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

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R19-11

Document Change History

Date	Release	Changed by	Description
2019-11-28	R19-11	AUTOSAR Release Management	• Requirements traceability changed to Foundation RS TimeSync specification • Add Time Validation • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Functional description detached from actual API • Improved resource discovery
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

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.

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.

The current concept on TimeSync is not in line with the port prototype approach. The topic on InstanceSpecifier is not yet finalized. Further changes to be expected in R19-11.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the Time Synchronization module that are not included in the [3, 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.

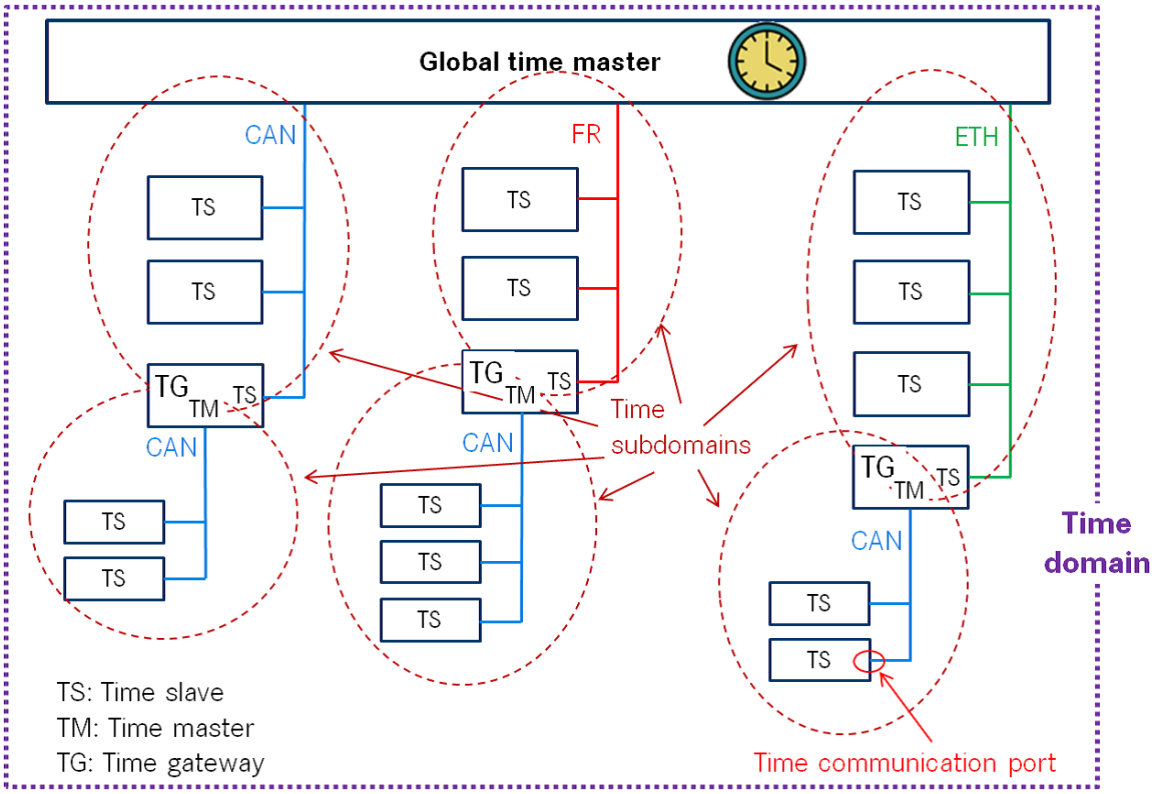

Figure 2.1: Terminology Example

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] Glossary
AUTOSAR_TR_Glossary

[4] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[5] Functional Cluster Shortnames
AUTOSAR_TR_FunctionalClusterShortnames

[6] Requirements on Time Synchronization for Adaptive Platform
AUTOSAR_RS_TimeSync

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

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

[9] Specification of Time Synchronization over Ethernet
AUTOSAR_SWS_TimeSyncOverEthernet

[10] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [4, 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).

The current concept on TimeSync is not in line with the port prototype approach. The topic on InstanceSpecifier is not yet finalized. Further changes to be expected in R19-11.

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 [5] namespace.

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Time Synchronization for Adaptive Platform [6] 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_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_00102] [SWS_TS_00108] [SWS_TS_00128] [SWS_TS_00128] [SWS_TS_00150] [SWS_TS_00151] [SWS_TS_00152] [SWS_TS_00154] [SWS_TS_00338] [SWS_TS_00339] [SWS_TS_00339] [SWS_TS_00344] [SWS_TS_00384] [SWS_TS_00413]
[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] [SWS_TS_0094]
[RS_TS_00005]	The Implementation of Time Synchronization shall allow customers to have access to the Synchronized Time Base	[SWS_TS_00014] [SWS_TS_00022] [SWS_TS_00031] [SWS_TS_00090] [SWS_TS_00128] [SWS_TS_00153] [SWS_TS_00341] [SWS_TS_00352] [SWS_TS_00373] [SWS_TS_00374] [SWS_TS_00375] [SWS_TS_00376] [SWS_TS_00377] [SWS_TS_00381]
[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] [SWS_TS_00327] [SWS_TS_00328]
[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_00115] [SWS_TS_00139] [SWS_TS_00140] [SWS_TS_00141] [SWS_TS_00324] [SWS_TS_00327] [SWS_TS_00328] [SWS_TS_00339] [SWS_TS_00344] [SWS_TS_00348]

Requirement	Description	Satisfied by
[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_00099] [SWS_TS_00100] [SWS_TS_00101] [SWS_TS_00102] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00108] [SWS_TS_00110] [SWS_TS_00327] [SWS_TS_00328] [SWS_TS_00339] [SWS_TS_00347] [SWS_TS_00348]
[RS_TS_00011]	The Implementation of Time Synchronization shall allow customers on master side to trigger time transmission by the TSP module	[SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00110] [SWS_TS_00328] [SWS_TS_00339] [SWS_TS_00348]
[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_00114] [SWS_TS_00334]
[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_00342] [SWS_TS_00353] [SWS_TS_00413]
[RS_TS_00016]	The Implementation of Time Synchronization shall notify customers about status events	[SWS_TS_00064] [SWS_TS_00201] [SWS_TS_00387] [SWS_TS_00388] [SWS_TS_00389] [SWS_TS_00390] [SWS_TS_00391] [SWS_TS_00392] [SWS_TS_00393]
[RS_TS_00017]	The Implementation of Time Synchronization shall notify customers about elapsed pre-defined time span.	[SWS_TS_00064] [SWS_TS_00387] [SWS_TS_00388] [SWS_TS_00389] [SWS_TS_00390] [SWS_TS_00391] [SWS_TS_00392] [SWS_TS_00393]
[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]

Requirement	Description	Satisfied by
		[SWS_TS_00084] [SWS_TS_00142] [SWS_TS_00329] [SWS_TS_00335] [SWS_TS_00349] [SWS_TS_00350] [SWS_TS_00379]
[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_00118] [SWS_TS_00119] [SWS_TS_00120] [SWS_TS_00121] [SWS_TS_00122] [SWS_TS_00123] [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_00330] [SWS_TS_00336] [SWS_TS_00340] [SWS_TS_00351] [SWS_TS_00354] [SWS_TS_00355] [SWS_TS_00356] [SWS_TS_00357] [SWS_TS_00380]
[RS_TS_00022]	The Implementation of Time Synchronization shall support custom clocks	[SWS_TS_00078] [SWS_TS_00132] [SWS_TS_00195] [SWS_TS_00326] [SWS_TS_00333] [SWS_TS_00338] [SWS_TS_00345] [SWS_TS_00346] [SWS_TS_00378] [SWS_TS_00411]
[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] [SWS_TS_00203] [SWS_TS_00331] [SWS_TS_00337] [SWS_TS_00343] [SWS_TS_00358] [SWS_TS_00359] [SWS_TS_00360] [SWS_TS_00361] [SWS_TS_00362] [SWS_TS_00363] [SWS_TS_00364] [SWS_TS_00365] [SWS_TS_00366] [SWS_TS_00367] [SWS_TS_00368] [SWS_TS_00369] [SWS_TS_00370] [SWS_TS_00371] [SWS_TS_00372] [SWS_TS_00410] [SWS_TS_00412]
[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_00072] [SWS_TS_00085] [SWS_TS_00090] [SWS_TS_00099] [SWS_TS_00100] [SWS_TS_00101] [SWS_TS_00102] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00108] [SWS_TS_00110] [SWS_TS_00112] [SWS_TS_00113] [SWS_TS_00115] [SWS_TS_00128] [SWS_TS_00133] [SWS_TS_00134] [SWS_TS_00153] [SWS_TS_00327]

Requirement	Description	Satisfied by
		[SWS_TS_00328] [SWS_TS_00329] [SWS_TS_00339] [SWS_TS_00342] [SWS_TS_00344] [SWS_TS_00347] [SWS_TS_00348] [SWS_TS_00349] [SWS_TS_00353] [SWS_TS_00394] [SWS_TS_00395] [SWS_TS_00396] [SWS_TS_00397] [SWS_TS_00398] [SWS_TS_00399] [SWS_TS_00413] [SWS_TS_0094]
[RS_TS_00029]	The configuration of the Time Synchronization implementation shall allow the implementation to behave as a (vehicle wide) Time Master	[SWS_TS_00008] [SWS_TS_00009] [SWS_TS_00104] [SWS_TS_00105] [SWS_TS_00106] [SWS_TS_00107] [SWS_TS_00112] [SWS_TS_00113] [SWS_TS_00115] [SWS_TS_00132] [SWS_TS_00133] [SWS_TS_00151] [SWS_TS_00152] [SWS_TS_00154] [SWS_TS_00326] [SWS_TS_00328] [SWS_TS_00329] [SWS_TS_00342] [SWS_TS_00346] [SWS_TS_00348] [SWS_TS_00349] [SWS_TS_00353] [SWS_TS_00385] [SWS_TS_00386] [SWS_TS_00419] [SWS_TS_00421] [SWS_TS_00423] [SWS_TS_00429]
[RS_TS_00030]	The configuration of the Time Synchronization implementation shall allow the implementation to behave as a Time Slave	[SWS_TS_00008] [SWS_TS_00009] [SWS_TS_00091] [SWS_TS_00092] [SWS_TS_00132] [SWS_TS_00134] [SWS_TS_00333] [SWS_TS_00382] [SWS_TS_00383] [SWS_TS_00411] [SWS_TS_00418] [SWS_TS_00420] [SWS_TS_00422] [SWS_TS_00428] [SWS_TS_0094]
[RS_TS_00034]	The Implementation of Time Synchronization shall provide measurement data to the application	[SWS_TS_00414] [SWS_TS_00415] [SWS_TS_00416] [SWS_TS_00417] [SWS_TS_00418] [SWS_TS_00419] [SWS_TS_00420] [SWS_TS_00421] [SWS_TS_00422] [SWS_TS_00423] [SWS_TS_00424] [SWS_TS_00425] [SWS_TS_00426] [SWS_TS_00427] [SWS_TS_00428] [SWS_TS_00429] [SWS_TS_14140] [SWS_TS_14141] [SWS_TS_14142] [SWS_TS_14150] [SWS_TS_14151] [SWS_TS_14152] [SWS_TS_14153] [SWS_TS_14154] [SWS_TS_14155] [SWS_TS_14156] [SWS_TS_14160] [SWS_TS_14161] [SWS_TS_14162] [SWS_TS_14163] [SWS_TS_14164] [SWS_TS_14165] [SWS_TS_14166] [SWS_TS_14167] [SWS_TS_14170] [SWS_TS_14171] [SWS_TS_14172] [SWS_TS_14173] [SWS_TS_14174]

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 [7]
- The Time posix interface [8]

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 TBRs or -where applies- it overrides certain functionality according to specific needs or requirements to be fulfilled by the given type of TBR.

7.1.1 Base functionality of every Time Base

Every Time Base has to provide a minimum set of functionality, as listed below:

- providing its own type information
- offer possibility to obtain the current clock value
- creating a snapshot of its parameters

This chapter briefly describes these functionalities. Details on how to use and the exact behavior of these core methods are given in chapter 8.

7.1.1.1 Time Base Status

[SWS_TS_00005]{DRAFT} ⌈Every Time Base Resource present in the system shall be able to generate a Status object to be passed to the application that requests it. (i.e. `TimeBaseStatus` object).⌋*(RS_TS_00021)*

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

[SWS_TS_00201]{DRAFT} ⌈Applications shall be a able to query for StatusFlags via the TimeBaseStatus. A list of possible status flags can be found in section 7.1.1.1.⌋ *(RS_TS_00016)*

[SWS_TS_00144]{DRAFT} ⌈In addition, the application shall be able to retrieve user defined data through the TimeBaseStatus. UserData is additional clock information, that is not standardized and can therefore be used to handle vendor/OEM specific data.⌋*(RS_TS_00014)*

7.1.1.2 Time Base Type

[SWS_TS_00132]{DRAFT} ⌈Applications might be interested in querying for the TBR's type. Therefore each TBR shall provide the possible to obtain the TB type information.⌋*(RS_TS_00029, RS_TS_00030, RS_TS_00022)*

7.1.1.3 Rate Deviation

Applications will have different thresholds for acceptable time drift values. Hence there needs to be a way, how applications can access this information.

[SWS_TS_00202]{DRAFT} ⌈It shall be possible to obtain, if already calculated, the rate deviation of a given TBR against the time source it is synchronized to.⌋*()*

7.1.1.4 Clock Time Value

Reading the clock's time value is very likely the most commonly performed operation by the applications interacting with TS.

[SWS_TS_00157]{DRAFT} ⌈It shall not be possible to mix time points from different TBs. Ergo time points need to be specific to one TB.⌋*(RS_TS_00023)*

[SWS_TS_00203]{DRAFT} ⌈To ensure type safe handling of time values, they shall be provided as std::chrono structures.⌋*(RS_TS_00023)*

More detailed information on how this is implemented is given in the further chapters and in chapter 8.

7.1.2 Status Flags of TBRs

Time Synchronization defines a set of status flags that are used to express specific status conditions of a TBR. Status flags can be queried by an application through a `TimeBaseStatus`.

- `Time Out`: Indicates whether a synchronization of a time base to its corresponding TBR is lost or delayed.
- `Synchronized`: Indicates if the time base of the corresponding TBR has been successfully synchronized at least once against its time source.
- `Synchronized To Gateway`: Indicates if the corresponding TBR updates are based on a Time Gateway below the Global Time Master.
- `Time Leap to Future`: Indicates if there has been a jump in time to the future.
- `Time Leap to Past`: Indicates if there has been a jump in time to the past.
- `Has Daylight Saving`: Indicates if a time base, respectively the corresponding TBR, makes use of DLS.
- `Daylight Saving is Active`: Indicates if the DLS is already considered in the time base provided by the corresponding TBR.

7.1.3 Time Synchronization and Protocols

Time Synchronization mechanisms and protocols (i.e. [9] are out of the Scope of this document, for protocol specification please refer to the PRS (see [1]).

7.2 Startup behavior

This chapter describes the necessary initializations, which are performed by the entity that has control over the Time Base Resources, in order to prepare the TS module for normal operation. After its initialization, the module is expected to provide all synchronized time services to the applications.

