *query the state of the high impedance termination mode*
- virtual void setInputHysteresis (channel_t channel, int value)=0

  *configure the hysteresis voltage of the input comparator*
- virtual int getInputHysteresis (channel_t channel)=0

  *query the hysteresis voltage of the input comparator*
- virtual void xtra_setAvgRisingFalling (channel_t channel, bool enable)=0

  *configures if the rising and falling events shall be averaged*
- virtual bool xtra_getAvgRisingFalling (channel_t channel)=0

  *query if the rising and falling events shall be averaged*
- virtual void xtra_setHighPrioChannel (channel_t channel, bool enable)=0

  *configures if this channel shall exit overflow regions.*
- virtual bool xtra_getHighPrioChannel (channel_t channel)=0

  *if this channel shall exit overflow regions*
- virtual void updateBMCFirmware (const std::string &firmware)=0

  *updates the firmware of the Time Tagger X board management controller*

## Public Member Functions inherited from TimeTaggerBase

- virtual unsigned int getFence (bool alloc_fence=true)=0

  *Generate a new fence object, which validates the current configuration and the current time.*
- virtual bool waitForFence (unsigned int fence, int64_t timeout=-1)=0

  *Wait for a fence in the data stream.*
- virtual bool sync (int64_t timeout=-1)=0

  *Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.*
- virtual channel_t getInvertedChannel (channel_t channel)=0

  *get the falling channel id for a rising channel and vice versa*
- virtual bool isUnusedChannel (channel_t channel)=0

  *compares the provided channel with CHANNEL_UNUSED*
- virtual void runSynchronized (const IteratorCallbackMap &callbacks, bool block=true)=0

  *Run synchronized callbacks for a list of iterators.*
- virtual std::string getConfiguration ()=0

  *Fetches the overall configuration status of the Time Tagger object.*
- virtual void setInputDelay (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelayHardware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelaySoftware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual timestamp_t getInputDelay (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t getDelaySoftware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t getDelayHardware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t setDeadtime (channel_t channel, timestamp_t deadtime)=0

  *set the deadtime between two edges on the same channel.*
- virtual timestamp_t getDeadtime (channel_t channel)=0

  *get the deadtime between two edges on the same channel.*
- virtual void setTestSignal (channel_t channel, bool enabled)=0

*enable/disable internal test signal on a channel.*
- virtual void setTestSignal (std::vector< channel_t > channel, bool enabled)=0

  *enable/disable internal test signal on multiple channels.*
- virtual bool getTestSignal (channel_t channel)=0

  *fetch the status of the test signal generator*
- virtual void setSoftwareClock (channel_t input_channel, double input_frequency=10e6, double averaging_↩ periods=1000, bool wait_until_locked=true)=0

  *enables a software PLL to lock the time to an external clock*
- virtual void disableSoftwareClock ()=0

  *disabled the software PLL*
- virtual SoftwareClockState getSoftwareClockState ()=0

  *queries all state information of the software clock*
- virtual long long getOverflows ()=0

  *get overflow count*
- virtual void clearOverflows ()=0

  *clear overflow counter*
- virtual long long getOverflowsAndClear ()=0

  *get and clear overflow counter*

**Additional Inherited Members**

## Public Types inherited from TimeTaggerBase

- typedef std::function< void(IteratorBase ∗) IteratorCallback)
- typedef std::map< IteratorBase ∗, IteratorCallback > IteratorCallbackMap

## Protected Member Functions inherited from TimeTaggerBase

- TimeTaggerBase ()

  *abstract interface class*
- virtual ∼TimeTaggerBase ()

  *destructor*
- TimeTaggerBase (const TimeTaggerBase &)=delete
- TimeTaggerBase & operator= (const TimeTaggerBase &)=delete
- virtual std::shared_ptr< IteratorBaseListNode > addIterator (IteratorBase ∗it)=0
- virtual void freeIterator (IteratorBase ∗it)=0
- virtual channel_t getNewVirtualChannel ()=0
- virtual void freeVirtualChannel (channel_t channel)=0
- virtual void registerChannel (channel_t channel)=0

  *register a FPGA channel.*
- virtual void registerChannel (std::set< channel_t > channels)=0
- virtual void unregisterChannel (channel_t channel)=0

  *release a previously registered channel.*
- virtual void unregisterChannel (std::set< channel_t > channels)=0
- virtual void addChild (TimeTaggerBase ∗child)=0
- virtual void removeChild (TimeTaggerBase ∗child)=0
- virtual void release ()=0

### 9.69.1 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.

### 9.69.2 Member Function Documentation

#### 9.69.2.1 autoCalibration()

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

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

#### 9.69.2.2 clearConditionalFilter()

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

deactivates the conditional filter

equivalent to setConditionalFilter({},{})

#### 9.69.2.3 disableFpgaLink()

```
virtual void TimeTagger::disableFpgaLink ( )  [pure virtual]
```

Disable the FPGA link of the Time Tagger X.

#### 9.69.2.4 disableLEDs()

```
virtual void TimeTagger::disableLEDs (
            bool disabled )  [pure virtual]
```

disables the LEDs on the TT

Caution: This feature currently lacks support for disabling the power LED on the Time Tagger X.

**Parameters**

| | |
|---|---|
| *disabled* | true to disable all LEDs on the TT |

#### 9.69.2.5 enableFpgaLink()

```
virtual void TimeTagger::enableFpgaLink (
            std::vector< channel_t > channels,
```

```
            std::string destination_mac,
            FpgaLinkInterface link_interface = FpgaLinkInterface::SFPP_10GE,
            bool exclusive = false )  [pure virtual]
```

Enable the FPGA link of the Time Tagger X.

**Parameters**

| | |
|---|---|
| *channels* | list of channels, which shall be streamed over the FPGA link |
| *destination_mac* | Destination MAC, use an empty string for the broadcast address of "FF:FF:FF:FF:FF:FF" |
| *link_interface* | selects which interface shall be used, default is FpgaLinkInterface::SFPP_10GE |
| *exclusive* | determines if time tags should exclusively be transmitted over Ethernet, increasing Ethernet performance and avoiding USB issues, default is mixed USB & ethernet |

#### 9.69.2.6 factoryAccess()

```
virtual uint32_t TimeTagger::factoryAccess (
            uint32_t pw,
            uint32_t addr,
            uint32_t data,
            uint32_t mask,
            bool use_wb = false )  [pure virtual]
```

Direct read/write access to WireIn/WireOuts in FPGA (mask==0 for readonly)

DO NOT USE. Only for internal debug purposes.

#### 9.69.2.7 getChannelList()

```
virtual std::vector< channel_t > TimeTagger::getChannelList (
            ChannelEdge type = ChannelEdge::All )  [pure virtual]
```

fetch a vector of all physical input channel ids

The function returns the channel of all rising and falling edges. For example for the Time Tagger 20 (8 input channels) TT_CHANNEL_NUMBER_SCHEME_ZERO: {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} and for TT_CHANNEL↩_NUMBER_SCHEME_ONE: {-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8}

TT_CHANNEL_RISING_EDGES returns only the rising edges SCHEME_ONE: {1,2,3,4,5,6,7,8} and TT_↩CHANNEL_FALLING_EDGES return only the falling edges SCHEME_ONE: {-1,-2,-3,-4,-5,-6,-7,-8} which are the invertedChannels of the rising edges.

#### 9.69.2.8 getChannelNumberScheme()

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

Fetch the configured numbering scheme for this TimeTagger object.

Please see setTimeTaggerChannelNumberScheme() for details.

#### 9.69.2.9 getConditionalFilterFiltered()

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

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.69.2.10 getConditionalFilterTrigger()

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

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.69.2.11 getDACRange()

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

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

#### 9.69.2.12 getDeviceLicense()

```
virtual std::string TimeTagger::getDeviceLicense ( ) [pure virtual]
```

gets the license, installed on this device currently

**Returns**

a JSON string containing the current device license

#### 9.69.2.13 getDistributionCount()

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

get internal calibration data

#### 9.69.2.14 getDistributionPSecs()

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

get internal calibration data

#### 9.69.2.15 getEventDivider()

```
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

#### 9.69.2.16 getFirmwareVersion()

```
virtual std::string TimeTagger::getFirmwareVersion ( ) [pure virtual]
```

Return an unique identifier for the applied firmware.

This function returns a comma separated list of the firmware version with

- the device identifier: TT-20 or TT-Ultra
- the firmware identifier: FW 3
- optional the timestamp of the assembling of the firmware
- the firmware identifier of the USB chip: OK 1.30 eg "TT-Ultra, FW 3, TS 2018-11-13 22:57:32, OK 1.30"

#### 9.69.2.17 getHardwareBufferSize()

```
virtual int TimeTagger::getHardwareBufferSize ( ) [pure virtual]
```

queries the size of the USB queue

See setHardwareBufferSize for more information.

**Returns**

the actual size of the USB queue in events

#### 9.69.2.18 getHardwareDelayCompensation()

```
virtual timestamp_t TimeTagger::getHardwareDelayCompensation (
            channel_t channel ) [pure virtual]
```

get hardware delay compensation of a channel

The physical input delays are calibrated and compensated. However this compensation is implemented after the conditional filter and so affects its result. This function queries the effective input delay, which compensates the hardware delay.

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the hardware delay compensation in picoseconds

### 9.69.2.19 getInputHysteresis()

```
virtual int TimeTagger::getInputHysteresis (
            channel_t channel )  [pure virtual]
```

query the hysteresis voltage of the input comparator

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

the hysteresis voltage in milli Volt

### 9.69.2.20 getInputImpedanceHigh()

```
virtual bool TimeTagger::getInputImpedanceHigh (
            channel_t channel )  [pure virtual]
```

query the state of the high impedance termination mode

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

true for the high impedance mode or false for the 50 Ohm termination mode

### 9.69.2.21 getInputMux()

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

fetches the configuration of the input multiplexer

**Parameters**

| | |
|---|---|
| *channel* | the physical channel of the input multiplexer |

**Returns**

the configuration mode of the input multiplexer

#### 9.69.2.22 getModel()

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

identifies the hardware by Time Tagger Model

#### 9.69.2.23 getNormalization()

```
virtual bool TimeTagger::getNormalization (
            channel_t channel )  [pure virtual]
```

returns the the normalization of the distribution.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *channel* | the channel to query |

**Returns**

if the normalization is enabled

#### 9.69.2.24 getPcbVersion()

```
virtual std::string TimeTagger::getPcbVersion ( )  [pure virtual]
```

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.

#### 9.69.2.25 getPsPerClock()

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

fetch the duration of each clock cycle in picoseconds

#### 9.69.2.26 getSensorData()

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

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

#### 9.69.2.27 getSerial()

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

identifies the hardware by serial number

#### 9.69.2.28 getStreamBlockSizeEvents()

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

#### 9.69.2.29 getStreamBlockSizeLatency()

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

#### 9.69.2.30 getTestSignalDivider()

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

get the divider for the frequency of the test signal

#### 9.69.2.31 getTriggerLevel()

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

get the trigger voltage threshold of a channel

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.69.2.32 isChannelRegistered()

```
virtual bool TimeTagger::isChannelRegistered (
            channel_t chan )  [pure virtual]
```

#### 9.69.2.33 isServerRunning()

```
virtual bool TimeTagger::isServerRunning ( )  [pure virtual]
```

check if the server is still running.

**Returns**

returns true if running; false, if not running

#### 9.69.2.34 reset()

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

reset the TimeTagger object to default settings and detach all iterators

#### 9.69.2.35 setConditionalFilter()

```
virtual void TimeTagger::setConditionalFilter (
            std::vector< channel_t > trigger,
            std::vector< channel_t > filtered,
            bool hardwareDelayCompensation = true ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *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 |

#### 9.69.2.36 setEventDivider()

```
virtual void TimeTagger::setEventDivider (
            channel_t channel,
            unsigned int divider ) [pure virtual]
```

Divides the amount of transmitted edge per channel.

This filter decimates the events on a given channel by a specified. factor. So for a divider n, every 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 at least 1 and smaller than 65536 |

#### 9.69.2.37 setHardwareBufferSize()

```
virtual void TimeTagger::setHardwareBufferSize (
            int size ) [pure virtual]
```

sets the maximum USB buffer size

This option controls the maximum buffer size of the USB connection. This can be used to balance low input latency vs high (peak) throughput.

**Parameters**

| | |
|---|---|
| *size* | the maximum buffer size in events |

#### 9.69.2.38 setInputHysteresis()

```
virtual void TimeTagger::setInputHysteresis (
            channel_t channel,
            int value ) [pure virtual]
```

configure the hysteresis voltage of the input comparator

Caution: This feature is only supported on the Time Tagger X The supported hysteresis voltages are 1 mV, 20 mV or 70 mV

**Parameters**

| | |
|---|---|
| *channel* | channel to be configured |
| *value* | the hysteresis voltage in milli Volt |

#### 9.69.2.39 setInputImpedanceHigh()

```
virtual void TimeTagger::setInputImpedanceHigh (
            channel_t channel,
            bool high_impedance ) [pure virtual]
```

enable high impedance termination mode

Caution: This feature is only supported on the Time Tagger X

**Parameters**

| | |
|---|---|
| *channel* | channel to be configured |
| *high_impedance* | set for the high impedance mode or cleared for the 50 Ohm termination mode |

#### 9.69.2.40 setInputMux()

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

configures the input multiplexer

Every physical input channel has an input multiplexer with 4 modes: 0: normal input mode 1: use the input from channel -1 (left) 2: use the input from channel +1 (right) 3: use the reference oscillator

Mode 1 and 2 cascades, so many inputs can be configured to get the same input events.

**Parameters**

| | |
|---|---|
| *channel* | the physical channel of the input multiplexer |
| *mux_mode* | the configuration mode of the input multiplexer |

#### 9.69.2.41 setLED()

```
virtual void TimeTagger::setLED (
            uint32_t bitmask ) [pure virtual]
```

Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.

#### 9.69.2.42 setNormalization()

```
virtual void TimeTagger::setNormalization (
            std::vector< channel_t > channels,
            bool state ) [pure virtual]
```

enables or disables the normalization of the distribution.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *channels* | list of channels to modify |
| *state* | the new state |

#### 9.69.2.43 setSoundFrequency()

```
virtual void TimeTagger::setSoundFrequency (
            uint32_t freq_hz ) [pure virtual]
```

Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)

**Parameters**

| | |
|---|---|
| *freq_hz* | the generated audio frequency |

#### 9.69.2.44 setStreamBlockSize()

```
virtual void TimeTagger::setStreamBlockSize (
            int max_events,
            int max_latency ) [pure virtual]
```

sets the maximum events and latency for the stream block size

This option controls the latency and the block size of the data stream. The default values are max_events = 131072 events and max_latency = 20 ms. Depending on which of the two parameters is exceeded first, the block stream size is adjusted accordingly. The block size will be reduced automatically for blocks when no signal is arriving for 512 ns on the Time Tagger Ultra and 1536 ns for the Time Tagger 20. $*$

**Parameters**

| | |
|---|---|
| *max_events* | maximum number of events |
| *max_latency* | maximum latency in ms |

#### 9.69.2.45 setTestSignalDivider()

```
virtual void TimeTagger::setTestSignalDivider (
            int divider ) [pure virtual]
```

set the divider for the frequency of the test signal

The base clock of the test signal oscillator for the Time Tagger Ultra is running at 100.8 MHz sampled down by an factor of 2 to have a similar base clock as the Time Tagger 20 ($\sim$50 MHz). The default divider is 63 -> $\sim$800 kEvents/s. The base clock for the TTX is 333.3 MHz. The default divider is tuned to $\sim$800 kEvents/s

**Parameters**

| | |
|---|---|
| *divider* | frequency divisor of the oscillator |

#### 9.69.2.46 setTimeTaggerNetworkStreamCompression()

```
virtual void TimeTagger::setTimeTaggerNetworkStreamCompression (
            bool active ) [pure virtual]
```

enable or disable additional compression of the timetag stream as ent over the network.

**Parameters**

| | |
|---|---|
| *active* | set if the compression is active or not. |

#### 9.69.2.47 setTriggerLevel()

```
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] |

#### 9.69.2.48 startServer()

```
virtual void TimeTagger::startServer (
            AccessMode access_mode,
            std::vector< channel_t > channels = std::vector< channel_t >(),
            uint32_t port = 41101 )  [pure virtual]
```

starts the Time Tagger server that will stream the time tags to the client.

**Parameters**

| | |
|---|---|
| *access_mode* | set the type of access a user can have. |
| *port* | port at which this time tagger server will be listening on. |
| *channels* | channels to be streamed, if empty, all the channels will be exposed. |

#### 9.69.2.49 stopServer()

```
virtual void TimeTagger::stopServer ( )  [pure virtual]
```

stops the time tagger server if currently running, otherwise does nothing.

#### 9.69.2.50 updateBMCFirmware()

```
virtual void TimeTagger::updateBMCFirmware (
            const std::string & firmware )  [pure virtual]
```

updates the firmware of the Time Tagger X board management controller

**Note**

The firmware is applied on the next power cycle of the device, *not* on pressing the power button.

**Parameters**

| | |
|---|---|
| *firmware* | filename of the new firmware on disc |

#### 9.69.2.51 xtra_getAuxOut()

```
virtual bool TimeTagger::xtra_getAuxOut (
            int channel )  [pure virtual]
```

fetch the status of the aux out signal generator

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *channel* | select Aux Out 1 or 2 |

**Returns**

true if the aux out signal generator is enabled

#### 9.69.2.52 xtra_getAuxOutSignalDivider()

```
virtual int TimeTagger::xtra_getAuxOutSignalDivider (
            int channel )  [pure virtual]
```

get the divider for the frequency of the aux out signal generator

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *channel* | select Aux Out 1 or 2 |

**Returns**

the configured divider

#### 9.69.2.53 xtra_getAuxOutSignalDutyCycle()

```
virtual double TimeTagger::xtra_getAuxOutSignalDutyCycle (
            int channel )  [pure virtual]
```

get the dutycycle of the aux out signal generator

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *channel* | select Aux Out 1 or 2 |

**Returns**

the configured duty cycle

### 9.69.2.54 xtra_getAvgRisingFalling()

```
virtual bool TimeTagger::xtra_getAvgRisingFalling (
            channel_t channel ) [pure virtual]
```

query if the rising and falling events shall be averaged

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

if the rising and falling events shall be averaged

### 9.69.2.55 xtra_getClockAutoSelect()

```
virtual bool TimeTagger::xtra_getClockAutoSelect ( ) [pure virtual]
```

queries if the auto clocking function is enabled

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Returns**

true if the external clock auto detection is enabled

### 9.69.2.56 xtra_getClockSource()

```
virtual int TimeTagger::xtra_getClockSource ( ) [pure virtual]
```

fetch the overwritten reference clock source

-1: auto selecting of below options 0: internal clock 1: external 10 MHz 2: external 500 MHz

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Returns**

selects the clock source

### 9.69.2.57 xtra_getHighPrioChannel()

```
virtual bool TimeTagger::xtra_getHighPrioChannel (
            channel_t channel ) [pure virtual]
```

if this channel shall exit overflow regions

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

if this channel shall exit overflow regions

#### 9.69.2.58 xtra_measureTriggerLevel()

```
virtual double TimeTagger::xtra_measureTriggerLevel (
            channel_t channel )  [pure virtual]
```

measures the electrically applied the trigger voltage threshold of a channel

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the voltage

#### 9.69.2.59 xtra_setAuxOut()

```
virtual void TimeTagger::xtra_setAuxOut (
            int channel,
            bool enabled )  [pure virtual]
```

enable or disable aux out

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

This will enable or disable the signal generator on the aux outputs.

**Parameters**

| | |
|---|---|
| *channel* | select Aux Out 1 or 2 |
| *enabled* | enabled / disabled flag |

#### 9.69.2.60 xtra_setAuxOutSignal()

```
virtual void TimeTagger::xtra_setAuxOutSignal (
            int channel,
```

```
int divider,
double duty_cycle = 0.5 )  [pure virtual]
```

set the divider for the frequency of the aux out signal generator and enable aux out

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

The base clock for the TTX is 333.3 MHz.

**Parameters**

| | |
|---|---|
| *channel* | select Aux Out 1 or 2 |
| *divider* | frequency divisor of the oscillator |
| *duty_cycle* | the duty cycle ratio, will be clamped and rounded to an integer divisor |

#### 9.69.2.61 xtra_setAvgRisingFalling()

```
virtual void TimeTagger::xtra_setAvgRisingFalling (
            channel_t channel,
            bool enable )  [pure virtual]
```

configures if the rising and falling events shall be averaged

This is implemented on the device before any filter like event divider and it does not require to transfer both events.

They need to be manually delayed to be within a window of +-500 ps of error, else events might get lost. This method has no side effects on the channel `getInvertedChannel(channel)`, you can still fetch the original events there. However if both are configured to return the averaged result, the timestamps will be identical.

**Parameters**

| | |
|---|---|
| *channel* | the channel, on which the average value shall be returned |
| *enable* | true if this channel shall yield the averaged timestamps |

#### 9.69.2.62 xtra_setClockAutoSelect()

```
virtual void TimeTagger::xtra_setClockAutoSelect (
            bool enabled )  [pure virtual]
```

activates auto clocking function

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *enabled* | true for auto detection of external clock |

#### 9.69.2.63 xtra_setClockOut()

```
virtual void TimeTagger::xtra_setClockOut (
             bool enabled )  [pure virtual]
```

enables the clock output

Caution: this feature is for development purposes only and may not be part of future builds without further notice.

**Parameters**

| | |
|---|---|
| *enabled* | true for enabling the 10 MHz clock output |

#### 9.69.2.64 xtra_setClockSource()

```
virtual void TimeTagger::xtra_setClockSource (
             int source )  [pure virtual]
```

manually overwrite the reference clock source

0: internal clock 1: external 10 MHz 2: external 500 MHz

**Parameters**

| | |
|---|---|
| *source* | selects the clock source |

#### 9.69.2.65 xtra_setFanSpeed()

```
virtual void TimeTagger::xtra_setFanSpeed (
             double percentage = -1 )  [pure virtual]
```

configures the FAN speed on TTU HW >= 1.3

**Parameters**

| | |
|---|---|
| *percentage* | the new speed, 0 means off, 100 means full on, negative means controlled. |

**Note**

This setting will get reset on USB errors.

#### 9.69.2.66 xtra_setHighPrioChannel()

