TimeServer Documentation

Release 2.0

A. Beckmann and A. Parenti

Nov 12, 2024

Contents

Contents:

CHAPTER 1

Implementation Specification for TimeServer Device

Authors A. Beckmann and A. Parenti, for the Control and Data Analysis Software Group

Version 2.0a, July 2016

1.1 Revisions

Version	Date	Description
1.0	20 Apr 2015	Initial version
1.0a	04 Jul 2016	Fixed broken cross reference, added section on Karabo interface, added appendix for Karabo device integration and TCP protocol
2.0	20 Jul 2016	Fixed FSM state names
2.0a	25 Jul 2016	Refined class descriptions

1.2 Preface

This document provides details on the implementation of the Karabo device for providing trigger ID and time stamp information to other devices.

This specification covers software release 2.0.

1.3 Introduction

The TimeServer device provides trigger ID and time stamp for other Karabo devices, which are not capable of retrieving this information directly from hardware. The information is sent out for each trigger via message broker.

The trigger ID is either read out from the XFEL timing system, generated internally, or received as a simple message over TCP from another server.

1.3.1 System Context

The TimeServer device is operated in the context as shown in Fig. 1.1. It has an interface to the Karabo framework for user control as well as to the XFEL Timing system and other TCP servers.

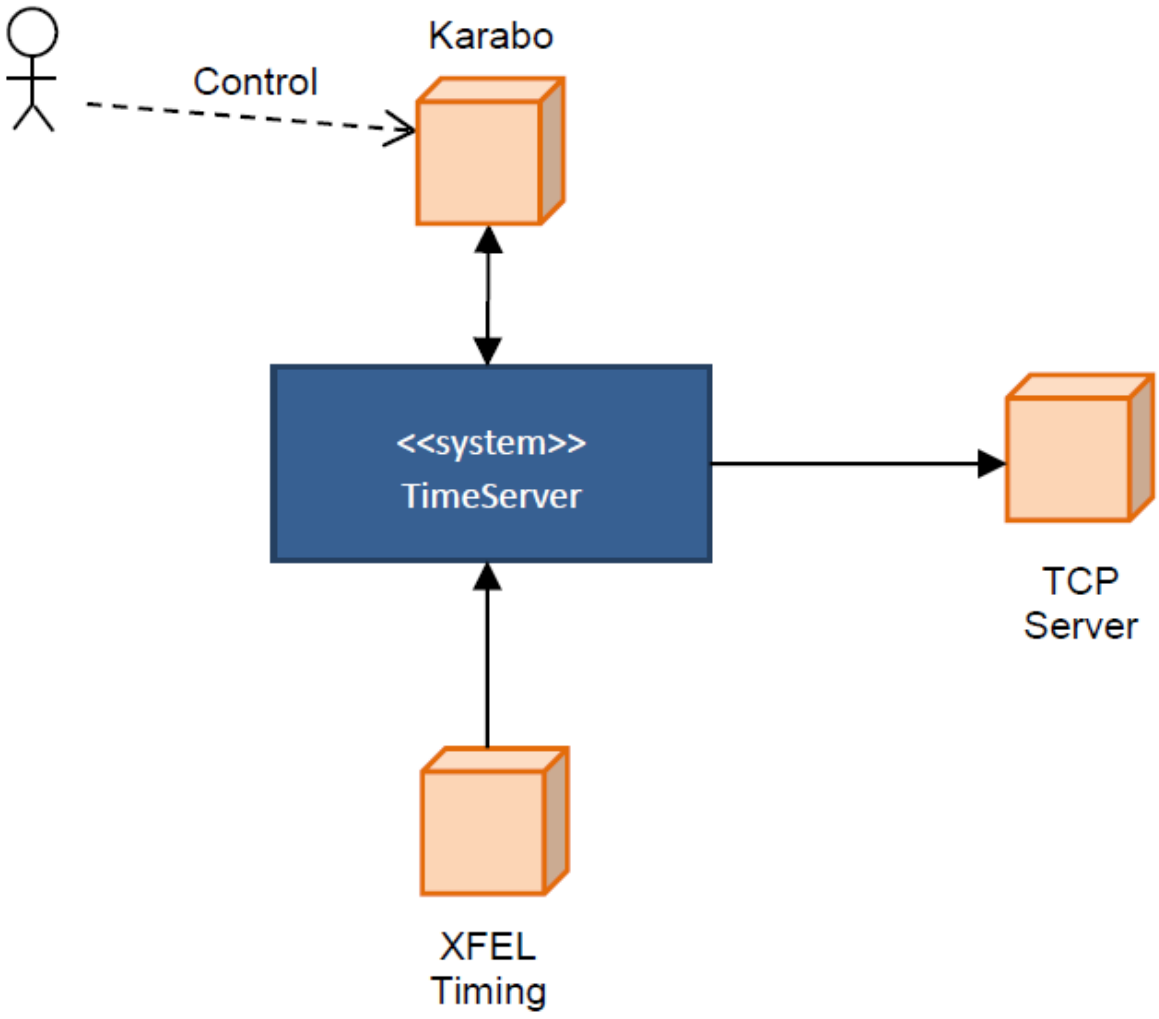

Fig. 1.1: System context.

1.3.2 Data Model

For each trigger, a time tick is sent by the TimeServer device, containing ID, timestamp as seconds since UNIX epoch and fractional seconds with attosecond resolution, and the averaged period in microsecond resolution.

The data model is depicted in Fig. 1.2.

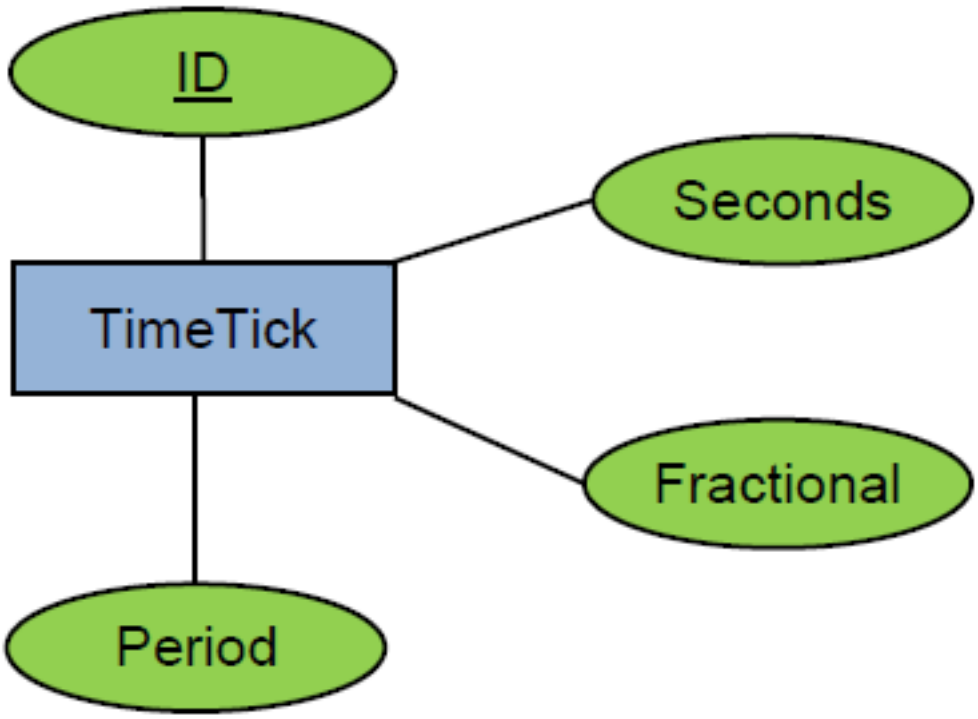

Fig. 1.2: Data model.

1.4 Architecture

The TimeServer device is divided into several classes to improve testability, as explained in the following section. Behavior is controlled by a finite state machine (FSM), which provides a set of methods to implement the desired logic.