7.2.1 Default values

When the system starts up, the TBRs have to be set to known default values so that their behavior is well defined.

[SWS_TS_00006]{DRAFT} ⌈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]{DRAFT} ⌈Characteristics of Time Base Resources shall be initialized as follows:

- Active Status Flags shall be invalidated.
- Clock Update Counter shall be set to zero.
- The User Data is to be deleted.
- Time Leap information shall be reset.
- Its clock shall be set to the specified default value.

⌋*(RS_TS_00009)*

[SWS_TS_00135]{DRAFT} ⌈A clock shall return the default value plus the elapsed local time, until it is configured for the first time.⌋*(RS_TS_00004)*

7.3 Shutdown behavior

This is to be defined in future releases of this specification.

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, the 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 taking into account 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 and updated whenever a new time value is received from its associated Master Time Base.

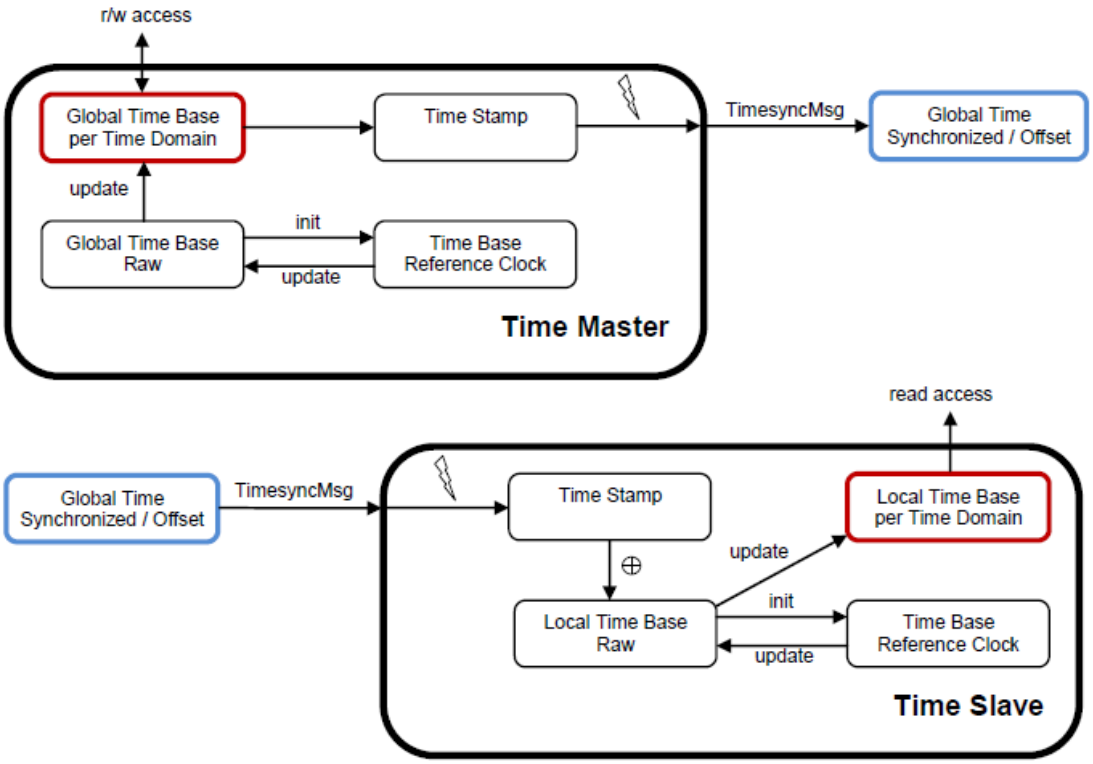

Figure 7.1: Global Time Base Distribution.

7.4.1.1 Time Base Manifestations

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

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 Slave Time Bases defined in an `Adaptive Application`, it is not possible to add a third one without recompiling. But these two Slave Time Bases can be configured to represent any Slave TBR in the system during runtime. During compile time the location of 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).

7.4.1.2 Configuration of Time Base Resources

The TBRs present in the system are specified in a pre-built configuration.

This pre-built configuration also contains the TBR type and in case of Offset Time Base types, it specifies 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.

[SWS_TS_00394]{DRAFT} ⌈TS shall create a *SynchMasterTBR* for each modeled *PortPrototype*, typed by a *TimeSynchronizationMasterInterface*, which has the attribute *timeBaseKind* set to *synchronized*.⌋*(RS_TS_00026)*

[SWS_TS_00395]{DRAFT} ⌈TS shall create an *OffsetMasterTBR* for each modeled *PortPrototype*, typed by a *TimeSynchronizationMasterInterface*, which has the attribute *timeBaseKind* set to *offset*.⌋*(RS_TS_00026)*

[SWS_TS_00396]{DRAFT} ⌈TS shall create a *SynchSlaveTBR* for each modeled *PortPrototype*, typed by a *TimeSynchronizationSlaveInterface*, which has the attribute *timeBaseKind* set to *synchronized*.⌋*(RS_TS_00026)*

[SWS_TS_00397]{DRAFT} ⌈TS shall create an *OffsetSlaveTBR* for each modeled *PortPrototype*, typed by a *TimeSynchronizationSlaveInterface*, which has the attribute *timeBaseKind* set to *offset*.⌋*(RS_TS_00026)*

[SWS_TS_00398]{DRAFT} ⌈TS shall create a *PureLocalTBR* for each modeled *PortPrototype*, typed by a *TimeSynchronizationPureLocalInterface*.⌋*(RS_TS_00026)*

[SWS_TS_00399]{DRAFT} ⌈TBRs shall be identified at run-time by the *shortName* path of the *PortPrototype*, passed as *InstanceSpecifier* to the *FindResource* method.⌋*(RS_TS_00026)*

7.4.2 Roles of the Time Base Resources

7.4.2.1 Global Time Master

A TBR can act as a Global Time Master, in which case it is the system wide origin for a given time value that is then distributed via the network to the Time Slaves.

7.4.2.2 Time Slave

In the role of a Time Slave, the TBR updates its internally-maintained local time to a value of a Global Time Base, which is 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]{DRAFT} ⌈Synchronized TBRs have to set their synchronization flag on reception of a Global Time Base value and adopt their local time value accordingly.⌋ *(RS_TS_00009)*

[SWS_TS_00009]{DRAFT} ⌈A Synchronized Time Base can be referenced by multiple Offset Time Bases.⌋*(RS_TS_00029, RS_TS_00030)*

7.4.3.1 Synchronized Master Time Base

[SWS_TS_00013]{DRAFT} ⌈A valid request to update or set the time value of a Synchronized Master TBR shall result in an updated local time of the corresponding Time Base.⌋*(RS_TS_00010)*

7.4.3.2 Synchronized Slave Time Base

[SWS_TS_00014]{DRAFT} ⌈Time values of a Synchronized TBR shall be returned to the requester by means of a data type compatible to std::chrono.⌋*(RS_TS_00005, RS_TS_00023)*

Master and slave clocks, although they are synchronized, will always have a time drift. In order to avoid them drifting apart over time, there is the need of periodic resynchronizations. The more alike the master and slave-reference clock are, the longer the time between re-syncs can be, before the time drift reaches a critical level. Some applications might be interested in this adjustment that is made to the mimicked clocks when they are re-synced with the foreign master clock.

[SWS_TS_00015]{DRAFT} ⌈Latest adjustments to the local clock value shall be returned as duration via a data type compatible to std::chrono.⌋*(RS_TS_00023)*

7.4.4 Offset Time Base Resources

A common requirement with regards to the existence of Offset Time Bases is that **[SWS_TS_00008]**{DRAFT} ⌈An Offset Time Base shall depend only on one Synchronized Time Base.⌋*(RS_TS_00029, RS_TS_00030)*

7.4.4.1 Offset Master Time Base

[SWS_TS_00016]{DRAFT} ⌈The offset duration value of an Offset Time Base shall be adjustable through an interface.⌋*(RS_TS_00013)*

[SWS_TS_00017]{DRAFT} ⌈It shall be possible to query for the offset time value of an Offset Time Base through an interface.⌋*(RS_TS_00012)*

The offset duration of an Offset Time Base is automatically calculated when the clock value is set using a time point. Therefore it is required that the underlying Synchronized TBR is already initialized.

[SWS_TS_00018]{DRAFT} ⌈An update of the clock value of an Offset Time Base shall only be possible, if the synchronization flag of the underlying TBR has been set. An exception shall be raised, if this is not the case.⌋*(RS_TS_00010)*

[SWS_TS_00019]{DRAFT} ⌈If all preconditions for an update of an Offset Time Base are met, the TBR shall calculate the offset duration 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 result of this calculation shall be used to maintain the internal clock of the TBR.⌋*(RS_TS_00007)*

[SWS_TS_00021]{DRAFT} ⌈If the Application needs to store additional data along with the time value, it shall be able to do so by setting this user defined data via an interface.⌋*(RS_TS_00015)*

[SWS_TS_00133]{DRAFT} ⌈For every Offset Time Base it shall be possible to access the underlying Synchronized TBR.⌋*(RS_TS_00026, RS_TS_00029)*

7.4.4.2 Offset Slave Time Base

[SWS_TS_00022]{DRAFT} ⌈The time value of an Offset Slave Time Base is calculated by adding the offset duration to the clock value of the underlying Synchronized TBR.⌋*(RS_TS_00005)*

[SWS_TS_00134]{DRAFT} ⌈For every Offset Time Base it shall be possible to access the underlying Synchronized TBR.⌋*(RS_TS_00026, RS_TS_00030)*

7.4.4.3 Pure Local Time Base

[SWS_TS_00023]{DRAFT} ⌈A Pure Local TBR shall maintain its Time Base autonomously.⌋*(RS_TS_00002)*

[SWS_TS_00324]{DRAFT} ⌈Until the clock was set for the first time, a Pure Local TBR will return the time since the creation of the resource.⌋*(RS_TS_00009)*

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

A Pure Local TB, as the name implies, is not synced to any foreign master clock and therefore most of the status flags that are meaningful for other TBRs can be disregarded.

[SWS_TS_00025]{DRAFT} ⌈For Pure Local TBRs all status flags other than the synchronization flag are ignored.⌋*(RS_TS_00009)*

7.4.5 Synchronization State

A clock is not only characterized by its time value, but also by its status flags and user data. Hence it is mandatory to evaluate them in the same context, in a snapshot.

[SWS_TS_00136]{DRAFT} ⌈For any type of TBR it shall be possible to obtain a snapshot of the Time Base Status, containing the current clock time, the status flags and the user data.⌋*(RS_TS_00021)*

[SWS_TS_00137]{DRAFT} ⌈For Offset TBRs the Time Base Status shall additionally contain a snapshot of the data of the underlying Synchronized TBR with the same creation time.⌋*(RS_TS_00021)*

[SWS_TS_00138]{DRAFT} ⌈Time Base Statuses for Synchronized TBRs have the same interface as the ones for Offset TBRs. Since the former is not linked to another TBR, its Time Base Status shall return a copy of itself instead of a snapshot of the underlying TBR.⌋*(RS_TS_00021)*

7.4.5.1 Slave Time Bases

Slave Time Bases locally reproduce foreign clocks and will therefore return the time since their creation prior to being synced for the first time. During the incomplete initialization phase it makes no sense to check for time leaps. Threshold monitoring shall be deactivated until the first successful synchronization.

[SWS_TS_00139]{DRAFT} ⌈Monitoring of time leaps to the future shall only be enabled, if a Time Leap Future Threshold other than zero was specified and the synchronization flag of the TBR is already set.⌋*(RS_TS_00009)*

[SWS_TS_00140]{DRAFT} ⌈Monitoring of time leaps to the past shall only be enabled, if a Time Leap Past Threshold other than zero was specified and the synchronization flag of the TBR is already set.⌋*(RS_TS_00009)*

[SWS_TS_00141]{DRAFT} ⌈A check for time leaps shall be performed on every successful synchronization with the master clock, but only after the clock has been synchronized once.⌋*(RS_TS_00009)*

[SWS_TS_00027]{DRAFT} ⌈If the adjustment made by the resynchronization exceeded the specified threshold values, the corresponding Time Leap Flags shall be set.⌋*(RS_TS_00009)*

[SWS_TS_00028]{DRAFT} ⌈Active Time Leap Status flags shall be revoked, if a defined consecutive number of synchronizations were all below the Time Leap Future and Past Thresholds.⌋*(RS_TS_00009)*

[SWS_TS_00030]{DRAFT} ⌈Each Slave TSP shall monitor for a synchronization time out by measuring the time since that last update and a specified timeout duration.⌋*(RS_TS_00009)*

[SWS_TS_00032]{DRAFT} ⌈The TBR shall set the Timeout Flag, if it detected a synchronization loss with the master.⌋*(RS_TS_00009)*

[SWS_TS_00011]{DRAFT} ⌈If a synchronization loss timeout takes place and the TBR in question is updated against a Time Gateway, the TBR shall indicate this by setting the Synchronized to Gateway status flag.⌋*(RS_TS_00009)*

[SWS_TS_00033]{DRAFT} ⌈The Timeout status flag shall be cleared on a successful update of the Time Base.⌋*(RS_TS_00009)*

[SWS_TS_00020]{DRAFT} ⌈The Synchronized to Gateway status flag shall be set on every successful synchronization of the Time Base, if such update is done against a Time Gateway and it should be cleared otherwise.⌋*(RS_TS_00009)*

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

Note: Once the Synchronization 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]{DRAFT} ⌈The updateCounter of a TBR shall have the value range 0 to 255.⌋*(RS_TS_00009, RS_TS_00021)*

[SWS_TS_00036]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈The TBR shall perform Rate Correction measurements to determine its rate deviation.⌋*(RS_TS_00002, RS_TS_00018)*

[SWS_TS_00042]{DRAFT} ⌈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)*

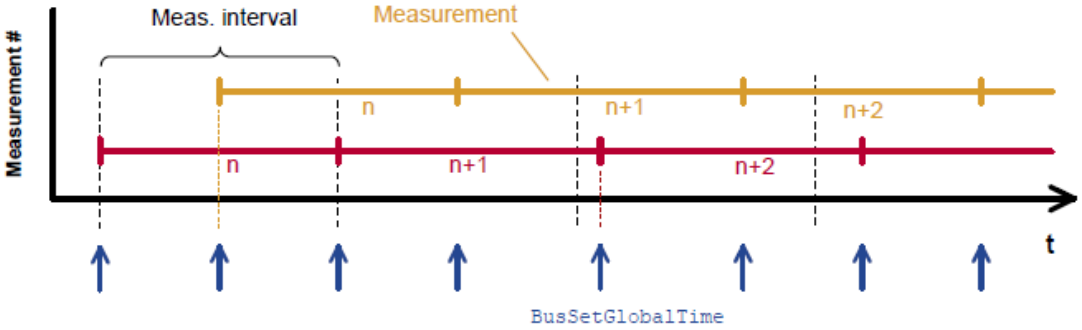

Figure 7.2: Visualization of two parallel measurements.