```
virtual void TimeTagger::xtra_setHighPrioChannel (
             channel_t channel,
             bool enable )  [pure virtual]
```

configures if this channel shall exit overflow regions.

If configured, each event of this channel within an overflow region will emit an OverflowEnd marker before the event and an OverflowBegin marker after the event. This can be used to split up regions by a slow trigger, e.g. for CountBetweenMarker usage.

**Warning**

Using this option disables the internal safety method for unrecoverable memory overflows. So only use this option on channels with a low datarate, else expect to get Error events, which invalidates the global time.

**Parameters**

| | |
|---|---|
| *channel* | the channel, which shall be configured as high priority |
| *enable* | true if this channel shall have a high priority |

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

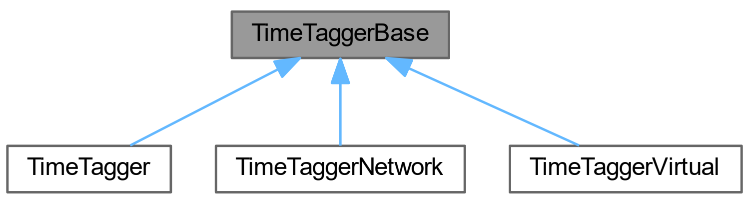
- TimeTagger.h

## 9.70 TimeTaggerBase Class Reference

Basis interface for all Time Tagger classes.

`#include <TimeTagger.h>`

Inheritance diagram for TimeTaggerBase:




### Public Types

- typedef std::function< void(IteratorBase ∗) IteratorCallback)
- typedef std::map< IteratorBase ∗, IteratorCallback > IteratorCallbackMap

### Public Member Functions

- virtual unsigned int getFence (bool alloc_fence=true)=0

    *Generate a new fence object, which validates the current configuration and the current time.*
- virtual bool waitForFence (unsigned int fence, int64_t timeout=-1)=0

    *Wait for a fence in the data stream.*
- virtual bool sync (int64_t timeout=-1)=0

    *Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.*
- virtual channel_t getInvertedChannel (channel_t channel)=0

    *get the falling channel id for a rising channel and vice versa*
- virtual bool isUnusedChannel (channel_t channel)=0

    *compares the provided channel with CHANNEL_UNUSED*
- virtual void runSynchronized (const IteratorCallbackMap &callbacks, bool block=true)=0

    *Run synchronized callbacks for a list of iterators.*
- virtual std::string getConfiguration ()=0

    *Fetches the overall configuration status of the Time Tagger object.*
- virtual void setInputDelay (channel_t channel, timestamp_t delay)=0

    *set time delay on a channel*
- virtual void setDelayHardware (channel_t channel, timestamp_t delay)=0

    *set time delay on a channel*
- virtual void setDelaySoftware (channel_t channel, timestamp_t delay)=0

    *set time delay on a channel*
- virtual timestamp_t getInputDelay (channel_t channel)=0

    *get time delay of a channel*
- virtual timestamp_t getDelaySoftware (channel_t channel)=0

    *get time delay of a channel*
- virtual timestamp_t getDelayHardware (channel_t channel)=0

    *get time delay of a channel*
- virtual timestamp_t setDeadtime (channel_t channel, timestamp_t deadtime)=0

    *set the deadtime between two edges on the same channel.*
- virtual timestamp_t getDeadtime (channel_t channel)=0

    *get the deadtime between two edges on the same channel.*
- virtual void setTestSignal (channel_t channel, bool enabled)=0

    *enable/disable internal test signal on a channel.*
- virtual void setTestSignal (std::vector< channel_t > channel, bool enabled)=0

    *enable/disable internal test signal on multiple channels.*
- virtual bool getTestSignal (channel_t channel)=0

    *fetch the status of the test signal generator*
- virtual void setSoftwareClock (channel_t input_channel, double input_frequency=10e6, double averaging_↩ periods=1000, bool wait_until_locked=true)=0

    *enables a software PLL to lock the time to an external clock*
- virtual void disableSoftwareClock ()=0

    *disabled the software PLL*
- virtual SoftwareClockState getSoftwareClockState ()=0

    *queries all state information of the software clock*
- virtual long long getOverflows ()=0

    *get overflow count*
- virtual void clearOverflows ()=0

    *clear overflow counter*
- virtual long long getOverflowsAndClear ()=0

    *get and clear overflow counter*

**Protected Member Functions**

- TimeTaggerBase ()

    *abstract interface class*
- virtual ∼TimeTaggerBase ()

    *destructor*
- TimeTaggerBase (const TimeTaggerBase &)=delete
- TimeTaggerBase & operator= (const TimeTaggerBase &)=delete
- virtual std::shared_ptr< IteratorBaseListNode > addIterator (IteratorBase *it)=0
- virtual void freeIterator (IteratorBase *it)=0
- virtual channel_t getNewVirtualChannel ()=0
- virtual void freeVirtualChannel (channel_t channel)=0
- virtual void registerChannel (channel_t channel)=0

    *register a FPGA channel.*
- virtual void registerChannel (std::set< channel_t > channels)=0
- virtual void unregisterChannel (channel_t channel)=0

    *release a previously registered channel.*
- virtual void unregisterChannel (std::set< channel_t > channels)=0
- virtual void addChild (TimeTaggerBase *child)=0
- virtual void removeChild (TimeTaggerBase *child)=0
- virtual void release ()=0

## 9.70.1 Detailed Description

Basis interface for all Time Tagger classes.

This basis interface represents all common methods to add, remove, and run measurements.

## 9.70.2 Member Typedef Documentation

### 9.70.2.1 IteratorCallback

```
typedef std::function<void(IteratorBase *) TimeTaggerBase::IteratorCallback)
```

### 9.70.2.2 IteratorCallbackMap

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

## 9.70.3 Constructor & Destructor Documentation

### 9.70.3.1 TimeTaggerBase() [1/2]

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

abstract interface class

#### 9.70.3.2 ∼TimeTaggerBase()

```
virtual TimeTaggerBase::~TimeTaggerBase ( ) [inline], [protected], [virtual]
```

destructor

#### 9.70.3.3 TimeTaggerBase() [2/2]

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

### 9.70.4 Member Function Documentation

#### 9.70.4.1 addChild()

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

#### 9.70.4.2 addIterator()

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

#### 9.70.4.3 clearOverflows()

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

clear overflow counter

Sets the overflow counter to zero

#### 9.70.4.4 disableSoftwareClock()

```
virtual void TimeTaggerBase::disableSoftwareClock ( ) [pure virtual]
```

disabled the software PLL

See setSoftwareClock for further details.

#### 9.70.4.5 freeIterator()

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

#### 9.70.4.6 freeVirtualChannel()

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

#### 9.70.4.7 getConfiguration()

```
virtual std::string TimeTaggerBase::getConfiguration ( ) [pure virtual]
```

Fetches the overall configuration status of the Time Tagger object.

**Returns**

a JSON serialized string with all configuration and status flags.

#### 9.70.4.8 getDeadtime()

```
virtual timestamp_t TimeTaggerBase::getDeadtime (
            channel_t channel ) [pure virtual]
```

get the deadtime between two edges on the same channel.

This function gets the user configurable deadtime.

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

the real configured deadtime in picoseconds

#### 9.70.4.9 getDelayHardware()

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

get time delay of a channel

see setDelayHardware

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the hardware delay in picoseconds

#### 9.70.4.10 getDelaySoftware()

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

get time delay of a channel

see setDelaySoftware

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the software delay in picoseconds

#### 9.70.4.11 getFence()

```
virtual unsigned int TimeTaggerBase::getFence (
            bool alloc_fence = true ) [pure virtual]
```

Generate a new fence object, which validates the current configuration and the current time.

This fence is uploaded to the earliest pipeline stage of the Time Tagger. Waiting on this fence ensures that all hardware settings such as trigger levels, channel registrations, etc., have propagated to the FPGA and are physically active. Synchronizes the Time Tagger internal memory, so that all tags arriving after the waitForFence call were actually produced after the getFence call. The waitForFence function waits until all tags, which are present at the time of the function call within the internal memory of the Time Tagger, are processed. This call might block to limit the amount of active fences.

**Parameters**

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

**Returns**

the allocated fence

#### 9.70.4.12 getInputDelay()

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

get time delay of a channel

see setInputDelay

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the software delay in picoseconds

#### 9.70.4.13 getInvertedChannel()

```
virtual channel_t TimeTaggerBase::getInvertedChannel (
            channel_t channel )  [pure virtual]
```

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

If this channel has no inverted channel, UNUSED_CHANNEL is returned. This is the case for most virtual channels.

**Parameters**

| | |
|---|---|
| *channel* | The channel id to query |

**Returns**

the inverted channel id

#### 9.70.4.14 getNewVirtualChannel()

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

#### 9.70.4.15 getOverflows()

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

get overflow count

Get the number of communication overflows occurred

#### 9.70.4.16 getOverflowsAndClear()

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

get and clear overflow counter

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

#### 9.70.4.17 getSoftwareClockState()

```
virtual SoftwareClockState TimeTaggerBase::getSoftwareClockState ( )  [pure virtual]
```

queries all state information of the software clock

See setSoftwareClock for further details.

#### 9.70.4.18 getTestSignal()

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

fetch the status of the test signal generator

**Parameters**

| | |
|---|---|
| *channel* | the channel |

Implemented in TimeTaggerNetwork.

#### 9.70.4.19 isUnusedChannel()

```
virtual bool TimeTaggerBase::isUnusedChannel (
            channel_t channel )  [pure virtual]
```

compares the provided channel with CHANNEL_UNUSED

But also keeps care about the channel number scheme and selects either CHANNEL_UNUSED or CHANNEL_↩
UNUSED_OLD

#### 9.70.4.20 operator=()

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

#### 9.70.4.21 registerChannel() [1/2]

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

register a FPGA channel.

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

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.70.4.22 registerChannel() [2/2]

```
virtual void TimeTaggerBase::registerChannel (
            std::set< channel_t > channels )  [protected], [pure virtual]
```

#### 9.70.4.23 release()

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

#### 9.70.4.24 removeChild()

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

#### 9.70.4.25 runSynchronized()

```
virtual void TimeTaggerBase::runSynchronized (
            const IteratorCallbackMap & callbacks,
            bool block = true )  [pure virtual]
```

Run synchronized callbacks for a list of iterators.

This method has a list of callbacks for a list of iterators. Those callbacks are called for a synchronized data set, but in parallel. They are called from an internal worker thread. As the data set is synchronized, this creates a bottleneck for one worker thread, so only fast and non-blocking callbacks are allowed.

**Parameters**

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

#### 9.70.4.26 setDeadtime()

```
virtual timestamp_t TimeTaggerBase::setDeadtime (
            channel_t channel,
            timestamp_t deadtime )  [pure virtual]
```

set the deadtime between two edges on the same channel.

This function sets the user configurable deadtime. The requested time will be rounded to the nearest multiple of the clock time. The deadtime will also be clamped to device specific limitations.

As the actual deadtime will be altered, the real value will be returned.

**Parameters**

| | |
|---|---|
| *channel* | channel to be configured |
| *deadtime* | new deadtime in picoseconds |

**Returns**

the real configured deadtime in picoseconds

#### 9.70.4.27 setDelayHardware()

```
virtual void TimeTaggerBase::setDelayHardware (
            channel_t channel,
            timestamp_t delay )  [pure virtual]
```

set time delay on a channel

When set, every event on this physical input channel is delayed by the given delay in picoseconds. This delay is implemented on the hardware before any filter with no performance overhead. The maximum delay on the Time Tagger Ultra series is 2 us. This affects both the rising and the falling event at the same time.

**Parameters**

| | |
|---|---|
| *channel* | the channel to set |
| *delay* | the hardware delay in picoseconds |

#### 9.70.4.28 setDelaySoftware()

```
virtual void TimeTaggerBase::setDelaySoftware (
            channel_t channel,
            timestamp_t delay )  [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds. This happens on the computer and so after the on-device filters. Please use setDelayHardware instead for better performance. This affects either the 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 software delay in picoseconds |

#### 9.70.4.29 setInputDelay()

```
virtual void TimeTaggerBase::setInputDelay (
            channel_t channel,
            timestamp_t delay )  [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds.

This method has the best performance with "small delays". The delay is considered "small" when less than 100 events arrive within the time of the largest delay set. For example, if the total event-rate over all channels used is 10 Mevent/s, the signal can be delayed efficiently up to 10 microseconds. For large delays, please use DelayedChannel instead.

**Parameters**

| | |
|---|---|
| *channel* | the channel to set |
| *delay* | the delay in picoseconds |

#### 9.70.4.30 setSoftwareClock()

```
virtual void TimeTaggerBase::setSoftwareClock (
            channel_t input_channel,
            double input_frequency = 10e6,
            double averaging_periods = 1000,
            bool wait_until_locked = true )  [pure virtual]
```

enables a software PLL to lock the time to an external clock

This feature implements a software PLL on the CPU. This can replace external clocks with no restrictions on correlated data to other inputs. It uses a first-order loop filter to ignore the discretization noise of the input and to provide some kind of cutoff frequency when to apply the extern clock.

**Note**

Within the first 100 ∗ averaging_factor ∗ clock_period, a frequency locking approach is applied. The phase gets locked afterwards.

**Parameters**

| | |
|---|---|
| *input_channel* | The physical input channel |
| *input_frequency* | Frequency of the configured external clock. Slight variations will be canceled out. Defaults to 10e6 for 10 MHz |
| *averaging_periods* | Times clock_period is the cutoff period for the filter. Shorter periods are evaluated with the Time Tagger's internal clock, longer periods are evaluated with the here configured external clock |
| *wait_until_locked* | Blocks the execution until the software clock is locked. Throws an exception on locking errors. All locking log messages are filtered while this call is executed. |

#### 9.70.4.31 setTestSignal() [1/2]

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

enable/disable internal test signal on a channel.

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

**Parameters**

| | |
|---|---|
| *channel* | the channel |
| *enabled* | enabled / disabled flag |

#### 9.70.4.32 setTestSignal() [2/2]

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

enable/disable internal test signal on multiple channels.

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

**Parameters**

| | |
|---|---|
| *channel* | a vector of channels |
| *enabled* | enabled / disabled flag |

#### 9.70.4.33 sync()

```
virtual bool TimeTaggerBase::sync (
             int64_t timeout = -1 )  [pure virtual]
```

Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.

This is a shortcut for calling getFence and waitForFence at once. See getFence for more details.

**Parameters**

| | |
|---|---|
| *timeout* | timeout in milliseconds. Negative means no timeout, zero returns immediately. |

**Returns**

true on success, false on timeout

#### 9.70.4.34 unregisterChannel() [1/2]

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

release a previously registered channel.

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.70.4.35 unregisterChannel() [2/2]

```
virtual void TimeTaggerBase::unregisterChannel (
            std::set< channel_t > channels )  [protected], [pure virtual]
```

#### 9.70.4.36 waitForFence()

```
virtual bool TimeTaggerBase::waitForFence (
            unsigned int fence,
            int64_t timeout = -1 )  [pure virtual]
```

Wait for a fence in the data stream.

See getFence for more details.

**Parameters**

| | |
|---|---|
| *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

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

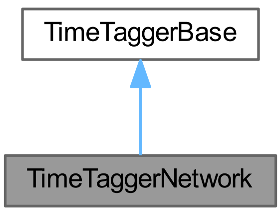
- TimeTagger.h

## 9.71 TimeTaggerNetwork Class Reference

network TimeTagger client.

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

Inheritance diagram for TimeTaggerNetwork:



### Public Member Functions

- virtual bool isConnected ()=0

  *check if the network time tagger is currently connected to a server*
- virtual void setTriggerLevel (channel_t channel, double voltage)=0

  *set the trigger voltage threshold of a channel*
- virtual double getTriggerLevel (channel_t channel)=0

  *get the trigger voltage threshold of a channel*
- virtual void setConditionalFilter (std::vector< channel_t > trigger, std::vector< channel_t > filtered, bool hardwareDelayCompensation=true)=0

  *configures the conditional filter*
- virtual void clearConditionalFilter ()=0

  *deactivates the conditional filter*
- virtual std::vector< channel_t > getConditionalFilterTrigger ()=0

  *fetches the configuration of the conditional filter*
- virtual std::vector< channel_t > getConditionalFilterFiltered ()=0

  *fetches the configuration of the conditional filter*
- virtual void setTestSignalDivider (int divider)=0

  *set the divider for the frequency of the test signal*
- virtual int getTestSignalDivider ()=0

  *get the divider for the frequency of the test signal*
- virtual bool getTestSignal (channel_t channel)=0

  *fetch the status of the test signal generator*
- virtual void setDelayClient (channel_t channel, timestamp_t time)=0

  *set time delay on a channel*
- virtual timestamp_t getDelayClient (channel_t channel)=0

  *get the time delay of a channel*
- virtual timestamp_t getHardwareDelayCompensation (channel_t channel)=0

  *get hardware delay compensation of a channel*
- virtual void setNormalization (std::vector< channel_t > channels, bool state)=0

  *enables or disables the normalization of the distribution.*
- virtual bool getNormalization (channel_t channel)=0

  *returns the the normalization of the distribution.*
- virtual void setHardwareBufferSize (int size)=0

  *sets the maximum USB buffer size*
- virtual int getHardwareBufferSize ()=0

  *queries the size of the USB queue*
- virtual void setStreamBlockSize (int max_events, int max_latency)=0

  *sets the maximum events and latency for the stream block size*
- virtual int getStreamBlockSizeEvents ()=0
- virtual int getStreamBlockSizeLatency ()=0
- virtual void setEventDivider (channel_t channel, unsigned int divider)=0

  *Divides the amount of transmitted edge per channel.*
- virtual unsigned int getEventDivider (channel_t channel)=0

  *Returns the factor of the dividing filter.*
- virtual std::string getSerial ()=0

  *identifies the hardware by serial number*
- virtual std::string getModel ()=0

  *identifies the hardware by Time Tagger Model*
- virtual int getChannelNumberScheme ()=0

  *Fetch the configured numbering scheme for this TimeTagger object.*

- virtual std::vector< double > getDACRange ()=0

  *returns the minimum and the maximum voltage of the DACs as a trigger reference*
- virtual std::vector< channel_t > getChannelList (ChannelEdge type=ChannelEdge::All)=0

  *fetch a vector of all physical input channel ids*
- virtual timestamp_t getPsPerClock ()=0

  *fetch the duration of each clock cycle in picoseconds*
- virtual std::string getPcbVersion ()=0

  *Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.*
- virtual std::string getFirmwareVersion ()=0

  *Return an unique identifier for the applied firmware.*
- virtual std::string getSensorData ()=0

  *Show the status of the sensor data from the FPGA and peripherals on the console.*
- virtual void setLED (uint32_t bitmask)=0

  *Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.*
- virtual std::string getDeviceLicense ()=0

  *gets the license, installed on this device currently*
- virtual void setSoundFrequency (uint32_t freq_hz)=0

  *Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)*
- virtual void setTimeTaggerNetworkStreamCompression (bool active)=0

  *enable or disable additional compression of the timetag stream as ent over the network.*
- virtual long long getOverflowsClient ()=0
- virtual void clearOverflowsClient ()=0
- virtual long long getOverflowsAndClearClient ()=0
- virtual void setInputImpedanceHigh (channel_t channel, bool high_impedance)=0

  *enable high impedance termination mode*
- virtual bool getInputImpedanceHigh (channel_t channel)=0

  *query the state of the high impedance termination mode*
- virtual void setInputHysteresis (channel_t channel, int value)=0

  *configure the hysteresis voltage of the input comparator*
- virtual int getInputHysteresis (channel_t channel)=0

  *query the hysteresis voltage of the input comparator*

## Public Member Functions inherited from TimeTaggerBase

- virtual unsigned int getFence (bool alloc_fence=true)=0

  *Generate a new fence object, which validates the current configuration and the current time.*
- virtual bool waitForFence (unsigned int fence, int64_t timeout=-1)=0

  *Wait for a fence in the data stream.*
- virtual bool sync (int64_t timeout=-1)=0

  *Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.*
- virtual channel_t getInvertedChannel (channel_t channel)=0

  *get the falling channel id for a rising channel and vice versa*
- virtual bool isUnusedChannel (channel_t channel)=0

  *compares the provided channel with CHANNEL_UNUSED*
- virtual void runSynchronized (const IteratorCallbackMap &callbacks, bool block=true)=0

  *Run synchronized callbacks for a list of iterators.*
- virtual std::string getConfiguration ()=0

*Fetches the overall configuration status of the Time Tagger object.*

- virtual void setInputDelay (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelayHardware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelaySoftware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual timestamp_t getInputDelay (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t getDelaySoftware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t getDelayHardware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t setDeadtime (channel_t channel, timestamp_t deadtime)=0

  *set the deadtime between two edges on the same channel.*
- virtual timestamp_t getDeadtime (channel_t channel)=0

  *get the deadtime between two edges on the same channel.*
- virtual void setTestSignal (channel_t channel, bool enabled)=0

  *enable/disable internal test signal on a channel.*
- virtual void setTestSignal (std::vector< channel_t > channel, bool enabled)=0

  *enable/disable internal test signal on multiple channels.*
- virtual void setSoftwareClock (channel_t input_channel, double input_frequency=10e6, double averaging_←periods=1000, bool wait_until_locked=true)=0

  *enables a software PLL to lock the time to an external clock*
- virtual void disableSoftwareClock ()=0

  *disabled the software PLL*
- virtual SoftwareClockState getSoftwareClockState ()=0

  *queries all state information of the software clock*
- virtual long long getOverflows ()=0

  *get overflow count*
- virtual void clearOverflows ()=0

  *clear overflow counter*
- virtual long long getOverflowsAndClear ()=0

  *get and clear overflow counter*

### Additional Inherited Members

## Public Types inherited from TimeTaggerBase

- typedef std::function< void(IteratorBase *) IteratorCallback)
- typedef std::map< IteratorBase *, IteratorCallback > IteratorCallbackMap

### Protected Member Functions inherited from TimeTaggerBase

- TimeTaggerBase ()

    *abstract interface class*
- virtual ∼TimeTaggerBase ()

    *destructor*
- TimeTaggerBase (const TimeTaggerBase &)=delete
- TimeTaggerBase & operator= (const TimeTaggerBase &)=delete
- virtual std::shared_ptr< IteratorBaseListNode > addIterator (IteratorBase ∗it)=0
- virtual void freeIterator (IteratorBase ∗it)=0
- virtual channel_t getNewVirtualChannel ()=0
- virtual void freeVirtualChannel (channel_t channel)=0
- virtual void registerChannel (channel_t channel)=0

    *register a FPGA channel.*
- virtual void registerChannel (std::set< channel_t > channels)=0
- virtual void unregisterChannel (channel_t channel)=0

    *release a previously registered channel.*
- virtual void unregisterChannel (std::set< channel_t > channels)=0
- virtual void addChild (TimeTaggerBase ∗child)=0
- virtual void removeChild (TimeTaggerBase ∗child)=0
- virtual void release ()=0

## 9.71.1 Detailed Description

network TimeTagger client.

The TimeTaggerNetwork class is a client that implements access to the Time Tagger server. TimeTaggerNetwork receives the time-tag stream from the Time Tagger server over the network and provides an interface for controlling connection and the Time Tagger hardware. Instance of this class can be transparently used to create measurements, virtual channels and other Iterator instances.

## 9.71.2 Member Function Documentation

### 9.71.2.1 clearConditionalFilter()

```
virtual void TimeTaggerNetwork::clearConditionalFilter ( )  [pure virtual]
```

deactivates the conditional filter

equivalent to setConditionalFilter({},{})

### 9.71.2.2 clearOverflowsClient()

```
virtual void TimeTaggerNetwork::clearOverflowsClient ( )  [pure virtual]
```

#### 9.71.2.3 getChannelList()

```
virtual std::vector< channel_t > TimeTaggerNetwork::getChannelList (
            ChannelEdge type = ChannelEdge::All )  [pure virtual]
```

fetch a vector of all physical input channel ids

The function returns the channel of all rising and falling edges. For example for the Time Tagger 20 (8 input channels) TT_CHANNEL_NUMBER_SCHEME_ZERO: {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} and for TT_CHANNEL↩
_NUMBER_SCHEME_ONE: {-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8}

TT_CHANNEL_RISING_EDGES returns only the rising edges SCHEME_ONE: {1,2,3,4,5,6,7,8} and TT_↩
CHANNEL_FALLING_EDGES return only the falling edges SCHEME_ONE: {-1,-2,-3,-4,-5,-6,-7,-8} which are the invertedChannels of the rising edges.

#### 9.71.2.4 getChannelNumberScheme()

```
virtual int TimeTaggerNetwork::getChannelNumberScheme ( )  [pure virtual]
```

Fetch the configured numbering scheme for this TimeTagger object.

Please see setTimeTaggerChannelNumberScheme() for details.

#### 9.71.2.5 getConditionalFilterFiltered()

```
virtual std::vector< channel_t > TimeTaggerNetwork::getConditionalFilterFiltered ( )  [pure
virtual]
```

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.71.2.6 getConditionalFilterTrigger()

```
virtual std::vector< channel_t > TimeTaggerNetwork::getConditionalFilterTrigger ( )  [pure
virtual]
```

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.71.2.7 getDACRange()

```
virtual std::vector< double > TimeTaggerNetwork::getDACRange ( )  [pure virtual]
```

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

#### 9.71.2.8 getDelayClient()

```
virtual timestamp_t TimeTaggerNetwork::getDelayClient (
            channel_t channel )  [pure virtual]
```

get the time delay of a channel

see setDelayClient

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the software delay in picoseconds

#### 9.71.2.9 getDeviceLicense()

```
virtual std::string TimeTaggerNetwork::getDeviceLicense ( ) [pure virtual]
```

gets the license, installed on this device currently

**Returns**

a JSON string containing the current device license

#### 9.71.2.10 getEventDivider()

```
virtual unsigned int TimeTaggerNetwork::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

#### 9.71.2.11 getFirmwareVersion()

```
virtual std::string TimeTaggerNetwork::getFirmwareVersion ( ) [pure virtual]
```

Return an unique identifier for the applied firmware.

