Document Title	Time Synchronization over CAN Protocol Specification
Document Category	Protocol specification (PRS)
Document UID	1089
Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and overview 6
1.1 Protocol purpose and objectives . . . 10
1.2 Applicability of the protocol . . . 10
1.2.1 Constraints and assumptions . . . 10
1.2.2 Limitations . . . 10
1.3 Dependencies . . . 10
1.3.1 Dependencies to other protocol layers . . . 10
1.3.2 Dependencies to other standards and norms . . . 11
1.3.3 Dependencies to the Application Layer . . . 11
2 Use Cases 12
3 Related documentation 13
3.1 Input documents & related standards and norms . . . 13
3.2 Related specification . . . 13
4 Protocol Requirements 14
4.1 Requirements Traceability . . . 14
5 Definition of terms and acronyms 17
5.1 Acronyms and abbreviations . . . 17
5.2 Definition of terms . . . 18
5.3 Inputs to the Time Synchronization protocol . . . 18
5.4 Outputs of the Time Synchronization protocol . . . 19
5.5 Timing Values used by the Time Synchronization protocol . . . 19
6 Protocol specification 21
6.1 Message types . . . 21
6.2 Message format . . . 21
6.2.1 SYNC Message Format . . . 23
6.2.2 Follow-up (FUP) Message Format . . . 23
6.2.3 Data Types . . . 25
6.3 Services / Commands . . . 26
6.4 Sequences (lower layer) . . . 26
6.4.1 General Behavior of the Time Synchronization Protocol . . . 26
6.4.1.1 CRC Protection of Time Synchronization Messages . . . 27
6.4.1.2 ICV Calculation of Time Synchronization Messages . . . 28
6.4.1.3 User Data . . . 29
6.4.2 Time Master . . . 29
6.4.2.1 State TM_Init . . . 30
6.4.2.1.1 Transition from `TM_Init` . . . 31
6.4.2.2 State TM_Idle . . . 31
6.4.2.2.1 Transition from `TM_Idle` . . . 31
6.4.2.3 State TM_BuildSyncMsg . . . 32

6.4.2.3.1 Set SYNC Message Type Field (*Type*) . . . 33
6.4.2.3.2 Set SYNC Message Domain Id Field (*D*) . . . 34
6.4.2.3.3 Set SYNC Message Sequence Counter Field (*SC*) . . . 34
6.4.2.3.4 Get Current Local Time and User/Status Data . . . 34
6.4.2.3.4.1 Set SYNC Message Seconds Field (*SyncTimeSec*) . . . 35
6.4.2.3.4.2 Set SYNC Message User Data Fields (*UserData_0/_1*) 35
6.4.2.3.5 Set SYNC Message Checksum Field (*CRC*) . . . 36
6.4.2.3.6 Transition from `TM_BuildSyncMsg` . . . 36
6.4.2.4 State TM_TransmitSyncMsg . . . 37
6.4.2.4.1 Transition from `TM_TransmitSyncMsg` . . . 38
6.4.2.5 State TM_DebounceBeforeFupMsg . . . 39
6.4.2.5.1 Transition from `TM_DebounceBeforeFupMsg` . . . 39
6.4.2.6 State TM_BuildFupMsg . . . 39
6.4.2.6.1 Set FUP Message Type Field (*Type*) . . . 41
6.4.2.6.2 Set FUP Message Domain Id Field (*D*) . . . 41
6.4.2.6.3 Set FUP Message Sequence Counter Field (*SC*) . . . 42
6.4.2.6.4 Set FUP Message Nanoseconds Field (*SyncTimeNSec* and *OVS*) . . . 42
6.4.2.6.5 Set FUP Message Time Gateway Synchronization Status Field (*SGW*) . . . 43
6.4.2.6.6 Set FUP Message User Data Field (*UserData_2*) . . . 43
6.4.2.6.7 Set FUP Message Checksum Field (*CRC*) . . . 43
6.4.2.6.8 Set Freshness Value Field (*FV* and *FVL*) . . . 44
6.4.2.6.9 Set Integrity Check Value Fields (*ICV* and *ICVL*) . . . 45
6.4.2.6.10 Transition from `TM_BuildFupMsg` . . . 47
6.4.2.7 State TM_TransmitFupMsg . . . 48
6.4.2.7.1 Transition from `TM_TransmitFupMsg` . . . 48
6.4.2.8 State TM_DebounceBeforeSyncMsg . . . 48
6.4.2.8.1 Transition from `TM_DebounceBeforeSyncMsg` . . . 48
6.4.3 Time Slave . . . 48
6.4.3.1 State TS_Init . . . 50
6.4.3.1.1 Transition from `TS_Init` . . . 50
6.4.3.2 State TS_Idle . . . 50
6.4.3.2.1 Transition from `TS_Idle` . . . 50
6.4.3.3 State TS ReceiveSyncMsg . . . 51
6.4.3.3.1 Check SYNC Message Type Field (*Type*) . . . 52
6.4.3.3.2 Check SYNC Message Domain Id Field (*D*) . . . 52
6.4.3.3.3 Check SYNC Message Sequence Counter Field (*SC*) . . . 52
6.4.3.3.3.1 Sequence Counter Hysteresis . . . 54
6.4.3.3.4 Check SYNC Message CRC Field (*CRC*) . . . 56
6.4.3.3.5 Get SYNC Message Seconds Field (`SyncTimeSec`) . . . 57
6.4.3.3.6 Get SYNC Message User Data Fields (`UserData_0/_1`) . 58
6.4.3.3.7 Transition from `TS_ReceiveSyncMsg` . . . 58
6.4.3.4 State TS_DebounceBeforeFupMsg . . . 59

6.4.3.5 State TS_WaitForFupMsg 60
6.4.3.6 State TS_ReceiveFupMsg 60
6.4.3.6.1 Check FUP Message Type Field (*Type*) 61
6.4.3.6.2 Check FUP Message Domain Id Field (*D*) 62
6.4.3.6.3 Check FUP Message Sequence Counter Field (*SC*) 62
6.4.3.6.4 Check FUP Message Nanoseconds Overflow Field (*OVS*) . 63
6.4.3.6.5 Check FUP Message CRC Field (*CRC*) 63
6.4.3.6.6 ICV Secured FUP Messages 64
6.4.3.6.6.1 Check Freshness Value Fields (*FV* and *FVL*) 65
6.4.3.6.6.2 Check ICV Value Fields (*ICV* and *ICVL*) 67
6.4.3.6.7 Update of Time Slave 70
6.4.3.6.7.1 Update Local Time (from *SyncTimeNSec* and *OVS*) Fields 71
6.4.3.6.7.2 Update User Data (from *UserData_2*) Field 71
6.4.3.6.7.3 Update Time Gateway Synchronization Status (from *SGW*) Field 72
6.4.3.6.8 Transition from `TS_ReceiveFupMsg` 72
6.4.3.7 State TS_DebounceBeforeSyncMsg 73
6.5 Error messages 73

7 Configuration Parameters 74
7.1 Configuration Parameters of a Time Domain 74
7.2 Configuration Parameters of the Time Master of a Time Domain 74
7.3 Configuration Parameters of a Time Slave of a Time Domain 75

8 Protocol usage and guidelines 77

A Change history of AUTOSAR traceable items 78
A.1 Traceable item history of this document according to AUTOSAR Release R26-11 78
A.1.1 Added Specification Items in R26-11 78
A.1.2 Changed Specification Items in R26-11 78
A.1.3 Deleted Specification Items in R26-11 78
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 78
A.2.1 Added Specification Items in R25-11 78
A.2.2 Changed Specification Items in R25-11 79
A.2.3 Deleted Specification Items in R25-11 79

1 Introduction and overview

This protocol specification specifies the message format, sequences and semantics of the AUTOSAR Time Synchronization Protocol over CAN.

The defined Time Synchronization Protocol messages are contained as regular payload of the CAN frames.

The protocol specifies a two-step mechanism:

1. In a first step (SYNC message transmission):
   (a) The transmitting CAN node, i.e. the Time Master, transmits in a first broadcast message, the so-called SYNC message, only the seconds portion $T_{TMGlobalTime_Sec}$ of its current local time $T_{TMGlobalTime}$ (this is due to payload length limitations).

   The Time Master, uses CAN low-level mechanisms like the "CAN transmit confirmation" or HW timestamping to detect the point in time $T_{SyncEgress}$ when the message was actually transmitted, i.e. it takes a timestamp of the SYNC message egress.

   (b) The receiving CAN node, i.e. the Time Slave, samples the timestamp of the ingress of the SYNC message ($T_{SyncIngress}$) and reads the time information $T_{TMGlobalTime_Sec}$ contained in the SYNC message.
2. In a second step (Follow-Up (FUP) message transmission):
   (a) The transmitting CAN node sends a second broadcast message, the so-called Follow-Up (FUP) message. Within that message the Time Master transmits the offset $\Delta t_{SyncTxOffset}$ between
   - the time information transmitted in the SYNC message $T_{TMGlobalTime_Sec}$
   - and the timestamp of the SYNC message egress $T_{SyncEgress}$, i.e., when the SYNC message was actually put on the bus.

   (b) Upon reception of the FUP message the receiving CAN node, i.e., the Time Slave, calculates its new synchronized time $t_{TSGlobalTime}$ by adding
   - the Time Master's time $T_{TMGlobalTime}$ received in the two messages, i.e.,
     - the timestamp within the SYNC message $T_{TMGlobalTime_Sec}$
     - and the time offset within the FUP message $\Delta t_{SyncTxOffset}$
   - and the time $\Delta t_{LinkDelay}$ that the SYNC message needs to travel from the Time Master to the Time Slave
   - and the offset $\Delta t_{SlaveRxOffset}$ that has elapsed from ingress of the received SYNC message ($T_{SyncIngress}$) to the current point in time $T_{TSLocalTime_ReSync}$, i.e., when the Time Slave's time is actually synchronized.

Time synchronization using this two-step mechanism is more precise than by receiving just one message and omitting the egress and ingress timestamps. Highest precision of the time synchronization can be achieved, if hardware timestamping for SYNC message ingress $T_{SyncIngress}$ and egress $T_{SyncEgress}$ is supported and used.

Figure 1.1 shows the CAN Time Synchronization mechanism.

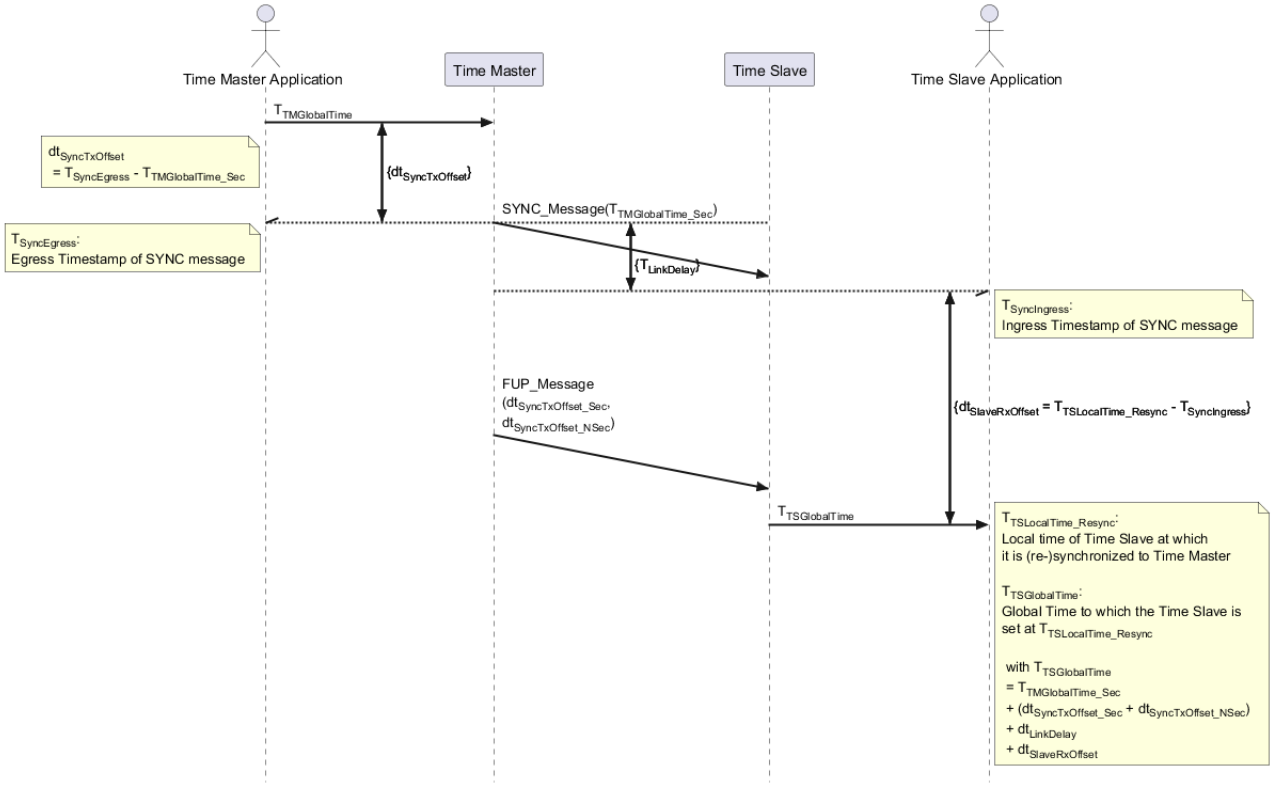

Figure 1.1: CAN Time Synchronization Protocol Overview

Figure 1.2 sketches the bigger picture on network level. The Time Synchronization Protocol over CAN is defined per `Time Domain`. Each Time Domain is identified by a Time Domain Identifier and represents a time source.

The figure illustrates an example with only one Time Domain per CAN bus. However, the Time Synchronization Protocol over CAN may support on each CAN bus multiple Time Domains, i.e., on each CAN bus there can be multiple Time Masters but only one per Time Domain. Each Time Domain has at least one Time Slave. The figure shows, that Time Domains may be chained. In such a chain a Time Gateway forwards time information from one CAN bus / Time Domain to another one. So, the Time Gateway takes the role of a Time Slave on one CAN bus and the role of a Time Master on the other CAN bus. The Time Master at the beginning of such a chain is called System-wide (or Global) Time Master for a Time Domain.

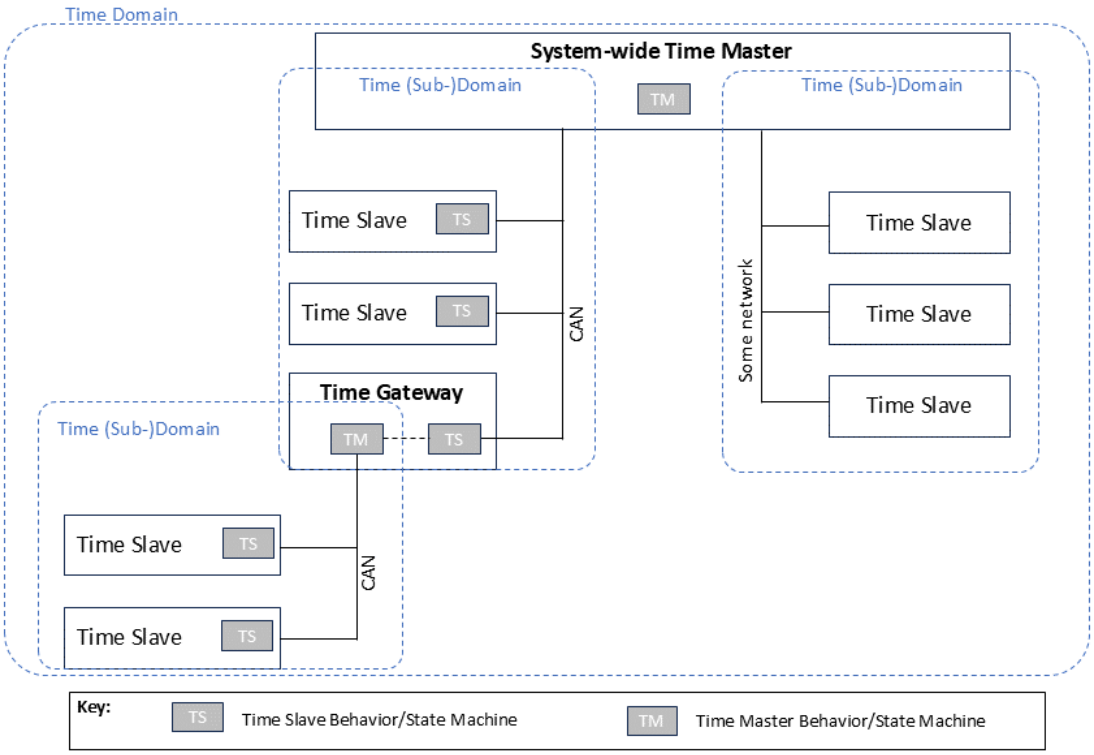

Figure 1.2: Network Topology Example

Figure 1.3 shows details of Local Time Management of Time Master or Time Slave, i.e., how to derive a Synchronized Local Time from the information delivered by an external Time Master. Only difference is, that on Time Master side the Reference Time is provided by some external application (e.g. GNSS), while on Time Slave side the Reference Time is provided by the Time Master via the Time Synchronization Protocol. Besides the Time Synchronization Protocol which delivers timestamps from the Time Master the following functions are required:

- Ingress/Egress Time Stamping: to derive the exact time, when a Time Synchronization message is sent on or received from the bus, respectively
- Time Correction: measures and corrects the frequency and phase offset between the Time Master time and the local Clock
- Time Progression: derives and provides a Local Time that is synchronized to the Time Master by applying correction parameters to the time read from the local Clock
- Local Clock: Provides a uncorrected local clock value.

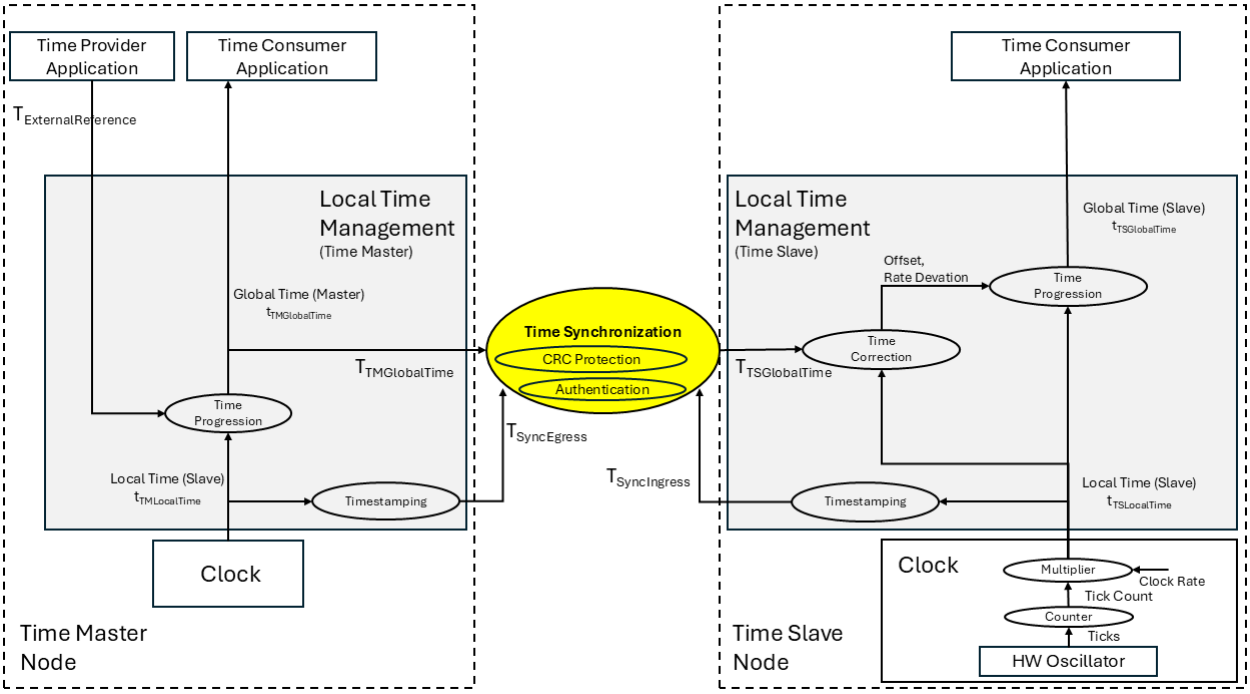

Figure 1.3: Functional Architecture of the Time Synchronization over CAN

Finally, Figure 1.4 shows the Time Synchronization over CAN in context of other relevant logical building blocks - and with that the (logical) inputs from and outputs to the application or external entities.