[SWS_TS_00043]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈The TBR shall perform as many simultaneous Rate Correction measurements as configured by the parameter 'RateCorrectionsPerMeasurementDuration'.⌋*(RS_TS_00018)*

[SWS_TS_00045]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈The last r_{rc} value has to be used until a new value is calculated.⌋*(RS_TS_00018, RS_TS_00019)*

[SWS_TS_00052]{DRAFT} ⌈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]{DRAFT} ⌈On invocation of getRateDeviation() the TBR shall return the calculated rate deviation (i.e. r_{rc}-1).⌋*(RS_TS_00018)*

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

[SWS_TS_00054]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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:
$TL = TL_{Sync} + (r_{rc} * (TV - TV_{Sync}) * r_{oc})$

- 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]{DRAFT} ⌈If the absolute time offset between the Global Time Base and the local instance of the Time Base (abs(TG - TL)) is smaller than OffsetCorrectionJumpThreshold, 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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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

TS offers the possibility to be notified after a certain timespan has elapsed. This so called Timer can be used by the applications and is offered by every TB. A timer can be created by passing a duration and a call-identifier to the corresponding API of any TB. The ID is used to tell, which of the possibly multiple timers, set by the Application, has expired.

This API will return a future, referring to the shared state that will contain the call-identifier once the promise is fulfilled after the given duration.

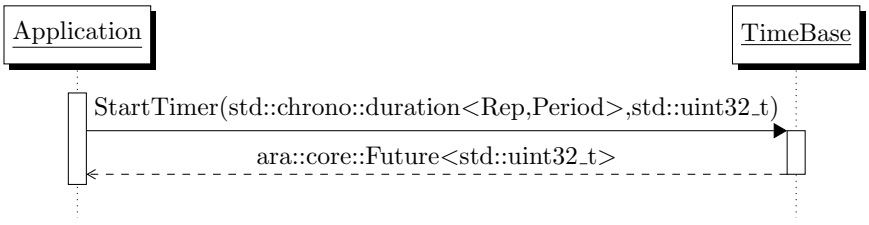

Figure 7.3: Mechanism of Time Notification.

[SWS_TS_00064]{DRAFT} ⌈Every TB shall offer the possibility to create a timer, which notifies the caller after a given timeframe.⌋*(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.

7.4.12 Time Validation

Figure 7.4 outlines the basic concept of the Time Validation feature.

A Time Slave collects information on the time synchronization process, to predict e.g. the Sync Ingress based on its local instance of Global Time and check whether Master and Slave agree upon the current time. The prediction itself will be locally analyzed by a separate Adaptive Application to detect any existing impairments. Furthermore, information on the time synchronization process from Time Masters and Slaves is also shared with a Validator Adaptive Application which may run anywhere in the network, e.g. on the owner of Global Time.

The Validator uses the information on the time synchronization process received from the Time Master and Time Slave Entities via a user defined feedback channel to reconstruct the whole synchronization process and check that a coherent time base is established among all peers.

The Time Validation feature only provides API to the Adaptive Application. The feedback channel and the actual validation performed by the respective Adaptive Application is not standardized in AUTOSAR. It is done in a user defined way on application level.

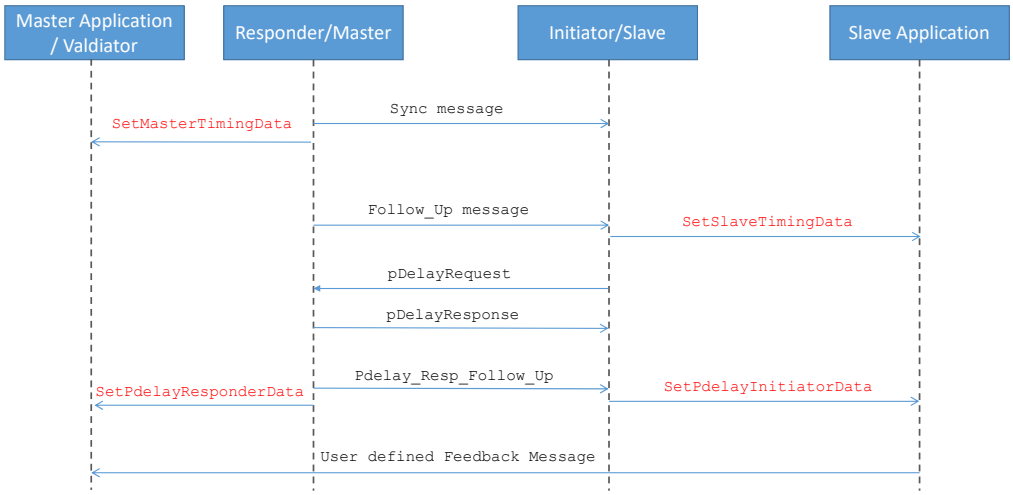

Figure 7.4: Time Validation mechanism

For an optional validation of the Timesync and Pdelay mechanisms, the Time Synchronization functional cluster provides the following functionality.

[SWS_TS_00424]{DRAFT} ⌈Everytime a `Follow_Up` message is received, all parameters defined by `TimeSlaveMeasurementType` shall be updated and the function `SetSlaveTimingData` shall be invoked.⌋*(RS_TS_00034)*

[SWS_TS_00425]{DRAFT} ⌈Everytime a Sync message is transmitted, all parameters defined by `TimeMasterMeasurementType` shall be updated and the function `SetMasterTimingData` shall be invoked.⌋*(RS_TS_00034)*

[SWS_TS_00426]{DRAFT} ⌈After the current Pdelay measurement is finished, i.e., upon reception of the `Pdelay_Resp_Follow_Up` message, all parameters defined by `PdelayInitiatorMeasurementType` shall be updated and the function `SetPdelayInitiatorData` shall be invoked.⌋*(RS_TS_00034)*

[SWS_TS_00427]{DRAFT} ⌈After the current `Pdelay` measurement is finished, i.e., upon transmission of the `Pdelay_Resp_Follow_Up`, all parameters defined by `PdelayResponderMeasurementType` shall be updated and the function `SetPdelayResponderData` shall be invoked.⌋*(RS_TS_00034)*

Note: Please note that there is a decoupling between reception and transmission of the respective PTP event messages and the forwarding of measurement data.

7.5 Error Handling

[SWS_TS_00065]{DRAFT} ⌈If no TBR could be found for a TB, an exception shall be fired. This can happen, because either no such TB was configured or due to internal communication errors.⌋*(RS_TS_00026)*

[SWS_TS_00072]{DRAFT} ⌈An exception shall be fired, if the TimeBaseType of the resource does not match the one of the TB. E.g. trying to use a SynchSlaveTBR to configure a PureLocalTB.⌋*(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 several 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

[SWS_TS_00066]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::TimeBaseType	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class TimeBaseType : std::uint8_t {...};`	
Values:	kSynchMasterTBType= 0	Synchronized Master Time Base.
	kSynchSlaveTBType= 1	Synchronized Slave Time Base.
	kOffsetMasterTBType= 2	Offset Master Time Base.
	kOffsetSlaveTBType= 3	Offset Slave Time Base.
	kPureLocalTBType= 4	Pure Local Time Base.
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	Enum that is used to distinguish between different types of time bases. 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. .	

⌋(*RS_TS_00026*)

8.1.2 StatusFlag

[SWS_TS_00067]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::StatusFlag	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class StatusFlag : std::uint8_t {...};`	
Values:	kTimeOut= 0x0	TB was not synchronized within a certain time frame.

▽

△

	kSynchronized= 0x1	The TB was synchronized at least once.
	kSynchToGateway= 0x2	The TB is in sync with the gateway.
	kTimeLeapFuture= 0x3	An adjustment greater than a certain threshold has been made.
	kTimeLeapPast= 0x4	An adjustment back in time greater than a certain threshold has been made.
	kHasDLS= 0x5	Daylight saving time is supported.
	kDLSActive= 0x6	Daylight saving time is enabled.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Enumeration that is used to express the state of a time base. Each enumeral represents a flag in the status of a TBR. .	

⌋(*RS_TS_00009*)

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 two enumerals defined. More identities can be defined by static_casting an integer to the identity (e.g. `static_cast<std::underlying_type<SynchSlaveIdentity>::type>(2)`)

[SWS_TS_00374]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::PureLocalIdentity	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class PureLocalIdentity : std::uint8_t {...};`	
Values:	k0= 0	–
	k1= 1	–
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	PureLocalIdentity is used to locally distinguish between multiple manifestations of PureLocal TBs. .	

⌋(*RS_TS_00005*)

[SWS_TS_00373]{DRAFT} ⌈

Kind:	enumeration
Symbol:	ara::tsync::SynchMasterIdentity
Scope:	namespace ara::tsync

▽

△

Underlying type:	std::uint8_t	
Syntax:	`enum class SynchMasterIdentity : std::uint8_t {...};`	
Values:	k0= 0	–
	k1= 1	–
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	SynchMasterIdentity is used to locally distinguish between multiple manifestations of Synch MasterTBs. .	

⌋(*RS_TS_00005*)

[SWS_TS_00375]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::SynchSlaveIdentity	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class SynchSlaveIdentity : std::uint8_t {...};`	
Values:	k0= 0	–
	k1= 1	–
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	SynchSlaveIdentity is used to locally distinguish between multiple manifestations of SynchSlave TBs. .	

⌋(*RS_TS_00005*)

[SWS_TS_00376]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::OffsetMasterIdentity	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class OffsetMasterIdentity : std::uint8_t {...};`	
Values:	k0= 0	–
	k1= 1	–
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	OffsetMasterIdentity is used to locally distinguish between multiple manifestations of Offset MasterTBs. .	

⌋(*RS_TS_00005*)

[SWS_TS_00377]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ara::tsync::OffsetSlaveIdentity	
Scope:	namespace ara::tsync	
Underlying type:	std::uint8_t	
Syntax:	`enum class OffsetSlaveIdentity : std::uint8_t {...};`	
Values:	k0= 0	–
	k1= 1	–
Header file:	#include "ara/tsync/time_base_type.h"	
Description:	OffsetSlaveIdentity is used to locally distinguish between multiple manifestations of OffsetSlave TBs. .	

⌋(*RS_TS_00005*)

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]{DRAFT} ⌈Member type aliases `duration`, `rep`, `period`, `time_point` shall be defined.⌋(*RS_TS_00022*, *RS_TS_00023*)

8.1.5 TimeMasterMeasurementType

[SWS_TS_00414]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::TimeMasterMeasurementType	
Scope:	namespace ara::tsync	
Syntax:	`template <typename TB>` `struct TimeMasterMeasurementType final {...};`	
Template param:	typename TB	Time Base resource
Header file:	#include "ara/tsync/measurement_types.h"	
Description:	Structure with detailed data for validation of the Time Master.	

⌋(*RS_TS_00034*)

[SWS_TS_14140]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeMasterMeasurementType::preciseOriginTimestamp
Scope:	struct ara::tsync::TimeMasterMeasurementType

▽

△

Type:	std::chrono::time_point< TB, std::chrono::nanoseconds >
Syntax:	`std::chrono::time_point<TB, std::chrono::nanoseconds> ara::tsync::TimeMasterMeasurementType< TB >::preciseOriginTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	egress timestamp of Sync frame in Global Time

⌋(*RS_TS_00034*)

[SWS_TS_14141]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeMasterMeasurementType::syncEgressTimestamp
Scope:	struct ara::tsync::TimeMasterMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::TimeMasterMeasurementType< TB >::syncEgressTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	egress timestamp of Sync frame

⌋(*RS_TS_00034*)

[SWS_TS_14142]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeMasterMeasurementType::sequenceId
Scope:	struct ara::tsync::TimeMasterMeasurementType
Type:	std::uint16_t
Syntax:	`std::uint16_t ara::tsync::TimeMasterMeasurementType< TB >::sequenceId;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	sequence Id of sent Ethernet frame

⌋(*RS_TS_00034*)

8.1.6 TimeSlaveMeasurementType

[SWS_TS_00415]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::tsync::TimeSlaveMeasurementType
Scope:	namespace ara::tsync
Syntax:	`template <typename TB>` `struct TimeSlaveMeasurementType final {...};`

▽

△

Template param:	typename TB	Time Base resource
Header file:	#include "ara/tsync/measurement_types.h"	
Description:	Structure with detailed data for validation of the Time Slave.	

⌋(*RS_TS_00034*)

[SWS_TS_14150]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::preciseOriginTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::chrono::time_point< TB, std::chrono::nanoseconds >
Syntax:	`std::chrono::time_point<TB, std::chrono::nanoseconds> ara::tsync::TimeSlaveMeasurementType< TB >::preciseOriginTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	preciseOriginTimestamp taken from the received Follow_Up frame

⌋(*RS_TS_00034*)

[SWS_TS_14151]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::referenceGlobalTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::chrono::time_point< TB, std::chrono::nanoseconds >
Syntax:	`std::chrono::time_point<TB, std::chrono::nanoseconds> ara::tsync::TimeSlaveMeasurementType< TB >::referenceGlobalTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	SyncLocal Time Tuple (Global Time part) .

⌋(*RS_TS_00034*)

[SWS_TS_14152]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::syncIngressTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::TimeSlaveMeasurementType< TB >::syncIngressTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	ingress timestamp of Sync frame converted to Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14153]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::correctionField
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::int64_t
Syntax:	`std::int64_t ara::tsync::TimeSlaveMeasurementType< TB >::correctionField;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	correctionField taken from the received Follow_Up frame

⌋(*RS_TS_00034*)

[SWS_TS_14154]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::referenceLocalTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::TimeSlaveMeasurementType< TB >::referenceLocalTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	SyncLocal Time Tuple (Virtual Local Time part) .

⌋(*RS_TS_00034*)

[SWS_TS_14155]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::pDelay
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::uint32_t
Syntax:	`std::uint32_t ara::tsync::TimeSlaveMeasurementType< TB >::pDelay;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	currently valid pDelay value

⌋(*RS_TS_00034*)

[SWS_TS_14156]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::TimeSlaveMeasurementType::sequenceId
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::uint16_t
Syntax:	`std::uint16_t ara::tsync::TimeSlaveMeasurementType< TB >::sequenceId;`
Header file:	#include "ara/tsync/measurement_types.h"

▽

△

Description:	sequence Id of received Sync frame

⌋(*RS_TS_00034*)

8.1.7 PdelayInitiatorMeasurementType

[SWS_TS_00416]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::tsync::PdelayInitiatorMeasurementType
Scope:	namespace ara::tsync
Syntax:	`struct PdelayInitiatorMeasurementType final {...};`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	Structure with detailed timing data for the pDelay Initiator .