This function returns a comma separated list of the firmware version with

- the device identifier: TT-20 or TT-Ultra
- the firmware identifier: FW 3
- optional the timestamp of the assembling of the firmware
- the firmware identifier of the USB chip: OK 1.30 eg "TT-Ultra, FW 3, TS 2018-11-13 22:57:32, OK 1.30"

#### 9.71.2.12 getHardwareBufferSize()

```
virtual int TimeTaggerNetwork::getHardwareBufferSize ( ) [pure virtual]
```

queries the size of the USB queue

See setHardwareBufferSize for more information.

**Returns**

the actual size of the USB queue in events

#### 9.71.2.13 getHardwareDelayCompensation()

```
virtual timestamp_t TimeTaggerNetwork::getHardwareDelayCompensation (
            channel_t channel ) [pure virtual]
```

get hardware delay compensation of a channel

The physical input delays are calibrated and compensated. However this compensation is implemented after the conditional filter and so affects its result. This function queries the effective input delay, which compensates the hardware delay.

**Parameters**

| | |
|---|---|
| *channel* | the channel |

**Returns**

the hardware delay compensation in picoseconds

#### 9.71.2.14 getInputHysteresis()

```
virtual int TimeTaggerNetwork::getInputHysteresis (
            channel_t channel ) [pure virtual]
```

query the hysteresis voltage of the input comparator

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

the hysteresis voltage in milli Volt

#### 9.71.2.15 getInputImpedanceHigh()

```
virtual bool TimeTaggerNetwork::getInputImpedanceHigh (
```

```
channel_t channel ) [pure virtual]
```

query the state of the high impedance termination mode

**Parameters**

| | |
|---|---|
| *channel* | channel to be queried |

**Returns**

true for the high impedance mode or false for the 50 Ohm termination mode

#### 9.71.2.16 getModel()

```
virtual std::string TimeTaggerNetwork::getModel ( ) [pure virtual]
```

identifies the hardware by Time Tagger Model

#### 9.71.2.17 getNormalization()

```
virtual bool TimeTaggerNetwork::getNormalization (
            channel_t channel ) [pure virtual]
```

returns the the normalization of the distribution.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *channel* | the channel to query |

**Returns**

if the normalization is enabled

#### 9.71.2.18 getOverflowsAndClearClient()

```
virtual long long TimeTaggerNetwork::getOverflowsAndClearClient ( ) [pure virtual]
```

#### 9.71.2.19 getOverflowsClient()

```
virtual long long TimeTaggerNetwork::getOverflowsClient ( ) [pure virtual]
```

#### 9.71.2.20 getPcbVersion()

```
virtual std::string TimeTaggerNetwork::getPcbVersion ( ) [pure virtual]
```

Return the hardware version of the PCB board. Version 0 is everything before mid 2018 and with the channel configuration ZERO. version >= 1 is channel configuration ONE.

#### 9.71.2.21 getPsPerClock()

```
virtual timestamp_t TimeTaggerNetwork::getPsPerClock ( )  [pure virtual]
```

fetch the duration of each clock cycle in picoseconds

#### 9.71.2.22 getSensorData()

```
virtual std::string TimeTaggerNetwork::getSensorData ( )  [pure virtual]
```

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

#### 9.71.2.23 getSerial()

```
virtual std::string TimeTaggerNetwork::getSerial ( )  [pure virtual]
```

identifies the hardware by serial number

#### 9.71.2.24 getStreamBlockSizeEvents()

```
virtual int TimeTaggerNetwork::getStreamBlockSizeEvents ( )  [pure virtual]
```

#### 9.71.2.25 getStreamBlockSizeLatency()

```
virtual int TimeTaggerNetwork::getStreamBlockSizeLatency ( )  [pure virtual]
```

#### 9.71.2.26 getTestSignal()

```
virtual bool TimeTaggerNetwork::getTestSignal (
             channel_t channel )  [pure virtual]
```

fetch the status of the test signal generator

**Parameters**

| | |
|---|---|
| *channel* | the channel |

Implements TimeTaggerBase.

#### 9.71.2.27 getTestSignalDivider()

```
virtual int TimeTaggerNetwork::getTestSignalDivider ( )  [pure virtual]
```

get the divider for the frequency of the test signal

#### 9.71.2.28 getTriggerLevel()

```
virtual double TimeTaggerNetwork::getTriggerLevel (
            channel_t channel ) [pure virtual]
```

get the trigger voltage threshold of a channel

**Parameters**

| | |
|---|---|
| *channel* | the channel |

#### 9.71.2.29 isConnected()

```
virtual bool TimeTaggerNetwork::isConnected ( ) [pure virtual]
```

check if the network time tagger is currently connected to a server

**Returns**

returns true if it's currently connected to a server; false, otherwise.

#### 9.71.2.30 setConditionalFilter()

```
virtual void TimeTaggerNetwork::setConditionalFilter (
            std::vector< channel_t > trigger,
            std::vector< channel_t > filtered,
            bool hardwareDelayCompensation = true ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *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 |

#### 9.71.2.31 setDelayClient()

```
virtual void TimeTaggerNetwork::setDelayClient (
            channel_t channel,
            timestamp_t time ) [pure virtual]
```

set time delay on a channel

When set, every event on this channel is delayed by the given delay in picoseconds.

This delay is implemented on the client and does not affect the server nor requires the Control flag.

**Parameters**

| | |
|---|---|
| *channel* | the channel to set |
| *time* | the delay in picoseconds |

#### 9.71.2.32 setEventDivider()

```
virtual void TimeTaggerNetwork::setEventDivider (
            channel_t channel,
            unsigned int divider )  [pure virtual]
```

Divides the amount of transmitted edge per channel.

This filter decimates the events on a given channel by a specified. factor. So for a divider n, every 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 at least 1 and smaller than 65536 |

#### 9.71.2.33 setHardwareBufferSize()

```
virtual void TimeTaggerNetwork::setHardwareBufferSize (
            int size )  [pure virtual]
```

sets the maximum USB buffer size

This option controls the maximum buffer size of the USB connection. This can be used to balance low input latency vs high (peak) throughput.

**Parameters**

| | |
|---|---|
| *size* | the maximum buffer size in events |

#### 9.71.2.34 setInputHysteresis()

```
virtual void TimeTaggerNetwork::setInputHysteresis (
```

```
            channel_t channel,
            int value )  [pure virtual]
```

configure the hysteresis voltage of the input comparator

Caution: This feature is only supported on the Time Tagger X The supported hysteresis voltages are 1 mV, 20 mV or 70 mV

**Parameters**

| | |
|---|---|
| *channel* | channel to be configured |
| *value* | the hysteresis voltage in milli Volt |

#### 9.71.2.35 setInputImpedanceHigh()

```
virtual void TimeTaggerNetwork::setInputImpedanceHigh (
            channel_t channel,
            bool high_impedance )  [pure virtual]
```

enable high impedance termination mode

Caution: This feature is only supported on the Time Tagger X

**Parameters**

| | |
|---|---|
| *channel* | channel to be configured |
| *high_impedance* | set for the high impedance mode or cleared for the 50 Ohm termination mode |

#### 9.71.2.36 setLED()

```
virtual void TimeTaggerNetwork::setLED (
            uint32_t bitmask )  [pure virtual]
```

Enforce a state to the LEDs 0: led_status[R] 16: led_status[R] - mux 1: led_status[G] 17: led_status[G] - mux 2: led_status[B] 18: led_status[B] - mux 3: led_power[R] 19: led_power[R] - mux 4: led_power[G] 20: led_power[G] - mux 5: led_power[B] 21: led_power[B] - mux 6: led_clock[R] 22: led_clock[R] - mux 7: led_clock[G] 23: led_clock[G] - mux 8: led_clock[B] 24: led_clock[B] - mux.

#### 9.71.2.37 setNormalization()

```
virtual void TimeTaggerNetwork::setNormalization (
            std::vector< channel_t > channels,
            bool state )  [pure virtual]
```

enables or disables the normalization of the distribution.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *channels* | list of channels to modify |
| *state* | the new state |

#### 9.71.2.38 setSoundFrequency()

```
virtual void TimeTaggerNetwork::setSoundFrequency (
            uint32_t freq_hz )  [pure virtual]
```

Set the Time Taggers internal buzzer to a frequency in Hz (freq_hz==0 to disable)

**Parameters**

| | |
|---|---|
| *freq_hz* | the generated audio frequency |

#### 9.71.2.39 setStreamBlockSize()

```
virtual void TimeTaggerNetwork::setStreamBlockSize (
            int max_events,
            int max_latency )  [pure virtual]
```

sets the maximum events and latency for the stream block size

This option controls the latency and the block size of the data stream. The default values are max_events = 131072 events and max_latency = 20 ms. Depending on which of the two parameters is exceeded first, the block stream size is adjusted accordingly. The block size will be reduced automatically for blocks when no signal is arriving for 512 ns on the Time Tagger Ultra and 1536 ns for the Time Tagger 20. ∗

**Parameters**

| | |
|---|---|
| *max_events* | maximum number of events |
| *max_latency* | maximum latency in ms |

#### 9.71.2.40 setTestSignalDivider()

```
virtual void TimeTaggerNetwork::setTestSignalDivider (
            int divider )  [pure virtual]
```

set the divider for the frequency of the test signal

The base clock of the test signal oscillator for the Time Tagger Ultra is running at 100.8 MHz sampled down by an factor of 2 to have a similar base clock as the Time Tagger 20 (∼50 MHz). The default divider is 63 -> ∼800 kEvents/s

**Parameters**

| | |
|---|---|
| *divider* | frequency divisor of the oscillator |

#### 9.71.2.41 setTimeTaggerNetworkStreamCompression()

```
virtual void TimeTaggerNetwork::setTimeTaggerNetworkStreamCompression (
            bool active )  [pure virtual]
```

enable or disable additional compression of the timetag stream as ent over the network.

**Parameters**

| | |
|---|---|
| *active* | set if the compressio is active or not. |

#### 9.71.2.42 setTriggerLevel()

```
virtual void TimeTaggerNetwork::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:

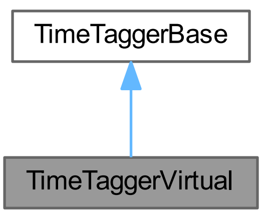
- TimeTagger.h

## 9.72 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*
- virtual std::vector< channel_t > getChannelList ()=0

  *Fetches channels from the input file.*

## Public Member Functions inherited from TimeTaggerBase

- virtual unsigned int getFence (bool alloc_fence=true)=0

  *Generate a new fence object, which validates the current configuration and the current time.*
- virtual bool waitForFence (unsigned int fence, int64_t timeout=-1)=0

  *Wait for a fence in the data stream.*
- virtual bool sync (int64_t timeout=-1)=0

  *Sync the timetagger pipeline, so that all started iterators and their enabled channels are ready.*
- virtual channel_t getInvertedChannel (channel_t channel)=0

  *get the falling channel id for a rising channel and vice versa*
- virtual bool isUnusedChannel (channel_t channel)=0

  *compares the provided channel with CHANNEL_UNUSED*
- virtual void runSynchronized (const IteratorCallbackMap &callbacks, bool block=true)=0

  *Run synchronized callbacks for a list of iterators.*
- virtual std::string getConfiguration ()=0

  *Fetches the overall configuration status of the Time Tagger object.*
- virtual void setInputDelay (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelayHardware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual void setDelaySoftware (channel_t channel, timestamp_t delay)=0

  *set time delay on a channel*
- virtual timestamp_t getInputDelay (channel_t channel)=0

  *get time delay of a channel*

- virtual timestamp_t getDelaySoftware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t getDelayHardware (channel_t channel)=0

  *get time delay of a channel*
- virtual timestamp_t setDeadtime (channel_t channel, timestamp_t deadtime)=0

  *set the deadtime between two edges on the same channel.*
- virtual timestamp_t getDeadtime (channel_t channel)=0

  *get the deadtime between two edges on the same channel.*
- virtual void setTestSignal (channel_t channel, bool enabled)=0

  *enable/disable internal test signal on a channel.*
- virtual void setTestSignal (std::vector< channel_t > channel, bool enabled)=0

  *enable/disable internal test signal on multiple channels.*
- virtual bool getTestSignal (channel_t channel)=0

  *fetch the status of the test signal generator*
- virtual void setSoftwareClock (channel_t input_channel, double input_frequency=10e6, double averaging_↵ periods=1000, bool wait_until_locked=true)=0

  *enables a software PLL to lock the time to an external clock*
- virtual void disableSoftwareClock ()=0

  *disabled the software PLL*
- virtual SoftwareClockState getSoftwareClockState ()=0

  *queries all state information of the software clock*
- virtual long long getOverflows ()=0

  *get overflow count*
- virtual void clearOverflows ()=0

  *clear overflow counter*
- virtual long long getOverflowsAndClear ()=0

  *get and clear overflow counter*

**Additional Inherited Members**

## Public Types inherited from TimeTaggerBase

- typedef std::function< void(IteratorBase *) IteratorCallback)
- typedef std::map< IteratorBase *, IteratorCallback > IteratorCallbackMap

## Protected Member Functions inherited from TimeTaggerBase

- TimeTaggerBase ()

  *abstract interface class*
- virtual ~TimeTaggerBase ()

  *destructor*
- TimeTaggerBase (const TimeTaggerBase &)=delete
- TimeTaggerBase & operator= (const TimeTaggerBase &)=delete
- virtual std::shared_ptr< IteratorBaseListNode > addIterator (IteratorBase *it)=0
- virtual void freeIterator (IteratorBase *it)=0
- virtual channel_t getNewVirtualChannel ()=0
- virtual void freeVirtualChannel (channel_t channel)=0
- virtual void registerChannel (channel_t channel)=0

  *register a FPGA channel.*
- virtual void registerChannel (std::set< channel_t > channels)=0

- virtual void unregisterChannel (channel_t channel)=0

    *release a previously registered channel.*
- virtual void unregisterChannel (std::set< channel_t > channels)=0
- virtual void addChild (TimeTaggerBase ∗child)=0
- virtual void removeChild (TimeTaggerBase ∗child)=0
- virtual void release ()=0

### 9.72.1 Detailed Description

virtual TimeTagger based on dump files

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

### 9.72.2 Member Function Documentation

#### 9.72.2.1 clearConditionalFilter()

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

deactivates the conditional filter

equivalent to setConditionalFilter({},{})

#### 9.72.2.2 getChannelList()

```
virtual std::vector< channel_t > TimeTaggerVirtual::getChannelList ( ) [pure virtual]
```

Fetches channels from the input file.

**Returns**

a vector of channels from the input file.

#### 9.72.2.3 getConditionalFilterFiltered()

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

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.72.2.4 getConditionalFilterTrigger()

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

fetches the configuration of the conditional filter

see setConditionalFilter

#### 9.72.2.5 getReplaySpeed()

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

fetches the speed factor

Please see setReplaySpeed for more details.

**Returns**

the speed factor

#### 9.72.2.6 replay()

```
virtual uint64_t TimeTaggerVirtual::replay (
            const std::string & file,
            timestamp_t begin = 0,
            timestamp_t duration = -1,
            bool queue = true )  [pure virtual]
```

replay a given dump file on the disc

This method adds the file to the replay queue. If the flag 'queue' is false, the current queue will be flushed and this file will be replayed immediately.

**Parameters**

| | |
|---|---|
| *file* | the file to be replayed, must be encoded as UTF-8 |
| *begin* | amount of ps to skip at the begin of the file. A negative 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

#### 9.72.2.7 reset()

```
virtual void TimeTaggerVirtual::reset ( )  [pure virtual]
```

stops the all queued files and resets the TimeTaggerVirtual to its default settings

This method stops the current file, clears the replay queue and resets the TimeTaggerVirtual to its default settings.

#### 9.72.2.8 setConditionalFilter()

```
virtual void TimeTaggerVirtual::setConditionalFilter (
            std::vector< channel_t > trigger,
            std::vector< channel_t > filtered ) [pure virtual]
```

configures the conditional filter

After each event on the trigger channels, one event per filtered channel will pass afterwards. This is handled in a very early stage in the pipeline, so all event limitations but the deadtime are suppressed. But the accuracy of the order of those events is low.

Refer the Manual for a description of this function.

**Parameters**

| | |
|---|---|
| *trigger* | the channels that sets the condition |
| *filtered* | the channels that are filtered by the condition |

#### 9.72.2.9 setReplaySpeed()

```
virtual void TimeTaggerVirtual::setReplaySpeed (
            double speed ) [pure virtual]
```

configures the speed factor for the virtual tagger.

This method configures the speed factor of this virtual Time Tagger. A value of 1.0 will replay in real time. All values $<$ 0.0 will replay the data as fast as possible, but stops at the end of all data. This is the default value.

**Parameters**

| | |
|---|---|
| *speed* | ratio of the replay speed and the real time |

#### 9.72.2.10 stop()

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

stops the current and all queued files.

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

#### 9.72.2.11 waitForCompletion()

```
virtual bool TimeTaggerVirtual::waitForCompletion (
            uint64_t ID = 0,
            int64_t timeout = -1 ) [pure virtual]
```

block the current thread until the replay finish

This method blocks the current execution and waits till the given file has finished its replay. If no ID is provided, it waits until all queued files are replayed.

This function does not block on a zero timeout. Negative timeouts are interpreted as infinite timeouts.

**Parameters**

| | |
|---|---|
| *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:

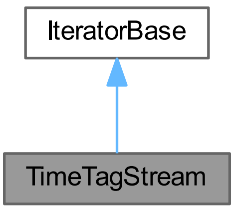
- TimeTagger.h

## 9.73 TimeTagStream Class Reference

access the time tag stream

`#include <Iterators.h>`

Inheritance diagram for TimeTagStream:




### Public Member Functions

- TimeTagStream (TimeTaggerBase ∗tagger, uint64_t n_max_events, std::vector< channel_t > channels)
  *constructor of a TimeTagStream thread*
- ∼TimeTagStream ()
- uint64_t getCounts ()
  *return the number of stored tags*
- TimeTagStreamBuffer getData ()
  *fetches all stored tags and clears the internal state*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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.*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.73.1 Detailed Description

access the time tag stream

## 9.73.2 Constructor & Destructor Documentation

### 9.73.2.1 TimeTagStream()

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

constructor of a TimeTagStream thread

Gives access to the time tag stream

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *n_max_events* | maximum number of tags stored |
| *channels* | channels which are dumped to the file |

#### 9.73.2.2 ∼TimeTagStream()

```
TimeTagStream::∼TimeTagStream ( )
```

### 9.73.3 Member Function Documentation

#### 9.73.3.1 clear_impl()

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

clear Iterator state.

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

Reimplemented from IteratorBase.

#### 9.73.3.2 getCounts()

```
uint64_t TimeTagStream::getCounts ( )
```

return the number of stored tags

#### 9.73.3.3 getData()

```
TimeTagStreamBuffer TimeTagStream::getData ( )
```

fetches all stored tags and clears the internal state

#### 9.73.3.4 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

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

- Iterators.h

## 9.74 TimeTagStreamBuffer Class Reference

return object for TimeTagStream::getData

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

### Public Member Functions

- ∼TimeTagStreamBuffer ()
- void getOverflows (std::function< unsigned char ∗(size_t)> array_out)
- void getChannels (std::function< channel_t ∗(size_t)> array_out)
- void getTimestamps (std::function< timestamp_t ∗(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

### 9.74.1 Detailed Description

return object for TimeTagStream::getData

### 9.74.2 Constructor & Destructor Documentation

#### 9.74.2.1 ∼TimeTagStreamBuffer()

```
TimeTagStreamBuffer::∼TimeTagStreamBuffer ( )
```

## 9.74.3 Member Function Documentation

### 9.74.3.1 getChannels()

```
void TimeTagStreamBuffer::getChannels (
            std::function< channel_t *(size_t)> array_out )
```

### 9.74.3.2 getEventTypes()

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

### 9.74.3.3 getMissedEvents()

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

### 9.74.3.4 getOverflows()

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

### 9.74.3.5 getTimestamps()

```
void TimeTagStreamBuffer::getTimestamps (
            std::function< timestamp_t *(size_t)> array_out )
```

## 9.74.4 Member Data Documentation

### 9.74.4.1 hasOverflows

```
bool TimeTagStreamBuffer::hasOverflows
```

### 9.74.4.2 size

```
uint64_t TimeTagStreamBuffer::size
```

### 9.74.4.3 tGetData

```
timestamp_t TimeTagStreamBuffer::tGetData
```

#### 9.74.4.4 tStart

```
timestamp_t TimeTagStreamBuffer::tStart
```

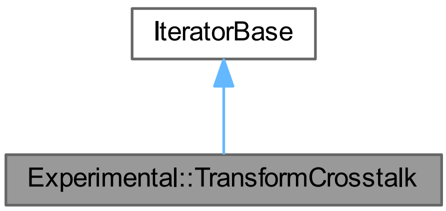
The documentation for this class was generated from the following file:

- Iterators.h

## 9.75 Experimental::TransformCrosstalk Class Reference

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

Inheritance diagram for Experimental::TransformCrosstalk:




### Public Member Functions

- TransformCrosstalk (TimeTaggerBase ∗tagger, channel_t input_channel, channel_t relay_input_channel, double delay, double tau, bool copy=false)

  *Construct a transformation that will apply crosstalk effect between an input channel and a relay channel.*
- ∼TransformCrosstalk ()
- channel_t getChannel ()

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

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

- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

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

  *update iterator state*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.75.1 Constructor & Destructor Documentation

### 9.75.1.1 TransformCrosstalk()

```
Experimental::TransformCrosstalk::TransformCrosstalk (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            channel_t relay_input_channel,
            double delay,
            double tau,
            bool copy = false )
```

Construct a transformation that will apply crosstalk effect between an input channel and a relay channel.