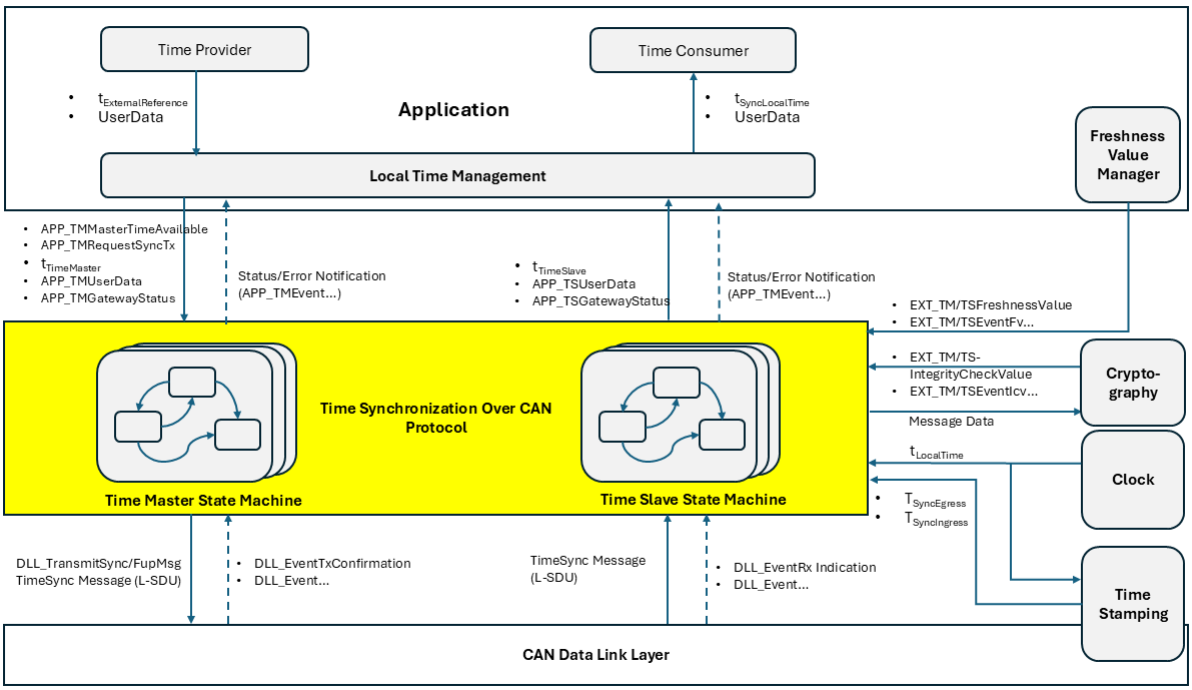

Figure 1.4: Logical Architecture of the Time Synchronization over CAN

1.1 Protocol purpose and objectives

Just transmitting the time information from a Time Master to the Time Slaves in a broadcast CAN message has the disadvantage that the time value becomes inaccurate due to CAN specific effects like arbitration and SW related delays. The two-step mechanism of the AUTOSAR Time Synchronization Protocol on CAN significantly reduces those inaccuracies.

In addition, the AUTOSAR Time Synchronization Protocol on CAN provides security and safety mechanisms to protect the time synchronization between Time Master and Time Slave from accidentally or intentionally corrupted on the CAN bus.

1.2 Applicability of the protocol

1.2.1 Constraints and assumptions

- The Time Synchronization on CAN protocol assumes a local reference clock with a maximum tick duration of 2μs (i.e., with a minimum frequency of 500 kHz).

1.2.2 Limitations

- The Time Synchronization on CAN protocol supports a maximum time value of 4294967295 seconds (2^{32}-1). This is because the time value field in the SYNC messages is limited to 32 bit.
- The Time Synchronization on CAN protocol does not support sharing of PDUs across domains and busses, i.e., the same PDU should not be used for different time domains.

1.3 Dependencies

1.3.1 Dependencies to other protocol layers

This Time Synchronization Protocol can be used on all CAN data link layers as defined in [1, ISO 11898-1:2024], i.e.,

- CAN Classic
- CAN FD
- and CAN XL.

1.3.2 Dependencies to other standards and norms

[1, ISO 11898-1:2024]

1.3.3 Dependencies to the Application Layer

The Time Synchronization over CAN protocol relies on the existence of a Local Time Management on top of the protocol, which provides/consumes the timestamps delivered by the protocol. The Local Time Management is responsible for the following tasks:

- on Time Master side
  - maintains a Global Time $t_{TMGlobalTime}$ driven by a local Clock
  - and distributes (cyclically or event driven) timestamps $T_{TMGlobalTime}$ of the Local Time to connected Time Slaves via the Time Synchronization Protocol
- on Time Slave side
  - adjusts its current Global Time $t_{TSGlobalTime}$ based on timestamps $T_{TSGlobalTime}$ received from the Time Master via the Time Synchronization Protocol
  - and extrapolates/progresses its Global Time $t_{TSGlobalTime}$ based on its local Clock between two timestamps $T_{TSGlobalTime(i)}$ and $T_{TSGlobalTime(i+1)}$ received from the Time Master.
- on both sides
  - and evaluates error states

In addition, the application may relay the time information received for one Time Domain on one CAN bus to another Time Domanin on another CAN bus and which manages the corresponding status information accordingly.

2 Use Cases

Use Case ID	Use Case Name	Description
1	Distributing a reference time on a CAN bus	A CAN node configured as a `Time Master` for a `Time Domain` broadcasts its `Local Time` to other CAN nodes configured as `Time Slave`s linked to that `Time Domain`.
2	Synchronizing to a reference time on a CAN bus	A CAN node configured as a `Time Slave` for a Time Domain reveives a timestamp from another CAN node configured as `Time Master` linked to that `Time Domain` and synchronizes its local time to the `Time Master`'s time.
3	Forwarding a reference time from one CAN bus to another one	A CAN node that is configured `Time Gateway`, i.e., it is configured • as a `Time Slave` for `Time Domain` on one CAN Bus (i.e., to synchronize its time to a `Time Master` on that CAN bus) • and at the same time as `Time Master` for the same `Time Domain` on a different CAN bus "forwards" time from the CAN bus, that it is connected to as a Time Slave, to the CAN bus that it is connected to as a Time Master.

Table 2.1: Use Cases

3 Related documentation

3.1 Input documents & related standards and norms

[1] ISO 11898-1:2024 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical coding sublayer

[2] Requirements on Time Synchronization
AUTOSAR_FO_RS_TimeSync

[3] Glossary
AUTOSAR_FO_TR_Glossary

[4] E2E Protocol Specification
AUTOSAR_FO_PRS_E2EProtocol

[5] NIST Special Publication 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[6] NIST: Announcing the Advanced Encryption Standard (AES)
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[7] Specification of Secure Onboard Communication
AUTOSAR_CP_SWS_SecureOnboardCommunication

3.2 Related specification

None

4 Protocol Requirements

4.1 Requirements Traceability

Requirement	Description	Satisfied by
[RS_TS_20031]	Trigger Time Sync Transmission over CAN	[FO_PRS_CanTSyn_00232] [FO_PRS_CanTSyn_10010] [FO_PRS_CanTSyn_10011] [FO_PRS_CanTSyn_10014] [FO_PRS_CanTSyn_10030] [FO_PRS_CanTSyn_10032] [FO_PRS_CanTSyn_10033] [FO_PRS_CanTSyn_10034] [FO_PRS_CanTSyn_10035] [FO_PRS_CanTSyn_10049] [FO_PRS_CanTSyn_10051]
[RS_TS_20032]	Provide Time Base after Valid CAN Sync Message	[FO_PRS_CanTSyn_00087] [FO_PRS_CanTSyn_00133] [FO_PRS_CanTSyn_10083] [FO_PRS_CanTSyn_10084] [FO_PRS_CanTSyn_10085]
[RS_TS_20033]	Protect Time Sync Protocol over CAN	[FO_PRS_CanTSyn_00017] [FO_PRS_CanTSyn_00018] [FO_PRS_CanTSyn_00076] [FO_PRS_CanTSyn_00133] [FO_PRS_CanTSyn_00156] [FO_PRS_CanTSyn_00224] [FO_PRS_CanTSyn_00225] [FO_PRS_CanTSyn_00226] [FO_PRS_CanTSyn_00227] [FO_PRS_CanTSyn_10005] [FO_PRS_CanTSyn_10006] [FO_PRS_CanTSyn_10007] [FO_PRS_CanTSyn_10008] [FO_PRS_CanTSyn_10012] [FO_PRS_CanTSyn_10013] [FO_PRS_CanTSyn_10015] [FO_PRS_CanTSyn_10016] [FO_PRS_CanTSyn_10020] [FO_PRS_CanTSyn_10021] [FO_PRS_CanTSyn_10023] [FO_PRS_CanTSyn_10028] [FO_PRS_CanTSyn_10029] [FO_PRS_CanTSyn_10031] [FO_PRS_CanTSyn_10036] [FO_PRS_CanTSyn_10054] [FO_PRS_CanTSyn_10055] [FO_PRS_CanTSyn_10056] [FO_PRS_CanTSyn_10065] [FO_PRS_CanTSyn_10066] [FO_PRS_CanTSyn_10069] [FO_PRS_CanTSyn_10070] [FO_PRS_CanTSyn_10071]

▽

△

Requirement	Description	Satisfied by
[RS_TS_20035]	Support Precise Time Measurement over CAN	[FO_PRS_CanTSyn_00007] [FO_PRS_CanTSyn_00008] [FO_PRS_CanTSyn_00015] [FO_PRS_CanTSyn_00016] [FO_PRS_CanTSyn_00017] [FO_PRS_CanTSyn_00018] [FO_PRS_CanTSyn_00063] [FO_PRS_CanTSyn_00087] [FO_PRS_CanTSyn_00144] [FO_PRS_CanTSyn_00155] [FO_PRS_CanTSyn_00156] [FO_PRS_CanTSyn_00182] [FO_PRS_CanTSyn_00224] [FO_PRS_CanTSyn_00225] [FO_PRS_CanTSyn_00226] [FO_PRS_CanTSyn_00227] [FO_PRS_CanTSyn_10001] [FO_PRS_CanTSyn_10002] [FO_PRS_CanTSyn_10003] [FO_PRS_CanTSyn_10004] [FO_PRS_CanTSyn_10008] [FO_PRS_CanTSyn_10009] [FO_PRS_CanTSyn_10012] [FO_PRS_CanTSyn_10013] [FO_PRS_CanTSyn_10014] [FO_PRS_CanTSyn_10015] [FO_PRS_CanTSyn_10016] [FO_PRS_CanTSyn_10017] [FO_PRS_CanTSyn_10018] [FO_PRS_CanTSyn_10022] [FO_PRS_CanTSyn_10023] [FO_PRS_CanTSyn_10024] [FO_PRS_CanTSyn_10025] [FO_PRS_CanTSyn_10026] [FO_PRS_CanTSyn_10027] [FO_PRS_CanTSyn_10028] [FO_PRS_CanTSyn_10029] [FO_PRS_CanTSyn_10030] [FO_PRS_CanTSyn_10031] [FO_PRS_CanTSyn_10032] [FO_PRS_CanTSyn_10033] [FO_PRS_CanTSyn_10034] [FO_PRS_CanTSyn_10035] [FO_PRS_CanTSyn_10047] [FO_PRS_CanTSyn_10048] [FO_PRS_CanTSyn_10049] [FO_PRS_CanTSyn_10050] [FO_PRS_CanTSyn_10051] [FO_PRS_CanTSyn_10052] [FO_PRS_CanTSyn_10053] [FO_PRS_CanTSyn_10057] [FO_PRS_CanTSyn_10060] [FO_PRS_CanTSyn_10061] [FO_PRS_CanTSyn_10062] [FO_PRS_CanTSyn_10063] [FO_PRS_CanTSyn_10064] [FO_PRS_CanTSyn_10067] [FO_PRS_CanTSyn_10068] [FO_PRS_CanTSyn_10083] [FO_PRS_CanTSyn_10084] [FO_PRS_CanTSyn_10085] ▽

▽

△

Requirement	Description	Satisfied by
		△ [FO_PRS_CanTSyn_10086] [FO_PRS_CanTSyn_10087] [FO_PRS_CanTSyn_10088] [FO_PRS_CanTSyn_10089]
[RS_TS_20037]	Support User-Specific Data in CAN Time Protocol	[FO_PRS_CanTSyn_10019] [FO_PRS_CanTSyn_10020] [FO_PRS_CanTSyn_10058] [FO_PRS_CanTSyn_10059] [FO_PRS_CanTSyn_10084] [FO_PRS_CanTSyn_10085]
[RS_TS_20073]	Secure Time Synchronization Protocol over CAN	[FO_PRS_CanTSyn_00155] [FO_PRS_CanTSyn_00156] [FO_PRS_CanTSyn_00182] [FO_PRS_CanTSyn_00194] [FO_PRS_CanTSyn_00195] [FO_PRS_CanTSyn_10002] [FO_PRS_CanTSyn_10028] [FO_PRS_CanTSyn_10029] [FO_PRS_CanTSyn_10037] [FO_PRS_CanTSyn_10038] [FO_PRS_CanTSyn_10039] [FO_PRS_CanTSyn_10040] [FO_PRS_CanTSyn_10041] [FO_PRS_CanTSyn_10042] [FO_PRS_CanTSyn_10043] [FO_PRS_CanTSyn_10044] [FO_PRS_CanTSyn_10045] [FO_PRS_CanTSyn_10046] [FO_PRS_CanTSyn_10062] [FO_PRS_CanTSyn_10063] [FO_PRS_CanTSyn_10072] [FO_PRS_CanTSyn_10073] [FO_PRS_CanTSyn_10074] [FO_PRS_CanTSyn_10075] [FO_PRS_CanTSyn_10076] [FO_PRS_CanTSyn_10077] [FO_PRS_CanTSyn_10078] [FO_PRS_CanTSyn_10079] [FO_PRS_CanTSyn_10080] [FO_PRS_CanTSyn_10081] [FO_PRS_CanTSyn_10082]

Table 4.1: Requirements Tracing

5 Definition of terms and acronyms

This section lists abbreviations and definitions related to the Time Synchronization on CAN protocol. For additional abbreviations and definitions refer

- to chapter 3 in the RS Time Synchronization [2]
- and to the AUTOSAR Glossary [3] for general terms and abbreviations.

5.1 Acronyms and abbreviations

Abbreviation/Acronym	Description
CAN FD	Controller Area Network (CAN) - Flexible Data Rate
CAN 2.0	CAN protocol according to CAN 2.0 [1, ISO 11898-1:2024]
CRC	Cyclic Redundancy Checksum
D	Domain Id: field in Time Synchronization messages
FV	Freshness Value: field in FUP message
FVL	Freshness Value Length: field in FUP message
FVM	Freshness Value Manager: entity responsible for providing freshness values
ICV	Integrity Check Value: field in FUP message
ICVL	Integrity Check Value Length: field in FUP message
MAC	Message Authentication Code
OVS	Overflow Seconds value (field in FUP message)
SC	Sequence Counter in Time Synchronization messages
SGW	"Synchronized to Gateway" / Time Gateway Synchronization Status of a Time Domain • 0 (= SyncToGTM), if the time sent by the Time Master in the Time Synchronization messages is synchronized to the system-wide Time Master of the Time Domain (and no synchronization loss has been detected) • 1 (= SyncToSubDomain), if the time sent by the Time Master in the Time Synchronization messages is not in sync with the system-wide Time Master of the corresponding Time Domain, but only with (one of) the Time Gateway(s). That is, the local CAN node which the Time Master resides on is a Time Gateway for the given Time Domain and – either a Time Gateway in a chain of Time Domains before the local CAN node is no longer synchronized with the system-wide Time Master – or the (local) Time Gateway itself is no longer synchronized to its predecessor Time Master in the chain of Time Domains.
SyncTimeSec	"SyncTimeSec" field of SYNC message
SyncTimeNSec	"SyncTimeNSec" field of FUP message

Table 5.1: Acronyms and abbreviations used in the scope of this Document

5.2 Definition of terms

Term	Description
Clock	Entity that measures time
FUP message	Follow-Up message
Global Time	Time that is synchronized to an external reference clock, i.e., on Time Slave side to a Time Master or on Time Master side to an external time provider (e.g. GNSS)
Local Time	Time that not synchronized to an external reference clock, i.e., it is directly derived from the local clock
stuck Sequence Counter	Difference between the Sequence Counter of the current message and the Sequence Counter of the previous message is 0
SYNC message	Synchronization message
Time Domain	A Time Domain on CAN denotes which CAN nodes are linked to a given Time Master as the source of time. All CAN nodes of the Time Domain are linked to the same CAN bus.
Time Gateway	A Time Gateway is a network node, that receives/consumes as a Time Slave the time information of a Time Domain from the Time Master within a certain segment of a communication network and sends the time information within another (or even the same) segment of a communication network (thus acting as a Time Master).
Time Master	A Time Master is a network node, that transmits/provides the time information of a Time Domain to a set of time receivers (denoted as Time Slaves) within a segment of a communication network.
Time Slave	A Time Slave is a network node, that receives/consumes the time information of a Time Domain from the Time Master within a certain segment of a communication network.
Message Type	Message Type field of the Time Synchronization messages (SYNC and FUP)
User Byte	User Byte fields of the Time Synchronization messages (UserBytes 1, 2 and 3)
validSequenceCounterJumpsDuringTimeoutCnt	Counter that counts the number of valid sequence counter values received in a row while the Time Domain is in $\Delta t_{SlaveSyncLossTimeout}$.

Table 5.2: Definition of terms in the scope of this document

5.3 Inputs to the Time Synchronization protocol

Term	Description
APP_TMRequestSyncTx	Input (request) from the application to trigger a SYNC message transmission.
APP_TMUserData	Input from the Time Master application for the User Data bytes
APP_TMGatewayStatus	Input from the Time Master application for the "Synchronized to Gateway" status flag of the Time Domain as transmitted by a Time Gateway. A Time Gateway sets this flag to 1, if it is no longer synchronized to the system-wide Time Master of the given Time Domain. Otherwise this status flag is reset/not set.

Term	Description
APP_TMMasterTimeAvailable	Input from the Time Master application to the Time Master state machine that indicates if a valid Time Master time is available to be transmitted to the Time Slaves.
DLL_EventSyncTxSuccess	Input to the Time Master state machine from Data Link Layer indicate a successful transmission of a SYNC message.
DLL_EventSyncTxFailure	Input to the Time Master state machine from the Data Link Layer to indicate that a transmission of a SYNC message has failed
EXT_TMEventFvCalcFailure	Input from the Freshness Value Manager that indicates a Freshness Value calculation failure.
EXT_TMEventFvCalcSuccess	Input from the Freshness Value Manager that indicates a successful Freshness Value calculation.
EXT_TMEventIcvCalcSuccess	Input from Cryptography that indicates a successful ICV calculation
EXT_TMEventIcvCalcFailure	Input from Cryptography that indicates a ICV calculation failure
EXT_TMFreshnessValue	Input from the Freshness Value Manager to the Time Master. that provides the ICV value.
EXT_TSFreshnessValue	Input from the Freshness Value Manager to the Time Slave that provides the ICV value.
EXT_TMIntegrityCheckValue	Input from Cryptography to the Time Master that provides the ICV value.
EXT_TSIntegrityCheckValue	Input from Cryptography to the Time Slave that provides the ICV value.