⌋(*RS_TS_00034*)

[SWS_TS_14160]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::requestOriginTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::requestOriginTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	egress timestamp of Pdelay_Req in Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14161]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::responseReceiptTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::responseReceiptTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	ingress timestamp of Pdelay_Resp in Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14162]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::requestReceiptTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::requestReceiptTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	ingress timestamp of Pdelay_Req in Global Time taken from the received Pdelay_Resp

⌋(*RS_TS_00034*)

[SWS_TS_14163]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::responseOriginTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::responseOriginTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	egress timestamp of Pdelay_Resp in Global Time taken from the received Pdelay_Resp_Follow_Up

⌋(*RS_TS_00034*)

[SWS_TS_14164]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::referenceLocalTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::referenceLocalTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	value of the Virtual Local Time of the reference Global Time Tuple

⌋(*RS_TS_00034*)

[SWS_TS_14165]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::referenceGlobalTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayInitiatorMeasurementType::referenceGlobalTimestamp;`

▽

△

Header file:	#include "ara/tsync/measurement_types.h"
Description:	value of the local instance of the Global Time of the reference Global Time Tuple

⌋(*RS_TS_00034*)

[SWS_TS_14166]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::pDelay
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint32_t
Syntax:	`std::uint32_t ara::tsync::PdelayInitiatorMeasurementType::pDelay;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	currently valid pDelay value

⌋(*RS_TS_00034*)

[SWS_TS_14167]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayInitiatorMeasurementType::sequenceId
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint16_t
Syntax:	`std::uint16_t ara::tsync::PdelayInitiatorMeasurementType::sequenceId;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	sequence Id of sent Pdelay_Req frame

⌋(*RS_TS_00034*)

8.1.8 PdelayResponderMeasurementType

[SWS_TS_00417]{DRAFT} ⌈

Kind:	struct
Symbol:	ara::tsync::PdelayResponderMeasurementType
Scope:	namespace ara::tsync
Syntax:	`struct PdelayResponderMeasurementType final {...};`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	Structure with detailed timing data for the pDelay Responder .

⌋(*RS_TS_00034*)

[SWS_TS_14170]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayResponderMeasurementType::requestReceiptTimestamp
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayResponderMeasurementType::requestReceiptTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	ingress timestamp of Pdelay_Req converted to Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14171]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayResponderMeasurementType::responseOriginTimestamp
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayResponderMeasurementType::responseOriginTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	egress timestamp of Pdelay_Resp converted to Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14172]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayResponderMeasurementType::referenceLocalTimestamp
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayResponderMeasurementType::referenceLocalTimestamp;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	value of the Virtual Local Time of the reference Global Time Tuple used to convert requestReceiptTimestamp and responseOriginTimestamp into Global Time

⌋(*RS_TS_00034*)

[SWS_TS_14173]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayResponderMeasurementType::referenceGlobalTimestamp
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t ara::tsync::PdelayResponderMeasurementType::referenceGlobalTimestamp;`

▽

△

Header file:	#include "ara/tsync/measurement_types.h"
Description:	value of the local instance of the Global Time of the reference Global Time Tuple used to convert requestReceiptTimestamp and responseOriginTimestamp into Global Time

⌋(*RS_TS_00034*)

[SWS_TS_14174]{DRAFT} ⌈

Kind:	variable
Symbol:	ara::tsync::PdelayResponderMeasurementType::sequenceId
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Type:	std::uint16_t
Syntax:	`std::uint16_t ara::tsync::PdelayResponderMeasurementType::sequenceId;`
Header file:	#include "ara/tsync/measurement_types.h"
Description:	sequence Id of received Pdelay_Req frame

⌋(*RS_TS_00034*)

8.2 Callable definitions

The TS covers the complete set of interfaces of the TBRs that are described below.

8.2.1 Common Function Definition of Master Time Bases

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

8.2.1.1 GetRateDeviation

Every TB implements the possibility to obtain the rate deviation between the local and the foreign clock, except for the PureLocalTB.

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

8.2.1.2 SetTime

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

[SWS_TS_00100]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈Implementation of `SetTime()` in the `PureLocalTB` shall update the internal clock according to the value which is passed as parameter in this method.⌋*(RS_TS_00002, RS_TS_00026, RS_TS_00010)*

8.2.1.3 UpdateTime

[SWS_TS_00104]{DRAFT} ⌈This method shall be implemented in class `OffsetMasterTB`, `SynchMasterTB` and in class `PureLocalTB`.⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00029, RS_TS_00026)*

[SWS_TS_00105]{DRAFT} ⌈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_00029, RS_TS_00026)*

[SWS_TS_00106]{DRAFT} ⌈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_00029, RS_TS_00026)*

[SWS_TS_00107]{DRAFT} ⌈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_00029, RS_TS_00026)*

[SWS_TS_00110]{DRAFT} ⌈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_00026)*

8.2.2 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 design of the Time Base please refer to section 7.1.

8.2.2.1 PureLocalTB::PureLocalTB

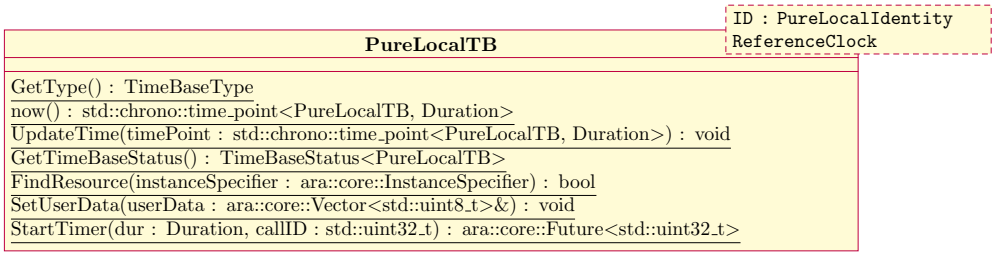

Figure 8.1: Class Diagram of the PureLocalTB.

[SWS_TS_00384]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::PureLocalTB	
Scope:	namespace ara::tsync	
Syntax:	`template <PureLocalIdentity ID, typename ReferenceClock = std::chrono::steady_clock> struct PureLocalTB {...};`	
Template param:	PureLocalIdentity ID	The PureLocalIdentity used to locally distinguish between different manifestations of the PureLocalTB
	typename ReferenceClock = std::chrono::steady_clock	The clock that is internally used to measure progress in time. Defaults to the std::chrono::steady_clock.
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	PureLocalTBs can be used to provide clock information through a local clock resource that is not synced to a foreign clock. In order to be able to use any type of PureLocalTB, they need to be configured by calling the FindResource() method with an instanceSpecifier that allows the mapping to the locally configured resource.	

⌋*(RS_TS_00002)*

[SWS_TS_00364]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::PureLocalTB::duration
Scope:	struct ara::tsync::PureLocalTB
Derived from:	std::chrono::nanoseconds

▽

△

Syntax:	`using ara::tsync::PureLocalTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;`
Header file:	#include "ara/tsync/pure_local_tb.h"
Description:	Member type alias to express the default unit of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00366]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::PureLocalTB::period
Scope:	struct ara::tsync::PureLocalTB
Derived from:	duration::period
Syntax:	`using ara::tsync::PureLocalTB< ID, ReferenceClock >::period = duration::period;`
Header file:	#include "ara/tsync/pure_local_tb.h"
Description:	Member type alias to express the default period of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00344]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::PureLocalTB::referenceClock
Scope:	struct ara::tsync::PureLocalTB
Derived from:	ReferenceClock
Syntax:	`using ara::tsync::PureLocalTB< ID, ReferenceClock >::referenceClock = ReferenceClock;`
Header file:	#include "ara/tsync/pure_local_tb.h"
Description:	Member type alias to store the ReferenceClock type .

⌋(*RS_TS_00009*, *RS_TS_00002*, *RS_TS_00026*)

[SWS_TS_00365]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::PureLocalTB::rep
Scope:	struct ara::tsync::PureLocalTB
Derived from:	duration::rep
Syntax:	`using ara::tsync::PureLocalTB< ID, ReferenceClock >::rep = duration::rep;`
Header file:	#include "ara/tsync/pure_local_tb.h"
Description:	Member type alias to express the default type of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00337]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::PureLocalTB::time_point
Scope:	struct ara::tsync::PureLocalTB
Derived from:	std::chrono::time_point<PureLocalTB, duration>
Syntax:	`using ara::tsync::PureLocalTB< ID, ReferenceClock >::time_point = std::chrono::time_point<PureLocalTB, duration>;`
Header file:	#include "ara/tsync/pure_local_tb.h"
Description:	Member type alias to express the time_point type used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00341]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::FindResource(ara::core::InstanceSpecifier instanceSpecifier)	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`static bool ara::tsync::PureLocalTB< ID, ReferenceClock >::Find Resource (ara::core::InstanceSpecifier instanceSpecifier);`	
Parameters (in):	instanceSpecifier	An ID that enables querying for a PureLocalTB configuration.
Return value:	bool	True if the resource, belonging to the instance specifier, could be successfully obtained, otherwise false.
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method that can be used to map a PureLocalTB to a TBR using the instance specifier.	

⌋(*RS_TS_00005*)

[SWS_TS_00340]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::GetTimeBaseStatus()	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`static TimeBaseStatus<PureLocalTB> ara::tsync::PureLocalTB< ID, ReferenceClock >::GetTimeBaseStatus ();`	
Return value:	TimeBaseStatus< PureLocalTB >	A clock specific TimeBaseStatus that contains all the relevant clock information.
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method to obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value.	

⌋(*RS_TS_00021*)

[SWS_TS_00338]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::GetType()	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`static constexpr TimeBaseType ara::tsync::PureLocalTB< ID, ReferenceClock >::GetType ();`	
Return value:	TimeBaseType	Always returns TimeBaseType::kPureLocalTBType for PureLocalTBs.
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method to obtain the TimeBaseType.	

⌋(*RS_TS_00002*, *RS_TS_00022*)

[SWS_TS_00128]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::now()	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`template <typename Duration = std::chrono::nanoseconds> static std::chrono::time_point<PureLocalTB, Duration> ara::tsync::PureLocalTB< ID, ReferenceClock >::now ();`	
Return value:	std::chrono::time_point< PureLocalTB, Duration >	The current time as clock specific time point.
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method to obtain the clocks current time.	

⌋(*RS_TS_00002*, *RS_TS_00002*, *RS_TS_00026*, *RS_TS_00005*)

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

[SWS_TS_00413]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::SetUserData(const ara::core::Vector< std::uint8_t > &userData)	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`static void ara::tsync::PureLocalTB< ID, ReferenceClock >::SetUserData (const ara::core::Vector< std::uint8_t > &userData);`	
Parameters (in):	userData	The user data to be set.
Return value:	None	
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method that can be used to set user data.	

⌋(*RS_TS_00002*, *RS_TS_00026*, *RS_TS_00015*)

[SWS_TS_00339]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::UpdateTime(std::chrono::time_point< PureLocalTB, Duration > time Point)	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`template <typename Duration = duration>` `static void ara::tsync::PureLocalTB< ID, ReferenceClock >::UpdateTime (std::chrono::time_point< PureLocalTB, Duration > timePoint);`	
Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	timePoint	The time information to be set.
Return value:	None	
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	A method that can be used to set a new time value for the clock.	

⌋(*RS_TS_00010*, *RS_TS_00011*, *RS_TS_00002*, *RS_TS_00026*, *RS_TS_00009*, *RS_TS_00002*)

[SWS_TS_00391]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::PureLocalTB::StartTimer(Duration dur, std::uint32_t callID)	
Scope:	struct ara::tsync::PureLocalTB	
Syntax:	`template <typename Duration = duration>` `static ara::core::Future<std::uint32_t> ara::tsync::PureLocalTB< ID, ReferenceClock >::StartTimer (Duration dur, std::uint32_t callID);`	
Parameters (in):	dur	The duration after which the result will be ready.
	callID	An ID that helps to identify, which timer returned.
Return value:	ara::core::Future< std::uint32_t >	A future referring to the shared state created by this call. : API signature
Header file:	#include "ara/tsync/pure_local_tb.h"	
Description:	Method that can be used to asynchronously wait for a timer event.	

⌋(*RS_TS_00016*, *RS_TS_00017*)

8.2.2.2 SynchMasterTB::SynchMasterTB

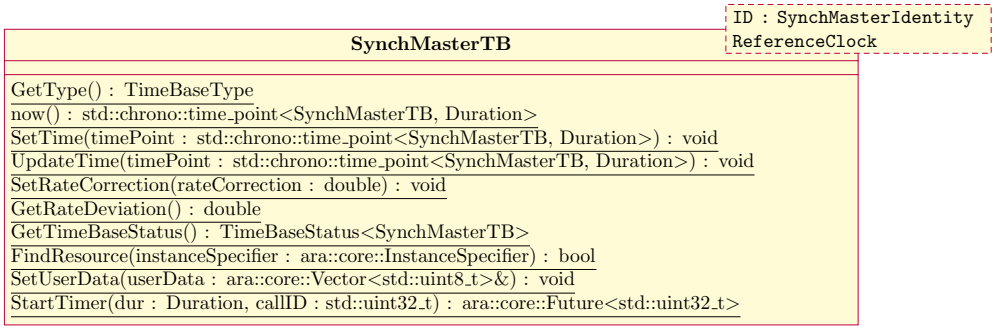

Figure 8.2: Class Diagram of the SynchMasterTB.

[SWS_TS_00385]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::SynchMasterTB	
Scope:	namespace ara::tsync	
Syntax:	`template <SynchMasterIdentity ID, typename ReferenceClock = std::chrono::steady_clock> struct SynchMasterTB {...};`	
Template param:	SynchMasterIdentity ID	The SynchMasterIdentity used to locally distinguish between different manifestations of the Synch MasterTB
	typename ReferenceClock = std::chrono::steady_clock	The clock that is internally used to measure progress of time. Defaults to the std::chrono::steady_clock.
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	SynchMasterTBs can be used to provide clock information via clock resources that are distributed to foreign clocks. In order to be able to use any type of SynchMasterTB, they need to be configured by calling the FindResource() method with an appropriate InstanceSpecifier.	

⌋(*RS_TS_00029*)

[SWS_TS_00370]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchMasterTB::duration
Scope:	struct ara::tsync::SynchMasterTB
Derived from:	std::chrono::nanoseconds
Syntax:	`using ara::tsync::SynchMasterTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;`
Header file:	#include "ara/tsync/synch_master_tb.h"
Description:	Member type alias to express the default unit of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00372]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchMasterTB::period
Scope:	struct ara::tsync::SynchMasterTB
Derived from:	duration::period
Syntax:	`using ara::tsync::SynchMasterTB< ID, ReferenceClock >::period = duration::period;`
Header file:	#include "ara/tsync/synch_master_tb.h"
Description:	Member type alias to express the default period of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00345]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchMasterTB::referenceClock
Scope:	struct ara::tsync::SynchMasterTB
Derived from:	ReferenceClock
Syntax:	`using ara::tsync::SynchMasterTB< ID, ReferenceClock >::referenceClock = ReferenceClock;`
Header file:	#include "ara/tsync/synch_master_tb.h"
Description:	Member type alias to store the ReferenceClock type .

⌋(*RS_TS_00022*)

[SWS_TS_00371]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchMasterTB::rep
Scope:	struct ara::tsync::SynchMasterTB
Derived from:	duration::rep
Syntax:	`using ara::tsync::SynchMasterTB< ID, ReferenceClock >::rep = duration::rep;`
Header file:	#include "ara/tsync/synch_master_tb.h"
Description:	Member type alias to express the default type of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00343]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchMasterTB::time_point
Scope:	struct ara::tsync::SynchMasterTB
Derived from:	std::chrono::time_point<SynchMasterTB, duration>
Syntax:	`using ara::tsync::SynchMasterTB< ID, ReferenceClock >::time_point = std::chrono::time_point<SynchMasterTB, duration>;`
Header file:	#include "ara/tsync/synch_master_tb.h"
Description:	Member type alias to express the time_point type used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00352]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::FindResource(ara::core::InstanceSpecifier instanceSpecifier)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static bool ara::tsync::SynchMasterTB< ID, ReferenceClock >::Find Resource (ara::core::InstanceSpecifier instanceSpecifier);`	
Parameters (in):	instanceSpecifier	An ID that allow querying for a SynchMasterTB configuration.