**Note**

this measurement is a transformation, it will modify the input channel unless its copy parameter is set to to true, in that case the modifications will be reflected on a virtual channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel to transform. |
| *relay_input_channel* | channel that causes the delays |
| *delay* | amount of delay triggered by relay channel. |
| *tau* | the decay after which an event of relay input channel has no effect anymore. |
| *copy* | tells if this transformation modifies the input or creates a new virtual channel with the transformation. |

#### 9.75.1.2 ∼TransformCrosstalk()

```
Experimental::TransformCrosstalk::~TransformCrosstalk ( )
```

### 9.75.2 Member Function Documentation

#### 9.75.2.1 getChannel()

```
channel_t Experimental::TransformCrosstalk::getChannel ( )
```

#### 9.75.2.2 next_impl()

```
bool Experimental::TransformCrosstalk::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.

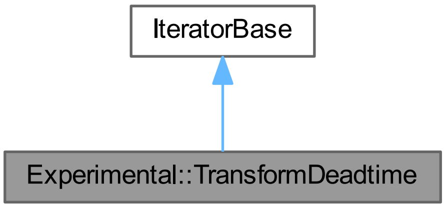
The documentation for this class was generated from the following file:

- Iterators.h

## 9.76 Experimental::TransformDeadtime Class Reference

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

Inheritance diagram for Experimental::TransformDeadtime:




### Public Member Functions

- TransformDeadtime (TimeTaggerBase *tagger, channel_t input_channel, double deadtime, bool copy=false)
  *Construct a transformation that will apply deadtime every event, filtering any events within the deadtime period.*
- ∼TransformDeadtime ()
- channel_t getChannel ()

## Public Member Functions inherited from IteratorBase

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

### Protected Member Functions

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

## Protected Member Functions inherited from IteratorBase

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

    *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

    *register a channel*
- void unregisterChannel (channel_t channel)

    *unregister a channel*
- channel_t getNewVirtualChannel ()

    *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

    *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

    *clear Iterator state.*
- virtual void on_start ()

    *callback when the measurement class is started*
- virtual void on_stop ()

    *callback when the measurement class is stopped*
- void lock ()

    *acquire update lock*
- void unlock ()

    *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

    *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

    *acquire update lock*
- void finish_running ()

    *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

    *list of channels used by the iterator*
- bool running

    *running state of the iterator*
- bool autostart

    *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

    *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

    *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

    *For internal use.*
- std::atomic< bool > aborting

## 9.76.1 Constructor & Destructor Documentation

### 9.76.1.1 TransformDeadtime()

```
Experimental::TransformDeadtime::TransformDeadtime (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double deadtime,
            bool copy = false )
```

Construct a transformation that will apply deadtime every event, filtering any events within the deadtime period.

**Note**

this measurement is a transformation, it will modify the input channel unless its copy parameter is set to to true, in that case the modifications will be reflected on a virtual channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel to transform. |
| *deadtime* | deadtime in seconds. |
| *copy* | tells if this transformation modifies the input or creates a new virtual channel with the transformation. |

### 9.76.1.2 ∼TransformDeadtime()

```
Experimental::TransformDeadtime::~TransformDeadtime ( )
```

## 9.76.2 Member Function Documentation

### 9.76.2.1 getChannel()

```
channel_t Experimental::TransformDeadtime::getChannel ( )
```

### 9.76.2.2 next_impl()

```
bool Experimental::TransformDeadtime::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.

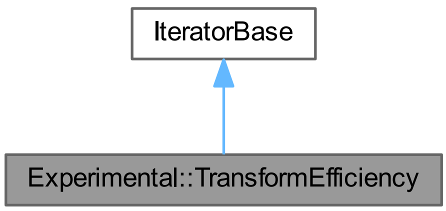
The documentation for this class was generated from the following file:

- Iterators.h

## 9.77 Experimental::TransformEfficiency Class Reference

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

Inheritance diagram for Experimental::TransformEfficiency:




### Public Member Functions

- TransformEfficiency (TimeTaggerBase *tagger, channel_t input_channel, double efficiency, bool copy=false, int32_t seed=-1)

  *Construct a transformation that will apply an efficiency filter to an specified channel. An efficiency filter will drop events based on an efficiency value. A perfect effcincy of 1.0 won't drop any events, an efficiency of 0.5 will drop half the events.*
- ∼TransformEfficiency ()
- channel_t getChannel ()

## Public Member Functions inherited from IteratorBase

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

### Protected Member Functions

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

## Protected Member Functions inherited from IteratorBase

- IteratorBase (TimeTaggerBase ∗tagger, std::string base_type_="IteratorBase", std::string extra_info_="")
  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)
  *register a channel*
- void unregisterChannel (channel_t channel)
  *unregister a channel*
- channel_t getNewVirtualChannel ()
  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()
  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()
  *clear Iterator state.*
- virtual void on_start ()
  *callback when the measurement class is started*
- virtual void on_stop ()
  *callback when the measurement class is stopped*
- void lock ()

*acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.77.1 Constructor & Destructor Documentation

### 9.77.1.1 TransformEfficiency()

```
Experimental::TransformEfficiency::TransformEfficiency (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double efficiency,
            bool copy = false,
            int32_t seed = -1 )
```

Construct a transformation that will apply an efficiency filter to an specified channel. An efficiency filter will drop events based on an efficiency value. A perfect effcincy of 1.0 won't drop any events, an efficiency of 0.5 will drop half the events.

**Note**

this measurement is a transformation, it will modify the input channel unless its copy parameter is set to to true, in that case the modifications will be reflected on a virtual channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel to be filtered. |
| *efficiency* | efficiency of the transformation. a 0.5 efficiency will drop half the events. A 1.0 won't drop any. |
| *copy* | tells if this transformation modifies the input or creates a new virtual channel with the transformation. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

#### 9.77.1.2 ∼TransformEfficiency()

```
Experimental::TransformEfficiency::~TransformEfficiency ( )
```

### 9.77.2 Member Function Documentation

#### 9.77.2.1 getChannel()

```
channel_t Experimental::TransformEfficiency::getChannel ( )
```

#### 9.77.2.2 next_impl()

```
bool Experimental::TransformEfficiency::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.

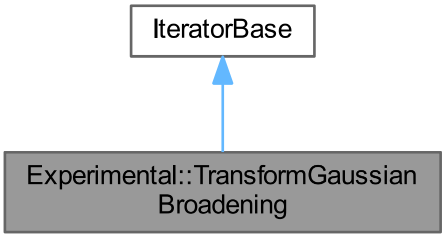
The documentation for this class was generated from the following file:

- Iterators.h

## 9.78 Experimental::TransformGaussianBroadening Class Reference

`#include <Iterators.h>`

Inheritance diagram for Experimental::TransformGaussianBroadening:




### Public Member Functions

- TransformGaussianBroadening (TimeTaggerBase *tagger, channel_t input_channel, double standard_↩ deviation, bool copy=false, int32_t seed=-1)

  *Construct a transformation that will apply gaussian brodening to each event in an specified channel.*
- ~TransformGaussianBroadening ()
- channel_t getChannel ()

### Public Member Functions inherited from IteratorBase

- virtual ~IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

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

  *update iterator state*

## Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- virtual void on_stop ()

  *callback when the measurement class is stopped*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)

  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()

  *acquire update lock*
- void finish_running ()

  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

- std::set< channel_t > channels_registered

  *list of channels used by the iterator*
- bool running

  *running state of the iterator*
- bool autostart

  *Condition if this measurement shall be started by the finishInitialization callback.*
- TimeTaggerBase * tagger

  *Pointer to the corresponding Time Tagger object.*
- timestamp_t capture_duration

  *Duration the iterator has already processed data.*
- timestamp_t pre_capture_duration

  *For internal use.*
- std::atomic< bool > aborting

## 9.78.1 Constructor & Destructor Documentation

### 9.78.1.1 TransformGaussianBroadening()

```
Experimental::TransformGaussianBroadening::TransformGaussianBroadening (
            TimeTaggerBase * tagger,
            channel_t input_channel,
            double standard_deviation,
            bool copy = false,
            int32_t seed = -1 )
```

Construct a transformation that will apply gaussian brodening to each event in an specified channel.

**Note**

this measurement is a transformation, it will modify the input channel unless its copy parameter is set to to true, in that case the modifications will be reflected on a virtual channel.

-2 broadening will be limited to 5 times the standard deviation.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *input_channel* | channel to be transformed. |
| *standard_deviation* | gaussian standard deviation which will affect the broadening |
| *copy* | tells if this transformation modifies the input or creates a new virtual channel with the transformation. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.78.1.2 ∼TransformGaussianBroadening()

```
Experimental::TransformGaussianBroadening::~TransformGaussianBroadening ( )
```

## 9.78.2 Member Function Documentation

### 9.78.2.1 getChannel()

```
channel_t Experimental::TransformGaussianBroadening::getChannel ( )
```

### 9.78.2.2 next_impl()

```
bool Experimental::TransformGaussianBroadening::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.

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

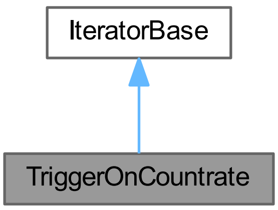
- Iterators.h

## 9.79 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.*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- 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*
- void clear_impl () override

  *clear Iterator state.*

## Protected Member Functions inherited from IteratorBase

- IteratorBase (TimeTaggerBase *tagger, std::string base_type_="IteratorBase", std::string extra_info_="")
  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)
  *register a channel*
- void unregisterChannel (channel_t channel)
  *unregister a channel*
- channel_t getNewVirtualChannel ()
  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()
  *method to call after finishing the initialization of the measurement*
- virtual void on_stop ()
  *callback when the measurement class is stopped*
- void lock ()
  *acquire update lock*
- void unlock ()
  *release update lock*
- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)
  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()
  *acquire update lock*
- void finish_running ()
  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template< typename T >
  void checkForAbort (T callback)

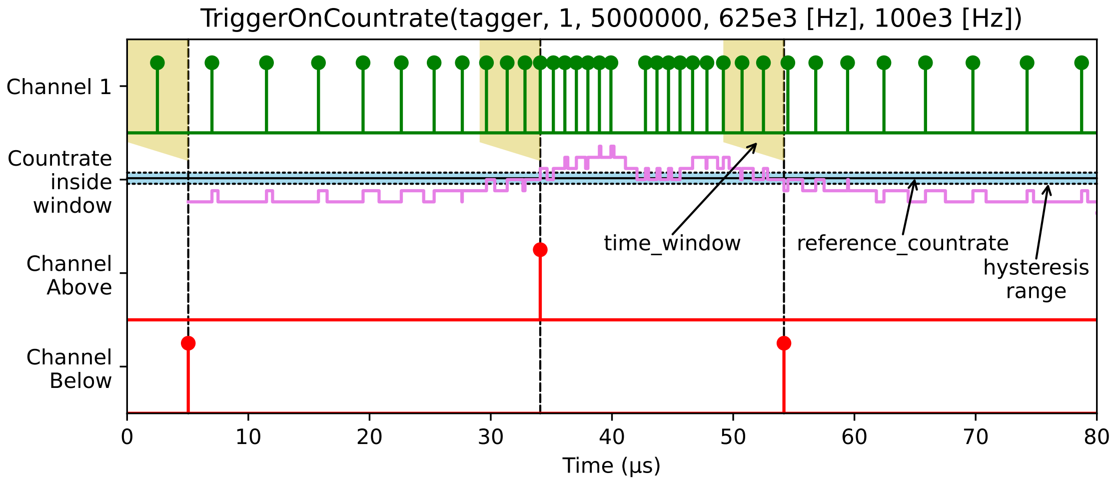
### Additional Inherited Members

## Protected Attributes inherited from IteratorBase

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

### 9.79.1 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`.

### 9.79.2 Constructor & Destructor Documentation

#### 9.79.2.1 TriggerOnCountrate()

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

constructor of a TriggerOnCountrate

**Parameters**

| | |
|---|---|
| *tagger* | Reference to a TimeTagger object. |
| *input_channel* | Channel number of the channel whose count rate will control the trigger channels. |

**Parameters**

| | |
|---|---|
| *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. |

#### 9.79.2.2 ∼TriggerOnCountrate()

```
TriggerOnCountrate::~TriggerOnCountrate ( )
```

### 9.79.3 Member Function Documentation

#### 9.79.3.1 clear_impl()

```
void TriggerOnCountrate::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.

#### 9.79.3.2 getChannelAbove()

```
channel_t TriggerOnCountrate::getChannelAbove ( )
```

Get the channel number of the `above` channel.

#### 9.79.3.3 getChannelBelow()

```
channel_t TriggerOnCountrate::getChannelBelow ( )
```

Get the channel number of the `below` channel.

#### 9.79.3.4 getChannels()

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

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

#### 9.79.3.5 getCurrentCountrate()

```
double TriggerOnCountrate::getCurrentCountrate ( )
```

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

#### 9.79.3.6 injectCurrentState()

```
bool TriggerOnCountrate::injectCurrentState ( )
```

Emit a time-tag into the respective channel according to the current state.

Emit a time-tag into the respective channel according to the current state. This is useful if you start a new measurement that requires the information. The function returns whether it was possible to inject the event. The injection is not possible if the Time Tagger is in overflow mode or the time window has not passed yet. The function call is non-blocking.

#### 9.79.3.7 isAbove()

```
bool TriggerOnCountrate::isAbove ( )
```

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

#### 9.79.3.8 isBelow()

```
bool TriggerOnCountrate::isBelow ( )
```

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

#### 9.79.3.9 next_impl()

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

update iterator state

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

The backend delivers each Tag on each registered channel to this callback function.

**Parameters**

| | |
|---|---|
| *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.

#### 9.79.3.10 on_start()

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

callback when the measurement class is started

This function is guarded by the update lock.

Reimplemented from IteratorBase.

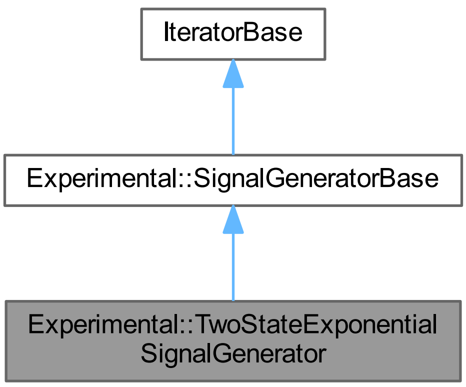
The documentation for this class was generated from the following file:

- Iterators.h

## 9.80 Experimental::TwoStateExponentialSignalGenerator Class Reference

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

Inheritance diagram for Experimental::TwoStateExponentialSignalGenerator:




### Public Member Functions

- TwoStateExponentialSignalGenerator (TimeTaggerBase *tagger, double excitation_time, double life_time, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

    *Construct a two-state exponential event channel.*
- ∼TwoStateExponentialSignalGenerator ()

## Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

## Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

  *After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

## Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

## Protected Member Functions inherited from IteratorBase

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

#### Additional Inherited Members

## Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

## Protected Attributes inherited from IteratorBase

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

## 9.80.1 Constructor & Destructor Documentation

### 9.80.1.1 TwoStateExponentialSignalGenerator()

```
Experimental::TwoStateExponentialSignalGenerator::TwoStateExponentialSignalGenerator (
            TimeTaggerBase * tagger,
            double excitation_time,
            double life_time,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a two-state exponential event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *excitation_time* | excitation time in seconds. |
| *life_time* | life time of the excited state in seconds |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created. |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

### 9.80.1.2 ∼TwoStateExponentialSignalGenerator()

```
Experimental::TwoStateExponentialSignalGenerator::~TwoStateExponentialSignalGenerator ( )
```

## 9.80.2 Member Function Documentation

### 9.80.2.1 get_next()

```
timestamp_t Experimental::TwoStateExponentialSignalGenerator::get_next ( ) [override], [protected],
[virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.80.2.2 initialize()

```
void Experimental::TwoStateExponentialSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

### 9.80.2.3 on_restart()

```
void Experimental::TwoStateExponentialSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

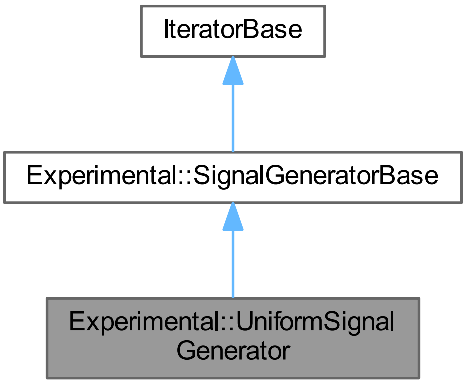
The documentation for this class was generated from the following file:

- Iterators.h

## 9.81 Experimental::UniformSignalGenerator Class Reference

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

Inheritance diagram for Experimental::UniformSignalGenerator:




### Public Member Functions

- UniformSignalGenerator (TimeTaggerBase ∗tagger, timestamp_t upper_bound, timestamp_t lower_bound=1, channel_t base_channel=CHANNEL_UNUSED, int32_t seed=-1)

  *Construct a random uniform event channel.*
- ∼UniformSignalGenerator ()

### Public Member Functions inherited from Experimental::SignalGeneratorBase

- SignalGeneratorBase (TimeTaggerBase ∗tagger, channel_t base_channel=CHANNEL_UNUSED)
- ∼SignalGeneratorBase ()
- channel_t getChannel ()

  *the new virtual channel*

### Public Member Functions inherited from IteratorBase

- virtual ∼IteratorBase ()

  *destructor, will unregister from the Time Tagger prior finalization.*
- void start ()

  *Starts or continues data acquisition.*
- void startFor (timestamp_t capture_duration, bool clear=true)

  *Starts or continues the data acquisition for the given duration.*
- bool waitUntilFinished (int64_t timeout=-1)

  *Blocks the execution until the measurement has finished. Can be used with startFor().*
- void stop ()

*After calling this method, the measurement will stop processing incoming tags.*
- void clear ()

  *Discards accumulated measurement data, initializes the data buffer with zero values, and resets the state to the initial state.*
- void abort ()

  *Immediately aborts the measurement, discarding accumulated measurement data, and resets the state to the initial state.*
- bool isRunning ()

  *Returns True if the measurement is collecting the data.*
- timestamp_t getCaptureDuration ()

  *Total capture duration since the measurement creation or last call to clear().*
- std::string getConfiguration ()

  *Fetches the overall configuration status of the measurement.*

#### Protected Member Functions

- void initialize (timestamp_t initial_time) override
- timestamp_t get_next () override
- void on_restart (timestamp_t restart_time) override

### Protected Member Functions inherited from Experimental::SignalGeneratorBase

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

  *update iterator state*
- void on_stop () override

  *callback when the measurement class is stopped*
- bool isProcessingFinished ()
- void set_processing_finished (bool is_finished)

### Protected Member Functions inherited from IteratorBase

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

  *Standard constructor, which will register with the Time Tagger backend.*
- void registerChannel (channel_t channel)

  *register a channel*
- void unregisterChannel (channel_t channel)

  *unregister a channel*
- channel_t getNewVirtualChannel ()

  *allocate a new virtual output channel for this iterator*
- void finishInitialization ()

  *method to call after finishing the initialization of the measurement*
- virtual void clear_impl ()

  *clear Iterator state.*
- virtual void on_start ()

  *callback when the measurement class is started*
- void lock ()

  *acquire update lock*
- void unlock ()

  *release update lock*

- OrderedBarrier::OrderInstance parallelize (OrderedPipeline &pipeline)
  *release lock and continue work in parallel*
- std::unique_lock< std::mutex > getLock ()
  *acquire update lock*
- void finish_running ()
  *Callback for the measurement to stop itself.*
- void checkForAbort ()
- template<typename T >
  void checkForAbort (T callback)

**Additional Inherited Members**

### Protected Attributes inherited from Experimental::SignalGeneratorBase

- std::unique_ptr< SignalGeneratorBaseImpl > impl

### Protected Attributes inherited from IteratorBase

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

## 9.81.1 Constructor & Destructor Documentation

### 9.81.1.1 UniformSignalGenerator()

```
Experimental::UniformSignalGenerator::UniformSignalGenerator (
            TimeTaggerBase * tagger,
            timestamp_t upper_bound,
            timestamp_t lower_bound = 1,
            channel_t base_channel = CHANNEL_UNUSED,
            int32_t seed = -1 )
```

Construct a random uniform event channel.

**Parameters**

| | |
|---|---|
| *tagger* | reference to a TimeTagger |
| *upper_bound* | Max possible offset of event generated compared to latest. |
| *lower_bound* | Min possible offset of event generated, must be higher than 0. |
| *base_channel* | base channel to which this signal will be added. If unused, a new channel will be created |
| *seed* | Seed number for the Pseudo-random number generator. Use -1 to use the current time as seed. |

#### 9.81.1.2 ∼UniformSignalGenerator()

```
Experimental::UniformSignalGenerator::~UniformSignalGenerator ( )
```

### 9.81.2 Member Function Documentation

#### 9.81.2.1 get_next()

```
timestamp_t Experimental::UniformSignalGenerator::get_next ( ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.81.2.2 initialize()

```
void Experimental::UniformSignalGenerator::initialize (
            timestamp_t initial_time ) [override], [protected], [virtual]
```

Implements Experimental::SignalGeneratorBase.

#### 9.81.2.3 on_restart()

```
void Experimental::UniformSignalGenerator::on_restart (
            timestamp_t restart_time ) [override], [protected], [virtual]
```

Reimplemented from Experimental::SignalGeneratorBase.

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

- Iterators.h

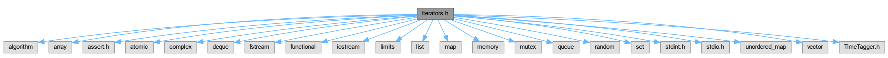
# Chapter 10

# File Documentation

## 10.1 Iterators.h File Reference

```
#include <algorithm>
#include <array>
#include <assert.h>
#include <atomic>
#include <complex>
#include <deque>
#include <fstream>
#include <functional>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <queue>
#include <random>
#include <set>
#include <stdint.h>
#include <stdio.h>
#include <unordered_map>
#include <vector>
#include "TimeTagger.h"
```

Include dependency graph for Iterators.h:




### Classes

- class FastBinning

  *Helper class for fast division with a constant divisor.*
- class Combiner

  *Combine some channels in a virtual channel which has a tick for each tick in the input channels.*
- class CountBetweenMarkers

*a simple counter where external marker signals determine the bins*

- class CounterData

  *Helper object as return value for Counter::getDataObject.*
- class Counter

  *a simple counter on one or more channels*
- class Coincidences

  *a coincidence monitor for many channel groups*
- class Coincidence

  *a coincidence monitor for one channel group*
- class Countrate

  *count rate on one or more channels*
- class DelayedChannel

  *a simple delayed queue*
- class TriggerOnCountrate

  *Inject trigger events when exceeding or falling below a given count rate within a rolling time window.*
- class GatedChannel

  *An input channel is gated by a gate channel.*
- class FrequencyMultiplier

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

  *a deprecated simple event queue*
- class TimeTagStreamBuffer

  *return object for TimeTagStream::getData*
- class TimeTagStream

  *access the time tag stream*
- class Dump

  *dump all time tags to a file*
- class StartStop

  *simple start-stop measurement*
- class TimeDifferences

  *Accumulates the time differences between clicks on two channels in one or more histograms.*
- class Histogram2D

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

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

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

  *Accumulate time differences into a histogram.*
- class FrequencyCounterData
- class FrequencyCounter

  *Calculate the phase of multiple channels at equidistant sampling points.*
- class HistogramLogBinsData

  *Helper object as return value for HistogramLogBins::getDataObject.*
- struct ChannelGate
- class HistogramLogBins

  *Accumulate time differences into a histogram with logarithmic increasing bin sizes.*
- class Correlation

  *Auto- and Cross-correlation measurement.*
- struct Event

  *Object for the return value of Scope::getData.*

- class Scope

  *a scope measurement*
- class SynchronizedMeasurements

  *start, stop and clear several measurements synchronized*
- class ConstantFractionDiscriminator

  *a virtual CFD implementation which returns the mean time between a rising and a falling pair of edges*
- class FileWriter

  *compresses and stores all time tags to a file*
- class FileReader

  *Reads tags from the disk files, which has been created by FileWriter.*
- class EventGenerator

  *Generate predefined events in a virtual channel relative to a trigger event.*
- class Combinations
- class CustomMeasurementBase

  *Helper class for custom measurements in Python and C#.*
- class FlimAbstract

  *Interface for FLIM measurements, Flim and FlimBase classes inherit from it.*
- class FlimBase

  *basic measurement, containing a minimal set of features for efficiency purposes*
- class FlimFrameInfo

  *object for storing the state of Flim::getCurrentFrameEx*
- class Flim

  *Fluorescence lifetime imaging.*
- class Sampler

  *a triggered sampling measurement*
- class SyntheticSingleTag

  *synthetic trigger timetag generator.*
- class FrequencyStabilityData

  *return data object for FrequencyStability::getData.*
- class FrequencyStability

  *Allan deviation (and related metrics) calculator.*
- class Experimental::PulsePerSecondData

  *Helper object as return value for PulsePerSecondMonitor::getDataObject.*
- class Experimental::PulsePerSecondMonitor

  *Monitors the synchronicity of 1 pulse per second (PPS) signals.*
- class Experimental::SignalGeneratorBase
- class Experimental::PhotonGenerator
- class Experimental::DlsSignalGenerator
- class Experimental::FcsSignalGenerator
- class Experimental::UniformSignalGenerator
- class Experimental::GaussianSignalGenerator
- class Experimental::OscillatorSimulation
- class Experimental::TwoStateExponentialSignalGenerator
- class Experimental::MarkovProcessGenerator
- class Experimental::ExponentialSignalGenerator
- class Experimental::GammaSignalGenerator
- class Experimental::PatternSignalGenerator
- class Experimental::SimSignalSplitter
- class Experimental::TransformEfficiency
- class Experimental::TransformGaussianBroadening
- class Experimental::TransformDeadtime
- class Experimental::TransformCrosstalk

- class Experimental::SimDetector
- class Experimental::SimLifetime
- class Experimental::PhotonNumber

    *Photon number resolution.*

### Namespaces

- namespace Experimental

    *Namespace for features, which are still in development and are likely to change.*

### Macros

- #define BINNING_TEMPLATE_HELPER(fun_name, binner, ...)