1.4.1 Class Diagram

Fig. 1.3 shows the top level class diagram of the TimeServer device. The main class is karabo::TimeServer, which is derived from karabo::core::Device with the FSM template parameter bound to the karabo::core::BaseFsm state machine class. It has an association with class xfel::timer::Timer, which implements the core logic*[0].

[0] Classes used inside the TimeServer device are defined in namespace karabo, if they depend on resources from the Karabo framework, otherwise they are defined in namespace xfel::timer.

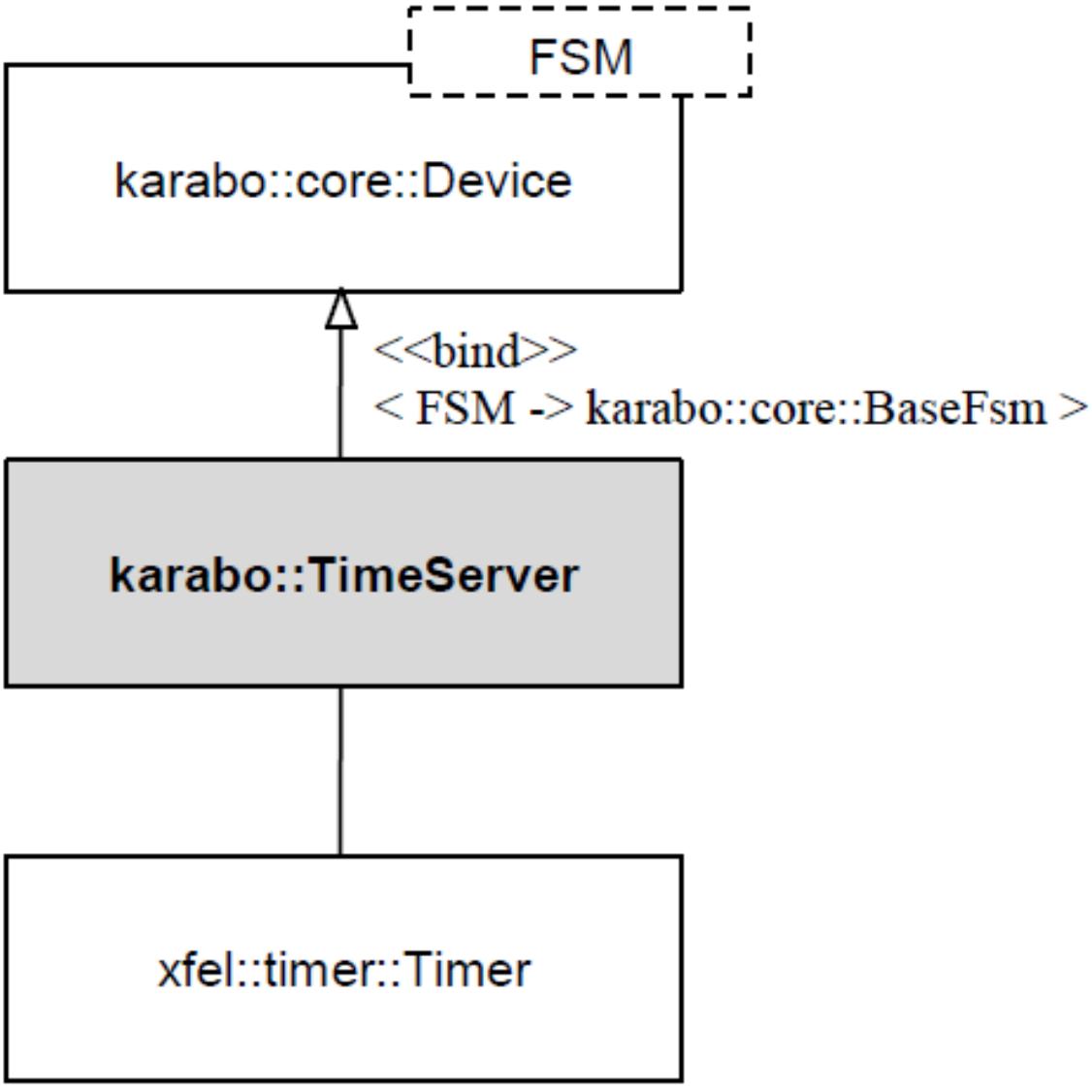

Fig. 1.3: Top level class diagram of TimeServer device.