Table 5.3: Inputs to the protocol

5.4 Outputs of the Time Synchronization protocol

Term	Description
APP_TMEventSyncTxFailure	Output (indication) to the application for a failed SYNC message transmission.
APP_TSUserData	Output to the Time Slave application for the User Data byte array.
APP_TSUserDataLength	Output to the Time Slave application for the length of the User Data.
APP_TSGatewayStatus	Output to the Time Slave application for the "Synchronized to Gateway" status flag of a Time Domain as received by the protocol.
DLL_TransmitSyncMsg	Output from the protocol Time Master to the Data Link Layer to request a transmission of a SYNC message on the bus.
DLL_TransmitFupMsg	Output from the protocol Time Master to the Data Link Layer to request a transmission of a FUP message on the bus.

Table 5.4: Outputs of the protocol

5.5 Timing Values used by the Time Synchronization protocol

Term	Description
$\Delta t_{FupRxTimeout}$	The timeout interval when waiting for FUP message after a corresponding SYNC message has been successfully received. Configured by parameter Cfg_TSDebounceInterval.
$\Delta t_{LinkDelay}$	Propagation delay of a message on the CAN bus. This value is not measured, but configured by the parameter Cfg_TSLinkDelay.
$\Delta t_{RxDebounceInterval}$	Time interval between the reception of two Time Synchronization messages. Configured by parameter Cfg_TSDebounceInterval.
$\Delta t_{SlaveSyncLossTimeout}$	Timeout interval between two valid FUP messages on Time Slave side. If timeout is exceeded the synchronization to the Time Master is considered lost. Configured by Cfg_TSSyncLossTimeout.
$\Delta t_{SlaveRxOffset}$	Time offset from SYNC ingress to actually re-synchronizing the Time Slave's Local Time (which can only be done after the FUP message has been received).
$\Delta t_{SyncTxOffset}$	Time offset from transmission request for SYNC message to actual egress of the SYNC mesage.
$\Delta t_{SyncTxInterval}$	Time interval for SYNC message transmission requests of the Time Master. Interval is configured by Cfg_TMSyncTxInterval.
$\Delta t_{SyncTxTimeout}$	The timeout interval when waiting for the transmission confirmation after transmitting a SYNC message. Configured by Cfg_TMSyncTxTimeout. The default value is set to 2 sec. This is to avoid an overflow of the OVS value in the FUP message which has a value range of: 0 .. 3.
$\Delta t_{TxDebounceInterval}$	Inhibit time interval between the transmission of Time Synchronition messages to avoid bursts on the CAN bus. Configured by parameter Cfg_TMDebounceInterval.
$T_{SyncEgress}$	Egress Timestamp of SYNC message on Time Master side.
$T_{SyncIngress}$	Ingress Timestamp of SYNC message on Time Slave side.
$t_{TMLocalTime}$	Current Local Time of a Time Master.
$T_{TMLocalTime}$	Timestamp of the Global Time sampled by the Time Master before transmission of the SYNC message.
$t_{TMGlobalTime}$	Current Global Time of a Time Master.
$T_{TMGlobalTime}$	Timestamp of the Global Time sampled by the Time Master before transmission of the SYNC message.
$T_{TMGlobalTime_Sec}$	Seconds part of $T_{TMGlobalTime}$.
$T_{TMGlobalTime_NSec}$	Nanoseconds part of $T_{TMGlobalTime}$.
$t_{TSGlobalTime}$	Current Global (i.e., synchronized) Time of the Time Slave.
$T_{TSGlobalTime}$	Time value calculated by the Time Slave from the SYNC and FUP message content to update $t_{TSGlobalTime}$.
$T_{TSGlobalTime_Sec}$	Seconds part of $T_{TSGlobalTime}$.
$T_{TSGlobalTime_NSec}$	Nanoseconds part of $T_{TSGlobalTime}$.
$t_{TSLocalTime}$	Current Local Time of the Time Slave.
$T_{TSLocalTime_ReSync}$	Local Time of the Time Slave, when Time Slave is re-synchronized to the Time Master.
r_{Slave}	Rate ratio between Local Time and Global Time on Time Slave side.

Table 5.5: Timestamps, time values and time intervals of the protocol

6 Protocol specification

6.1 Message types

There are two types of Time Synchronization messages

- SYNC messages
- and FUP messages.

SYNC and FUP messages of the same Time Domain may share the same CAN ID. The multiplexer (indicating if SYNC or FUP payload is contained) is located at byte 0, named as `Type`.

If Time Synchronization messages are sent by the same Time Master the same CAN ID typcially share the same different Time Domains. If Time Synchronization messages are sent by different Time Masters different CAN IDs need to be used for different Time Domains.

6.2 Message format

Time Synchronization messages may be protected by a `CRC`. The `CRC` is optional.

Time Synchronization messages can be secured by an `ICV`, that provides the integrity and authenticity protection of these messages. The authentication of time is supported on CAN FD/XL only. The usage of a `ICV` is optional.

The `ICV` is included in the FUP messages. This is to ensure a higher performance of the system. However, the data used for `ICV` calculation includes the payload of SYNC and FUP messages.

The actual message layout of the SYNC and FUP messages varies depending on the following configuration settings of the Time Master

- `CRC` protection (on / off)
- `ICV` protection (on / off)

The message layout is indicated by the Message `Type` Field in the individual message as given in Table 6.1.

Configuration			Message			
with ICV	**with CRC**	**Bus**	**Type**	**Field**	**Length**	**Layout**
no	no	CAN FD/XL/2.0	SYNC	`0x10`	8	[FO_PRS_CanTSyn_00015]
no	yes	CAN FD/XL/2.0		`0x20`	8	[FO_PRS_CanTSyn_00017]
no	no	CAN FD/XL/2.0	FUP	`0x18`	8	[FO_PRS_CanTSyn_00016]
no	yes	CAN FD/XL/2.0		`0x28`	8	[FO_PRS_CanTSyn_00018]
yes	no	CAN FD/XL		`0x78`	10 + (`FVL`+7)/8 + `ICVL`	[FO_PRS_CanTSyn_00155]

yes	yes	CAN FD/XL		0x88	10 + (FVL+7)/8 + ICVL	[FO_PRS_CanTSyn_00156]

Table 6.1: Message Types supported by Time Synchronization Over CAN

10+(FVL+7)/8 +ICVL-1

Note: For calculation of the Message Type Field refer to [FO_PRS_CanTSyn_10012] and [FO_PRS_CanTSyn_10028].

Note: For calculation of the Message Length refer to [FO_PRS_CanTSyn_10001] and [FO_PRS_CanTSyn_10002].

[FO_PRS_CanTSyn_00008] Endianess of Time Synchronization messages

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00008

⌈The byte order for time value signals in Time Synchronization messages is "Big Endian".⌋

[FO_PRS_CanTSyn_10001] Message Length of not ICV secured Time Sync messages

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00010

⌈If the Message Type is

- 0x10 (SYNC, not CRC protected)
- or 0x20 (SYNC, CRC protected)
- or 0x18 (FUP, not CRC protected, not IVC secured)
- or 0x28 (FUP, CRC protected, not IVC secured),

then the Message Length of SYNC and FUP messages shall be 8 Bytes.⌋

[FO_PRS_CanTSyn_10002] Message Length of ICV secured Time Sync messages

Upstream requirements: RS_TS_20035, RS_TS_20073
Previous identifiers: SWS_CanTSyn_00010

⌈If the Message Type is

- 0x78 (FUP, not CRC protected, IVC secured)
- or 0x88 (FUP, CRC protected, IVC secured),

then the Message Length shall be 10 Bytes + (FVL+7)/8 + ICVL.

The Message Length shall not exceed 64 Bytes.⌋

Note: Since the value FVL is defined in bits, the term (FVL + 7) / 8 is the resulting length in bytes. The protocol configuration needs to make sure that the value (FVL + 7) / 8 + ICVL does not exceed 54 Bytes. The detailed message layout of the SYNC and FUP messages is defined by the following two chapters.

6.2.1 SYNC Message Format

[FO_PRS_CanTSyn_00015] SYNC message format - not CRC protected

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00015

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x10	
1		UserByte_1	default: 0	
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3		UserByte_0	default: 0	
4..7		SyncTimeSec	0 .. 4294967295 s	Seconds part of the time

⌋

[FO_PRS_CanTSyn_00017] SYNC message format - CRC protected

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00017

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x20	
1		CRC		
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3		UserByte_0	default: 0	
4..7		SyncTimeSec	0 .. 4294967295 s	Seconds part of the time

⌋

6.2.2 Follow-up (FUP) Message Format

[FO_PRS_CanTSyn_00016] FUP message format - not CRC protected, not ICV secured

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00016

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x18	
1		UserByte_2	default: 0	
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3	7..3	reserved	default: 0	
	2	SGW	SyncToGTM = 0 SyncToSubDomain = 1	
	1..0	OVS	0 .. 3 s	Overflow into seconds
4..7		SyncTimeNSec	0 .. 999999999 ns	Nanoseconds part of time

⌋

[FO_PRS_CanTSyn_00018] FUP message format - CRC protected, not ICV secured

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00018

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x28	
1		CRC		
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3	7..3	reserved	default: 0	
	2	SGW	SyncToGTM = 0 SyncToSubDomain = 1	
	1..0	OVS	0 .. 3 s	Overflow into seconds
4..7		SyncTimeNSec	0 .. 999999999 ns	Nanoseconds part of time

⌋

[FO_PRS_CanTSyn_00155] FUP message format - not CRC protected, ICV secured

Upstream requirements: RS_TS_20035, RS_TS_20073
Previous identifiers: SWS_CanTSyn_00155

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x78	
1		UserByte_2	default: 0	
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3	7..3	reserved	default: 0	
	2	SGW	SyncToGTM = 0 SyncToSubDomain = 1	
	1..0	OVS	0 .. 3 s	Overflow into seconds
4..7		SyncTimeNSec	0 .. 999999999 ns	Nanoseconds part of time
8	7	reserved	always 0	

	6..0	FVL	0..64	FV Length in bits
9	7..6	reserved	always 0	
	5..0	ICVL	0..54	ICV Length in bytes
If FVL > 0				
10 .. 10+(FVL+7)/8-1		FV		FV
If ICVL > 0				
10+(FVL+7)/8 .. 10+(FVL+7)/8 +ICVL-1		ICV		ICV

⌋

[FO_PRS_CanTSyn_00156] FUP message format - CRC protected, ICV secured

Upstream requirements: RS_TS_20033, RS_TS_20035, RS_TS_20073
Previous identifiers: SWS_CanTSyn_00156

⌈

Byte	Bit Position	Field Name	Field Value Range	Description
0		Type	0x88	
1		CRC		
2	7..4	D	0..15	Time Domain Id
	3..0	SC	0..15	Sequence Counter
3	7..3	reserved	default: 0	
	2	SGW	SyncToGTM = 0 SyncToSubDomain = 1	
	1..0	OVS	0 .. 3 s	Overflow into seconds
4..7		SyncTimeNSec	0 .. 999999999 ns	Nanoseconds part of time
8	7	reserved	always 0	
	6..0	FVL	0..64	FV Length in bits
9	7..6	reserved	always 0	
	5..0	ICVL	0..54	ICV Length in bytes
If FVL > 0				
10 .. 10+(FVL+7)/8-1		FV		FV
If ICVL > 0				
10+(FVL+7)/8 .. 10+(FVL+7)/8 +ICVL-1		ICV		ICV

⌋

6.2.3 Data Types

Not applicable.

6.3 Services / Commands

Not applicable

6.4 Sequences (lower layer)

Figure 6.1 shows the protocol sequence in Figure 1.1 with a little more detail by adding the Data Link Layer.

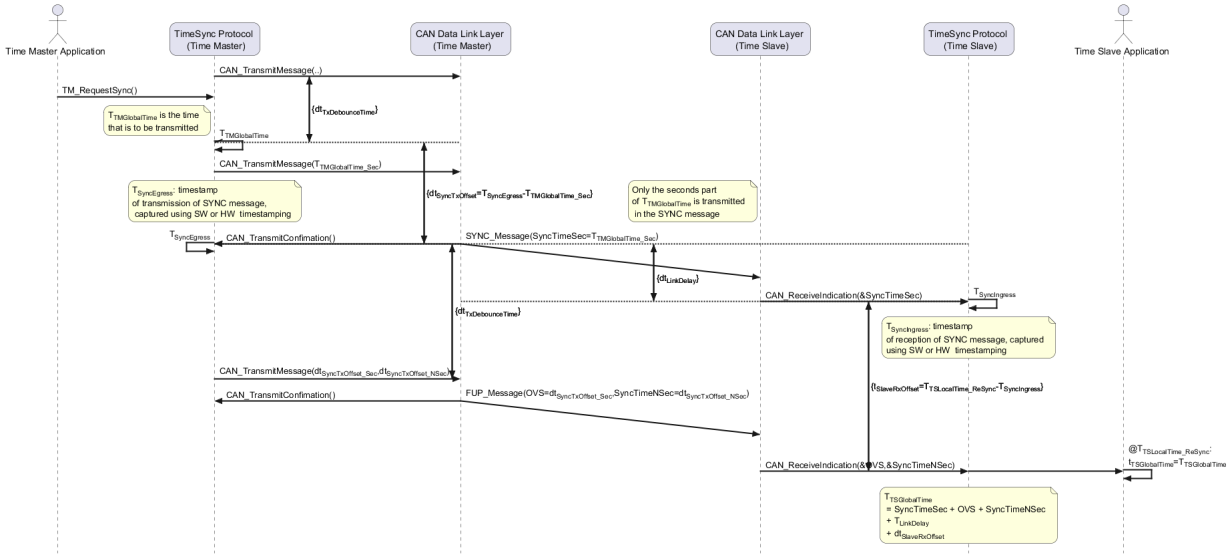

Figure 6.1: CAN Time Synchronization Protocol Sequence

6.4.1 General Behavior of the Time Synchronization Protocol

[FO_PRS_CanTSyn_10003] Unique Domain Id

Upstream requirements: RS_TS_20035

⌈The `Cfg_TDDomainId` shall be unique per CAN bus.⌋

Note: That means no two Time Domains with the same `Cfg_TDDomainId` will coexist on a given CAN bus.

[FO_PRS_CanTSyn_10004] Mapping of Time Domain to CAN ID

Upstream requirements: RS_TS_20035

⌈For a given CAN bus each Time Domain `Cfg_TDDomainId` shall be mapped to a different CAN ID (refer to parameter `Cfg_TDCanId`).⌋

6.4.1.1 CRC Protection of Time Synchronization Messages

The CRC protection of the payload of the Time Synchronization messages is based on AUTOSAR E2E Profile 2. For details refer to [4, FO-PRS-E2EProtocol].

The use of a CRC checksum is optional and depends on the setting of

- configuration parameter `Cfg_TMCrcProtected` on Time Master side
- configuration parameter `Cfg_TSCrcVerified` on Time Slave side

The 8 bit CRC checksum is calculated taking into account

- the payload to be protected
- and a configured Data Id

[FO_PRS_CanTSyn_10005] CRC Checksum Algorithm of Time Synchronization Protocol

Upstream requirements: RS_TS_20033

⌈The *8 bit CRC checksum* of the Time Synchronization message shall be calculated using the following parameters

- generator polynomial: `0x2F`
- initial value: `0xFF`
- XOR value: `0xFF`
- input data: not reflected
- result data: not reflected

⌋

Due to the limited length of the 8 bit polynomial, it is possible that a masquerading fault cannot be detected in a specific cycle when evaluating a received CRC value. By including a so called Data ID into the CRC calculation a masquerading fault can be detected in one of the successive communication cycles.

[FO_PRS_CanTSyn_10006] Calculation of the Data ID for SYNC Messages

Upstream requirements: RS_TS_20033

⌈The `DataID` of a SYNC message shall be calculated as `DataID` = `Cfg_TDSyncMsgDataIDList[Sequence Counter]`, with

- `Sequence Counter` as the sequence counter of the SYNC message.

⌋

[FO_PRS_CanTSyn_10007] Calculation of the Data ID for FUP Messages

Upstream requirements: RS_TS_20033

⌈The `DataID` of a FUP message shall be calculated as `DataID` = `Cfg_TDFupMsg-DataIDList[Sequence Counter]`, with

- `Sequence Counter` as the sequence counter of the FUP message.

⌋

Note: The `DataID` itself is not transmitted on the bus. It is only locally derived from configuration.

6.4.1.2 ICV Calculation of Time Synchronization Messages

The Time Synchronization protocol as described in this document provides a mechanism to verify the authenticity and freshness of **secured** Time Synchronization messages.

The requirements in

- chapter "Set Integrity Check Value Fields (*ICV* and *ICVL*)",
- chapter "Set Freshness Value Field (*FV* and *FVL*)",
- chapter "Check Freshness Value Fields (*FV* and *FVL*)"
- and chapter "Check ICV Value Fields (*ICV* and *ICVL*)"

define only the general flow of calculating the FreshnessValue and ICV but have no details about the actual MAC or Signature calculation.

For the resource constraint embedded use case with static participants, the use of Message Authentication Codes (MACs) as a basis for authentication (e.g. a CMAC [5] based on AES [6] with an adequate key length) is proposed.

In case a MAC is used, it is possible to transmit and compare only parts of the MAC. This is known as MAC truncation. However, this results in a lower security level at least for forgery of single MACs. It is proposed to always use a key length of at least 128 bit, a MAC truncation can be beneficial. Of course, the actual length of the MAC for each use case has to be chosen carefully. For some guidance, we refer to appendix A of [5]. In general, MAC sizes of 64 bit and above are considered by NIST to provide sufficient protection against guessing attacks. Depending on the use case, different MAC sizes can be appropriate, but this requires careful judgment by a security expert.

Although this document consequently uses the terms and concepts from symmetric cryptography, the Time Synchronization protocol supports both symmetric and asymmetric cryptographic algorithms. In case of an asymmetric approach using digital signatures instead of the MAC approach, some adaptations must be considered:

- Instead of a shared secret between Time Master and (all) Time Slaves, a key pair consisting of public key and secret key is used. The secret (or private) key is used by the sender to generate the signature, the corresponding public keys is used by (all) Time Slaves(s) to verify the signature. The private key must not be feasibly computable from the public key and it shall not be assessable by the Time Slaves.
- In order to verify a message, the receiver needs access to the complete signature output of the signature generation algorithm. Therefore, a truncation of the signature as proposed in the MAC case is **not** possible. Truncation as with the MAC approach may be desired when the message payload is limited in length and does not have sufficient space to include the full ICV.
- The signature verification uses a different algorithm then the signature generation. So instead of "rebuilding" the MAC on receiver side and comparing it with the received (truncated) MAC as given above, the Time Slave performs the verification algorithm using the ICV and the signature as inputs and getting a Boolean value as output, determining whether the verification passed or failed.

6.4.1.3 User Data

User Data are part of each Time Domain. User Data are set by the system-wide Time Master of each Time Base and distributed as part of the Time Synchronizatoin messages.

User Data are used for application specific purposes, e.g.

- for debugging purposes during integration
- and/or to characterize the Time Base, e.g.,
  - regarding the quality of the underlying clock source
  - or regarding the progress of time (e.g., if Synchronized Time has been restored from non-volatile memory after reset).