    *FastBinning caller helper.*

### Enumerations

- enum class CoincidenceTimestamp : uint32_t { Last = 0 , Average = 1 , First = 2 , ListedFirst = 3 }

    *type of timestamp for the Coincidence virtual channel (Last, Average, First, ListedFirst)*
- enum class GatedChannelInitial : uint32_t { Closed = 0 , Open = 1 }

    *Initial state of the gate of a GatedChannel (Closed, Open)*
- enum State { UNKNOWN , HIGH , LOW }

    *Input state in the return object of Scope.*

## 10.1.1 Macro Definition Documentation

### 10.1.1.1 BINNING_TEMPLATE_HELPER

```
#define BINNING_TEMPLATE_HELPER(
                fun_name,
                binner,
                ... )
```

**Value:**

```
switch (binner.getMode()) {
    \
case FastBinning::Mode::ConstZero:
    \
  fun_name<FastBinning::Mode::ConstZero>(__VA_ARGS__);
    \
  break;
    \
case FastBinning::Mode::Dividend:
    \
  fun_name<FastBinning::Mode::Dividend>(__VA_ARGS__);
    \
  break;
    \
case FastBinning::Mode::PowerOfTwo:
    \
  fun_name<FastBinning::Mode::PowerOfTwo>(__VA_ARGS__);
    \
  break;
    \
case FastBinning::Mode::FixedPoint_32:
    \
  fun_name<FastBinning::Mode::FixedPoint_32>(__VA_ARGS__);
    \
```

```
    break;
      \
  case FastBinning::Mode::FixedPoint_64:
      \
    fun_name<FastBinning::Mode::FixedPoint_64>(__VA_ARGS__);
      \
    break;
      \
  case FastBinning::Mode::Divide_32:
      \
    fun_name<FastBinning::Mode::Divide_32>(__VA_ARGS__);
      \
    break;
      \
  case FastBinning::Mode::Divide_64:
      \
    fun_name<FastBinning::Mode::Divide_64>(__VA_ARGS__);
      \
    break;
      \
  }
```

FastBinning caller helper.

## 10.1.2 Enumeration Type Documentation

### 10.1.2.1 CoincidenceTimestamp

```
enum class CoincidenceTimestamp : uint32_t [strong]
```

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

**Enumerator**

| | |
|---|---|
| Last | time of the last event completing the coincidence (fastest option - default) |
| Average | average time of all tags completing the coincidence |
| First | time of the first event received of the coincidence |
| ListedFirst | time of the first channel of the list with which the Coincidence was initialized |

### 10.1.2.2 GatedChannelInitial

```
enum class GatedChannelInitial : uint32_t [strong]
```

Initial state of the gate of a GatedChannel (Closed, Open)

**Enumerator**

| | |
|---|---|
| Closed | the gate is closed initially (default) |
| Open | the gate is open initially |

### 10.1.2.3 State

```
enum State
```

Input state in the return object of Scope.

#### Enumerator

| | |
|---|---|
| UNKNOWN | |
| HIGH | |
| LOW | |

## 10.2 Iterators.h

Go to the documentation of this file.

```
/*
This file is part of Time Tagger software defined digital data acquisition.

Copyright (C) 2011-2019 Swabian Instruments
All Rights Reserved

Unauthorized copying of this file is strictly prohibited.
*/

#ifndef TT_ITERATORS_H_
#define TT_ITERATORS_H_

#include <algorithm>
#include <array>
#include <assert.h>
#include <atomic>
#include <complex>
#include <deque>
#include <fstream>
#include <functional>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <queue>
#include <random>
#include <set>
#include <stdint.h>
#include <stdio.h>
#include <unordered_map>
#include <vector>

// Include mulh helpers on MSVC
#if !defined(__SIZEOF_INT128__) && (defined(_M_X64) || defined(_M_ARM64))
#include <intrin.h>
#endif

#include "TimeTagger.h"

class TT_API FastBinning {
public:
  enum class Mode {
    ConstZero,
    Dividend,
    PowerOfTwo,
    FixedPoint_32,
    FixedPoint_64,
    Divide_32,
    Divide_64,
  };

  FastBinning() {}

  FastBinning(uint64_t divisor, uint64_t max_duration_);

  template <Mode mode> uint64_t divide(uint64_t duration) const {
    assert(duration <= max_duration);
    assert(mode == this->mode);
    uint64_t out;
    switch (mode) {
    case Mode::ConstZero:
      out = 0;
      break;
    case Mode::Dividend:
      out = duration;
      break;
```

```
    case Mode::PowerOfTwo:
      out = duration » bits_shift;
      break;
    case Mode::FixedPoint_32:
      out = (duration * factor) » 32;
      break;
    case Mode::FixedPoint_64:
      out = MulHigh(duration, factor);
      break;
    case Mode::Divide_32:
      out = uint32_t(duration) / uint32_t(divisor);
      break;
    case Mode::Divide_64:
      out = duration / divisor;
      break;
    }
    assert(out == duration / divisor);
    return out;
  }

  Mode getMode() const { return mode; }

private:
  // returns (a*b) » 64 in a generic but accelerated way
  uint64_t MulHigh(uint64_t a, uint64_t b) const {
#ifdef __SIZEOF_INT128__
    return ((unsigned __int128)a * (unsigned __int128)b) » 64; // GCC, clang, ...
#elif defined(_M_X64) || defined(_M_ARM64)
    return __umulh(a, b); // MSVC
#else
    // Generic fallback
    uint64_t a_lo = uint32_t(a);
    uint64_t a_hi = a » 32;
    uint64_t b_lo = uint32_t(b);
    uint64_t b_hi = b » 32;

    uint64_t a_x_b_hi = a_hi * b_hi;
    uint64_t a_x_b_mid = a_hi * b_lo;
    uint64_t b_x_a_mid = b_hi * a_lo;
    uint64_t a_x_b_lo = a_lo * b_lo;

    uint64_t carry_bit = ((uint64_t)(uint32_t)a_x_b_mid + (uint64_t)(uint32_t)b_x_a_mid + (a_x_b_lo » 32)) » 32;

    uint64_t multhi = a_x_b_hi + (a_x_b_mid » 32) + (b_x_a_mid » 32) + carry_bit;

    return multhi;
#endif
  }

  uint64_t divisor;
  uint64_t max_duration;
  uint64_t factor;
  int bits_shift;
  Mode mode;
};

#define BINNING_TEMPLATE_HELPER(fun_name, binner, ...) \
  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; \
  }
class CombinerImpl;
class TT_API Combiner : public IteratorBase {
public:
  Combiner(TimeTaggerBase *tagger, std::vector<channel_t> channels);
  ~Combiner();
  GET_DATA_1D(getChannelCounts, int64_t, array_out, );
  GET_DATA_1D(getData, int64_t, array_out, );
  channel_t getChannel();
protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time)
override;
  void clear_impl() override;
private:
  friend class CombinerImpl;
  std::unique_ptr<CombinerImpl> impl;
};
class CountBetweenMarkersImpl;
class TT_API CountBetweenMarkers : public IteratorBase {
public:
  CountBetweenMarkers(TimeTaggerBase *tagger, channel_t click_channel, channel_t begin_channel,
                      channel_t end_channel = CHANNEL_UNUSED, int32_t n_values = 1000);
  ~CountBetweenMarkers();
  bool ready();
  GET_DATA_1D(getData, int32_t, array_out, );
  GET_DATA_1D(getBinWidths, timestamp_t, array_out, );
  GET_DATA_1D(getIndex, timestamp_t, array_out, );
protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time)
override;
  void clear_impl() override;
private:
  friend class CountBetweenMarkersImpl;
  std::unique_ptr<CountBetweenMarkersImpl> impl;
};
class CounterDataState;
class CounterImpl;
class Counter;
class TT_API CounterData {
public:
  ~CounterData();
  GET_DATA_2D(getData, int32_t, array_out, );
  GET_DATA_2D_OP1(getFrequency, double, array_out, timestamp_t, time_scale, 1000000000000, );
  GET_DATA_2D(getDataNormalized, double, array_out, );
  GET_DATA_1D(getDataTotalCounts, uint64_t, array_out, );
  GET_DATA_1D(getIndex, timestamp_t, array_out, );
  GET_DATA_1D(getTime, timestamp_t, array_out, );
  GET_DATA_1D(getOverflowMask, signed char, array_out, );
```

```
  GET_DATA_1D(getChannels, channel_t, array_out, );

  const uint32_t size;
  const uint32_t dropped_bins;
  const bool overflow;

private:
  friend class CounterImpl;
  friend class Counter;

  CounterData(uint32_t size_, uint32_t dropped_bins_, bool overflow_, std::shared_ptr<CounterDataState> data_);

  const std::shared_ptr<CounterDataState> data;
};

class TT_API Counter : public IteratorBase {
public:
  Counter(TimeTaggerBase *tagger, std::vector<channel_t> channels, timestamp_t binwidth = 1000000000,
          int32_t n_values = 1);

  ~Counter();

  GET_DATA_2D_OP1(getData, int32_t, array_out, bool, rolling, true, );

  GET_DATA_2D_OP1(getDataNormalized, double, array_out, bool, rolling, true, );

  GET_DATA_1D(getDataTotalCounts, uint64_t, array_out, );

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

  CounterData getDataObject(bool remove = false);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class CounterImpl;
  std::unique_ptr<CounterImpl> impl;
};

enum class CoincidenceTimestamp : uint32_t {
  Last = 0,
  Average = 1,
  First = 2,
  ListedFirst = 3,
};

class CoincidencesImpl;
class TT_API Coincidences : public IteratorBase {
public:
  Coincidences(TimeTaggerBase *tagger, std::vector<std::vector<channel_t» coincidenceGroups,
               timestamp_t coincidenceWindow, CoincidenceTimestamp timestamp = CoincidenceTimestamp::Last);

  ~Coincidences();

  std::vector<channel_t> getChannels();

  void setCoincidenceWindow(timestamp_t coincidenceWindow);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  friend class CoincidencesImpl;
  std::unique_ptr<CoincidencesImpl> impl;
};

class TT_API Coincidence : public Coincidences {
public:
  Coincidence(TimeTaggerBase *tagger, std::vector<channel_t> channels, timestamp_t coincidenceWindow = 1000,
              CoincidenceTimestamp timestamp = CoincidenceTimestamp::Last)
      : Coincidences(tagger, {channels}, coincidenceWindow, timestamp) {}

  channel_t getChannel() { return getChannels()[0]; }
};

class CountrateImpl;
class TT_API Countrate : public IteratorBase {
public:
```

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

  ~Countrate();

  GET_DATA_1D(getData, double, array_out, );

  GET_DATA_1D(getCountsTotal, int64_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class CountrateImpl;
  std::unique_ptr<CountrateImpl> impl;
};

class DelayedChannelImpl;
class TT_API DelayedChannel : public IteratorBase {
public:
  DelayedChannel(TimeTaggerBase *tagger, channel_t input_channel, timestamp_t delay);

#ifndef SWIG
  DelayedChannel(TimeTaggerBase *tagger, std::vector<channel_t> input_channels, timestamp_t delay);
#endif

  ~DelayedChannel();

  channel_t getChannel();

#ifndef SWIG
  std::vector<channel_t> getChannels();
#endif

  void setDelay(timestamp_t delay);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void on_start() override;

private:
  friend class DelayedChannelImpl;
  std::unique_ptr<DelayedChannelImpl> impl;
};

class TriggerOnCountrateImpl;
class TT_API TriggerOnCountrate : public IteratorBase {
public:
  TriggerOnCountrate(TimeTaggerBase *tagger, channel_t input_channel, double reference_countrate, double hysteresis,
                     timestamp_t time_window);

  ~TriggerOnCountrate();

  channel_t getChannelAbove();

  channel_t getChannelBelow();

  std::vector<channel_t> getChannels();

  bool isAbove();

  bool isBelow();

  double getCurrentCountrate();

  bool injectCurrentState();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void on_start() override;
  void clear_impl() override;

private:
  friend class TriggerOnCountrateImpl;
  std::unique_ptr<TriggerOnCountrateImpl> impl;
};

enum class GatedChannelInitial : uint32_t {
  Closed = 0,
  Open = 1,
};

```

```
class GatedChannelImpl;
class TT_API GatedChannel : public IteratorBase {
public:
  GatedChannel(TimeTaggerBase *tagger, channel_t input_channel, channel_t gate_start_channel,
               channel_t gate_stop_channel, GatedChannelInitial initial = GatedChannelInitial::Closed);

  ~GatedChannel();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class GatedChannelImpl;
  std::unique_ptr<GatedChannelImpl> impl;
};

class FrequencyMultiplierImpl;
class TT_API FrequencyMultiplier : public IteratorBase {
public:
  FrequencyMultiplier(TimeTaggerBase *tagger, channel_t input_channel, int32_t multiplier);

  ~FrequencyMultiplier();

  channel_t getChannel();
  int32_t getMultiplier();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class FrequencyMultiplierImpl;
  std::unique_ptr<FrequencyMultiplierImpl> impl;
};

class IteratorImpl;
class TT_API Iterator : public IteratorBase {
public:
  Iterator(TimeTaggerBase *tagger, channel_t channel);

  ~Iterator();

  timestamp_t next();

  uint64_t size();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class IteratorImpl;
  std::unique_ptr<IteratorImpl> impl;
};

class TimeTagStreamImpl;
class FileReaderImpl;
class TT_API TimeTagStreamBuffer {
  friend class TimeTagStreamImpl;
  friend class FileReaderImpl;

public:
  ~TimeTagStreamBuffer();

  GET_DATA_1D(getOverflows, unsigned char, array_out, ); // deprecated, please use getEventTypes instead
  GET_DATA_1D(getChannels, channel_t, array_out, );
  GET_DATA_1D(getTimestamps, timestamp_t, array_out, );
  GET_DATA_1D(getMissedEvents, unsigned short, array_out, );
  GET_DATA_1D(getEventTypes, unsigned char, array_out, );

  uint64_t size;
  bool hasOverflows;
  timestamp_t tStart;
  timestamp_t tGetData;

private:
  TimeTagStreamBuffer();

  std::vector<channel_t> tagChannels;
```

```
  std::vector<timestamp_t> tagTimestamps;
  std::vector<unsigned short> tagMissedEvents;
  std::vector<Tag::Type> tagTypes;
};

class TT_API TimeTagStream : public IteratorBase {
public:
  TimeTagStream(TimeTaggerBase *tagger, uint64_t n_max_events, std::vector<channel_t> channels);
  ~TimeTagStream();

  uint64_t getCounts();

  TimeTagStreamBuffer getData();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class TimeTagStreamImpl;
  std::unique_ptr<TimeTagStreamImpl> impl;
};

class DumpImpl;
class TT_API Dump : public IteratorBase {
public:
  Dump(TimeTaggerBase *tagger, std::string filename, int64_t max_tags,
       std::vector<channel_t> channels = std::vector<channel_t>());
  ~Dump();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;
  void on_stop() override;

private:
  friend class DumpImpl;
  std::unique_ptr<DumpImpl> impl;
};

class StartStopImpl;
class TT_API StartStop : public IteratorBase {
public:
  StartStop(TimeTaggerBase *tagger, channel_t click_channel, channel_t start_channel = CHANNEL_UNUSED,
            timestamp_t binwidth = 1000);

  ~StartStop();

  GET_DATA_2D(getData, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class StartStopImpl;
  std::unique_ptr<StartStopImpl> impl;
};

template <typename T> class TimeDifferencesImpl;
class TT_API TimeDifferences : public IteratorBase {
public:
  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);

  ~TimeDifferences();

  GET_DATA_2D(getData, int32_t, array_out, );

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

  void setMaxCounts(uint64_t max_counts);

  uint64_t getCounts();

  int32_t getHistogramIndex() const;

  bool ready();

protected:
```

```
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class TimeDifferencesImpl<TimeDifferences>;
  std::unique_ptr<TimeDifferencesImpl<TimeDifferences» impl;
};

template <typename T> class HistogramNDImpl;
class TT_API Histogram2D : public IteratorBase {
public:
  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);
  ~Histogram2D();

  GET_DATA_2D(getData, int32_t, array_out, );

  GET_DATA_3D(getIndex, timestamp_t, array_out, );

  GET_DATA_1D(getIndex_1, timestamp_t, array_out, );

  GET_DATA_1D(getIndex_2, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class HistogramNDImpl<Histogram2D>;
  std::unique_ptr<HistogramNDImpl<Histogram2D» impl;
};

class TT_API HistogramND : public IteratorBase {
public:
  HistogramND(TimeTaggerBase *tagger, channel_t start_channel, std::vector<channel_t> stop_channels,
              std::vector<timestamp_t> binwidths, std::vector<int32_t> n_bins);
  ~HistogramND();

  GET_DATA_1D(getData, int32_t, array_out, );

  GET_DATA_1D_OP1(getIndex, timestamp_t, array_out, int32_t, dim, 0, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class HistogramNDImpl<HistogramND>;
  std::unique_ptr<HistogramNDImpl<HistogramND» impl;
};

class TimeDifferencesNDImpl;
class TT_API TimeDifferencesND : public IteratorBase {
public:
  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);

  ~TimeDifferencesND();

  GET_DATA_2D(getData, int32_t, array_out, );

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class TimeDifferencesNDImpl;
  std::unique_ptr<TimeDifferencesNDImpl> impl;
};

class TT_API Histogram : public IteratorBase {
public:
  Histogram(TimeTaggerBase *tagger, channel_t click_channel, channel_t start_channel = CHANNEL_UNUSED,
            timestamp_t binwidth = 1000, int32_t n_bins = 1000);

  ~Histogram();

```

```
  GET_DATA_1D(getData, int32_t, array_out, );

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class TimeDifferencesImpl<Histogram>;
  std::unique_ptr<TimeDifferencesImpl<Histogram» impl;
};

struct FrequencyCounterDataImpl;

class TT_API FrequencyCounterData {
public:
  ~FrequencyCounterData();

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

  GET_DATA_1D(getTime, timestamp_t, array_out, );

  GET_DATA_2D(getOverflowMask, signed char, array_out, );

  GET_DATA_2D(getPeriodsCount, timestamp_t, array_out, );

  GET_DATA_2D(getPeriodsFraction, double, array_out, );

  GET_DATA_2D_OP1(getFrequency, double, array_out, timestamp_t, time_scale, 1000000000000, );

  GET_DATA_2D(getFrequencyInstantaneous, double, array_out, );

  GET_DATA_2D_OP1(getPhase, double, array_out, double, reference_frequency, 0, );

  const timestamp_t overflow_samples;

  const unsigned int size;

  const bool align_to_reference;

  const timestamp_t sampling_interval;

  const timestamp_t sample_offset;

  const bool channels_last_dim;

private:
  FrequencyCounterData(timestamp_t overflow_samples, unsigned int size, bool align_to_reference,
                       timestamp_t sampling_interval, timestamp_t index_offset, bool channels_last_dim);
  friend class FrequencyCounter;

  std::shared_ptr<FrequencyCounterDataImpl> data;
};

class FrequencyCounterImpl;

class TT_API FrequencyCounter : public IteratorBase {
public:
  FrequencyCounter(TimeTaggerBase *tagger, std::vector<channel_t> channels, timestamp_t sampling_interval,
                   timestamp_t fitting_window, int32_t n_values = 0);
  ~FrequencyCounter();
  FrequencyCounterData getDataObject(uint16_t event_divider = 1, bool remove = false, bool channels_last_dim = false);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class FrequencyCounterImpl;
  std::unique_ptr<FrequencyCounterImpl> impl;
};

class HistogramLogBinsImpl;

struct HistogramLogBinsDataImpl;

class TT_API HistogramLogBinsData {
public:
  ~HistogramLogBinsData();
```

```
  GET_DATA_1D(getCounts, uint64_t, array_out, );

  GET_DATA_1D(getG2Normalization, double, array_out, );

  GET_DATA_1D(getG2, double, array_out, );

  const timestamp_t accumulation_time_start;
  const timestamp_t accumulation_time_click;

private:
  HistogramLogBinsData(timestamp_t accumulation_time_start, timestamp_t accumulation_time_click);
  friend HistogramLogBinsImpl;

  std::shared_ptr<HistogramLogBinsDataImpl> data;
};

struct TT_API ChannelGate {
  ChannelGate(channel_t gate_open_channel, channel_t gate_close_channel,
              GatedChannelInitial initial = GatedChannelInitial::Open)
      : gate_open_channel(gate_open_channel), gate_close_channel(gate_close_channel), initial(initial){};
  const channel_t gate_open_channel;
  const channel_t gate_close_channel;
  const GatedChannelInitial initial;
};

class TT_API HistogramLogBins : public IteratorBase {
public:
  HistogramLogBins(TimeTaggerBase *tagger, channel_t click_channel, channel_t start_channel, double exp_start,
                   double exp_stop, int32_t n_bins, const ChannelGate *click_gate = nullptr,
                   const ChannelGate *start_gate = nullptr);
  ~HistogramLogBins();

  HistogramLogBinsData getDataObject();

  GET_DATA_1D(getData, uint64_t, array_out, );

  GET_DATA_1D(getDataNormalizedCountsPerPs, double, array_out, );

  GET_DATA_1D(getDataNormalizedG2, double, array_out, );

