Document Title	Specification of Time Synchronization for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	880

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-10

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-10-31	18-10	AUTOSAR Release Management	• Minor changes and bugfixes • Editorial changes
2018-03-29	18-03	AUTOSAR Release Management	• Class design changed to ensure type safety • API related sections moved from chapter 7 to chapter 8 • Minor changes and bugfixes
2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview

Time Synchronization between different applications and/or ECUs is of paramount importance when correlation of different events across a distributed system is needed, either to be able to track such events in time or to trigger them at an accurate point in time.

For this reason, a Time Synchronization API is offered to the Application, so it can retrieve the time information synchronized with other Entities / ECUs.

For the format, message sequences and semantics of the time synchronization protocols to use, please refer to the Protocol Requirements Specicification (PRS) of the AUTOSAR Time synchronization Protocol (see [1]).

The Time Synchronization functionality is then offered by means of different "Time Base Resources" (from now on referred to as TBR) which are present in the system via a pre-build configuration.

These TBRs are classified in different types. These types have an equivalent design to the types of the time bases offered in the Synchronized Time Base Manager specification [2] (from now on referred to as StbM). The classification is the following:

- Synchronized Master Time Base
- Offset Master Time Base
- Synchronized Slave Time Base
- Offset Slave Time Base
- Pure Local Time Base

As in StbM, the TBRs offered by the Time Synchronization module (TS from now on), are also synchronized with other Time Bases on other nodes of a distributed system, with the exception of the Pure Local Time Bases.

The Application will have access to a different specialized class implementation for each TBR.

The TBRs are offered as resources through services as described in ara::com [3] design and therefore it is adopting the following architectural design patterns of ara::com:

- proxy: Similar to the ara::com Service proxy skeleton pattern, TS provides a service proxy pattern, omitting the skeleton part.
- proxy Methods: Similar to the ara::com proxy Methods pattern, TS uses a Methods pattern also adhering to the asynchronous Future pattern.

This architectural design puts the Time Synchronization design apparently in a frontal conflict when talking about avoiding latencies, since the latter are inherently added by the asynchronous behavior of the design pattern of the ara::com API.

To avoid having the latency present, yet being consistent with the ara::com design pattern, instead of offering a remote resource handler, a local handler will be provided.

From this handle, the Application will be able to inquire about the type of Time Base offered (which shall be one of the five types presented above) to then obtain a specialized class implementation for that type of Time Base. From this handle, the Application will also be able to create a timer directly.

The TS module itself does not provide means to synchronize TBRs to Time Bases on other nodes and/or ECUs like network time protocols or time agreement protocols.

An implementation of TBRs may have a dedicated cyclic functionality, which retrieves the time information from the Time Synchronization Ethernet module or alike to synchronize the TBRs.

The Application consumes the time information provided and managed by the TBRs. Therefore, the TBRs serve as Time Base brokers, offering access to Synchronized Time Bases. By doing so, the TS module abstracts from the "real" Time Base provider.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Time Synchronization module that are not included in the [4, AUTOSAR glossary].

2.1 Acronyms and Abbreviations

Abbreviation / Acronym:	**Description:**
StbM	Synchronized Time Base Manager
TS	Time Synchronization
TBR	Time Base Resource
NTP	Network Time Protocol
PTP	Precision Time Protocol
gPTP	Generalized Precision Time Protocol
Timesync	Time Synchronization (Refers to the action of Synchronizing the Time by means of a time synchronization protocol/bus/messages)
TSP	A bus specific Time Synchronization Provider
UTC	Coordinated Universal Time
OS	Operating System
DLS	Day Light Saving, also know as Daylight Saving Time (abbreviated DST), is the practice of advancing clocks during summer months so that evening daylight lasts longer, while sacrificing normal sunrise times. Typically, regions that use daylight saving time adjust clocks forward one hour close to the start of spring and adjust them backward in the autumn to standard time

2.2 Definitions

2.2.1 Clock

Definition: A Clock refers to the unit conformed by the combination of a Time Base (either synchronized against an external source or not) and a hardware capable of changing cyclically the electric state of its output (e.g. toggling between two different voltage levels). The frequency of such electric state changes can be adjustable. This hardware could be e.g. part of a microcontroller, or an external electronic component. Likewise the Synchronized Time Base information can be acquired from an external source like a RTC, GPS, Ethernet, etc.

Therefore when talking about a Clock we may refer to either its quality (e.g. rate, accuracy, etc.) or to the Time Base it holds (e.g. time information relative to the Global Position, daylight, etc.) depending on the context that holds the term.

2.2.2 Global Time Master

Definition: A Global Time Master is the global owner and origin for a certain Time Base and on the top of the Time Base hierarchy for that Time Base.

2.2.3 Time Base

Definition: A Time Base is a unique time entity characterized by:

- Progression of time, which denotes how time progresses, i.e. the rate (which for instance, might be derived from a local quartz oscillator) and absolute changes of the time value at certain point in times (e.g. effects of offset correction in NTP).
- Ownership, which denotes who is the owner of the Time Base. A distributed NTP Time Base e.g. has multiple owners and the progression of time with respect to rate and offset corrections is a result of involving a subset of NTP nodes.
- Reference to the physical world, i.e. whether the Time Base is a relative Time Base counting local operation time of an ECU or representing an absolute time like UTC. A Time Base can have more than one reference, e.g. it can be a relative time which, in combination with an offset value, also represents an absolute time.

Examples of Time Bases in vehicles are:

- Absolute, which is based on a GPS based time.
- Relative, which represents the accumulated overall operating time of a vehicle, i.e. this Time Base does not start with a value of zero whenever the vehicle starts operating.
- Relative, starting at zero when the ECU begins its operation.

A Time Base implies the availability of a Clock.

Special case "Pure Local Time Base":

A Pure Local Time Base is a Time Base with a local scope as it is neither propagated to other nodes nor received from other nodes. A Pure Local Time Base will only locally be set and read. It is therefore possible to have multiple Pure Local Time Bases with the same Time Domain number in various nodes in parallel. A Pure Local Time Base behaves like a Synchronized Time Base since it progresses in time, however it is not synchronized via TSP modules. Pure Local Time Bases behaving like an Offset Time Bases are not supported.

2.2.4 Synchronized Time Base

Definition: A Synchronized Time Base is a Time Base existing at a processing entity (actor / processor / node of a distributed system) that is synchronized with Time Bases at different processing entities. A Synchronized Time Base can be achieved by time protocols or time agreement protocols that derive the Synchronized Time Base in a defined way from one or more physical Time Bases (e.g. Network Time Protocol (NTP)). The synchronization will apply to the clock rate and optionally also to the Time Base absolute value.

A Synchronized Time Base allows synchronized action of the processing units. Synchronized Time Bases are often called "Global Time".

More than one Synchronized Time Base can exist at one processing unit, e.g. a NTP node will have the Synchronized Time Base retrieved from NTP in the network cluster but might also have a Synchronized Time Base derived from the time provided by a UTC time server (which is based on a set of atomic clocks). Both Synchronized Time Bases will probably have slightly different rates, and there is no relationship defined between their absolute values.

2.2.5 Offset Time Base

Definition: An Offset Time Base is a Time Base existing at a processing entity (actor / processor / node of a distributed system). An Offset Time Base depends on one particular Synchronized Time Base, therefore it is synchronized with the same Time Base Source as its underlying TBR.

An Offset Time Base holds an offset value relative to the Time Base of its underlying Synchronized TBR. Therefore, it provides to the Application a time base with a value of its underlying Synchronized TBR plus the Offset value it holds. Since an Offset Time Base receives its time value from the same TSP as its underlying Synchronized TBR, it will present the same rate deviation and correction properties.

2.2.6 Time Base Provider

Definition: A Time Base Provider is the role that a TSP module takes for a given Time Base. Therefore a TSP module can contain one or more Time Base providers. Time Base providers are either of type importer or exporter, whereas an importer acts as Time Slave and an exporter acts as Time Master. A Time Gateway consists of one Time Base importer and one or more Time Base exporters for a given Time Base. In order to limit the terminology, importers are denoted as slaves and exporters are denoted as masters.

2.2.7 Time Communication Port

Definition: A Time Communication Port is a physical communication interface (in Classic Platform coverable by the item: Physical Connector) at an ECU which is used to transport time information.

2.2.8 Time Communication Service

Definition: A Time Communication Service is an interaction between Time Bases which is performed by Time Base providers. Time communication services are message based between a Time Master and one or more Time Slaves or between one Time Slave and his Time Master.

The following figure shows a network topology example and the related terminology.

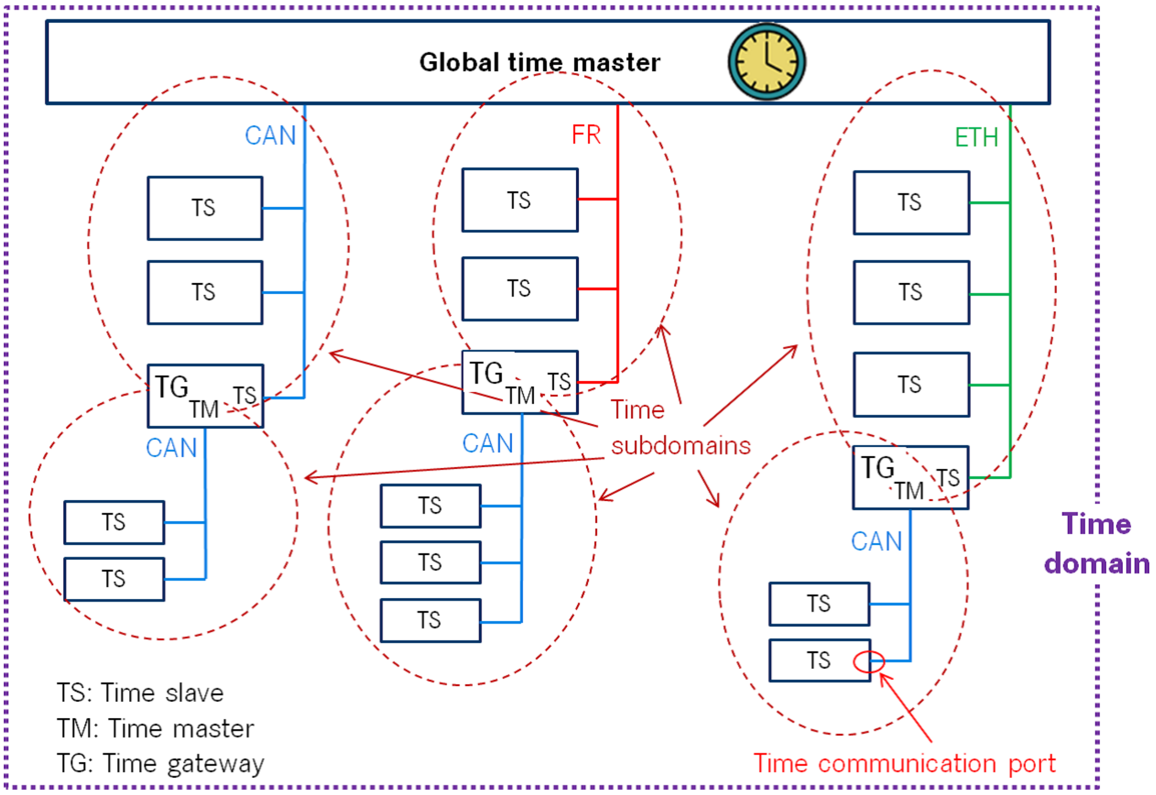


Figure 2.1: Terminology Example

2.2.9 Time Base Application

1. **Active Application**
   This kind of Application autonomously calls the TS either:
   - To read time information from the TBRs
   - To update the Time Base maintained by a TBR, according to application information.
2. **Triggered Application**
   This feature will be provided at a later release/version of the TS.
3. **Notification Application**
   This feature will be provided at a later release/version of the TS.

2.2.10 Time Domain

Definition: A Time Domain denotes which components (e.g. nodes, communication systems) are linked to a certain Time Base. A Time Domain can contain zero or more Time Sub-Domains. If the timing hierarchy of a Time Domain contains no Time Gateways, i.e. all nodes are connected to the same bus system, then there is no dedicated Time Sub-Domain which otherwise would be equal to the Time Domain itself.

2.2.11 Time Gateway

Definition: A Time Gateway is a set of entities where one entity is acting as Time Slave for a certain Time Base. The other (one or more) entities are acting as Time Masters which are distributing this Time Base to sets of Time Slaves. A Timesync ECU can contain multiple Time Gateways.

2.2.12 Time Hierarchy

Definition: The Time Hierarchy describes how a certain Time Base is distributed, starting at the Global Time Master and being distributed across various Time Gateways (if present) to various Time Slaves.

2.2.13 Time Master

Definition: A Time Master is an entity which is the master for a certain Time Base and which propagates this Time Base to a set of Time Slaves within a certain segment of a communication network, being a source for this Time Base.

If a Time Master is also the owner of the Time Base then he is the Global Time Master. A Time Gateway typically consists of one Time Slave and one or more Time Masters. When mapping time entities to real ECUs it has to be noted, that an ECU could be Time Master (or even Global Time Master) for one Time Base and Time Slave for another Time Base.

Special case "Pure Local Time Master":

A Pure Local Time Master is an entity which is the master of a Pure Local Time Base and which therefore does not propagate this Time Base to any Time Slave.

2.2.14 Time Slave

Definition: A Time Slave is an entity, which is the recipient for a certain Time Base within a certain segment of a communication network, being a consumer for this Time Base.

2.2.15 Time Sub-domain

Definition: A Time Sub-Domain denotes which components (e.g. nodes) are linked to a certain Time Base, whereas the scope is limited to one communication bus.

2.2.16 Timesync ECU

Definition: A Timesync ECU is an ECU which is part of a Time Domain by containing one or more Time Slaves or Time Masters.

2.2.17 TSP Module

Definition: TSP modules are bus specific modules to receive or transmit time information on bus systems by applying bus specific mechanisms. A Timesync module can serve multiple communication buses of the same type.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Protocol Requirements on Time Synchronization for Adaptive Platform
AUTOSAR_PRS_TimeSync

[2] Specification of Synchronized Time-Base Manager
AUTOSAR_SWS_SynchronizedTimeBaseManager

[3] Explanation of ara::com API
AUTOSAR_EXP_ARAComAPI

[4] Glossary
AUTOSAR_TR_Glossary

[5] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[6] Functional Cluster Shortnames
AUTOSAR_TR_FunctionalClusterShortnames

[7] Requirements on Time Synchronization for Adaptive Platform
AUTOSAR_RS_TimeSync

[8] ISO/IEC 14882:2011, Information technology – Programming languages – C++
http://www.iso.org

[9] Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7
http://pubs.opengroup.org/onlinepubs/9699919799/

[10] Specification of Time Synchronization over Ethernet
AUTOSAR_SWS_TimeSyncOverEthernet

[11] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [5, SWS BSW General], which is also valid for TS.

Thus, the specification SWS BSW General shall be considered as additional and required specification for TS.

4 Constraints and assumptions

4.1 Limitations

The Time Synchronization module is bound to Adaptive Platform Systems.

For the TS, it is necessary that at least there is one TBR in the system, otherwise no functionality can be provided to the `Adaptive Applications` (i.e. the `Adaptive Applications` should not get any handle for Time Base Resources).

API design is not fully compliant to Adaptive Platform Design Rules which request the usage of UpperCamelCase.

4.1.1 Configuration

Please refer to the corresponding model elements.

4.1.2 Time Gateway

Time Gateway functionality is currently not in scope of the Time Synchronization module for the Adaptive Platform.

4.1.3 Out of Scope

Errors, which occurred during Global Time establishment and which are not caused by the module itself (i.e. loss of PTP global time is not an issue of the TS but of the TSP modules) are out of the scope of this module.

4.2 Applicability to car domains

The concept is targeted at supporting time-critical automotive applications. This does not mean that the concept has all that is required by such systems though, but crucial timing-related features which cannot be deferred to implementation are considered.

4.3 Recommendation

In the case where the TSP is based on Ethernet, the protocol to be used is defined in the PRS (see [1]).

5 Dependencies to other modules

TS is part of the ara::tsync [6] namespace.

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Time Synchronization for Adaptive Platform [7] and links to the fulfillment of these.

Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_TS_00001]	The configuration shall allow the TS module to support different roles for a Time Base	[SWS_TS_00001] [SWS_TS_00008] [SWS_TS_00009] [SWS_TS_00088] [SWS_TS_00091] [SWS_TS_00092] [SWS_TS_00094] [SWS_TS_00103] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00109] [SWS_TS_00110] [SWS_TS_00112] [SWS_TS_00113] [SWS_TS_00114] [SWS_TS_00115] [SWS_TS_00132] [SWS_TS_00133] [SWS_TS_00134] [SWS_TS_00150] [SWS_TS_00152] [SWS_TS_00154]
[RS_TS_00002]	The Implementation of Time Synchronization, independently of the Role it is acting as, shall always maintain its own Time Base	[SWS_TS_00023] [SWS_TS_00029] [SWS_TS_00037] [SWS_TS_00038] [SWS_TS_00039] [SWS_TS_00040] [SWS_TS_00041] [SWS_TS_00042] [SWS_TS_00091] [SWS_TS_00092] [SWS_TS_00097] [SWS_TS_00102] [SWS_TS_00108] [SWS_TS_00150] [SWS_TS_00152] [SWS_TS_00154]
[RS_TS_00003]	The Implementation of Time Synchronization shall initialize the Local Time Base with zero at startup	[SWS_TS_00006]
[RS_TS_00004]	The Implementation of Time Synchronization shall initialize the Global Time Base with a configurable startup value.	[SWS_TS_00135]
[RS_TS_00005]	The Implementation of Time Synchronization shall allow customers to have access to the Synchronized Time Base	[SWS_TS_00002] [SWS_TS_00014] [SWS_TS_00022] [SWS_TS_00031] [SWS_TS_00090] [SWS_TS_00128] [SWS_TS_00151] [SWS_TS_00153] [SWS_TS_00168]
[RS_TS_00007]	The Implementation of Time Synchronization shall synchronize the Time Base of a Time Slave, on reception of a Time Master value	[SWS_TS_00019] [SWS_TS_00037] [SWS_TS_00042]

Requirement	Description	Satisfied by
[RS_TS_00009]	The Implementation of Time Synchronization shall maintain the synchronization status of a Time Base	[SWS_TS_00007] [SWS_TS_00011] [SWS_TS_00012] [SWS_TS_00020] [SWS_TS_00024] [SWS_TS_00025] [SWS_TS_00027] [SWS_TS_00028] [SWS_TS_00030] [SWS_TS_00032] [SWS_TS_00033] [SWS_TS_00034] [SWS_TS_00035] [SWS_TS_00036] [SWS_TS_00067] [SWS_TS_00087] [SWS_TS_00139] [SWS_TS_00140] [SWS_TS_00141]
[RS_TS_00010]	The Implementation of Time Synchronization shall allow customer on master side to set the Global Time	[SWS_TS_00013] [SWS_TS_00018] [SWS_TS_00098] [SWS_TS_00099] [SWS_TS_00100] [SWS_TS_00101] [SWS_TS_00102] [SWS_TS_00103] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00108] [SWS_TS_00110]
[RS_TS_00011]	The Implementation of Time Synchronization shall allow customers on master side to trigger time transmission by the TSP module	[SWS_TS_00103] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00110]
[RS_TS_00012]	The Implementation of Time Synchronization shall allow customers and TSP modules to read the offset value of an Offset Time Base	[SWS_TS_00017] [SWS_TS_00095] [SWS_TS_00114]
[RS_TS_00013]	The Implementation of Time Synchronization shall allow the customers and TSP modules to set the offset value of an Offset Master Time Base	[SWS_TS_00016] [SWS_TS_00055] [SWS_TS_00056] [SWS_TS_00057] [SWS_TS_00058] [SWS_TS_00059] [SWS_TS_00060] [SWS_TS_00112] [SWS_TS_00113]
[RS_TS_00014]	The Implementation of Time Synchronization shall allow customers to read User Data propagated via the TSP modules.	[SWS_TS_00119] [SWS_TS_00120] [SWS_TS_00144]
[RS_TS_00015]	The Implementation of Time Synchronization shall allow customers to set User Data propagated via the TSP modules.	[SWS_TS_00021] [SWS_TS_00088]
[RS_TS_00016]	The Implementation of Time Synchronization shall notify customers about status events	[SWS_TS_00064] [SWS_TS_00068]
[RS_TS_00017]	The Implementation of Time Synchronization shall notify customers about elapsed pre-defined time span.	[SWS_TS_00064] [SWS_TS_00068]

Requirement	Description	Satisfied by
[RS_TS_00018]	The Implementation of Time Synchronization shall support rate correction	[SWS_TS_00029] [SWS_TS_00037] [SWS_TS_00038] [SWS_TS_00039] [SWS_TS_00040] [SWS_TS_00041] [SWS_TS_00042] [SWS_TS_00043] [SWS_TS_00044] [SWS_TS_00045] [SWS_TS_00046] [SWS_TS_00047] [SWS_TS_00048] [SWS_TS_00049] [SWS_TS_00050] [SWS_TS_00051] [SWS_TS_00052] [SWS_TS_00053] [SWS_TS_00054] [SWS_TS_00061] [SWS_TS_00062] [SWS_TS_00063] [SWS_TS_00070] [SWS_TS_00071] [SWS_TS_00084] [SWS_TS_00109] [SWS_TS_00142]
[RS_TS_00019]	The Implementation of Time Synchronization shall support damping offset correction	[SWS_TS_00042] [SWS_TS_00045] [SWS_TS_00050] [SWS_TS_00051] [SWS_TS_00052] [SWS_TS_00054] [SWS_TS_00056] [SWS_TS_00057] [SWS_TS_00058] [SWS_TS_00071]
[RS_TS_00021]	The Implementation of Time Synchronization shall provide interfaces to query the synchronization status	[SWS_TS_00005] [SWS_TS_00035] [SWS_TS_00086] [SWS_TS_00118] [SWS_TS_00119] [SWS_TS_00120] [SWS_TS_00121] [SWS_TS_00122] [SWS_TS_00123] [SWS_TS_00124] [SWS_TS_00125] [SWS_TS_00126] [SWS_TS_00127] [SWS_TS_00129] [SWS_TS_00130] [SWS_TS_00131] [SWS_TS_00136] [SWS_TS_00137] [SWS_TS_00138] [SWS_TS_00143] [SWS_TS_00145] [SWS_TS_00149]
[RS_TS_00022]	The Implementation of Time Synchronization shall support custom clocks	[SWS_TS_00001] [SWS_TS_00078] [SWS_TS_00132] [SWS_TS_00188] [SWS_TS_00195]
[RS_TS_00023]	The Implementation of Time Synchronization shall offer interfaces able to handle std::chrono data types.	[SWS_TS_00014] [SWS_TS_00015] [SWS_TS_00078] [SWS_TS_00157]
[RS_TS_00026]	The Implementation of Time Synchronization shall provide to the customers a specific API per type of Time Base Resource	[SWS_TS_00031] [SWS_TS_00065] [SWS_TS_00066] [SWS_TS_00069] [SWS_TS_00072] [SWS_TS_00085] [SWS_TS_00088] [SWS_TS_00090] [SWS_TS_00093] [SWS_TS_00094] [SWS_TS_00096] [SWS_TS_00098] [SWS_TS_00099] [SWS_TS_00100] [SWS_TS_00101] [SWS_TS_00102] [SWS_TS_00103] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00108] [SWS_TS_00109] [SWS_TS_00110]

Requirement	Description	Satisfied by
		[SWS_TS_00111] [SWS_TS_00112] [SWS_TS_00113] [SWS_TS_00114] [SWS_TS_00115] [SWS_TS_00116] [SWS_TS_00117] [SWS_TS_00124] [SWS_TS_00128] [SWS_TS_00133] [SWS_TS_00134] [SWS_TS_00151] [SWS_TS_00153] [SWS_TS_00188]

7 Functional specification

The functional behavior is described under the following specific contexts:

- Startup Behavior
- Construction Behavior (Initialization)
- Shutdown Behavior
- Normal Operation
- Error Handling
- Error Classification
- Version Check

7.1 General Overview of TS

For the Adaptive Platform, three different technologies were considered to fulfill such Time Synchronization requirements. These technologies were:

- StbM of the Classic Platform
- Library chrono - either std::chrono (C++11) or boost::chrono [8]
- The Time posix interface [9]

After an analysis of the interfaces of these modules and the Time Synchronization features they cover, the motivation is to design a Time Synchronization API that provides a functionality wrapped around the StbM module of the Classic Platform, but with a std::chrono like flavor.

The following table shows the interfaces provided to the Application by means of this API and their equivalent interface in StbM.

Time Synchronization API - AP	***StbM - CP***
now	StbM_GetCurrentTime
calculateTimeDiff	StbM_GetCurrentTimeDiff
setTime	StbM_SetGlobalTime
updateTime	StbM_UpdateGlobalTime
setUserData	StbM_SetUserData
setOffset	StbM_SetOffset
getOffset	StbM_GetOffset

▽

△

getRateDeviation	StbM_GetRateDeviation
setRateCorrection	StbM_SetRateCorrection
timeLeap (attribute of the TimeBase Status class)	StbM_GetTimeLeap
getTimeBaseStatus	StbM_GetTimeBaseStatus
startTimer (under methods namespace)	StbM_StartTimer
updateCounter (attribute of the TimeBase Status class)	StbM_GetTimeBaseUpdateCounter
This information is accessible via the Status flags	StbM_GetMasterConfig

Table 7.1: Interface comparison between TS and STBM

The TS design offers five different Time Base interfaces to the Application. Each Time Base interface is corresponding to a particular Time Base type. Time Base types can be any of the following - as explained in chapter 1:

- Master Time Bases
  - Synchronized Master
  - Offset Master
- Slave Time Bases
  - Synchronized Slave
  - Offset Slave
- Pure Local Time Base

Time Synchronization functionality is offered via the different TBRs.

The TS design provides the Application with a specific set of interfaces, according to the type of TBR. In this way, each type of TBR offers specific functionality that is not offered by other TBR or -where applies- it overrides certain functionality according to specific needs or requirements to be fulfilled by the given type of TBR.

Figure 7.1 depicts the Class Diagram of the Time Base Resources:

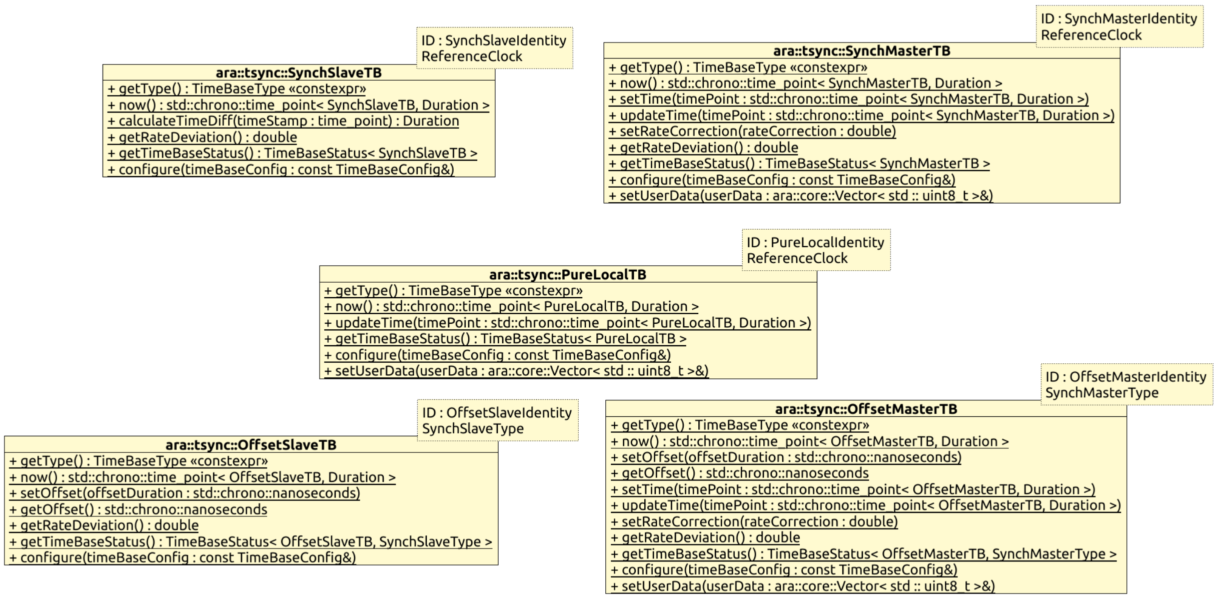

ID : SynchSlaveIdentity
ReferenceClock
ara::tsync::SynchSlaveTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< SynchSlaveTB, Duration >
+ calculateTimeDiff(timeStamp : time_point) : Duration
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< SynchSlaveTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)
ID : SynchMasterIdentity
ReferenceClock
ara::tsync::SynchMasterTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< SynchMasterTB, Duration >
+ setTime(timePoint : std::chrono::time_point< SynchMasterTB, Duration >)
+ updateTime(timePoint : std::chrono::time_point< SynchMasterTB, Duration >)
+ setRateCorrection(rateCorrection : double)
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< SynchMasterTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)
ID : PureLocalIdentity
ReferenceClock
ara::tsync::PureLocalTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< PureLocalTB, Duration >
+ updateTime(timePoint : std::chrono::time_point< PureLocalTB, Duration >)
+ getTimeBaseStatus() : TimeBaseStatus< PureLocalTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)
ID : OffsetSlaveIdentity
SynchSlaveType
ara::tsync::OffsetSlaveTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< OffsetSlaveTB, Duration >
+ setOffset(offsetDuration : std::chrono::nanoseconds)
+ getOffset() : std::chrono::nanoseconds
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< OffsetSlaveTB, SynchSlaveType >
+ configure(timeBaseConfig : const TimeBaseConfig&)
ID : OffsetMasterIdentity
SynchMasterType
ara::tsync::OffsetMasterTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< OffsetMasterTB, Duration >
+ setOffset(offsetDuration : std::chrono::nanoseconds)
+ getOffset() : std::chrono::nanoseconds
+ setTime(timePoint : std::chrono::time_point< OffsetMasterTB, Duration >)
+ updateTime(timePoint : std::chrono::time_point< OffsetMasterTB, Duration >)
+ setRateCorrection(rateCorrection : double)
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< OffsetMasterTB, SynchMasterType >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)

Figure 7.1: Class Diagram of the Time Base Resources.

[SWS_TS_00005] ⌈ Every Time Base Resource present in the system shall be able to generate a Status object to be passed to the Application under request. (i.e. `TimeBaseStatus` object). ⌋*(RS_TS_00021)*

This `TimeBaseStatus` object contains information relevant to the Time Base Resource it is related to, like status flags, counter of the times the TBR has been updated, time leap information (possibly generated during the last synchronization of the Time Base Resource), etc.

The method `isStatusFlagActive()` of the class `TimeBaseStatus` returns the state of the flag that corresponds to the enumeration parameter. The `StatusFlag` enum provides the semantical meaning.

[SWS_TS_00144] ⌈ In case the Application wants to retrieve the User Data, it shall do it by means of the `getUserData()` method of a `TimeBaseStatus` instance. Please refer to figure 4, as well as to section 7.1.1.1. ⌋*(RS_TS_00014)*

7.1.1 Base functionality of every Time Base class

Each TimeBase provides at least four member functions:

- getType
- getRateDeviation
- now
- getTimeBaseStatus

This chapter describes briefly the general functionality provided by these methods. Details about the usage and / or behavior of these core methods are given in following chapters.

7.1.1.1 getType method

For any type of TBR, the Application might be interested in querying for the TBR's type.

[SWS_TS_00132] ⌈ For all types of TBR, the `getType` method shall return a `TimeBaseType` enumeration value, which denotes its TBR type. ⌋*(RS_TS_00001, RS_TS_00022)*

7.1.1.2 getRateDeviation method

The `getRateDeviation` method returns, if already calculated, the rate deviation of a given TBR against the time source it is synchronized to.

More detailed information about this method is given in the following chapters and in chapter 8.

7.1.1.3 now method

The `now` method is very likely the most commonly used by the Applications that interact with TS. This method returns the *time_point* of a TBR at the time at which it is called. The *time_point* information returned by this method should be type safe.

