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	R20-11

Document Change History

Date	Release	Changed by	Description
2020-11-30	R20-11	AUTOSAR Release Management	• TSYNC API redesign and requirments updates • Harmomized with CP and RS Documents • Document adapted to new template • Terminology clarification and cleanup
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

Table of Contents

1 Introduction and functional overview 7

2 Acronyms and Abbreviations 8

2.1 Acronyms and Abbreviations 8
2.2 Definitions 9
2.2.1 ara::core::SteadyClock 9
2.2.2 Time Base Application 9

3 Related documentation 10

3.1 Input documents & related standards and norms 10
3.2 Further applicable specification 10

4 Constraints and assumptions 11

4.1 Known limitations 11
4.1.1 Configuration 11
4.1.2 Time Gateway 11
4.1.3 Out of Scope 11
4.2 Applicability to car domains 11
4.3 Recommendation 11

5 Dependencies to other Functional Clusters 12

6 Requirements Tracing 13

7 Functional specification 16

7.1 General Overview of TS 16
7.1.1 Base functionality of every Time Base 17
7.1.1.1 Time Base Status 17
7.1.1.2 Rate Deviation 17
7.1.1.3 Clock Time Value 17
7.1.2 Status Flags of TBRs 18
7.1.3 Time Synchronization and Protocols 18
7.2 Functional cluster life cycle 18
7.2.1 Startup 18
7.2.1.1 Default values 19
7.2.2 Shutdown 19
7.3 Normal Operation 19
7.3.1 Introduction 19
7.3.1.1 Time Base Manifestations 20
7.3.2 Roles of the Time Base Resources 20
7.3.2.1 Global Time Master 20
7.3.2.2 Time Slave 21
7.3.3 Time Base Resources 21
7.3.3.1 Slave Time Bases 21
7.3.4 Immediate Time Synchronization 22

7.3.5 User Data . . . 22
7.3.6 Time Correction . . . 22
7.3.6.1 Rate Correction for Time Slaves . . . 23
7.3.6.2 Offset Correction for Time Consumer . . . 24
7.3.6.3 Rate Correction for Global Time Masters . . . 26
7.3.7 Notifications of Time Base Consumer . . . 26
7.3.7.1 Status flags notification . . . 27
7.3.7.2 Synchronization status notification . . . 27
7.3.7.3 LeapJump notification . . . 27
7.3.8 Global Time Precision Measurement Support . . . 27
7.3.9 Global Time Validation Measurement Support . . . 28

8 API specification 31

8.1 API Common Data Types . . . 31
8.1.1 Timestamp . . . 31
8.1.2 LeapJump . . . 31
8.1.3 SynchronizationStatus . . . 31
8.2 Common Function Definition of Time Bases Provider . . . 32
8.2.1 SynchronizedTimeBaseProvider . . . 32
8.2.1.1 Special member functions . . . 32
8.2.1.2 SetTime . . . 34
8.2.1.3 UpdateTime . . . 35
8.2.1.4 GetCurrentTime . . . 35
8.2.1.5 SetRateCorrection . . . 36
8.2.1.6 GetRateCorrection . . . 36
8.2.1.7 SetUserData . . . 37
8.2.1.8 GetUserData . . . 37
8.2.1.9 RegisterTimeValidationNotification . . . 37
8.2.1.10 UnregisterTimeValidationNotification . . . 38
8.2.2 OffsetTimeBaseProvider . . . 38
8.2.2.1 Special member functions . . . 39
8.2.2.2 SetOffsetTime . . . 40
8.2.2.3 GetCurrentTime . . . 41
8.2.2.4 SetRateCorrection . . . 41
8.2.2.5 GetRateCorrection . . . 42
8.2.2.6 SetUserData . . . 42
8.2.2.7 GetUserData . . . 43
8.2.2.8 RegisterTimeValidationNotification . . . 43
8.2.2.9 UnregisterTimeValidationNotification . . . 43
8.3 Common Function Definition of Time Bases Consumer . . . 44
8.3.1 SynchronizedTimeBaseConsumer . . . 44
8.3.1.1 Special member functions . . . 44
8.3.1.2 GetCurrentTime . . . 46
8.3.1.3 GetRateDeviation . . . 47
8.3.1.4 GetTimeWithStatus . . . 47
8.3.1.5 RegisterStatusChangeNotifier . . . 47

8.3.1.6 UnregisterStatusChangeNotifier 48
8.3.1.7 RegisterSynchronizationStateChangeNotifier 48
8.3.1.8 UnregisterSynchronizationStateChangeNotifier . . . 49
8.3.1.9 RegisterTimeLeapNotifier 49
8.3.1.10 UnregisterTimeLeapNotifier 50
8.3.1.11 RegisterTimeValidationNotification 50
8.3.1.12 UnregisterTimeValidationNotification 51
8.3.1.13 RegisterTimePrecisionMeasurementNotifier 51
8.3.1.14 UnregisterTimePrecisionMeasurementNotifier 51
8.3.2 SynchronizedTimeBaseStatus 52
8.3.2.1 Special member functions 52
8.3.2.2 GetCreationTime 54
8.3.2.3 GetSynchronizationStatus 54
8.3.2.4 GetLeapJump 54
8.3.2.5 GetUserData 55
8.4 C++ Time Validation Interface 55
8.4.1 Type definitions 55
8.4.1.1 TimeMasterMeasurementType 55
8.4.1.2 TimeSlaveMeasurementType 57
8.4.1.3 PdelayInitiatorMeasurementType 59
8.4.1.4 PdelayResponderMeasurementType 61
8.4.2 Provider TimeBase Validation Notification 63
8.4.2.1 SetPdelayResponderData 63
8.4.2.2 SetMasterTimingData 64
8.4.3 Consumer TimeBase Provider Notification 64
8.4.3.1 SetPdelayInitiatorData 65
8.4.3.2 SetSlaveTimingData 65
8.5 C++ Time Precision Interface 65
8.5.1 Type definitions 65
8.5.1.1 TimePrecisionMeasurement type 65

9 Sequence diagrams 69

9.1 Interaction with Offset Time Bases 69
9.2 Application request status of a Synchronized TBR - and then takes information from such status. 70
9.3 Application request status of an Offset TBR 70

A Mentioned Class Tables 72

B Interfaces to other Functional Clusters (informative) 76

B.1 Overview 76
B.2 Interface Tables 76

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).

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

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.

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

2 Acronyms and Abbreviations

The glossary below includes acronyms, abbreviations and definitions relevant to the Time Synchronization module that are not included in the [3, AUTOSAR glossary] or in [4].

2.1 Acronyms and Abbreviations

Abbreviation / Acronym:	Description:
DST	Daylight Saving Time, also know as Day Light Saving (abbreviated DLS), 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.
gPTP	Generalized Precision Time Protocol
NTP	Network Time Protocol
OS	Operating System
Pdelay	Propagation/path delay as given in IEEE 802.1AS
Pdelay$_{Req}$	Propagation / path delay request message
Pdelay$_{Resp}$	Propagation / path delay response message
Pdelay$_{RespFollowUp}$	Propagation / path delay Follow-Up message
Sync	Time synchronization message (Sync)
PTP	Precision Time Protocol
r$_{oc}$	Rate for time offset elimination via Rate Adaption.
r$_{rc}$	Current rate for correcting the local instance of the Time Base.
StbM	Synchronized Time Base Manager
TBR	Time Base Resource
TCorrInt	OffsetCorrectionAdaptionInterval
TG	Received value of the Global Time.
TG$_{Start}$	Current time of the global Time Base Time Master.
TG$_{Stop}$	Current time of the Global Time Base Time Master.
Timesync	Time Synchronization (Refers to the action of Synchronizing the Time by means of a time synchronization protocol/bus/messages).
TL$_{Sync}$	Value of the local instance of the Time Base before the new value of the Global Time is applied.
TO$_{Start}$	Current corrected time provided by the local instance of the associated Time Base.
Current Offset of the Offset Time Base given as function parameter TO$_{Stop}$	Current Offset of the Offset Time Base given as function parameter.
TS	Time Synchronization
TSP	A bus specific Time Synchronization Provider.
TS$_{Start}$	Current corrected time provided by the local instance of the associated Time Base.
TS$_{Stop}$	Current corrected time provided by the local instance of the associated Time Base.
TV	Current value of the Virtual Local Time.
TV$_{Start}$	Current time of the Virtual Local Time of the associated Time Base.

Abbreviation / Acronym:	Description:
TV_{Stop}	Current time of the Virtual Local Time of the associated Time Base.
TV_{Sync}	Value of the Virtual Local Time
UTC	Coordinated Universal Time

2.2 Definitions

2.2.1 ara::core::SteadyClock

Definition: TS is using `ara::core::SteadyClock` as the basis for its interfaces and for synchronization with the daemon process realizing the time-sync protocol.

2.2.2 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. **Notification Application**
   This feature will be provided at a later release/version of the TS.

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] Requirements on Time Synchronization for Adaptive Platform
AUTOSAR_RS_TimeSync

[5] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[6] Specification of the Adaptive Core
AUTOSAR_SWS_AdaptiveCore

[7] Functional Cluster Shortnames
AUTOSAR_TR_FunctionalClusterShortnames

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

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

[10] Specification of Time Synchronization over Ethernet
AUTOSAR_SWS_TimeSyncOverEthernet

[11] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

NOTE: [5, RS-RSGeneral] is listed here as an input document because it applies to SWS TimeSync as well as to all SWS documents of the Adaptive Platform. Since it includes only non-functional requirements the tracing is not necessary.

3.2 Further applicable specification

AUTOSAR provides a core specification [6, SWS AdaptiveCore] which is also applicable for `Time Synchronization`. The chapter "General requirements for all Functional Clusters" of this specification shall be considered as an additional and required specification for implementation of `Time Synchronization`.

4 Constraints and assumptions

4.1 Known limitations

The Time Synchronization module is bound to Adaptive Platform Systems.

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 Functional Clusters

TS is part of the ara::tsync [7] namespace.