1.4.2 State Machine

The state machine of the TimeServer device is depicted in Fig. 1.4.

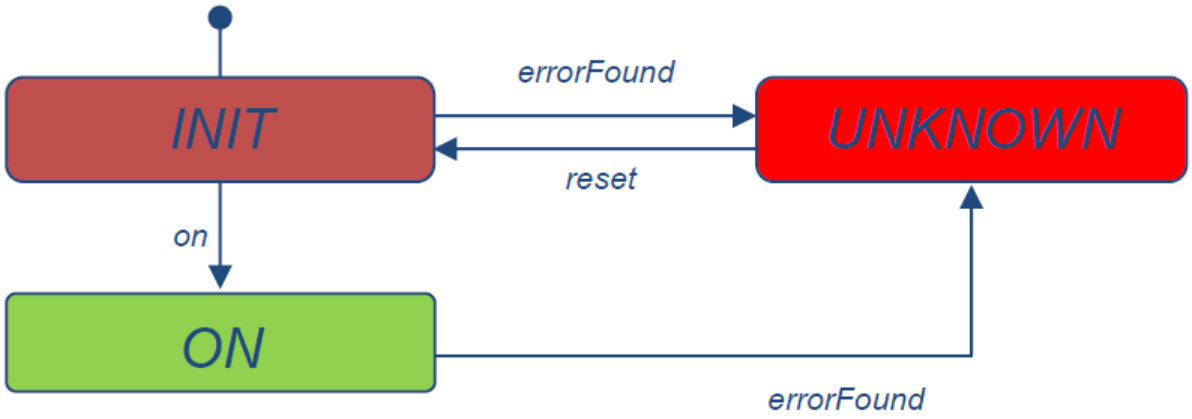

Fig. 1.4: State machine of TimeServer device.

The initial state is the INIT state, in which the TimeServer device either starts a thread to generate internal time ticks, or connects to XFEL timing system, or connects to a remote TCP server. Afterwards, the FSM changes into the ON state, if there was no error during initialization.

In case of errors during initialization or at run time in ON state, the errorFound event is triggered and the FSM changes into the UNKNOWN state. Triggering the reset event changes the FSM back into INIT state.

1.5 TimeServer Class

The class karabo::TimeServer is a thin layer on top of the xfel::timer::Timer class. It defines the methods that are called from the Karabo framework, while it executes the FSM. These methods call methods from the Timer class for further processing, and optionally convert data between the karabo::util::Hash representation, which is used inside Karabo framework, and basic data types, which are used inside xfel::timer::Timer.

The TimeServer class also implements the ITimer interface, which is used by xfel::timer::Timer to provide the time ticks.

1.5.1 Construction and Destruction

When the TimeServer device is created, it registers all slots and signals, and sets up the internal logging.

1.5.2 FSM Methods

For time consuming tasks, the FSM entry exit and action methods usually start worker threads to do the job in the background. This enables the FSM to finish state transitions, so that the state is displayed correctly within the GUI.

INIT State Entry

When entering the INIT state, the time source and period is set, and the TimerServer class sets itself as the class implementing the ITimer interface. Finally, it initializes the Timer. Then the on event is triggered, so that the FSM transits into the ON state.

In case of errors, the errorFound event is triggered to transit the FSM into the UNKNOWN state.

UNKNOWN State Entry

When entering the UNKNOWN state, the Timer is shut down.

1.5.3 Signals

The TimeServer device provides information by emitting the following signal

- signalTimeTick

It is described in the following section in more detail

signalTimeTick

The signal signalTimeTick is emitted each time, the trigger ID changes. It provides the following values:

- trigger ID,
- timestamp in seconds
- timestamp fraction in attoseconds
- average period in microseconds

All values are 64 bit integers. The timestamp is the time of the change of trigger ID in seconds since epoch (1.1.1970). The fractional part is provided in attoseconds, the effective resolution is however depending on the selected ID source, as described in Section *xfel::timer::ITimer interface*. The average period is the time difference between consecutive ID changes, averaged over the last 100 changes.

