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

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R23-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Start of transmission of sync message updated • List of Adaptive Platform Functional Clusters added • Application errors added
2022-11-24	R22-11	AUTOSAR Release Management	• Support for restorage of Global Time from persistent memory added • Several minor clarifications and corrections
2021-11-25	R21-11	AUTOSAR Release Management	• Document clean-up, Title changed • Chapter "9 Sequence diagrams" temporarily deleted for clean-up purposes • Uptraces updated • Review findings (terminology, typos, etc.) resolved
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

Disclaimer

Contents

1 Introduction and functional overview

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

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

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

The Time Synchronization functionality is then offered by means of different "Time Base Resources" (from now on referred to as TBR).

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:
backupTimestamp	Value of the Global Time which is stored into persistent memory and used at startup for calculating the initial value of the Time Base.
Clock Update Counter	Counter value belonging to the Time Base, that indicates a Time Base update.
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
$\text{Pdelay}_{RespFollowUp}$	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}	Global Time part of the Updated Rx Time Tuple taken at the start of a Rate Correction measurement.
TG_{Stop}	Global Time part of the Updated Rx Time Tuple taken at the end of a Rate Correction measurement.
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.
TS	Time Synchronization
TSP	A bus specific Time Synchronization Provider.
TV	Current value of the Virtual Local Time.
TV_{Start}	Virtual Local Time part of the Updated Rx Time Tuple taken at the start of a Rate Correction measurement.
TV_{Stop}	Virtual Local Time part of the Updated Rx Time Tuple taken at the end of a Rate Correction measurement.

Abbreviation / Acronym:	Description:
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] Time Synchronization Protocol Specification
AUTOSAR_FO_PRS_TimeSyncProtocol

[2] Specification of Synchronized Time-Base Manager
AUTOSAR_CP_SWS_SynchronizedTimeBaseManager

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] Requirements on Time Synchronization
AUTOSAR_FO_RS_TimeSync

[5] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[6] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[7] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[8] ISO/IEC 14882:2011, Information technology – Programming languages – C++
https://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_CP_SWS_TimeSyncOverEthernet

[11] Specification of Manifest
AUTOSAR_AP_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.

4.1.4 Security

Secured Time Synchronization using the AUTOSAR Sub-TLV: Time Authenticated (see PRS-TimeSync [1]) is currently not supported for the Adaptive Platform.

Note: Secured Time Synchronization messages received in AP ECUs works without verifying the security measures (i.e., AUTOSAR Sub-TLV:Time Authenticated is ignored).

4.1.5 Offset Time Bases

The Time Synchronization module of the Adaptive Platform does not support rate correction for Offset Time Bases.

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

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by `Time Synchronization` to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by `Time Synchronization`.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [7].

5.1 Provided Interfaces

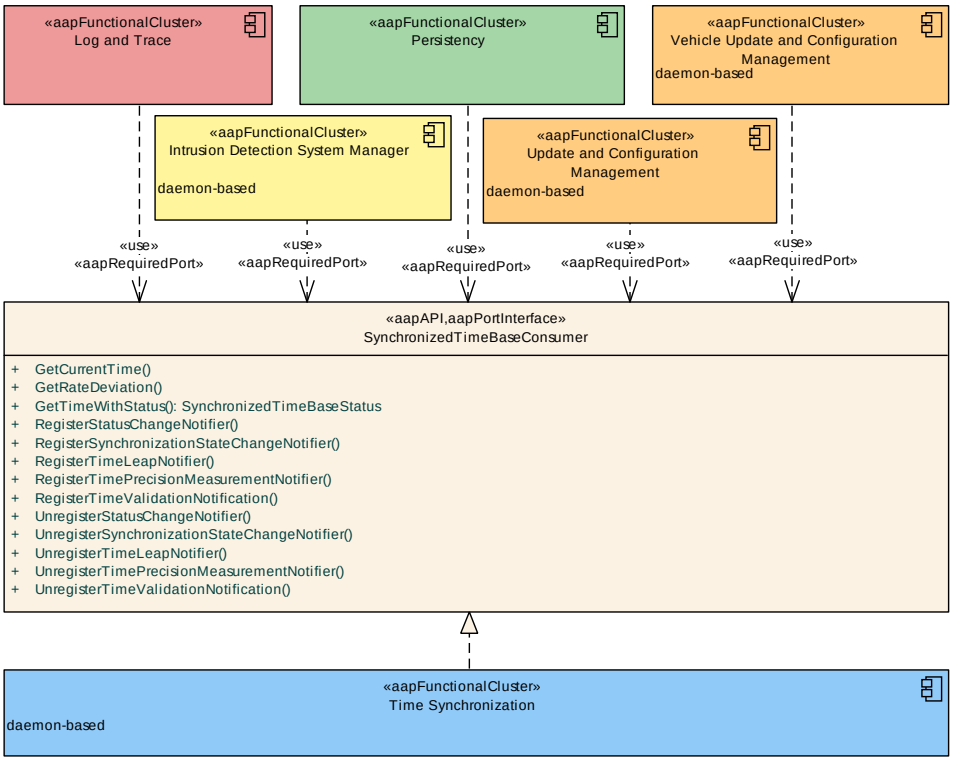

Figure 5.1: Interfaces provided by TimeSynchronization to other Functional Clusters

Figure 5.1 shows the interfaces provided by `Time Synchronization` to other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
`SynchronizedTime BaseConsumer`	Intrusion Detection System Manager	`Adaptive Intrusion Detection System Manager` shall use this interface to determine timestamps of security events.

△

Interface	*Functional Cluster*	*Purpose*
	Log and Trace	`Log and Trace` shall use this interface to determine the timestamps that are associated with log messages.
	Persistency	`Persistency` should use this interface to determine timestamps included in the meta-information of files, e.g., modification timestamp.
	Update and Configuration Management	`Update and Configuration Management` shall use this interface to get latest timestamp.
	Vehicle Update and Configuration Management	`Vehicle Update and Configuration Management` shall use this interface to get latest timestamp.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

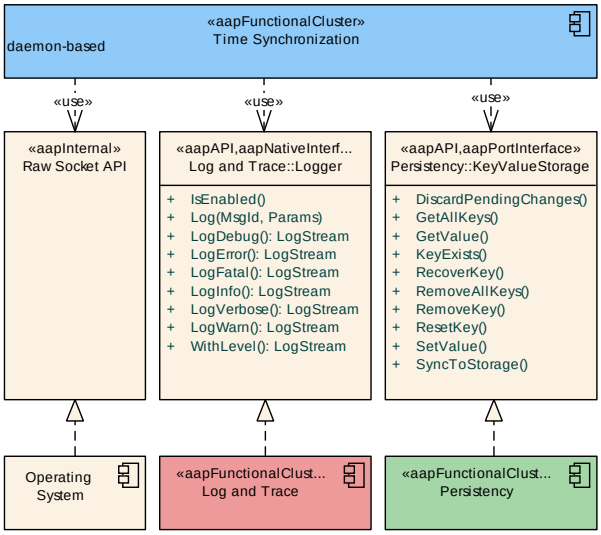

Figure 5.2: Interfaces required by Time Synchronization from other Functional Clusters

Figure 5.2 shows the interfaces required by `Time Synchronization` from other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.2 provides a complete list of required interfaces from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
Execution Management	ExecutionClient	`Time Synchronization` shall use this interface to report the state of its daemon process.
Log and Trace	Logger	`Time Synchronization` shall use this interface to log standardized messages.
Persistency	KeyValueStorageOperations	`Time Synchronization` should use this interface to persist the last received timestamp to enable a faster startup.