[SWS_TS_00157] ⌈ An application, obtaining *time_points* from different TBRs, shall not be able to perform arithmetic operations on them. ⌋*(RS_TS_00023)*

More detailed information about this method is given in the further chapters and on Chapter 8.

7.1.1.4 getTimeBaseStatus method

The `getTimeBaseStatus` method provides an instance of a `TimeBaseStatus` class, which contains all the status-related information at time of calling. The `TimeBaseStatus` instance and the information it offers, are bound to the type of the TBR from which this method has been called. To be able to have a `TimeBaseStatus` bound to the type of the TBR, this method has to be templated, and therefore it has to be defined and implemented on each of the classes of the different TBRs.

Additionally to this and since the class `TimeBaseStatus` contains status information and *time_point* strictly bounded to a specific type of TBR, this class is implemented as a template. As a reference to the class, see Figure 7.1 and for the methods of this class please refer to chapter 8.

7.1.2 Status Flags of the TBRs

TS defines the following status flags.

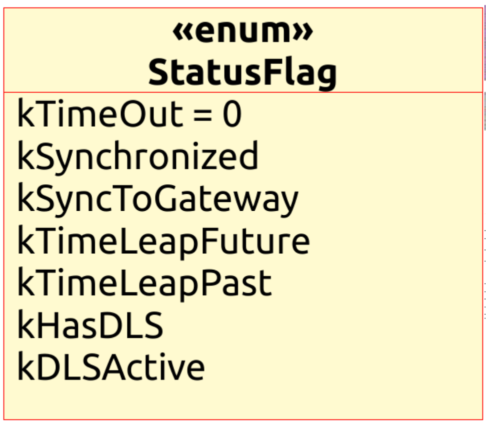


Figure 7.2: Status Flags Enumeration.

The status of the TBRs will be encapsulated within an instance of a `TimeBaseStatus` class.

The Application can query for specific status information by means of these flags.

The meaning of these internal flags (when they are set) are:

- `kTimeOut`: Indicate whether a synchronization of the time base of the corresponding TBR is lost / delayed.
- `kSynchronized`: Indicates if the time base of the corresponding TBR has been successfully synchronized at least once against its time source.
- `kSyncToGateway`: Indicates if the corresponding TBR updates are based on a Time Gateway below the Global Time Master.
- `kTimeLeapFuture`: Indicates if there has been a time leap jump into the future.
- `kTimeLeapPast`: Indicates if there has been a time leap jump into the past.
- `kHasDLS`: Indicates if the time base of the corresponding TBR have DLS.
- `kDLSActive`: Indicates if the DLS is considered in the time base provided by the corresponding TBR.

The enumeration values serve the only purpose of allowing the `Adaptive Application` to refer to a specific status flag, when querying for its value.

7.1.3 TS and Synchronization

Time Synchronization mechanisms and protocols (i.e. [10] are out of the Scope of this Specification, for protocol specification please refer to the PRS (see [1]).

7.2 Startup behavior

This chapter describes the initialization performed by the constructor of the Time Base Resources, in order to prepare the TS module for normal behavior - in other words, to prepare the module for providing to the application developer the synchronized time services.

7.2.1 Construction behavior

When the system starts-up, the TBRs have to set the conditions necessary to start working with the TBR.

[SWS_TS_00006] ⌈ The clock of a Time Base and of a Time Base Resource shall be initialized with a configurable value. ⌋*(RS_TS_00003)*

[SWS_TS_00007] ⌈ Internal elements of Time Base Resources shall be initialized as follows:

- Status Flags shall be set to zero
- Update Counter shall be set to zero
- User Data shall be set to zero
- Time Leap information shall be set to zero
- Its clock shall be set to zero.

⌋*(RS_TS_00009)*

[SWS_TS_00135] ⌈ A clock shall return default values until it is configured the first time. ⌋*(RS_TS_00004)*

7.3 Shutdown behavior

7.4 Normal Operation

7.4.1 Introduction

A Global Time network consists of a Time Master and at least one Time Slave. For each Time Domain, its Time Master is distributing the Global Time Base to the connected Time Slaves via Time Synchronization messages. The Time Slave corrects the received Global Time Base by considering the Time Stamp at the transmitter side and the own generated receiver Time Stamp.

The local time of a Slave Time Base will be maintained autonomously up to the point when it is updated with a new time value from its associated Master Time Base.

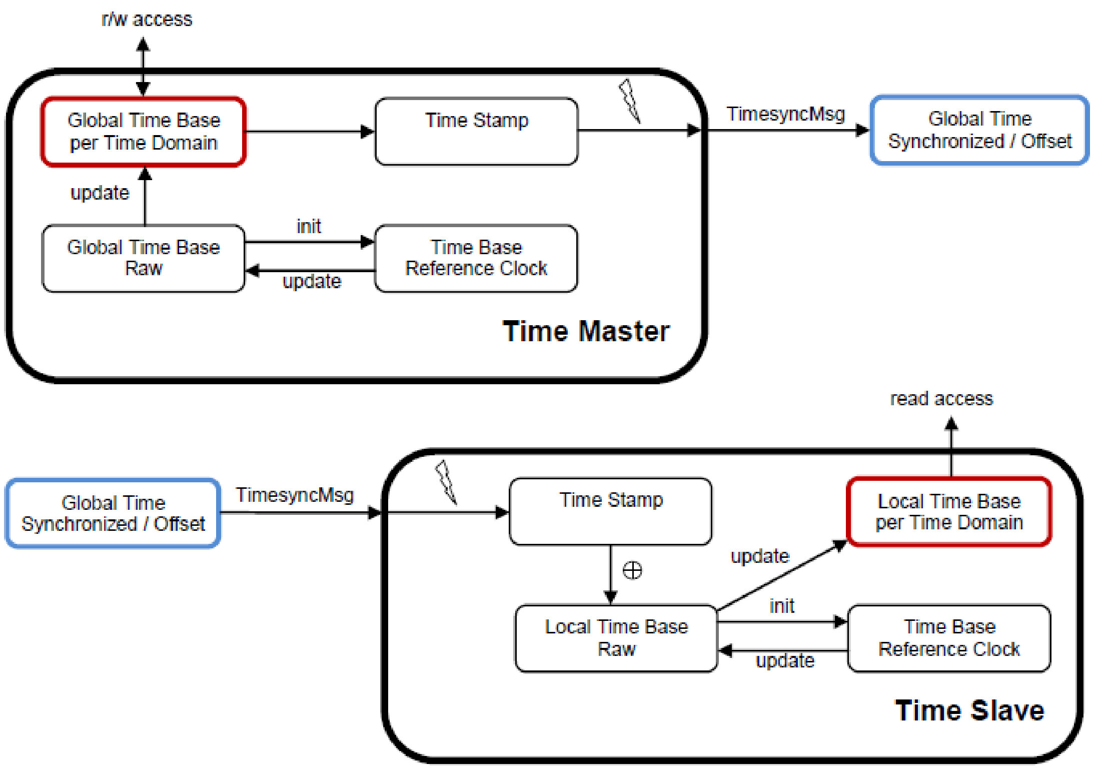

r/w access
Global Time Base per Time Domain
Time Stamp
TimesyncMsg
Global Time Synchronized / Offset
update
init
Global Time Base Raw
update
Time Base Reference Clock
Time Master
read access
Global Time Synchronized / Offset
TimesyncMsg
Time Stamp
Local Time Base per Time Domain
update
⊕
init
Local Time Base Raw
update
Time Base Reference Clock
Time Slave

Figure 7.3: Global Time Base Distribution.

7.4.1.1 Time Base Resources in the system

The TBRs are present in the System according to a prebuild configuration, which specifies the number of TBRs to be available in the system.

This prebuild configuration specifies also the type of TBR and in case of Offset Time Base types, it should also specify the Synchronized Time Base Resource they are based on.

The Application gets access to the modeled TBRs in the system by means of a find resources mechanism.

7.4.1.2 Types of Time Bases

From the Time Domain point of view, Time Bases are classified in Synchronized, Offset and Pure Local.

As already mentioned, TBRs are configured previously to a build. This means that it is not possible to dynamically add new clock types to an already compiled `Adaptive Application`. It is also not possible to change from one clock type to another one without recompiling, but it is possible to change the underlying resource of a clock during runtime. If there are for instance two SynchSlaveTBs defined in an `Adaptive Application`, it is not possible to add a third one without recompiling. But these two SynchSlaveTBs can be configured to represent any SynchSlaveTBR in the system during runtime. During compile time the TBRs don't have to be known.

The number of Synchronized Time Bases and Offset Time Bases is not limited by the TS functionality, but by the functional needs of the system to be fulfilled (i.e. the TS does not define a limit of Offset/Synchronized Time Bases identifiers in the system).

The only requirement in regards of the existence of Offset Time Bases states:

[SWS_TS_00008] ⌈ One Offset Time Base shall depend only on one Synchronized Time Base. ⌋*(RS_TS_00001)*

[SWS_TS_00009] ⌈ For a Synchronized Time Base it shall be possible to be referenced by multiple Offset Time Bases. ⌋*(RS_TS_00001)*

Therefore, having an Offset Time Base in the system without a dependency to a Synchronized Time Base is not possible.

Pure Local Time Bases will be set and read locally; they behave like Synchronized Time Bases since they progress in time, but they are not synchronized via any TSP module.

7.4.2 Roles of the Time Base Resources

7.4.2.1 Global Time Master

Additionally to the Type of Time Bases, a TBR can act as a Global Time Master, in which case it is the system wide origin for a given Time Base and its values are distributed then via the network to other Time Slaves.

7.4.2.2 Time Slave

In the role of a Time Slave, the TBR updates its internally-maintained local time based on Global Time Base values, which are provided by the corresponding TSP module.

7.4.3 Synchronized Time Base Resources

The Synchronized TBRs maintain their local time autonomously, regardless if they have already received a Global Time Base value or not.

[SWS_TS_00012] ⌈ If a Synchronized TBR has already received a Global Time Base value its `kSynchronized` status flag shall be set. ⌋*(RS_TS_00009)*

7.4.3.1 Synchronized Master Time Base

[SWS_TS_00013] ⌈ On a valid invocation of `setTime()` or `updateTime()` the Synchronized Master TBR shall update the local time of the corresponding Time Base. ⌋*(RS_TS_00010)*

7.4.3.2 Synchronized Slave Time Base

[SWS_TS_00014] ⌈ On a valid invocation of `now()`, a Synchronized TBR shall offer the current Time Base by means of a *time_point* data type compatible to std::chrono. ⌋*(RS_TS_00005, RS_TS_00023)*

In this way, the Application is able to cast the returned *time_point* to the resolution that best suits its requirements.

[SWS_TS_00015] ⌈ `calculateTimeDiff()` shall return a 'duration' data type compatible to std::chrono. ⌋*(RS_TS_00023)*

The calculation of this duration is the result of subtracting the given time point (as parameter) from the referenced Time Base (i.e. the Synchronized TBR this TBR is based on).

7.4.4 Offset Time Base Resources

7.4.4.1 Offset Master Time Base

[SWS_TS_00016] ⌈ `setOffset()` shall update the Offset Time of the corresponding Time Base. ⌋*(RS_TS_00013)*

[SWS_TS_00017] ⌈ `getOffset()` shall return the Offset Time of the corresponding Time Base. ⌋*(RS_TS_00012)*

[SWS_TS_00018] ⌈ On invocation of `setTime()` or `updateTime()` shall check the `kSynchronized` status flag of the underlying Synchronized TBR and shall raise an exception if such status flag is not set. ⌋*(RS_TS_00010)*

[SWS_TS_00019] ⌈ If during a call to `setTime()` or `updateTime()`, the `kSynchronized` flag is set, the Offset Master TBR, shall calculate the Offset Time by obtaining the actual Time Base value of the underlying Synchronized TBR and subtract that from the time point which is passed as parameter. The resulting calculation (e.g. Offset Time) shall be used to maintain the internal clock of the according TBR. ⌋ *(RS_TS_00007)*

[SWS_TS_00021] ⌈ The Application that interacts with an Offset Master TBR shall set the User Data whenever it is more convenient by means of the `setUserData()` method. ⌋*(RS_TS_00015)*

Note: For information about the retrieving of the User Data, please refer to section section 7.1.

[SWS_TS_00133] ⌈ The underlying Synchronized TBR can be accessed via the member type SynchMasterTimebase. ⌋*(RS_TS_00026, RS_TS_00001)*

7.4.4.2 Offset Slave Time Base

[SWS_TS_00022] ⌈ For an Offset Slave TBR, the `now()` method shall return a time point calculated by adding its offset to the current Time Base of the referenced Time Domain (i.e. Synchronized TBR). ⌋*(RS_TS_00005)*

[SWS_TS_00134] ⌈ The underlying Synchronized TBR can be accessed via the member type SynchMasterTimebase. ⌋*(RS_TS_00026, RS_TS_00001)*

7.4.4.3 Pure Local Time Base

Pure Local TBR behaving like an Offset TBR is not supported.

[SWS_TS_00023] ⌈ A Pure Local TBR shall maintain the Time Base autonomously. ⌋ *(RS_TS_00002)*

Until the `setTime()` method was called for the first time, a Pure Local TBR will return a *time_point* with duration set to zero.

[SWS_TS_00024] ⌈ Once the Pure Local TBR has been updated with a new Time Base value, its `kSynchronized` status flag shall be set. ⌋*(RS_TS_00009)*

[SWS_TS_00025] ⌈ For Pure Local TBRs all status flags shall be set to zero, except for the `kSynchronized`, which shall be set to 1 by a valid invocation of `setTime()` or `updateTime()` and only set to zero during its constructor execution. ⌋ *(RS_TS_00009)*

7.4.5 Synchronization State

[SWS_TS_00136] ⌈ For any type of TBR, the method `getTimeBaseStatus()` shall return a new instance of class `TimeBaseStatus` containing a copy of the status flags and other status information at the point of time of its creation.

For Offset TBRs, the method `getTimeBaseStatus()` shall additionally obtain a `TimeBaseStatus` instance of its underlying Synchronized TBR, adhering to the same creation time. ⌋*(RS_TS_00021)*

[SWS_TS_00137] ⌈ For Offset TBRs, the method `getSynchStatus()` of the `TimeBaseStatus` object associated shall return a copy of another `TimeBaseStatus` object; the later corresponding to the underlying Synchronized TBR of the associated Offset TBR. ⌋*(RS_TS_00021)*

[SWS_TS_00138] ⌈ For Synchronized TBRs, the `TimeBaseStatus` object associated shall return a copy of itself upon a call of its method `getSynchStatus()`. ⌋*(RS_TS_00021)*

7.4.5.1 Slave Time Bases

Usually a Slave Time Base starts its local Time Base from zero. So, after initialization the 1^{st} check against the 'timeLeapFutureThreshold' or the 'timeLeapPastThreshold' would most likely always fail and the `kTimeLeapFuture` or the `kTimeLeapPast` status flag would always be set. To avoid this, threshold monitoring should be deactivated.