1.5.4 Expected Parameters

The TimeServer device has a set of expected parameters, which are explained in more detail in the following sections.

source

The expected parameter with key 'source' defines the source for the trigger ID and timestamp. It is specified as a string in URI syntax, as explained in Section *Initialization*. This parameter can only be set before initialization.

id

The expected parameter with key 'id' is set with each change of the trigger ID, so that it always contains the latest valid ID. This parameter is read only.

periodActual

The expected parameter with key 'periodActual' contains the actual average period. The period is calculated as the difference of timestamps of two consecutive ID changes, and is displayed in milliseconds. The period is averaged over the last 100 ID changes. This parameter is read only.

periodSet

The expected parameter with key 'periodSet' defines the setpoint for the period of the internal timing information generation. It is a floating point number in milliseconds. A change of this value has immediate effect. However, due to some filtering of the average (simple P type control loop), it takes some time to get to the new period.

1.6 Timer Class

The class xfel::timer::Timer is the top level class of the core logic, which is explained in more detail int the following section. Main purpose of the Timer class is to send time ticks over the xfel::timer::ITimer interface.

1.6.1 Class Diagram

The class diagram of the core logic is depicted in Fig. 1.5. Application logic is contained in the xfel::timer::Model class. The connection to remote TCP servers is implemented in class xfel::timer::TcpView according to the Model-View-Presenter pattern, as described in[1]. Access to the XFEL timing system is provided by the interface†[0] xfel::timer::ISystem together with the implementation xfel::timer::System. ID and timing information is provided via the interface xfel::timer::ITimer, which is implemented by karabo::TimeServer (the top level Karabo device class).

xfel::timer::Model

The class xfel::timer::Model implements the application logic of the time server. It configures the timing information source, handles trigger ID and timestamp updates and provides the ID and timestamps via time ticks.

The xfel::timer::Timer class interacts with the core logic only by calling public methods of the model.

xfel::timer::Presenter

The class xfel::timer::Presenter implements the behavior on the interface to remote TCP servers, such as opening and closing a connection and acting on received data.

xfel::timer::TcpView

The class xfel::timer::TcpView interacts with the TCP stack to connect to remote TCP servers and to receive data. It uses events towards the presenter to indicate the reception of new data and to indicate possible errors during listening.

[1] M. Alles, D. Crosby, C. Erickson, B. Harleton, M. Marsiglia, G. Pattison, C. Stienstra, "Presenter First: Organizing Complex GUI Applications for Test-Driven Development," Agile 2006, Minneapolis, July 2006.

[0] In C++, interfaces are abstract classes that contain only pure virtual methods.