▽

△

Functional Cluster	*Interface*	*Purpose*
Persistency	KeyValueStorage	`Time Synchronization` should use this interface to persist the last received timestamp to enable a faster startup.
Platform Health Management	SupervisedEntity	`Time Synchronization` should use this interface to enable supervision of its daemon process by `Platform Health Management`

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Time Synchronization [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_00120]	Method and Function names.	[SWS_TS_00903] [SWS_TS_00905] [SWS_TS_00906] [SWS_TS_00907] [SWS_TS_00908] [SWS_TS_00909] [SWS_TS_00910]
[RS_AP_00121]	Parameter names.	[SWS_TS_00903] [SWS_TS_00907] [SWS_TS_00908] [SWS_TS_00910]
[RS_AP_00122]	Type names.	[SWS_TS_00901] [SWS_TS_00902] [SWS_TS_00904]
[RS_AP_00127]	Usage of ara::core types.	[SWS_TS_00901] [SWS_TS_00902] [SWS_TS_00904]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment.	[SWS_TS_00901] [SWS_TS_00902] [SWS_TS_00903] [SWS_TS_00904] [SWS_TS_00905] [SWS_TS_00906] [SWS_TS_00907] [SWS_TS_00908] [SWS_TS_00909] [SWS_TS_00910] [SWS_TS_01251] [SWS_TS_01260] [SWS_TS_01261] [SWS_TS_01262] [SWS_TS_01263] [SWS_TS_01264] [SWS_TS_01265]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_TS_00903] [SWS_TS_00905] [SWS_TS_00906] [SWS_TS_00907] [SWS_TS_00909] [SWS_TS_00910]
[RS_AP_00150]	Provide only interfaces that are intended to be used by AUTOSAR applications and other Functional Clusters.	[SWS_TS_00902] [SWS_TS_00904]
[RS_AP_00154]	Internal namespaces.	[SWS_TS_00901] [SWS_TS_00902] [SWS_TS_00904] [SWS_TS_00909] [SWS_TS_00910]
[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_00004]	The Implementation of Time Synchronization shall initialize the Global Time Base with a configurable startup value.	[SWS_TS_00213]
[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] [SWS_TS_01251] [SWS_TS_01260] [SWS_TS_01261] [SWS_TS_01262] [SWS_TS_01263] [SWS_TS_01264] [SWS_TS_01265]
[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]

△

Requirement	Description	Satisfied by
[RS_TS_00009]	The Implementation of Time Synchronization shall maintain the synchronization status of a Time Base	[SWS_TS_00007] [SWS_TS_00011] [SWS_TS_00027] [SWS_TS_00028] [SWS_TS_00030] [SWS_TS_00032] [SWS_TS_00033] [SWS_TS_00064] [SWS_TS_00139] [SWS_TS_00140] [SWS_TS_00141] [SWS_TS_00702] [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]
[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_00202] [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_00701] [SWS_TS_01009] [SWS_TS_01052] [SWS_TS_01053] [SWS_TS_01054] [SWS_TS_01055] [SWS_TS_01056] [SWS_TS_01057] [SWS_TS_01058] [SWS_TS_01059] [SWS_TS_01060] [SWS_TS_01113] [SWS_TS_01212] [SWS_TS_01403]

▽

△

Requirement	Description	Satisfied by
[RS_TS_00023]	No description	[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_01204] [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] [SWS_TS_00213] [SWS_TS_00214] [SWS_TS_00215]
[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]
[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] [SWS_TS_01300] [SWS_TS_01301]
[RS_TS_00033]	The Implementation of Time Synchronization shall use a time format with a resolution of 1 ns	[SWS_TS_01251] [SWS_TS_01260] [SWS_TS_01261] [SWS_TS_01262] [SWS_TS_01263] [SWS_TS_01264] [SWS_TS_01265]
[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_00703] [SWS_TS_00800] [SWS_TS_00801] [SWS_TS_00803] [SWS_TS_01010] [SWS_TS_01011] [SWS_TS_01012] [SWS_TS_01013] [SWS_TS_01014] [SWS_TS_01015] [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_01300] [SWS_TS_01301] [SWS_TS_01400] [SWS_TS_01401] [SWS_TS_01402] [SWS_TS_01403] [SWS_TS_01404] [SWS_TS_01405] [SWS_TS_01406] [SWS_TS_01407] [SWS_TS_01408] [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] ▽

▽

△

Requirement	Description	Satisfied by
		△ [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]
[RS_TS_20047]	The Timesync over Ethernet module shall trigger Time Base Synchronization transmission	[SWS_TS_14175]
[RS_TS_20058]	The Timesync over Ethernet module shall provide the precision of Synchronized Time Bases	[SWS_TS_14175]

Table 6.1: RequirementsTracing

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] ⌈`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 0.0 shall be returned.⌋*(RS_TS_00018)*

Note: For more information of how rate deviation is calculated see: 7.3.6 Time Correction.

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 `ara::tsync::SynchronizedTimeBaseConsumer::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.

[SWS_TS_14175]{DRAFT} ⌈A TBR configured as Time Master shall start the transmission of sync message only if the network is available (link is up) and in one of the following conditions:

- a valid time is set or updated by the application or
- the backupTimeStamp is successfully restored from persistent memory

⌋*(RS_TS_20047, RS_TS_20058)*

[SWS_TS_00213]{DRAFT} ⌈For each TBR configured as Time Master for which storage to persistent memory is activated, i.e. a 'TimeBaseProvider-ToPersistencyMapping.timeBaseProvider' is present, the value of Global Time shall be restored from persistent memory such that the value of backupTimestamp is used to initialize the Time Base. Immediately after successfully loading the stored backupTimestamp, the Time Master shall store a new backupTimestamp ($= loaded(old) backupTimestamp +$ TimeBaseProviderToPersistencyMapping.cyclicBackupInterval)⌋*(RS_TS_00024, RS_TS_00004)*

[SWS_TS_00214]{DRAFT} ⌈In case the restore from persistent memory is not successful, the Time Base shall start with zero.⌋*(RS_TS_00024)*

[SWS_TS_00215]{DRAFT} ⌈For each TBR configured as Time Slave, Clock Update Counter shall be initialized with zero.⌋*(RS_TS_00024)*

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.
- 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 TBR configured as Time Master for which storage to persistent memory is activated, i.e a 'TimeBaseProviderToPersisten-

cyMapping.timeBaseProvider' is present, first the current value of the Global Time shall be read every 'TimeBaseProviderToPersistencyMapping.cyclicBackupInterval'. Then the value of the 'TimeBaseProviderToPersistencyMapping.cyclicBackupInterval' itself shall be added and this check-pointed value of the Global Time shall then be stored into persistent memory 'TimeBaseProviderToPersistencyMapping.timeBaseProvider' (see [11]) as backupTimestamp if persistent storage is required. The initial value of backupTimestamp shall be set to 0.

Upon a graceful shutdown the Global Time shall be stored without applying another 'TimeBaseProviderToPersistencyMapping.cyclicBackupInterval' as back-off.⌋ *(RS_TS_00024)*

Note: Regardless of the exact shutdown event, the last stored value of backupTimestamp will be restored during the next startup (see 7.2.1).

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.