▽

△

Return value:	bool	True if the resource, belonging to the instance specifier, could be successfully obtained, otherwise false.
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method that can be used to configure a SynchMasterTB.	

⌋(*RS_TS_00005*)

[SWS_TS_00350]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::GetRateDeviation()	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static double ara::tsync::SynchMasterTB< ID, ReferenceClock >::GetRateDeviation ();`	
Return value:	double	The current rate deviation.
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋(*RS_TS_00018*)

[SWS_TS_00351]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::GetTimeBaseStatus()	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static TimeBaseStatus<SynchMasterTB> ara::tsync::SynchMasterTB< ID, ReferenceClock >::GetTimeBaseStatus ();`	
Return value:	TimeBaseStatus< SynchMasterTB >	A clock specific TimeBaseStatus that contains all the relevant clock information.
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method to obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value.	

⌋(*RS_TS_00021*)

[SWS_TS_00346]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::GetType()	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static constexpr TimeBaseType ara::tsync::SynchMasterTB< ID, ReferenceClock >::GetType ();`	
Return value:	TimeBaseType	Always returns TimeBaseType::kSynchMasterTBType for SynchMasterTBs.
Header file:	#include "ara/tsync/synch_master_tb.h"	

▽

△

Description:	Method to obtain the TimeBaseType.

⌋(*RS_TS_00029*, *RS_TS_00022*)

[SWS_TS_00153]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::now()	
Scope:	struct ara::tsync::SynchMasterTB	

Syntax:	`template <typename Duration = duration>`
	`static std::chrono::time_point<SynchMasterTB, Duration>`
	`ara::tsync::SynchMasterTB< ID, ReferenceClock >::now ();`	

Return value:	std::chrono::time_point< SynchMaster TB, Duration >	The current time as clock specific time point.
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method to obtain the clocks current time.	

⌋(*RS_TS_00026*, *RS_TS_00005*)

[SWS_TS_00154]{DRAFT} ⌈The time point offered shall be relative to the clock of the SynchMasterTB, from which this method is called.⌋(*RS_TS_00029*, *RS_TS_00002*)

[SWS_TS_00349]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::SetRateCorrection(double rateCorrection)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::SetRate Correction (double rateCorrection);`	
Parameters (in):	rateCorrection	The rate correction to be applied. 0.5 is two times slower, whilst 2.0 is 2 times faster.
Return value:	None	
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	This method can be used to set the rate correction that will be applied to time values.	

⌋(*RS_TS_00029*, *RS_TS_00026*, *RS_TS_00018*)

[SWS_TS_00347]{DRAFT} ⌈

Kind:	function
Symbol:	ara::tsync::SynchMasterTB::SetTime(std::chrono::time_point< SynchMasterTB, Duration > time Point)
Scope:	struct ara::tsync::SynchMasterTB
Syntax:	`template <typename Duration = duration>` `static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::SetTime (std::chrono::time_point< SynchMasterTB, Duration > timePoint);`

▽

△

Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	timePoint	The time information to be set.
Return value:	None	
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	A method that can be used to set a new time value for the clock. Setting a new time also triggers transmission on the bus.	

⌋(*RS_TS_00010*, *RS_TS_00026*)

[SWS_TS_00353]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::SetUserData(const ara::core::Vector< std::uint8_t > &userData)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::SetUserData (const ara::core::Vector< std::uint8_t > &userData);`	
Parameters (in):	userData	The user data to be set.
Return value:	None	
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method that can be used to set user data.	

⌋(*RS_TS_00029*, *RS_TS_00026*, *RS_TS_00015*)

[SWS_TS_00348]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::UpdateTime(std::chrono::time_point< SynchMasterTB, Duration > timePoint)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`template <typename Duration = duration>` `static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::UpdateTime (std::chrono::time_point< SynchMasterTB, Duration > timePoint);`	
Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	timePoint	The time information to be set.
Return value:	None	
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	A method that can be used to set a new time value for the clock. The clock value is only updated locally, transmission on the bus will happen in the next cycle.	

⌋(*RS_TS_00010*, *RS_TS_00011*, *RS_TS_00029*, *RS_TS_00026*, *RS_TS_00009*)

[SWS_TS_00392]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::StartTimer(Duration dur, std::uint32_t callID)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`template <typename Duration = duration> static ara::core::Future<std::uint32_t> ara::tsync::SynchMasterTB< ID, ReferenceClock >::StartTimer (Duration dur, std::uint32_t callID);`	
Parameters (in):	dur	The duration after which the result will be ready.
	callID	An ID that helps to identify, which timer returned.
Return value:	ara::core::Future< std::uint32_t >	A future referring to the shared state created by this call. : API signature
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method that can be used to asynchronously wait for a timer event.	

⌋(*RS_TS_00016*, *RS_TS_00017*)

[SWS_TS_00429]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchMasterTB::RegisterTimeBaseNotification(MasterTBProviderNotification< SynchMasterTB > &timeBaseProviderNotification)	
Scope:	struct ara::tsync::SynchMasterTB	
Syntax:	`static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::Register TimeBaseNotification (MasterTBProviderNotification< SynchMasterTB > &timeBaseProviderNotification);`	
Parameters (in):	timeBaseProviderNotification	Master (Validator) Application notification object that will be notified about the availability of a new data block recorded for the Time Base
Return value:	None	
Header file:	#include "ara/tsync/synch_master_tb.h"	
Description:	Method that can be used by Master applications to receive time sync parameters.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

8.2.2.3 SynchSlaveTB::SynchSlaveTB

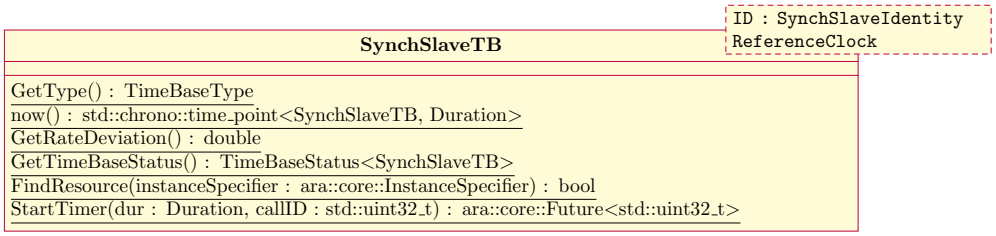

Figure 8.3: Class Diagram of the SynchSlaveTB.

[SWS_TS_00382]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::SynchSlaveTB	
Scope:	namespace ara::tsync	
Syntax:	`template <SynchSlaveIdentity ID, typename ReferenceClock = std::chrono::steady_clock> struct SynchSlaveTB {...};`	
Template param:	SynchSlaveIdentity ID	The SynchSlaveIdentity used to locally distinguish between different manifestations of the SynchSlave TB
	typename ReferenceClock = std::chrono::steady_clock	The clock that is internally used to measure progress of time. Defaults to the std::chrono::steady_clock.
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	SynchSlaveTBs can be used to abstract clock information from clock resources that are synced to a foreign clock. In order to be able to use any type of SynchSlaveTB, they need to be configured by calling the FindResource() method with an appropriate InstanceSpecifier.	

⌋(*RS_TS_00030*)

[SWS_TS_00367]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchSlaveTB::duration
Scope:	struct ara::tsync::SynchSlaveTB
Derived from:	std::chrono::nanoseconds
Syntax:	`using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;`
Header file:	#include "ara/tsync/synch_slave_tb.h"
Description:	Member type alias to express the default unit of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00369]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchSlaveTB::period
Scope:	struct ara::tsync::SynchSlaveTB
Derived from:	duration::period
Syntax:	`using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::period = duration::period;`
Header file:	#include "ara/tsync/synch_slave_tb.h"
Description:	Member type alias to express the default period of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00378]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchSlaveTB::referenceClock
Scope:	struct ara::tsync::SynchSlaveTB
Derived from:	ReferenceClock
Syntax:	`using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::referenceClock = ReferenceClock;`
Header file:	#include "ara/tsync/synch_slave_tb.h"
Description:	Member type alias to store the ReferenceClock type .

⌋(*RS_TS_00022*)

[SWS_TS_00368]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchSlaveTB::rep
Scope:	struct ara::tsync::SynchSlaveTB
Derived from:	duration::rep
Syntax:	`using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::rep = duration::rep;`
Header file:	#include "ara/tsync/synch_slave_tb.h"
Description:	Member type alias to express the default type of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00410]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::SynchSlaveTB::time_point
Scope:	struct ara::tsync::SynchSlaveTB
Derived from:	std::chrono::time_point<SynchSlaveTB, duration>
Syntax:	`using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::time_point = std::chrono::time_point<SynchSlaveTB, duration>;`
Header file:	#include "ara/tsync/synch_slave_tb.h"
Description:	Member type alias to express the time_point type used by this clock .

⌋(*RS_TS_00023*)

[SWS_TS_00381]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::FindResource(ara::core::InstanceSpecifier instanceSpecifier)	
Scope:	struct ara::tsync::SynchSlaveTB	
Syntax:	`static bool ara::tsync::SynchSlaveTB< ID, ReferenceClock >::Find Resource (ara::core::InstanceSpecifier instanceSpecifier);`	
Parameters (in):	instanceSpecifier	An ID that allow querying for a SynchSlaveTB configuration.

▽

△

Return value:	bool	True if the resource, belonging to the instance specifier, could be successfully obtained, otherwise false.
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method that can be used to configure a SynchSlaveTB.	

⌋(*RS_TS_00005*)

[SWS_TS_00379]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::GetRateDeviation()	
Scope:	struct ara::tsync::SynchSlaveTB	
Syntax:	`static double ara::tsync::SynchSlaveTB< ID, ReferenceClock >::GetRate Deviation ();`	
Return value:	double	The current rate deviation.
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋(*RS_TS_00018*)

[SWS_TS_00380]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::GetTimeBaseStatus()	
Scope:	struct ara::tsync::SynchSlaveTB	
Syntax:	`static TimeBaseStatus<SynchSlaveTB> ara::tsync::SynchSlaveTB< ID, ReferenceClock >::GetTimeBaseStatus ();`	
Return value:	TimeBaseStatus< SynchSlaveTB >	A clock specific TimeBaseStatus that contains all the relevant clock information.
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method to obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value.	

⌋(*RS_TS_00021*)

[SWS_TS_00411]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::GetType()	
Scope:	struct ara::tsync::SynchSlaveTB	
Syntax:	`static constexpr TimeBaseType ara::tsync::SynchSlaveTB< ID, Reference Clock >::GetType ();`	
Return value:	TimeBaseType	Always returns TimeBaseType::kSynchSlaveTBType for SynchSlaveTBs.
Header file:	#include "ara/tsync/synch_slave_tb.h"	

▽

△

Description:	Method to obtain the TimeBaseType.

⌋(*RS_TS_00030*, *RS_TS_00022*)

[SWS_TS_00031]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::now()	
Scope:	struct ara::tsync::SynchSlaveTB	

Syntax:	`template <typename Duration = duration>`
	`static std::chrono::time_point<SynchSlaveTB, Duration>`
	`ara::tsync::SynchSlaveTB< ID, ReferenceClock >::now ();`	

Return value:	std::chrono::time_point< SynchSlaveTB, Duration >	The current time as clock specific time point.
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method to obtain the clocks current time.	

⌋(*RS_TS_00026*, *RS_TS_00005*)

[SWS_TS_00393]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SynchSlaveTB::StartTimer(Duration dur, std::uint32_t callID)	
Scope:	struct ara::tsync::SynchSlaveTB	
Syntax:	`template <typename Duration = duration>` `static ara::core::Future<std::uint32_t> ara::tsync::SynchSlaveTB< ID, ReferenceClock >::StartTimer (Duration dur, std::uint32_t callID);`	
Parameters (in):	dur	The duration after which the result will be ready.
	callID	An ID that helps to identify, which timer returned.
Return value:	ara::core::Future< std::uint32_t >	A future referring to the shared state created by this call. : API signature
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method that can be used to asynchronously wait for a timer event.	

⌋(*RS_TS_00016*, *RS_TS_00017*)

[SWS_TS_00418]{DRAFT} ⌈

Kind:	function
Symbol:	ara::tsync::SynchSlaveTB::RegisterTimeBaseNotification(SlaveTBProviderNotification< SynchSlaveTB > &timeBaseProviderNotification)
Scope:	struct ara::tsync::SynchSlaveTB
Syntax:	`static void ara::tsync::SynchSlaveTB< ID, ReferenceClock >::RegisterTimeBaseNotification (SlaveTBProviderNotification< SynchSlaveTB > &timeBaseProviderNotification);`

▽

△

Parameters (in):	timeBaseProviderNotification	Slave Application notification object that will be notified about the availability of a new data block recorded for the Time Base
Return value:	None	
Header file:	#include "ara/tsync/synch_slave_tb.h"	
Description:	Method that can be used by Slave applications to receive time sync parameters.	

⌋*(RS_TS_00034, RS_TS_00030)*

[SWS_TS_00091]{DRAFT} ⌈The time point offered shall be relative to the epoch of the SynchSlaveTB, from which this method is called.⌋*(RS_TS_00030, RS_TS_00002)*

8.2.2.4 OffsetMasterTB::OffsetMasterTB

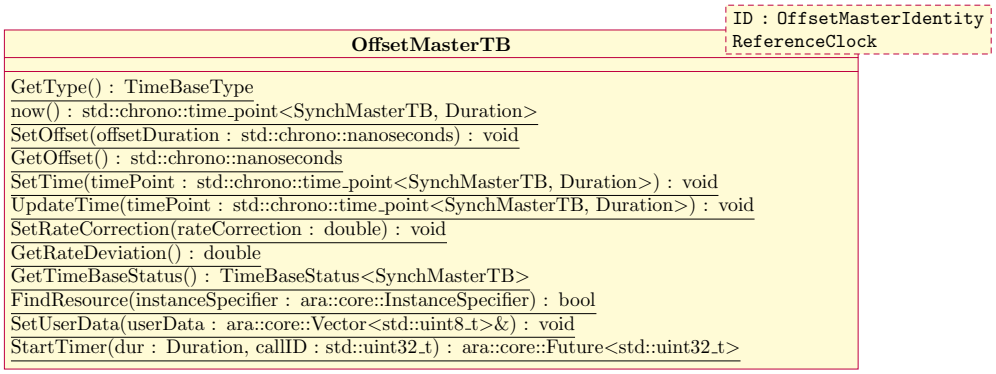

Figure 8.4: Class Diagram of the OffsetMasterTB.

[SWS_TS_00386]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::OffsetMasterTB	
Scope:	namespace ara::tsync	
Syntax:	`template <OffsetMasterIdentity ID, typename ReferenceClock>` `struct OffsetMasterTB {...};`	
Template param:	OffsetMasterIdentity ID	The OffsetMasterIdentity used to locally distinguish between different manifestations of the OffsetMaster TB
	typename ReferenceClock	The clock that is internally used to measure progress in time. The offset value will be applied to time_points provided by this clock.
Header file:	#include "ara/tsync/offset_master_tb.h"	

▽

△

Description:	OffsetMasterTBs can be used to provide clock information by adding an offset to an already existing clock resource and and distributing the result to foreign clocks. In order to be able to use any type of OffsetMasterTB, they need to be configured by calling the FindResource() method with an instanceSpecifier that allows the mapping to the locally configured resource.