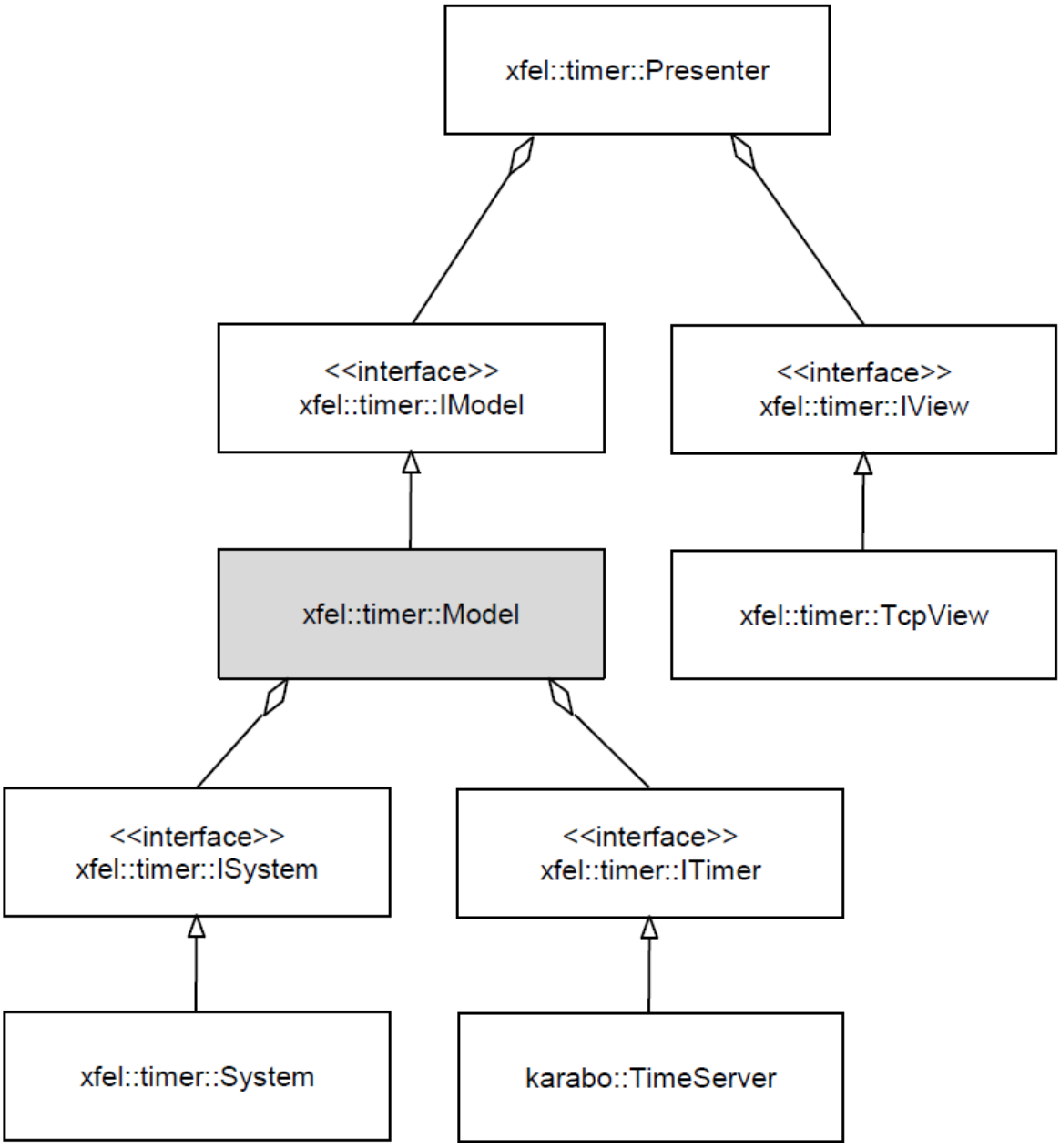

Fig. 1.5: Class diagram of the TimeServer core logic.

xfel::timer::System

The xfel::timer::System class implements the interface xfel::timer::ISystem to give access to the XFEL timing system.

In order to get timing information from the XFEL timing system, the board need to be configured to raise an interrupt SIGUSR1 on a train ID change. This is done using the `ioctl()` system call on the device file of the timing board, which is usually `/dev/x2timersn`, where n is the slot number of the board.

> **Warning:** The `ioctl()` related structures and macros are defined by 2 files `pciedev_io.h` and `x1timer_io.h`. which are part of the Linux kernel driver for the timing board. These files always need to match the ones from the driver installed on the μTCA crate, where the TimeServer device is run.

1.6.2 Initialization

The steps performed during initialization depends on the selected timing source. The source is specified using the generic URI syntax `scheme:hierarchical_part`. The following values are supported:

- `local:internal` for internally generated timing information,
- `local:x2timer` for XFEL timing, using a μTCA x2timer board
- `tcp://<host:port>` for timing from remote TCP servers.

If no scheme is specified, `local` is assumed.

For internally generated timing, a thread is created that simply counts upwards with the specified period. For XFEL timing, an interrupt handler for `SIGUSR1` is registered to read out timing information from the hardware. For timing from remote TCP servers, a TCP channel is opened to the specified host.