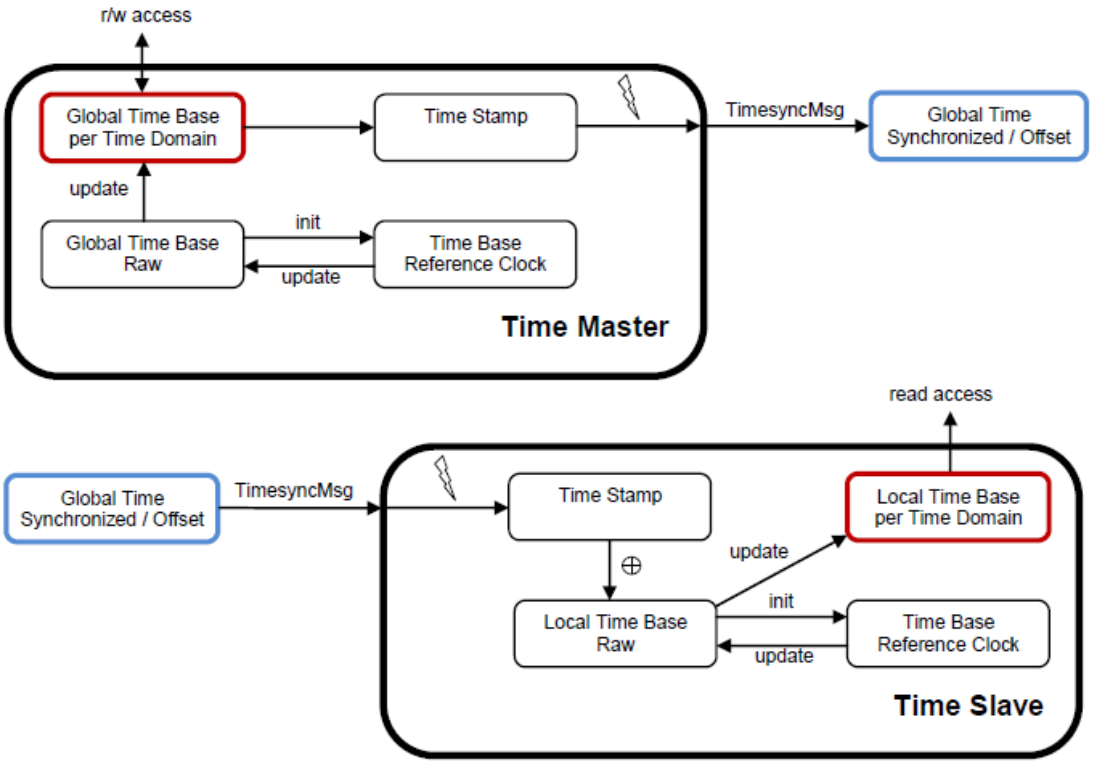

Figure 7.1: Global Time Base Distribution.

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] ⌈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] ⌈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] ⌈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] ⌈If the adjustment made by the resynchronization exceeded the specified threshold values, the corresponding `ara::tsync::LeapJump` status shall be set to `kTimeLeapNone` if no leap jump occurred. `kTimeLeapFuture`: if jump occured in time to the future greater than `timeLeapFutureThreshold`. `kTimeLeapPast`: if jump occured in time to the past greater than `timeLeapPastThreshold`.⌋ *(RS_TS_00009)*

[SWS_TS_00064] ⌈The initial value of `ara::tsync::LeapJump` shall be `kTimeLeapNone`.⌋*(RS_TS_00009)*

[SWS_TS_00028] ⌈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] ⌈Each instance of `ara::tsync::SynchronizedTimeBaseConsumer` shall independently 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] ⌈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] ⌈If the update of the Time Base is successful and SYNC_TO_GATEWAY bit is set, the `ara::tsync::SynchronizationStatus` shall be set to `kSynchToGateway`.⌋*(RS_TS_00009)*

[SWS_TS_00033] ⌈If the update of the Time Base is successful 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`.⌋*(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.⌋*(RS_TS_00002, RS_TS_00007, RS_TS_00018, RS_TS_00019)*

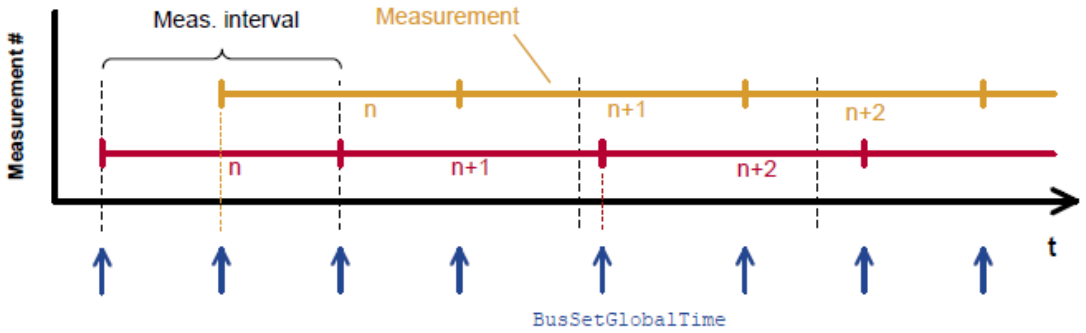

Figure 7.2: Visualization of two parallel measurements.

[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 `TVStart` 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:

- `TSStart`
- `TOStart`

⌋*(RS_TS_00018)*

[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]{OBSOLETE} ⌈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 Synchronized 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 correction 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]{OBSOLETE} ⌈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 `ara::tsync::SynchronizedTimeBaseProvider::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] ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterStatusChangeNotifier` shall be invoked, if one of the following content is changed: `ara::tsync::SynchronizationStatus`, `ara::tsync::LeapJump` or the `user data`.⌋*(RS_TS_00021)*

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] ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterSynchronizationStateChangeNotifier` shall be invoked, if `ara::tsync::SynchronizationStatus` is changed.⌋*(RS_TS_00009)*

7.3.7.3 LeapJump notification

A leap jump can be notified.

[SWS_TS_00703] ⌈A registered notifier via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimeLeapNotifier` shall be invoked, if `ara::tsync::LeapJump` is changed.⌋*(RS_TS_00034)*

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] ⌈A registration via `ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimePrecisionMeasurementNotifier` 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] ⌈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] ⌈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.

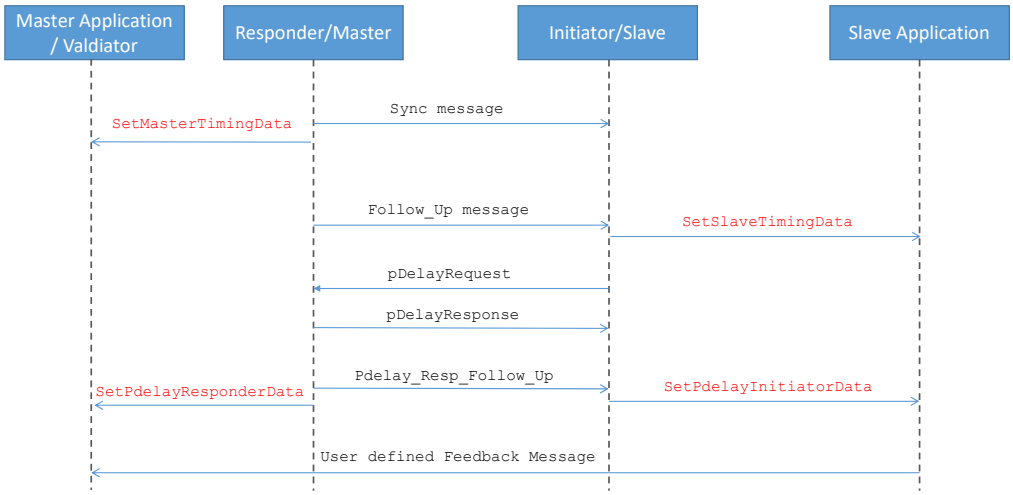

Figure 7.3: Time Validation mechanism

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} **Definition of API type ara::tsync::Timestamp** ⌈

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

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2 TimeBase struct

[SWS_TS_01260]{DRAFT} **Definition of API class ara::tsync::TimeBase** ⌈

Kind:	struct
Header file:	#include "ara/tsync/timestamp.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	TimeBase
Syntax:	`struct TimeBase {...};`
Description:	The class TimeBase is a pseudo-clock. The class TimeBase is a pseudo-clock that is used as the first template argument to std::chrono::time_point to indicate that the time point represents local time with respect of a not-yet-specified timeBaseResource.