  GET_DATA_1D(getBinEdges, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class HistogramLogBinsImpl;
  std::unique_ptr<HistogramLogBinsImpl> impl;
};

class CorrelationImpl;
class TT_API Correlation : public IteratorBase {
public:
  Correlation(TimeTaggerBase *tagger, channel_t channel_1, channel_t channel_2 = CHANNEL_UNUSED,
              timestamp_t binwidth = 1000, int n_bins = 1000);

  ~Correlation();

  GET_DATA_1D(getData, int32_t, array_out, );

  GET_DATA_1D(getDataNormalized, double, array_out, );

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class CorrelationImpl;
  std::unique_ptr<CorrelationImpl> impl;
};

enum State {
  UNKNOWN,
  HIGH,
  LOW,
};
struct Event {
  timestamp_t time;
  State state;
```

```
};
class ScopeImpl;
class TT_API Scope : public IteratorBase {
public:
  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);

  ~Scope();

  bool ready();

  int32_t triggered();

  std::vector<std::vector<Event>> getData();

  timestamp_t getWindowSize();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class ScopeImpl;
  std::unique_ptr<ScopeImpl> impl;
};

class TimeTaggerProxy;
class TT_API SynchronizedMeasurements {
public:
  SynchronizedMeasurements(TimeTaggerBase *tagger);

  ~SynchronizedMeasurements();

  void registerMeasurement(IteratorBase *measurement);

  void unregisterMeasurement(IteratorBase *measurement);

  void clear();

  void start();

  void stop();

  void startFor(timestamp_t capture_duration, bool clear = true);

  bool waitUntilFinished(int64_t timeout = -1);

  bool isRunning();

  TimeTaggerBase *getTagger();

protected:
  void runCallback(TimeTaggerBase::IteratorCallback callback, std::unique_lock<std::mutex> &lk, bool block = true);

private:
  friend class TimeTaggerProxy;

  void release();

  std::set<IteratorBase *> registered_measurements;
  std::mutex measurements_mutex;
  TimeTaggerBase *tagger;
  bool has_been_released = false;
  std::unique_ptr<TimeTaggerProxy> proxy;
};

class ConstantFractionDiscriminatorImpl;
class TT_API ConstantFractionDiscriminator : public IteratorBase {
public:
  ConstantFractionDiscriminator(TimeTaggerBase *tagger, std::vector<channel_t> channels, timestamp_t search_window);

  ~ConstantFractionDiscriminator();

  std::vector<channel_t> getChannels();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void on_start() override;

private:
  friend class ConstantFractionDiscriminatorImpl;
  std::unique_ptr<ConstantFractionDiscriminatorImpl> impl;
};
```

```cpp

class FileWriterImpl;
class TT_API FileWriter : public IteratorBase {
public:
  FileWriter(TimeTaggerBase *tagger, const std::string &filename, std::vector<channel_t> channels);
  ~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:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time)
override;
  void clear_impl() override;
  void on_start() override;
  void on_stop() override;

private:
  friend class FileWriterImpl;
  std::unique_ptr<FileWriterImpl> impl;
};

class TT_API FileReader {
public:
  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::vector<channel_t> getChannelList();

  std::string getLastMarker();

private:
  friend class FileReaderImpl;
  std::unique_ptr<FileReaderImpl> impl;
};

class EventGeneratorImpl;
class TT_API EventGenerator : public IteratorBase {
public:
  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);

  ~EventGenerator();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time)
override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class EventGeneratorImpl;
  std::unique_ptr<EventGeneratorImpl> impl;
};

class CombinationsImpl;
class TT_API Combinations : public IteratorBase {
public:
  Combinations(TimeTaggerBase *tagger, std::vector<channel_t> const &channels, timestamp_t
window_size);

  ~Combinations();

  channel_t getChannel(std::vector<channel_t> const &input_channels) const;

```

```
  channel_t getSumChannel(int n_channels) const;

  std::vector<channel_t> getCombination(channel_t virtual_channel) const;

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class CombinationsImpl;
  std::unique_ptr<CombinationsImpl> impl;
};

class TT_API CustomMeasurementBase : public IteratorBase {
protected:
  // Only usable for subclasses.
  CustomMeasurementBase(TimeTaggerBase *tagger);

public:
  ~CustomMeasurementBase() override;

  // Stop all running custom measurements. Use this to avoid races on shutdown the target language.
  static void stop_all_custom_measurements();

  // Forward the public API of the measurement
  void register_channel(channel_t channel);
  void unregister_channel(channel_t channel);
  void finalize_init();
  bool is_running() const;
  void _lock();
  void _unlock();

protected:
  // By default, this calls next_impl_cs
  virtual bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

  // Handler with easier to wrap API. By default, this does nothing
  virtual void next_impl_cs(void *tags_ptr, uint64_t num_tags, timestamp_t begin_time, timestamp_t end_time);

  // Forward the public handlers for swig to detect this virtual methods. By default, they do nothing
  virtual void clear_impl() override;
  virtual void on_start() override;
  virtual void on_stop() override;
};

class TT_API FlimAbstract : public IteratorBase {
public:
  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);

  ~FlimAbstract();

  bool isAcquiring() const { return acquiring; }

protected:
  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;
  void clear_impl() override;
  void on_start() override;

  virtual void on_frame_end() = 0;

  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;

  // state
```

```
  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;
};

class TT_API FlimBase : public FlimAbstract {
public:
  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);

  ~FlimBase();

  void initialize();

protected:
  void on_frame_end() override final;

  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);

  uint32_t total_frames;
};

class TT_API FlimFrameInfo {
  friend class Flim;

public:
  ~FlimFrameInfo();

  int32_t getFrameNumber() const { return frame_number; }

  bool isValid() const { return valid; }

  uint32_t getPixelPosition() const { return pixel_position; }

  GET_DATA_2D(getHistograms, uint32_t, array_out, );
  GET_DATA_1D(getIntensities, float, array_out, );
  GET_DATA_1D(getSummedCounts, uint64_t, array_out, );
  GET_DATA_1D(getPixelBegins, timestamp_t, array_out, );
  GET_DATA_1D(getPixelEnds, timestamp_t, array_out, );

private:
  FlimFrameInfo();
  std::vector<uint32_t> histograms;
  std::vector<timestamp_t> pixel_begins;
  std::vector<timestamp_t> pixel_ends;

public:
  uint32_t pixels;
  uint32_t bins;
  int32_t frame_number;
  uint32_t pixel_position;
  bool valid;
};

class TT_API Flim : public FlimAbstract {
public:
  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);

  ~Flim();
```

```
  void initialize();

  GET_DATA_2D_OP1(getReadyFrame, uint32_t, array_out, int32_t, index, -1, );

  GET_DATA_1D_OP1(getReadyFrameIntensity, float, array_out, int32_t, index, -1, );

  GET_DATA_2D(getCurrentFrame, uint32_t, array_out, );

  GET_DATA_1D(getCurrentFrameIntensity, float, array_out, );

  GET_DATA_2D_OP2(getSummedFrames, uint32_t, array_out, bool, only_ready_frames, true, bool, clear_summed, false, );

  GET_DATA_1D_OP2(getSummedFramesIntensity, float, array_out, bool, only_ready_frames, true, bool, clear_summed,
                  false, );

  FlimFrameInfo getReadyFrameEx(int32_t index = -1);

  FlimFrameInfo getCurrentFrameEx();

  FlimFrameInfo getSummedFramesEx(bool only_ready_frames = true, bool clear_summed = false);

  uint32_t getFramesAcquired() const { return total_frames; }

  GET_DATA_1D(getIndex, timestamp_t, array_out, );

protected:
  void on_frame_end() override final;
  void clear_impl() override;

  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);

  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;
};

class SamplerImpl;
class TT_API Sampler : public IteratorBase {
public:
  Sampler(TimeTaggerBase *tagger, channel_t trigger, std::vector<channel_t> channels, size_t max_triggers);
  ~Sampler();

  GET_DATA_2D(getData, timestamp_t, array_out, );

  GET_DATA_2D(getDataAsMask, timestamp_t, array_out, );

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class SamplerImpl;
  std::unique_ptr<SamplerImpl> impl;
};

class SyntheticSingleTagImpl;
class TT_API SyntheticSingleTag : public IteratorBase {
public:
  SyntheticSingleTag(TimeTaggerBase *tagger, channel_t base_channel = CHANNEL_UNUSED);
  ~SyntheticSingleTag();

  void trigger();

  channel_t getChannel() const;

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
```

```

private:
  friend class SyntheticSingleTagImpl;
  std::unique_ptr<SyntheticSingleTagImpl> impl;
};

class FrequencyStabilityImpl;
struct FrequencyStabilityDataImpl;
class FrequencyStability;

class TT_API FrequencyStabilityData {
public:
  ~FrequencyStabilityData();

  GET_DATA_1D(getSTDD, double, array_out, );

  GET_DATA_1D(getADEV, double, array_out, );

  GET_DATA_1D(getMDEV, double, array_out, );

  GET_DATA_1D(getTDEV, double, array_out, );

  GET_DATA_1D(getHDEV, double, array_out, );

  GET_DATA_1D(getADEVScaled, double, array_out, );

  GET_DATA_1D(getHDEVScaled, double, array_out, );

  GET_DATA_1D(getTau, double, array_out, );

  GET_DATA_1D(getTracePhase, double, array_out, );

  GET_DATA_1D(getTraceFrequency, double, array_out, );

  GET_DATA_1D_OP1(getTraceFrequencyAbsolute, double, array_out, double, input_frequency, 0.0, );

  GET_DATA_1D(getTraceIndex, double, array_out, );

private:
  FrequencyStabilityData();
  friend class FrequencyStability;

  std::shared_ptr<FrequencyStabilityDataImpl> data;
};

class TT_API FrequencyStability : public IteratorBase {
public:
  FrequencyStability(TimeTaggerBase *tagger, channel_t channel, std::vector<uint64_t> steps, timestamp_t average = 1000,
                     uint64_t trace_len = 1000);
  ~FrequencyStability();

  FrequencyStabilityData getDataObject();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class FrequencyStabilityImpl;
  std::unique_ptr<FrequencyStabilityImpl> impl;
};

class PRBS;

namespace Experimental {

class PulsePerSecondImpl;
class PulsePerSecondDataState;

class TT_API PulsePerSecondData {
public:
  GET_DATA_1D(getIndices, int64_t, array_out, );
  GET_DATA_1D(getReferenceOffsets, double, array_out, );
  GET_DATA_2D(getSignalOffsets, double, array_out, );
  GET_DATA_1D(getUtcSeconds, double, array_out, );
  std::vector<std::string> getUtcDates();
  GET_DATA_1D(getStatus, bool, array_out, );
  const size_t size;
  ~PulsePerSecondData();

private:
  PulsePerSecondData(std::shared_ptr<PulsePerSecondDataState> data_ptr, const std::vector<channel_t> channel_list,
                     size_t size);
```

```
  std::shared_ptr<PulsePerSecondDataState> data;
  const std::vector<channel_t> channel_list;
  friend PulsePerSecondDataState;
  friend PulsePerSecondImpl;
};

class TT_API PulsePerSecondMonitor : public IteratorBase {
public:
  PulsePerSecondMonitor(TimeTaggerBase *tagger, channel_t reference_channel, std::vector<channel_t> signal_channels,
                        std::string filename = "", timestamp_t period = 1E12);
  ~PulsePerSecondMonitor();
  PulsePerSecondData getDataObject(bool remove = false);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;
  void on_start() override;

private:
  friend class PulsePerSecondImpl;
  std::unique_ptr<PulsePerSecondImpl> impl;
};

class SignalGeneratorBaseImpl;

class TT_API SignalGeneratorBase : public IteratorBase {
public:
  SignalGeneratorBase(TimeTaggerBase *tagger, channel_t base_channel = CHANNEL_UNUSED);
  ~SignalGeneratorBase();

  channel_t getChannel();

  // void registerReactor(std::string property, channel_t trigger_channel, std::vector<float> values, bool repeat);

protected:
  virtual void initialize(timestamp_t initial_time) = 0;
  virtual timestamp_t get_next() = 0;

  // void addReactable(std::string property, std::function<void, float> &&callback);

  virtual void on_restart(timestamp_t restart_time);

  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void on_stop() override;

  // callbacks
  bool isProcessingFinished();
  void set_processing_finished(bool is_finished);

  friend class SignalGeneratorBaseImpl;
  std::unique_ptr<SignalGeneratorBaseImpl> impl;
};

class TT_API PhotonGenerator : public SignalGeneratorBase {
public:
  PhotonGenerator(TimeTaggerBase *tagger, double countrate, channel_t base_channel, int32_t seed = -1);
  ~PhotonGenerator();
  void finalize_init();
  void set_T_PERIOD(timestamp_t new_T);
  timestamp_t get_T_PERIOD();

protected:
  void initialize(timestamp_t initial_time) override;
  void on_restart(timestamp_t restart_time) override;
  timestamp_t get_next() override;
  virtual double get_intensity() = 0;
  timestamp_t T_PERIOD;

private:
  timestamp_t get_new_stamp();
  std::minstd_rand0 generator;
  std::exponential_distribution<double> exp_distribution;
  timestamp_t accumulated;
  timestamp_t base_time;
  timestamp_t t_evolution;
  double current_intensity;
};

class DlsSignalGeneratorImpl;

class TT_API DlsSignalGenerator : public PhotonGenerator {
public:
```

```
  DlsSignalGenerator(TimeTaggerBase *tagger, double decay_time, double countrate,
                     channel_t output_channel = CHANNEL_UNUSED, int32_t seed = -1);
  DlsSignalGenerator(TimeTaggerBase *tagger, std::vector<double> decay_times, double countrate,
                     channel_t output_channel = CHANNEL_UNUSED, int32_t seed = -1);
  ~DlsSignalGenerator();
  unsigned int get_N();

protected:
  double get_intensity() override;

private:
  friend class DlsSignalGeneratorImpl;
  std::unique_ptr<DlsSignalGeneratorImpl> impl;
};

class FcsSignalGeneratorImpl;

class TT_API FcsSignalGenerator : public PhotonGenerator {
public:
  FcsSignalGenerator(TimeTaggerBase *tagger, double correlation_time, double N_focus, double countrate,
                     channel_t output_channel = CHANNEL_UNUSED, int32_t seed = -1);
  ~FcsSignalGenerator();
  unsigned int get_N();
  void set_boundary_limit(double new_boundary);

protected:
  double get_intensity() override;

private:
  friend class FcsSignalGeneratorImpl;
  std::unique_ptr<FcsSignalGeneratorImpl> impl;
};

class TT_API UniformSignalGenerator : public SignalGeneratorBase {
public:
  UniformSignalGenerator(TimeTaggerBase *tagger, timestamp_t upper_bound, timestamp_t lower_bound = 1,
                         channel_t base_channel = CHANNEL_UNUSED, int32_t seed = -1);
  ~UniformSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::unique_ptr<PRBS> generator;
  timestamp_t lower_bound;
  timestamp_t period;
  timestamp_t accumulated;
  timestamp_t base_time;
};

class TT_API GaussianSignalGenerator : public SignalGeneratorBase {
public:
  GaussianSignalGenerator(TimeTaggerBase *tagger, double mean, double standard_deviation,
                          channel_t base_channel = CHANNEL_UNUSED, int32_t seed = -1);
  ~GaussianSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::minstd_rand0 generator;
  std::normal_distribution<double> distr;
  timestamp_t accumulated;
  timestamp_t base_time;
};

class FlickerDistributionVossMcCartney;

class TT_API OscillatorSimulation : public SignalGeneratorBase {
public:
  OscillatorSimulation(TimeTaggerBase *tagger, double nominal_frequency, double coeff_phase_white = 0.0,
                       double coeff_phase_flicker = 0.0, double coeff_freq_white = 0.0, double coeff_freq_flicker = 0.0,
                       double coeff_random_drift = 0.0, double coeff_linear_drift = 0.0,
                       channel_t base_channel = CHANNEL_UNUSED, int32_t seed = -1);

  ~OscillatorSimulation();

protected:
```

```
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  double const coeff_phase_white, coeff_phase_flicker, coeff_freq_white, coeff_freq_flicker, coeff_random_drift,
      coeff_linear_drift;

  timestamp_t const nominal_period_int;
  double const nominal_period_fractional;

  std::unique_ptr<FlickerDistributionVossMcCartney> flicker_phase, flicker_freq;
  std::normal_distribution<double> white;
  std::mt19937_64 generator;
  double freq_random_walk_acc,
      fractional_ps_acc;
  timestamp_t last_time;
  uint64_t num_periods_passed{};
};

class TT_API TwoStateExponentialSignalGenerator : public SignalGeneratorBase {
public:
  TwoStateExponentialSignalGenerator(TimeTaggerBase *tagger, double excitation_time, double life_time,
                                     channel_t base_channel = CHANNEL_UNUSED, int32_t seed = -1);
  ~TwoStateExponentialSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::minstd_rand0 generator;
  std::exponential_distribution<double> excitation_time_distr;
  std::exponential_distribution<double> life_time_distr;
  timestamp_t accumulated;
  timestamp_t base_time;
};

class MarkovProcessGeneratorImpl;
class TT_API MarkovProcessGenerator : public IteratorBase {
public:
  MarkovProcessGenerator(TimeTaggerBase *tagger, uint64_t num_states, std::vector<double> frequencies,
                         std::vector<channel_t> ref_channels,
                         std::vector<channel_t> base_channels = std::vector<channel_t>(), int32_t seed = -1);
  ~MarkovProcessGenerator();

  channel_t getChannel();
  std::vector<channel_t> getChannels();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void on_stop() override;

private:
  friend class MarkovProcessGeneratorImpl;
  std::unique_ptr<MarkovProcessGeneratorImpl> impl;
};

class TT_API ExponentialSignalGenerator : public SignalGeneratorBase {
public:
  ExponentialSignalGenerator(TimeTaggerBase *tagger, double rate, channel_t base_channel = CHANNEL_UNUSED,
                             int32_t seed = -1);
  ~ExponentialSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::minstd_rand0 generator;
  std::exponential_distribution<double> distr;
  timestamp_t accumulated;
  timestamp_t base_time;
};

class TT_API GammaSignalGenerator : public SignalGeneratorBase {
public:
  GammaSignalGenerator(TimeTaggerBase *tagger, double alpha, double beta, channel_t base_channel =
```

```
                    CHANNEL_UNUSED,
                        int32_t seed = -1);
  ~GammaSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::minstd_rand0 generator;
  std::gamma_distribution<double> distr;
  timestamp_t accumulated;
  timestamp_t base_time;
};

class TT_API PatternSignalGenerator : public SignalGeneratorBase {
public:
  PatternSignalGenerator(TimeTaggerBase *tagger, std::vector<timestamp_t> sequence, bool repeat = false,
                         timestamp_t start_delay = 0, timestamp_t spacing = 0, channel_t base_channel = CHANNEL_UNUSED);
  ~PatternSignalGenerator();

protected:
  void initialize(timestamp_t initial_time) override;
  timestamp_t get_next() override;

  void on_restart(timestamp_t restart_time) override;

private:
  std::vector<timestamp_t> sequence;
  bool repeat;
  int64_t index;
  timestamp_t base_time;
  timestamp_t accumulated;
  timestamp_t start_delay;
  timestamp_t spacing;
};

class SimSignalSplitterImpl;
class TT_API SimSignalSplitter : public IteratorBase {
public:
  SimSignalSplitter(TimeTaggerBase *tagger, channel_t input_channel, double ratio = 0.5, int32_t seed = -1);

  ~SimSignalSplitter();

  std::vector<channel_t> getChannels();
  channel_t getLeftChannel();
  channel_t getRightChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  friend class SimSignalSplitterImpl;
  std::unique_ptr<SimSignalSplitterImpl> impl;
};

class TT_API TransformEfficiency : public IteratorBase {
public:
  TransformEfficiency(TimeTaggerBase *tagger, channel_t input_channel, double efficiency, bool copy = false,
                      int32_t seed = -1);

  ~TransformEfficiency();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  std::vector<Tag> mirror;

  const channel_t input_channel;
  const channel_t output_channel;

  const uint32_t limit;
  std::unique_ptr<PRBS> generator;
};

class TT_API TransformGaussianBroadening : public IteratorBase {
```

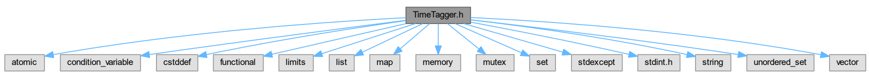
```
public:
  TransformGaussianBroadening(TimeTaggerBase *tagger, channel_t input_channel, double standard_deviation,
                              bool copy = false, int32_t seed = -1);

  ~TransformGaussianBroadening();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  std::vector<Tag> mirror;

  const channel_t input_channel;
  const channel_t output_channel;

  std::minstd_rand0 generator;
  std::normal_distribution<double> distr;

  bool overflow_state_on{};

  std::vector<Tag> accumulated_tags;
  timestamp_t delay{};
  std::deque<Tag> delayed_tags;
};

class TT_API TransformDeadtime : public IteratorBase {
public:
  TransformDeadtime(TimeTaggerBase *tagger, channel_t input_channel, double deadtime, bool copy = false);

  ~TransformDeadtime();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  std::vector<Tag> mirror;

  const channel_t input_channel;
  const channel_t output_channel;

  timestamp_t deadtime{};
  timestamp_t last_gen_event{};
};

class TT_API TransformCrosstalk : public IteratorBase {
public:
  TransformCrosstalk(TimeTaggerBase *tagger, channel_t input_channel, channel_t relay_input_channel, double delay,
                     double tau, bool copy = false);

  ~TransformCrosstalk();

  channel_t getChannel();

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  std::vector<Tag> mirror;

  const channel_t input_channel;
  const channel_t relay_input_channel;
  const channel_t output_channel;

  double delay{};
  double tau{};
  double accumulated_delay{};
  timestamp_t last_timestamp{};
  std::deque<Tag> delayed_tags;
  bool overflow_state_on{};
};

class TT_API SimDetector {
public:
  SimDetector(TimeTaggerBase *tagger, channel_t input_channel, double efficiency = 1.0, double darkcount_rate = 0.0,
              double jitter = 0, double deadtime = 0.0, int32_t seed = -1);

```

```
  ~SimDetector();

  channel_t getChannel();

private:
  channel_t output_channel;
  std::unique_ptr<TransformEfficiency> efficiency_meas;
  std::unique_ptr<ExponentialSignalGenerator> added_noise_meas;
  std::unique_ptr<TransformGaussianBroadening> jitter_meas;
  std::unique_ptr<TransformDeadtime> deadtime_meas;
};

class TT_API SimLifetime : public IteratorBase {
public:
  SimLifetime(TimeTaggerBase *tagger, channel_t input_channel, double lifetime, double emission_rate = 0.1,
              int32_t seed = -1);

  ~SimLifetime();

  channel_t getChannel();

  void registerLifetimeReactor(channel_t trigger_channel, std::vector<double> lifetimes, bool repeat);

  void registerEmissionReactor(channel_t trigger_channel, std::vector<double> emissions, bool repeat);

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;

private:
  std::vector<Tag> mirror;

  const channel_t input_channel;
  const channel_t output_channel;

  std::minstd_rand0 generator;
  std::exponential_distribution<double> lifetime_distr;
  std::poisson_distribution<uint32_t> emission_distr;
  std::vector<Tag> accumulated_tags;
  bool overflow_state_on{};