Note, that the status information, if contained in the User Data, is from the Time Master's point of view.

User Data consist of up to three bytes. Due to the frame format of various Time Synchronization messages it is not possible to transmit all three bytes on the CAN system. It is the responsibility of the system designer to only use those User Data bytes that can be distributed inside the CAN network.

6.4.2 Time Master

The AUTOSAR Time Synchronization protocol over CAN specifies a protocol state machine for the Time Master and one for the Time Slave. The protocol state machines

define, how to process the SYNC and FUP messages of the given Time Domain on Time Master and on Time Slave side.

In the following the term "Time Master" refers to a CAN node that is configured as a Time Master for a given Time Domain and that complies to the Time Master protocol state machine of the AUTOSAR Time Synchronization Protocol over CAN.

Similarly, the term "Time Slave" refers to a CAN node that is configured as a Time Slave for a given Time Domain and that complies to the Time Slave protocol state machine of the AUTOSAR Time Synchronization Protocol over CAN.

Figure 6.2 illustrates the protocol state machine of a Time Master.

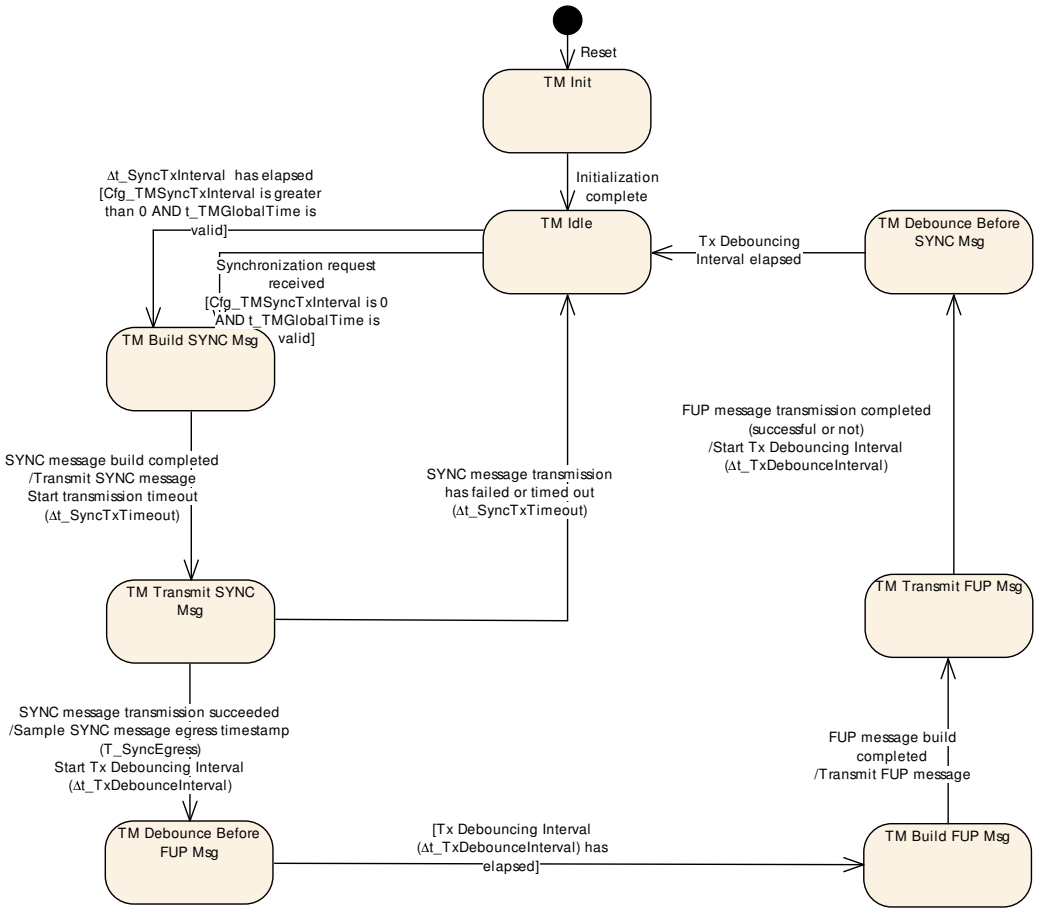

Figure 6.2: State Machine of a Time Master

6.4.2.1 State TM_Init

[FO_PRS_CanTSyn_10008] Initialize Sequence Counter of Time Master

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈If in state `TM_Init`, then the Time Master shall

- initialize the Sequence Counter value for the given Time Domain with 0

- and set the $\Delta t_{SyncTxInterval}$ to 0.

⌋

6.4.2.1.1 Transition from TM_Init

[FO_PRS_CanTSyn_10009] Transition from state TM_Init

Upstream requirements: RS_TS_20035

⌈When initialization (TM_Init) is completed, the Time Master shall go to state TM_Idle.⌋

6.4.2.2 State TM_Idle

In state TM_Idle the Time Master waits for the next SYNC message transmission request from the application. The Time Master is in this state after initialization and returns to this state when the Time Master has successfully completed or aborted the Time Synchronization sequence.

6.4.2.2.1 Transition from TM_Idle

The Time Synchronization over CAN protocol supports cyclic as well as event driven transmission requests to start a Time Synchronization sequence. If the parameter Cfg_TMSyncTxInterval

- is set to 0, the protocol waits for an external trigger request by the application to start the synchronization sequence
- is greater than 0, the protocol itself cyclically triggers a synchronization sequence

[FO_PRS_CanTSyn_10010] Triggering of synchronization sequence by external request

Upstream requirements: RS_TS_20031

⌈If

- in state TM_Idle
- and Cfg_TMSyncTxInterval is set to 0
- and $t_{TMGlobalTime}$ is valid (refer to APP_TMMasterTimeAvailable),

when a request to start a Time Synchronization sequence is received (refer to APP_TMRequestSyncTx), then the Time Master shall set the protocol state of that Time Domain to state TM_BuildSyncMsg.⌋

[FO_PRS_CanTSyn_10011] Cyclic triggering of synchronization sequence

Upstream requirements: RS_TS_20031

⌈If

- in state TM_Idle
- and Cfg_TMSyncTxInterval is greater than 0
- and $t_{TMGlobalTime}$ is valid (refer to APP_TMMasterTimeAvailable),

when $\Delta t_{SyncTxInterval}$ has elapsed (i.e., $\Delta t_{SyncTxInterval}$ equals 0), then the Time Master shall

- re-start the $\Delta t_{SyncTxInterval}$ (i.e., set it to Cfg_TMSyncTxInterval)
- and set the protocol state of that Time Domain to state TM_BuildSyncMsg.

⌋

Note: There are different scenarios for when a Time Master application may indicate to the Time Synchronization protocol that $t_{TMGlobalTime}$ is valid, e.g., after reset:

- when the Time Master has set the time to a value derived from application specific requirements
- when the Time Master has restored its time from persistant memory
- when the Time Master has successfully synchronized its time with an external time source (e.g. GNSS time) for the first time.

[FO_PRS_CanTSyn_00232] No queuing of SYNC transmission requests

Upstream requirements: RS_TS_20031
Previous identifiers: SWS_CanTSyn_00232

⌈If

- in state TM_Idle
- and multiple SYNC message requests are pending for a given Time Domain,

the Time Master shall execute only the latest request and shall discard the others.⌋

Rationale: There is no queuing of multiple transmission requests. The latest request is the best one.

6.4.2.3 State TM_BuildSyncMsg

Figure 6.3 illustrates the flow of actions when building a SYNC message.

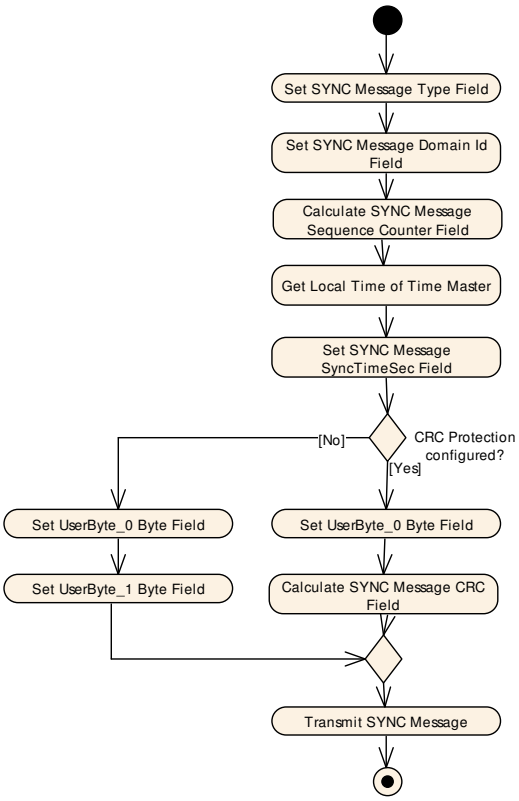

Figure 6.3: Flow of actions for building a SYNC message

6.4.2.3.1 Set SYNC Message Type Field (*Type*)

[FO_PRS_CanTSyn_10012] Calculation of the Message Type Field of a SYNC Message

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈If in state `TM_BuildSyncMsg`, then the Time Master shall build a SYNC message and set the Message `Type` field of the SYNC message depending on the configuration parameter `Cfg_TMCrcProtected` of the given Time Domain according to requirement [FO_PRS_CanTSyn_10013].⌋

[FO_PRS_CanTSyn_10013] Message Type of a SYNC Message

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈

Configuration	Message	
`Cfg_TMCrcProtected`	**Message Type**	**Description**
`CRC_NOT_SUPPORTED`	`0x10`	SYNC message, not CRC protected
`CRC_SUPPORTED`	`0x20`	SYNC message, CRC protected

⌋

6.4.2.3.2 Set SYNC Message Domain Id Field (*D*)

[FO_PRS_CanTSyn_10014] Calculate Time Domain Id of a SYNC Message

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈The Time Master shall set the `D` field of the SYNC message to the Time Domain Id `Cfg_TDDomainId` of the given Time Domain.⌋

6.4.2.3.3 Set SYNC Message Sequence Counter Field (*SC*)

[FO_PRS_CanTSyn_10015] Time Master Calculation of the Sequence Counter value of a SYNC Message

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈If in state `TM_BuildSyncMsg`, then the Time Master shall increment sequence counter value by 1 for the given Time Domain.

When the sequence counter value reaches 16, the Time Master shall set the sequence counter value back to 0.⌋

Note: The message transmission does not have to be successful for incrementing the sequence counter. The sequence counter is incremented for each SYNC message request.

[FO_PRS_CanTSyn_10016] Time Master Update of the SC Field of a SYNC Message

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈When the Time Master has updated the sequence counter value of the given Time Domain (refer to [FO_PRS_CanTSyn_10015]), then the Time Master shall set the `SC` field of the SYNC message to the updated sequence counter value.⌋

6.4.2.3.4 Get Current Local Time and User/Status Data

The Time Master has to derive the current time (and corresponding Time Domain status and user data) of the Time Master, before setting the corresponding fields of the SYNC and FUP message.

[FO_PRS_CanTSyn_10017] Get Current Time and Status from Application

Upstream requirements: RS_TS_20035

⌈If in state `TM_BuildSyncMsg`, then the Time Master shall read

- the current Time Master's Global Time $T_{TMGlobalTime}$
- and the current Time Master's Local Time $T_{TMLocalTime}$

- and the Time Gateway Synchronization status (via APP_TMGatewayStatus)
- and the user data (via to APP_TMUserData)

for the given Time Domain in a consistent way from the application.⌋

Note: "in a consistent way" is typically achieved by reading the data in "atomic", i.e., "non-interruptable" way.

Note: Although the Time Gateway Synchronization status is not needed for the transmission of the SYNC message (because it is contained in the SGW field of the subsequent FUP message) the Time Gateway Synchronization status shall reflect the status at the point in time when capturing the time information for the SYNC message.

6.4.2.3.4.1 Set SYNC Message Seconds Field (*SyncTimeSec*)

[FO_PRS_CanTSyn_10018] Calculation of the SyncTimeSec Field of a SYNC Message

Upstream requirements: RS_TS_20035

⌈When the Time Master has read the current time (refer to [FO_PRS_CanTSyn_10017]) then the Time Master set SyncTimeSec field of the SYNC message to the seconds part $T_{TMGlobalTime_Sec}$ of $T_{TMGlobalTime}$.⌋

Note: Taking the seconds part means truncating the nanoseconds part.

6.4.2.3.4.2 Set SYNC Message User Data Fields (*UserData_0/_1*)

[FO_PRS_CanTSyn_10019] Time Master Calculation of the UserData_0 Field of a SYNC Message

Upstream requirements: RS_TS_20037
Previous identifiers: SWS_CanTSyn_00013

⌈When the Time Master has read the current user data (refer to [FO_PRS_CanTSyn_10017]) then the Time Master shall set the field UserData_0 of the SYNC message to the value APP_TMUserData[0].⌋

[FO_PRS_CanTSyn_10020] Time Master Calculation of the UserData_1 Field of a SYNC Message

Upstream requirements: RS_TS_20033, RS_TS_20037
Previous identifiers: SWS_CanTSyn_00013

⌈If the configuration parameter Cfg_TMCrcProtected is set to CRC_NOT_SUPPORTED, when the Time Master has read the current user data from the application (refer to [FO_PRS_CanTSyn_10017]) then the Time Master shall set the field UserData_1 of the SYNC message to the value APP_TMUserData[1].⌋

Note: The configuration parameter `Cfg_TMCrcProtected` needs to be considered, because only if the message is not CRC protected, the `UserData_1` field is available at all. Otherwise, the `UserData_1` field is used for the CRC.

6.4.2.3.5 Set SYNC Message Checksum Field (*CRC*)

[FO_PRS_CanTSyn_10021] Calculation of CRC Field for SYNC Messages

Upstream requirements: RS_TS_20033

⌈If

- the configuration parameter `Cfg_TMCrcProtected` is set to `CRC_SUPPORTED`
- and the
  - `Type`,
  - `D`,
  - `SC`
  - `SyncTimeSec`,
  - and `UserByte_0`

  fields of the SYNC message have been set,

then the Time Master shall generate the `CRC` field of the SYNC message by calculating the `CRC` according to [FO_PRS_CanTSyn_10005] over bytes 2 to 7 and the `DataID`, where

- the `DataID` is calculated according to [FO_PRS_CanTSyn_10006]
- and byte 2 is applied first, followed by the other bytes in ascending order, and the `DataID` last.

⌋

6.4.2.3.6 Transition from `TM_BuildSyncMsg`

[FO_PRS_CanTSyn_10022] Transition from state `TM_BuildSyncMsg`

Upstream requirements: RS_TS_20035

⌈If

- in state `TM_BuildSyncMsg`
- and the configuration parameter `Cfg_TMCrcProtected` is set to `CRC_NOT_SUPPORTED`
- and the

  - Type,
  - D,
  - SC,
  - SyncTimeSec,
  - UserByte_0,
  - and UserByte_1

  of the SYNC message have been set,

then the Time Master

- shall transmit the SYNC message (refer to DLL_TransmitSyncMsg)
- and start the SYNC message transmission timeout $\Delta t_{SyncTxTimeout}$
- and transition to the protocol state TM_TransmitSyncMsg for the given Time Domain.

.⌋

[FO_PRS_CanTSyn_10023] Transition from state TM_BuildSyncMsg, if SYNC Message is CRC protected

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈If

- in state TM_BuildSyncMsg
- and the configuration parameter Cfg_TMCrcProtected is set to CRC_SUPPORTED
- and the CRC checksum of the SYNC message has been calculated,

then the Time Master shall

- transmit the SYNC message (refer to DLL_TransmitSyncMsg)
- and start the SYNC message transmission timeout $\Delta t_{SyncTxTimeout}$
- and transition to the protocol state TM_TransmitSyncMsg for the given Time Domain.

⌋

6.4.2.4 State TM_TransmitSyncMsg

In state TM_TransmitSyncMsg the Time Master is waiting for the SYNC message transmission confirmation of the Data Link Layer.

6.4.2.4.1 Transition from TM_TransmitSyncMsg

[FO_PRS_CanTSyn_10024] Successful Transmission of a SYNC Message

Upstream requirements: RS_TS_20035

⌈If

- in state TM_TransmitSyncMsg
- and the SYNC message transmission timeout $\Delta t_{SyncTxTimeout}$ has not yet elapsed,

when the transmission of the SYNC message completes successfully (refer to DLL_EventSyncTxSuccess) then the Time Master shall

- sample the $T_{SyncEgress}$
- and start the Tx debouncing interval $\Delta t_{TxDebounceInterval}$
- and set the protocol state for the given Time Domain to TM_DebounceBeforeFupMsg

⌋

Note: Sampling of the SYNC egress timestamp $T_{SyncEgress}$ is done either by SW or by reading it from the HW. If the egress timestamp $T_{SyncEgress}$ needs to be taken by SW (SW timestamping), this is highly time critical. The timestamp should to be taken in the interrupt service routine of the CAN driver for the CAN Transmission Confirmation interrupt. Any delay between the actual egress of the SYNC message and taking the timestamp in SW will worsen the precision of the time synchronization.

[FO_PRS_CanTSyn_10025] Transmission Timeout of a SYNC Message

Upstream requirements: RS_TS_20035

⌈If

- in state TM_TransmitSyncMsg

when the SYNC message transmission timeout interval $\Delta t_{SyncTxTimeout}$ elapses, then the Time Master shall

- set the protocol state for the given Time Domain to TM_Idle

⌋

[FO_PRS_CanTSyn_10026] Transmission Failure of a SYNC Message

Upstream requirements: RS_TS_20035

⌈If

- in state TM_TransmitSyncMsg
- and the transmission of the SYNC message has failed (refer to DLL_EventSyncTxFailure), i.e., no message has been sent,

then the Time Master shall

- set the protocol state for the given Time Domain to TM_Idle

⌋

6.4.2.5 State TM_DebounceBeforeFupMsg

In state TM_DebounceBeforeFupMsg the Time Master is waiting for the $\Delta t_{\text{TxDebounceInterval}}$ interval to elapse. The debouncing between the SYNC message and the FUP message avoids message bursts on the CAN bus.

6.4.2.5.1 Transition from TM_DebounceBeforeFupMsg

[FO_PRS_CanTSyn_10027] SYNC to FUP Intermessage Gap has elapsed

Upstream requirements: RS_TS_20035

⌈If in state TM_DebounceBeforeFupMsg, when the debounce interval $\Delta t_{\text{TxDebounceInterval}}$ has elapsed, then the Time Master shall transition to the protocol state TM_BuildFupMsg for the given Time Domain.⌋

6.4.2.6 State TM_BuildFupMsg

Figure 6.4 illustrates the flow of actions when building a FUP message.