After the device is initialized, it is no longer possible to switch the timing source.

The methods to initialize the core are listed in Table 1.1. Before starting operation, the methods setIdSource(), setTimer(), and setPeriod() need to be called to inject the necessary run time parameters. Then initialize() is called to actually start operation.

Table 1.1: Initialization methods

Method	Description
setIdSource(sourceUri)	sets the source of timing information (in URI form *scheme:hierarchical_part*)
setPeriod(period)	sets the time period for internal timing
setTimer(timer)	sets the receiver of the time tick (a class implementing the xfel::timer::ITimer interface)
initialize()	starts operation according to specified source

1.6.3 Operation

Once the core is initialized, it automatically starts generating time ticks, which are provided over the xfel::timer::ITimer interface. For the TimeServer device, the karabo::TimeServer class itself implements this interface, so that the ticks are available on Karabo level, where a tick is converted into a Karabo signal emitted to the message broker.

xfel::timer::ITimer interface

The xfel::timer::ITimer interface defines one method tick(), with the trigger ID, timestamp as seconds and fractional seconds, and period as parameters.

The timestamp is the number of seconds since UNIX epoch, which is 1.1.1970 UTC. The fractional part of the timestamp is specified in attoseconds, however the resolution is depending on the selected ID source. If the ID is generated by the XFEL timing system, then the associated timestamp is provided only with microseconds resolution. For all other ID sources, the system call clock_gettime() is used, which returns a time with nanosecond resolution.

The period is specified in microseconds, and is the averaged time difference between consecutive trigger timestamps

1.6.4 Shutdown

The steps performed during shutdown depends on the specified timing source. For internal timing, the thread is terminated. For XFEL timing, the signal handler for SIGUSR1 is unregistered. For timing from remote TCP servers, the TCP connection is closed.

The method to shut down the core is listed in Table 1.2.

Table 1.2: Shutdown method.

Method	Description
shutdown()	stops operation according to specified source

1.7 Logging

Logging for other classes than those derived from the karabo::core::Device base class is provided by the Log class. Macros simplify the process of creating log messages, and a handler needs to be installed, which will process the log message

1.7.1 Macros

The Log class has an internal output stream, which can be filled using the macros

- LOG_ERROR
- LOG_WARN
- LOG_INFO
- LOG_DEBUG

as in the following example:

```
LOG_DEBUG << "some debug message";
```

The macro instantiates the Log class, fills the internal output stream, and finally destructs the Log class, which results in calling the handler, if any was installed, with the string representation of the output stream.

1.7.2 Handler

In order to forward a log message from the Log class to the Karabo framework, the TimeServer installs a handler. The handler is defined as:

```
void TimeServer::logEvent(int level, const char* message);
```

The handler is registered with the static method karabo::Log::setHandler(). Since it is a class method of the TimeServer, the function pointer needs to be created using the templated method CreateMessageHandler():

```
xfel::timer::Log::setHandler(
    xfel::timer::CreateMessageHandler<TimeServer>(
        this, &TimeServer::logEvent
    )
);
```

1.8 Integration into Karabo Devices

The Karabo base device has been extended in order to make use of a TimeServer. It also requires a specific instance name for the TimeServer device.

1.8.1 TimeServer Instantiation

If the TimeServer device is required as a source for train ID and timestamp, it needs to be initialized with a fixed instance name **Karabo_TimeServer**.

1.8.2 Karabo Device Configuration

If a Karabo device is configured to use the TimeServer device, then it subscribes to the signal 'signalTimeTick' of the device named 'Karabo_TimeServer'. The associated slot saves this data locally and also calls a hook onTimeUpdate(), which a derived Karabo device can override.

1.8.3 Calculating ID and Timestamp

When an expected parameter is set without using the timestamp argument, then the missing timestamp and train ID is calculated based on the values provided from the last TimeServer update (see karabo::core::Device::getActualTimestamp()). First, the difference between the current time and the time of the last train ID change is calculated. Then, the difference is compared with the period, and if the difference is larger (due to missing TimeServer updates), then the train ID is interpolated based on the given average period.