[SWS_TS_00139] ⌈ Time leap future monitoring shall be enabled only if time 'timeLeapFutureThreshold' is set different than zero and if the `kSynchronized` status flag is set. ⌋*(RS_TS_00009)*

[SWS_TS_00140] ⌈ Time leap past monitoring shall be enabled only if time 'timeLeapPastThreshold' is set different than zero and if the `kSynchronized` status flag is set. ⌋*(RS_TS_00009)*

[SWS_TS_00141] ⌈ If at least one Time Base value has been successfully received (i.e. if the flag `kSynchronized` is set), then it shall be checked during the update of the Global Time if the time difference between the current and the updated Time Base value exceeds the configured threshold of 'timeLeapFutureThreshold' or 'TimeLeapPastThreshold'. ⌋*(RS_TS_00009)*

[SWS_TS_00027] ⌈ In case of the new Time Base value exceeding either the 'timeLeapFutureThreshold' or the 'timeLeapPastThreshold', then the corresponding status flag (i.e. `kTimeLeapFuture` or `kTimeLeapPast`) shall be set. ⌋*(RS_TS_00009)*

[SWS_TS_00028] ⌈ If the next number of updates of Time Base values, as defined by parameter 'clearTimeleapCount', are within the threshold of 'timeLeapFutureThreshold' of 'timeLeapPastThreshold' (depending on the case), then the corresponding status flag (i.e. `kTimeLeapFuture` or `kTimeLeapPast`) shall be cleared. ⌋*(RS_TS_00009)*

[SWS_TS_00030] ⌈ A timeout 'syncLossTimeout' shall be monitored by each Time Slave. The timeout 'SyncLossTimeout' shall be measured from the last update of the Time Base (i.e. last synchronization with/from TSP). ⌋*(RS_TS_00009)*

[SWS_TS_00032] ⌈ If the synchronization loss timeout takes place, the TBR shall set the `kTimeOut` status flag. ⌋*(RS_TS_00009)*

[SWS_TS_00011] ⌈ If the synchronization loss timeout takes place, and the TBR in question is updated against a Time Gateway, the TBR shall set the `kSyncToGateway` status flag. ⌋*(RS_TS_00009)*

[SWS_TS_00033] ⌈ The `kTimeOut` status flag shall be cleared on a successful update of the Time Base (i.e. successful synchronization with/from TSP). ⌋*(RS_TS_00009)*

[SWS_TS_00020] ⌈ The `kSyncToGateway` status flag shall be set on every successful update of the Time Base (i.e. successful synchronization with/from TSP), if such update is done against a Time Gateway and it should be cleared otherwise. ⌋ *(RS_TS_00009)*

[SWS_TS_00034] ⌈ If the Time Base of a Time Slave is updated, the status flag `kSynchronized` shall be set. ⌋*(RS_TS_00009)*

Note: Once the status flag is set, it will never be cleared.

7.4.6 Immediate Time Synchronization

All TSP Modules are working independently of the TS regarding the handling of the bus-specific Time Synchronization protocol (i.e. autonomous transmission of Timesync messages on the bus).

Time information is passed from a TSP to the TBR. Implementation details as well as the interaction of such a TSP with the TBR are outside of the scope of this specification(, for protocol specification please refer to [1]).

Nevertheless, it might be necessary, that the TBRs provide an interface, based on an `updateCounter`, to allow the TSP Binding Entity to detect if a TBR has been updated or not and thus may perform an immediate transmission of Timesync messages in order to speed up re-synchronization.

[SWS_TS_00035] ⌈ The updateCounter of a TBR shall have the value range 0 to 255. ⌋*(RS_TS_00009, RS_TS_00021)*

[SWS_TS_00036] ⌈ On a valid invocation of `setTime`, or a valid update of the Time Base, the TBR shall increment its updateCounter by 1. At 255 it shall wrap around to 0. ⌋*(RS_TS_00009)*

7.4.7 User Data

User Data is part of each Time Base. User Data is set by the Global Time Master of each Time Base and distributed as part of the Timesync messages.

User Data can be used to characterize the Time Base, e.g., regarding the quality of the underlying clock source or regarding the progress of time.

User Data consists of a vector of bytes. Due to the frame format of various Timesync messages it might not be possible to transmit the complete vector on every bus system. It is the responsibility of the system designer to use only those User Data bytes in the vector that can be distributed inside the vehicle network.

7.4.8 Time Correction

TS provides the ability for Time Slaves to perform Rate and Offset Correction of the Synchronized TBR and Rate Correction of an Offset Time Base.

For Global Time Masters, the TS provides the ability to perform Rate Correction of their Time Base(s).

Time correction can be configured individually for each Time Base.

7.4.8.1 Rate Correction for Time Slaves

Rate Correction detects and eliminates rate deviations of local instances of Time Bases and of Offset Time Bases. Rate Correction determines the rate deviation in the scope of a measurement. This rate deviation is used as correction factor which the TBR uses to correct the Time Base's time whenever it is read (e.g. in the scope of `now()`).

[SWS_TS_00037] ⌈ The TBR shall not perform Rate Correction if the measurement duration parameter '`RateDevMeasurementDuration`' is *false*. ⌋*(RS_TS_00002, RS_TS_00007, RS_TS_00018)*

[SWS_TS_00038] ⌈ For Rate Correction measurements, the TBR shall evaluate state changes of the `kTimeLeapFuture` and the `kTimeLeapPast` status flags during measurements. The TBR shall discard the measurement if any of these flags state changes. ⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00029] ⌈ For Rate Correction measurements, the TBR shall evaluate state changes of the `kSyncToGateway` flag during measurements. The TBR shall discard the measurement if the state of this flag changes. ⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00039] ⌈ For Rate Correction measurements, the TBR shall evaluate state changes of the `kTimeOut` status flag during measurements. The TBR shall discard the measurement if the flag state changes. ⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00040] ⌈ For Rate Correction measurements, the TBR shall evaluate the `kTimeLeapFuture` and the `kTimeLeapPast` status flags during the start of a measurement. The TBR shall not start a Rate Correction measurement when any of these status flags are set. ⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00041] ⌈ The TBR shall perform Rate Correction measurements to determine its rate deviation. ⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00042] ⌈ The TBR shall perform Rate Correction measurements continuously. The end of a measurement marks the start of the next measurement.

The start and end of measurements is always triggered by (and aligned to) the reception of time values for Synchronized or Offset Time Bases. ⌋*(RS_TS_00002, RS_TS_00007, RS_TS_00018, RS_TS_00019)*

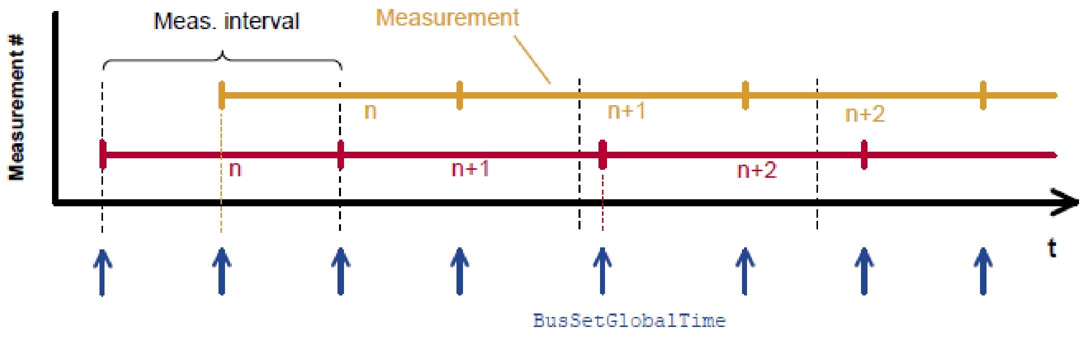


Figure 7.4: Visualization of two parallel measurements.

[SWS_TS_00043] ⌈ During runtime, the Synchronized TBR shall determine the timespan of a Rate Correction measurement on the basis of its own clock. ⌋*(RS_TS_00018)*

[SWS_TS_00142] ⌈ During runtime, the Offset TBR shall determine the timespan of a Rate Correction measurement on the basis of its associated Synchronized TBR's clock. ⌋*(RS_TS_00018)*

[SWS_TS_00044] ⌈ The TBR shall perform as many simultaneous Rate Correction measurements as configured by the parameter 'RateCorrectionsPerMeasurementDuration'. ⌋*(RS_TS_00018)*

[SWS_TS_00045] ⌈ Simultaneous Rate Correction measurements shall be started with a defined offset (to_n) to yield Rate Corrections evenly distributed over the measurement duration. The value will be calculated according to the following formula: `to`$_n$ `= n * (rateDevMeasurementDuration / RateCorrectionPerMeasurementDuration)` (where 'n' is the zero-based index of the current measurement) ⌋*(RS_TS_00018, RS_TS_00019)*

[SWS_TS_00046] ⌈ At the start of a Rate Correction measurement, the Synchronized TBR shall take the following time-snapshots in the scope of TSP: ⌋*(RS_TS_00018)*

- TGStart - Current time of the global Time Base Time Master

- TVStart - Current time of the Virtual Local Time of the associated Time Base.

[SWS_TS_00047] ⌈ At the start of a Rate correction measurement, the Offset TBR, shall take the following time-snapshots in the scope of TSP: ⌋*(RS_TS_00018)*

- TSStart - Current corrected time provided by the local instance of the associated Time Base
- TOStart - Current Offset of the Offset Time Base given as function parameter.

[SWS_TS_00048] ⌈ At the end of the Rate Correction measurement, the Synchronized TBR shall take the following time-snapshots in the scope TSP: ⌋*(RS_TS_00018)*

- TGStop - Current time of the Global Time Base Time Master
- TVStop - Current time of the Virtual Local Time of the associated Time Base.

[SWS_TS_00049] ⌈ At the end of the Rate Correction measurement, the Offset TBR shall take the following time-snapshots in the scope TSP: ⌋*(RS_TS_00018)*

- TSStop - Current corrected time provided by the local instance of the associated Time Base
- TOStop - Current Offset of the Offset Time Base given as function parameter.

[SWS_TS_00050] ⌈ At the end of a Rate Correction measurement, the Synchronized TBR shall calculate the resulting correction rate (r_{rc}) according to the following formula: $r_{rc} = (TG_{Stop} - TG_{Start}) / (TV_{Stop} - TV_{Start})$ ⌋*(RS_TS_00018, RS_TS_00019)*

Note: To determine the resulting rate deviation the value 1 has to be subtracted from r_{rc}.

[SWS_TS_00051] ⌈ The last r_{rc} value has to be used until a new value is calculated. ⌋*(RS_TS_00018, RS_TS_00019)*

[SWS_TS_00052] ⌈ Offset TBRs shall not perform yet another rate correction, because this is done by the underlying TBR already. ⌋*(RS_TS_00018, RS_TS_00019)*

[SWS_TS_00053] ⌈ On invocation of getRateDeviation() the TBR shall return the calculated rate deviation (i.e. r_{rc}-1). ⌋*(RS_TS_00018)*

[SWS_TS_00070] ⌈ If no rate deviation has yet been calculated, getRateDeviation() shall return 0.0. ⌋*(RS_TS_00018)*

[SWS_TS_00054] ⌈ If a valid correction rate (r_{rc}) has been calculated, the Synchronized TBR shall apply a Rate Correction. ⌋*(RS_TS_00018, RS_TS_00019)*

[SWS_TS_00071] ⌈ If a valid correction rate (r_{orc}) has been calculated, the Offset TBR shall apply a Rate Correction. ⌋*(RS_TS_00018, RS_TS_00019)*

7.4.8.2 Offset Correction for Time Slaves

Offset Correction eliminates time offsets of local instances of Synchronized Time Bases. This correction takes place whenever the current time is read (e.g. in the scope of `now()`). The offset is measured when the local instance of the Time Base is synchronized in the scope of TSP.

[SWS_TS_00055] ⌈ For Synchronized TBRs, it shall be measured the offset between its local instance of the Time Base and the Global Time Base whenever the Time Base is synchronized in the scope of the function TSP by taking a snapshot of the following values: ⌋*(RS_TS_00013)*

- TL_{Sync} = Value of the local instance of the Time Base before the new value of the Global Time is applied
- TV_{Sync} = Value of the Virtual Local Time

[SWS_TS_00056] ⌈ If the absolute value of the time offset between Global Time Base and local instance of the Time Base (abs(TG - TL_{Sync})) is equal or greater than '`OffsetCorrectionJumpThreshold`', the TBR shall calculate the corrected time (TL) of its local instance of the Time Base according to the following formula:
$TL = TG + (TV - TV_{Sync}) * r_{rc}$

- TV = Current value of the Virtual Local Time

- TV_{Sync} = Value of the Virtual Local Time

- TG = Received value of the Global Time

- r_{rc} = Most current rate for correcting the local instance of the Time Base ⌋*(RS_TS_00013, RS_TS_00019)*

Note:
This correction shall be done whenever the time is read in the scope of the `now()` method.

Note:
This correction shall be done when the TBR needs to determine the time of the local instance of the Time Base.

[SWS_TS_00057] ⌈ The TBR shall correct absolute time offsets between the Global Time Base and the local instance of the Time Base (abs(TG - TL_{Sync})), which are smaller than the value given by '`OffsetCorrectionJumpThreshold`' by temporarily applying an additional rate (r_{oc}) to r_{rc}. This rate shall be used for the duration defined by parameter '`OffsetCorrectionAdaptionInterval`'. r_{oc} is calculated according to the following formula:
$r_{oc} = (TG - TL_{Sync}) / (T_{CorrInt}) + 1$

- $T_{CorrInt}$ = `OffsetCorrectionAdaptionInterval`

- TL_{Sync} = Value of the local instance of the Time Base before the new value of the Global Time is applied

- TG = Received value of the Global Time ⌋*(RS_TS_00013, RS_TS_00019)*

[SWS_TS_00058] ⌈ If the absolute time offset between Global Time Base and local instance of the Time Base (abs(TG - TL_{Sync})) is smaller than '`OffsetCorrectionJumpThreshold`', the TBR shall calculate the corrected time (TL) of its local instance of the Time Base **within** the period of '`OffsetCorrectionAdaptionInterval`' according to the following formula:
$\text{TL} = \text{TL}_{Sync} + (r_{rc} \star (\text{TV} - \text{TV}_{Sync}) \star r_{oc})$

- TL_{Sync} = Corrected current value of the local instance of the Time Base

- TV = Current value of the Virtual Local Time of the Time Base

- TV_{Sync} = Value of the Virtual Local Time

- r_{rc} = Actual rate for correcting the local instance of the Time Base

- r_{oc} = Rate for time offset elimination via Rate Adaption ⌋*(RS_TS_00013, RS_TS_00019)*

Note:
This correction shall be done whenever the time is read in the scope of these function `now()`.

Note:
This correction shall be done when the TBR needs to determine the time of the local instance of the Time Base.

[SWS_TS_00059] ⌈ If the absolute time offset between the Global Time Base and the local instance of the Time Base (abs(TG - TL)) is smaller than OffsetCorrection-JumpThreshold, the TBR shall calculate the corrected time (TL) of its local instance of the Time Base **after** the period of OffsetCorrectionAdaptionInterval as specified in [SWS_TS_00056] ⌋*(RS_TS_00013)*

[SWS_TS_00060] ⌈ If `OffsetCorrectionJumpThreshold` is set to 0, Offset Correction shall be performed by Jump Correction only. ⌋*(RS_TS_00013)*

7.4.8.3 Rate Correction for Global Time Masters

Rate correction in Global Time Masters can be applied to Synchronized and Offset Time Bases Resources.

Rate correction is applied by setting a correction factor which the TBR uses to correct the Time Base's time whenever it is transmitted over the network. This happens independent of the rate correction done by the slave.

[SWS_TS_00061] ⌈ If 'AllowMasterRateCorrection' equals *true*, an invocation of `setRateCorrection()` shall set the rate correction value. Otherwise `setRateCorrection()` shall do nothing and throw an exception. ⌋*(RS_TS_00018)*

[SWS_TS_00062] ⌈ The TBR shall apply rate correction, if `AllowMasterRateCorrection` equals TRUE and a valid rate correction value has been set by `setRateCorrection()`. ⌋*(RS_TS_00018)*

[SWS_TS_00063] ⌈ If the absolute value of the rate correction parameter `rateCorrection`, which is passed to `setRateCorrection()`, is greater than `MasterRateDeviationMax`, `setRateCorrection()` shall set the actually applied rate correction value to either (`MasterRateDeviationMax`) or (-`MasterRateDeviationMax`)(depending on sign of `rateCorrection`). ⌋*(RS_TS_00018)*

Note: The actual applied resulting rate will be the passed deviation value + 1. If aligning the rate of one Time Base to the rate of another one, it is possible to use `getRateDeviation()` and pass the value as argument to `setRateCorrection()`.

7.4.9 Notification of Applications

The Application might either request to be notified of status change events for a specific TBR, or request to be notified when a timer, which has been previously set by the Application, expires.

***Note:** Notifications to Application about changes in the status of the Time Base Resources is a feature considered to be offered in future version/releases of TS.*

7.4.9.1 Time Notifications

The TS allows Notification to Applications to set a Timer using the Method `StartTimer` of the methods namespace of a modeled `ServiceProxy`. The Application uses this method passing as parameter the duration of the timer and a '`callID`'.

The method returns `Future<>` object corresponding to the `callID` set by the Application. The `callID` on the `Future<>` object will be available upon the expiration of the timer. This allows the Application to check for the status of the timer by inquiring the availability of the `callID` on the `Future<>` object.

Additionally, the `callID` available in a `Future<>` object denotes or identifies which of the possibly multiple timers set by the Application has expired.