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Time Synchronization for Adaptive Platform [4] 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_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment.	[SWS_TS_01251]
[RS_TS_00002]	The Implementation of Time Synchronization shall maintain its own Time Base independently of the acting role.	[SWS_TS_00041] [SWS_TS_00042]
[RS_TS_00005]	The Implementation of Time Synchronization shall allow customers to have access to the Synchronized Time Base	[SWS_TS_01007] [SWS_TS_01109] [SWS_TS_01208]
[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_00042]
[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_00026] [SWS_TS_00027] [SWS_TS_00028] [SWS_TS_00030] [SWS_TS_00032] [SWS_TS_00033] [SWS_TS_00139] [SWS_TS_00140] [SWS_TS_00141] [SWS_TS_01050] [SWS_TS_01051] [SWS_TS_01108]
[RS_TS_00010]	The Implementation of Time Synchronization shall allow customer on master side to set the Global Time	[SWS_TS_01107] [SWS_TS_01108] [SWS_TS_01207]
[RS_TS_00011]	The Implementation of Time Synchronization shall allow customers on master side to trigger time transmission by the TSP module	[SWS_TS_01108]
[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_00055] [SWS_TS_00056] [SWS_TS_00057] [SWS_TS_00058] [SWS_TS_00059] [SWS_TS_00060]
[RS_TS_00014]	The Implementation of Time Synchronization shall allow customers to read User Data propagated via the TSP modules.	[SWS_TS_00120] [SWS_TS_01056] [SWS_TS_01113] [SWS_TS_01212]

Requirement	Description	Satisfied by
[RS_TS_00015]	The Implementation of Time Synchronization shall allow customers to set User Data propagated via the TSP modules.	[SWS_TS_01112] [SWS_TS_01211]
[RS_TS_00018]	The Implementation of Time Synchronization shall support rate correction	[SWS_TS_00041] [SWS_TS_00042] [SWS_TS_00043] [SWS_TS_00044] [SWS_TS_00045] [SWS_TS_00046] [SWS_TS_00047] [SWS_TS_00048] [SWS_TS_00049] [SWS_TS_00050] [SWS_TS_00051] [SWS_TS_00052] [SWS_TS_00053] [SWS_TS_00054] [SWS_TS_00061] [SWS_TS_00062] [SWS_TS_00063] [SWS_TS_00070] [SWS_TS_00071] [SWS_TS_01008] [SWS_TS_01110] [SWS_TS_01111] [SWS_TS_01209] [SWS_TS_01210]
[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_00120] [SWS_TS_00127] [SWS_TS_00129] [SWS_TS_00131] [SWS_TS_01009] [SWS_TS_01052] [SWS_TS_01053] [SWS_TS_01054] [SWS_TS_01055] [SWS_TS_01056] [SWS_TS_01113] [SWS_TS_01212]
[RS_TS_00023]	The Implementation of Time Synchronization shall offer interfaces able to handle std::chrono data types.	[SWS_TS_01001] [SWS_TS_01002] [SWS_TS_01003] [SWS_TS_01004] [SWS_TS_01005] [SWS_TS_01006] [SWS_TS_01101] [SWS_TS_01102] [SWS_TS_01103] [SWS_TS_01104] [SWS_TS_01105] [SWS_TS_01106] [SWS_TS_01201] [SWS_TS_01202] [SWS_TS_01203] [SWS_TS_01205] [SWS_TS_01206]
[RS_TS_00024]	The Implementation of Time Synchronization shall support storage of the Time Base value at shutdown if configured as Time Master	[SWS_TS_00212]
[RS_TS_00026]	The Implementation of Time Synchronization shall provide to the customers a specific API per type of Time Base Resource	[SWS_TS_01007] [SWS_TS_01107] [SWS_TS_01108] [SWS_TS_01109] [SWS_TS_01110] [SWS_TS_01112] [SWS_TS_01207] [SWS_TS_01208] [SWS_TS_01209] [SWS_TS_01211]
[RS_TS_00029]	The configuration of the Time Synchronization implementation shall allow the implementation to behave as a (vehicle wide) Time Master	[SWS_TS_00419] [SWS_TS_00421] [SWS_TS_00423] [SWS_TS_01108] [SWS_TS_01110] [SWS_TS_01112] [SWS_TS_01209] [SWS_TS_01211]

Requirement	Description	Satisfied by
[RS_TS_00030]	The configuration of the Time Synchronization implementation shall allow the implementation to behave as a Time Slave	[SWS_TS_00420] [SWS_TS_00422] [SWS_TS_00428] [SWS_TS_01000] [SWS_TS_01016] [SWS_TS_01017] [SWS_TS_01100] [SWS_TS_01114] [SWS_TS_01115] [SWS_TS_01200] [SWS_TS_01213] [SWS_TS_01214]
[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_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_00800] [SWS_TS_00801] [SWS_TS_00803] [SWS_TS_01016] [SWS_TS_01017] [SWS_TS_01018] [SWS_TS_01019] [SWS_TS_01114] [SWS_TS_01115] [SWS_TS_01213] [SWS_TS_01214] [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
- Shutdown Behavior
- Construction Behavior (Initialization)
- Normal Operation
- Error Handling
- Error Classification
- Version Check

7.1 General Overview of TS

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

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

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***
GetCurrentTime	StbM_GetCurrentTime
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
n/a	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

7.1.1 Base functionality of every Time Base

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

- offer possibility to obtain the current timestamp
- 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

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.

7.1.1.2 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} ⌈`ara::tsync::SynchronizedTimeBaseConsumer::GetRateDeviation` shall return the calculated rate deviation of its TBR against the time source it is synchronized to. In case there is no rate deviation calculated yet, the initial rate deviation of 1 shall be returned.⌋*()*

7.1.1.3 Clock Time Value

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

To ensure type safe handling of time values, the timepoint is provided as std::chrono structure.

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 GetTimeWithStatus.

Synchronization status GetSynchronizationStatus includes:

- `kNotSynchronizedUntilStartup`: Indicates whether a synchronization of a time base to its corresponding TBR happend until start-up (initial state)
- `kTimeOut`: Indicates whether a synchronization of a time base to its corresponding TBR is lost or delayed.
- `kSynchronized`: Indicates if the time base of the corresponding TBR has been successfully synchronized at least once against its time source.
- `kSynchToGateway`: Indicates if the corresponding TBR updates are based on a Time Gateway below the Global Time Master.

The status if a leap jump happend since the last status request through a GetTimeWithStatus could be retrieved via GetLeapJump:

- `kTimeLeapNone`: Indicates that no leap jump happend
- `kTimeLeapFuture`: Indicates if there has been a jump in time to the future.
- `kTimeLeapPast`: Indicates if there has been a jump in time to the past.

7.1.3 Time Synchronization and Protocols

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

7.2 Functional cluster life cycle

7.2.1 Startup

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.

7.2.1.1 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_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.

⌋*(RS_TS_00009)*

7.2.2 Shutdown

[SWS_TS_00212]{DRAFT} ⌈For each Time Base configured as Time Master the value of Global Time shall be stored into persistent memory ('`TimeBaseProviderToPersistencyMapping.timeBaseProvider`') (see [11]) if persistent storage is required by configuration parameter.⌋*(RS_TS_00024)*

7.3 Normal Operation

7.3.1 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.

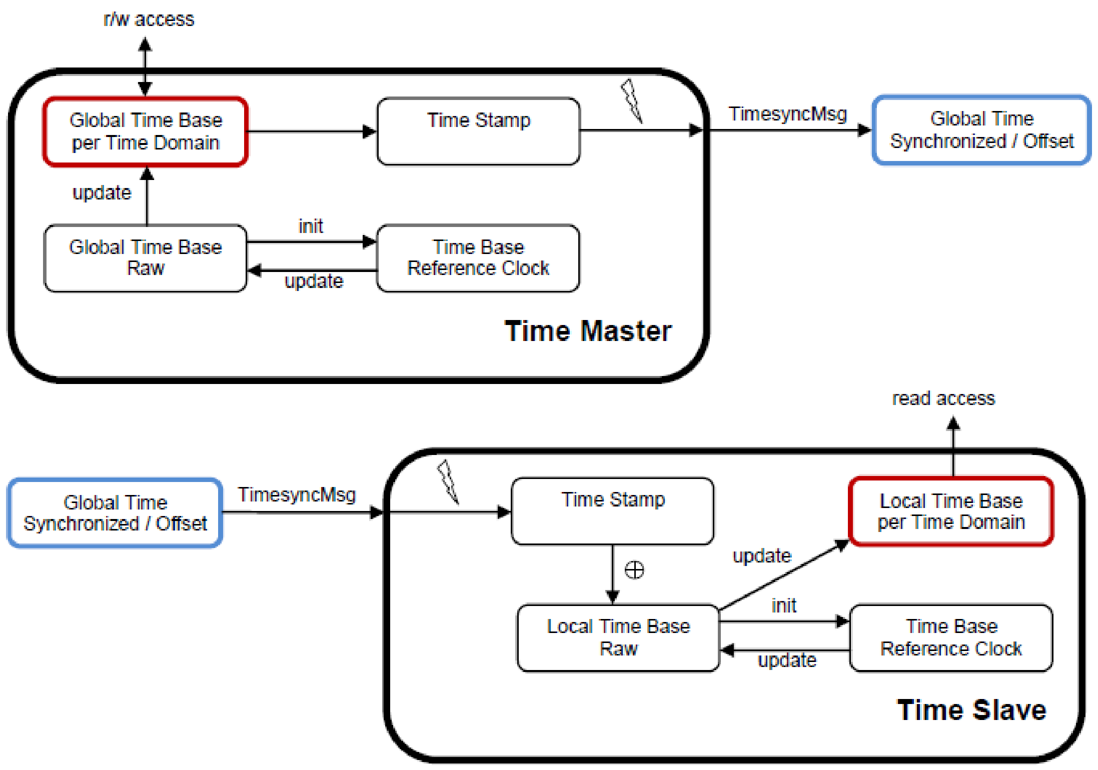
r/w access
Global Time Base per Time Domain
Time Stamp
TimesyncMsg
Global Time Synchronized / Offset
update
init
Global Time Base Raw
update
Time Base Reference Clock
Time Master
read access
Global Time Synchronized / Offset
TimesyncMsg
Time Stamp
Local Time Base per Time Domain
update
⊕
init
Local Time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

Figure 7.1: Global Time Base Distribution.

7.3.1.1 Time Base Manifestations

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

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).

7.3.2 Roles of the Time Base Resources

7.3.2.1 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.

7.3.2.2 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.

7.3.3 Time Base Resources

7.3.3.1 Slave Time Bases

[SWS_TS_00139]{DRAFT} ⌈Monitoring of time leaps to the future shall only be enabled, if a `timeLeapFutureThreshold` is other than zero and `ara::tsync::SynchronizationStatus` unequal to `kNotSynchronizedUntilStartup`.⌋*(RS_TS_00009)*

[SWS_TS_00140]{DRAFT} ⌈Monitoring of time leaps to the past shall only be enabled, if a `timeLeapPastThreshold` is other than zero and `ara::tsync::SynchronizationStatus` unequal to `kNotSynchronizedUntilStartup`.⌋*(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 (`ara::tsync::SynchronizationStatus` unequal to `kNotSynchronizedUntilStartup`).⌋*(RS_TS_00009)*

[SWS_TS_00027]{DRAFT} ⌈If the adjustment made by the resynchronization exceeded the specified threshold values, the corresponding `ara::tsync::LeapJump` status shall be set: `kTimeLeapNone`: if no leap jump happend. `kTimeLeapFuture`: if there has been a jump in time to the future greater than `timeLeapFutureThreshold`. `kTimeLeapPast`: if there has been a jump in time to the past greater than `timeLeapPastThreshold`.⌋*(RS_TS_00009)*

[SWS_TS_00026]{DRAFT} ⌈The initail value of (`ara::tsync::LeapJump` shall be `kTimeLeapNone`).⌋*(RS_TS_00009)*

[SWS_TS_00028]{DRAFT} ⌈Active Time Leap Status (`ara::tsync::LeapJump` shall be set to `kTimeLeapNone`), if a consecutive number `timeLeapHealingCounter` of synchronizations were all below the Time Leap Future and Past Thresholds.⌋*(RS_TS_00009)*

[SWS_TS_00030]{DRAFT} ⌈Each instance of `ara::tsync::SynchronizedTimeBaseConsumer` shall independetly monitor for a synchronization timeout by measuring the time since that last update and a specified timeout duration in `syncLossTimeout`.⌋*(RS_TS_00009)*

[SWS_TS_00032]{DRAFT} ⌈In case of a monitored timeout (refer [SWS_TS_00030]) the `ara::tsync::SynchronizationStatus` shall be set to `kTimeOut`.⌋*(RS_TS_00009)*

[SWS_TS_00011]{DRAFT} ⌈On a successful update of the Time Base and the SYNC_TO_GATEWAY bit is set, the `ara::tsync::SynchronizationStatus` shall be set to `kSynchToGateway`.⌋*(RS_TS_00009)*

[SWS_TS_00033]{DRAFT} ⌈On a successful update of the Time Base and the SYNC_TO_GATEWAY bit is NOT set, the `ara::tsync::SynchronizationStatus` shall be set to `kSynchronized`.⌋*(RS_TS_00009)*

7.3.4 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]).