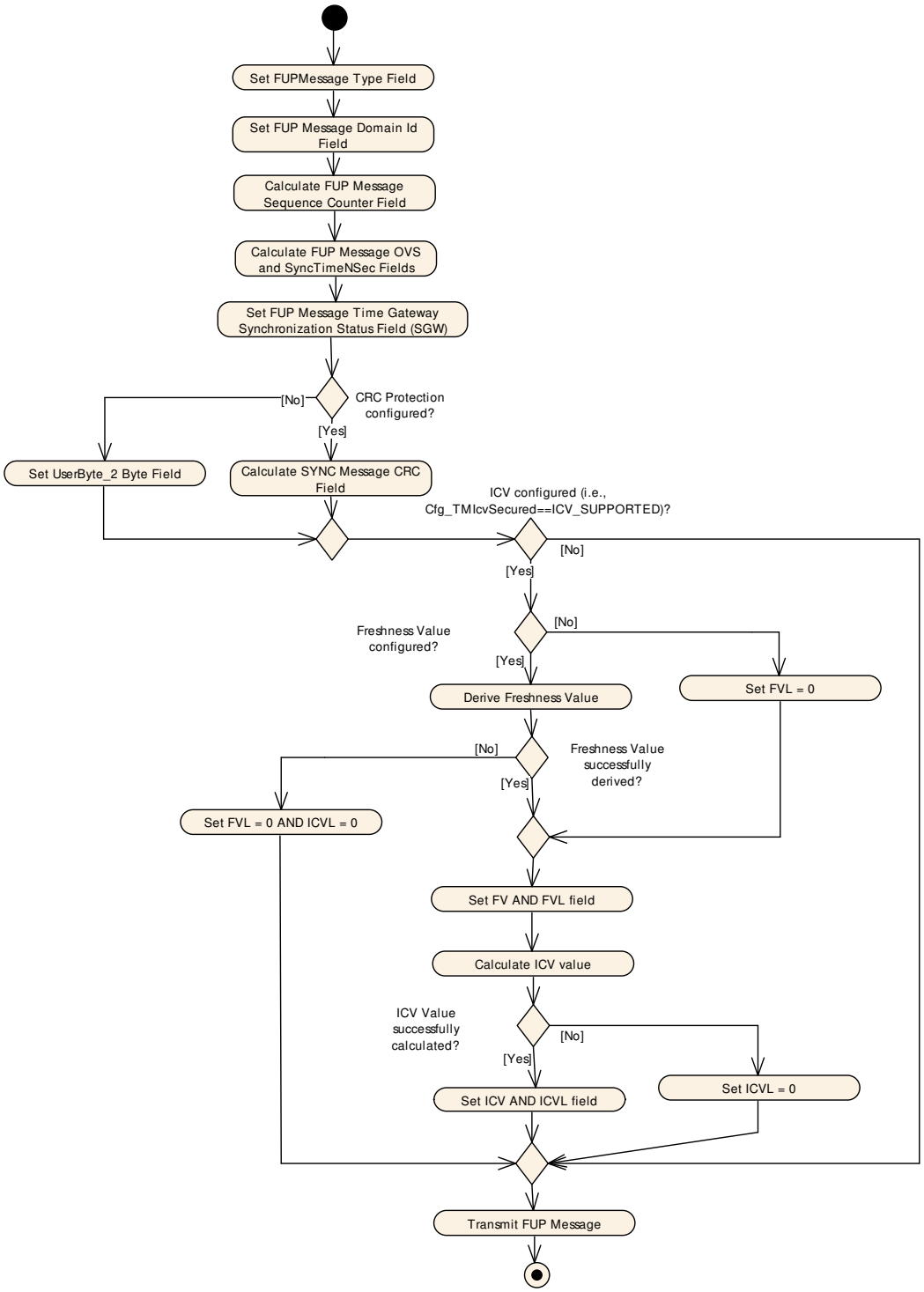

Figure 6.4: Flow of actions for building a FUP message

6.4.2.6.1 Set FUP Message Type Field (*Type*)

[FO_PRS_CanTSyn_10028] Calculation of FUP Message Type Field

Upstream requirements: RS_TS_20033, RS_TS_20035, RS_TS_20073

⌈If in state `TM_BuildFupMsg`, then the Time Master shall build a FUP message and set the `Type` field of the FUP message according to table [FO_PRS_CanTSyn_10029] depending on the configuration parameters

- `Cfg_TMCrcProtected`
- `Cfg_TMIcvSecured`.

⌋

[FO_PRS_CanTSyn_10029] Message Type of a FUP Message

Upstream requirements: RS_TS_20033, RS_TS_20035, RS_TS_20073

⌈

Configuration		Message	
`Cfg_TMIcvSecured`	`Cfg_TMCrcProtected`	**Type**	**Description**
`ICV_NOT_SUPPORTED`	`CRC_NOT_SUPPORTED`	`0x18`	FUP message ICV not secured CRC not protected
	`CRC_SUPPORTED`	`0x28`	FUP message ICV not secured CRC protected
`ICV_SUPPORTED`	`CRC_NOT_SUPPORTED`	`0x78`	FUP message ICV secured CRC not protected
	`CRC_SUPPORTED`	`0x88`	FUP message ICV secured CRC protected

⌋

6.4.2.6.2 Set FUP Message Domain Id Field (*D*)

[FO_PRS_CanTSyn_10030] Calculate Time Domain Id of a FUP Message

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈The Time Master shall set the `D` field of the FUP message to the Time Domain Id `Cfg_TDDomainId` of the given Time Domain.⌋

6.4.2.6.3 Set FUP Message Sequence Counter Field (*SC*)

[FO_PRS_CanTSyn_10031] Calculation of the Sequence Counter for a FUP Message

Upstream requirements: RS_TS_20033, RS_TS_20035

⌈The Time Master shall set the SC field of the FUP message to value of the sequence counter of the latest SYNC message for that Time Domain (refer to [FO_PRS_CanTSyn_10016]).⌋

6.4.2.6.4 Set FUP Message Nanoseconds Field (*SyncTimeNSec* and *OVS*)

The Time Synchronization protocol has to determine the SYNC message transmission offset $\Delta t_{SyncTxOffset}$. This accounts for the time interval $T_{SyncEgress}$ - $T_{TMLocalTime}$ between

- sampling the current Global Time of the Time Master $T_{TMGlobalTime}$ for transmission in the SYNC message (refer to [FO_PRS_CanTSyn_10017])
- and the actual egress of that SYNC message $T_{SyncEgress}$ on the bus,

Additionally, $\Delta t_{SyncTxOffset}$ accounts for the nanoseconds part of the Global Time $T_{TMGlobalTime_NSec}$, which has not been transmitted in the SYNC message.

In the FUP message $\Delta t_{SyncTxOffset}$ is split into a seconds part and a nanoseconds part. The seconds part is stored in the OVS field and the nanoseconds part is stored in the SyncTimeNSec field of the FUP message.

[FO_PRS_CanTSyn_10032] Calculate the SYNC Transmission Offset

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈If in state TM_BuildFupMsg, then the Time Master shall calculate the SYNC message transmission offset $\Delta t_{SyncTxOffset}$ as $\Delta t_{SyncTxOffset}$ = $T_{TMGlobalTime_NSec}$ + $T_{SyncEgress}$ - $T_{TMLocalTime}$.⌋

Note: The equation above and with that the Time Synchronization protocol currently assumes that there is no rate deviation between $t_{TMLocalTime}$ and $t_{TMGlobalTime}$. If the rate of $t_{TMLocalTime}$ would deviate from the rate of $t_{TMGlobalTime}$, the calculation of $T_{SyncEgress}$ - $T_{TMLocalTime}$ would need to be corrected accordingly.

Also, if egress timestamping (i.e., $T_{SyncEgress}$) uses a different Clock than the Time Master for maintaining the Local Time, then the Time Master needs to correlate the two clocks using cross-timestamping. This is to correct the rate deviation and offset of the clocks.

[FO_PRS_CanTSyn_10033] Setting the *SyncTimeNSec* and *OVS* Field

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈When $\Delta t_{SyncTxOffset}$ has been calculated then the Time Master shall

- set the OVS field of the FUP message to the seconds part of $\Delta t_{SyncTxOffset}$ (i.e., number of overflows into seconds)
- and set the SyncTimeNSec field to the remaining nanoseconds part of $\Delta t_{SyncTxOffset}$

⌋

Example: For $\Delta t_{SyncTxOffset}$ = 1.9989 * 10 ^{9} ns, the *OVS* field is set to 1 and the SyncTimeNSec field is set to 998900000 ns.

6.4.2.6.5 Set FUP Message Time Gateway Synchronization Status Field (*SGW*)

[FO_PRS_CanTSyn_10034] Setting the *SGW* Field

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈The Time Master shall set the SGW field to value of the Time Gateway Synchronization Status APP_TMGatewayStatus for the given Time Domain (refer to [FO_PRS_CanTSyn_10017]).⌋

6.4.2.6.6 Set FUP Message User Data Field (*UserData_2*)

[FO_PRS_CanTSyn_10035] Time Master Calculation of the UserData_2 Field of a FUP Message

Upstream requirements: RS_TS_20031, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00013

⌈If Cfg_TMCrcProtected is CRC_NOT_SUPPORTED, then the Time Master shall set the field UserByte_2 of the FUP message to the user data byte APP_TMUserData[2] for the given Time Domain (refer to [FO_PRS_CanTSyn_10017]).⌋

6.4.2.6.7 Set FUP Message Checksum Field (*CRC*)

[FO_PRS_CanTSyn_10036] Calculation of CRC Field for FUP Messages

Upstream requirements: RS_TS_20033

⌈If

- and Cfg_TMCrcProtected is CRC_SUPPORTED
- and the
  - Type,
  - D,
  - SC

- SyncTimeNSec,
- OVS,
- and SGW

fields of the FUP message have been set,

then the Time Master shall generate the CRC field of the FUP message by calculating the CRC according to [FO_PRS_CanTSyn_10005] over bytes 2 to 7 and the DataID, where

- the DataID is calculated according to [FO_PRS_CanTSyn_10007]
- and byte 2 is applied first, followed by the other bytes in ascending order, and the DataIDList last.

⌋

6.4.2.6.8 Set Freshness Value Field (*FV* and *FVL*)

Each Time Domain can be configured to include a Freshness Value (FV) into the Integrity Check Value (ICV) calculation. The FV refers to a monotonic counter that is used to ensure freshness of the timebase. Such a monotonic counter could be realized by means of individual message counters, called Freshness Counter, or by a time stamp value called Freshness Timestamp. How the Freshness Value is calculated in detail is out of scope of the protocol.

[FO_PRS_CanTSyn_10037] FVL and FV field, if FreshnessValue not supported

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and the parameter Cfg_TMFreshnessValueSupport is set to FALSE,

then the Time Master shall set the FVL field to 0.⌋

[FO_PRS_CanTSyn_10038] FVL and FV field, if Freshness Value generation failed

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and the parameter Cfg_TMFreshnessValueSupport is set to TRUE
- and the Freshness Value EXT_TMFreshnessValue could not be derived (refer EXT_TMEventFvCalcFailure),

then the Time Master shall set the FVL field to 0.⌋

[FO_PRS_CanTSyn_10039] FVL and FV field, if Freshness Value generation successful

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and the parameter Cfg_TMFreshnessValueSupport is set to TRUE
- and the Freshness Value EXT_TMFreshnessValue has been successfully derived (refer to EXT_TMEventFvCalcSuccess),

then the Time Master shall

- set the FVL field to the length of the value Freshness Value EXT_TMFreshnessValue as configured by the parameter Cfg_TMFreshnessValueLength
- set the FV field to the value of the Freshness Value (EXT_TMFreshnessValue)

⌋

6.4.2.6.9 Set Integrity Check Value Fields (*ICV* and *ICVL*)

The Integrity Check Value (ICV) refers to the result of a cryptographic operation, that is used to ensure that unauthorized modifications of a message are detected. A cryptographic operation can be either use an asymmetric algorithm calculating a signature of the messages or a symmetric algorithm calculating a message authentication code (MAC). However, the details of those cryptographic algotithms are out of scope and not standardized by the protocol. The ICV shall be calculated over the SYNC message payload, FUP message payload and (if configured) the Freshness Value FV.

The ICV is optinal and, if present, included in the FUP message.

[FO_PRS_CanTSyn_10040] ICV Calculation with Freshness Value

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and parameter Cfg_TMFreshnessValueSupport is set to TRUE
- and Freshness Value EXT_TMFreshnessValue has been successfully derived (EXT_TMEventFvCalcSuccess)
- and the FUP message bytes 0 .. N have been set,

then the Time Master shall calculate the ICV value (EXT_TMIntegrityCheckValue) by

- concatenating (in that order) the values
  - the SYNC message bytes 0 .. 7
  - and the FUP message bytes 0 .. N
- and passing this concatenated value as a bit array to the cryptographic algorithm configured by Cfg_TMIcvGenerationAlgorithm.

With

- N: index of the last ICV secured message byte as calculated according to table [FO_PRS_CanTSyn_10041]

⌋

[FO_PRS_CanTSyn_10041] Index of Last ICV Secured Time Synchroniziation Message Byte

Upstream requirements: RS_TS_20073

⌈

Configuration	Index of Last Secured Byte
Cfg_TMFreshnessValueSupport	
FALSE	9
TRUE	9 + (FVL + 7)/8

⌋

[FO_PRS_CanTSyn_10042] ICVL and ICV field, if ICV calculation failed

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and the generation of the ICV value has failed (refer to EXT_TMEventIcvCalcFailure),

then the Time Master shall set the ICVL field to 0.

In that case no ICV field is included in the FUP message.⌋

[FO_PRS_CanTSyn_10043] ICVL and ICV field, if ICV calculation successful

Upstream requirements: RS_TS_20073

⌈If

- Cfg_TMIcvSecured is ICV_SUPPORTED
- and the generation of the ICV value has been successful (refer to EXT_TMEventIcvCalcSuccess),

then the Time Master shall

- set the ICVL field to the value of the parameter Cfg_TMIcvLength
- set the ICV field to the value of the ICV as derived from the application (EXT_TMIntegrityCheckValue).

⌋

[FO_PRS_CanTSyn_10044] ICVL and ICV field, if FreshnessValue generation failed

Upstream requirements: RS_TS_20073

⌈If

- parameter Cfg_TMIcvSecured is ICV_SUPPORTED
- and the parameter Cfg_TMFreshnessValueSupport is set to TRUE
- and the generation of the FreshnessValue has failed (refer EXT_TMEventFvCalcFailure),

then the Time Master shall set the ICVL field to 0.

In that case no ICV field is included in the FUP message.⌋

[FO_PRS_CanTSyn_10045] Use of Message Authenication Code (MAC) for ICV calculation

Upstream requirements: RS_TS_20073

⌈If parameter Cfg_TMIcvGenerationAlgorithm is set to ICV_MAC for the Time Domain, then the Time Master shall generate a MAC for the message data to be secured as cryptographic algorithm for calculating the ICV.⌋

[FO_PRS_CanTSyn_10046] Use of Signature for ICV calculation

Upstream requirements: RS_TS_20073

⌈If parameter Cfg_TMIcvGenerationAlgorithm is set to ICV_SIGNATURE for the Time Domain, then the Time Master shall generate a digital signature for the message data to be secured as the cryptographic algorithm for calculating the ICV.⌋

6.4.2.6.10 Transition from TM_BuildFupMsg

[FO_PRS_CanTSyn_10047] Transition from state TM_BuildFupMsg

Upstream requirements: RS_TS_20035

⌈If

- in state TM_BuildFupMsg
- and all message fields of the FUP message (refer to chapter "Follow-up (FUP) Message Format") have been set,

then the Time Master shall

- send the FUP message on the CAN bus (refer to DLL_TransmitFupMsg)
- and transition for the given Time Domain to state TM_TransmitFupMsg.

⌋

6.4.2.7 State TM_TransmitFupMsg

6.4.2.7.1 Transition from TM_TransmitFupMsg

[FO_PRS_CanTSyn_10048] Transition from state TM_TransmitFupMsg

Upstream requirements: RS_TS_20035

⌈If

- in state TM_TransmitFupMsg
- and the FUP message transmission has completed (successfully or failed),

then the Time Master shall

- start the debounce interval $\Delta t_{\text{TxDebounceInterval}}$ (as configured by parameter Cfg_TMDebounceInterval)
- and transition for the given Time Domain to state TM_DebounceBeforeSyncMsg.

⌋

6.4.2.8 State TM_DebounceBeforeSyncMsg

6.4.2.8.1 Transition from TM_DebounceBeforeSyncMsg

[FO_PRS_CanTSyn_10049] FUP to SYNC Debounce Interval elapsed

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈If in state TM_DebounceBeforeSyncMsg, when the debounce interval $\Delta t_{\text{TxDebounceInterval}}$ (as configured by parameter Cfg_TMDebounceInterval) has elapsed, then the Time Master shall transition to the protocol state TM_Idle for the given Time Domain.⌋

6.4.3 Time Slave

Figure 6.5 illustrates the protocol state machine of the Time Slave.

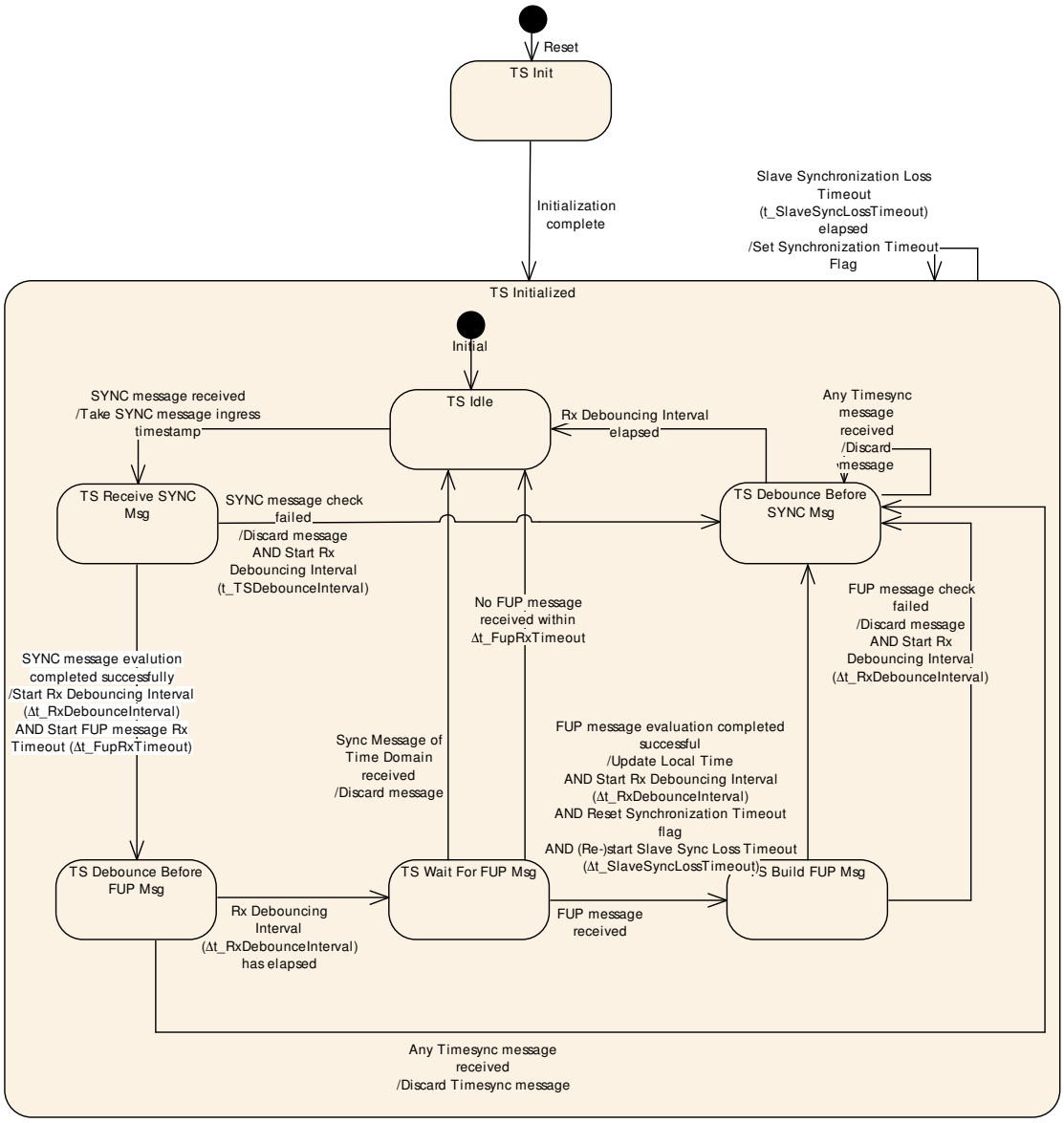

Figure 6.5: State Machine of a Time Slave

[FO_PRS_CanTSyn_10050] Slave Synchronization Lost Timeout elapsed

Upstream requirements: RS_TS_20035

⌈When the Slave Synchronization Lost Timeout ($\Delta t_{SlaveSyncLossTimeout}$) elapses, the Time Slave shall set the Synchronization Timeout flag to `TRUE`.⌋

Note: The Synchronization Timeout flag is reset to `FALSE` and the timeout $\Delta t_{SlaveSyncLossTimeout}$ is (re-)started when the Time Slave has received a valid FUP message. Refer to [FO_PRS_CanTSyn_10086]

6.4.3.1 State TS_Init

[FO_PRS_CanTSyn_00007] Initialize Sequence Counter of Time Slave

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00007

⌈If in state `TS_Init`, then the Time Slave shall initialize

- the Sequence Counter value for the given Time Domain with 0
- and the Synchronization Timeout flag to `FALSE`.