  // Reactors.
  bool has_reactor{};

  std::vector<double> reactor_lifetimes;
  channel_t reactor_trigger_lifetimes;
  bool repeat_lifetimes;
  size_t current_index_lifetimes;

  std::vector<double> reactor_emissions;
  channel_t reactor_trigger_emissions;
  bool repeat_emissions;
  size_t current_index_emissions;
};

class PhotonNumberImpl;
class TT_API PhotonNumber : public IteratorBase {
public:
  PhotonNumber(TimeTaggerBase *tagger, channel_t trigger_ch, channel_t signal_start_ch, channel_t signal_stop_ch,
               double slope, std::vector<double> x_intercepts, timestamp_t dead_time);

  ~PhotonNumber();

  std::vector<channel_t> const &getChannels() const;

protected:
  bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) override;
  void clear_impl() override;

private:
  friend class PhotonNumberImpl;
  std::unique_ptr<PhotonNumberImpl> impl;
};

} // namespace Experimental

#endif /* TT_ITERATORS_H_ */
```

## 10.3 TimeTagger.h File Reference

```
#include <atomic>
#include <condition_variable>
#include <cstddef>
#include <functional>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <set>
#include <stdexcept>
#include <stdint.h>
#include <string>
#include <unordered_set>
#include <vector>
```

Include dependency graph for TimeTagger.h:




### Classes

- struct SoftwareClockState
- class CustomLogger

  *Helper class for setLogger.*
- class TimeTaggerBase

  *Basis interface for all Time Tagger classes.*
- class TimeTaggerVirtual

  *virtual TimeTagger based on dump files*
- class TimeTaggerNetwork

  *network TimeTagger client.*
- class TimeTagger

  *backend for the TimeTagger.*
- struct Tag

  *a single event on a channel*
- class OrderedBarrier

  *Helper for implementing parallel measurements.*
- class OrderedBarrier::OrderInstance

  *Internal object for serialization.*
- class OrderedPipeline

  *Helper for implementing parallel measurements.*
- class IteratorBase

  *Base class for all iterators.*
- class IteratorBase::AbortError

  *A custom runtime error thrown by the `abort` call. This can be caught and handled by measurement classes, including `CustomMeasurement`, to perform actions within the abortion process.*

### Macros

- #define TT_API __declspec(dllimport)
- #define timestamp_t long long

  *The type for all timestamps used in the Time Tagger suite, always in picoseconds.*
- #define channel_t int

  *The type for storing a channel identifier.*
- #define TIMETAGGER_VERSION "2.17.4"

  *The version of this software suite.*
- #define GET_DATA_1D(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t)> argout)

  *This are the default wrapper functions without any overloadings.*
- #define GET_DATA_1D_OP1(function_name, type, argout, optional_type, optional_name, optional_default, attribute) attribute void function_name(std::function<type *(size_t)> argout, optional_type optional_name = optional_default)
- #define GET_DATA_1D_OP2(function_name, type, argout, optional_type, optional_name, optional_default, optional_type2, optional_name2, optional_default2, attribute)
- #define GET_DATA_2D(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t, size_t)> argout)
- #define GET_DATA_2D_OP1(function_name, type, argout, optional_type, optional_name, optional_default, attribute)
- #define GET_DATA_2D_OP2(function_name, type, argout, optional_type, optional_name, optional_default, optional_type2, optional_name2, optional_default2, attribute)
- #define GET_DATA_3D(function_name, type, argout, attribute) attribute void function_name(std::function<type *(size_t, size_t, size_t)> argout)
- #define LogMessage(level, ...) LogBase(level, __FILE__, __LINE__, false, __VA_ARGS__);
- #define ErrorLog(...) LogMessage(LOGGER_ERROR, __VA_ARGS__);
- #define WarningLog(...) LogMessage(LOGGER_WARNING, __VA_ARGS__);
- #define InfoLog(...) LogMessage(LOGGER_INFO, __VA_ARGS__);
- #define LogMessageSuppressed(level, ...) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);
- #define ErrorLogSuppressed(...) LogMessageSuppressed(LOGGER_ERROR, __VA_ARGS__);
- #define WarningLogSuppressed(...) LogMessageSuppressed(LOGGER_WARNING, __VA_ARGS__);
- #define InfoLogSuppressed(...) LogMessageSuppressed(LOGGER_INFO, __VA_ARGS__);

### Typedefs

- typedef void(* logger_callback) (LogLevel level, std::string msg)
- using _Iterator = IteratorBase

### Enumerations

- enum class Resolution { Standard = 0 , HighResA = 1 , HighResB = 2 , HighResC = 3 }

  *This enum selects the high resolution mode of the Time Tagger series.*
- enum class ChannelEdge : int32_t {
  NoFalling = 1 << 0 , NoRising = 1 << 1 , NoStandard = 1 << 2 , NoHighRes = 1 << 3 ,
  All = 0 , Rising = 1 , Falling = 2 , HighResAll = 4 ,
  HighResRising = 4 | 1 , HighResFalling = 4 | 2 , StandardAll = 8 , StandardRising = 8 | 1 ,
  StandardFalling = 8 | 2 }

  *Enum for filtering the channel list returned by getChannelList.*
- enum class FpgaLinkInterface { SFPP_10GE , QSFPP_40GE }

  *Enum for selecting the fpga link output interface.*
- enum LogLevel { LOGGER_ERROR = 40 , LOGGER_WARNING = 30 , LOGGER_INFO = 10 }
- enum class AccessMode { Listen = 0 , Control = 2 , SynchronousControl = 3 }

- enum class LanguageUsed : std::uint32_t {
  Cpp = 0 , Python , Csharp , Matlab ,
  Labview , Mathematica , Unknown = 255 }
- enum class FrontendType : std::uint32_t {
  Undefined = 0 , WebApp , Firefly , Pyro5RPC ,
  UserFrontend }
- enum class UsageStatisticsStatus { Disabled , Collecting , CollectingAndUploading }

## Functions

- TT_API std::string getVersion ()

  *Get the version of the TimeTagger cxx backend.*
- TT_API TimeTagger ∗ createTimeTagger (std::string serial="", Resolution resolution=Resolution::Standard)

  *default constructor factory.*
- TT_API TimeTaggerVirtual ∗ createTimeTaggerVirtual ()

  *default constructor factory for the createTimeTaggerVirtual class.*
- TT_API TimeTaggerNetwork ∗ createTimeTaggerNetwork (std::string address="localhost:41101")

  *default constructor factory for the TimeTaggerNetwork class.*
- TT_API void setCustomBitFileName (const std::string &bitFileName)

  *set path and filename of the bitfile to be loaded into the FPGA*
- TT_API bool freeTimeTagger (TimeTaggerBase ∗tagger)

  *free a copy of a TimeTagger reference.*
- TT_API std::vector< std::string > scanTimeTagger ()

  *fetches a list of all available TimeTagger serials.*
- TT_API std::string getTimeTaggerServerInfo (std::string address="localhost:41101")

  *connect to a Time Tagger server.*
- TT_API std::vector< std::string > scanTimeTaggerServers ()

  *scan the local network for running time tagger servers.*
- TT_API std::string getTimeTaggerModel (const std::string &serial)
- TT_API void setTimeTaggerChannelNumberScheme (int scheme)

  *Configure the numbering scheme for new TimeTagger objects.*
- TT_API int getTimeTaggerChannelNumberScheme ()

  *Fetch the currently configured global numbering scheme.*
- TT_API bool hasTimeTaggerVirtualLicense ()

  *Check if a license for the TimeTaggerVirtual is available.*
- TT_API void flashLicense (const std::string &serial, const std::string &license)

  *Update the license on the device.*
- TT_API std::string extractDeviceLicense (const std::string &license)

  *Converts binary license to JSON.*
- TT_API logger_callback setLogger (logger_callback callback)

  *Sets the notifier callback which is called for each log message.*
- TT_API void LogBase (LogLevel level, const char ∗file, int line, bool suppressed, const char ∗fmt,...)

  *Raise a new log message. Please use the XXXLog macro instead.*
- TT_API void checkSystemLibraries ()

  *Checks the MSVCP and okFrontPanel system library if they match the expected versions.*
- TT_API bool operator== (Tag const &a, Tag const &b)
- 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.*
- TT_API void mergeStreamFiles (const std::string &output_filename, const std::vector< std::string > &input_filenames, const std::vector< int > &channel_offsets, const std::vector< timestamp_t > &time_offsets, bool overlap_only)

  *merges several tag streams.*

### Variables

- constexpr channel_t CHANNEL_UNUSED = -134217728

  *Constant for unused channel.*
- constexpr channel_t CHANNEL_UNUSED_OLD = -1
- constexpr int TT_CHANNEL_NUMBER_SCHEME_AUTO = 0

  *Allowed values for setTimeTaggerChannelNumberScheme().*
- constexpr int TT_CHANNEL_NUMBER_SCHEME_ZERO = 1
- constexpr int TT_CHANNEL_NUMBER_SCHEME_ONE = 2
- constexpr int TT_CHANNEL_NUMBER_SCHEME_DEFAULT = 3
- 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

## 10.3.1 Macro Definition Documentation

### 10.3.1.1 channel_t

```
#define channel_t int
```

The type for storing a channel identifier.

### 10.3.1.2 ErrorLog

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

### 10.3.1.3 ErrorLogSuppressed

```
#define ErrorLogSuppressed(
            ... ) LogMessageSuppressed(LOGGER_ERROR, __VA_ARGS__);
```

### 10.3.1.4 GET_DATA_1D

```
#define GET_DATA_1D(
            function_name,
            type,
            argout,
            attribute ) attribute void function_name(std::function<type *(size_t)> argout)
```

This are the default wrapper functions without any overloadings.

#### 10.3.1.5 GET_DATA_1D_OP1

```
#define GET_DATA_1D_OP1(
              function_name,
              type,
              argout,
              optional_type,
              optional_name,
              optional_default,
              attribute )  attribute void function_name(std::function<type *(size_t)> argout,
optional_type optional_name = optional_default)
```

#### 10.3.1.6 GET_DATA_1D_OP2

```
#define GET_DATA_1D_OP2(
              function_name,
              type,
              argout,
              optional_type,
              optional_name,
              optional_default,
              optional_type2,
              optional_name2,
              optional_default2,
              attribute )
```

**Value:**

```
attribute void function_name(std::function<type *(size_t)> argout, optional_type optional_name =
    optional_default,    \
                             optional_type2 optional_name2 = optional_default2)
```

#### 10.3.1.7 GET_DATA_2D

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

#### 10.3.1.8 GET_DATA_2D_OP1

```
#define GET_DATA_2D_OP1(
              function_name,
              type,
              argout,
              optional_type,
              optional_name,
              optional_default,
              attribute )
```

**Value:**

```
attribute void function_name(std::function<type *(size_t, size_t)> argout,
    \
                             optional_type optional_name = optional_default)
```

#### 10.3.1.9 GET_DATA_2D_OP2

```
#define GET_DATA_2D_OP2(
            function_name,
            type,
            argout,
            optional_type,
            optional_name,
            optional_default,
            optional_type2,
            optional_name2,
            optional_default2,
            attribute )
```

**Value:**

```
attribute void function_name(std::function<type *(size_t, size_t)> argout,
    \
                            optional_type optional_name = optional_default,
    \
                            optional_type2 optional_name2 = optional_default2)
```

#### 10.3.1.10 GET_DATA_3D

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

#### 10.3.1.11 InfoLog

```
#define InfoLog(
            ... ) LogMessage(LOGGER_INFO, __VA_ARGS__);
```

#### 10.3.1.12 InfoLogSuppressed

```
#define InfoLogSuppressed(
            ... ) LogMessageSuppressed(LOGGER_INFO, __VA_ARGS__);
```

#### 10.3.1.13 LogMessage

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

#### 10.3.1.14 LogMessageSuppressed

```
#define LogMessageSuppressed(
            level,
            ... ) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);
```

#### 10.3.1.15 timestamp_t

```
#define timestamp_t long long
```

The type for all timestamps used in the Time Tagger suite, always in picoseconds.

#### 10.3.1.16 TIMETAGGER_VERSION

```
#define TIMETAGGER_VERSION "2.17.4"
```

The version of this software suite.

#### 10.3.1.17 TT_API

```
#define TT_API __declspec(dllimport)
```

#### 10.3.1.18 WarningLog

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

#### 10.3.1.19 WarningLogSuppressed

```
#define WarningLogSuppressed(
              ... ) LogMessageSuppressed(LOGGER_WARNING, __VA_ARGS__);
```

### 10.3.2 Typedef Documentation

#### 10.3.2.1 _Iterator

```
using _Iterator = IteratorBase
```

#### 10.3.2.2 logger_callback

```
typedef void(* logger_callback) (LogLevel level, std::string msg)
```

### 10.3.3 Enumeration Type Documentation

#### 10.3.3.1 AccessMode

```
enum class AccessMode  [strong]
```

**Enumerator**

| | |
|---|---|
| Listen | |
| Control | |
| SynchronousControl | |

#### 10.3.3.2 ChannelEdge

```
enum class 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 | |

#### 10.3.3.3 FpgaLinkInterface

```
enum class FpgaLinkInterface [strong]
```

Enum for selecting the fpga link output interface.

**Enumerator**

| | |
|---|---|
| SFPP_10GE | |
| QSFPP_40GE | |

#### 10.3.3.4 FrontendType

```
enum class FrontendType : std::uint32_t [strong]
```

**Enumerator**

| | |
|---|---|
| Undefined | |

**Enumerator**

| | |
|---|---|
| WebApp | |
| Firefly | |
| Pyro5RPC | |
| UserFrontend | |

#### 10.3.3.5 LanguageUsed

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

**Enumerator**

| | |
|---|---|
| Cpp | |
| Python | |
| Csharp | |
| Matlab | |
| Labview | |
| Mathematica | |
| Unknown | |

#### 10.3.3.6 LogLevel

```
enum LogLevel
```

**Enumerator**

| | |
|---|---|
| LOGGER_ERROR | |
| LOGGER_WARNING | |
| LOGGER_INFO | |

#### 10.3.3.7 Resolution

```
enum class 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 | |

#### 10.3.3.8 UsageStatisticsStatus

```
enum class UsageStatisticsStatus  [strong]
```

**Enumerator**

| | |
|---|---|
| Disabled | |
| Collecting | |
| CollectingAndUploading | |

### 10.3.4 Function Documentation

#### 10.3.4.1 checkSystemLibraries()

```
TT_API void checkSystemLibraries ( )
```

Checks the MSVCP and okFrontPanel system library if they match the expected versions.

#### 10.3.4.2 createTimeTagger()

```
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

#### 10.3.4.3 createTimeTaggerNetwork()

```
TT_API TimeTaggerNetwork * createTimeTaggerNetwork (
            std::string address = "localhost:41101" )
```

default constructor factory for the TimeTaggerNetwork class.

**Parameters**

| | |
|---|---|
| *address* | IP address of the server. Use hostname:port. |

#### 10.3.4.4 createTimeTaggerVirtual()

```
TT_API TimeTaggerVirtual * createTimeTaggerVirtual ( )
```

default constructor factory for the createTimeTaggerVirtual class.

#### 10.3.4.5 extractDeviceLicense()

```
TT_API std::string extractDeviceLicense (
            const std::string & license )
```

Converts binary license to JSON.

**Parameters**

| | |
|---|---|
| *license* | the binary license, encoded as a hexadecimal string |

**Returns**

a JSON string containing the current device license

#### 10.3.4.6 flashLicense()

```
TT_API void flashLicense (
            const std::string & serial,
            const std::string & license )
```

Update the license on the device.

Updated license may be fetched by getRemoteLicense. The Time Tagger must not be instantiated while updating the license.

**Parameters**

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

#### 10.3.4.7 freeTimeTagger()

```
TT_API bool freeTimeTagger (
            TimeTaggerBase * tagger )
```

free a copy of a TimeTagger reference.

**Parameters**

| | |
|---|---|
| *tagger* | the TimeTagger reference to free |

#### 10.3.4.8 getTimeTaggerChannelNumberScheme()

```
TT_API int getTimeTaggerChannelNumberScheme ( )
```

Fetch the currently configured global numbering scheme.

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

#### 10.3.4.9 getTimeTaggerModel()

```
TT_API std::string getTimeTaggerModel (
            const std::string & serial )
```

#### 10.3.4.10 getTimeTaggerServerInfo()

```
TT_API std::string getTimeTaggerServerInfo (
            std::string address = "localhost:41101" )
```

connect to a Time Tagger server.

**Parameters**

| | |
|---|---|
| *address* | ip address or domain and port of the server hosting time tagger. Use hostname:port. |

#### 10.3.4.11 getUsageStatisticsReport()

```
TT_API std::string getUsageStatisticsReport ( )
```

gets the current recorded data by the usage statistics system.

Use this function to see what data has been collected so far and what will be sent to Swabian Instruments if 'CollectingAndUploading' is enabled. All data is pseudonymous.

**Note**

if no data has been collected or due to a system error, the database was corrupted, it will return an error. else it will be a database in json format.

**Returns**

the current recorded data by the usage statistics system.

#### 10.3.4.12 getUsageStatisticsStatus()

```
TT_API UsageStatisticsStatus getUsageStatisticsStatus ( )
```

gets the status of the usage statistics system.

**Returns**

the current status of the usage statistics system.

#### 10.3.4.13 getVersion()

```
TT_API std::string getVersion ( )
```

Get the version of the TimeTagger cxx backend.

#### 10.3.4.14 hasTimeTaggerVirtualLicense()

```
TT_API bool hasTimeTaggerVirtualLicense ( )
```

Check if a license for the TimeTaggerVirtual is available.

#### 10.3.4.15 LogBase()

```
TT_API void LogBase (
            LogLevel level,
            const char * file,
            int line,
            bool suppressed,
            const char * fmt,
             ... )
```

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

#### 10.3.4.16 mergeStreamFiles()

```
TT_API void mergeStreamFiles (
            const std::string & output_filename,
            const std::vector< std::string > & input_filenames,
            const std::vector< int > & channel_offsets,
            const std::vector< timestamp_t > & time_offsets,
            bool overlap_only )
```

merges several tag streams.

The function reads tags from several input streams, adjusts channel numbers and tag time as specified by 'channel_offsets' and 'time_offsets' respectively, and merges them to a single output stream. Throws if merge cannot be done.

**Parameters**

| | |
|---|---|
| *output_filename* | output stream file name, splitting is done as in 'FileWriter', with 1GB file size limit. |
| *input_filenames* | file names of input streams. |
| *channel_offsets* | offsets to shift channel numbers for corresponding input streams. |
| *time_offsets* | offsets to shift tag time for corresponding input streams. |
| *overlap_only* | specifies if only events in the time overlapping region of all input streams should be merged. |

#### 10.3.4.17 operator==()

```
TT_API bool operator== (
            Tag const & a,
            Tag const & b )
```

#### 10.3.4.18 scanTimeTagger()

```
TT_API std::vector< std::string > scanTimeTagger ( )
```

fetches a list of all available TimeTagger serials.

This function may return serials blocked by other processes or already disconnected some milliseconds later.

#### 10.3.4.19 scanTimeTaggerServers()

```
TT_API std::vector< std::string > scanTimeTaggerServers ( )
```

scan the local network for running time tagger servers.

**Returns**

a vector of strings of "ip_address:port" for each active server in local network.

#### 10.3.4.20 setCustomBitFileName()

```
TT_API void setCustomBitFileName (
            const std::string & bitFileName )
```

set path and filename of the bitfile to be loaded into the FPGA

For debugging/development purposes the firmware loaded into the FPGA can be set manually with this function. To load the default bitfile set bitFileName = ""

**Parameters**

| | |
|---|---|
| *bitFileName* | custom bitfile to use for the FPGA. |

#### 10.3.4.21 setFrontend()

```
TT_API void setFrontend (
            FrontendType frontend )
```

sets the frontend being used currently for usage statistics system.

**Parameters**

| | |
|---|---|
| *frontend* | the frontend currently being used. |

#### 10.3.4.22 setLanguageInfo()

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

sets the language being used currently for usage statistics system.

**Parameters**

| | |
|---|---|
| *pw* | password for authorization to change the language. |
| *language* | programming language being used. |
| *version* | version of the programming language being used. |

#### 10.3.4.23 setLogger()

```
TT_API logger_callback setLogger (
            logger_callback callback )
```

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

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

**Returns**

The old callback

#### 10.3.4.24 setTimeTaggerChannelNumberScheme()

```
TT_API void setTimeTaggerChannelNumberScheme (
            int scheme )
```

Configure the numbering scheme for new TimeTagger objects.

This function sets the numbering scheme for newly created TimeTagger objects. The default value is _AUTO.

Note: TimeTagger objects are cached internally, so the scheme should be set before the first call of createTimeTagger().

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

_ONE will typically allocate the channel numbers 1 to 8 for the 8 input channels. -1 to -8 will be allocated for the corresponding falling events.

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

**Parameters**

| | |
|---|---|
| *scheme* | new numbering scheme, must be TT_CHANNEL_NUMBER_SCHEME_AUTO, TT_CHANNEL_NUMBER_SCHEME_ZERO or TT_CHANNEL_NUMBER_SCHEME_ONE |

#### 10.3.4.25 setUsageStatisticsStatus()

```
TT_API void setUsageStatisticsStatus (
            UsageStatisticsStatus new_status )
```

sets the status of the usage statistics system.

This functionality allows configuring the usage statistics system.

**Parameters**

| | |
|---|---|
| *new_status* | new status of the usage statistics system. |

### 10.3.5 Variable Documentation

#### 10.3.5.1 CHANNEL_UNUSED

```
constexpr channel_t CHANNEL_UNUSED = -134217728 [constexpr]
```

Constant for unused channel.

Magic channel_t value to indicate an unused channel. So the iterators either have to disable this channel, or to choose a default one.

This value changed in version 2.1. The old value -1 aliases with falling events. The old value will still be accepted for now if the old numbering scheme is active.

#### 10.3.5.2 CHANNEL_UNUSED_OLD

```
constexpr channel_t CHANNEL_UNUSED_OLD = -1 [constexpr]
```

#### 10.3.5.3 TT_CHANNEL_FALLING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_FALLING_EDGES = ChannelEdge::Falling [constexpr]
```

#### 10.3.5.4 TT_CHANNEL_NUMBER_SCHEME_AUTO

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_AUTO = 0 [constexpr]
```

Allowed values for setTimeTaggerChannelNumberScheme().

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

_ONE will typically allocate the channel numbers 1 to 8 for the 8 input channels. -1 to -8 will be allocated for the corresponding falling events.

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

_DEFAULT will always pick _ONE, but it will yield a warning if _AUTO would have picked _ZERO.

#### 10.3.5.5 TT_CHANNEL_NUMBER_SCHEME_DEFAULT

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_DEFAULT = 3 [constexpr]
```

#### 10.3.5.6 TT_CHANNEL_NUMBER_SCHEME_ONE

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_ONE = 2 [constexpr]
```

#### 10.3.5.7 TT_CHANNEL_NUMBER_SCHEME_ZERO

```
constexpr int TT_CHANNEL_NUMBER_SCHEME_ZERO = 1 [constexpr]
```

#### 10.3.5.8 TT_CHANNEL_RISING_AND_FALLING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_RISING_AND_FALLING_EDGES = ChannelEdge::All [constexpr]
```

#### 10.3.5.9 TT_CHANNEL_RISING_EDGES

```
constexpr ChannelEdge TT_CHANNEL_RISING_EDGES = ChannelEdge::Rising [constexpr]
```

## 10.4 TimeTagger.h

Go to the documentation of this file.

```
/*
This file is part of Time Tagger software defined digital data acquisition.