7.3.5 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.

7.3.6 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.

7.3.6.1 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 `ara::tsync::SynchronizedTimeBaseConsumer::GetCurrentTime`).

[SWS_TS_00041]{DRAFT} ⌈The TBR shall perform Rate Correction measurements to determine its rate deviation if `ara::tsync::SynchronizationStatus` is set to `kSynchronized`. The measurement⌋*(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)*

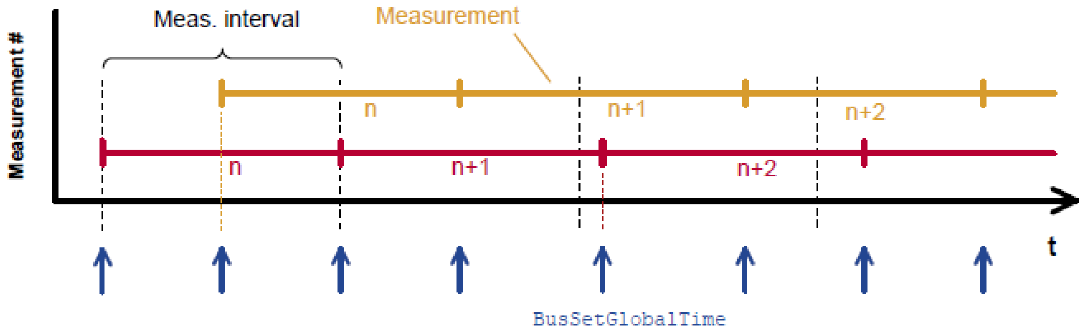

Figure 7.2: Visualization of two parallel measurements.

[SWS_TS_00043]{DRAFT} ⌈During runtime, the Synchronized TBR shall determine the timespan of a Rate Correction measurement on the basis of clock `ara::core::SteadyClock`.⌋*(RS_TS_00018)*

[SWS_TS_00044]{DRAFT} ⌈The TBR shall perform as many simultaneous Rate Correction measurements as configured by the parameter '`TimeSyncCorrection.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 * (rateDeviationMeasurementDuration / 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 time-snapshots TGStart and TOStart in the scope of TSP.⌋*(RS_TS_00018)*

[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`
- `TOStart`

[SWS_TS_00048]{DRAFT} ⌈At the end of the Rate Correction measurement, the Synchronized TBR shall take the time-snapshots TGStop and TVStop in the scope TSP.⌋ *(RS_TS_00018)*

[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)*

[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 `ara::tsync::SynchronizedTimeBaseConsumer::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 r$_{rc}$ has yet been calculated, `ara::tsync::SynchronizedTimeBaseConsumer::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$_{oc}$) has been calculated, the Offset TBR shall apply a Rate Correction.⌋*(RS_TS_00018, RS_TS_00019)*

7.3.6.2 Offset Correction for Time Consumer

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 `ara::tsync::SynchronizedTimeBaseConsumer::GetCurrentTime`). 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 TLSync and TVSync.⌋*(RS_TS_00013)*

[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 'TimeSyncCorrection.offsetCorrectionJumpThreshold', the TBR shall calculate the corrected time (TL) of its local instance of the Time Base according to the following formula:
$TL = TG + (TV - TV_{Sync}) * r_{rc}$⌋*(RS_TS_00013, RS_TS_00019)*

Note:
This correction will be done whenever the time is read in the scope of e.g. the function ara::tsync::SynchronizedTimeBaseConsumer::GetCurrentTime.

Note:
This correction will 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 'TimeSyncCorrection.offsetCorrectionJumpThreshold' by temporarily applying an additional rate (r_{oc}) to r_{rc}. This rate shall be used for the duration defined by parameter 'TimeSyncCorrection.offsetCorrectionAdaptionInterval'. r_{oc} is calculated according to the following formula:
$r_{oc} = (TG - TL_{Sync}) / (T_{CorrInt}) + 1$

⌋*(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 'TimeSyncCorrection.offsetCorrectionJumpThreshold', the TBR shall calculate the corrected time (TL) of its local instance of the Time Base **within** the period of 'TimeSyncCorrection.offsetCorrectionAdaptionInterval' according to the following formula:

$TL = TL_{Sync} + (r_{rc} * (TV - TV_{Sync}) * r_{oc})$

⌋*(RS_TS_00013, RS_TS_00019)*

Note:
This correction will be done whenever the time is read in the scope of e.g. the function ara::tsync::SynchronizedTimeBaseConsumer::GetCurrentTime.

Note:
This correction will 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 `TimeSyncCorrection.offsetCorrectionJumpThreshold`, the TBR shall calculate the corrected time (TL) of its local instance of the Time Base **after** the period of `TimeSyncCorrection.offsetCorrectionAdaptionInterval` as specified in [SWS_TS_00056]⌋*(RS_TS_00013)*

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

7.3.6.3 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 '`TimeSyncCorrection.allowProviderRateCorrection`' equals *true*, an invocation of `ara::tsync::SynchronizedTimeBaseProvider::SetRateCorrection` shall set the rate correction value. Otherwise `ara::tsync::SynchronizedTimeBaseProvider::SetRateCorrection` shall do nothing and return the error kLimitsExceeded⌋*(RS_TS_00018)*

[SWS_TS_00062]{DRAFT} ⌈The TBR shall apply rate correction, if `allowProviderRateCorrection` equals TRUE and a valid rate correction value has been set by `ara::tsync::SynchronizedTimeBaseProvider::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 `ara::tsync::SynchronizedTimeBaseProvider::SetRateCorrection`.

7.3.7 Notifications of Time Base Consumer

The Application might request to be notified of dedicated events for a specific TBR.

7.3.7.1 Status flags notification

A change in the StatusFlags of the `ara::tsync::SynchronizedTimeBaseStatus` can be notified.

[SWS_TS_00701]{DRAFT} ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterStatusChangeNotifier` shall be invoked, if one of the following content will change: `ara::tsync::SynchronizationStatus`, `ara::tsync::LeapJump` or the `user data`.⌋*()*

7.3.7.2 Synchronization status notification

A change in the StatusFlags of the `ara::tsync::SynchronizationStatus` (e.g. if the timebase is in Timeout) can be notified.

[SWS_TS_00702]{DRAFT} ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterSynchronizationStateChangeNotifier` shall be invoked, if `ara::tsync::SynchronizationStatus` is changing.⌋ *()*

7.3.7.3 LeapJump notification

A leap jump can be notified.

[SWS_TS_00703]{DRAFT} ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimeLeapNotifier` shall be invoked, if `ara::tsync::LeapJump` is changing.⌋*()*

7.3.8 Global Time Precision Measurement Support

To verify the precision of each Local Time Base compared to the Global Time Base a recording mechanism shall be optionally supported for Time Slaves and Time Gateways. In principle, a snapshot is taken of all required data at the point in time, where a synchronization event takes place. Access is provided to those values by an actively pushed API function on each successful assembled data block. An Off-Board Tester collects each block and calculates the precision afterwards and maintains a history of recorded blocks and their elements accordingly. How and by which protocol the data will be transferred to the Off-Board Tester will be specified by the Application.

[SWS_TS_00803]{DRAFT} ⌈A registration via shall only be possible for Synchronized Time Bases and Offset Time Bases, for which `isSystemWideGlobalTimeMaster` is set to FALSE.⌋*(RS_TS_00034)*

[SWS_TS_00800]{DRAFT} ⌈For Synchronized Time Bases, a registered TimePrecisionMeasurement notifier (via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimePrecisionMeasurementNotifier`) shall write the block elements

- `glbSeconds`
- `glbNanoSeconds`
- `timeBaseStatus`
- `virtualLocalTimeLow`
- `rateDeviation`
- `locSeconds`
- `locNanoSeconds`
- `pathDelay`

to the related measurement recording table after updating the Main Time Tuple (i.e., after updating the Local Time Base by the Global Time Base). GlbSeconds, GlbNanoSeconds are the elements of the Global Time part of the Received Time Tuple (i.e., TGRx); VirtualLocalTimeLow is the nanosecondsLo element of the Virtual Local Time part of the Received Time Tuple (i.e., TVRx).⌋*(RS_TS_00034)*

[SWS_TS_00801]{DRAFT} ⌈For Offset Time Bases, a registered TimePrecisionMeasurement notifier (via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimePrecisionMeasurementNotifier`) shall only write the block elements GlbSeconds, GlbNanoSeconds and TimeBaseStatus to the related measurement recording table.⌋*(RS_TS_00034)*

7.3.9 Global Time Validation Measurement Support

Figure 7.3 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.

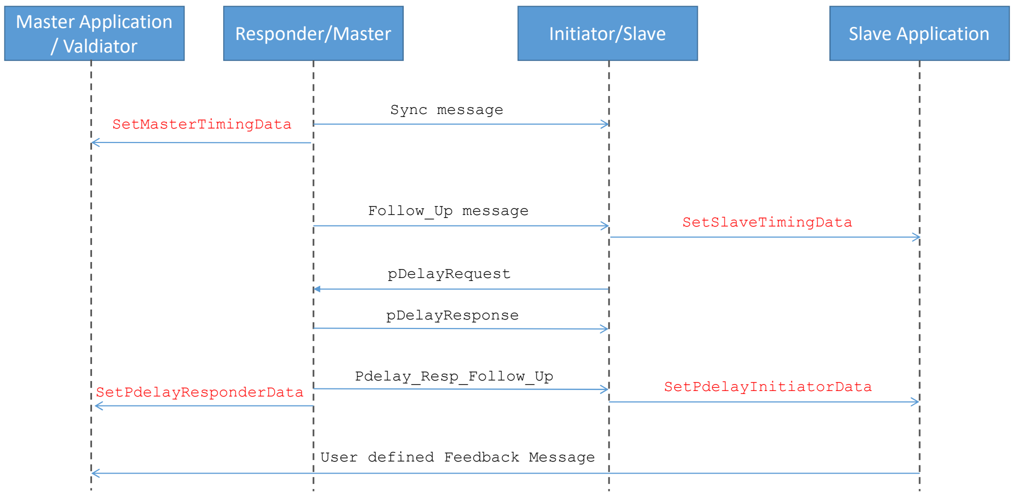

Figure 7.3: Time Validation mechanism

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 `ara::tsync::TimeSlaveMeasurementType` shall be updated and the function `ara::tsync::ConsumerTimeBaseValidationNotification::SetSlaveTimingData` shall be invoked.⌋*(RS_TS_00034)*

[SWS_TS_00425]{DRAFT} ⌈Everytime a Sync message is transmitted, all parameters defined by `ara::tsync::TimeMasterMeasurementType` shall be updated and the function `ara::tsync::ProviderTimeBaseValidationNotification::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 `ara::tsync::PdelayInitiatorMeasurementType` shall be updated and the function `ara::tsync::ConsumerTimeBaseValidationNotification::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 `ara::tsync::PdelayResponderMeasurementType` shall be updated and the function `ara::tsync::ProviderTimeBaseValidationNotification::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.