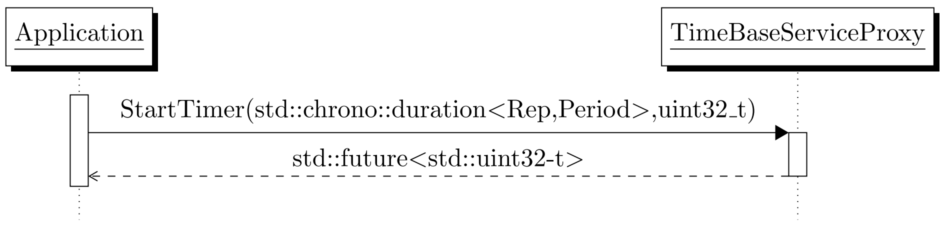

Application
TimeBaseServiceProxy
StartTimer(std::chrono::duration<Rep,Period>,uint32_t)
std::future<std::uint32-t>

Figure 7.5: Mechanism of Time Notification.

[SWS_TS_00064] ⌈ On invocation of `StartTimer()` for a Time Notification Application of a Time Base Resource, a measurement of the '`expireTime`' (a period of time expressed as a duration data type) shall be performed. ⌋*(RS_TS_00016, RS_TS_00017)*

7.4.9.2 Status Notifications

Note: *Notification to Application about changes in the status of the Time Base Resources is a feature considered to be offered in future version/releases of TS.*

7.4.10 Triggering Application

It is considered to offer Triggering Application functionality in a future version / release of TS.

7.4.11 Global Time Precision Measurement Support

It is considered to offer Global Time Precision Measurement Support in a future version / release of TS.

7.5 Error Handling

[SWS_TS_00065] ⌈ If the application could not obtain a handle during the creation of a TimeBaseConfig via a *findResource*, an exception shall be fired. This can happen, because either no such TB was configured or due to communication errors.⌋ *(RS_TS_00026)*

[SWS_TS_00072] ⌈ An exception shall be fired, if the TimeBaseType of the configuration does not match the one of the TB. E.g. trying to use a SynchSlaveTB config to configure a PureLocalTB. ⌋*(RS_TS_00026)*

[SWS_TS_00069] ⌈ If the Application tries to get a specialized implementation, which do not correspond to the type of TBR, an exception shall be fired.⌋*(RS_TS_00026)*

7.6 Error Classification

7.7 Version Check

It is considered to offer a Version Check feature in future version / release of TS.

8 API specification

8.1 Type definitions

TS defines seven enumeration classes. One enumeration for the `Adaptive Application` to identify the type of TBRs and one enumeration to classify the status flags of the TBRs. Furthermore there is a specific `Identity`-enum for every TB to enable distinction between multiple manifestations of the same clock type.

8.1.1 TimebaseType

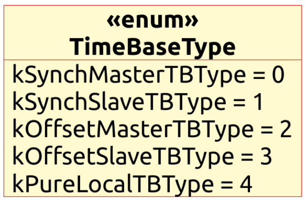


Figure 8.1: Enumeration defined in the scope of ara::tsync

[SWS_TS_00066] ⌈ TimebaseType shall be defined as described in table 8.1. ⌋ *(RS_TS_00026)*

Name:	TimeBaseType	
Type:	Enumeration	
Range:	0 .. 4	0 kSynchMasterTBType –> Synchronized Master TB 1 kSynchSlaveTBType –> Synchronized Slave TB 2 kOffsetMasterTBType –> Offset Master TB 3 kOffsetSlaveTBType –> Offset Slave TB 4 kPureLocalTBType –> Pure Local TB
Description:	Each value of this enumeration lets the application know which type of TB it is working with or which type of TB reference a particular handle contains.	

Table 8.1: TimeBaseType

8.1.2 StatusFlag

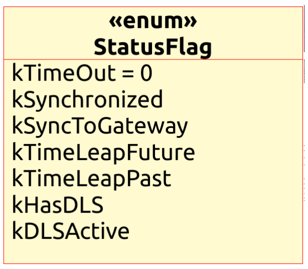


Figure 8.2: Status Flag enumeration

[SWS_TS_00067] ⌈ StatusFlag shall be defined as described in table 8.2. ⌋ *(RS_TS_00009)*

Name:	StatusFlag	
Type:	Enumeration	
Range:	0 .. n	0 kTimeOut kSynchronized kSynchToGateway kTimeLeapFuture kTimeLeapPast kHasDLS kDLSActive
Description:	Each enumeration represents a flag in the status of a TBR. These flags will be set or cleared according to the behavior described in chapter 7 for each type of TBR.	

Table 8.2: StatusFlag

8.1.3 Identities

Time Base Identities are used to locally distinguish between multiple manifestations of the same clock type. Otherwise having for instance two SynchSlaveTBs would be potentially harmful, because the developer could confuse them. By default there is only one enumeral defined. More identities can be defined by static_casting an integer to the identity (e.g. `static_cast<std::underlying_type<SynchSlaveIdentity>::type>(1)`)

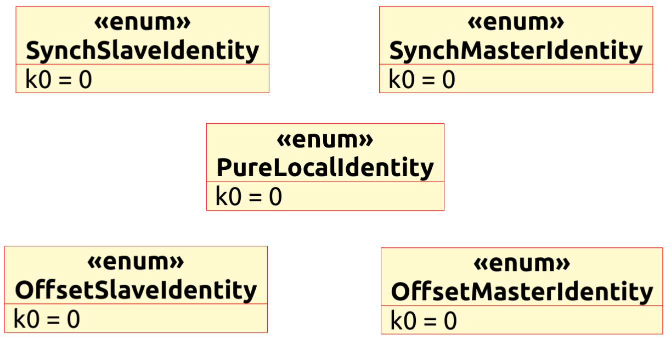


Figure 8.3: Time Base Identities

8.1.4 Chrono Clock Paradigm

In order to be able to rely on std::chrono functionality, the clocks need to be compatible to the std::chrono-clock paradigm. This can be achieved by four member using declaratives.

[SWS_TS_00078] ⌈ Member type aliases `duration`, `rep`, `period`, `time_point` shall be defined as described in table 8.3, 8.4, 8.5 and 8.6. ⌋ *(RS_TS_00022, RS_TS_00023)*

Type alias name:	duration
Syntax:	`using duration = std::chrono::duration<Rep,Period>`
Description:	This member type alias defines the resolution of the clock.

Table 8.3: Type alias definition - duration

Type alias name:	rep
Syntax:	`using rep = duration::rep`
Description:	This member type alias defines the datatype used to store the duration value.

Table 8.4: Type alias definition - rep

Type alias name:	period
Syntax:	`using period = duration::period`
Description:	This member type alias defines the resolution of the stored duration value.

Table 8.5: Type alias definition - period

Type alias name:	time_point
Syntax:	`using time_point = std::chrono::time_point<Clock, duration>;`
Description:	This member type alias defines the time_point, specific to the clock type.

Table 8.6: Type alias definition - time_point

8.1.5 Underlying Time Bases

8.1.5.1 Master/Slave/PureLocalTB::referenceClock

Type alias name:	referenceClock
Syntax:	`using referenceClock = ReferenceClock`
Description:	This member type alias refers to the clock that this TB will use to obtain the local time. The default value shall be std::chrono::steady_clock.

Table 8.7: Type alias definition - referenceClock

8.1.5.2 OffsetSlaveTB::synchSlaveTimebase

[SWS_TS_00094] ⌈ Member type alias `synchSlaveTimebase` shall be defined as described in table 8.8. ⌋*(RS_TS_00026, RS_TS_00001)*

Type alias name:	synchSlaveTimebase
Syntax:	`using synchSlaveTimebase = SynchSlaveType`
Description:	This member type alias refers to the Synchronized Slave TB this Offset Slave TB is based on.

Table 8.8: Type alias definition - synchSlaveTimebase

8.1.5.3 OffsetMasterTB::synchMasterTimebase

[SWS_TS_00115] ⌈ Member type alias `synchMasterTimebase` shall be defined as described in table 8.9. ⌋*(RS_TS_00001, RS_TS_00026)*

Type alias name:	synchMasterTimebase
Syntax:	`using synchMasterTimebase = SynchMasterType`
Description:	This member type alias refers to the Synchronized Master TB this Offset Master TB is based on.

Table 8.9: Type alias definition - synchMasterTimebase

8.1.5.4 TimeBaseStatus::timeBase

[SWS_TS_00124] ⌈ Member type alias `timeBase` shall be defined as described in table 8.10. ⌋*(RS_TS_00021, RS_TS_00026)*

Type alias name:	timeBase
Syntax:	`using timeBase = TB`
Description:	This member type alias refers to the TB this TimeBaseStatus is based on.

Table 8.10: Type alias definition - timeBase

8.2 Function definitions

The TS covers the complete set of interfaces of the TBRs and part of the interfaces provided in the Time Base Resource Proxies, the methods namespace as well as the methods of the Proxies that are not static.

8.2.1 findResource Function Definition

[SWS_TS_00168] ⌈ Function `findResource` in namespace ara::tsync shall be defined as described in table 8.11. ⌋*(RS_TS_00005)*

Method name:	findResource	

Syntax:	`template <typename ServiceProxy>`
	`ara::tsync::TimeBaseConfig findResource()`	

Sync/Async:	Asynchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TimeBaseConfig	A TimeBaseConfig object that can be used to configure a local TB type.
Description:	The findResource function takes a ServiceProxy type and returns a TimeBaseConfig object, which is constructed using the values of the fields of the proxy.	

Table 8.11: Function definition - findResource

[SWS_TS_00001] ⌈ The different TBR types shall be identified by a *TimeBaseType* enumeration. ⌋*(RS_TS_00001, RS_TS_00022)*

[SWS_TS_00002] ⌈ Configurations for TBs, configuring a locally defined type to use a certain TBR, can be obtained by the Application by means of calling *FindResource* on a modeled proxy dedicated for one single Time Base Resource. ⌋*(RS_TS_00005)*

8.2.1.1 methods::StartTimer

This section includes the Functor within the "methods" namespace. This namespace is in the scope of ara::tsync.

For a more detailed information, and only as an explanatory reference of the methods and the mechanics of *FindResource*, please refer to Figure 9.1.

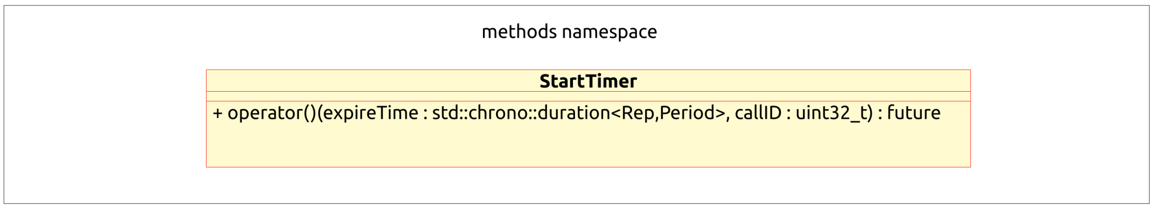

methods namespace
StartTimer
+ operator()(expireTime : std::chrono::duration<Rep,Period>, callID : uint32_t) : future

Figure 8.4: The "methods" namespace.

[SWS_TS_00068] ⌈ Method `StartTimer` shall be defined as described in table 8.12. ⌋*(RS_TS_00016, RS_TS_00017)*

Method name:	StartTimer	
Syntax:	`Future<uint32_t> StartTimer ( std::chrono::duration<Rep, Period> expireTime, uint32_t callID)`	
Sync/Async:	Asynchronous	
Parameters (in):	expireTime	Duration as defined in C++11 Standard. This represents the amount of time after which the timer will expire.
	CallID	ID assigned to this timer (should be unique to the Application).
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	Future<uint32_t>	The timer ID set by the Application will be returned Asynchronously.
Description:	Once the timer expires, the callID set by Application will be returned, so the Application could distinguish by callID, which of the possible multiple timers is expiring.	

Table 8.12: Method definition - StartTimer

8.2.2 Common Function Definition of Time Bases

The function definitions in this chapter are to be implemented by every Time Base

For more information on the classes of the Time Bases design and/or to consult a specific class/method, please refer to Figure 7.1 and to section 7.1.

8.2.2.1 getRateDeviation

[SWS_TS_00084] ⌈ Method `getRateDeviation` shall be defined as described in table 8.13. ⌋*(RS_TS_00018)*

Method name:	getRateDeviation	
Syntax:	`static double getRateDeviation()`	
Implemented by:	SynchMasterTB, OffsetMasterTB, SynchSlaveTB, OffsetSlaveTB	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	double	Value of the current rate deviation of the TBR.
Description:	Returns a value of the current rate deviation of the TBR	

Table 8.13: Method definition - getRateDeviation

It is intended, that all the TBRs implement this method. One exception is the PureLocalTB, because it maintains its own time base.

[SWS_TS_00085] ⌈ The Pure Local TBRs shall not implement this method. ⌋*(RS_TS_00026)*

8.2.2.2 getTimeBaseStatus

[SWS_TS_00086] ⌈ Method `getTimeBaseStatus` shall be defined as described in table 8.14. ⌋*(RS_TS_00021)*

Method name:	getTimeBaseStatus	
Syntax:	`template<typename TB1, typename TB2 = TB1> static TimeBaseStatus<TB1, TB2> getTimeBaseStatus()`	
Implemented by:	SynchMasterTB, OffsetMasterTB, SynchSlaveTB, OffsetSlaveTB, PureLocalTB	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TimeBaseStatus<TB1,TB2>	A TimeBaseStatus object bounded to a specific type of TBR - contains all the status information of the TBR.

Description:	Upon the call of this Method, the TBR will create an object of this class, which will be bounded to the type of the calling TBR and it will populate all the status information on this newly created object to then return it to the Application.

Table 8.14: Method definition - getTimeBaseStatus

[SWS_TS_00087] ⌈ Upon the call of this method, every TBR shall:

- create an object of type `TimeBaseStatus`
- Populate this newly created object with the status information of the TBR
- Return this newly created -and populated- object to the Application

⌋*(RS_TS_00009)*

8.2.2.3 getType

[SWS_TS_00093] ⌈ Method `getType` shall be defined as described in table 8.15. ⌋ *(RS_TS_00026)*

Method name:	getType	
Syntax:	`static constexpr TimebaseType getType()`	
Implemented by:	SynchMasterTB, OffsetMasterTB, SynchSlaveTB, OffsetSlaveTB, PureLocalTB	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TimebaseType	TimebaseType enumeration value
Description:	This Method returns a TimebaseType enumeration value corresponding to the type of the TBR in question.	

Table 8.15: Method definition - getType

8.2.2.4 setTime

[SWS_TS_00098] ⌈ Method `setTime` shall be defined as described in table 8.16. ⌋ *(RS_TS_00010, RS_TS_00026)*

Method name:	setTime	
Syntax:	`template <typename Duration = duration> static void setTime( std::chrono::time_point<Clock, Duration> timePoint)`	
Implemented by:	SynchMasterTB, OffsetMasterTB, PureLocalTB	
Sync/Async:	Synchronous	
Parameters (in):	timePoint	New time stamp

Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Allows the Application to set the new global time that has to be valid for the system, which will be sent to the TSP.	

Table 8.16: Method definition - setTime

[SWS_TS_00099] ⌈ This method shall have its own implementation in class `OffsetMasterTB`, `SynchMasterTB` and in class `OffsetMasterTB`. ⌋*(RS_TS_00010, RS_TS_00026)*

[SWS_TS_00100] ⌈ Implementation of `setTime()` method in the `OffsetMasterTB` shall check if the TBR is configured to act as Global Time Base and in case it is, it shall calculate the Offset Time by obtaining the actual Time Base value of the underlying Synchronized Time Base and subtract that from the Absolute Time value which is passed as parameter in this method. ⌋*(RS_TS_00026, RS_TS_00010)*

[SWS_TS_00101] ⌈ Implementation of `setTime()` method in the `OffsetMasterTB` and in the `SynchMasterTB` shall check if the TBR is configured to act as a Global Time Base and in case it is not, it shall return to the application without any return type. ⌋*(RS_TS_00026, RS_TS_00010)*

[SWS_TS_00102] ⌈ Implementation of `setTime()` method in the `SynchMasterTB` shall check if the TBR is configured to act as Global Time Base and in case it is, it shall update its internal clock according to the value which is passed as parameter in this Method. ⌋*(RS_TS_00010, RS_TS_00026, RS_TS_00002)*

[SWS_TS_00108] ⌈ Implementation of `setTime()` method in the `PureLocalTB` shall update its internal clock according to the value which is passed as parameter in this Method. ⌋*(RS_TS_00002, RS_TS_00026, RS_TS_00010)*

8.2.2.5 updateTime

[SWS_TS_00103] ⌈ Method `updateTime` shall be defined as described in table 8.17. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

Method name:	updateTime	
Syntax:	`template <typename Duration = duration> static void updateTime( std::chrono::time_point<Clock, Duration> timePoint)`	
Implemented by:	SynchMasterTB, OffsetMasterTB, PureLocalTB	
Sync/Async:	Synchronous	
Parameters (in):	timePoint	New time stamp
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	

Description:	Allows the Application to set the new global time that has to be valid for the system, which will be sent to the TSP. This method will not lead to an immediate transmission of the Global Time.