⌋(*RS_TS_00029*)

[SWS_TS_00360]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetMasterTB::duration
Scope:	struct ara::tsync::OffsetMasterTB
Derived from:	std::chrono::nanoseconds
Syntax:	`using ara::tsync::OffsetMasterTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;`
Header file:	#include "ara/tsync/offset_master_tb.h"
Description:	Member type alias to express the default unit of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00358]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetMasterTB::period
Scope:	struct ara::tsync::OffsetMasterTB
Derived from:	duration::period
Syntax:	`using ara::tsync::OffsetMasterTB< ID, ReferenceClock >::period = duration::period;`
Header file:	#include "ara/tsync/offset_master_tb.h"
Description:	Member type alias to express the default period of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00115]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetMasterTB::referenceClock
Scope:	struct ara::tsync::OffsetMasterTB
Derived from:	ReferenceClock
Syntax:	`using ara::tsync::OffsetMasterTB< ID, ReferenceClock >::referenceClock = ReferenceClock;`
Header file:	#include "ara/tsync/offset_master_tb.h"
Description:	Member type alias to store the ReferenceClock type. .

⌋(*RS_TS_00009*, *RS_TS_00029*, *RS_TS_00026*)

[SWS_TS_00359]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetMasterTB::rep
Scope:	struct ara::tsync::OffsetMasterTB
Derived from:	duration::rep
Syntax:	`using ara::tsync::OffsetMasterTB< ID, ReferenceClock >::rep = duration::rep;`
Header file:	#include "ara/tsync/offset_master_tb.h"
Description:	Member type alias to express the default type of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00412]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetMasterTB::time_point
Scope:	struct ara::tsync::OffsetMasterTB
Derived from:	std::chrono::time_point<OffsetMasterTB, duration>
Syntax:	`using ara::tsync::OffsetMasterTB< ID, ReferenceClock >::time_point = std::chrono::time_point<OffsetMasterTB, duration>;`
Header file:	#include "ara/tsync/offset_master_tb.h"
Description:	Member type alias to express the time_point type used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00387]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::FindResource(ara::core::InstanceSpecifier instanceSpecifier)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static bool ara::tsync::OffsetMasterTB< ID, ReferenceClock >::Find Resource (ara::core::InstanceSpecifier instanceSpecifier);`	
Parameters (in):	instanceSpecifier	An ID that allow querying for an OffsetMasterTB configuration.
Return value:	bool	True if the resource, belonging to the instance specifier, could be successfully obtained, otherwise false. : API signature
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method that can be used to configure a OffsetMasterTB.	

⌋(*RS_TS_00016*, *RS_TS_00017*)

[SWS_TS_00114]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::GetOffset()	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static std::chrono::nanoseconds ara::tsync::OffsetMasterTB< ID, ReferenceClock >::GetOffset ();`	
Return value:	std::chrono::nanoseconds	The current offset value in nanoseconds.
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to obtain the current offset value of the clock.	

⌋(*RS_TS_00012*)

[SWS_TS_00112]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::SetOffset(std::chrono::nanoseconds offsetDuration)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static void ara::tsync::OffsetMasterTB< ID, ReferenceClock >::SetOffset (std::chrono::nanoseconds offsetDuration);`	
Parameters (in):	offsetDuration	The offset value to be set.
Return value:	None	
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to set a new offset value for the clock.	

⌋(*RS_TS_00013*, *RS_TS_00029*, *RS_TS_00026*)

[SWS_TS_00113]{DRAFT} ⌈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_00029*, *RS_TS_00026*, *RS_TS_00013*)

[SWS_TS_00084]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::GetRateDeviation()	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static double ara::tsync::OffsetMasterTB< ID, ReferenceClock >::GetRateDeviation ();`	
Return value:	double	The current rate deviation.
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋(*RS_TS_00018*)

[SWS_TS_00329]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::SetRateCorrection(double rateCorrection)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static void ara::tsync::OffsetMasterTB< ID, ReferenceClock >::SetRate Correction (double rateCorrection);`	
Parameters (in):	rateCorrection	The rate correction to be applied.
Return value:	None	
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	This method can be used to set the rate correction that will be applied to time values.	

⌋(*RS_TS_00029*, *RS_TS_00026*, *RS_TS_00018*)

[SWS_TS_00330]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::GetTimeBaseStatus()	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static TimeBaseStatus<OffsetMasterTB, referenceClock> ara::tsync::OffsetMasterTB< ID, ReferenceClock >::GetTimeBaseStatus ();`	
Return value:	TimeBaseStatus< OffsetMasterTB, referenceClock >	A clock specific TimeBaseStatus that contains all the relevant clock information.
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value.	

⌋(*RS_TS_00021*)

[SWS_TS_00327]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::SetTime(std::chrono::time_point< OffsetMasterTB, Duration > time Point)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`template <typename Duration = duration> static void ara::tsync::OffsetMasterTB< ID, ReferenceClock >::SetTime (std::chrono::time_point< OffsetMasterTB, Duration > timePoint);`	
Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	timePoint	The time information to be set.
Return value:	None	
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	A method that can be used to set a new time value for the clock. Setting a new time also triggers transmission on the bus.	

⌋(*RS_TS_00010*, *RS_TS_00026*, *RS_TS_00007*, *RS_TS_00009*)

[SWS_TS_00326]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::GetType()	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static constexpr TimeBaseType ara::tsync::OffsetMasterTB< ID, ReferenceClock >::GetType ();`	
Return value:	TimeBaseType	Always returns TimeBaseType::kOffsetMaster TBType for OffsetMasterTBs.
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to obtain the TimeBaseType.	

⌋(*RS_TS_00029*, *RS_TS_00022*)

[SWS_TS_00342]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::SetUserData(const ara::core::Vector< std::uint8_t > &userData)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`static void ara::tsync::OffsetMasterTB< ID, ReferenceClock >::SetUserData (const ara::core::Vector< std::uint8_t > &userData);`	
Parameters (in):	userData	The user data to be set.
Return value:	None	
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method that can be used to set user data.	

⌋(*RS_TS_00029*, *RS_TS_00026*, *RS_TS_00015*)

[SWS_TS_00151]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::now()	
Scope:	struct ara::tsync::OffsetMasterTB	

Syntax:	`template <typename Duration = duration>`
	`static std::chrono::time_point<OffsetMasterTB, Duration>`
	`ara::tsync::OffsetMasterTB< ID, ReferenceClock >::now ();`	

Return value:	std::chrono::time_point< OffsetMaster TB, Duration >	The current time as clock specific time point.
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method to obtain the current time of the clock.	

⌋(*RS_TS_00029*, *RS_TS_00002*)

[SWS_TS_00389]{DRAFT} ⌈

Kind:	function
Symbol:	ara::tsync::OffsetMasterTB::StartTimer(Duration dur, std::uint32_t callID)
Scope:	struct ara::tsync::OffsetMasterTB

▽

△

Syntax:	`template <typename Duration = duration> static ara::core::Future<std::uint32_t> ara::tsync::OffsetMasterTB< ID, ReferenceClock >::StartTimer (Duration dur, std::uint32_t callID);`	
Parameters (in):	dur	The duration after which the result will be ready.
	callID	An ID that helps to identify, which timer returned.
Return value:	ara::core::Future< std::uint32_t >	A future referring to the shared state created by this call. : API signature
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	Method that can be used to asynchronously wait for a timer event.	

⌋*(RS_TS_00016, RS_TS_00017)*

[SWS_TS_00152]{DRAFT} ⌈The time point offered shall be relative to the clock of the OffsetMasterTB, from which this method is called.⌋*(RS_TS_00029, RS_TS_00002)*

[SWS_TS_00328]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetMasterTB::UpdateTime(std::chrono::time_point< OffsetMasterTB, Duration > timePoint)	
Scope:	struct ara::tsync::OffsetMasterTB	
Syntax:	`template <typename Duration = duration> static void ara::tsync::OffsetMasterTB< ID, ReferenceClock >::Update Time (std::chrono::time_point< OffsetMasterTB, Duration > timePoint);`	
Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	timePoint	The time information to be set.
Return value:	None	
Header file:	#include "ara/tsync/offset_master_tb.h"	
Description:	A method that can be used to set a new time value for the clock. The clock value is only updated locally, transmission on the bus will happen in the next cycle.	

⌋*(RS_TS_00010, RS_TS_00011, RS_TS_00029, RS_TS_00026, RS_TS_00007, RS_TS_00009)*

8.2.2.5 OffsetSlaveTB::OffsetSlaveTB

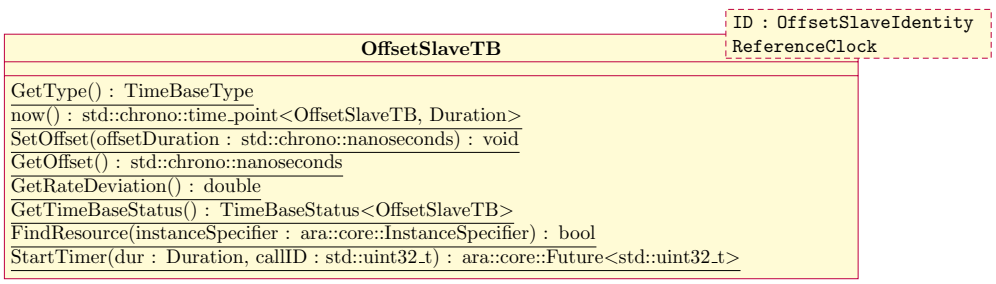

Figure 8.5: Class Diagram of the OffsetSlaveTB.

[SWS_TS_00383]{DRAFT} ⌈

Kind:	struct	
Symbol:	ara::tsync::OffsetSlaveTB	
Scope:	namespace ara::tsync	
Syntax:	`template <OffsetSlaveIdentity ID, typename ReferenceClock>` `struct OffsetSlaveTB {...};`	
Template param:	OffsetSlaveIdentity ID	The OffsetSlaveIdentity used to locally distinguish between different manifestations of the OffsetSlave TB
	typename ReferenceClock	The clock that is internally used to measure progress in time. The offset value will be applied to time_points provided by this clock.
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	OffsetSlaveTBs can be used to provide clock information by adding an offset to an already existing clock resource that is synced to a foreign clock. In order to be able to use any type of OffsetSlaveTB, they need to be configured by calling the FindResource() method with an instanceSpecifier that allows the mapping to the locally configured resource.	

⌋*(RS_TS_00030)*

[SWS_TS_00361]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetSlaveTB::duration
Scope:	struct ara::tsync::OffsetSlaveTB
Derived from:	std::chrono::nanoseconds
Syntax:	`using ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;`
Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Member type alias to express the default unit of durations used by this clock. .

⌋*(RS_TS_00023)*

[SWS_TS_00363]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetSlaveTB::period
Scope:	struct ara::tsync::OffsetSlaveTB
Derived from:	duration::period
Syntax:	`using ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::period = duration::period;`
Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Member type alias to express the default period of durations used by this clock. .

⌋*(RS_TS_00023)*

[SWS_TS_00362]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetSlaveTB::rep
Scope:	struct ara::tsync::OffsetSlaveTB
Derived from:	duration::rep
Syntax:	`using ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::rep = duration::rep;`
Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Member type alias to express the default type of durations used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_0094]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetSlaveTB::referenceClock
Scope:	struct ara::tsync::OffsetSlaveTB
Derived from:	ReferenceClock
Syntax:	`using ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::referenceClock = ReferenceClock;`
Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Member type alias to store the underlying time base. .

⌋(*RS_TS_00004*, *RS_TS_00030*, *RS_TS_00026*)

[SWS_TS_00331]{DRAFT} ⌈

Kind:	type alias
Symbol:	ara::tsync::OffsetSlaveTB::time_point
Scope:	struct ara::tsync::OffsetSlaveTB
Derived from:	std::chrono::time_point<OffsetSlaveTB, duration>
Syntax:	`using ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::time_point = std::chrono::time_point<OffsetSlaveTB, duration>;`
Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Member type alias to express the time_point type used by this clock. .

⌋(*RS_TS_00023*)

[SWS_TS_00388]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::FindResource(ara::core::InstanceSpecifier instanceSpecifier)	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static bool ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::Find Resource (ara::core::InstanceSpecifier instanceSpecifier);`	
Parameters (in):	instanceSpecifier	An ID that allow querying for a OffsetSlaveTB configuration.

▽

△

Return value:	bool	True if the resource, belonging to the instance specifier, could be successfully obtained, otherwise false. : API signature
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method that can be used to configure a OffsetSlaveTB.	

⌋(*RS_TS_00016*, *RS_TS_00017*)

[SWS_TS_00334]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::GetOffset()	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static std::chrono::nanoseconds ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::GetOffset ();`	
Return value:	std::chrono::nanoseconds	The current offset value.
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method to obtain the current offset value of the clock.	

⌋(*RS_TS_00012*)

[SWS_TS_00335]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::GetRateDeviation()	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static double ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::GetRate Deviation ();`	
Return value:	double	The current rate deviation.
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋(*RS_TS_00018*)

[SWS_TS_00336]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::GetTimeBaseStatus()	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static TimeBaseStatus<OffsetSlaveTB, referenceClock> ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::GetTimeBaseStatus ();`	
Return value:	TimeBaseStatus< OffsetSlaveTB, referenceClock >	A clock specific TimeBaseStatus that contains all the relevant clock information.
Header file:	#include "ara/tsync/offset_slave_tb.h"	

▽

△

Description:	Method to obtain a snapshot of the current state of the clock. This includes status flags, clock configuration and the actual time value.

⌋(*RS_TS_00021*)

[SWS_TS_00333]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::GetType()	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static constexpr TimeBaseType ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::GetType ();`	
Return value:	TimeBaseType	Always returns TimeBaseType::kOffsetSlaveTBType for OffsetSlaveTBs.
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method to obtain the TimeBaseType.	

⌋(*RS_TS_00030*, *RS_TS_00022*)

[SWS_TS_00090]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::now()	
Scope:	struct ara::tsync::OffsetSlaveTB	

Syntax:	`template <typename Duration = duration>`
	`static std::chrono::time_point<OffsetSlaveTB, Duration>`
	`ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::now ();`	

Return value:	std::chrono::time_point< OffsetSlaveTB, Duration >	The current time as clock specific time point.
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method to obtain the clocks current time.	

⌋(*RS_TS_00026*, *RS_TS_00005*)

[SWS_TS_00390]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::StartTimer(Duration dur, std::uint32_t callID)	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`template <typename Duration = duration>` `static ara::core::Future<std::uint32_t> ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::StartTimer (Duration dur, std::uint32_t callID);`	
Parameters (in):	dur	The duration after which the result will be ready.
	callID	An ID that helps to identify, which timer returned.
Return value:	ara::core::Future< std::uint32_t >	A future referring to the shared state created by this call. : API signature

▽

△

Header file:	#include "ara/tsync/offset_slave_tb.h"
Description:	Method that can be used to asynchronously wait for a timer event.