8 API specification

8.1 API Common Data Types

8.1.1 Timestamp

[SWS_TS_01251]{DRAFT} ⌈

Kind:	type alias
Symbol:	Timestamp
Scope:	namespace ara::tsync
Derived from:	std::chrono::time_point<TimeBase, std::chrono::nanoseconds>
Syntax:	`using Timestamp = std::chrono::time_point<TimeBase, std::chrono::nanoseconds>;`
Header file:	#include "ara/tsync/timestamp.h"
Description:	Standard timestamp type as alias of a generic time_point .

⌋*(RS_AP_00130)*

8.1.2 LeapJump

[SWS_TS_01051]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	LeapJump	
Scope:	namespace ara::tsync	
Underlying type:	std::uint32_t	
Syntax:	`enum class LeapJump : std::uint32_t {...};`	
Values:	kTimeLeapNone= 0	No adjustment back or greater than a certain threshold has been made.
	kTimeLeapFuture= 1	An adjustment greater than a certain threshold has been made.
	kTimeLeapPast= 2	An adjustment back in time greater than a certain threshold has been made.
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Enumeration that is used to express the leap jump of a time base. .	

⌋*(RS_TS_00009)*

8.1.3 SynchronizationStatus

[SWS_TS_01050]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	SynchronizationStatus	
Scope:	namespace ara::tsync	
Underlying type:	std::uint32_t	
Syntax:	`enum class SynchronizationStatus : std::uint32_t {...};`	
Values:	kNotSynchronizedUntilStartup= 0	The TB is not synchronized until startup (inital state)
	kTimeOut= 0x1	The TB was not synchronized within a certain time frame.
	kSynchronized= 0x2	The TB is in sync with the time master.
	kSynchToGateway= 0x3	The TB is in sync with the gateway.
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Enumeration that is used to express the communication state of a time base. .	

⌋*(RS_TS_00009)*

8.2 Common Function Definition of Time Bases Provider

8.2.1 SynchronizedTimeBaseProvider

[SWS_TS_01100]{DRAFT} ⌈

Kind:	class
Symbol:	SynchronizedTimeBaseProvider
Scope:	namespace ara::tsync
Syntax:	`class SynchronizedTimeBaseProvider final {...};`
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	Class SynchronizedTimeBaseProvider is the access to the synchronized timebase referenced by the IntstanceSpecifier. It allows to get the current time_point, the rate deviation, the current status and the received user data

⌋*(RS_TS_00030)*

8.2.1.1 Special member functions

[SWS_TS_01101]{DRAFT} ⌈

Kind:	function
Symbol:	SynchronizedTimeBaseProvider(const ara::core::InstanceSpecifier &specifier)
Scope:	class ara::tsync::SynchronizedTimeBaseProvider

▽

△

Syntax:	`explicit SynchronizedTimeBaseProvider (const ara::core::InstanceSpecifier &specifier) noexcept;`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an TimeSynchronizationInterface
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	SynchronizedTimeBaseProvider constructor.	

⌋(*RS_TS_00023*)

[SWS_TS_01102]{DRAFT} ⌈

Kind:	function	
Symbol:	SynchronizedTimeBaseProvider(SynchronizedTimeBaseProvider &&stbc)	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`SynchronizedTimeBaseProvider (SynchronizedTimeBaseProvider &&stbc) noexcept;`	
Parameters (in):	stbc	The SynchronizedTimeBaseProvider object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	Move constructor for SynchronizedTimeBaseProvider.	

⌋(*RS_TS_00023*)

[SWS_TS_01103]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(SynchronizedTimeBaseProvider &&stbc)	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`SynchronizedTimeBaseProvider& operator= (SynchronizedTimeBaseProvider &&stbc) &noexcept;`	
Parameters (in):	stbc	The SynchronizedTimeBaseProvider object to be moved.
Return value:	SynchronizedTimeBaseProvider &	The moved SynchronizedTimeBaseProvider object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	Move assignment operator for SynchronizedTimeBaseProvider.	

⌋(*RS_TS_00023*)

[SWS_TS_01104]{DRAFT} ⌈

Kind:	function
Symbol:	SynchronizedTimeBaseProvider(const SynchronizedTimeBaseProvider &)
Scope:	class ara::tsync::SynchronizedTimeBaseProvider
Syntax:	`SynchronizedTimeBaseProvider (const SynchronizedTimeBaseProvider &)=delete;`
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	The copy constructor for SynchronizedTimeBaseProvider shall not be used.

⌋*(RS_TS_00023)*

[SWS_TS_01105]{DRAFT} ⌈

Kind:	function
Symbol:	operator=(const SynchronizedTimeBaseProvider &)
Scope:	class ara::tsync::SynchronizedTimeBaseProvider
Syntax:	`SynchronizedTimeBaseProvider& operator= (const SynchronizedTimeBaseProvider &)=delete;`
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	The copy assignment operator for SynchronizedTimeBaseProvider shall not be used.

⌋*(RS_TS_00023)*

[SWS_TS_01106]{DRAFT} ⌈

Kind:	function
Symbol:	~SynchronizedTimeBaseProvider()
Scope:	class ara::tsync::SynchronizedTimeBaseProvider
Syntax:	`~SynchronizedTimeBaseProvider () noexcept;`
Exception Safety:	noexcept
Thread Safety:	no
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	Destructor for SynchronizedTimeBaseProvider.

⌋*(RS_TS_00023)*

8.2.1.2 SetTime

[SWS_TS_01107]{DRAFT} ⌈

Kind:	function
Symbol:	SetTime(ara::tsync::Timestamp timePoint, ara::core::Span< const ara::core::Byte > userData={})
Scope:	class ara::tsync::SynchronizedTimeBaseProvider

▽

△

Syntax:	`ara::core::Result<void> SetTime (ara::tsync::Timestamp timePoint, ara::core::Span< const ara::core::Byte > userData={}) noexcept;`	
Parameters (in):	timePoint	The time information to be set.
	userData	The user data to be set.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Errors:	TsyncErrc::kTimeCannotSet	the action cannot be executed, because the connection to time sync daemon is currently lost
Header file:	#include "ara/tsync/synchronized_time_base_provider.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)*

8.2.1.3 UpdateTime

[SWS_TS_01108]{DRAFT} ⌈

Kind:	function	
Symbol:	UpdateTime(ara::tsync::Timestamp, ara::core::Span< const ara::core::Byte > userData={})	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`ara::core::Result<void> UpdateTime (ara::tsync::Timestamp, ara::core::Span< const ara::core::Byte > userData={}) noexcept;`	
Template param:	Duration	The duration type of the time point passed as parameter.
Parameters (in):	userData	The user data to be set.
DIRECTION NOT DEFINED	ara::tsync::Timestamp	–
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Errors:	TsyncErrc::kDaemonConnectionLost	the action cannot be executed, because the connection to time sync daemon is currently lost
Header file:	#include "ara/tsync/synchronized_time_base_provider.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)*

8.2.1.4 GetCurrentTime

[SWS_TS_01109]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCurrentTime()	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`ara::tsync::Timestamp GetCurrentTime () const noexcept;`	
Return value:	ara::tsync::Timestamp	The current time as clock specific time point.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	Method to obtain the current time (regardless of the current sync status).	

⌋*(RS_TS_00026, RS_TS_00005)*

8.2.1.5 SetRateCorrection

[SWS_TS_01110]{DRAFT} ⌈

Kind:	function	
Symbol:	SetRateCorrection(double rateCorrection)	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`ara::core::Result<void> SetRateCorrection (double rateCorrection) noexcept;`	
Parameters (in):	rateCorrection	The rate correction to be applied. 0.5 is two times slower, whilst 2.0 is 2 times faster.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Errors:	ara::tsync::TsyncErrorDomain::Errc::kLimitsExceeded	–
Header file:	#include "ara/tsync/synchronized_time_base_provider.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)*

8.2.1.6 GetRateCorrection

[SWS_TS_01111]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRateDeviation()	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`double GetRateDeviation () const noexcept;`	
Return value:	double	The current rate deviation.
Exception Safety:	noexcept	

▽

△

Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	Method to obtain the current rate deviation of the clock.

⌋*(RS_TS_00018)*

8.2.1.7 SetUserData

[SWS_TS_01112]{DRAFT} ⌈

Kind:	function	
Symbol:	SetUserData(ara::core::Span< const ara::core::Byte > userData)	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`ara::core::Result<void> SetUserData (ara::core::Span< const ara::core::Byte > userData) noexcept;`	
Parameters (in):	userData	The user data to be set.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	Method that can be used to set user data.	

⌋*(RS_TS_00029, RS_TS_00026, RS_TS_00015)*

8.2.1.8 GetUserData

[SWS_TS_01113]{DRAFT} ⌈

Kind:	function	
Symbol:	GetUserData()	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`ara::core::Span<const ara::core::Byte> GetUserData () const noexcept;`	
Return value:	ara::core::Span< const ara::core::Byte >	A vector of bytes holding the user data that was set during the creation of the status.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	A method to return the user defined data of the time base.	

⌋*(RS_TS_00021, RS_TS_00014)*

8.2.1.9 RegisterTimeValidationNotification

[SWS_TS_01114]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterTimeValidationNotification(ProviderTimeBaseValidationNotification &timeBase ValidationNotification)	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Syntax:	`void RegisterTimeValidationNotification (ProviderTimeBaseValidation Notification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidationNotification	–
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Description:	Method that can be used by time provider applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

⌋*(RS_TS_00034, RS_TS_00030)*

8.2.1.10 UnregisterTimeValidationNotification

[SWS_TS_01115]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterTimeValidationNotification()
Scope:	class ara::tsync::SynchronizedTimeBaseProvider
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Description:	Method that can be used by time provider applications to receive time sync parameters.

⌋*(RS_TS_00034, RS_TS_00030)*

8.2.2 OffsetTimeBaseProvider

[SWS_TS_01200]{DRAFT} ⌈

Kind:	class
Symbol:	OffsetTimeBaseProvider
Scope:	namespace ara::tsync
Syntax:	`class OffsetTimeBaseProvider final {...};`
Header file:	#include "ara/tsync/offset_time_base_provider.h"

▽

△

Description:	Class OffsetTimeBaseProvider is the access to the offset timebase referenced by the Intstance Specifier. It allows to get the current time_point, the rate deviation, the current status and the received user data

⌋(*RS_TS_00030*)

8.2.2.1 Special member functions

[SWS_TS_01201]{DRAFT} ⌈

Kind:	function	
Symbol:	OffsetTimeBaseProvider(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`explicit OffsetTimeBaseProvider (const ara::core::InstanceSpecifier &specifier) noexcept;`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an Time SynchronizationInterface
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	OffsetTimeBaseProvider constructor.	

⌋(*RS_TS_00023*)

[SWS_TS_01202]{DRAFT} ⌈

Kind:	function	
Symbol:	OffsetTimeBaseProvider(OffsetTimeBaseProvider &&stbc)	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`OffsetTimeBaseProvider (OffsetTimeBaseProvider &&stbc) noexcept;`	
Parameters (in):	stbc	The OffsetTimeBaseProvider object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Move constructor for OffsetTimeBaseProvider.	