1.9 TCP Server Protocol

The TCP server protocol uses simple ASCII string messages‡[0]. The only information contained in a message is an integer number specifying the trigger ID. On the TimeServer side, the function `strtoull()` is used to convert the ASCII string into an integer for further processing.

Warning: The time stamp that is associated with the received train ID is the time, when the message was received. This results in a slight inaccuracy, which depends on network latency.

[0] It was used in a setup at Trieste Synchroton, where an AdqDigitizer and TimeServer device were run together. The TimeServer device received bunch ID from the TANGO domain via a TCP server using this protocol.

1.10 Abbreviations and acronyms

Core	Main logic part of the TimeServer device
MVP	Model-View-Presenter design pattern
Timing System	Delivers reference clock, trigger synchronous to a train of XFEL pulses, and ID of pulse train.
Trigger ID	The ID of the event that triggers activity. For XFEL, this ID is called Train ID, other facilities may call this macro pulse or bunch ID.

1.11 References

CHAPTER 2

Train ID Workflow

The workflow presented in Fig. 2.1 shows how the trainId information is provided to the karabo devices. The time-server is a critical timing-reference for the devices which do not get the trainId from DESY accelerator machine directly through the hardware.

Fig. 2.1

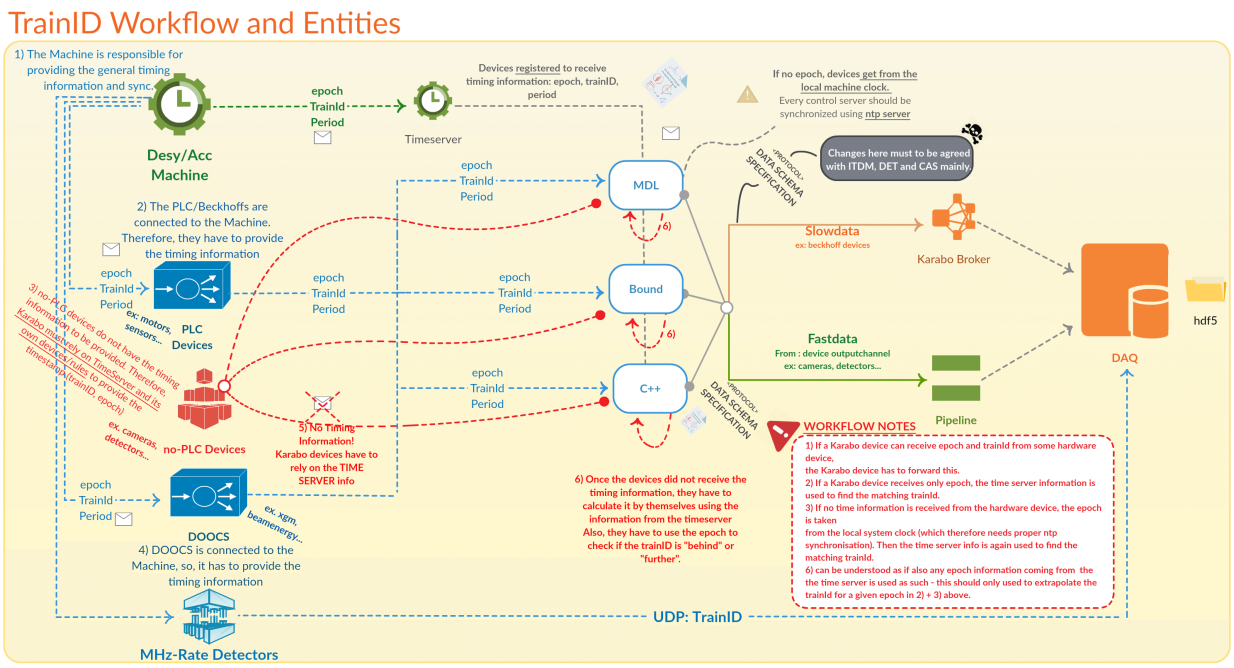

Fig. 2.1: TrainId Workflow and Entities.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search