Table 8.17: Method definition - updateTime

[SWS_TS_00104] ⌈ This method shall be implemented in class `OffsetMasterTB`, `SynchMasterTB` and in class `PureLocalTB`. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

[SWS_TS_00105] ⌈ Implementation of `updateTime()` method in the `OffsetMasterTB` shall check if the TBR is configured to act as Global Time Base and in case it is, it shall calculate the Offset Time by obtaining the actual Time Base value of the underlying Synchronized Time Base and subtract that from the Absolute Time value which is passed as parameter in this Method. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

[SWS_TS_00106] ⌈ Implementation of `updateTime()` method in the `OffsetMasterTB` and in the `SynchMasterTB` shall check if the TBR is configured to act as a Global Time Base and in case it is not, it shall return to the application without any return type. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

[SWS_TS_00107] ⌈ Implementation of `updateTime()` method in the `SynchMasterTB` shall check if the TBR is configured to act as Global Time Base and in case it is, it shall update its internal clock according to the value which is passed as parameter in this Method. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

[SWS_TS_00110] ⌈ Implementation of `updateTime()` method in the `PureLocalTB` shall update its internal clock according to the value which is passed as parameter in this Method. ⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00001, RS_TS_00026)*

8.2.2.6 setRateCorrection

[SWS_TS_00109] ⌈ Method `setRateCorrection` shall be defined as described in table 8.18. ⌋*(RS_TS_00001, RS_TS_00026, RS_TS_00018)*

Method name:	setRateCorrection	
Syntax:	`static void setRateCorrection( double rateCorrection)`	
Implemented by:	SynchMasterTB, OffsetMasterTB	
Sync/Async:	Synchronous	
Parameters (in):	rateCorrection	Rate correction that shall be applied during time calculation.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Set the rate correction for a synchronized time base.	

Table 8.18: Method definition - setRateCorrection

8.2.2.7 setUserData

[SWS_TS_00088] ⌈ Method `setUserData` shall be defined as described in table 8.19. ⌋ *(RS_TS_00001, RS_TS_00026, RS_TS_00015)*

Method name:	setUserData	
Syntax:	`static void setUserData ( ara::core::Vector<std::uint8_t>& userData )`	
Implemented by:	SynchMasterTB, OffsetMasterTB, PureLocalTB	
Sync/Async:	Synchronous	
Parameters (in):	userData	Vector of bytes containing the new User Data.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Allows the Application to set the new User Data that has to be valid for the system, which will be sent to the busses.	

Table 8.19: Method definition - setUserData

8.2.2.8 configure

[SWS_TS_00188] ⌈ Method `configure` shall be defined as described in table 8.20. ⌋ *(RS_TS_00022, RS_TS_00026)*

Method name:	configure	
Syntax:	`static void configure(const TimeBaseConfig& time-BaseConfig)`	
Implemented by:	SynchMasterTB, SynchSlaveTB, OffsetMasterTB, OffsetSlaveTB, Pure-LocalTB	
Sync/Async:	Synchronous	
Parameters (in):	timeBaseConfig	A TimeBaseConfig object that contains all the necessary information to configure the TB.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Allows the Application to configure/map a locally defined TB using a TimeBaseConfig.	

Table 8.20: Method definition - configure

8.2.3 Specific Function Definition of Time Bases

The function definitions on this chapter are those of the different Time Base classes.

For more information on the classes of the Time Base design and/or to consult a specific class/method, please refer to Figure 7.1 and to section 7.1.

8.2.3.1 OffsetSlaveTB::OffsetSlaveTB

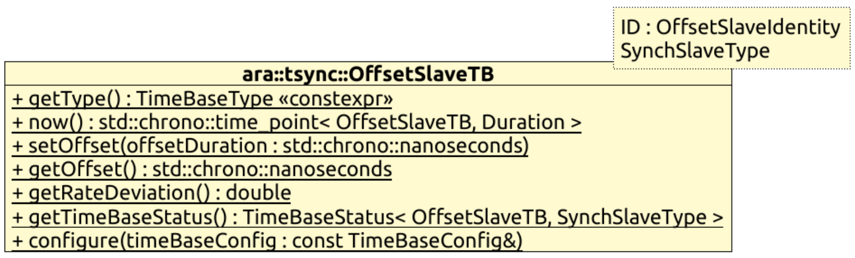

ID : OffsetSlaveIdentity
SynchSlaveType
ara::tsync::OffsetSlaveTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< OffsetSlaveTB, Duration >
+ setOffset(offsetDuration : std::chrono::nanoseconds)
+ getOffset() : std::chrono::nanoseconds
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< OffsetSlaveTB, SynchSlaveType >
+ configure(timeBaseConfig : const TimeBaseConfig&)

Figure 8.5: Class Diagram of the OffsetSlaveTB.

[SWS_TS_00095] ⌈ Constructor `OffsetSlaveTB` shall be defined as described in table 8.21. ⌋*(RS_TS_00012)*

Method name:	OffsetSlaveTB	
Syntax:	`OffsetSlaveTB () = delete`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:		
Description:	Explicitly deleted constructor to prevent instantiation.	

Table 8.21: Constructor definition - OffsetSlaveTB

8.2.3.2 OffsetSlaveTB::now

[SWS_TS_00090] ⌈ Method `now` shall be defined as described in table 8.22. ⌋ *(RS_TS_00026, RS_TS_00005)*

Method name:	now	
Syntax:	`template <typename Duration = duration> static std::chrono::time_point<OffsetSlaveTB, Duration> now ()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::time_point <OffsetSlaveTB, Duration>	The point in time at which this method was called.
Description:	Returns the point in time at which this method was called. The *time_point* represents a "Duration" interval relative to the start of the Clock's epoch.	

Table 8.22: Method definition - now

[SWS_TS_00092] ⌈ The time point offered shall be relative to the epoch of the OffsetSlaveTB, from which this method is called. ⌋*(RS_TS_00001, RS_TS_00002)*

8.2.3.3 OffsetSlaveTB::setOffset

[SWS_TS_00195] ⌈ Method `setOffset` shall be defined as described in table 8.23. ⌋ *(RS_TS_00022)*

Method name:	setOffset	
Syntax:	`static void setOffset(std::chrono::nanoseconds offsetTimestamp)`	
Sync/Async:	Synchronous	
Parameters (in):	std::chrono::nanoseconds offsetTimestamp	The offset in nanoseconds, that shall be added to the clock value of the underlying SyncSlaveTB.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	This Method sets the offset, that will be applied to the clock value of the underlying SyncSlaveTB when calling OffsetSlaveTB::now().	

Table 8.23: Method definition - setOffset

8.2.3.4 OffsetSlaveTB::getOffset

Method name:	getOffset	
Syntax:	`static std::chrono::nanoseconds getOffset()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::nanoseconds	Current Offset relative to the underlying Synchronized TB.
Description:	Returns the Offset of this TBR in relation to the underlying Synchronized TB.	

Table 8.24: Method definition - getOffset

8.2.3.5 SynchSlaveTB::SynchSlaveTB

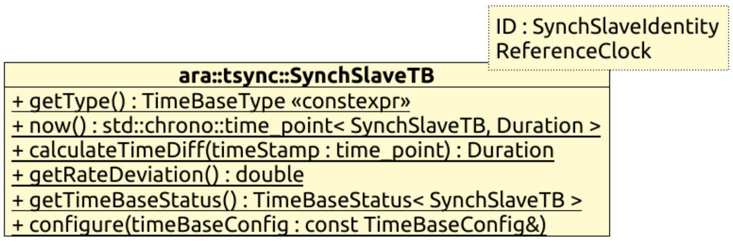

ID : SynchSlaveIdentity
ReferenceClock
ara::tsync::SynchSlaveTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< SynchSlaveTB, Duration >
+ calculateTimeDiff(timeStamp : time_point) : Duration
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< SynchSlaveTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)

Figure 8.6: Class Diagram of the SynchSlaveTB.

[SWS_TS_00096] ⌈ Constructor `SynchSlaveTB` shall be defined as described in table 8.25. ⌋ *(RS_TS_00026)*

Method name:	SynchSlaveTB	
Syntax:	`SynchSlaveTB () = delete`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:		
Description:	Explicitly deleted constructor to prevent instantiation.	

Table 8.25: Constructor definition - SynchSlaveTB

8.2.3.6 SynchSlaveTB::calculateTimeDiff

[SWS_TS_00097] ⌈ Method `calculateTimeDiff` shall be defined as described in table 8.26. ⌋*(RS_TS_00002)*

Method name:	calculateTimeDiff	
Syntax:	`template <typename Duration = duration> static Duration calculateTimeDiff(time_point timeStamp)`	
Sync/Async:	Synchronous	
Parameters (in):	timeStamp	A *time_point* to be subtracted from the time point at which this method was called.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	duration	The difference of current time stamp (at the point in time where this method is called) minus the time stamp passed as parameter.
Description:	Returns the time difference of current time (at point in time where this method was called) minus given time, by using a most accurate time source.	

Table 8.26: Method definition - calculateTimeDiff

8.2.3.7 SynchSlaveTB::now

[SWS_TS_00031] ⌈ Method `now` shall be defined as described in table 8.27. ⌋ *(RS_TS_00026, RS_TS_00005)*

Method name:	now	
Syntax:	`template <typename Duration = duration> static std::chrono::time_point<SynchSlaveTB, Duration> now ()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	

Return value:	std::chrono::time_point <SynchSlaveTB, Duration>	The point in time at which this method was called.
Description:	Returns the point in time at which this method was called. The *time_point* represents a "Duration" interval relative to the start of the Clock's epoch.	

Table 8.27: Method definition - now

[SWS_TS_00091] ⌈ The time point offered shall be relative to the epoch of the SynchSlaveTB, from which this method is called. ⌋*(RS_TS_00001, RS_TS_00002)*

8.2.3.8 PureLocalTB::PureLocalTB

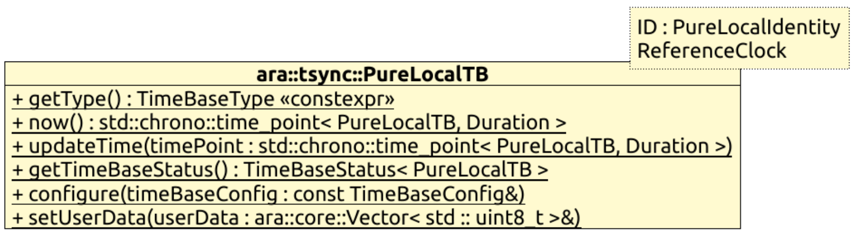

ID : PureLocalIdentity ReferenceClock
ara::tsync::PureLocalTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< PureLocalTB, Duration >
+ updateTime(timePoint : std::chrono::time_point< PureLocalTB, Duration >)
+ getTimeBaseStatus() : TimeBaseStatus< PureLocalTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)

Figure 8.7: Class Diagram of the PureLocalTB.

[SWS_TS_00111] ⌈ Constructor `PureLocalTB` shall be defined as described in table 8.28. ⌋*(RS_TS_00026)*

Method name:	PureLocalTB	
Syntax:	`PureLocalTB () = delete`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Explicitly deleted constructor to prevent instantiation.	

Table 8.28: Constructor definition - PureLocalTB

8.2.3.9 PureLocalTB::now

[SWS_TS_00128] ⌈ Method `now` shall be defined as described in table 8.29. ⌋*(RS_TS_00026, RS_TS_00005)*

Method name:	now	
Syntax:	`template <typename Duration = duration> static std::chrono::time_point<PureLocalTB, Duration> now()`	
Sync/Async:	Synchronous	
Parameters (in):	none	

Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::time_point <PureLocalTB, Duration>	The point in time at which this method was called.
Description:	Returns the point in time at which this method was called. The *time_point* represents a "Duration" interval relative to the start of the Clock's epoch.	

Table 8.29: Method definition - now

[SWS_TS_00150] ⌈ The time point offered shall be relative to the clock of the PureLocalTB, from which this method is called. ⌋*(RS_TS_00001, RS_TS_00002)*

8.2.3.10 OffsetMasterTB::OffsetMasterTB

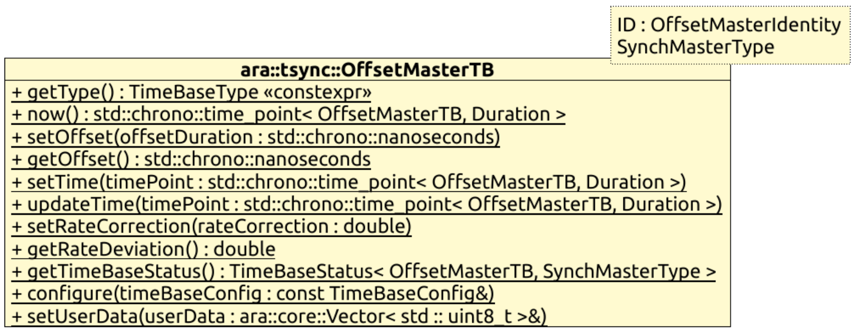

ID : OffsetMasterIdentity
SynchMasterType
ara::tsync::OffsetMasterTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< OffsetMasterTB, Duration >
+ setOffset(offsetDuration : std::chrono::nanoseconds)
+ getOffset() : std::chrono::nanoseconds
+ setTime(timePoint : std::chrono::time_point< OffsetMasterTB, Duration >)
+ updateTime(timePoint : std::chrono::time_point< OffsetMasterTB, Duration >)
+ setRateCorrection(rateCorrection : double)
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< OffsetMasterTB, SynchMasterType >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)

Figure 8.8: Class Diagram of the OffsetMasterTB.

[SWS_TS_00116] ⌈ Constructor `OffsetMasterTB` shall be defined as described in table 8.30. ⌋*(RS_TS_00026)*

Method name:	OffsetMasterTB	
Syntax:	`OffsetMasterTB () = delete`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:		
Description:	Explicitly deleted constructor to prevent instantiation.	

Table 8.30: Constructor definition - OffsetMasterTB

8.2.3.11 OffsetMasterTB::setOffset

[SWS_TS_00112] ⌈ Method `setOffset` shall be defined as described in table 8.31. ⌋ *(RS_TS_00001, RS_TS_00026, RS_TS_00013)*

Method name:	setOffset	
Syntax:	`static void setOffset( std::chrono::nanoseconds offsetDuration )`	
Sync/Async:	Synchronous	
Parameters (in):	offsetDuration	Offset against the Synchronized TB this TBR is based on.
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	void	
Description:	Allows the Application and the TSP Modules to set the Offset Time.	

Table 8.31: Method definition - setOffset

[SWS_TS_00113] ⌈ Implementation of `setOffset()` method in the `OffsetMasterTB` shall check if the TBR is configured to act as Global Time Base and in case it is, it shall set the Offset which will be relative to the underlying Synchronized TB. ⌋ *(RS_TS_00001, RS_TS_00026, RS_TS_00013)*

8.2.3.12 OffsetMasterTB::getOffset

[SWS_TS_00114] ⌈ Method `getOffset` shall be defined as described in table 8.32. ⌋ *(RS_TS_00001, RS_TS_00026, RS_TS_00012)*

Method name:	getOffset	
Syntax:	`static std::chrono::nanoseconds getOffset()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	duration	Current Offset relative to the underlying Synchronized TB.
Description:	Returns the Offset of this TBR in relation to the underlying Synchronized TB.	