⌋(*RS_TS_00023*)

[SWS_TS_01203]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(OffsetTimeBaseProvider &&stbc)	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`OffsetTimeBaseProvider& operator= (OffsetTimeBaseProvider &&stbc) &noexcept;`	
Parameters (in):	stbc	The OffsetTimeBaseProvider object to be moved.
Return value:	OffsetTimeBaseProvider &	The moved OffsetTimeBaseProvider object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Move assignment operator for OffsetTimeBaseProvider.	

⌋*(RS_TS_00023)*

[SWS_TS_01205]{DRAFT} ⌈

Kind:	function
Symbol:	operator=(const OffsetTimeBaseProvider &)
Scope:	class ara::tsync::OffsetTimeBaseProvider
Syntax:	`OffsetTimeBaseProvider& operator= (const OffsetTimeBaseProvider &)=delete;`
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Description:	The copy assignment operator for OffsetTimeBaseProvider shall not be used.

⌋*(RS_TS_00023)*

[SWS_TS_01206]{DRAFT} ⌈

Kind:	function
Symbol:	~OffsetTimeBaseProvider()
Scope:	class ara::tsync::OffsetTimeBaseProvider
Syntax:	`~OffsetTimeBaseProvider () noexcept;`
Exception Safety:	noexcept
Thread Safety:	no
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Description:	Destructor for OffsetTimeBaseProvider.

⌋*(RS_TS_00023)*

8.2.2.2 SetOffsetTime

[SWS_TS_01207]{DRAFT} ⌈

Kind:	function	
Symbol:	SetOffsetTime(ara::tsync::Timestamp timePoint, ara::core::Span< const ara::core::Byte > user Data={})	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`ara::core::Result<void> SetOffsetTime (ara::tsync::Timestamp time Point, ara::core::Span< const ara::core::Byte > userData={}) noexcept;`	
Parameters (in):	timePoint	The time information to be set.
	userData	The user data to be set.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Errors:	TsyncErrc::kTimeCannotSet	the action cannot be executed, because the connection to time sync daemon is currently lost
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	A method that can be used to set a new offset time value for the clock. Setting a new time also triggers. transmission on the bus.	

⌋*(RS_TS_00010, RS_TS_00026)*

8.2.2.3 GetCurrentTime

[SWS_TS_01208]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCurrentTime()	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`ara::tsync::Timestamp GetCurrentTime () const noexcept;`	
Return value:	ara::tsync::Timestamp	The current time of the synchronized clock.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Method to obtain the current time (regardless of the current sync status).	

⌋*(RS_TS_00026, RS_TS_00005)*

8.2.2.4 SetRateCorrection

[SWS_TS_01209]{DRAFT} ⌈

Kind:	function
Symbol:	SetRateCorrection(double rateCorrection)
Scope:	class ara::tsync::OffsetTimeBaseProvider
Syntax:	`ara::core::Result<void> SetRateCorrection (double rateCorrection) noexcept;`

▽

△

Parameters (in):	rateCorrection	The rate correction to be applied. 0.5 is two times slower, whilst 2.0 is 2 times faster.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Errors:	ara::tsync::TsyncErrorDomain::Errc::kLimitsExceeded	–
Header file:	#include "ara/tsync/offset_time_base_provider.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)*

8.2.2.5 GetRateCorrection

[SWS_TS_01210]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRateDeviation()	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`double GetRateDeviation () const noexcept;`	
Return value:	double	The current rate deviation.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋*(RS_TS_00018)*

8.2.2.6 SetUserData

[SWS_TS_01211]{DRAFT} ⌈

Kind:	function	
Symbol:	SetUserData(ara::core::Span< const ara::core::Byte > userData)	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`ara::core::Result<void> SetUserData (ara::core::Span< const ara::core::Byte > userData) noexcept;`	
Parameters (in):	userData	The user data to be set.
Return value:	ara::core::Result< void >	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Method that can be used to set user data.	

⌋*(RS_TS_00029, RS_TS_00026, RS_TS_00015)*

8.2.2.7 GetUserData

[SWS_TS_01212]{DRAFT} ⌈

Kind:	function	
Symbol:	GetUserData()	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`ara::core::Span<const ara::core::Byte> GetUserData () const noexcept;`	
Return value:	ara::core::Span< const ara::core::Byte >	User data transported with the timesync protocol.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	A method to return the user defined data of the time base.	

⌋*(RS_TS_00021, RS_TS_00014)*

8.2.2.8 RegisterTimeValidationNotification

[SWS_TS_01213]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterTimeValidationNotification(ProviderTimeBaseValidationNotification &timeBaseValidationNotification)	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Syntax:	`void RegisterTimeValidationNotification (ProviderTimeBaseValidationNotification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidationNotification	–
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Description:	Method that can be used by time provider applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

⌋*(RS_TS_00034, RS_TS_00030)*

8.2.2.9 UnregisterTimeValidationNotification

[SWS_TS_01214]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterTimeValidationNotification()
Scope:	class ara::tsync::OffsetTimeBaseProvider
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Description:	Method that can be used by time provider applications to receive time sync parameters. .

⌋*(RS_TS_00034, RS_TS_00030)*

8.3 Common Function Definition of Time Bases Consumer

8.3.1 SynchronizedTimeBaseConsumer

[SWS_TS_01000]{DRAFT} ⌈

Kind:	class
Symbol:	SynchronizedTimeBaseConsumer
Scope:	namespace ara::tsync
Syntax:	`class SynchronizedTimeBaseConsumer final {...};`
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Class SynchronizedTimeBaseConsumer is the access to the synchronized timebase referenced by the IntstanceSpecifier. It allows to get the current time_point, the rate deviation, the current status and the received user data

⌋*(RS_TS_00030)*

8.3.1.1 Special member functions

[SWS_TS_01001]{DRAFT} ⌈

Kind:	function	
Symbol:	SynchronizedTimeBaseConsumer(const ara::core::InstanceSpecifier &specifier)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`explicit SynchronizedTimeBaseConsumer (const ara::core::Instance Specifier &specifier) noexcept;`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an Time SynchronizationInterface
Exception Safety:	noexcept	

▽

△

Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	SynchronizedTimeBaseConsumer constructor.

⌋(*RS_TS_00023*)

[SWS_TS_01002]{DRAFT} ⌈

Kind:	function	
Symbol:	~SynchronizedTimeBaseConsumer()	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`~SynchronizedTimeBaseConsumer () noexcept;`	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	SynchronizedTimeBaseConsumer destructor.	

⌋(*RS_TS_00023*)

[SWS_TS_01003]{DRAFT} ⌈

Kind:	function	
Symbol:	SynchronizedTimeBaseConsumer(SynchronizedTimeBaseConsumer &&stbc)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`SynchronizedTimeBaseConsumer (SynchronizedTimeBaseConsumer &&stbc) noexcept;`	
Parameters (in):	stbc	The SynchronizedTimeBaseConsumer object to be moved.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Move constructor for SynchronizedTimeBaseConsumer.	

⌋(*RS_TS_00023*)

[SWS_TS_01004]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(SynchronizedTimeBaseConsumer &&stbc)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`SynchronizedTimeBaseConsumer& operator= (SynchronizedTimeBaseConsumer &&stbc) &noexcept;`	
Parameters (in):	stbc	The SynchronizedTimeBaseConsumer object to be moved.
Return value:	SynchronizedTimeBaseConsumer &	The moved SynchronizedTimeBaseConsumer object.
Exception Safety:	noexcept	

▽

△

Thread Safety:	re-entrant
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Move assignment operator for SynchronizedTimeBaseConsumer. *

⌋*(RS_TS_00023)*

[SWS_TS_01005]{DRAFT} ⌈

Kind:	function
Symbol:	SynchronizedTimeBaseConsumer(const SynchronizedTimeBaseConsumer &)
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`SynchronizedTimeBaseConsumer (const SynchronizedTimeBaseConsumer &)=delete;`
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	The copy constructor for SynchronizedTimeBaseConsumer shall not be used.

⌋*(RS_TS_00023)*

[SWS_TS_01006]{DRAFT} ⌈

Kind:	function
Symbol:	operator=(const SynchronizedTimeBaseConsumer &)
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`SynchronizedTimeBaseConsumer& operator= (const SynchronizedTimeBaseConsumer &)=delete;`
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	The copy assignment operator for SynchronizedTimeBaseConsumer shall not be used.

⌋*(RS_TS_00023)*

8.3.1.2 GetCurrentTime

[SWS_TS_01007]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCurrentTime()	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`Timestamp GetCurrentTime () const noexcept;`	
Return value:	Timestamp	The current time of the synchronized clock.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	

▽

△

Description:	Method to obtain the current time (regardless of the current sync status).

⌋(*RS_TS_00026*, *RS_TS_00005*)

8.3.1.3 GetRateDeviation

[SWS_TS_01008]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRateDeviation()	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`double GetRateDeviation () const noexcept;`	
Return value:	double	The current rate deviation.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Method to obtain the current rate deviation of the clock.	

⌋(*RS_TS_00018*)

8.3.1.4 GetTimeWithStatus

[SWS_TS_01009]{DRAFT} ⌈

Kind:	function	
Symbol:	GetTimeWithStatus()	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`SynchronizedTimeBaseStatus GetTimeWithStatus () const noexcept;`	
Return value:	SynchronizedTimeBaseStatus	A clock specific TimeBaseStatus that contains all the relevant clock information.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.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 of the created status object.	

⌋(*RS_TS_00021*)

8.3.1.5 RegisterStatusChangeNotifier

[SWS_TS_01010]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterStatusChangeNotifier(std::function< void(const SynchronizedTimeBaseStatus &)> notifier)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`void RegisterStatusChangeNotifier (std::function< void(const SynchronizedTimeBaseStatus &)> notifier) noexcept;`	
DIRECTION NOT DEFINED	notifier	–
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Register a notifier function which is called if a StatusFlag is changed (i.e. synchronization state, time leap or userdata). A maximum of one notifier can be registered. Every further registration overwrites the current registration. @uptrace{}	

⌋*()*

8.3.1.6 UnregisterStatusChangeNotifier

[SWS_TS_01011]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterStatusChangeNotifier()
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`void UnregisterStatusChangeNotifier () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Unregister a notifier function which is called if a StatusFlag is changed (i.e. synchronization state, time leap or userdata). @uptrace{}.

⌋*()*

8.3.1.7 RegisterSynchronizationStateChangeNotifier

[SWS_TS_01012]{DRAFT} ⌈

Kind:	function
Symbol:	RegisterSynchronizationStateChangeNotifier(std::function< void(const SynchronizationStatus &)> notifier)
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer

▽

△

Syntax:	`void RegisterSynchronizationStateChangeNotifier (std::function< void(const SynchronizationStatus &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to unregister.
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Register a notifier function which is called if a synchronization state is changed. A maximum of one notifier can be registered. Every further registration overwrites the current registration. @uptrace{}	

⌋*()*

8.3.1.8 UnregisterSynchronizationStateChangeNotifier

[SWS_TS_01013]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterSynchronizationStateChangeNotifier()
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`void UnregisterSynchronizationStateChangeNotifier () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Unregister a notifier function which is called if a synchronization state is changed. @uptrace{}.