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2.1 rep

[SWS_TS_01261]{DRAFT} **Definition of API type ara::tsync::TimeBase::rep** ⌈

Kind:	type alias
Header file:	#include "ara/tsync/timestamp.h"
Scope:	struct ara::tsync::TimeBase
Symbol:	rep
Syntax:	`using rep = std::int64_t;`
Description:	required type declaration to fullfill the C++ Clock requirements representing the number of ticks

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2.2 period

[SWS_TS_01262]{DRAFT} **Definition of API type ara::tsync::TimeBase::period** ⌈

Kind:	type alias
Header file:	#include "ara/tsync/timestamp.h"
Scope:	struct ara::tsync::TimeBase
Symbol:	period
Syntax:	`using period = std::nano;`
Description:	required type declaration to fullfill the C++ Clock requirements representing the tick period of the duration

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2.3 duration

[SWS_TS_01263]{DRAFT} **Definition of API type ara::tsync::TimeBase::duration** ⌈

Kind:	type alias
Header file:	#include "ara/tsync/timestamp.h"
Scope:	struct ara::tsync::TimeBase
Symbol:	duration
Syntax:	`using duration = std::chrono::duration<rep, period>;`
Description:	required type declaration to fullfill the C++ Clock requirements used to measure the time since epoch

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2.4 time_point

[SWS_TS_01264]{DRAFT} **Definition of API type ara::tsync::TimeBase::time_point** ⌈

Kind:	type alias
Header file:	#include "ara/tsync/timestamp.h"
Scope:	struct ara::tsync::TimeBase
Symbol:	time_point
Syntax:	`using time_point = std::chrono::time_point<TimeBase>;`
Description:	represents a point in time

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.2.5 is_steady

[SWS_TS_01265]{DRAFT} **Definition of API variable ara::tsync::TimeBase::is_steady** ⌈

Kind:	variable
Header file:	#include "ara/tsync/timestamp.h"
Scope:	struct ara::tsync::TimeBase
Symbol:	is_steady
Type:	bool
Syntax:	`static constexpr bool is_steady = false;`
Description:	required constant to fullfill the C++ Clock requirements as steady clock

⌋(*RS_AP_00130*, *RS_TS_00033*, *RS_TS_00005*)

8.1.3 LeapJump

[SWS_TS_01051] Definition of API enum ara::tsync::LeapJump ⌈

Kind:	enumeration	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"	
Scope:	namespace ara::tsync	
Symbol:	LeapJump	
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.
Description:	Enumeration that is used to express the leap jump of a time base. .	

⌋(*RS_TS_00009*)

8.1.4 SynchronizationStatus

[SWS_TS_01050] Definition of API enum ara::tsync::SynchronizationStatus ⌈

Kind:	enumeration
Header file:	#include "ara/tsync/synchronized_time_base_status.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	SynchronizationStatus
Underlying type:	std::uint32_t

▽

△

Syntax:	`enum class SynchronizationStatus : std::uint32_t {...};`	
Values:	kNotSynchronizedUntil Startup= 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.
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} **Definition of API class ara::tsync::SynchronizedTime BaseProvider** ⌈

Kind:	class
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	SynchronizedTimeBaseProvider
Syntax:	`class SynchronizedTimeBaseProvider final {...};`
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} **Definition of API function ara::tsync::Synchronized TimeBaseProvider::SynchronizedTimeBaseProvider** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	SynchronizedTimeBaseProvider(const ara::core::InstanceSpecifier &specifier)	
Syntax:	`explicit SynchronizedTimeBaseProvider (const ara::core::Instance Specifier &specifier) noexcept;`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an TimeSynchronization Interface
Exception Safety:	noexcept	
Description:	SynchronizedTimeBaseProvider constructor.	

⌋(*RS_TS_00023*)

[SWS_TS_01102]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::SynchronizedTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01103]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::operator=** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01104]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::SynchronizedTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01105]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::operator=** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01106]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::~SynchronizedTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

8.2.1.2 SetTime

[SWS_TS_01107]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::SetTime** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	SetTime(ara::tsync::Timestamp timePoint, ara::core::Span< const ara::core::Byte > userData={})	
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::kDaemon ConnectionLost	the action cannot be executed, because the connection to time sync daemon is currently lost
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::UpdateTime** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	UpdateTime(ara::tsync::Timestamp, ara::core::Span< const ara::core::Byte > userData={})	
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::kDaemon ConnectionLost	the action cannot be executed, because the connection to time sync daemon is currently lost
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::GetCurrentTime** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	GetCurrentTime()	
Syntax:	`ara::tsync::Timestamp GetCurrentTime () const noexcept;`	
Return value:	ara::tsync::Timestamp	The current time as clock specific time point.
Exception Safety:	noexcept	
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::SetRateCorrection** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	SetRateCorrection(double rateCorrection)	
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:	TsyncErrc::kLimits Exceeded	–
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::GetRateDeviation** ⌈

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

⌋(*RS_TS_00018*)

8.2.1.7 SetUserData

[SWS_TS_01112]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::SetUserData** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Scope:	class ara::tsync::SynchronizedTimeBaseProvider

▽

△

Symbol:	SetUserData(ara::core::Span< const ara::core::Byte > userData)	
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	
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::GetUserData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	GetUserData()	
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	
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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::RegisterTimeValidationNotification** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseProvider	
Symbol:	RegisterTimeValidationNotification(ProviderTimeBaseValidationNotification &timeBaseValidationNotification)	
Syntax:	`void RegisterTimeValidationNotification (ProviderTimeBaseValidationNotification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidationNotification	–
Return value:	None	
Exception Safety:	noexcept	

▽

△

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} **Definition of API function ara::tsync::SynchronizedTimeBaseProvider::UnregisterTimeValidationNotification** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_provider.h"
Scope:	class ara::tsync::SynchronizedTimeBaseProvider
Symbol:	UnregisterTimeValidationNotification()
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
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} **Definition of API class ara::tsync::OffsetTimeBaseProvider** ⌈

Kind:	class
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	OffsetTimeBaseProvider
Syntax:	`class OffsetTimeBaseProvider final {...};`
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::OffsetTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01202]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::OffsetTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01203]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::operator=** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	operator=(OffsetTimeBaseProvider &&stbc)	
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	
Description:	Move assignment operator for OffsetTimeBaseProvider.	