⌋(*RS_TS_00016*, *RS_TS_00017*)

[SWS_TS_00092]{DRAFT} ⌈The time point offered shall be relative to the epoch of the OffsetSlaveTB, from which this method is called.⌋(*RS_TS_00030*, *RS_TS_00002*)

[SWS_TS_00195]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::OffsetSlaveTB::SetOffset(std::chrono::nanoseconds offsetDuration)	
Scope:	struct ara::tsync::OffsetSlaveTB	
Syntax:	`static void ara::tsync::OffsetSlaveTB< ID, ReferenceClock >::SetOffset (std::chrono::nanoseconds offsetDuration);`	
Parameters (in):	offsetDuration	The offset value to be set.
Return value:	None	
Header file:	#include "ara/tsync/offset_slave_tb.h"	
Description:	Method to set a new offset value for the clock.	

⌋(*RS_TS_00022*)

8.2.2.6 Master TimeBase Provider Notification

[SWS_TS_00419]{DRAFT} ⌈

Kind:	class	
Symbol:	ara::tsync::MasterTBProviderNotification	
Scope:	namespace ara::tsync	
Syntax:	`template <typename TB>` `class MasterTBProviderNotification {...};`	
Template param:	typename TB	Time Base resource
Header file:	#include "ara/tsync/master_tb_provider_notification.h"	
Description:	Callback interface to notify Master (Validator) Application about the availability of a new data block recorded for the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

[SWS_TS_00423]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::MasterTBProviderNotification::SetPdelayResponderData(const PdelayResponder MeasurementType &measurementData)	
Scope:	class ara::tsync::MasterTBProviderNotification	
Syntax:	`virtual void ara::tsync::MasterTBProviderNotification< TB >::SetPdelay ResponderData (const PdelayResponderMeasurementType &measurement Data)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Responder
Return value:	None	
Header file:	#include "ara/tsync/master_tb_provider_notification.h"	
Description:	Provide the recorded data block for the pPelay Responder of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

[SWS_TS_00421]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::MasterTBProviderNotification::SetMasterTimingData(const TimeMaster MeasurementType< TB > &measurementData)	
Scope:	class ara::tsync::MasterTBProviderNotification	
Syntax:	`virtual void ara::tsync::MasterTBProviderNotification< TB >::SetMaster TimingData (const TimeMasterMeasurementType< TB > &measurementData)=0;`	
Parameters (in):	measurementData	Detailed data for validation of the Time Master
Return value:	None	
Header file:	#include "ara/tsync/master_tb_provider_notification.h"	
Description:	Provide the recorded data block for the Time Master of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

8.2.2.7 Slave TimeBase Provider Notification

[SWS_TS_00428]{DRAFT} ⌈

Kind:	class	
Symbol:	ara::tsync::SlaveTBProviderNotification	
Scope:	namespace ara::tsync	
Syntax:	`template <typename TB>` `class SlaveTBProviderNotification {...};`	
Template param:	typename TB	Time Base resource
Header file:	#include "ara/tsync/slave_tb_provider_notification.h"	
Description:	Callback interface to notify Slave Application about the availability of a new data block recorded for the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00030*)

[SWS_TS_00422]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SlaveTBProviderNotification::SetPdelayInitiatorData(const PdelayInitiator MeasurementType &measurementData)	
Scope:	class ara::tsync::SlaveTBProviderNotification	
Syntax:	`virtual void ara::tsync::SlaveTBProviderNotification< TB >::SetPdelay InitiatorData (const PdelayInitiatorMeasurementType &measurement Data)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Initiator
Return value:	None	
Header file:	#include "ara/tsync/slave_tb_provider_notification.h"	
Description:	Provide the recorded data block for the pPelay Initiator of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00030*)

[SWS_TS_00420]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::SlaveTBProviderNotification::SetSlaveTimingData(const TimeSlaveMeasurement Type< TB > &measurementData)	
Scope:	class ara::tsync::SlaveTBProviderNotification	
Syntax:	`virtual void ara::tsync::SlaveTBProviderNotification< TB >::SetSlave TimingData (const TimeSlaveMeasurementType< TB > &measurementData)=0;`	
Parameters (in):	measurementData	Detailed data for validation of the Time Slave
Return value:	None	
Header file:	#include "ara/tsync/slave_tb_provider_notification.h"	
Description:	Provide the recorded data block for the Time Slave of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00030*)

8.2.3 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.

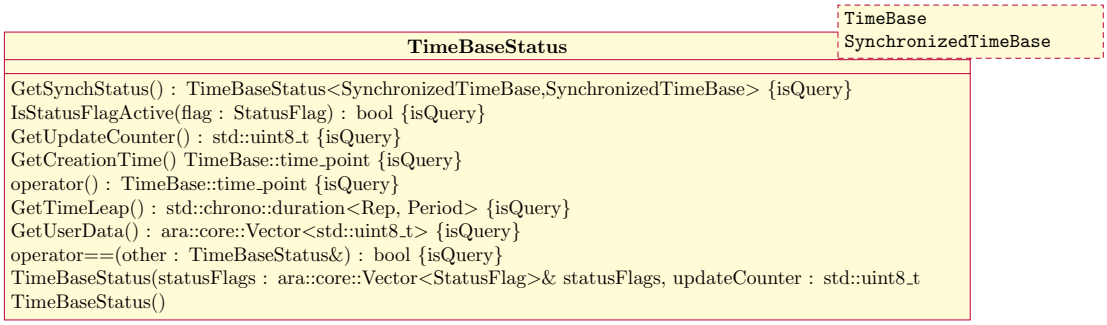

Figure 8.6: TimeBaseStatus and StatusFlags

[SWS_TS_00122]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::GetCreationTime()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`TB::time_point ara::tsync::TimeBaseStatus< TB, STB >::GetCreationTime () const;`	
Return value:	TB::time_point	The point in time at which this object was created. Time point is expressed in context of the clock that created this object.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	A method to obtain the creation time of this object.	

⌋*(RS_TS_00021)*

[SWS_TS_00123]{DRAFT} ⌈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)*

[SWS_TS_00126]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::GetSynchStatus()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`TimeBaseStatus<STB, STB> ara::tsync::TimeBaseStatus< TB, STB >::Get SynchStatus () const;`	
Return value:	TimeBaseStatus< STB, STB >	A copy of the inheriting TimeBaseStatus object.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	A method to obtain a copy of the time base status in case of a SynchronizedTB or , that was obtained during construction of this object.	

⌋*(RS_TS_00021)*

[SWS_TS_00127]{DRAFT} ⌈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]{DRAFT} ⌈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]{DRAFT} ⌈The creation time of the Offset TBR's `TimeBaseStatus` object and the creation time of the Synchronized TBR associated to the Offset TBR this `TimeBaseStatus` object corresponds to, shall be identical.⌋*(RS_TS_00021)*

[SWS_TS_00125]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::GetTimeLeap()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`template <typename Rep, typename Period> std::chrono::duration<Rep, Period> ara::tsync::TimeBaseStatus< TB, STB >::GetTimeLeap () const;`	
Return value:	std::chrono::duration< Rep, Period >	A duration object representing the time leap that was present during creation of this object.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	A method to obtain the duration of the current time leap, if the corresponding leap threshold flag is set.	

⌋(*RS_TS_00021*)

[SWS_TS_00121]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::GetUpdateCounter()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`std::uint8_t ara::tsync::TimeBaseStatus< TB, STB >::GetUpdateCounter () const;`	
Return value:	std::uint8_t	The update counter of the time base that created this object.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	A method to obtain the update counter value of the time base at creation time of this object.	

⌋(*RS_TS_00021*)

[SWS_TS_00119]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::GetUserData()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`ara::core::Vector<std::uint8_t> ara::tsync::TimeBaseStatus< TB, STB >::GetUserData () const;`	
Return value:	ara::core::Vector< std::uint8_t >	A vector of bytes holding the user data that was set during the creation of the status.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	A method to return the user defined data of the clock.	

⌋(*RS_TS_00021*, *RS_TS_00014*)

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

[SWS_TS_00118]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::IsStatusFlagActive(StatusFlag flag)	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`bool ara::tsync::TimeBaseStatus< TB, STB >::IsStatusFlagActive (StatusFlag flag) const;`	
Parameters (in):	flag	The StatusFlag that shall be checked.
Return value:	bool	True if the flag was active, otherwise false.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Method that can be used to check, if a certain flag was active during the time when the status was obtained.	

⌋(*RS_TS_00021*)

[SWS_TS_00354]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::operator==(const TimeBaseStatus &other)	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`bool ara::tsync::TimeBaseStatus< TB, STB >::operator== (const TimeBaseStatus &other) const;`	
Parameters (in):	other	TimeBaseStatus that shall be compared
Return value:	bool	True if the objects are equal, otherwise false.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Comparison operator to check, if two TimeBaseStatuses are equal. Two TimeBaseStatuses can be considered equal, if their addresses are the same, or if all the members are set to the same values.	

⌋(*RS_TS_00021*)

[SWS_TS_00130]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::operator typename TB::time_point()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`ara::tsync::TimeBaseStatus< TB, STB >::operator typename TB::time_point () const;`	
Return value:	TB::time_point	The point in time at which this object was created. Time point is expressed in context of the clock that created this object.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Cast operator to enable implicit casting of TimeBaseStatuses to their underlying time_points in order to improve usability.	

⌋(*RS_TS_00021*)

[SWS_TS_00355]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::TimeBaseStatus(const ara::core::Vector< StatusFlag > &status Flags, std::uint8_t updateCounter)	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`ara::tsync::TimeBaseStatus< TB, STB >::TimeBaseStatus (const ara::core::Vector< StatusFlag > &statusFlags, std::uint8_t update Counter);`	
Parameters (in):	statusFlags	- The status flags that shall be set for this status.
	updateCounter	- The update counter that shall be set for this status.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Constructor that is available to SynchMasterTBs, SynchSlaveTBs and PureLocalTBs.	

⌋(*RS_TS_00021*)

[SWS_TS_00356]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::TimeBaseStatus()	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`ara::tsync::TimeBaseStatus< TB, STB >::TimeBaseStatus ();`	
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Constructor that is available to OffsetSlaveTBs. All member variables are filled by the underlying time base. .	

⌋(*RS_TS_00021*)

[SWS_TS_00357]{DRAFT} ⌈

Kind:	function	
Symbol:	ara::tsync::TimeBaseStatus::TimeBaseStatus(std::uint8_t updateCounter)	
Scope:	struct ara::tsync::TimeBaseStatus	
Syntax:	`explicit ara::tsync::TimeBaseStatus< TB, STB >::TimeBaseStatus (std::uint8_t updateCounter);`	
Parameters (in):	updateCounter	- The update counter value that shall be set for this status.
Header file:	#include "ara/tsync/time_base_status.h"	
Description:	Constructor that is available to OffsetMasterTBs. All member variables are filled by the underlying time base, except the update counter, which is handled separately by the Offset MasterTB.	

⌋(*RS_TS_00021*)

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 to then interact with it (i.e. starting a timer).

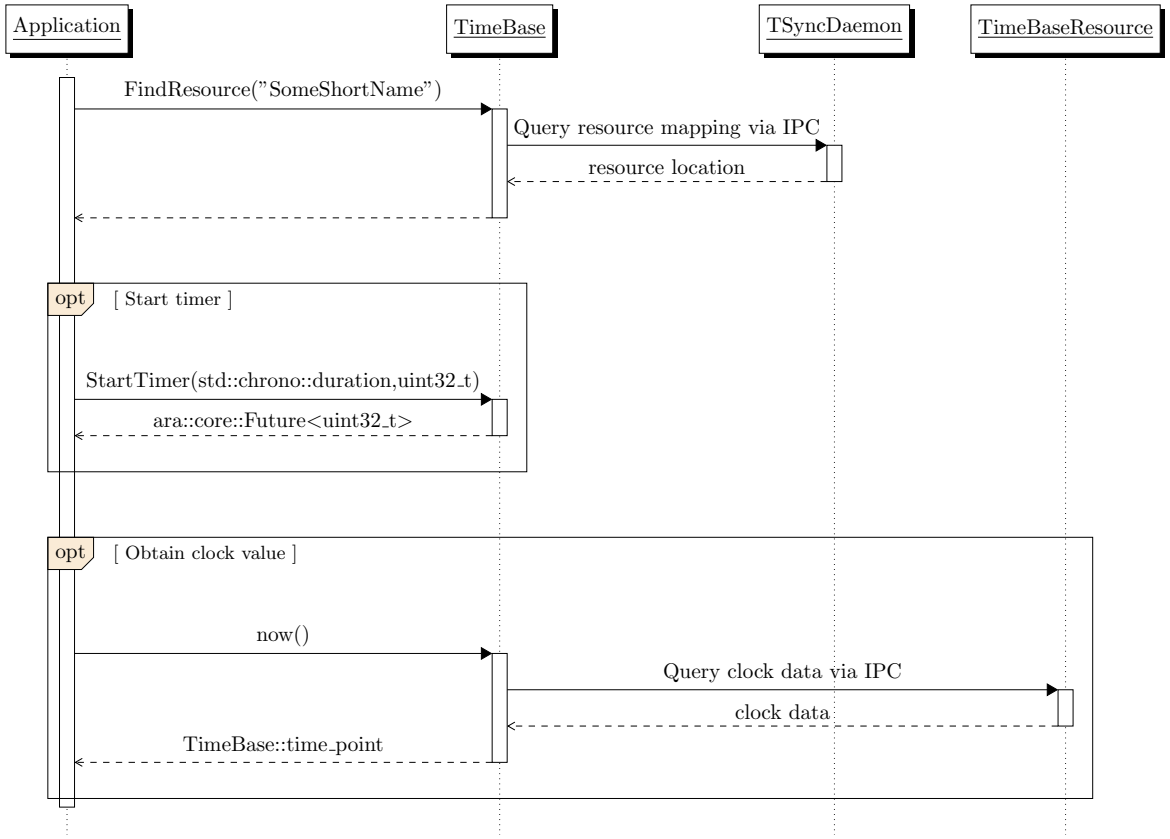

Figure 9.1: Application finds a TBR

9.2 Application starts a Timer

The following diagrams show how the application can make use of 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 [10].

9.2.1 Querying for the Future

Diagram 9.2 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. If the wait was successful, calling *valid()* on the future will return true and *get()* will return the CallID passed to the *StartTimer()* method.