⌋*()*

8.3.1.9 RegisterTimeLeapNotifier

[SWS_TS_01014]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterTimeLeapNotifier(std::function< void(const SynchronizedTimeBaseStatus &)> notifier)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`void RegisterTimeLeapNotifier (std::function< void(const SynchronizedTimeBaseStatus &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to be called if the TimeBaseStatus has changed. @uptrace{}
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	

▽

△

Description:	Register a notifier function which is called if a time leap happend. A maximum of one notifier can be registered. Every further registration overwrites the current registration.

⌋*()*

8.3.1.10 UnregisterTimeLeapNotifier

[SWS_TS_01015]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterTimeLeapNotifier()
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`void UnregisterTimeLeapNotifier () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Unregister a notifier function which is called if a time leap happend. @uptrace{}.

⌋*()*

8.3.1.11 RegisterTimeValidationNotification

[SWS_TS_01016]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterTimeValidationNotification(ConsumerTimeBaseValidationNotification &timeBase ValidationNotification)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`void RegisterTimeValidationNotification (ConsumerTimeBaseValidation Notification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidationNotification	–
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Method that can be used by time consumer applications to receive time sync parameters. A maximum of one notifier can be registered. Every further registration overwrites the current registration.	

⌋*(RS_TS_00034, RS_TS_00030)*

8.3.1.12 UnregisterTimeValidationNotification

[SWS_TS_01017]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterTimeValidationNotification()
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Method that can be used by time consumer applications to receive time sync parameters. .

⌋*(RS_TS_00034, RS_TS_00030)*

8.3.1.13 RegisterTimePrecisionMeasurementNotifier

[SWS_TS_01018]{DRAFT} ⌈

Kind:	function	
Symbol:	RegisterTimePrecisionMeasurementNotifier(std::function< void(const TimePrecision Measurement &)> notifier)	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Syntax:	`void RegisterTimePrecisionMeasurementNotifier (std::function< void(const TimePrecisionMeasurement &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to be called.
Return value:	None	
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Description:	Register a notifier function which is called if a new time precision snapshot is available. A maximum of one notifier can be registered. Every further registration overwrites the current registration. The Tsync will not do any queuing. If needed it has to be done within the notifier.	

⌋*(RS_TS_00034)*

8.3.1.14 UnregisterTimePrecisionMeasurementNotifier

[SWS_TS_01019]{DRAFT} ⌈

Kind:	function
Symbol:	UnregisterTimePrecisionMeasurementNotifier()
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer

▽

△

Syntax:	`void UnregisterTimePrecisionMeasurementNotifier () noexcept;`
Return value:	None
Exception Safety:	noexcept
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Description:	Unregister a notifier function which is called if a new time precision snapshot is available. .

⌋*(RS_TS_00034)*

8.3.2 SynchronizedTimeBaseStatus

[SWS_TS_01052]{DRAFT} ⌈

Kind:	class
Symbol:	SynchronizedTimeBaseStatus
Scope:	namespace ara::tsync
Syntax:	`class SynchronizedTimeBaseStatus final {...};`
Header file:	#include "ara/tsync/synchronized_time_base_status.h"
Description:	This class represents a snapshot of a time point including his states. .

⌋*(RS_TS_00021)*

8.3.2.1 Special member functions

[SWS_TS_01057]{DRAFT} ⌈

Kind:	function	
Symbol:	SynchronizedTimeBaseStatus(SynchronizedTimeBaseStatus &&)	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`SynchronizedTimeBaseStatus (SynchronizedTimeBaseStatus &&) noexcept;`	
DIRECTION NOT DEFINED	SynchronizedTimeBaseStatus &&	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Move constructor of SynchronizedTimeBaseStatus.	

⌋*()*

[SWS_TS_01058]{DRAFT} ⌈

Kind:	function	
Symbol:	SynchronizedTimeBaseStatus(const SynchronizedTimeBaseStatus &)	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`SynchronizedTimeBaseStatus (const SynchronizedTimeBaseStatus &) noexcept;`	
DIRECTION NOT DEFINED	const SynchronizedTimeBaseStatus &	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Copy constructor of SynchronizedTimeBaseStatus (needed for return by value)	

⌋*()*

[SWS_TS_01059]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(SynchronizedTimeBaseStatus &&)	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`SynchronizedTimeBaseStatus& operator= (SynchronizedTimeBaseStatus &&) &noexcept;`	
DIRECTION NOT DEFINED	SynchronizedTimeBaseStatus &&	–
Return value:	SynchronizedTimeBaseStatus &	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Move assignment operator of SynchronizedTimeBaseStatus.	

⌋*()*

[SWS_TS_01060]{DRAFT} ⌈

Kind:	function	
Symbol:	operator=(const SynchronizedTimeBaseStatus &)	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`SynchronizedTimeBaseStatus& operator= (const SynchronizedTimeBaseStatus &) noexcept;`	
DIRECTION NOT DEFINED	const SynchronizedTimeBaseStatus &	–
Return value:	SynchronizedTimeBaseStatus &	–
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Copy assignment operator of SynchronizedTimeBaseStatus.	

⌋*()*

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

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

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

8.3.2.2 GetCreationTime

[SWS_TS_01055]{DRAFT} ⌈

Kind:	function	
Symbol:	GetCreationTime()	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`ara::tsync::Timestamp GetCreationTime () const noexcept;`	
Return value:	ara::tsync::Timestamp	The point in time at which this object was created. Time point is expressed in context of the clock that created this object.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	A method to obtain the creation time of this object.	

⌋*(RS_TS_00021)*

8.3.2.3 GetSynchronizationStatus

[SWS_TS_01053]{DRAFT} ⌈

Kind:	function	
Symbol:	GetSynchronizationStatus()	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`SynchronizationStatus GetSynchronizationStatus () const noexcept;`	
Return value:	SynchronizationStatus	SynchronizationStatus
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Method that return the synchronization state the object was created.	

⌋*(RS_TS_00021)*

8.3.2.4 GetLeapJump

[SWS_TS_01054]{DRAFT} ⌈

Kind:	function	
Symbol:	GetLeapJump()	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`LeapJump GetLeapJump () const noexcept;`	
Return value:	LeapJump	LeapJump
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	Method that can be used to determin the direction of a leap jump. Only the jump until the previous object creation is included.	

⌋*(RS_TS_00021)*

8.3.2.5 GetUserData

[SWS_TS_01056]{DRAFT} ⌈

Kind:	function	
Symbol:	GetUserData()	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Syntax:	`ara::core::Span<const ara::core::Byte> GetUserData () const noexcept;`	
Return value:	ara::core::Span< const ara::core::Byte >	A vector of bytes holding the user data that was set during the creation of the status. A size of zero indicates that no user data is available.
Exception Safety:	noexcept	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Description:	A method to return the user defined data of the time base.	

⌋*(RS_TS_00021, RS_TS_00014)*

[SWS_TS_00120]{DRAFT} ⌈In case there are no User Data stored, `ara::tsync::SynchronizedTimeBaseStatus::GetUserData` shall return an empty vector.⌋*(RS_TS_00014, RS_TS_00021)*

8.4 C++ Time Validation Interface

8.4.1 Type definitions

8.4.1.1 TimeMasterMeasurementType

[SWS_TS_00414]{DRAFT} ⌈

Kind:	struct
Symbol:	TimeMasterMeasurementType
Scope:	namespace ara::tsync
Syntax:	`struct TimeMasterMeasurementType final {...};`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	Structure with detailed data for validation of the Time Master .

⌋*(RS_TS_00034)*

[SWS_TS_14140]{DRAFT} ⌈

Kind:	variable
Symbol:	preciseOriginTimestamp
Scope:	struct ara::tsync::TimeMasterMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp preciseOriginTimestamp;`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	egress timestamp of Sync frame in Global Time

⌋*(RS_TS_00034)*

[SWS_TS_14141]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14142]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

8.4.1.2 TimeSlaveMeasurementType

[SWS_TS_00415]{DRAFT} ⌈

Kind:	struct
Symbol:	TimeSlaveMeasurementType
Scope:	namespace ara::tsync
Syntax:	`struct TimeSlaveMeasurementType final {...};`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	Structure with detailed data for validation of the Time Slave .

⌋*(RS_TS_00034)*

[SWS_TS_14150]{DRAFT} ⌈

Kind:	variable
Symbol:	preciseOriginTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp preciseOriginTimestamp;`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	preciseOriginTimestamp taken from the received Follow_Up frame

⌋*(RS_TS_00034)*

[SWS_TS_14151]{DRAFT} ⌈

Kind:	variable
Symbol:	referenceGlobalTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp referenceGlobalTimestamp;`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	SyncLocal Time Tuple (Global Time part) .

⌋*(RS_TS_00034)*

[SWS_TS_14152]{DRAFT} ⌈

Kind:	variable
Symbol:	syncIngressTimestamp
Scope:	struct ara::tsync::TimeSlaveMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t syncIngressTimestamp;`

▽

△

Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	ingress timestamp of Sync frame converted to Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14153]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14154]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14155]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14156]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

8.4.1.3 PdelayInitiatorMeasurementType

[SWS_TS_00416]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14160]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14161]{DRAFT} ⌈

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

▽

△

Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	ingress timestamp of Pdelay_Resp in Virtual Local Time

⌋(*RS_TS_00034*)

[SWS_TS_14162]{DRAFT} ⌈

Kind:	variable
Symbol:	requestReceiptTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp requestReceiptTimestamp;`
Header file:	#include "ara/tsync/time_validation_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:	responseOriginTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp responseOriginTimestamp;`
Header file:	#include "ara/tsync/time_validation_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:	referenceLocalTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint64_t
Syntax:	`std::uint64_t referenceLocalTimestamp;`
Header file:	#include "ara/tsync/time_validation_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:	referenceGlobalTimestamp
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp referenceGlobalTimestamp;`
Header file:	#include "ara/tsync/time_validation_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:	pDelay
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Type:	std::uint32_t
Syntax:	`std::uint32_t pDelay;`
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	currently valid pDelay value

⌋*(RS_TS_00034)*

[SWS_TS_14167]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

8.4.1.4 PdelayResponderMeasurementType

[SWS_TS_00417]{DRAFT} ⌈

Kind:	struct
Symbol:	PdelayResponderMeasurementType
Scope:	namespace ara::tsync
Syntax:	`struct PdelayResponderMeasurementType final {...};`

▽

△

Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Description:	Structure with detailed timing data for the pDelay Responder .

⌋(*RS_TS_00034*)

[SWS_TS_14170]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14171]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14172]{DRAFT} ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14173]{DRAFT} ⌈

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

⌋*(RS_TS_00034)*

8.4.2 Provider TimeBase Validation Notification

[SWS_TS_00419]{DRAFT} ⌈

Kind:	class
Symbol:	ProviderTimeBaseValidationNotification
Scope:	namespace ara::tsync
Syntax:	`class ProviderTimeBaseValidationNotification {...};`
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"
Description:	Callback interface to notify (Validator) Application about the availability of a new data block recorded for the Time Base. gfddfgfdg

⌋*(RS_TS_00034, RS_TS_00029)*

8.4.2.1 SetPdelayResponderData

[SWS_TS_00423]{DRAFT} ⌈

Kind:	function	
Symbol:	SetPdelayResponderData(const PdelayResponderMeasurementType &measurementData)	
Scope:	class ara::tsync::ProviderTimeBaseValidationNotification	
Syntax:	`virtual void SetPdelayResponderData (const PdelayResponderMeasurementType &measurementData)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Responder
Return value:	None	
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"	
Description:	Provide the recorded data block for the pPelay Responder of the Time Base.	