⌋(*RS_TS_00023*)

[SWS_TS_01204]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::OffsetTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01205]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::operator=** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01206]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::~OffsetTimeBaseProvider** ⌈

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

⌋(*RS_TS_00023*)

8.2.2.2 SetOffsetTime

[SWS_TS_01207]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::SetOffsetTime** ⌈

Kind:	function
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Scope:	class ara::tsync::OffsetTimeBaseProvider

▽

△

Symbol:	SetOffsetTime(ara::tsync::Timestamp timePoint, ara::core::Span< const ara::core::Byte > user Data={})	
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::kDaemon ConnectionLost	the action cannot be executed, because the connection to time sync daemon is currently lost
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::GetCurrentTime** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	GetCurrentTime()	
Syntax:	`ara::tsync::Timestamp GetCurrentTime () const noexcept;`	
Return value:	ara::tsync::Timestamp	The current time of the synchronized clock.
Exception Safety:	noexcept	
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::SetRateCorrection** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	SetRateCorrection(double rateCorrection)	
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:	TsyncErrc::kLimits Exceeded	–
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::GetRateDeviation** ⌈

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

⌋(*RS_TS_00018*)

8.2.2.6 SetUserData

[SWS_TS_01211]{DRAFT} **Definition of API function ara::tsync::OffsetTimeBase Provider::SetUserData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	SetUserData(ara::core::Span< const ara::core::Byte > userData)	
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	
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::GetUserData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	GetUserData()	
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	
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::RegisterTimeValidationNotification** ⌈

Kind:	function	
Header file:	#include "ara/tsync/offset_time_base_provider.h"	
Scope:	class ara::tsync::OffsetTimeBaseProvider	
Symbol:	RegisterTimeValidationNotification(ProviderTimeBaseValidationNotification &timeBaseValidationNotification)	
Syntax:	`void RegisterTimeValidationNotification (ProviderTimeBaseValidationNotification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidationNotification	–
Return value:	None	
Exception Safety:	noexcept	
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} **Definition of API function ara::tsync::OffsetTimeBase Provider::UnregisterTimeValidationNotification** ⌈

Kind:	function
Header file:	#include "ara/tsync/offset_time_base_provider.h"
Scope:	class ara::tsync::OffsetTimeBaseProvider
Symbol:	UnregisterTimeValidationNotification()
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
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] Definition of API class ara::tsync::SynchronizedTimeBaseConsumer ⌈

Kind:	class
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	SynchronizedTimeBaseConsumer
Syntax:	`class SynchronizedTimeBaseConsumer final {...};`
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} **Definition of API function ara::tsync::Synchronized TimeBaseConsumer::SynchronizedTimeBaseConsumer** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer

▽

△

Symbol:	SynchronizedTimeBaseConsumer(const ara::core::InstanceSpecifier &specifier)	
Syntax:	`explicit SynchronizedTimeBaseConsumer (const ara::core::InstanceSpecifier &specifier) noexcept;`	
Parameters (in):	specifier	InstanceSpecifier to an PortPrototype of an TimeSynchronizationInterface
Exception Safety:	noexcept	
Description:	SynchronizedTimeBaseConsumer constructor.	

⌋(*RS_TS_00023*)

[SWS_TS_01002]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::~SynchronizedTimeBaseConsumer** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01003]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::SynchronizedTimeBaseConsumer** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01004]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::operator=** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Symbol:	operator=(SynchronizedTimeBaseConsumer &&stbc)
Syntax:	`SynchronizedTimeBaseConsumer & operator= (SynchronizedTimeBaseConsumer &&stbc) & noexcept;`

▽

△

Parameters (in):	stbc	The SynchronizedTimeBaseConsumer object to be moved.
Return value:	SynchronizedTimeBase Consumer &	The moved SynchronizedTimeBaseConsumer object.
Exception Safety:	noexcept	
Thread Safety:	re-entrant	
Description:	Move assignment operator for SynchronizedTimeBaseConsumer. *	

⌋(*RS_TS_00023*)

[SWS_TS_01005]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::SynchronizedTimeBaseConsumer** ⌈

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

⌋(*RS_TS_00023*)

[SWS_TS_01006]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::operator=** ⌈

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

⌋(*RS_TS_00023*)

8.3.1.2 GetCurrentTime

[SWS_TS_01007]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::GetCurrentTime** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Symbol:	GetCurrentTime()
Syntax:	`Timestamp GetCurrentTime () const noexcept;`

▽

△

Return value:	Timestamp	The current time of the synchronized clock.
Exception Safety:	noexcept	
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] Definition of API function ara::tsync::SynchronizedTimeBase Consumer::GetRateDeviation ⌈

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

⌋(*RS_TS_00018*)

8.3.1.4 GetTimeWithStatus

[SWS_TS_01009]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseConsumer::GetTimeWithStatus** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Symbol:	GetTimeWithStatus()	
Syntax:	`SynchronizedTimeBaseStatus GetTimeWithStatus () const noexcept;`	
Return value:	SynchronizedTimeBase Status	A clock specific TimeBaseStatus that contains all the relevant clock information.
Exception Safety:	noexcept	
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] Definition of API function ara::tsync::SynchronizedTimeBase Consumer::RegisterStatusChangeNotifier ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Symbol:	RegisterStatusChangeNotifier(std::function< void(const SynchronizedTimeBaseStatus &)> notifier)	
Syntax:	`void RegisterStatusChangeNotifier (std::function< void(const SynchronizedTimeBaseStatus &)> notifier) noexcept;`	
DIRECTION NOT DEFINED	notifier	–
Return value:	None	
Exception Safety:	noexcept	
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.	

⌋(*RS_TS_00034*)

8.3.1.6 UnregisterStatusChangeNotifier

[SWS_TS_01011]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseConsumer::UnregisterStatusChangeNotifier** ⌈

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

⌋(*RS_TS_00034*)

8.3.1.7 RegisterSynchronizationStateChangeNotifier

[SWS_TS_01012] Definition of API function ara::tsync::SynchronizedTimeBase Consumer::RegisterSynchronizationStateChangeNotifier ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Symbol:	RegisterSynchronizationStateChangeNotifier(std::function< void(const SynchronizationStatus &)> notifier)	
Syntax:	`void RegisterSynchronizationStateChangeNotifier (std::function< void(const SynchronizationStatus &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to unregister.
Return value:	None	
Exception Safety:	noexcept	
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.	

⌋(*RS_TS_00034*)

8.3.1.8 UnregisterSynchronizationStateChangeNotifier

[SWS_TS_01013]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseConsumer::UnregisterSynchronizationStateChangeNotifier** ⌈

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

⌋(*RS_TS_00034*)

8.3.1.9 RegisterTimeLeapNotifier

[SWS_TS_01014] Definition of API function ara::tsync::SynchronizedTimeBase Consumer::RegisterTimeLeapNotifier ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer

▽

△

Symbol:	RegisterTimeLeapNotifier(std::function< void(const SynchronizedTimeBaseStatus &)> notifier)	
Syntax:	`void RegisterTimeLeapNotifier (std::function< void(const SynchronizedTimeBaseStatus &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to be called if the TimeBaseStatus has changed.
Return value:	None	
Exception Safety:	noexcept	
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.	