⌋

6.4.3.1.1 Transition from `TS_Init`

[FO_PRS_CanTSyn_10051] Transition from state `TS_Init`

Upstream requirements: RS_TS_20031, RS_TS_20035

⌈When initialization (`TS_Init`) is completed, the Time Slave shall go to state `TS_Idle`.⌋

6.4.3.2 State TS_Idle

In state `TS_Idle` the Time Slave waits for the reception of the next SYNC message on the bus. The Time Slave is in this state after initialization and returns to this state when a Time Synchronization sequence has been completed.

6.4.3.2.1 Transition from `TS_Idle`

[FO_PRS_CanTSyn_00144] Timestamping of SYNC message ingress

Upstream requirements: RS_TS_20035

⌈If in state `TS_Idle`, when a SYNC message, i.e., a Time Synchronization Message with Message Type `0x10` or `0x20`, is received, then the Time Slave shall

- take the ingress timestamp $T_{SyncIngress}$
- and transition to state `TS_ReceiveSyncMsg`.

⌋

Note: Sampling of the SYNC ingress timestamp $T_{SyncIngress}$ is done either by SW or by reading it from the HW. If the ingress timestamp $T_{SyncIngress}$ needs to be taken by SW (SW timestamping), this is highly time critical. The timestamp should to be taken in the interrupt service routine of the CAN driver for the CAN Receive interrupt. Any

delay between the actual ingress of the SYNC message and taking the timestamp in SW will worsen the precision of the time synchronization.

6.4.3.3 State TS ReceiveSyncMsg

Figure 6.6 illustrates the flow of actions after receiving a SYNC message.

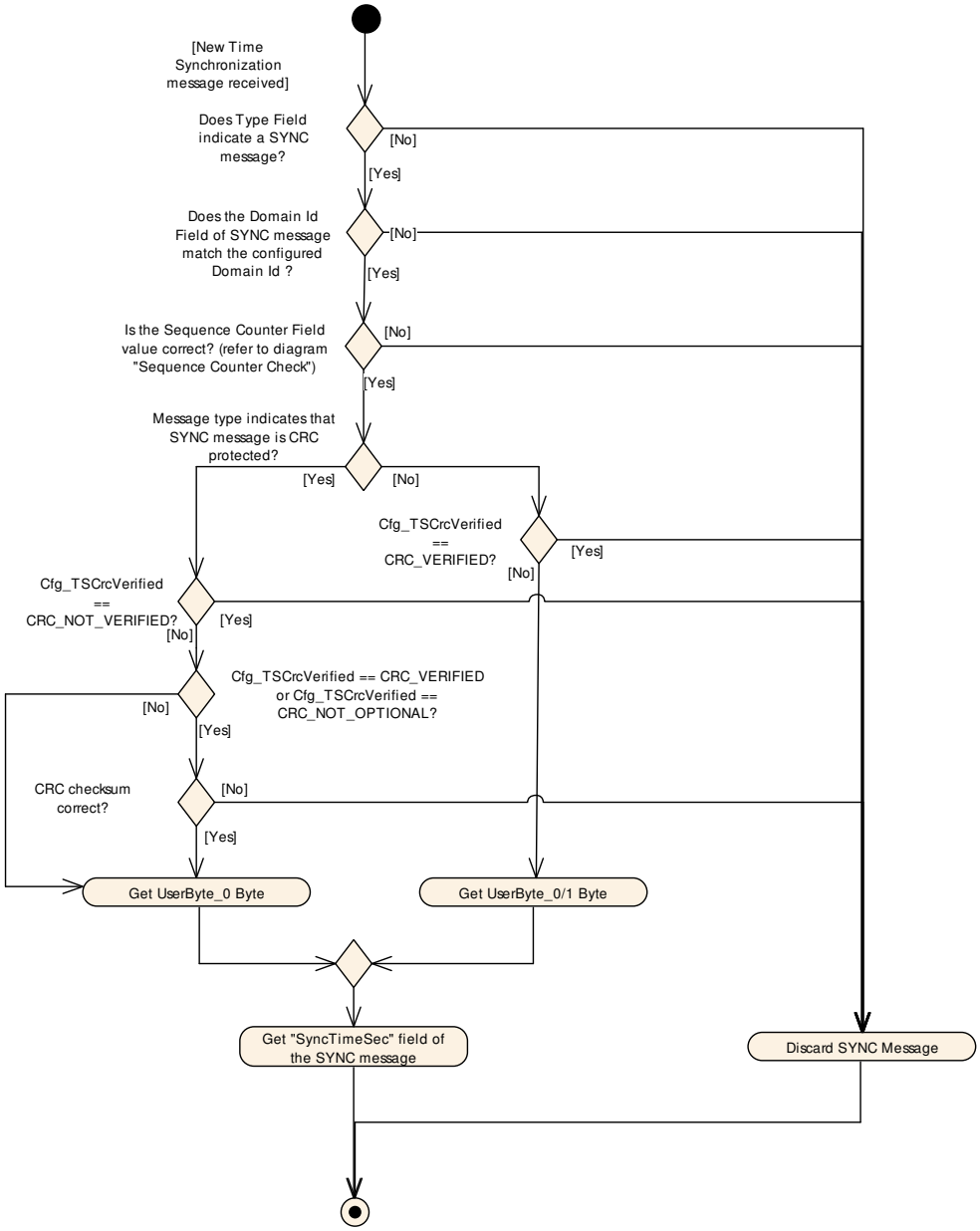

Figure 6.6: Flow of actions after receiving a SYNC message

6.4.3.3.1 Check SYNC Message Type Field (*Type*)

Note: [FO_PRS_CanTSyn_00144] already checks the Type field of the received SYNC message to be equal to `0x10` or `0x20`. Otherwise, the Time Slave would not transition to state `TS_ReceiveSyncMsg`.

6.4.3.3.2 Check SYNC Message Domain Id Field (*D*)

[FO_PRS_CanTSyn_10052] Check of Time Domain Id Field of SYNC message

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00086

⌈If the value of the `D` field of the received SYNC message does not match the Time Domain `Cfg_TDDomainId`, that is configured for the received SYNC message with the CAN ID (as given by `Cfg_TDCanId`), then the Time Slave shall discard the SYNC message.⌋

6.4.3.3.3 Check SYNC Message Sequence Counter Field (*SC*)

Figure 6.7 illustrates the Sequence Counter check of a Time Slave for SYNC messages.

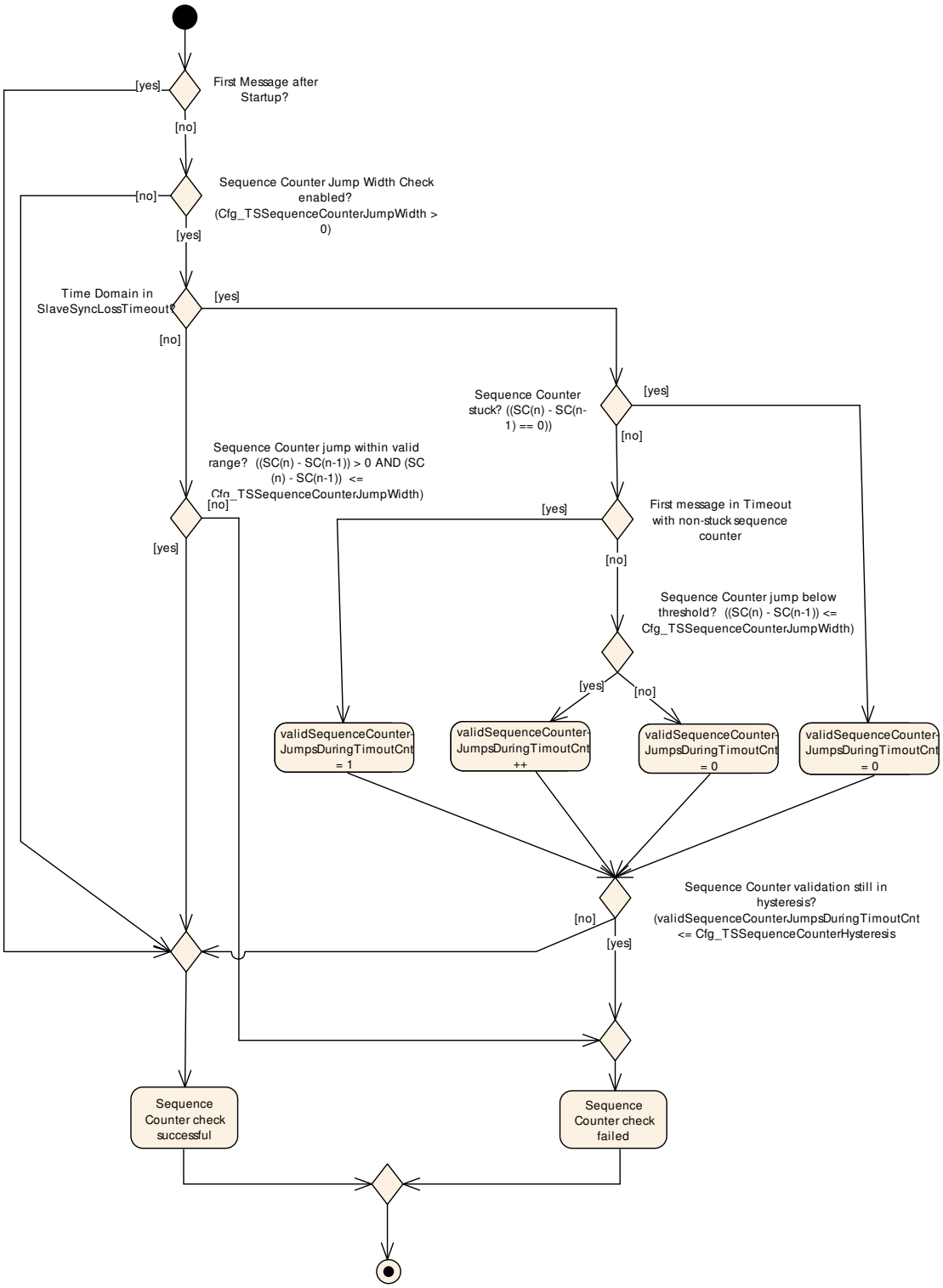

Figure 6.7: Sequence Counter Check

[FO_PRS_CanTSyn_10053] SYNC message Sequence Counter check, if not in timeout

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00221, SWS_CanTSyn_00222, SWS_CanTSyn_00223

⌈If

- the Sequence Counter check is enabled (i.e., Cfg_TSSequenceCounterJumpWidth > 0)
- and the Time Domain is not in Synchronization Lost Timeout $\Delta t_{SlaveSyncLossTimeout}$,
- and the received SYNC message is not the first SYNC message after startup
- and the difference between the Sequence Counter value (i.e., the value of the SC Field) of the SYNC message and the Sequence Counter value of the previous SYNC message is 0 or greater than Cfg_TSSequenceCounterJumpWidth

then a Time Slave shall discard the SYNC message.⌋

6.4.3.3.3.1 Sequence Counter Hysteresis

This chapter specifies how to apply an optional hysteresis (Cfg_TSSequenceCounterHysteresis) when checking the Sequence Counter of a received SYNC message. This hysteresis applies only while the Time Slave in Synchronization Lost Timeout $\Delta t_{SlaveSyncLossTimeout}$ for the given Time Domain.

While in Synchronization Lost Timeout $\Delta t_{SlaveSyncLossTimeout}$ the Time Slave checks each time a SYNC message is received the difference between the Sequence Counter value (i.e., the value of the SC Field) of the current message and the Sequence Counter value of the previous SYNC message. If the difference value is greater than 0 and less or equal than Cfg_TSSequenceCounterJumpWidth, then a Time Slave will increment the validSequenceCounterJumpsDuringTimeoutCnt.

While validSequenceCounterJumpsDuringTimeoutCnt has not reached the threshold Cfg_TSSequenceCounterHysteresis, the Time Slave will consider the Sequence Counter check as failed and will discard the SYNC message. That means, only if Cfg_TSSequenceCounterHysteresis consecutive SYNC messages with a correct SC Field have been received, the Time Slave will pass the Sequence Counter check and no longer discard the SYNC message.

The hysteresis improves robustness against a scenario with a buggy master implementation or injection of invalid master messages, i.e., when the Sequence Counter increments by more than Cfg_TSSequenceCounterJumpWidth. In such a scenario (without any hysteresis) a message with any (also invalid) Sequence Counter value would cause the Time Slave to leave the Timeout state although the Sequence Counter is not incremented correctly. A hysteresis avoids this.

[FO_PRS_CanTSyn_00224] Message with stuck Sequence Counter in Hysteresis

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00224

⌈If

- Sequence Counter check is enabled (i.e., `Cfg_TSSequenceCounterJumpWidth` > 0)
- and the Time Domain is in Synchronization Lost Timeout $\Delta t_{\texttt{SlaveSyncLossTimeout}}$
- and the Sequence Counter of the received SYNC message is `stuck`,

then a Time Slave shall reset the `validSequenceCounterJumpsDuringTimeoutCnt` to 0.⌋

[FO_PRS_CanTSyn_00225] First message with non-stuck Sequence Counter in Hysteresis

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00225

⌈If

- Sequence Counter check is enabled (i.e., `Cfg_TSSequenceCounterJumpWidth` > 0)
- and the Time Domain is in Synchronization Lost Timeout $\Delta t_{\texttt{SlaveSyncLossTimeout}}$
- and the received message is the first SYNC message in Synchronization Lost Timeout for which the Sequence Counter is not `stuck`,

then a Time Slave shall set the `validSequenceCounterJumpsDuringTimeoutCnt` to 1.⌋

Rationale: After a Timeout (e.g., due to a reset or disconnect of the Time Master) it is very likely that the Sequence Counter of the first received Timesync message is out of sync, i.e., the Sequence Counter difference exceeds `Cfg_TSSequenceCounterJumpWidth`. To allow for faster re-synchronization of the Sequence Counter to the Time Master, the Sequence Counter of the first Timesync message is not checked for `Cfg_TSSequenceCounterJumpWidth`. However, a stuck Sequence Counter will always, i.e., also in this situation, be considered as invalid (refer to [FO_PRS_CanTSyn_00224]).

[FO_PRS_CanTSyn_00226] Not first message with non-stuck Sequence Counter in Hysteresis

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00226

⌈If

- Sequence Counter check is enabled (i.e., `Cfg_TSSequenceCounterJumpWidth` > 0)

- and the Time Domain is in Synchronization Lost Timeout $\Delta t_{SlaveSyncLossTimeout}$
- and the Sequence Counter of the received SYNC message is not `stuck`
- and the received SYNC message is not the first SYNC message in the Synchronization Lost Timeout for which the Sequence Counter is not `stuck`

then a Time Slave shall check if value of the difference between the Sequence Counter of the current message and the Sequence Counter of the previous SYNC message exceeds the threshold `Cfg_TSSequenceCounterJumpWidth`.

If the difference value exceeds the threshold `Cfg_TSSequenceCounterJumpWidth`, then the Time Slave shall increment the `validSequenceCounterJumpsDuringTimeoutCnt` by 1, else the Time Slave shall reset the `validSequenceCounterJumpsDuringTimeoutCnt` to 0.⌋

[FO_PRS_CanTSyn_00227] Sequence Counter hysteresis check

Upstream requirements: RS_TS_20033, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00227

⌈If

- Sequence Counter check is enabled (i.e., `Cfg_TSSequenceCounterJumpWidth` > 0)
- and the Time Domain is in Synchronization Lost Timeout $\Delta t_{SlaveSyncLossTimeout}$

when the Time Slave has updated the `validSequenceCounterJumpsDuringTimeoutCnt` then the Time Slave shall check `validSequenceCounterJumpsDuringTimeoutCnt`.

If the number of consecutive valid Sequence Counter jumps `validSequenceCounterJumpsDuringTimeoutCnt` is equal or less than the value `Cfg_TSSequenceCounterHysteresis`, then the Time Slave shall discard the SYNC message.⌋

6.4.3.3.4 Check SYNC Message CRC Field (*CRC*)

[FO_PRS_CanTSyn_10054] Discard not CRC protected SYNC messages in CRC_VERIFIED case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00058, SWS_CanTSyn_00059, SWS_CanTSyn_00109

⌈If

- Time Slave is in state `TS_ReceiveSyncMsg`
- and the `Type` of the received Time Synchronization message is `0x10`
- and `Cfg_TSCrcVerified` is set to `CRC_VERIFIED`

then the Time Slave shall discard the Time Synchronization message.⌋

[FO_PRS_CanTSyn_10055] Discard CRC protected SYNC messages in CRC_NOT_VERIFIED case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00057, SWS_CanTSyn_00059, SWS_CanTSyn_00109

⌈If

- Time Slave is in state `TS_ReceiveSyncMsg`
- and the `Type` of the received Time Synchronization message is `0x20`
- and `Cfg_TSCrcVerified` is set to `CRC_NOT_VERIFIED`

then the Time Slave discard the Time Synchronization message.⌋

[FO_PRS_CanTSyn_10056] Check of CRC protected SYNC message in CRC_VERIFIED and CRC_OPTIONAL case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00084, SWS_CanTSyn_00085

⌈If

- the Time Slave is in state `TS_ReceiveSyncMsg` for the reveived Time Domain
- and the `Type` of the received Time Synchronization message is equal to `0x20`
- and the parameter `Cfg_TSCrcVerified` is set to `CRC_VERIFIED` or `CRC_OPTIONAL`

then the Time Slave shall calculate the CRC of a received SYNC message according to [FO_PRS_CanTSyn_10005] over bytes 2 to 7 and the `DataID`, where

- the `DataID` is calculated according to [FO_PRS_CanTSyn_10006]
- and byte 2 is applied first, followed by the other bytes in ascending order, and the `DataID` last.

If the calculated CRC is not equal to the value of the `CRC` Field of the SYNC message, the Time Slave shall discard the Time Synchronization message.⌋

6.4.3.3.5 Get SYNC Message Seconds Field (`SyncTimeSec`)

[FO_PRS_CanTSyn_10057] Reading `SyncTimeSec` field

Upstream requirements: RS_TS_20035

⌈If the Time Slave is in state `TS_ReceiveSyncMsg`, then the Time Slave shall set $T_{TSGlobalTime_Sec}$ to the value of the `SyncTimeSec` field of the received SYNC message.⌋

6.4.3.3.6 Get SYNC Message User Data Fields (UserData_0/_1)

[FO_PRS_CanTSyn_10058] Setting APP_TSUserData[0]/[1] for non-CRC protected SYNC message

Upstream requirements: RS_TS_20037

⌈

If

- the Time Slave is in state TS_ReceiveSyncMsg
- and the value of the Type field of the received Time Synchronization message is 0x10,

then the Time Slave shall set

- APP_TSUserData[0] to the value of the UserByte_0 field
- and APP_TSUserData[1] to the value of the UserByte_1 field.