⌋*(RS_TS_00034, RS_TS_00029)*

8.4.2.2 SetMasterTimingData

[SWS_TS_00421]{DRAFT} ⌈

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

⌋*(RS_TS_00034, RS_TS_00029)*

8.4.3 Consumer TimeBase Provider Notification

[SWS_TS_00428]{DRAFT} ⌈

Kind:	class
Symbol:	ConsumerTimeBaseValidationNotification
Scope:	namespace ara::tsync
Syntax:	`class ConsumerTimeBaseValidationNotification {...};`
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"
Description:	Callback interface to notify Consumer Application about the availability of a new data block recorded for the Time Base. .

⌋*(RS_TS_00034, RS_TS_00030)*

8.4.3.1 SetPdelayInitiatorData

[SWS_TS_00422]{DRAFT} ⌈

Kind:	function	
Symbol:	SetPdelayInitiatorData(const PdelayInitiatorMeasurementType &measurementData)	
Scope:	class ara::tsync::ConsumerTimeBaseValidationNotification	
Syntax:	`virtual void SetPdelayInitiatorData (const PdelayInitiatorMeasurement Type &measurementData)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Initiator
Return value:	None	
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"	
Description:	Provide the recorded data block for the pPelay Initiator of the Time Base.	

⌋*(RS_TS_00034, RS_TS_00030)*

8.4.3.2 SetSlaveTimingData

[SWS_TS_00420]{DRAFT} ⌈

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

⌋*(RS_TS_00034, RS_TS_00030)*

8.5 C++ Time Precision Interface

8.5.1 Type definitions

8.5.1.1 TimePrecisionMeasurement type

[SWS_TS_01400]{DRAFT} ⌈

Kind:	struct
Symbol:	TimePrecisionMeasurement
Scope:	namespace ara::tsync
Syntax:	`struct TimePrecisionMeasurement final {...};`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Structure with detailed data for precision measurement of the Time Slave @uptrace{}.

⌋*()*

[SWS_TS_01401]{DRAFT} ⌈

Kind:	variable
Symbol:	glbSeconds
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t glbSeconds;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Seconds of the Local Time Base directly after synchronization with the Global Time Base. Range: 0..4294967295 (4 Byte)

⌋*()*

[SWS_TS_01402]{DRAFT} ⌈

Kind:	variable
Symbol:	glbNanoSeconds
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t glbNanoSeconds;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Nanoseconds of the Local Time Base directly after synchronization with the Global Time Base. Range: 0..999999999 (4 Byte)

⌋*()*

[SWS_TS_01403]{DRAFT} ⌈

Kind:	variable
Symbol:	timeBaseStatus
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint8_t
Syntax:	`std::uint8_t timeBaseStatus;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"

▽

△

Description:	TimeBaseStatus Time Base Status of the Local Time Base directly after synchronization with the Global Time Base TIMEOUT 0x01 Bit 0 (LSB): 0x00: No Timeout on receiving Synchronisation Messages 0x01: Timeout on receiving Synchronisation Messages SYNC_TO_GATEWAY 0x04 Bit 2 0x00: Local Time Base is synchronous to Global Time Master 0x04: Local Time Base updates are based on a Time Gateway below the Global Time Master GLOBAL_TIME_BASE 0x08 Bit 3 0x00: Local Time Base is based on Local Time Base reference clock only (never synchronized with Global Time Base) 0x08: Local Time Base was at least synchronized with Global Time Base one time TIMELEAP_FUTURE 0x10 Bit 4 0x00: No leap into the future within the received time for Time Base 0x10: Leap into the future within the received time for Time Base exceeds a configured threshold TIMELEAP_PAST 0x20 Bit 5 0x00: No leap into the past within the received time for Time Base 0x20: Leap into the past within the received time for Time Base exceeds a configured threshold Description Bit 1, 6, and 7 are always 0 (reserved for future usage) Variables of this type are used to express if and how a Local Time Base is synchronized to the Global Time Master. The type is a bitfield of individual status bits, although not every combination is possible, i.e. any of the bits TIMEOUT, TIMELEAP_FUTURE, TIMELEAP_PAST and SYNC_TO_GATEWAY can only be set if the GLOBAL_TIME_BASE bit is set. .

⌋*()*

[SWS_TS_01404]{DRAFT} ⌈

Kind:	variable
Symbol:	virtualLocalTimeLow
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t virtualLocalTimeLow;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Least significant 32 bit of the Virtual Local Time directly after synchronization with the Global Time Base. Range: 0..4294967295 (4 Byte)

⌋*()*

[SWS_TS_01405]{DRAFT} ⌈

Kind:	variable
Symbol:	rateDeviation
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::int16_t
Syntax:	`std::int16_t rateDeviation;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Calculated Rate Deviation directly after rate deviation measurement. Range: 0..+-32000 (2 Byte)

⌋*()*

[SWS_TS_01406]{DRAFT} ⌈

Kind:	variable
Symbol:	locSeconds
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t locSeconds;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Seconds of the Local Time Base directly before synchronization with the Global Time Base. Range: 0..4294967295 (4 Byte)

⌋*()*

[SWS_TS_01407]{DRAFT} ⌈

Kind:	variable
Symbol:	locNanoSeconds
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t locNanoSeconds;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Nanoseconds of the Local Time Base directly before synchronization with the Global Time Base. Range: 0..999999999 (4 Byte)

⌋*()*

[SWS_TS_01408]{DRAFT} ⌈

Kind:	variable
Symbol:	pathDelay
Scope:	struct ara::tsync::TimePrecisionMeasurement
Type:	std::uint32_t
Syntax:	`std::uint32_t pathDelay;`
Header file:	#include "ara/tsync/time_precision_measurement_type.h"
Description:	Current propagation delay in nanoseconds. Range: 0..4294967295 (4 Byte)

⌋*()*

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 Interaction with Offset Time Bases

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

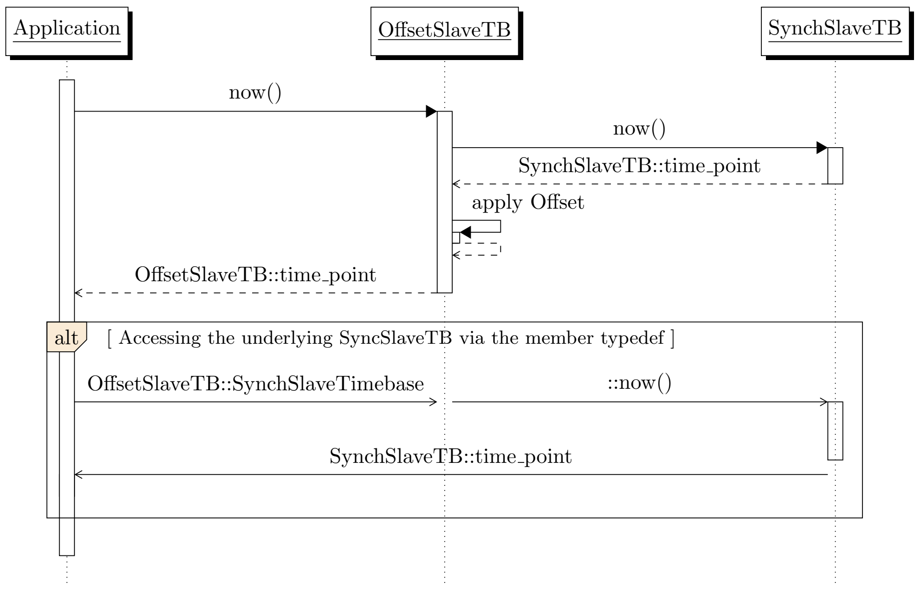
Application
OffsetSlaveTB
SynchSlaveTB
now()
now()
SynchSlaveTB::time_point
apply Offset
OffsetSlaveTB::time_point
alt
[Accessing the underlying SyncSlaveTB via the member typedef]
OffsetSlaveTB::SynchSlaveTimebase
::now()
SynchSlaveTB::time_point

Figure 9.1: Offset Time Base Handling.

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

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

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

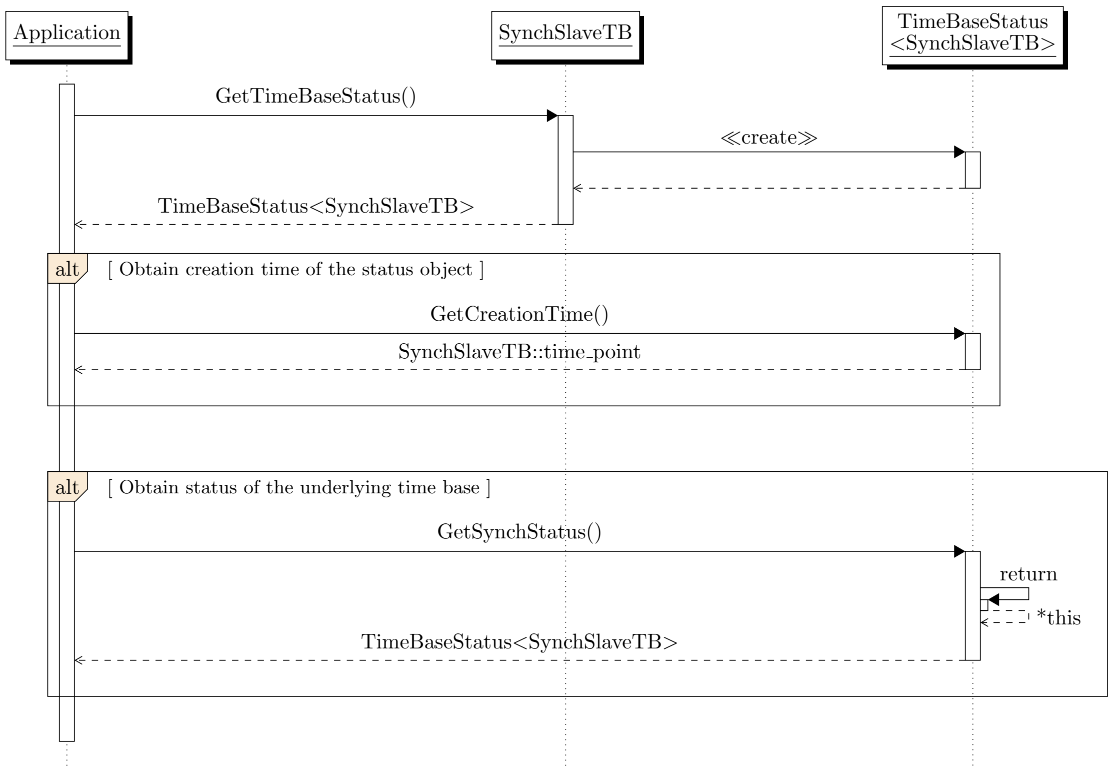
Application
SynchSlaveTB
TimeBaseStatus <SynchSlaveTB>
GetTimeBaseStatus()
≪create≫
TimeBaseStatus<SynchSlaveTB>
alt
[Obtain creation time of the status object]
GetCreationTime()
SynchSlaveTB::time_point
alt
[Obtain status of the underlying time base]
GetSynchStatus()
return
*this
TimeBaseStatus<SynchSlaveTB>

Figure 9.2: Request time base status of SynchTB.