⌋(*RS_TS_00034*)

8.3.1.10 UnregisterTimeLeapNotifier

[SWS_TS_01015]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::UnregisterTimeLeapNotifier** ⌈

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

⌋(*RS_TS_00034*)

8.3.1.11 RegisterTimeValidationNotification

[SWS_TS_01016]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimeValidationNotification** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Symbol:	RegisterTimeValidationNotification(ConsumerTimeBaseValidationNotification &timeBase ValidationNotification)	
Syntax:	`void RegisterTimeValidationNotification (ConsumerTimeBaseValidation Notification &timeBaseValidationNotification) noexcept;`	
DIRECTION NOT DEFINED	timeBaseValidation Notification	–
Return value:	None	
Exception Safety:	noexcept	

▽

△

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} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::UnregisterTimeValidationNotification** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Symbol:	UnregisterTimeValidationNotification()
Syntax:	`void UnregisterTimeValidationNotification () noexcept;`
Return value:	None
Exception Safety:	noexcept
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] Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::RegisterTimePrecisionMeasurementNotifier ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer	
Symbol:	RegisterTimePrecisionMeasurementNotifier(std::function< void(const TimePrecision Measurement &)> notifier)	
Syntax:	`void RegisterTimePrecisionMeasurementNotifier (std::function< void(const TimePrecisionMeasurement &)> notifier) noexcept;`	
Parameters (in):	notifier	The function to be called.
Return value:	None	
Exception Safety:	noexcept	
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} **Definition of API function ara::tsync::SynchronizedTimeBaseConsumer::UnregisterTimePrecisionMeasurementNotifier** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_consumer.h"
Scope:	class ara::tsync::SynchronizedTimeBaseConsumer
Symbol:	UnregisterTimePrecisionMeasurementNotifier()
Syntax:	`void UnregisterTimePrecisionMeasurementNotifier () noexcept;`
Return value:	None
Exception Safety:	noexcept
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] Definition of API class ara::tsync::SynchronizedTimeBaseStatus ⌈

Kind:	class
Header file:	#include "ara/tsync/synchronized_time_base_status.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	SynchronizedTimeBaseStatus
Syntax:	`class SynchronizedTimeBaseStatus final {...};`
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} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::SynchronizedTimeBaseStatus** ⌈

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

⌋(*RS_TS_00021*)

[SWS_TS_01058]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::SynchronizedTimeBaseStatus** ⌈

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

⌋(*RS_TS_00021*)

[SWS_TS_01059]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::operator=** ⌈

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

⌋(*RS_TS_00021*)

[SWS_TS_01060]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::operator=** ⌈

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

⌋(*RS_TS_00021*)

[SWS_TS_01061]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::SynchronizedTimeBaseStatus** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_status.h"
Scope:	class ara::tsync::SynchronizedTimeBaseStatus
Symbol:	SynchronizedTimeBaseStatus()
Syntax:	`SynchronizedTimeBaseStatus ()=delete;`
Description:	Constructor of SynchronizedTimeBaseStatus cannot be used.

⌋()

[SWS_TS_01062]{DRAFT} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::~SynchronizedTimeBaseStatus** ⌈

Kind:	function
Header file:	#include "ara/tsync/synchronized_time_base_status.h"
Scope:	class ara::tsync::SynchronizedTimeBaseStatus
Symbol:	~SynchronizedTimeBaseStatus()
Syntax:	`~SynchronizedTimeBaseStatus ()=default;`
Description:	Destructor of SynchronizedTimeBaseStatus.

⌋()

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

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

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

8.3.2.2 GetCreationTime

[SWS_TS_01055]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseStatus::GetCreationTime** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Symbol:	GetCreationTime()	
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	
Description:	A method to obtain the creation time of this object.	

⌋(*RS_TS_00021*)

8.3.2.3 GetSynchronizationStatus

[SWS_TS_01053]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseStatus::GetSynchronizationStatus** ⌈

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

⌋(*RS_TS_00021*)

8.3.2.4 GetLeapJump

[SWS_TS_01054]{DRAFT} **Definition of API function ara::tsync::Synchronized TimeBaseStatus::GetLeapJump** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Symbol:	GetLeapJump()	
Syntax:	`LeapJump GetLeapJump () const noexcept;`	
Return value:	LeapJump	LeapJump

▽

△

Exception Safety:	noexcept
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} **Definition of API function ara::tsync::SynchronizedTimeBaseStatus::GetUserData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/synchronized_time_base_status.h"	
Scope:	class ara::tsync::SynchronizedTimeBaseStatus	
Symbol:	GetUserData()	
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	
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 Errors

The `Time Synchronization` cluster implements an error handling based on `ara::core::Result`. The errors supported by the `Time Synchronization` cluster are listed in section 8.4.1.

8.4.1 Time Synchronization error codes

[SWS_TS_00901]{DRAFT} **Definition of API enum ara::tsync::TsyncErrc** ⌈

Kind:	enumeration
Header file:	#include "ara/tsync/tsync_error_domain.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"

▽

△

Scope:	namespace ara::tsync	
Symbol:	TsyncErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class TsyncErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kDaemonConnection Lost= 1	The action cannot be executed, because the connection to time sync daemon is currently lost.
	kLimitsExceeded= 2	The action is not allowed because rate correction limit is exceeded.
Description:	Defines an enumeration class for the Time Synchronization error codes.	

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*, *RS_AP_00154*)

8.4.2 TsyncException class

[SWS_TS_00902]{DRAFT} **Definition of API class ara::tsync::TsyncException** ⌈

Kind:	class
Header file:	#include "ara/tsync/tsync_error_domain.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	TsyncException
Base class:	ara::core::Exception
Syntax:	`class TsyncException : public ara::core::Exception {...};`
Description:	Defines a class for exceptions to be thrown by the Time Synchronization.

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*, *RS_AP_00154*, *RS_AP_00150*)

8.4.2.1 TsyncException::TsyncException

[SWS_TS_00903]{DRAFT} **Definition of API function ara::tsync::TsyncException::TsyncException** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	class ara::tsync::TsyncException	
Symbol:	TsyncException(ara::core::ErrorCode errorCode)	
Syntax:	`explicit TsyncException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	noexcept	
Description:	Constructs a new TsyncException object containing an error code.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*, *RS_AP_00132*)

8.4.3 GetTsyncErrorDomain function

[SWS_TS_00909]{DRAFT} **Definition of API function ara::tsync::GetTsyncErrorDomain** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	namespace ara::tsync	
Symbol:	GetTsyncErrorDomain()	
Syntax:	`const ara::core::ErrorDomain & GetTsyncErrorDomain () noexcept;`	
Return value:	const ara::core::ErrorDomain &	Return a reference to the global TsyncErrorDomain object.
Exception Safety:	noexcept	
Description:	Returns a reference to the global TsyncErrorDomain object.	

⌋(*RS_AP_00120*, *RS_AP_00130*, *RS_AP_00132*, *RS_AP_00154*)

8.4.4 MakeErrorCode function

[SWS_TS_00910]{DRAFT} **Definition of API function ara::tsync::MakeErrorCode** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	namespace ara::tsync	
Symbol:	MakeErrorCode(ara::tsync::TsyncErrc code, ara::core::ErrorDomain::SupportDataType data)	
Syntax:	`ara::core::ErrorCode MakeErrorCode (ara::tsync::TsyncErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error.
Return value:	ara::core::ErrorCode	An ErrorCode object.
Exception Safety:	noexcept	
Description:	Creates an instance of ErrorCode.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*, *RS_AP_00132*, *RS_AP_00154*)

8.4.5 TsyncErrorDomain class

The error handling requires an `ara::core::ErrorDomain`, which can be used to check the errors returned via `ara::core::Result`.

[SWS_TS_00904]{DRAFT} **Definition of API class ara::tsync::TsyncErrorDomain** ⌈

Kind:	class
Header file:	#include "ara/tsync/tsync_error_domain.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	TsyncErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class TsyncErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'0901
Description:	Defines a class representing the Time Synchronization error domain.