Table 8.32: Method definition - getOffset

8.2.3.13 OffsetMasterTB::now

[SWS_TS_00151] ⌈ Method `now` shall be defined as described in table 8.33. ⌋ *(RS_TS_00026, RS_TS_00005)*

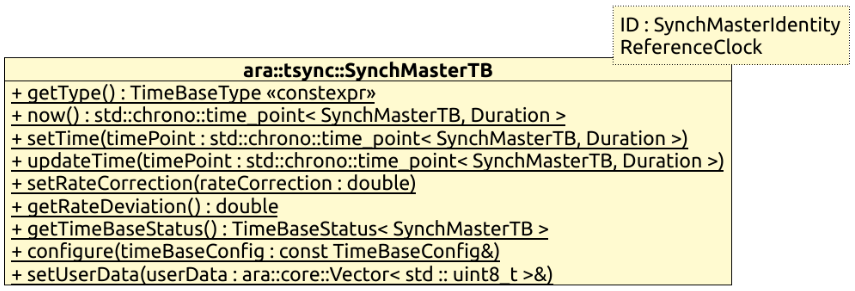
Method name:	now
Remarks:	
Syntax:	`template <typename Duration = duration> static std::chrono::time_point<OffsetMasterTB, Duration> now()`
Sync/Async:	Synchronous

Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::time_point <OffsetMasterTB, Duration>	The point in time at which this method was called.
Description:	Returns the point in time at which this method was called. The *time_point* represents a "Duration" interval relative to the start of the Clock's epoch.	

Table 8.33: Method definition - now

[SWS_TS_00152] ⌈ The time point offered shall be relative to the clock of the Offset-MasterTB, from which this method is called. ⌋*(RS_TS_00001, RS_TS_00002)*

8.2.3.14 SynchMasterTB::SynchMasterTB


ID : SynchMasterIdentity
ReferenceClock
ara::tsync::SynchMasterTB
+ getType() : TimeBaseType «constexpr»
+ now() : std::chrono::time_point< SynchMasterTB, Duration >
+ setTime(timePoint : std::chrono::time_point< SynchMasterTB, Duration >)
+ updateTime(timePoint : std::chrono::time_point< SynchMasterTB, Duration >)
+ setRateCorrection(rateCorrection : double)
+ getRateDeviation() : double
+ getTimeBaseStatus() : TimeBaseStatus< SynchMasterTB >
+ configure(timeBaseConfig : const TimeBaseConfig&)
+ setUserData(userData : ara::core::Vector< std :: uint8_t >&)
Figure 8.9: Class Diagram of the SynchMasterTB.

[SWS_TS_00117] ⌈ Constructor `SynchMasterTB` shall be defined as described in table 8.34. ⌋*(RS_TS_00026)*

Method name:	SynchMasterTB	
Syntax:	`SynchMasterTB () = delete`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:		
Description:	Explicitly deleted constructor to prevent instantiation.	

Table 8.34: Constructor definition - SynchMasterTB

8.2.3.15 SynchMasterTB::now

[SWS_TS_00153] ⌈ Method `now` shall be defined as described in table 8.35. ⌋*(RS_TS_00026, RS_TS_00005)*

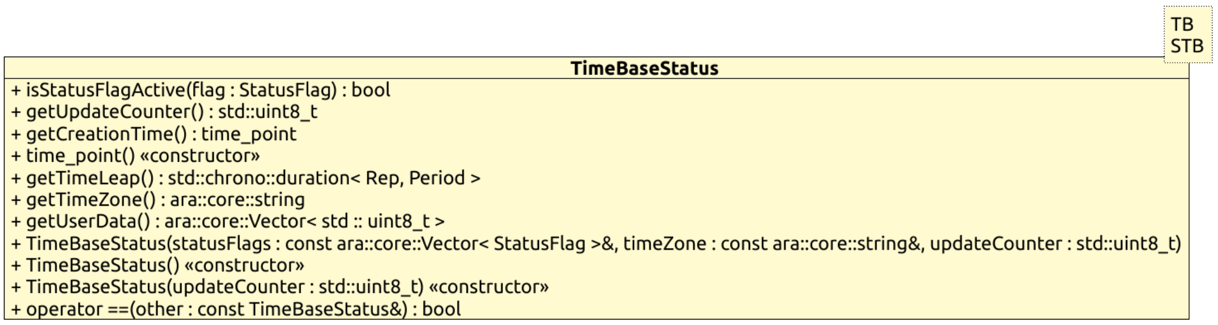
Method name:	now	
Syntax:	`template <typename Duration = duration> static std::chrono::time_point<SynchMasterTB, Duration> now()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::time_point <SynchMasterTB, Duration>	The point in time at which this method was called.
Description:	Returns the point in time at which this method was called. The *time_point* represents a "Duration" interval relative to the start of the Clock's epoch.	

Table 8.35: Method definition - now

[SWS_TS_00154] ⌈ The time point offered shall be relative to the clock of the SynchMasterTB, from which this method is called. ⌋*(RS_TS_00001, RS_TS_00002)*

8.2.4 TimeBaseStatus

TimeBaseStatus is a templated class, that takes two template arguments. The first one, TB, is the TB-type that is creating the TimeBaseStatus. The second one, STB, is the type of the underlying TB. Both types need to follow the chrono clock paradigm 8.1.4.


TB
STB
TimeBaseStatus
+ isStatusFlagActive(flag : StatusFlag) : bool
+ getUpdateCounter() : std::uint8_t
+ getCreationTime() : time_point
+ time_point() «constructor»
+ getTimeLeap() : std::chrono::duration< Rep, Period >
+ getTimeZone() : ara::core::string
+ getUserData() : ara::core::Vector< std :: uint8_t >
+ TimeBaseStatus(statusFlags : const ara::core::Vector< StatusFlag >&, timeZone : const ara::core::string&, updateCounter : std::uint8_t)
+ TimeBaseStatus() «constructor»
+ TimeBaseStatus(updateCounter : std::uint8_t) «constructor»
+ operator ==(other : const TimeBaseStatus&) : bool
Figure 8.10: TimeBaseStatus and StatusFlags

8.2.4.1 TimeBaseStatus::isStatusFlagActive

[SWS_TS_00118] ⌈ Method `isStatusFlagActive` shall be defined as described in table 8.36. ⌋*(RS_TS_00021)*

Method name:	isStatusFlagActive	
Syntax:	`bool isStatusFlagActive( StatusFlag flag) const`	
Sync/Async:	Synchronous	
Parameters (in):	flag	Value of the StatusFlag enumeration

Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	bool	True: inquired flag is set False: inquired flag is not set
Description:	Returns a Boolean indicating whether the flag passed as parameter is set or not.	

Table 8.36: Method definition - isStatusFlagActive

8.2.4.2 TimeBaseStatus::getUserData

[SWS_TS_00119] ⌈ Method `getUserData` shall be defined as described in table 8.37. ⌋*(RS_TS_00021, RS_TS_00014)*

Method name:	getUserData	
Syntax:	`ara::core::Vector<uint8_t> getUserData () const`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	ara::core::Vector<uint8_t>	Vector containing the current User Data
Description:	Returns a vector containing the current User Data as it was set by the method "setUserData()".	

Table 8.37: Method definition - getUserData

[SWS_TS_00120] ⌈ In case the TBR has no User Data stored, an empty vector shall be returned. ⌋*(RS_TS_00014, RS_TS_00021)*

8.2.4.3 TimeBaseStatus::getUpdateCounter

[SWS_TS_00121] ⌈ Method `getUpdateCounter` shall be defined as described in table 8.38. ⌋*(RS_TS_00021)*

Method name:	getUpdateCounter	
Syntax:	`uint8_t getUpdateCounter () const`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	uint8_t	Value of the update counter of this TBR
Description:	Returns the value of the update counter of this TBR. Value of the counter is not particularly of interest. This counter serves to the purpose of giving a notion of whether the TBR is being synchronized or not. Please refer to subsection 7.4.6 for additional information.	

Table 8.38: Method definition - getUpdateCounter

8.2.4.4 TimeBaseStatus::getCreationTime

[SWS_TS_00122] ⌈ Method `getCreationTime` shall be defined as described in table 8.39. ⌋*(RS_TS_00021)*

Method name:	getCreationTime	
Syntax:	`TB::time_point getCreationTime() const`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TB::time_point	Point in time bounded to the specific type of TBR template argument "TB".
Description:	Returns the point in time at which this "TimeBaseStatus" object was created.	

Table 8.39: Method definition - getCreationTime

[SWS_TS_00123] ⌈ The return *time_point* value shall be based on the Clock its TBR is based on as well as on its resolution. ⌋*(RS_TS_00021)*

8.2.4.5 TimeBaseStatus::getTimeLeap

[SWS_TS_00125] ⌈ Method `getTimeLeap` shall be defined as described in table 8.40. ⌋*(RS_TS_00021)*

Method name:	getTimeLeap	
Syntax:	`template <typename Rep, typename Period> std::chrono::duration<Rep, Period> getTimeLeap () const`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	std::chrono::duration<Rep, Period>	Time Leap value
Description:	Returns the duration of the current leap - if the corresponding leap threshold flag is set.	

Table 8.40: Method definition - getTimeLeap

8.2.4.6 TimeBaseStatus::getTimeZone

[SWS_TS_00149] ⌈ Method `getTimeZone` shall be defined as described in table 8.41. ⌋*(RS_TS_00021)*

Method name:	getTimeZone
Syntax:	`ara::core::string getTimeZone () const`

Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	ara::core::string	The time zone this time base adheres to.
Description:	Returns the information of the time zone as a ara::core::string.	

Table 8.41: Method definition - getTimeZone

8.2.4.7 TimeBaseStatusImpl::getSynchStatus

[SWS_TS_00126] ⌈ Method `getSynchStatus` shall be defined as described in table 8.42. ⌋*(RS_TS_00021)*

Method name:	getSynchStatus	
Remarks:	TimeBaseStatusImpl is a helper that is used to achieve one common implementation for TimeBaseStatus<TB,TB> and TimeBaseStatus<TB,STB>. Please have a look at the demonstrator code to see how it needs to be implemented.	
Syntax:	`TimeBaseStatus<STB, STB> getSynchStatus () const`	
Implemented by:	template <typename TB> struct TimeBaseStatusImpl<TB, TB> template <typename TB, typename STB> struct TimeBaseStatusImpl	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TimeBaseStatus	"TimeBaseStatus" object
Description:	In the Offset TBRs, this method returns a copy of a local "TimeBaseStatus" object corresponding to the Synch TBR this Offset TBR is based on.	

Table 8.42: Method definition - getSynchStatus

[SWS_TS_00127] ⌈ For `TimeBaseStatus` objects that correspond to a Synchronized TBR, this method shall return a copy of the same `TimeBaseStatus` object this method belongs to. ⌋*(RS_TS_00021)*

[SWS_TS_00129] ⌈ For `TimeBaseStatus` objects that correspond to an Offset TBR, the TimeBaseStatus object returned by this method shall contain the related information of the Synchronized TBR associated to the Offset TBR this `TimeBaseStatus` object corresponds to. ⌋*(RS_TS_00021)*

[SWS_TS_00131] ⌈ The creation time of the Offset TBR's `TimeBaseStatus` object and the creation time of the Sinchronized TBR associated to the Offset TBR this `TimeBaseStatus` object corresponds to, shall be identical. ⌋*(RS_TS_00021)*

8.2.4.8 TimeBaseStatus::operator TB::time_point()

[SWS_TS_00130] ⌈ `operator` shall be defined as described in table 8.43. ⌋ *(RS_TS_00021)*

Method name:	operator TB::time_point()	
Syntax:	`operator typename TB::time_point ()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:	TB::time_point	Point in time relative to a given clock TB.
Description:	In the Offset TBRs, this method returns a *time_point* providing the point in time when this object was created (same as the method getCreation-Time()).	

Table 8.43: Method definition - operator

8.2.4.9 TimeBaseStatus::TimeBaseStatus

[SWS_TS_00143] ⌈ One constructor for `TimeBaseStatus` shall be defined as described in table 8.44. ⌋*(RS_TS_00021)*

Method name:	TimeBaseStatus	
Syntax:	`TimeBaseStatus()`	
Sync/Async:	Synchronous	
Parameters (in):	none	
Parameters (in-/out):	none	
Parameters (out):	none	
Return value:		
Description:	Constructor of the "TimeBaseStatus". To be used by the OffsetSlaveTB, because all the information is stored in the underlying SynchSlaveTB.	

Table 8.44: Constructor definition - TimeBaseStatus

[SWS_TS_00145] ⌈ One constructor for `TimeBaseStatus` shall be defined as described in table 8.45. ⌋*(RS_TS_00021)*

Method name:	TimeBaseStatus	
Syntax:	`explicit TimeBaseStatus(std::uint8_t updateCounter)`	
Sync/Async:	Synchronous	
Parameters (in):	updateCounter	Value of the updateCounter, passed by the Offset-MasterTB.
Parameters (in-/out):	none	
Parameters (out):	none	
Description:	Constructor of the "TimeBaseStatus". To be used by the OffsetMasterTB, because all the information is stored in the underlying SynchMasterTB, except the updateCounter that has to be maintained by the OffsetMasterTB itself.	

Table 8.45: Constructor definition - TimeBaseStatus

9 Sequence diagrams

The following diagrams intend to depict the usage of the TS API, specifically when it is required that some internal interaction between different Time Bases takes place.

These sequence diagrams should be taken as illustrational purposes only.

9.1 Application "finds" a resource.

The following diagram shows how the application finds a TBR as well as how the TBR proxy is instantiated to then interact with it (i.e. starting a timer or obtaining the TBR's specialized interface from it.

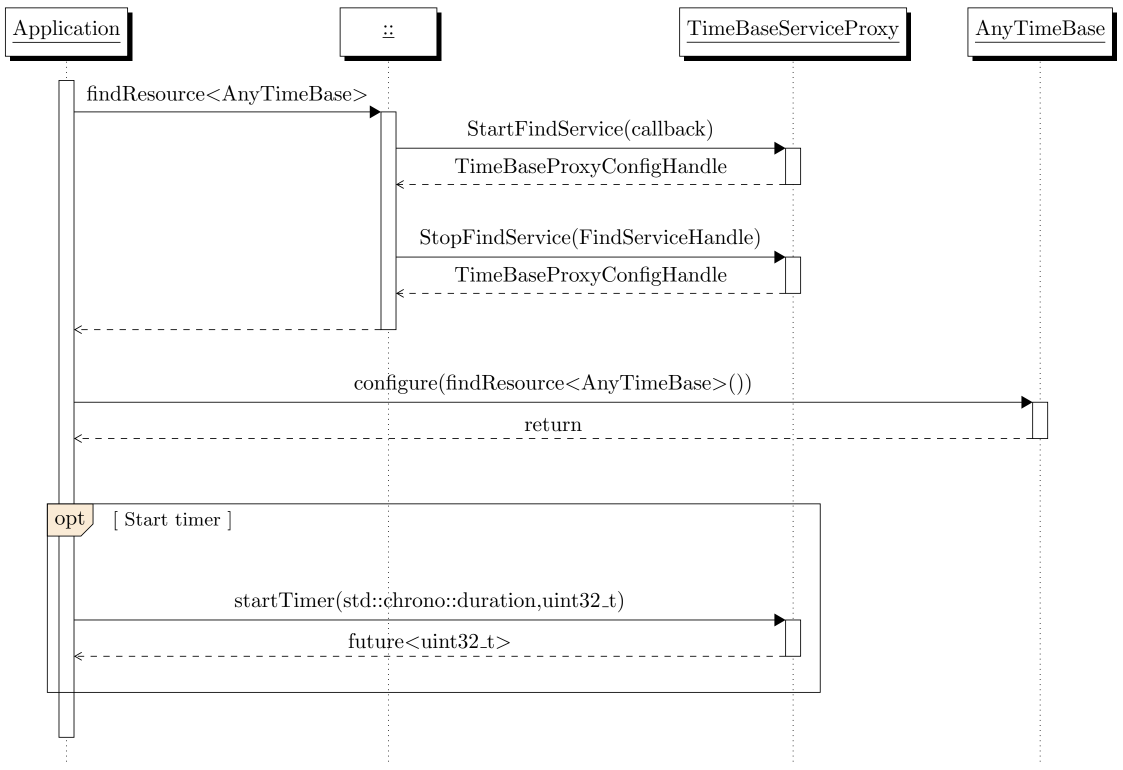

Application
::
TimeBaseServiceProxy
AnyTimeBase
findResource<AnyTimeBase>
StartFindService(callback)
TimeBaseProxyConfigHandle
StopFindService(FindServiceHandle)
TimeBaseProxyConfigHandle
configure(findResource<AnyTimeBase>())
return
opt
[Start timer]
startTimer(std::chrono::duration,uint32_t)
future<uint32_t>

Figure 9.1: Application finds a TBR

9.2 Application starts a Timer with the instantiated proxy of a Handle

The following diagrams show how the application can "subscribe" itself for the timer feature and how it then can be triggered, once the time has expired.