9.3 Application request status of an Offset TBR

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

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

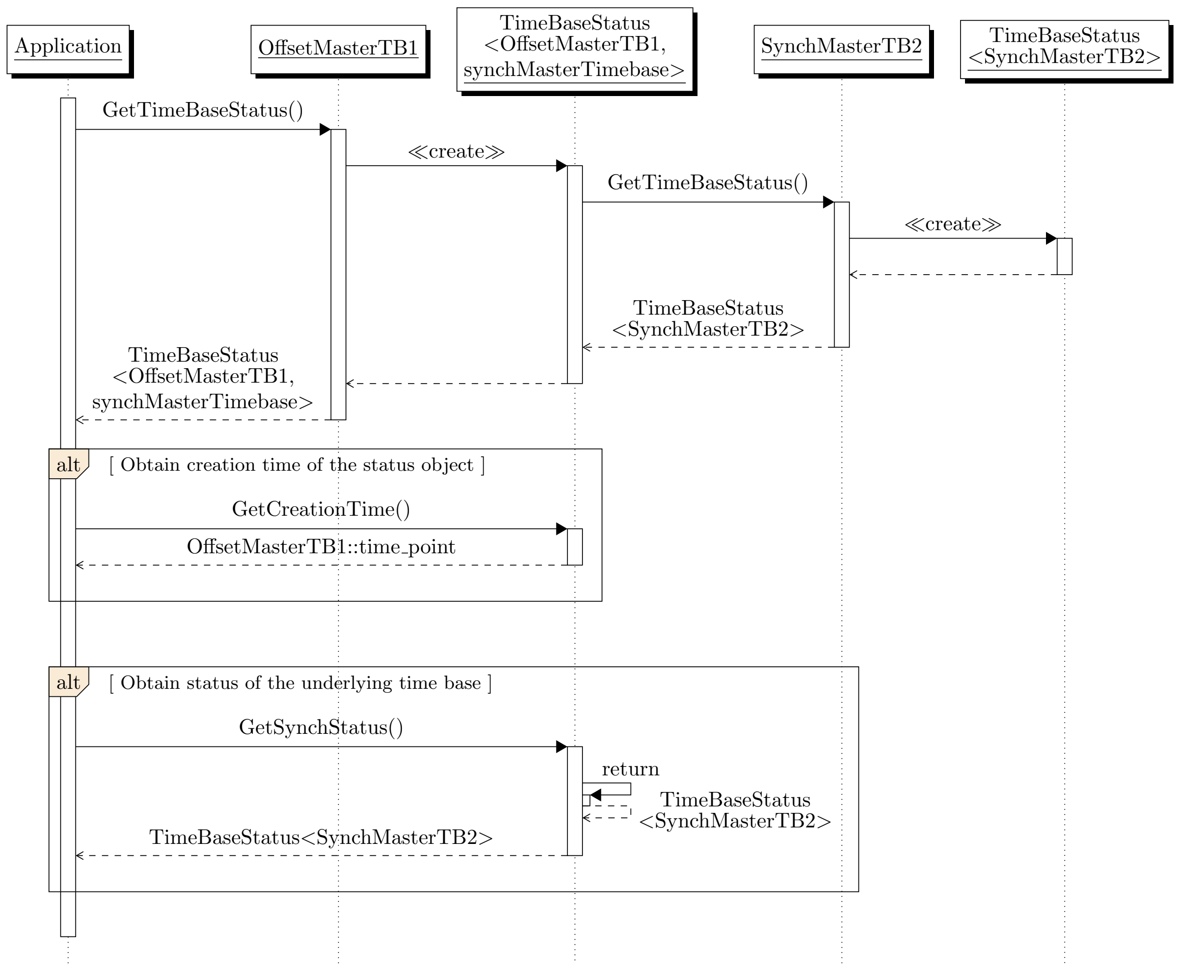
Application
OffsetMasterTB1
TimeBaseStatus <OffsetMasterTB1, synchMasterTimebase>
SynchMasterTB2
TimeBaseStatus <SynchMasterTB2>
GetTimeBaseStatus()
≪create≫
GetTimeBaseStatus()
≪create≫
TimeBaseStatus <SynchMasterTB2>
TimeBaseStatus <OffsetMasterTB1, synchMasterTimebase>
alt
[Obtain creation time of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

Figure 9.3: Request time base status of OffsetTB

A Mentioned Class Tables

For the sake of completeness, this chapter contains class tables representing meta-classes mentioned in the context of this document.

Class	**TimeSyncCorrection**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::TimeSync			
Note	This meta-class represents the attributes used for the correction of time synchronization. **Tags:**atp.Status=draft			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allowProviderRateCorrection	Boolean	0..1	attr	Defines whether the rate correction value of a Time Base can be set by means of the method setRateCorrection(). false: rate correction cannot be set by method setRateCorrection(). true: rate correction can be set by method setRateCorrection(). **Tags:**atp.Status=draft
offsetCorrectionAdaptionInterval	TimeValue	0..1	attr	Defines the interval during which the adaptive rate correction cancels out the rate and time deviation. Unit: seconds. **Tags:**atp.Status=draft
offsetCorrectionJumpThreshold	TimeValue	0..1	attr	Threshold for the correction method. Deviations below this value will be corrected by a linear reduction over a defined timespan. Values equal and greater than this value will be corrected by immediately setting the correct time and rate in form of a jump. Unit: seconds. **Tags:**atp.Status=draft
rateCorrectionsPerMeasurementDuration	PositiveInteger	0..1	attr	Number of simultaneous rate measurements to determine the current rate deviation. **Tags:**atp.Status=draft
rateDeviationMeasurementDuration	TimeValue	0..1	attr	Time span used to calculate the rate deviation. Unit: seconds. **Tags:**atp.Status=draft

Table A.1: TimeSyncCorrection

Class	**TimeBaseProviderToPersistencyMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::TimeSync			

Note	This meta-class represents the ability to define a mapping between a TimeBaseProvider and a PersistencyDeploymentElement for the purpose of storing and retrieving the time value.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=FCInteractions			

Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**TimeBaseProviderToPersistencyMapping**			
persistency Deployment Element	PersistencyDeployment Element	0..1	ref	This reference represents the PersistencyDeployment Element where the time value shall be stored in and retrieved from. **Tags:**atp.Status=draft
timeBase Provider	SynchronizedTimeBase Provider	0..1	ref	This reference represents the mapped TimeBase Provider. **Tags:**atp.Status=draft

Table A.2: TimeBaseProviderToPersistencyMapping

Class	**GlobalTimeDomain**			
Package	M2::AUTOSARTemplates::SystemTemplate::GlobalTime			
Note	This represents the ability to define a global time domain. **Tags:**atp.recommendedPackage=GlobalTimeDomains			
Base	*ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
debounceTime	TimeValue	0..1	attr	Defines the minimum amount of time between two time sync messages are transmitted.
domainId	PositiveInteger	1	attr	This represents the ID of the GlobalTimeDomain used in the network messages sent on behalf of global time management.
gateway	GlobalTimeGateway	*	aggr	A GlobalTimeGateway may exist in the context of a GlobalTimeDomain to actively update the global time information as it is routed from one GlobalTimeDomain to another. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
globalTime CorrectionProps	GlobalTimeCorrection Props	0..1	aggr	Defintion of attributes for rate and offset correction.
globalTime Domain Property	AbstractGlobalTime DomainProps	0..1	aggr	Additional properties of the GlobalTimeDomain. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
globalTime Master	GlobalTimeMaster	0..1	aggr	This represents the single master of a GlobalTime Domain. A GlobalTimeDomain may have no GlobalTime Domain.master, e.g. when it gets its time from a GPS receiver. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
globalTimeSub Domain	GlobalTimeDomain	*	ref	By this means it is possible to create a hierarchy of sub Domains where one global time domain can declare one or more other global time domains as its subDomains. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
network SegmentId	NetworkSegment Identification	0..1	aggr	Defines the numerical identification of a GlobalTime sub domain.
offsetTime Domain	GlobalTimeDomain	0..1	ref	Reference to a synchronized time domain this offset time domain is based on. The reference source is the offset time domain. The reference target is the synchronized time domain.

▽

△

Class	**GlobalTimeDomain**			
pduTriggering	PduTriggering	0..1	ref	This PduTriggering will be taken to transmit the global time information from a GlobalTimeMaster to a the associated GlobalTimeSlaves. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
slave	GlobalTimeSlave	*	aggr	This represents the collections of slaves of the Global TimeDomain. A GlobalTimeDomain may have no Global TimeDomain.slaves, e.g. when it propagates its time directly to sub domains. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
syncLoss Timeout	TimeValue	0..1	attr	This attribute describes the timeout for the situation that the time synchronization gets lost in the scope of the time domain.

Table A.3: GlobalTimeDomain

Class	***GlobalTimeMaster*** (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::GlobalTime			
Note	This represents the generic concept of a global time master.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	GlobalTimeCanMaster, GlobalTimeEthMaster, GlobalTimeFrMaster, UserDefinedGlobalTimeMaster			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
communication Connector	Communication Connector	1	ref	The GlobalTimeMaster is bound to the Communication Connector.
immediate ResumeTime	TimeValue	0..1	attr	Defines the minimum time between an "immediate" message and the next periodic message.
isSystemWide GlobalTime Master	Boolean	1	attr	If set to TRUE, the GlobalTimeMaster is supposed to act as the root of global time information.
syncPeriod	TimeValue	1	attr	This represents the period. Unit: seconds

Table A.4: GlobalTimeMaster

Class	***GlobalTimeSlave*** (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::GlobalTime			
Note	This represents the generic concept of a global time slave.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	GlobalTimeCanSlave, GlobalTimeEthSlave, GlobalTimeFrSlave, UserDefinedGlobalTimeSlave			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
communication Connector	Communication Connector	1	ref	The GlobalTimeSlave is bound to the Communication Connector.
followUp TimeoutValue	TimeValue	0..1	attr	Rx timeout for the follow-up message.
timeLeapFuture Threshold	TimeValue	0..1	attr	Defines the maximum allowed positive difference between the current Local Time Base value and a newly received Global Time Base value.

▽

△

Class	***GlobalTimeSlave*** (abstract)			
timeLeap HealingCounter	PositiveInteger	0..1	attr	Defines the required number of updates to the Time Base where the time difference to the previous received value has to remain within the bounds of timeLeapFuture Threshold and timeLeapPastThreshold until that Time Base is considered healed.
timeLeapPast Threshold	TimeValue	0..1	attr	Defines the maximum allowed negative difference between the current Local Time Base value and a newly received Global Time Base value.

Table A.5: GlobalTimeSlave

B Interfaces to other Functional Clusters (informative)

Note: This is chapter is created in the scope of the new SWS document template and in the current version is not applicable.

B.1 Overview

AUTOSAR decided not to standardize interfaces which are exclusively used between Functional Clusters (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides informative guidelines how the interaction between Functional Clusters looks like, by clustering the relevant requirements of this document to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapters 8 can also be used for interaction between Functional Clusters.

The goal is to provide a clear understanding of Functional Cluster boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementation of different Functional Clusters. Details of the interfaces are up to the platform provider. Additional interfaces, parameters and return values can be added.

B.2 Interface Tables