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*, *RS_AP_00154*, *RS_AP_00150*)

8.4.5.1 TsyncErrorDomain::TsyncErrorDomain

[SWS_TS_00905]{DRAFT} **Definition of API function ara::tsync::TsyncErrorDomain::TsyncErrorDomain** ⌈

Kind:	function
Header file:	#include "ara/tsync/tsync_error_domain.h"
Scope:	class ara::tsync::TsyncErrorDomain
Symbol:	TsyncErrorDomain()
Syntax:	`TsyncErrorDomain () noexcept;`
Exception Safety:	noexcept
Description:	Constructs a new TsyncErrorDomain object.

⌋(*RS_AP_00120*, *RS_AP_00130*, *RS_AP_00132*)

8.4.5.2 TsyncErrorDomain::Name

[SWS_TS_00906]{DRAFT} **Definition of API function ara::tsync::TsyncErrorDomain::Name** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	class ara::tsync::TsyncErrorDomain	
Symbol:	Name()	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Tsync"
Exception Safety:	noexcept	
Description:	Returns a string constant associated with TsyncErrorDomain.	

⌋(*RS_AP_00120*, *RS_AP_00130*, *RS_AP_00132*)

8.4.5.3 TsyncErrorDomain::Message

[SWS_TS_00907]{DRAFT} **Definition of API function ara::tsync::TsyncErrorDomain::Message** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	class ara::tsync::TsyncErrorDomain	
Symbol:	Message(CodeType errorCode)	
Syntax:	`const char * Message (CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	noexcept	
Description:	Returns the message associated with errorCode.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*, *RS_AP_00132*)

8.4.5.4 TsyncErrorDomain::ThrowAsException

[SWS_TS_00908]{DRAFT} **Definition of API function ara::tsync::TsyncErrorDomain::ThrowAsException** ⌈

Kind:	function	
Header file:	#include "ara/tsync/tsync_error_domain.h"	
Scope:	class ara::tsync::TsyncErrorDomain	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	noexcept(false)	
Description:	Creates a new instance of TsyncException from errorCode and throws it as a C++ exception.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*)

8.5 C++ Time Validation Interface

8.5.1 Type definitions

8.5.1.1 TimeMasterMeasurementType

[SWS_TS_00414]{DRAFT} **Definition of API class ara::tsync::TimeMasterMeasurementType** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14140]{DRAFT} **Definition of API variable ara::tsync::TimeMasterMeasurementType::preciseOriginTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14141]{DRAFT} **Definition of API variable ara::tsync::TimeMasterMeasurementType::syncEgressTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14142]{DRAFT} **Definition of API variable ara::tsync::TimeMasterMeasurementType::sequenceId** ⌈

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

⌋*(RS_TS_00034)*

8.5.1.2 TimeSlaveMeasurementType

[SWS_TS_00415]{DRAFT} **Definition of API class ara::tsync::TimeSlaveMeasurementType** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14150]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::preciseOriginTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14151]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::referenceGlobalTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14152]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::syncIngressTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14153]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::correctionField** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14154]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::referenceLocalTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14155]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::pDelay** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14156]{DRAFT} **Definition of API variable ara::tsync::TimeSlaveMeasurementType::sequenceId** ⌈

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

⌋(*RS_TS_00034*)

8.5.1.3 PdelayInitiatorMeasurementType

[SWS_TS_00416]{DRAFT} **Definition of API class ara::tsync::PdelayInitiatorMeasurementType** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14160]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::requestOriginTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14161]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::responseReceiptTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14162]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::requestReceiptTimestamp** ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_14163]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::responseOriginTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14164]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::referenceLocalTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14165]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::referenceGlobalTimestamp** ⌈

Kind:	variable
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Symbol:	referenceGlobalTimestamp
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp referenceGlobalTimestamp;`
Description:	value of the local instance of the Global Time of the reference Global Time Tuple

⌋(*RS_TS_00034*)

[SWS_TS_14166]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::pDelay** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14167]{DRAFT} **Definition of API variable ara::tsync::PdelayInitiator MeasurementType::sequenceId** ⌈

Kind:	variable
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Scope:	struct ara::tsync::PdelayInitiatorMeasurementType
Symbol:	sequenceId
Type:	std::uint16_t

▽

△

Syntax:	`std::uint16_t sequenceId;`
Description:	sequence Id of sent Pdelay_Req frame

⌋(*RS_TS_00034*)

8.5.1.4 PdelayResponderMeasurementType

[SWS_TS_00417]{DRAFT} **Definition of API class ara::tsync::PdelayResponder MeasurementType** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14170]{DRAFT} **Definition of API variable ara::tsync::PdelayResponder MeasurementType::requestReceiptTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14171]{DRAFT} **Definition of API variable ara::tsync::PdelayResponder MeasurementType::responseOriginTimestamp** ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_14172]{DRAFT} **Definition of API variable ara::tsync::PdelayResponderMeasurementType::referenceLocalTimestamp** ⌈

Kind:	variable
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Symbol:	referenceLocalTimestamp
Type:	std::uint64_t
Syntax:	`std::uint64_t referenceLocalTimestamp;`
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} **Definition of API variable ara::tsync::PdelayResponderMeasurementType::referenceGlobalTimestamp** ⌈

Kind:	variable
Header file:	#include "ara/tsync/time_validation_measurement_types.h"
Scope:	struct ara::tsync::PdelayResponderMeasurementType
Symbol:	referenceGlobalTimestamp
Type:	ara::tsync::Timestamp
Syntax:	`ara::tsync::Timestamp referenceGlobalTimestamp;`
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} **Definition of API variable ara::tsync::PdelayResponderMeasurementType::sequenceId** ⌈

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

⌋(*RS_TS_00034*)

8.5.2 Provider TimeBase Validation Notification

[SWS_TS_00419]{DRAFT} **Definition of API class ara::tsync::ProviderTimeBase ValidationNotification** ⌈

Kind:	class
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	ProviderTimeBaseValidationNotification
Syntax:	`class ProviderTimeBaseValidationNotification {...};`
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*)

[SWS_TS_01301]{DRAFT} **Definition of API function ara::tsync::ProviderTime BaseValidationNotification::~ProviderTimeBaseValidationNotification** ⌈

Kind:	function
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"
Scope:	class ara::tsync::ProviderTimeBaseValidationNotification
Symbol:	~ProviderTimeBaseValidationNotification()
Syntax:	`virtual ~ProviderTimeBaseValidationNotification ()=default;`
Description:	Destructor .

⌋(*RS_TS_00034*, *RS_TS_00030*)

8.5.2.1 SetPdelayResponderData

[SWS_TS_00423]{DRAFT} **Definition of API function ara::tsync::ProviderTime BaseValidationNotification::SetPdelayResponderData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"	
Scope:	class ara::tsync::ProviderTimeBaseValidationNotification	
Symbol:	SetPdelayResponderData(const PdelayResponderMeasurementType &measurementData)	
Syntax:	`virtual void SetPdelayResponderData (const PdelayResponderMeasurement Type &measurementData)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Responder
Return value:	None	
Exception Safety:	not exception safe	
Description:	Provide the recorded data block for the pPelay Responder of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

8.5.2.2 SetMasterTimingData

[SWS_TS_00421]{DRAFT} **Definition of API function ara::tsync::ProviderTimeBaseValidationNotification::SetMasterTimingData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/provider_time_base_validation_notification.h"	
Scope:	class ara::tsync::ProviderTimeBaseValidationNotification	
Symbol:	SetMasterTimingData(const TimeMasterMeasurementType &measurementData)	
Syntax:	`virtual void SetMasterTimingData (const TimeMasterMeasurementType &measurementData)=0;`	
Parameters (in):	measurementData	Detailed data for validation of the Time Master
Return value:	None	
Exception Safety:	not exception safe	
Description:	Provide the recorded data block for the Time Master of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00029*)