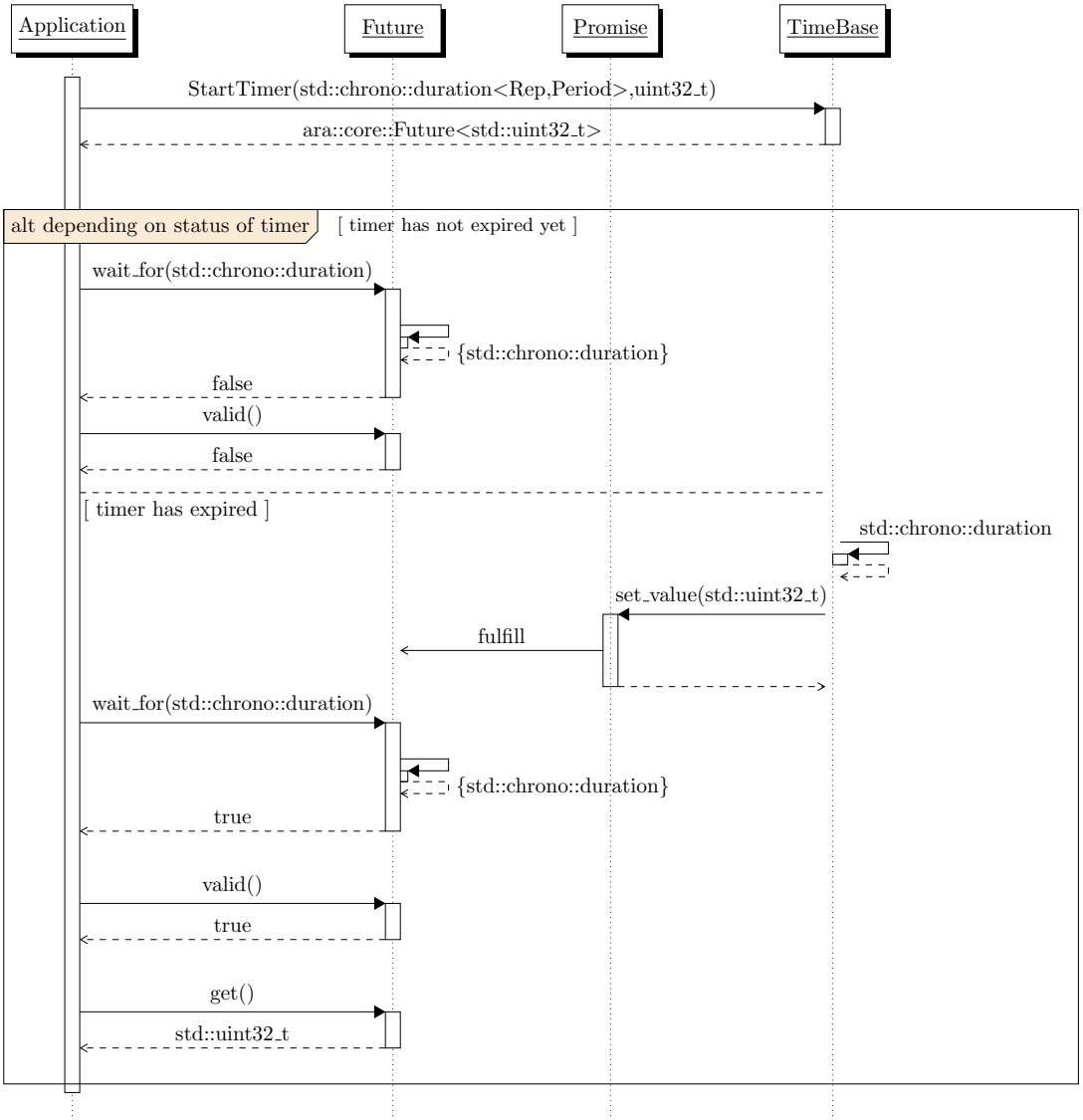

Figure 9.2: StartTimer - waiting for the future to be valid

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.

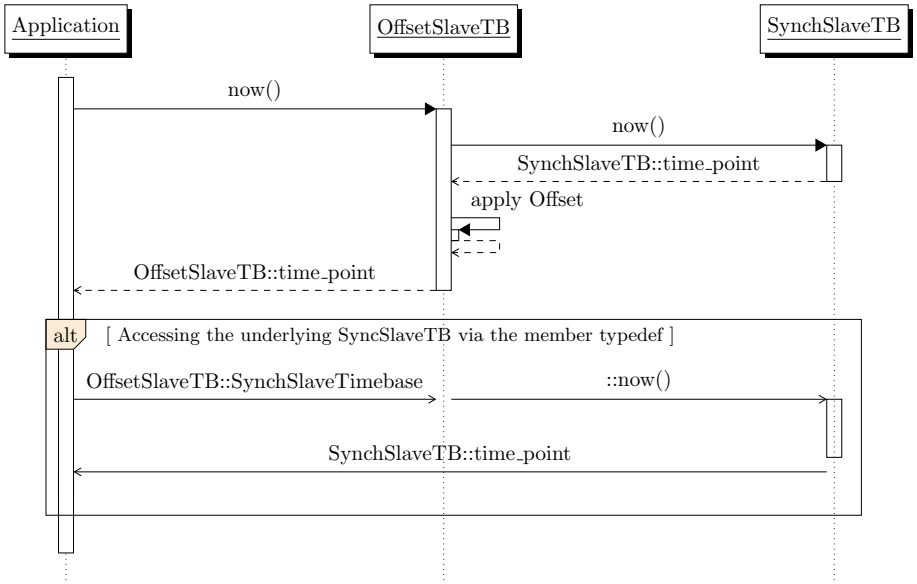

Figure 9.3: Offset Time Base Handling.

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.

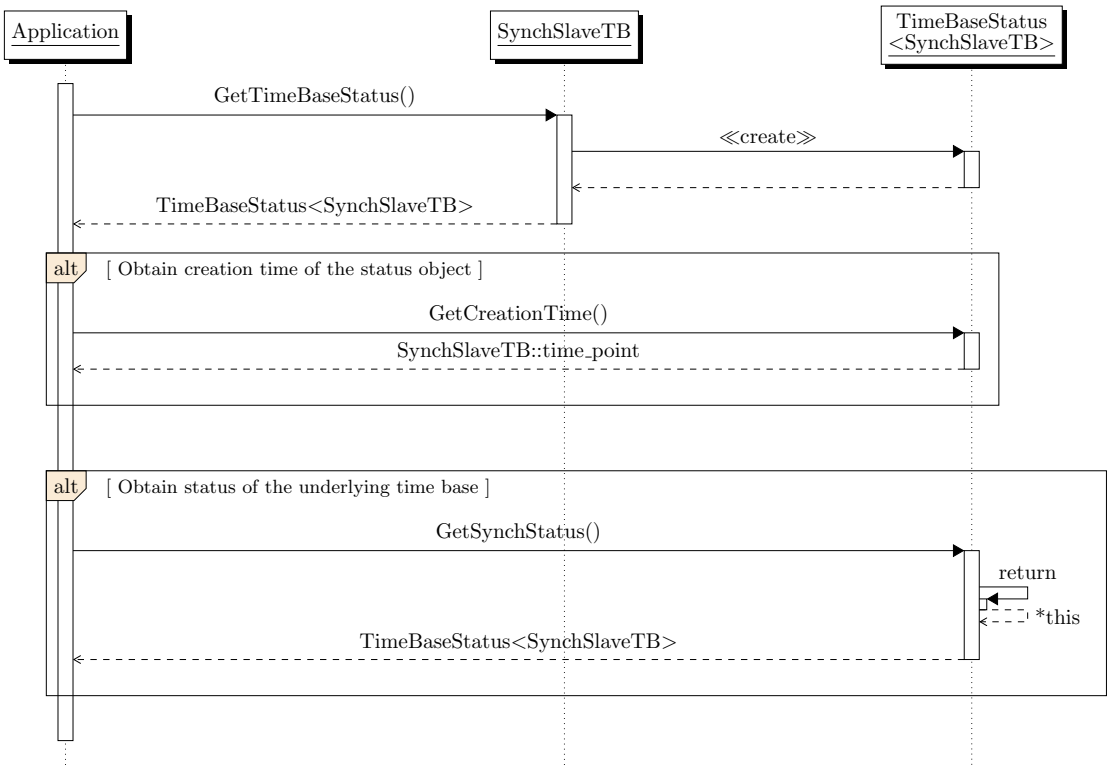

Figure 9.4: Request time base status of SynchTB.

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.

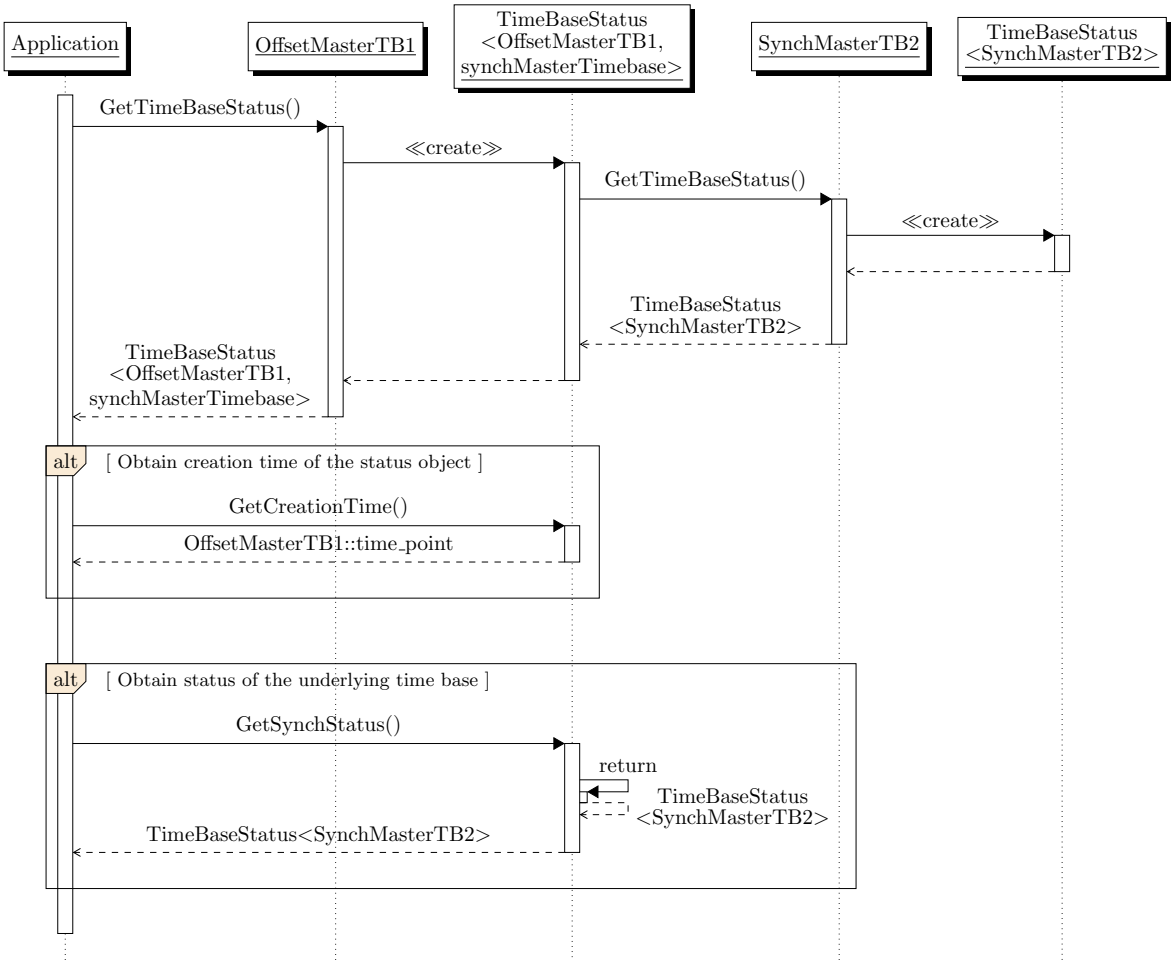

Figure 9.5: Request time base status of OffsetTB

A Specification Item History of this document compared to AUTOSAR R19-03.

A.1 Added Traceables in R19-11

Number	Heading
[SWS_TS_00031]	
[SWS_TS_00084]	
[SWS_TS_00090]	
[SWS_TS_00112]	

▽

△

Number	Heading
[SWS_TS_00114]	
[SWS_TS_00118]	
[SWS_TS_00119]	
[SWS_TS_00121]	
[SWS_TS_00122]	
[SWS_TS_00125]	
[SWS_TS_00126]	
[SWS_TS_00128]	
[SWS_TS_00130]	
[SWS_TS_00151]	
[SWS_TS_00153]	
[SWS_TS_00195]	
[SWS_TS_00326]	
[SWS_TS_00327]	
[SWS_TS_00328]	
[SWS_TS_00329]	
[SWS_TS_00330]	
[SWS_TS_00333]	
[SWS_TS_00334]	
[SWS_TS_00335]	
[SWS_TS_00336]	
[SWS_TS_00338]	
[SWS_TS_00339]	
[SWS_TS_00340]	
[SWS_TS_00341]	
[SWS_TS_00342]	
[SWS_TS_00346]	
[SWS_TS_00347]	
[SWS_TS_00348]	
[SWS_TS_00349]	
[SWS_TS_00350]	
[SWS_TS_00351]	
[SWS_TS_00352]	
[SWS_TS_00353]	
[SWS_TS_00354]	
[SWS_TS_00355]	
[SWS_TS_00356]	

▽

△

Number	Heading
[SWS_TS_00357]	
[SWS_TS_00379]	
[SWS_TS_00380]	
[SWS_TS_00381]	
[SWS_TS_00387]	
[SWS_TS_00388]	
[SWS_TS_00389]	
[SWS_TS_00390]	
[SWS_TS_00391]	
[SWS_TS_00392]	
[SWS_TS_00393]	
[SWS_TS_00411]	
[SWS_TS_00413]	
[SWS_TS_00414]	
[SWS_TS_00415]	
[SWS_TS_00416]	
[SWS_TS_00417]	
[SWS_TS_00418]	
[SWS_TS_00419]	
[SWS_TS_00420]	
[SWS_TS_00421]	
[SWS_TS_00422]	
[SWS_TS_00423]	
[SWS_TS_00424]	
[SWS_TS_00425]	
[SWS_TS_00426]	
[SWS_TS_00427]	
[SWS_TS_00428]	
[SWS_TS_00429]	
[SWS_TS_14140]	
[SWS_TS_14141]	
[SWS_TS_14142]	
[SWS_TS_14150]	
[SWS_TS_14151]	
[SWS_TS_14152]	
[SWS_TS_14153]	
[SWS_TS_14154]	

▽

△

Number	Heading
[SWS_TS_14155]	
[SWS_TS_14156]	
[SWS_TS_14160]	
[SWS_TS_14161]	
[SWS_TS_14162]	
[SWS_TS_14163]	
[SWS_TS_14164]	
[SWS_TS_14165]	
[SWS_TS_14166]	
[SWS_TS_14167]	
[SWS_TS_14170]	
[SWS_TS_14171]	
[SWS_TS_14172]	
[SWS_TS_14173]	
[SWS_TS_14174]	

Table A.1: Added Traceables in R19-11

A.2 Changed Traceables in R19-11

Number	Heading
[SWS_TS_00008]	
[SWS_TS_00009]	
[SWS_TS_00091]	
[SWS_TS_00092]	
[SWS_TS_00104]	
[SWS_TS_00105]	
[SWS_TS_00106]	
[SWS_TS_00107]	
[SWS_TS_00110]	
[SWS_TS_00113]	
[SWS_TS_00132]	
[SWS_TS_00133]	
[SWS_TS_00134]	
[SWS_TS_00150]	
[SWS_TS_00152]	

▽

△

Number	Heading
[SWS_TS_00154]	

Table A.2: Changed Traceables in R19-11

A.3 Deleted Traceables in R19-11

none