⌋

[FO_PRS_CanTSyn_10059] Setting APP_TSUserData[0]/[1] for CRC protected SYNC message

Upstream requirements: RS_TS_20037

⌈If

- the Time Slave is in state TS_ReceiveSyncMsg
- and the value of the Type field of the received Time Synchronization message is 0x20,

then the Time Slave shall set

- set APP_TSUserData[0] to the value of the UserByte_0 field
- set APP_TSUserData[1] to 0.

⌋

6.4.3.3.7 Transition from TS_ReceiveSyncMsg

[FO_PRS_CanTSyn_10060] SYNC Message Verification Failure

Upstream requirements: RS_TS_20035

⌈If in state TS_ReceiveSyncMsg for a given Time Domain when the SYNC message check has failed (i.e., message has been discarded), then the Time Slave shall transition to state TS_Idle⌋

[FO_PRS_CanTSyn_10061] SYNC Message Verification Successful

Upstream requirements: RS_TS_20035

⌈If in state TS_ReceiveSyncMsg for a given Time Domain when the SYNC message has been processed successfully (i.e., message has not been discarded), then the Time Slave shall

- Start Rx Debouncing Interval $\Delta t_{\texttt{RxDebounceInterval}}$
- and transition to state TS_DebounceBeforeFupMsg

⌋

6.4.3.4 State TS_DebounceBeforeFupMsg

[FO_PRS_CanTSyn_10062] Discard Time Synchronization Message while debouncing after SYNC message

Upstream requirements: RS_TS_20035, RS_TS_20073
Previous identifiers: SWS_CanTSyn_00183

⌈If in state TS_DebounceBeforeFupMsg for a given Time Domain when any Time Synchronization message is received for the Time Domain, then the Time Slave shall

- discard that message.
- and transition to state TS_DebounceBeforeSyncMsg

⌋

Rationale: The inhibit time $\Delta t_{\texttt{RxDebounceInterval}}$ is to improve robustness against message sequence errors, specifically injection of fake SYNC messages by an attacker. Note that this will not allow to filter out all possible fake SYNC scenarios.

Note: After receiving the first Time Synchronization message while in $\Delta t_{\texttt{RxDebounceInterval}}$ the Time Slave will not transition to state TS_Idle immediately, but will wait in TS_DebounceBeforeSyncMsg until the $\Delta t_{\texttt{RxDebounceInterval}}$ has elapsed.

[FO_PRS_CanTSyn_10063] Debouncing successful after SYNC message

Upstream requirements: RS_TS_20035, RS_TS_20073
Previous identifiers: SWS_CanTSyn_00183

⌈If in state TS_DebounceBeforeFupMsg for a given Time Domain when Rx Debouncing Interval $\Delta t_{\texttt{RxDebounceInterval}}$ has elapsed, then the Time Slave shall transition to state TS_WaitForFupMsg⌋

6.4.3.5 State TS_WaitForFupMsg

[FO_PRS_CanTSyn_00182] SYNC Message received while waiting for FUP message

Upstream requirements: RS_TS_20035, RS_TS_20073

⌈If in state `TS_WaitForFupMsg` for a given Time Domain when a SYNC message, i.e., a Time Synchronization Message with Message Type `0x10` or `0x20`, is received for that Time Domain,

- discard that SYNC message.
- and transition to state `TS_Idle`

⌋

[FO_PRS_CanTSyn_00063] FUP message not received in time

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00063

⌈If in state `TS_WaitForFupMsg` for a given Time Domain when the FUP message reception timeout $\Delta t_{\texttt{FupRxTimeout}}$ has elapsed, then the Time Slave shall transition to state `TS_Idle`.⌋

[FO_PRS_CanTSyn_10064] FUP message received in time

Upstream requirements: RS_TS_20035

⌈If in state `TS_WaitForFupMsg` for a given Time Domain when a FUP message, i.e., a Time Synchronization Message with Message Type `0x18`,`0x28`,`0x78` or `0x88`, is received for that Time Domain, then the Time Slave shall transition to state `TS_ReceiveFupMsg`.⌋

6.4.3.6 State TS_ReceiveFupMsg

Figure 6.8 illustrates the flow of actions after receiving a FUP message.

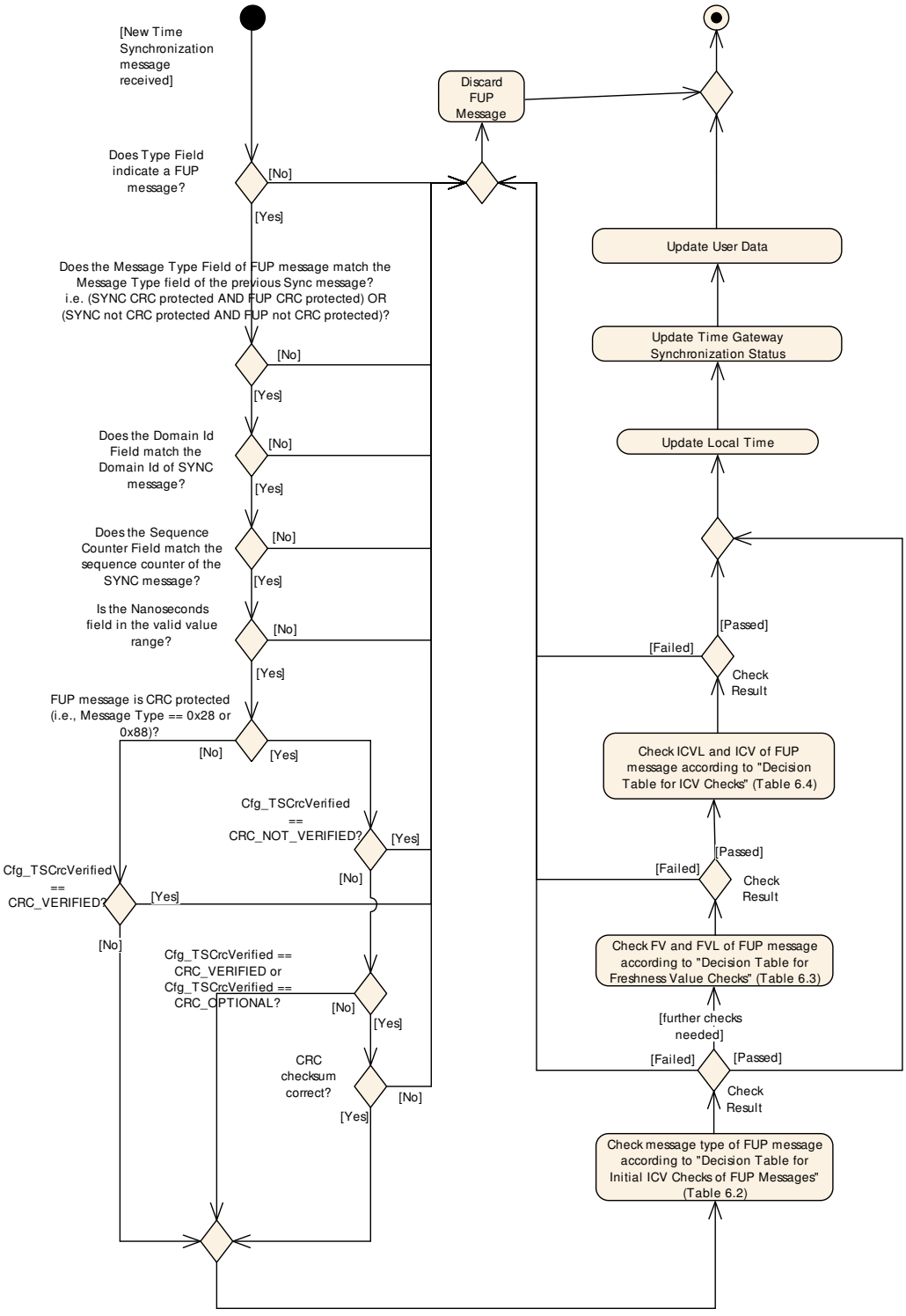

Figure 6.8: Flow of actions after receiving a FUP message

6.4.3.6.1 Check FUP Message Type Field (*Type*)

Note: [FO_PRS_CanTSyn_10064] already checks the Type field of the received Time Synchronization message to be equal to `0x18`, `0x28`, `0x78` or `0x88`, i.e., if it is a FUP

message. Otherwise, the Time Slave would not transition to state TS_ReceiveSyncMsg. Additionally, the Time Slave shall check the following conditions:

[FO_PRS_CanTSyn_10065] Check of CRC protected FUP message

Upstream requirements: RS_TS_20033

⌈If for a given CAN ID

- the SYNC message is not CRC protected (i.e., Message Type 0x10)
- and the corresponding FUP message is CRC protected (i.e., Message Type: 0x28 or 0x88)

then the Time Slave shall discard the FUP message.⌋

[FO_PRS_CanTSyn_10066] Check of not CRC protected FUP message

Upstream requirements: RS_TS_20033

⌈If for a given CAN ID

- the SYNC message is CRC protected (i.e., Message Type 0x20)
- and the corresponding FUP message is not CRC protected (i.e., Message Type: 0x18 or 0x78)

then the Time Slave shall discard the FUP message.⌋

6.4.3.6.2 Check FUP Message Domain Id Field (*D*)

[FO_PRS_CanTSyn_10067] Check of the Time Domain Id Field of a FUP message

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00086

⌈If the value of the D Field of the received FUP message does not match the Time Domain of the previous SYNC message received for the same CAN ID (given by Cfg_TDCanId), then the Time Slave shall discard the FUP message.⌋

6.4.3.6.3 Check FUP Message Sequence Counter Field (*SC*)

[FO_PRS_CanTSyn_00076] FUP Message Sequence Counter check failure

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00076

⌈If the value of the SC Field of the received FUP message does not match to the Sequence Counter value of the previous SYNC message of the same Time Domain, then the Time Slave shall discard the FUP message.⌋

6.4.3.6.4 Check FUP Message Nanoseconds Overflow Field (*OVS*)

[FO_PRS_CanTSyn_10068] Range check of `SyncTimeNSec` value

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00086

⌈If the value of the `SyncTimeNSec` field exceeds 999999999 ns, then the Time Slave shall discard the FUP message.⌋

6.4.3.6.5 Check FUP Message CRC Field (*CRC*)

[FO_PRS_CanTSyn_10069] CRC check of not CRC protected FUP message in CRC_VERIFIED case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00061, SWS_CanTSyn_00062, SWS_CanTSyn_00110, SWS_CanTSyn_00181

⌈If

- Time Slave is in state `TS_ReceiveFupMsg` for a given Time Domain
- and the `Type` of the received FUP message is `0x18` or `0x78` (i.e., FUP message without CRC)
- and `Cfg_TSCrcVerified` is `CRC_VERIFIED` for that Time Domain

then the Time Slave shall discard the received FUP message.⌋

[FO_PRS_CanTSyn_10070] CRC check of CRC protected FUP message in CRC_NOT_VERIFIED case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00060, SWS_CanTSyn_00062, SWS_CanTSyn_00110, SWS_CanTSyn_00181

⌈If

- Time Slave is in state `TS_ReceiveFupMsg`
- and the `Type` of the received FUP message is `0x28` or `0x88` (i.e., FUP message with CRC)
- and `Cfg_TSCrcVerified` is set to `CRC_NOT_VERIFIED`

then the Time Slave discard the received FUP message.⌋

[FO_PRS_CanTSyn_10071] CRC check of CRC protected FUP message in CRC_VERIFIED and CRC_OPTIONAL case

Upstream requirements: RS_TS_20033
Previous identifiers: SWS_CanTSyn_00084, SWS_CanTSyn_00085

⌈If

- Time Slave in state TS_ReceiveFupMsg
- and the Type of the received FUP message is 0x28 or 0x88 (i.e., FUP message with CRC)
- and parameter Cfg_TSCrcVerified is set to CRC_VERIFIED or CRC_OPTIONAL

then the Time Slave shall calculate the CRC of a received SYNC message according to [FO_PRS_CanTSyn_10005] over bytes 2 to 7 and the DataID, where

- the DataID is calculated according to [FO_PRS_CanTSyn_10007]
- and byte 2 is applied first, followed by the other bytes in ascending order, and the DataID last.

If the calculated CRC is not equal to the value of the CRC Field of the FUP message, the Time Slave shall discard the received FUP message.⌋

6.4.3.6.6 ICV Secured FUP Messages

Table 6.2 lists the initial checks related to ICV secured FUP messages based on the message type and the parameter Cfg_TSIcvVerified to check if the FUP message is valid or not. If the check fails the FUP message will be discarded. Otherwise the FUP message will be further processed (refer to chapter "Update of Time Slave").

Only rule 7 in Table 6.2 requires additional processing of the FVL, FV, ICVL and ICV fields as specified in chapter "Check Freshness Value Fields (*FV* and *FVL*)" and "Check ICV Value Fields (*ICV* and *ICVL*)" before it can be decided, if the FUP message is valid or not. All other checks given in Table 6.2 can be done without processing these fields.

Rule Id	FUP Message Type	Cfg_TSIcvVerified	Check Result	Further Processing
1	w/o ICV	ICV_IGNORED	pass	refer to chapter "Update of Time Slave"
2		ICV_NOT_VERIFIED	pass	refer to chapter "Update of Time Slave"
3		ICV_VERIFIED	fail	[FO_PRS_CanTSyn_10072]
4		ICV_OPTIONAL	pass	refer to chapter "Update of Time Slave"
5	with ICV	ICV_IGNORED	pass	refer to chapter"Update of Time Slave"
6		ICV_NOT_VERIFIED	fail	[FO_PRS_CanTSyn_10073]
7		ICV_VERIFIED	conditional	refer to chapters 6.4.3.6.6.1 and 6.4.3.6.6.2
		ICV_OPTIONAL		

Table 6.2: Decision Table for Initial ICV Checks of FUP Messages

[FO_PRS_CanTSyn_10072] ICV Check of not authenticated FUP message in ICV_VERIFIED case

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00086, SWS_CanTSyn_00178

⌈If

- the Time Slave is in state TS_ReceiveFupMsg for a given Time Domain
- and the Type of the received FUP message is 0x18 or 0x28 (i.e., FUP message without ICV)
- and the parameter Cfg_TSIcvVerified is set to ICV_VERIFIED for that Time Domain

then the Time Slave shall discard the received FUP message.⌋

[FO_PRS_CanTSyn_10073] ICV Check of authenticated FUP message in ICV_NOT_VERIFIED case

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00086, SWS_CanTSyn_00179

⌈If

- the Time Slave is in state TS_ReceiveFupMsg for a given Time Domain
- and the Type of the received FUP message is set to 0x78 or 0x88 (i.e., FUP message with ICV)
- and the parameter Cfg_TSIcvVerified is set to ICV_NOT_VERIFIED for that Time Domain

then the Time Slave shall discard the received FUP message.⌋

6.4.3.6.6.1 Check Freshness Value Fields (*FV* and *FVL*)

Table 6.3 summarizes the checks related to the FVL and FV fields of ICV secured FUP message. The checks are only applicable if the configuration parameter Cfg_TSIcv-Verified is set to ICV_VERIFIED or ICV_OPTIONAL for that Time Domain. If the check fails the FUP message will be discarded. Otherwise the Freshness Value will be further processed (refer to "Check ICV Value Fields (*ICV* and *ICVL*)").

Rule Id	FVL	Cfg_TSFreshness-ValueLength	FreshnessValue from FVM	Check Result	Further Processing
1	don't care	<>FVL	don't care	fail	[FO_PRS_CanTSyn_10074]
2	0	==FVL	don't care	pass	refer to chapter 6.4.3.6.6.2
3	<>0	==FVL	==FV	pass	refer to chapter 6.4.3.6.6.2
4			<>FV	fail	[FO_PRS_CanTSyn_10076] [FO_PRS_CanTSyn_10077]

Table 6.3: Decision Table for Freshness Value Checks

[FO_PRS_CanTSyn_10074] FVL does not match configuration Freshness Value Length

Upstream requirements: RS_TS_20073

⌈If

- the `Type` of the received FUP message is `0x78` or `0x88` (i.e., FUP message with ICV)
- and `Cfg_TSIcvVerified` is `ICV_VERIFIED` or `ICV_OPTIONAL`
- and the value of the `FVL` field of the received FUP message is not equal to the configured Freshness Value Length `Cfg_TSFreshnessValueLength`

then the Time Slave shall discard the received FUP message.⌋

[FO_PRS_CanTSyn_10075] Precoditions for Freshness Value Check

Upstream requirements: RS_TS_20073

⌈If

- the `Type` of the received FUP message is `0x78` or `0x88` (i.e., FUP message with ICV)
- and `Cfg_TSIcvVerified` is `ICV_VERIFIED` or `ICV_OPTIONAL`
- and the value of the `FVL` field of the received FUP message
  - is greater than 0
  - and is equal to configured Freshness Value Length `Cfg_TSFreshness-ValueLength`,

then the Time Slave shall derive the Freshness Value `EXT_TSFreshnessValue` from the Freshness Value Manager (FVM).⌋

Note: The Freshness Value Manager (FVM) is an external component which is not part of the Time Slave. The FVM is responsible to manage and provide the Freshness Value `EXT_TSFreshnessValue` for a given Time Domain. The way how the FVM derives the Freshness Value is out of scope of this document. Details on how the Freshness Value is could be derived can be found in Annex A in [7, Specification of Secure Onboard Communication].

[FO_PRS_CanTSyn_10076] FV does not match calculated Freshness Value

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00178, SWS_CanTSyn_00181

⌈If

- the Freshness Value `EXT_TSFreshnessValue` needs to be derived (refer to [FO_PRS_CanTSyn_10075])
- and the `FV` field of the reveived FUP message is not equal to the Freshness Value `EXT_TSFreshnessValue` retrieved from the Freshness Value Manager,

then the Time Slave shall discard the received FUP message⌋

[FO_PRS_CanTSyn_10077] Failure to derive Freshness Value

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00178, SWS_CanTSyn_00181

⌈If

- the Freshness Value `EXT_TSFreshnessValue` needs to be derived (refer to [FO_PRS_CanTSyn_10075])
- but it cannot be retrieved from the Freshness Value Manager,

then the Time Slave shall discard the received FUP message.⌋

6.4.3.6.6.2 Check ICV Value Fields (*ICV* and *ICVL*)

Table 6.4 summarizes the checks related to the message fields `ICVL` and `ICV` of ICV secured messages. These checks are only applicable if the configuration parameter `Cfg_TSIcvVerified` is set to `ICV_VERIFIED` or `ICV_OPTIONAL` for that Time Domain. Precondition is, that Freshness Value Check according to chapter "Check Freshness Value Fields (*FV* and *FVL*)" has been performed and has passed. If the `ICV` check fails the FUP message will be discarded. Otherwise the FUP message will be further processed (refer to chapter "Update of Time Slave").