Copyright (C) 2011-2019 Swabian Instruments
All Rights Reserved

Unauthorized copying of this file is strictly prohibited.
*/

#ifndef TIMETAGGER_H_
#define TIMETAGGER_H_

#ifdef _MSC_VER
#pragma warning(disable : 4251)
#endif

#ifdef LIBTIMETAGGER_EXPORTS
#ifdef _WIN32
#define TT_API __declspec(dllexport)
#else
#define TT_API __attribute__((visibility("default")))
#endif
#else
#if defined(__linux) || defined(SWIG) || defined(NOEXPORT)
#define TT_API
#else
#define TT_API __declspec(dllimport)
#ifdef _DEBUG
#pragma comment(lib, "TimeTaggerD")
#else
#pragma comment(lib, "TimeTagger")
#endif
#endif
#endif

#include <atomic>
#include <condition_variable>
#include <cstddef>
#include <functional>
#include <limits>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <set>
#include <stdexcept>
#include <stdint.h>
#include <string>
#include <unordered_set>
#include <vector>

class IteratorBase;
class IteratorBaseListNode;
class TimeTagger;
class TimeTaggerBase;
class TimeTaggerNetwork;
class TimeTaggerRunner;
class TimeTaggerVirtual;

#define timestamp_t long long

#define channel_t int

#ifndef SWIG
#define TIMETAGGER_VERSION "2.17.4"
#endif

TT_API std::string getVersion();

constexpr channel_t CHANNEL_UNUSED = -134217728;
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 int TT_CHANNEL_NUMBER_SCHEME_DEFAULT = 3;

#ifndef TIMETAGGER_NO_WRAPPER
#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)                                                    \
  attribute void function_name(std::function<type *(size_t)> argout, optional_type optional_name = optional_default,    \
                               optional_type2 optional_name2 = optional_default2)
#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)         \
  attribute void function_name(std::function<type *(size_t, size_t)> argout,                                            \
                               optional_type optional_name = optional_default)
#define GET_DATA_2D_OP2(function_name, type, argout, optional_type, optional_name, optional_default, optional_type2,    \
                        optional_name2, optional_default2, attribute)                                                    \
  attribute void function_name(std::function<type *(size_t, size_t)> argout,                                            \
                               optional_type optional_name = optional_default,                                          \
                               optional_type2 optional_name2 = optional_default2)
#define GET_DATA_3D(function_name, type, argout, attribute)                                                             \
  attribute void function_name(std::function<type *(size_t, size_t, size_t)> argout)
#endif

enum class Resolution { Standard = 0, HighResA = 1, HighResB = 2, HighResC = 3 };

enum class ChannelEdge : int32_t {
// Bitwise filters, shall not be exported to wrapped languages
#ifndef SWIG
  NoFalling = 1 « 0,
  NoRising = 1 « 1,
  NoStandard = 1 « 2,
  NoHighRes = 1 « 3,
#endif

  All = 0,
  Rising = 1,
  Falling = 2,
  HighResAll = 4,
  HighResRising = 4 | 1,
  HighResFalling = 4 | 2,
  StandardAll = 8,
  StandardRising = 8 | 1,
  StandardFalling = 8 | 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;

struct SoftwareClockState {
  // configuration state
  timestamp_t clock_period;
  channel_t input_channel;
  channel_t ideal_clock_channel;
  double averaging_periods;
  bool enabled;

  // runtime information
  bool is_locked;
  uint32_t error_counter;
  timestamp_t last_ideal_clock_event;
  double period_error;           // in picoseconds
  double phase_error_estimation; // in picoseconds, including TDC discretization error
};

enum class FpgaLinkInterface {
  SFPP_10GE,
  QSFPP_40GE,
};

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

TT_API TimeTaggerVirtual *createTimeTaggerVirtual();

TT_API TimeTaggerNetwork *createTimeTaggerNetwork(std::string address = "localhost:41101");

TT_API void setCustomBitFileName(const std::string &bitFileName);

TT_API bool freeTimeTagger(TimeTaggerBase *tagger);
```

```
TT_API std::vector<std::string> scanTimeTagger();

TT_API std::string getTimeTaggerServerInfo(std::string address = "localhost:41101");

TT_API std::vector<std::string> scanTimeTaggerServers();

/*
 * \brief returns the model name of the Time Tagger identified by the serial number.
 *
 * \param serial the Time Tagger serial number to query
 */
TT_API std::string getTimeTaggerModel(const std::string &serial);

TT_API void setTimeTaggerChannelNumberScheme(int scheme);

TT_API int getTimeTaggerChannelNumberScheme();

TT_API bool hasTimeTaggerVirtualLicense();

TT_API void flashLicense(const std::string &serial, const std::string &license);

TT_API std::string extractDeviceLicense(const std::string &license);

// log values are taken from https://docs.python.org/3/library/logging.html
enum LogLevel { LOGGER_ERROR = 40, LOGGER_WARNING = 30, LOGGER_INFO = 10 };
typedef void (*logger_callback)(LogLevel level, std::string msg);

TT_API logger_callback setLogger(logger_callback callback);

TT_API void LogBase(LogLevel level, const char *file, int line, bool suppressed, const char *fmt, ...)
#ifdef __GNUC__
    __attribute__((format(printf, 5, 6)))
#endif
    ;
#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__);

// This suppressed methods are used when the log may contain private/confidential data and we
// don't want the usage statistics system to record such data.
#define LogMessageSuppressed(level, ...) LogBase(level, __FILE__, __LINE__, true, __VA_ARGS__);
#define ErrorLogSuppressed(...) LogMessageSuppressed(LOGGER_ERROR, __VA_ARGS__);
#define WarningLogSuppressed(...) LogMessageSuppressed(LOGGER_WARNING, __VA_ARGS__);
#define InfoLogSuppressed(...) LogMessageSuppressed(LOGGER_INFO, __VA_ARGS__);

class TT_API CustomLogger {
public:
  CustomLogger();
  virtual ~CustomLogger();

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

private:
  static void LogCallback(LogLevel level, std::string msg);
  static CustomLogger *instance;
  static std::mutex instance_mutex;
};

TT_API void checkSystemLibraries();

class ClientNetworkStream;
class TT_API TimeTaggerBase {
  friend class IteratorBase;
  friend class TimeTaggerProxy;
  friend class TimeTaggerRunner;
  friend class ClientNetworkStream;

public:
  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;

  virtual bool isUnusedChannel(channel_t channel) = 0;

  typedef std::function<void(IteratorBase *)> IteratorCallback;
  typedef std::map<IteratorBase *, IteratorCallback> IteratorCallbackMap;

  virtual void runSynchronized(const IteratorCallbackMap &callbacks, bool block = true) = 0;

```

```
  virtual std::string getConfiguration() = 0;

  virtual void setInputDelay(channel_t channel, timestamp_t delay) = 0;

  virtual void setDelayHardware(channel_t channel, timestamp_t delay) = 0;

  virtual void setDelaySoftware(channel_t channel, timestamp_t delay) = 0;

  virtual timestamp_t getInputDelay(channel_t channel) = 0;

  virtual timestamp_t getDelaySoftware(channel_t channel) = 0;

  virtual timestamp_t getDelayHardware(channel_t channel) = 0;

  virtual timestamp_t setDeadtime(channel_t channel, timestamp_t deadtime) = 0;

  virtual timestamp_t getDeadtime(channel_t channel) = 0;

  virtual void setTestSignal(channel_t channel, bool enabled) = 0;

  virtual void setTestSignal(std::vector<channel_t> channel, bool enabled) = 0;

  virtual bool getTestSignal(channel_t channel) = 0;

  virtual void setSoftwareClock(channel_t input_channel, double input_frequency = 10e6, double
averaging_periods = 1000,
                                bool wait_until_locked = true) = 0;

  virtual void disableSoftwareClock() = 0;

  virtual SoftwareClockState getSoftwareClockState() = 0;

  virtual long long getOverflows() = 0;

  virtual void clearOverflows() = 0;

  virtual long long getOverflowsAndClear() = 0;

protected:
  TimeTaggerBase() {}

  virtual ~TimeTaggerBase(){};

  // Non Copyable
  TimeTaggerBase(const TimeTaggerBase &) = delete;
  TimeTaggerBase &operator=(const TimeTaggerBase &) = delete;

  // Used by IteratorBase to add itself
  virtual std::shared_ptr<IteratorBaseListNode> addIterator(IteratorBase *it) = 0;

  // Used by IteratorBase to specify when it's being deleted.
  virtual void freeIterator(IteratorBase *it) = 0;

  // allocate a new virtual output channel
  virtual channel_t getNewVirtualChannel() = 0;

  // free a virtual channel being used.
  virtual void freeVirtualChannel(channel_t channel) = 0;

  virtual void registerChannel(channel_t channel) = 0;
  virtual void registerChannel(std::set<channel_t> channels) = 0;

  virtual void unregisterChannel(channel_t channel) = 0;
  virtual void unregisterChannel(std::set<channel_t> channels) = 0;

  // Used by proxy time tagger to add itself as a dependent tagger.
  virtual void addChild(TimeTaggerBase *child) = 0;

  // Used by proxy time tagger to remove itself as a dependent tagger.
  virtual void removeChild(TimeTaggerBase *child) = 0;

  // Used by a proxy time tagger to allow its parent to release it and its dependencies.
  virtual void release() = 0;
};

class TT_API TimeTaggerVirtual : virtual public TimeTaggerBase {
public:
  virtual uint64_t replay(const std::string &file, timestamp_t begin = 0, timestamp_t duration = -1,
                          bool queue = true) = 0;

  virtual void stop() = 0;

  virtual void reset() = 0;

  virtual bool waitForCompletion(uint64_t ID = 0, int64_t timeout = -1) = 0;

  virtual void setReplaySpeed(double speed) = 0;
```

```

  virtual double getReplaySpeed() = 0;

  virtual void setConditionalFilter(std::vector<channel_t> trigger, std::vector<channel_t> filtered) =
0;

  virtual void clearConditionalFilter() = 0;

  virtual std::vector<channel_t> getConditionalFilterTrigger() = 0;

  virtual std::vector<channel_t> getConditionalFilterFiltered() = 0;

  virtual std::vector<channel_t> getChannelList() = 0;
};

enum class AccessMode { Listen = 0, Control = 2, SynchronousControl = 3 };

class TT_API TimeTaggerNetwork : virtual public TimeTaggerBase {
public:
  virtual bool isConnected() = 0;

  virtual void setTriggerLevel(channel_t channel, double voltage) = 0;

  virtual double getTriggerLevel(channel_t channel) = 0;

  virtual void setConditionalFilter(std::vector<channel_t> trigger, std::vector<channel_t> filtered,
                                    bool hardwareDelayCompensation = true) = 0;

  virtual void clearConditionalFilter() = 0;

  virtual std::vector<channel_t> getConditionalFilterTrigger() = 0;

  virtual std::vector<channel_t> getConditionalFilterFiltered() = 0;

  virtual void setTestSignalDivider(int divider) = 0;

  virtual int getTestSignalDivider() = 0;

  virtual bool getTestSignal(channel_t channel) = 0;

  virtual void setDelayClient(channel_t channel, timestamp_t time) = 0;

  virtual timestamp_t getDelayClient(channel_t channel) = 0;

  virtual timestamp_t getHardwareDelayCompensation(channel_t channel) = 0;

  virtual void setNormalization(std::vector<channel_t> channels, bool state) = 0;

  virtual bool getNormalization(channel_t channel) = 0;

  virtual void setHardwareBufferSize(int size) = 0;

  virtual int getHardwareBufferSize() = 0;

  virtual void setStreamBlockSize(int max_events, int max_latency) = 0;
  virtual int getStreamBlockSizeEvents() = 0;
  virtual int getStreamBlockSizeLatency() = 0;

  virtual void setEventDivider(channel_t channel, unsigned int divider) = 0;

  virtual unsigned int getEventDivider(channel_t channel) = 0;

  virtual std::string getSerial() = 0;

  virtual std::string getModel() = 0;

  virtual int getChannelNumberScheme() = 0;

  virtual std::vector<double> getDACRange() = 0;

  virtual std::vector<channel_t> getChannelList(ChannelEdge type = ChannelEdge::All) = 0;

  virtual timestamp_t getPsPerClock() = 0;

  virtual std::string getPcbVersion() = 0;

  virtual std::string getFirmwareVersion() = 0;

  virtual std::string getSensorData() = 0;

  virtual void setLED(uint32_t bitmask) = 0;

  virtual std::string getDeviceLicense() = 0;

  virtual void setSoundFrequency(uint32_t freq_hz) = 0;

  virtual void setTimeTaggerNetworkStreamCompression(bool active) = 0;
```

```
  virtual long long getOverflowsClient() = 0;
  virtual void clearOverflowsClient() = 0;
  virtual long long getOverflowsAndClearClient() = 0;

  virtual void setInputImpedanceHigh(channel_t channel, bool high_impedance) = 0;

  virtual bool getInputImpedanceHigh(channel_t channel) = 0;

  virtual void setInputHysteresis(channel_t channel, int value) = 0;

  virtual int getInputHysteresis(channel_t channel) = 0;
};

class TT_API TimeTagger : virtual public TimeTaggerBase {
public:
  virtual void reset() = 0;

  virtual bool isChannelRegistered(channel_t chan) = 0;

  virtual void setTestSignalDivider(int divider) = 0;

  virtual int getTestSignalDivider() = 0;

  virtual void xtra_setAuxOutSignal(int channel, int divider, double duty_cycle = 0.5) = 0;

  virtual int xtra_getAuxOutSignalDivider(int channel) = 0;

  virtual double xtra_getAuxOutSignalDutyCycle(int channel) = 0;

  virtual void xtra_setAuxOut(int channel, bool enabled) = 0;

  virtual bool xtra_getAuxOut(int channel) = 0;

  virtual void xtra_setFanSpeed(double percentage = -1) = 0;

  virtual void setTriggerLevel(channel_t channel, double voltage) = 0;

  virtual double getTriggerLevel(channel_t channel) = 0;

  virtual double xtra_measureTriggerLevel(channel_t channel) = 0;

  virtual timestamp_t getHardwareDelayCompensation(channel_t channel) = 0;

  virtual void setInputMux(channel_t channel, int mux_mode) = 0;

  virtual int getInputMux(channel_t channel) = 0;

  virtual void setConditionalFilter(std::vector<channel_t> trigger, std::vector<channel_t> filtered,
                                    bool hardwareDelayCompensation = true) = 0;

  virtual void clearConditionalFilter() = 0;

  virtual std::vector<channel_t> getConditionalFilterTrigger() = 0;

  virtual std::vector<channel_t> getConditionalFilterFiltered() = 0;

  virtual void setNormalization(std::vector<channel_t> channels, bool state) = 0;

  virtual bool getNormalization(channel_t channel) = 0;

  virtual void setHardwareBufferSize(int size) = 0;

  virtual int getHardwareBufferSize() = 0;

  virtual void setStreamBlockSize(int max_events, int max_latency) = 0;
  virtual int getStreamBlockSizeEvents() = 0;
  virtual int getStreamBlockSizeLatency() = 0;

  virtual void setEventDivider(channel_t channel, unsigned int divider) = 0;

  virtual unsigned int getEventDivider(channel_t channel) = 0;

  GET_DATA_1D(autoCalibration, double, array_out, virtual) = 0;

  virtual std::string getSerial() = 0;

  virtual std::string getModel() = 0;

  virtual int getChannelNumberScheme() = 0;

  virtual std::vector<double> getDACRange() = 0;

  GET_DATA_2D(getDistributionCount, uint64_t, array_out, virtual) = 0;

  GET_DATA_2D(getDistributionPSecs, double, array_out, virtual) = 0;

```

```
  virtual std::vector<channel_t> getChannelList(ChannelEdge type = ChannelEdge::All) = 0;

  virtual timestamp_t getPsPerClock() = 0;

  virtual std::string getPcbVersion() = 0;

  virtual std::string getFirmwareVersion() = 0;

  virtual void xtra_setClockSource(int source) = 0;

  virtual int xtra_getClockSource() = 0;

  virtual void xtra_setClockAutoSelect(bool enabled) = 0;

  virtual bool xtra_getClockAutoSelect() = 0;

  virtual void xtra_setClockOut(bool enabled) = 0;

  virtual std::string getSensorData() = 0;

  virtual void setLED(uint32_t bitmask) = 0;

  virtual void disableLEDs(bool disabled) = 0;

  virtual std::string getDeviceLicense() = 0;

  virtual uint32_t factoryAccess(uint32_t pw, uint32_t addr, uint32_t data, uint32_t mask, bool use_wb = false) = 0;

  virtual void setSoundFrequency(uint32_t freq_hz) = 0;

  virtual void enableFpgaLink(std::vector<channel_t> channels, std::string destination_mac,
                              FpgaLinkInterface link_interface = FpgaLinkInterface::SFPP_10GE,
                              bool exclusive = false) = 0;

  virtual void disableFpgaLink() = 0;

  virtual void startServer(AccessMode access_mode, std::vector<channel_t> channels = std::vector<channel_t>(),
                           uint32_t port = 41101) = 0;

  virtual bool isServerRunning() = 0;

  virtual void stopServer() = 0;

  virtual void setTimeTaggerNetworkStreamCompression(bool active) = 0;

  virtual void setInputImpedanceHigh(channel_t channel, bool high_impedance) = 0;

  virtual bool getInputImpedanceHigh(channel_t channel) = 0;

  virtual void setInputHysteresis(channel_t channel, int value) = 0;

  virtual int getInputHysteresis(channel_t channel) = 0;

  virtual void xtra_setAvgRisingFalling(channel_t channel, bool enable) = 0;

  virtual bool xtra_getAvgRisingFalling(channel_t channel) = 0;

  virtual void xtra_setHighPrioChannel(channel_t channel, bool enable) = 0;

  virtual bool xtra_getHighPrioChannel(channel_t channel) = 0;

  virtual void updateBMCFirmware(const std::string &firmware) = 0;
};

struct TT_API Tag {
  enum class Type : unsigned char {
    TimeTag = 0,
    Error = 1,
    OverflowBegin = 2,
    OverflowEnd = 3,
    MissedEvents = 4
  } type{Type::TimeTag};

  char reserved{};

  unsigned short missed_events{};

  channel_t channel{};

  timestamp_t time{};

  Tag() {}
  Tag(timestamp_t ts, channel_t ch, Type type = Type::TimeTag) : type{type}, channel{ch}, time{ts} {}
  Tag(Type type, char reserved, unsigned short missed_events, channel_t ch, timestamp_t ts)
      : type{type}, reserved{reserved}, missed_events{missed_events}, channel{ch}, time{ts} {}
```

```
};

TT_API bool operator==(Tag const &a, Tag const &b);

class TT_API OrderedBarrier {
public:
  class TT_API OrderInstance {
  public:
    OrderInstance();
    OrderInstance(OrderedBarrier *parent, uint64_t instance_id);
    ~OrderInstance();
    void sync();
    void release();

  private:
    friend class OrderedBarrier;

    OrderedBarrier *parent{};
    bool obtained{};
    uint64_t instance_id{};
  };

  OrderedBarrier();
  ~OrderedBarrier();

  OrderInstance queue();
  void waitUntilFinished();

private:
  friend class OrderInstance;

  void release(uint64_t index);
  void obtain(uint64_t index);

  uint64_t accumulator{};
  uint64_t current_state{};
  std::mutex inner_mutex;
  std::condition_variable cv;
};

class TT_API OrderedPipeline {
public:
  OrderedPipeline();
  ~OrderedPipeline();

private:
  friend class IteratorBase;

  bool initialized = false;
  std::list<OrderedBarrier>::iterator stage;
};

class TT_API IteratorBase {
  friend class TimeTaggerRunner;
  friend class TimeTaggerProxy;
  friend class SynchronizedMeasurements;
  friend class IteratorTest;

private:
  // Abstract class
  IteratorBase() = delete;

  // Non Copyable
  IteratorBase(const IteratorBase &) = delete;
  IteratorBase &operator=(const IteratorBase &) = delete;
  void clearWithoutLock();

protected:
  IteratorBase(TimeTaggerBase *tagger, std::string base_type_ = "IteratorBase", std::string extra_info_ = "");

public:
  virtual ~IteratorBase();

  void start();

  void startFor(timestamp_t capture_duration, bool clear = true);

  bool waitUntilFinished(int64_t timeout = -1);

  void stop();

  void clear();

  void abort();

  bool isRunning();
```

```
  timestamp_t getCaptureDuration();

  std::string getConfiguration();

  class AbortError : public std::runtime_error {
  public:
    AbortError(const std::string &what_arg) : std::runtime_error(what_arg){};
    ~AbortError(){};
  };

protected:
  void registerChannel(channel_t channel);

  void unregisterChannel(channel_t channel);

  channel_t getNewVirtualChannel();

  void finishInitialization();

  virtual void clear_impl() {};

  virtual void on_start() {};

  virtual void on_stop() {};

  void lock();

  void unlock();

  OrderedBarrier::OrderInstance parallelize(OrderedPipeline &pipeline);

  std::unique_lock<std::mutex> getLock();

  virtual bool next_impl(std::vector<Tag> &incoming_tags, timestamp_t begin_time, timestamp_t end_time) = 0;

  void finish_running();

  std::set<channel_t> channels_registered;

  bool running;

  bool autostart;

  TimeTaggerBase *tagger;

  timestamp_t capture_duration;
  timestamp_t pre_capture_duration;

  // to abort measurement;
  std::atomic<bool> aborting;
  // This call shall be placed in every next_impl loop to allow single threaded measurements to abort.
  // It will be inlined wherever header is included
  void checkForAbort() {
    if (aborting) {
      on_abort();
    }
  };
  // Overload for multithreaded measurements
  template <typename T> void checkForAbort(T callback) {
    if (aborting) {
      callback();
      on_abort();
    }
  };

private:
  struct TelemetryData {
    uint64_t duration;
    bool is_on;
  };

  void next(std::unique_lock<std::mutex> &lock, std::vector<Tag> &incoming_tags, timestamp_t begin_time,
            timestamp_t end_time, uint32_t fence, TelemetryData &telem_data);

  void pre_stop();
  void on_abort();
  std::shared_ptr<IteratorBaseListNode> iter;
  timestamp_t max_capture_duration; // capture duration at which the .stop() method will be called, <0 for infinity
  std::mutex pre_stop_mutex;
  uint32_t min_fence;
  std::unordered_set<channel_t> virtual_channels;
  const std::string base_type;
  const std::string extra_info;
```

```
  uint64_t id{};
  bool initialized{};
  uint64_t clear_tick{};
};
using _Iterator = IteratorBase;

enum class LanguageUsed : std::uint32_t {
  Cpp = 0,
  Python,
  Csharp,
  Matlab,
  Labview,
  Mathematica,
  // Add more languages/Platforms
  Unknown = 255,
};

enum class FrontendType : std::uint32_t {
  Undefined = 0,
  WebApp,
  Firefly,
  Pyro5RPC,
  UserFrontend,
};

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

TT_API void setFrontend(FrontendType frontend);

enum class UsageStatisticsStatus {
  Disabled,               // User Opted out
  Collecting,             // User enabled it to collect for debug purpose
  CollectingAndUploading, // User gave their consent to collect and upload
};

TT_API void setUsageStatisticsStatus(UsageStatisticsStatus new_status);

TT_API UsageStatisticsStatus getUsageStatisticsStatus();

TT_API std::string getUsageStatisticsReport();

TT_API void mergeStreamFiles(const std::string &output_filename, const std::vector<std::string> &input_filenames,
                             const std::vector<int> &channel_offsets, const std::vector<timestamp_t> &time_offsets,
                             bool overlap_only);

#endif /* TIMETAGGER_H_ */
```