8.5.3 Consumer TimeBase Provider Notification

[SWS_TS_00428]{DRAFT} **Definition of API class ara::tsync::ConsumerTimeBaseValidationNotification** ⌈

Kind:	class
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"
Forwarding header file:	#include "ara/tsync/tsync_fwd.h"
Scope:	namespace ara::tsync
Symbol:	ConsumerTimeBaseValidationNotification
Syntax:	`class ConsumerTimeBaseValidationNotification {...};`
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*)

[SWS_TS_01300]{DRAFT} **Definition of API function ara::tsync::ConsumerTimeBaseValidationNotification::~ConsumerTimeBaseValidationNotification** ⌈

Kind:	function
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"
Scope:	class ara::tsync::ConsumerTimeBaseValidationNotification
Symbol:	~ConsumerTimeBaseValidationNotification()
Syntax:	`virtual ~ConsumerTimeBaseValidationNotification ()=default;`
Description:	Destructor .

⌋(*RS_TS_00034*, *RS_TS_00030*)

8.5.3.1 SetPdelayInitiatorData

[SWS_TS_00422]{DRAFT} **Definition of API function ara::tsync::ConsumerTimeBaseValidationNotification::SetPdelayInitiatorData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"	
Scope:	class ara::tsync::ConsumerTimeBaseValidationNotification	
Symbol:	SetPdelayInitiatorData(const PdelayInitiatorMeasurementType &measurementData)	
Syntax:	`virtual void SetPdelayInitiatorData (const PdelayInitiatorMeasurementType &measurementData)=0;`	
Parameters (in):	measurementData	Detailed timing data for the pDelay Initiator
Return value:	None	
Exception Safety:	not exception safe	
Description:	Provide the recorded data block for the pPelay Initiator of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00030*)

8.5.3.2 SetSlaveTimingData

[SWS_TS_00420]{DRAFT} **Definition of API function ara::tsync::ConsumerTimeBaseValidationNotification::SetSlaveTimingData** ⌈

Kind:	function	
Header file:	#include "ara/tsync/consumer_time_base_validation_notification.h"	
Scope:	class ara::tsync::ConsumerTimeBaseValidationNotification	
Symbol:	SetSlaveTimingData(const TimeSlaveMeasurementType &measurementData)	
Syntax:	`virtual void SetSlaveTimingData (const TimeSlaveMeasurementType &measurementData)=0;`	
Parameters (in):	measurementData	Detailed data for validation of the Time Slave
Return value:	None	
Exception Safety:	not exception safe	
Description:	Provide the recorded data block for the Time Slave of the Time Base.	

⌋(*RS_TS_00034*, *RS_TS_00030*)

8.6 C++ Time Precision Interface

8.6.1 Type definitions

8.6.1.1 TimePrecisionMeasurement type

[SWS_TS_01400] Definition of API class ara::tsync::TimePrecisionMeasurement ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01401] Definition of API variable ara::tsync::TimePrecisionMeasurement::glbSeconds ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01402] Definition of API variable ara::tsync::TimePrecisionMeasurement::glbNanoSeconds ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01403]{DRAFT} **Definition of API variable ara::tsync::TimePrecision Measurement::timeBaseStatus** ⌈

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

⌋*(RS_TS_00034, RS_TS_00021)*

[SWS_TS_01404] Definition of API variable ara::tsync::TimePrecisionMeasurement::virtualLocalTimeLow ⌈

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

⌋*(RS_TS_00034)*

[SWS_TS_01405] Definition of API variable ara::tsync::TimePrecisionMeasurement::rateDeviation ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01406] Definition of API variable ara::tsync::TimePrecisionMeasurement::locSeconds ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01407] Definition of API variable ara::tsync::TimePrecisionMeasurement::locNanoSeconds ⌈

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

⌋(*RS_TS_00034*)

[SWS_TS_01408] Definition of API variable ara::tsync::TimePrecisionMeasurement::pathDelay ⌈

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

⌋(*RS_TS_00034*)

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.			
Base	*ARObject*			
Aggregated by	SynchronizedTimeBaseProvider.timeSyncCorrection			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allowProvider RateCorrection	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 setRate Correction(). true: rate correction can be set by method setRate Correction().
offsetCorrection AdaptionInterval	TimeValue	0..1	attr	Defines the interval during which the adaptive rate correction cancels out the rate and time deviation. Unit: seconds.
offsetCorrection JumpThreshold	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.
rateCorrections Per Measurement Duration	PositiveInteger	0..1	attr	Number of simultaneous rate measurements to determine the current rate deviation.
rateDeviation Measurement Duration	TimeValue	0..1	attr	Time span used to calculate the rate deviation. Unit: seconds.

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.recommendedPackage=FCInteractions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
cyclicBackup Interval	TimeValue	0..1	attr	Time interval in seconds to store the time base value periodically to persistence.
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.
timeBase Provider	SynchronizedTimeBase Provider	0..1	ref	This reference represents the mapped TimeBase Provider.

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	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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	0..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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=gateway.shortName, gateway.variation Point.shortLabel 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeDomainProperty, globalTime DomainProperty.variationPoint.shortLabel 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeMaster.shortName, globalTime Master.variationPoint.shortLabel 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeSubDomain.globalTimeDomain, globalTimeSubDomain.variationPoint.shortLabel 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=pduTriggering.pduTriggering, pdu Triggering.variationPoint.shortLabel 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:** atpSplitable; atpVariation **Tags:** atp.Splitkey=slave.shortName, slave.variationPoint.short Label 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			
Aggregated by	GlobalTimeDomain.globalTimeMaster			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
communication Connector	Communication Connector	0..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	0..1	attr	If set to TRUE, the GlobalTimeMaster is supposed to act as the root of global time information.
syncPeriod	TimeValue	0..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			
Aggregated by	GlobalTimeDomain.slave			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
communication Connector	Communication Connector	0..1	ref	The GlobalTimeSlave is bound to the Communication Connector.

▽

△

Class	***GlobalTimeSlave*** (abstract)			
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.
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

No content.

C Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

C.1 Traceable item history of this document according to AUTOSAR Release R23-11

C.1.1 Added Specification Items in R23-11

[SWS_TS_00901] [SWS_TS_00902] [SWS_TS_00903] [SWS_TS_00904] [SWS_TS_00905] [SWS_TS_00906] [SWS_TS_00907] [SWS_TS_00908] [SWS_TS_00909] [SWS_TS_00910] [SWS_TS_01061] [SWS_TS_01062] [SWS_TS_01204] [SWS_TS_01300] [SWS_TS_01301] [SWS_TS_14175]

C.1.2 Changed Specification Items in R23-11

[SWS_TS_00042] [SWS_TS_00047] [SWS_TS_00049] [SWS_TS_00052] [SWS_TS_00058] [SWS_TS_00064] [SWS_TS_00070] [SWS_TS_00071] [SWS_TS_00202] [SWS_TS_00420] [SWS_TS_00421] [SWS_TS_00422] [SWS_TS_00423] [SWS_TS_00803] [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_01107] [SWS_TS_01110] [SWS_TS_01201] [SWS_TS_01202] [SWS_TS_01203] [SWS_TS_01205] [SWS_TS_01206] [SWS_TS_01207] [SWS_TS_01209]

C.1.3 Deleted Specification Items in R23-11

none