Rule Id	`ICVL`	`Cfg_TSIcvLength`	Calculated `ICV` Value	Check Result	Further Processing
1	0	==`ICVL`	n/a	pass	refer to chapter "Update of Time Slave"
2	0	<>`ICVL`	don't care	fail	[FO_PRS_CanTSyn_10078]
3	<>0	==`ICVL`	<>`ICV`	fail	[FO_PRS_CanTSyn_10082]
4			==`ICV`	pass	refer to chapter "Update of Time Slave"
5	<>0	<>`ICVL`	don't care	fail	[FO_PRS_CanTSyn_10078]

Table 6.4: Decision Table for ICV Checks

[FO_PRS_CanTSyn_10078] ICV Value Length does not match configuration ICV Value Length

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00086, SWS_CanTSyn_00181

⌈If

- the `Type` of the received FUP message is `0x78` or `0x88` (i.e., FUP message with ICV)
- and `Cfg_TSIcvVerified` is `ICV_VERIFIED` or `ICV_OPTIONAL`
- and the value of the `ICVL` field of the received FUP message is not equal to the configured ICV Value Length `Cfg_TSIcvLength`,

then the Time Slave shall discard the FUP message.⌋

[FO_PRS_CanTSyn_10079] Precoditions for ICV Value Check

Upstream requirements: RS_TS_20073

⌈If

- the `Type` of the received FUP message is `0x78` or `0x88` (i.e., FUP message with ICV)
- and `Cfg_TSIcvVerified` is `ICV_VERIFIED` or `ICV_OPTIONAL`
- and the value of the `ICVL` field of the received FUP message
  - is greater than 0
  - and is equal to configured ICV Value Length `Cfg_TSIcvLength`,

then the Time Slave shall calculate the `ICV`.⌋

[FO_PRS_CanTSyn_10080] ICV Calculation with Freshness Value

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00086

⌈If

- the `ICV` needs to be calculated (refer to [FO_PRS_CanTSyn_10079])
- and the value of the `FVL` field of the received FUP message
  - is greater than 0
  - and is equal to configured Freshness Value Length `Cfg_TSFreshness-ValueLength`
- and the Freshness Value `EXT_TSFreshnessValue` derived from the Freshness Value Manager is equal to the `FV` field of the received FUP message,

then the Time Slave shall calculate the `ICV` value by

- concatenating (in that order) the values
  - of the bytes 0 .. 7 of the last valid `SYNC` message,
  - and of the bytes 0 .. N of the received `FUP` message
- and passing this concatenated value as a bit array to the cryptographic algorithm configured by `Cfg_TSIcvVerificationAlgorithm`.

With:

- N: index of the last ICV secured message byte calculated as N = 9 + (`FVL`+7)/8

⌋

[FO_PRS_CanTSyn_10081] ICV Calculation with Freshness Value

Upstream requirements: RS_TS_20073
Previous identifiers: SWS_CanTSyn_00086

⌈If

- the ICV needs to be calculated (refer to [FO_PRS_CanTSyn_10079])
- and the value of the FVL field of the received FUP message
  - is equal to 0
  - and is equal to configured Freshness Value Length Cfg_TSFreshnessValueLength

then the Time Slave shall calculate the ICV value by

- concatenating (in that order) the values
  - of the bytes 0 .. 7 of the last valid SYNC message,
  - and of the bytes 0 .. 9 of the received FUP message
- and passing this concatenated value as a bit array to the cryptographic algorithm configured by Cfg_TSIcvVerificationAlgorithm.

⌋

[FO_PRS_CanTSyn_00194] Message Authentication Code (MAC) as ICV value

Upstream requirements: RS_TS_20073

⌈If Cfg_TSIcvVerificationAlgorithm for the Time Domain is configured to ICV_MAC, then the Time Slave shall calculate a Message Authentication Code (MAC) of the message as the ICV value.⌋

[FO_PRS_CanTSyn_00195] Digital Signature as ICV value

Upstream requirements: RS_TS_20073

⌈If Cfg_TSIcvVerificationAlgorithm for the Time Domain is configured to ICV_SIGNATURE, then the Time Slave shall calculate a Digital Signature of the message as the ICV value.⌋

[FO_PRS_CanTSyn_10082] Discard FUP message if ICV could not be derived

Upstream requirements: RS_TS_20073

⌈If

- the ICV needs to be calculated (refer to [FO_PRS_CanTSyn_10079])
  - but the ICV calculation fails
  - or the calculated ICV value is not equal to the ICV field,

then the Time Slave shall discard the FUP message.⌋

6.4.3.6.7 Update of Time Slave

The Time Slave will calculate its new Synchronized Time $T_{TSGlobalTime}$ and related user/status data, only if all checks for the FUP message fields have been successfully passed and the FUP did not get discarded (refer to previous chapters).

As shown in Figure 6.1 upon reception of the FUP message the Time Slave calculates its new Synchronized Time $t_{TSGlobalTime}$ as the sum of

- the Time Master's time $T_{TMGlobalTime}$ received in the two messages, i.e.,
  - the timestamp within the SYNC message $T_{TMGlobalTime_Sec}$
  - and the time offset within the FUP message $\Delta t_{SyncTxOffset}$
- and the time $\Delta t_{LinkDelay}$ that the SYNC message needs to travel from the Time Master to the Time Slave
- and the offset $\Delta t_{SlaveRxOffset}$ that has elapsed from ingress of the received SYNC message ($T_{SyncIngress}$) to the current point in time $T_{TSLocalTime_ReSync}$ when the Time Slave's time is actually synchronized.

In the simplest case this correction would lead to a hard jump in the Time Slave's Synchronized Time $t_{TSGlobalTime}$ at the point in time $T_{TSLocalTime_ReSync}$ when the correction is applied. Alternatively, the Time Slave may implement a smooth time correction, i.e., a gradual adjustment of its Local Time to the new Synchronized Time over some time interval starting at ($T_{TSLocalTime_ReSync}$).

Also note, since the Time Master's clock and the Time Slave's clock run at slightly different rates, the Time Slave needs to account for that rate difference when applying the correction. So, a rate correction factor r_{Slave} should be applied to the offset $\Delta t_{SlaveRxOffset}$.

Note: Measuring that rate deviation $\Delta t_{SlaveRxOffset}$ or specifying the algorithm for smoothing out the hard jump in $t_{TSGlobalTime}$ are out of scope of this specification.

[FO_PRS_CanTSyn_00087] Update Local Time and User Data of Time Slave
Upstream requirements: RS_TS_20032, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00087

⌈If

- the Time Slave is in state `TS_ReceiveFupMsg`
- and all checks for the FUP message fields succeeded, i.e., the received FUP message did not get discarded,

then the Time Slave shall:

- calculate Global Time of the Time Slave $t_{TSGlobalTime}$ from the time values received in the SYNC and FUP messages according to (refer to [FO_PRS_CanTSyn_10083])

- and set its APP_TSUserData and APP_TSUserDataLength
- and set its APP_TSGatewayStatus (refer to [FO_PRS_CanTSyn_00133])

"in a consistent way".⌋

Note: *"immediately"* implies that step 2 above is time critical. Any delay between sampling the Local Time (step 1) and setting the Local Time to its new value (step 2) will worsen the precision of the Local Time. *"in a consistent way"* is typically achieved by setting the data in "atomic", i.e., "non-interruptable", way.

6.4.3.6.7.1 Update Local Time (from *SyncTimeNSec* and *OVS*) Fields

[FO_PRS_CanTSyn_10083] Calculation of $T_{TSGlobalTime}$

Upstream requirements: RS_TS_20032, RS_TS_20035
Previous identifiers: SWS_CanTSyn_00087, SWS_CanTSyn_00206

⌈The Time Slave shall calculate the value of its Synchronized Time $T_{TSGlobalTime}$ as

$$T_{TSGlobalTime} = (SyncTimeSec + OVS + SyncTimeNSec) + \Delta t_{LinkDelay} + (T_{TSLocalTime_ReSync} - T_{SyncIngress})$$⌋

6.4.3.6.7.2 Update User Data (from *UserData_2*) Field

[FO_PRS_CanTSyn_10084] User Data if not CRC Protected

Upstream requirements: RS_TS_20032, RS_TS_20035, RS_TS_20037

⌈

If the Type of the received FUP message is equal to 0x18 or 0x78 then the Time Slave shall update the user data as follows

- set APP_TSUserData[2] to the value of the UserByte_2 field of the received FUP message
- and the value APP_TSUserDataLength to 3.

⌋

[FO_PRS_CanTSyn_10085] User Data if CRC Protected

Upstream requirements: RS_TS_20032, RS_TS_20035, RS_TS_20037

⌈

If the Type of the received FUP message is equal to 0x28 or 0x88 then the Time Slave shall update the user data as follows

- set APP_TSUserData[2] to 0
- and the value APP_TSUserDataLength to 1.

⌋

6.4.3.6.7.3 Update Time Gateway Synchronization Status (from *SGW*) Field

[FO_PRS_CanTSyn_00133] Reading the SGW field of a FUP message

Upstream requirements: RS_TS_20032, RS_TS_20033

⌈The Time Slave shall set APP_TSGatewayStatus to the value of the SGW field of the received FUP message.⌋

6.4.3.6.8 Transition from TS_ReceiveFupMsg

[FO_PRS_CanTSyn_10086] Debouncing after successful FUP message reception

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00086

⌈If in state TS_ReceiveFupMsg for a given Time Domain when the FUP message has been processed successfully (i.e., all checks for the FUP message fields have been successful and the FUP message got not discarded), then the Time Slave shall

- start Rx Debouncing Interval ($\Delta t_{RxDebounceInterval}$)
- and (re-)start Synchronization Lost Timeout interval $\Delta t_{SlaveSyncLossTimeout}$
- and reset the Synchronization Timeout flag to FALSE
- and transition to state TS_DebounceBeforeSyncMsg

⌋

[FO_PRS_CanTSyn_10087] Debouncing after FUP message reception failed

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00086

⌈If in state TS_ReceiveFupMsg for a given Time Domain when the FUP message has been discarded (i.e., one of the checks for the FUP message fields has failed), then the Time Slave shall

- start Rx Debouncing Interval ($\Delta t_{RxDebounceInterval}$)
- and transition to state TS_DebounceBeforeSyncMsg

⌋

6.4.3.7 State TS_DebounceBeforeSyncMsg

[FO_PRS_CanTSyn_10088] Discard Time Synchronization Message while debouncing after FUP message

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00183

⌈If in state `TS_DebounceBeforeSyncMsg` for a given Time Domain when any Time Synchronization message is received for the Time Domain, then the Time Slave shall discard that message.⌋

Rationale: This is to improve robustness against message sequence errors, specifically injection of fake SYNC messages by an attacker. Note that this will not allow to filter out all possible fake SYNC scenarios

[FO_PRS_CanTSyn_10089] Wait for next SYNC message after debouncing after FUP message

Upstream requirements: RS_TS_20035
Previous identifiers: SWS_CanTSyn_00183

⌈If in state `TS_DebounceBeforeSyncMsg` for a given Time Domain when Rx Debouncing Interval ($\Delta t_{\texttt{RxDebounceInterval}}$) has elapsed, then the Time Slave shall transition to state `TS_Idle`⌋

6.5 Error messages

<Add description of dedicated error messages of the protocol here>

<Describe the error on-wire format only. Error handling behavior shall be described in the context of the SWC (protocol user)>

7 Configuration Parameters

7.1 Configuration Parameters of a Time Domain

Parameter	Description
Cfg_TDDomainId	Parameter that defines the Id of a Time Domain.
Cfg_TDCanId	CAN Id of the CAN frame for the SYNC and FUP message of the Time Domain Cfg_TDDomainId.
Cfg_TDSyncMsgDataIDList	An array of 16 Data Ids used for CRC protection of SYNC messages.
Cfg_TDFupMsgDataIDList	An array of 16 Data Ids used for CRC protection of FUP messages.

Table 7.1: Parameter Definitions of a Time Domain

7.2 Configuration Parameters of the Time Master of a Time Domain

Parameter	Description
Cfg_TMCrcProtected	Parameter that determines, if the Time Master generates a CRC. Possible settings: • CRC_NOT_SUPPORTED • CRC_SUPPORTED.
Cfg_TMIcvSecured	Parameter that determines, if the Time Master generates an ICV. Possible settings: • ICV_NOT_SUPPORTED • ICV_SUPPORTED.
Cfg_TMSyncTxInterval	Parameter to configure the time interval for cyclic transmissions of SYNC messages by the Time Master. A value of 0 disables cyclic transmission requests.
Cfg_TMSyncTxTimeout	Timeout interval for transmission of SYNC message after requesting the transmissionon the CAN bus.
Cfg_TMDebounceTimeout	Inhibit time between transmission of successive Time Synchronization messages.
Cfg_TMIcvLength	Length of ICV to be transmitted within Follow_Up Message on the bus (in bytes). Range: 0 .. 54 bytes.
Cfg_TMFreshnessValueSupport	Parameter that determines, if the Time Master derives a Freshness Value. Possible settings: TRUE/FALSE.
Cfg_TMFreshnessValueLength	This parameter defines the complete length in bits of the Freshness Value. As long as the key does not change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter. Range: 0 .. 64 bits.

Parameter	Description
Cfg_TMIcvGenerationAlgorithm	This represents the configuration of cryptographic algorithm used for ICV generation. These algorithms are supported: • ICV_MAC • ICV_SIGNATURE.

Table 7.2: Parameter Definitions of a Time Master

7.3 Configuration Parameters of a Time Slave of a Time Domain

Parameter	Description
Cfg_TSIcvVerified	Parameter that determines, if the Time Slave verifies the ICV. Possible settings: • ICV_NOT_VERIFIED • ICV_IGNORED • ICV_OPTIONAL • ICV_VERIFIED.
Cfg_TSCrcVerified	Parameter that determines, if the Time Slave verifies the CRC. Possible settings: • CRC_NOT_VERIFIED • CRC_IGNORED • CRC_OPTIONAL • CRC_VERIFIED.
Cfg_TSSequenceCounterJumpWidth	This parameter defines the maximum allowed jump width of the sequence counter. Range: 0 .. 255.
Cfg_TSSequenceCounterHysteresis	This parameter defines the number of consecutive SYNC messages with a valid sequence counter jump while in SYNC message timeout before the sequence counter check is considered to be successful.
Cfg_TSSyncLossTimeout	Timeout interval between receiving two valid FUP message. Range: 0 .. INF. Unit: seconds.
Cfg_TSFupRxTimeout	Timeout interval for a FUP message reception after a SYNC message. Range: 0 .. INF. Unit: seconds.
Cfg_TSDebounceInterval	Debounce time for the reception of two consecutive Time Synchronization messages. Range: 0 .. 4. Unit: seconds.
Cfg_TSLinkDelay	Physical Layer specific propagation delay of a SYNC message to be compensated upon SYNC reception. Range: 0..1. Unit: seconds.

Parameter	Description
Cfg_TSFreshnessValueLength	This parameter defines the complete length in bits of the Freshness Value. As long as the key does not change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter. Range: 0 .. 64 bits.
Cfg_TSIcvLength	Length of ICV to be received within Follow_Up Message on the bus. Range: 0 .. 54 bytes.
Cfg_TSIcvVerificationAlgorithm	This represents the configuration of the cryptographic algorithm used for ICV verification. These algorithms are supported: • ICV_MAC • ICV_SIGNATURE.

Table 7.3: Parameter Definitions of a Time Slave

8 Protocol usage and guidelines

There are no guidelines on the usage of the protocol.

A 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.

A.1 Traceable item history of this document according to AUTOSAR Release R26-11

A.1.1 Added Specification Items in R26-11

none

A.1.2 Changed Specification Items in R26-11

none

A.1.3 Deleted Specification Items in R26-11

none

A.2 Traceable item history of this document according to AUTOSAR Release R25-11

A.2.1 Added Specification Items in R25-11

[FO_PRS_CanTSyn_00007] [FO_PRS_CanTSyn_00008] [FO_PRS_CanTSyn_00015] [FO_PRS_CanTSyn_00016] [FO_PRS_CanTSyn_00017] [FO_PRS_CanTSyn_00018] [FO_PRS_CanTSyn_00063] [FO_PRS_CanTSyn_00076] [FO_PRS_CanTSyn_00087] [FO_PRS_CanTSyn_00133] [FO_PRS_CanTSyn_00144] [FO_PRS_CanTSyn_00155] [FO_PRS_CanTSyn_00156] [FO_PRS_CanTSyn_00182] [FO_PRS_CanTSyn_00194] [FO_PRS_CanTSyn_00195] [FO_PRS_CanTSyn_00224] [FO_PRS_CanTSyn_00225] [FO_PRS_CanTSyn_00226] [FO_PRS_CanTSyn_00227] [FO_PRS_CanTSyn_00232] [FO_PRS_CanTSyn_10001] [FO_PRS_CanTSyn_10002] [FO_PRS_CanTSyn_10003] [FO_PRS_CanTSyn_10004] [FO_PRS_CanTSyn_10005] [FO_PRS_CanTSyn_10006] [FO_PRS_CanTSyn_10007] [FO_PRS_CanTSyn_10008] [FO_PRS_CanTSyn_10009] [FO_PRS_CanTSyn_10010] [FO_PRS_CanTSyn_10011] [FO_PRS_CanTSyn_10012] [FO_PRS_CanTSyn_10013] [FO_PRS_CanTSyn_10014] [FO_PRS_CanTSyn_10015] [FO_PRS_CanTSyn_10016] [FO_PRS_CanTSyn_10017] [FO_PRS_CanTSyn_10018]

[FO_PRS_CanTSyn_10019] [FO_PRS_CanTSyn_10020] [FO_PRS_CanTSyn_10021] [FO_PRS_CanTSyn_10022] [FO_PRS_CanTSyn_10023] [FO_PRS_CanTSyn_10024] [FO_PRS_CanTSyn_10025] [FO_PRS_CanTSyn_10026] [FO_PRS_CanTSyn_10027] [FO_PRS_CanTSyn_10028] [FO_PRS_CanTSyn_10029] [FO_PRS_CanTSyn_10030] [FO_PRS_CanTSyn_10031] [FO_PRS_CanTSyn_10032] [FO_PRS_CanTSyn_10033] [FO_PRS_CanTSyn_10034] [FO_PRS_CanTSyn_10035] [FO_PRS_CanTSyn_10036] [FO_PRS_CanTSyn_10037] [FO_PRS_CanTSyn_10038] [FO_PRS_CanTSyn_10039] [FO_PRS_CanTSyn_10040] [FO_PRS_CanTSyn_10041] [FO_PRS_CanTSyn_10042] [FO_PRS_CanTSyn_10043] [FO_PRS_CanTSyn_10044] [FO_PRS_CanTSyn_10045] [FO_PRS_CanTSyn_10046] [FO_PRS_CanTSyn_10047] [FO_PRS_CanTSyn_10048] [FO_PRS_CanTSyn_10049] [FO_PRS_CanTSyn_10050] [FO_PRS_CanTSyn_10051] [FO_PRS_CanTSyn_10052] [FO_PRS_CanTSyn_10053] [FO_PRS_CanTSyn_10054] [FO_PRS_CanTSyn_10055] [FO_PRS_CanTSyn_10056] [FO_PRS_CanTSyn_10057] [FO_PRS_CanTSyn_10058] [FO_PRS_CanTSyn_10059] [FO_PRS_CanTSyn_10060] [FO_PRS_CanTSyn_10061] [FO_PRS_CanTSyn_10062] [FO_PRS_CanTSyn_10063] [FO_PRS_CanTSyn_10064] [FO_PRS_CanTSyn_10065] [FO_PRS_CanTSyn_10066] [FO_PRS_CanTSyn_10067] [FO_PRS_CanTSyn_10068] [FO_PRS_CanTSyn_10069] [FO_PRS_CanTSyn_10070] [FO_PRS_CanTSyn_10071] [FO_PRS_CanTSyn_10072] [FO_PRS_CanTSyn_10073] [FO_PRS_CanTSyn_10074] [FO_PRS_CanTSyn_10075] [FO_PRS_CanTSyn_10076] [FO_PRS_CanTSyn_10077] [FO_PRS_CanTSyn_10078] [FO_PRS_CanTSyn_10079] [FO_PRS_CanTSyn_10080] [FO_PRS_CanTSyn_10081] [FO_PRS_CanTSyn_10082] [FO_PRS_CanTSyn_10083] [FO_PRS_CanTSyn_10084] [FO_PRS_CanTSyn_10085] [FO_PRS_CanTSyn_10086] [FO_PRS_CanTSyn_10087] [FO_PRS_CanTSyn_10088] [FO_PRS_CanTSyn_10089]

A.2.2 Changed Specification Items in R25-11

none

A.2.3 Deleted Specification Items in R25-11

none