The figures below depict a use case in which the user polls for the Future object to inquire for the status of the timer. For more information about the Future Objects and the possibilities that they offer, to make their asynchronous value available, please refer to [11].

9.2.1 Querying for the Future<T>.valid() method of the returned object.

This diagram shows how the application can query for the status of the timer by means of the valid method of the future object it was returned to it.

9.2.2 Querying for the Future<T>.wait_for() method of the returned object.

This diagram shows how the application can query for the status of the timer by means of the wait_for method of the future object. The duration specified in StartTimer is independent of the configured TBs and will potentially be using a different underlying clock.

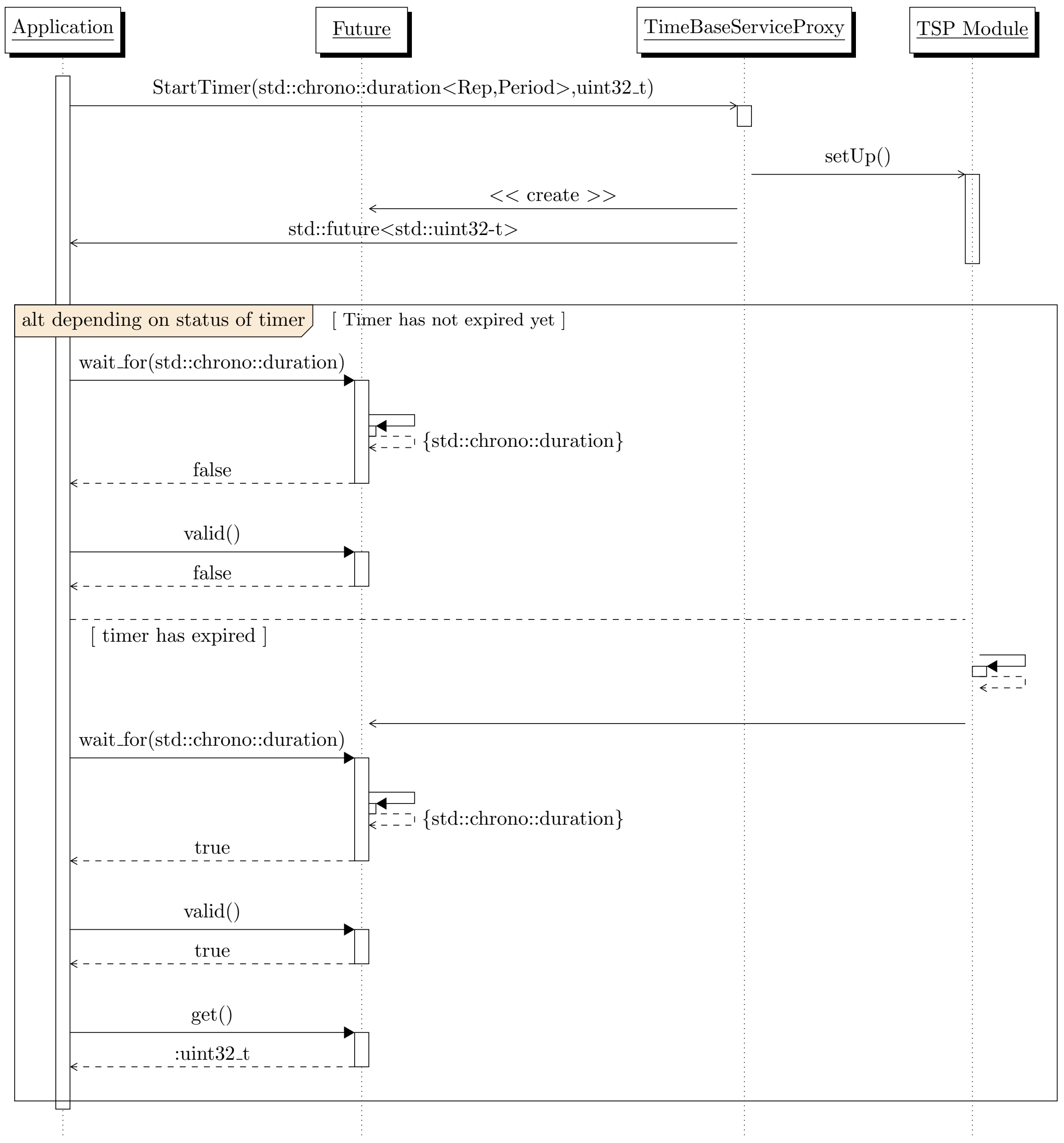

Application
Future
TimeBaseServiceProxy
TSP Module
StartTimer(std::chrono::duration<Rep,Period>,uint32_t)
setUp()
<< create >>
std::future<std::uint32-t>
alt depending on status of timer
[Timer has not expired yet]
wait_for(std::chrono::duration)
{std::chrono::duration}
false
valid()
false
[timer has expired]
wait_for(std::chrono::duration)
{std::chrono::duration}
true
valid()
true
get()
:uint32_t

Figure 9.2: StartTimer - query for valid

9.3 Interaction with Offset Time Bases

This diagram shows the mechanism used to provide the current time of an Offset TBR. It also shows how the Application can query for its underlying Synchronized TBR.

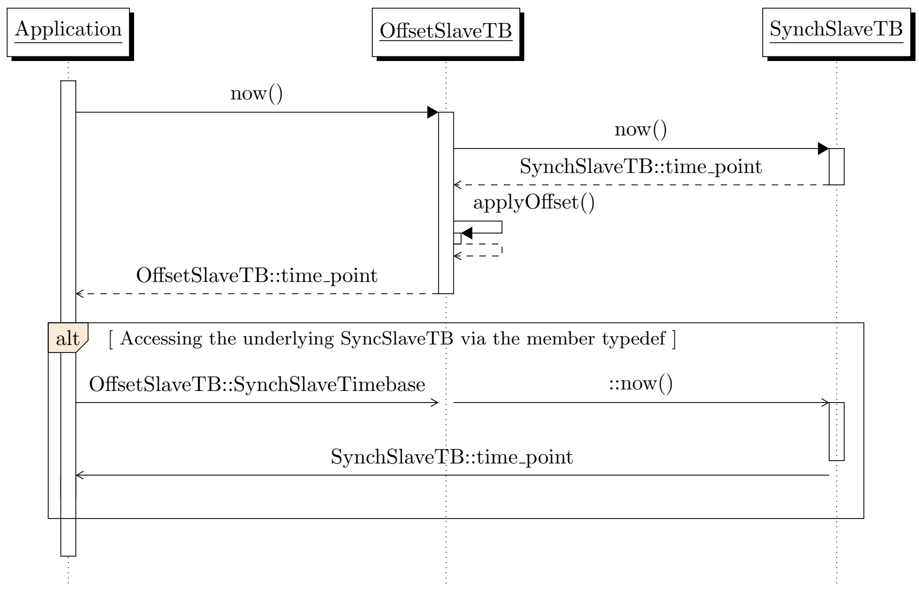

Application
OffsetSlaveTB
SynchSlaveTB
now()
now()
SynchSlaveTB::time_point
applyOffset()
OffsetSlaveTB::time_point
alt
[Accessing the underlying SyncSlaveTB via the member typedef]
OffsetSlaveTB::SynchSlaveTimebase
::now()
SynchSlaveTB::time_point

Figure 9.3: Offset Time Base Handling.

9.4 Application request status of a Synchronized TBR - and then takes information from such status.

This diagram shows how the application queries for the status of a Synchronized TBR and how it can then get specific status information. The application queries for the specifics of a TBR Status in the same way on any Type of TBR.

For Synchronized Time Base resources, the method `getSynchStatus()` will return a copy of the same `TimeBaseStatus` object.

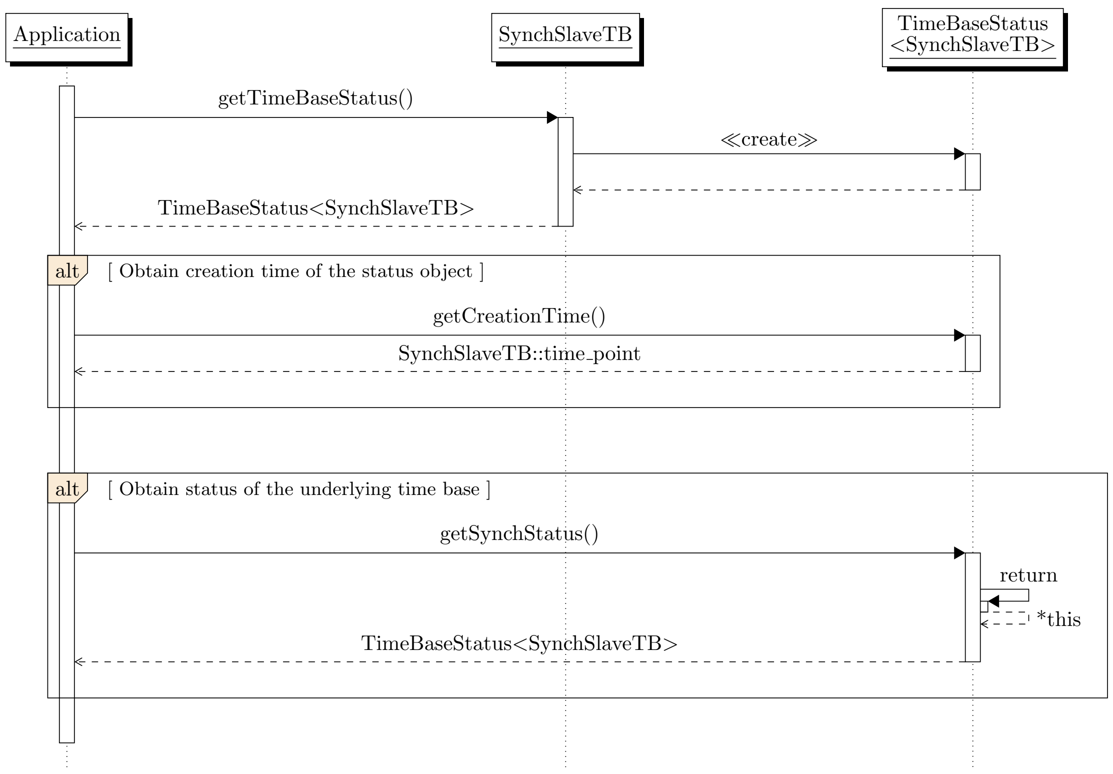

Application
SynchSlaveTB
TimeBaseStatus <SynchSlaveTB>
getTimeBaseStatus()
≪create≫
TimeBaseStatus<SynchSlaveTB>
alt
[Obtain creation time of the status object]
getCreationTime()
SynchSlaveTB::time_point
alt
[Obtain status of the underlying time base]
getSynchStatus()
return
*this
TimeBaseStatus<SynchSlaveTB>

Figure 9.4: Request time base status of SynchTB.

9.5 Application request status of an Offset TBR

This diagram shows how the application queries for the status of an Offset TBR.

For Offset Time Base resources, the method `getSynchStatus()` will return a copy of the underlying Synchronized TBR of the Offset TBR in question. The Application will then be able to query for specifics on both the `TimeBaseStatus` objects of the Offset TB as well as its underlying Synchronized TB.

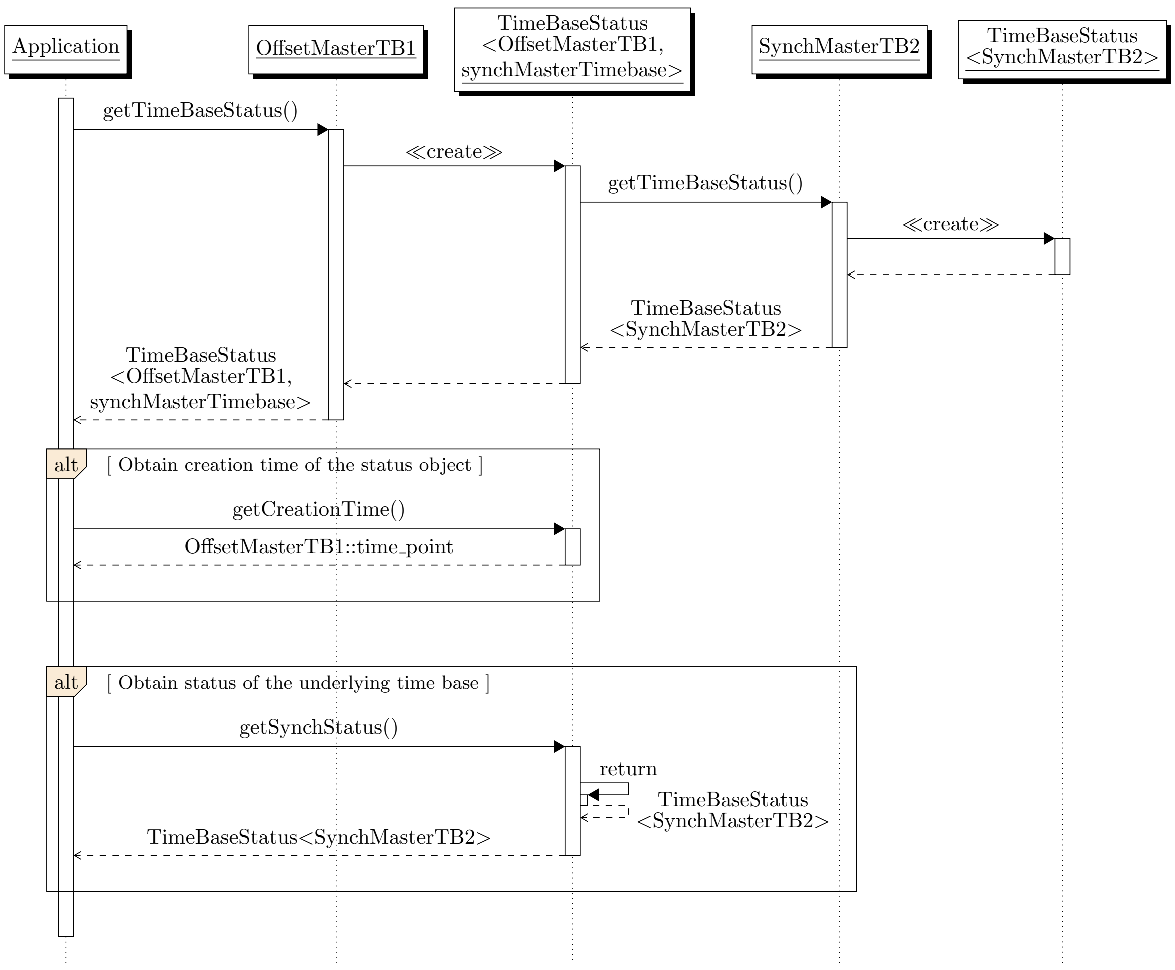

Application
OffsetMasterTB1
TimeBaseStatus <OffsetMasterTB1, synchMasterTimebase>
SynchMasterTB2
TimeBaseStatus <SynchMasterTB2>
getTimeBaseStatus()
≪create≫
getTimeBaseStatus()
≪create≫
TimeBaseStatus <SynchMasterTB2>
TimeBaseStatus <OffsetMasterTB1, synchMasterTimebase>
alt
[Obtain creation time of the status object]
getCreationTime()
OffsetMasterTB1::time_point
alt
[Obtain status of the underlying time base]
getSynchStatus()
return
TimeBaseStatus <SynchMasterTB2>
TimeBaseStatus<SynchMasterTB2>

Figure 9.5: Request time base status of OffsetTB