Document Title	Specification of Time Service
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	624

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Major rework of whole specification, added service interface • [SRS_BSW_00334] removed from [SWS_Tm_NA_00059]
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Changed [SWS_Tm_00059] to [SWS_Tm_NA_00059] • Editorial changes
2021-11-25	R21-11	AUTOSAR Release Management	• Artefact incluseion based on ArtefactAnalysis corrected
2020-11-30	R20-11	AUTOSAR Release Management	• Included Development Errors and Runtime Errors as artifacts
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Header File Cleanup
2017-12-08	4.3.1	AUTOSAR Release Management	• Changed TM_E_HARDWARE_TIMER to Runtime Error • Renamed “default error” to “development error”

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Removed the definition of “configuration variants” from 10.2.1 Variants • Added line “Supported Config Variants” to the table of hte module definition in 10.2.2 Tm • Removed [SWS_Tm_00058] • Removed [SRS_BSW_00326], [SRS_BSW_00338], [SRS_BSW_00376], [SRS_BSW_00435], [SRS_BSW_00436]
2015-17-30	4.2.2	AUTOSAR Release Management	• Editorial changes
2014-10-30	4.2.1	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Initial release

Disclaimer

Contents

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module "Time Service".

The Time Service module is part of the Services Layer. The module provides services for time-based functionality. Use cases are:

- Time measurement
- time-based state machine
- Timeout supervision
- Busy waiting

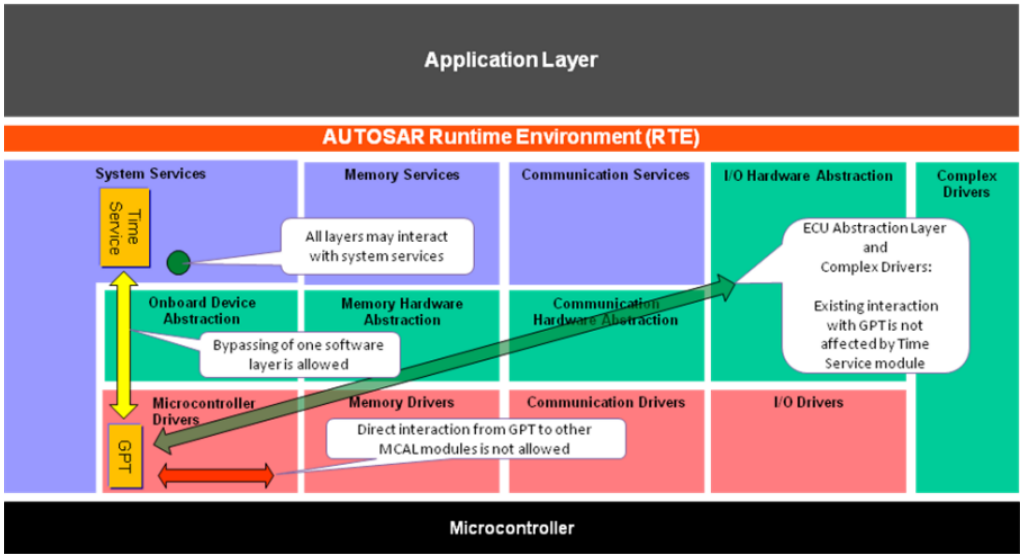

Figure 1.1: Architectural overview

The Time Service module does not use and distribute all features of the GPT driver.

Several types of - so called "Time Service Predef Timers" - are available, if supported by hardware and enabled by configuration.

Each Predef Timer has a predefined tick duration (physical time unit) and a predefined number of bits (physical range, width). By this, compatibility of time-based functionality is ensured for all platforms which support the required Predef Timers.

The Time Service Predef Timers are based on so-called "GPT Predef Timers", which are free running hardware timers, provided by the GPT driver [1].

The following time-based services are provided:

- `Tm_ResetTimer`
- `Tm_GetTimeSpan`
- `Tm_ShiftTimer`

- Tm_SyncTimer
- Tm_BusyWait1us

All services are called by user (polling mode). Notifications are not supported.

The time services can be used in:

- Initialization phase (after a call to Tm_Init)
- Tasks
- Cat2 interrupt service routines
- OS hooks

For implementation of the Time Service module no interrupts are needed.

1.1 Use cases

1.1.1 Time measurement

By using the Time Service module, execution time and cycle time of code can be measured, even run time and cycle time of:

- Tasks
- Cat2 interrupt service routines
- Functions
- Pieces of software

Time stamps can be generated.

Services of the Time Service module may be used to measure CPU load and task load, because the services may be called in the PreTaskHook (and PostTaskHook) of the Operating System.

1.1.2 time-based state machine

"Time base state machine" means: State transitions depending on timing. By using the Time Service module, time-based state machines can be implemented, which are nearly independently from the cycle time of the calling task. The user software has to ensure that the cycle time of the task is short enough relating to the desired timing behavior, due to polling of time information.

1.1.3 Timeout supervision and busy waiting

By using the Time Service module, errors and ambiguous behavior may be prevented in software modules by applying Predef Timers instead of "loops" or "nop instructions" to implement timeout supervision or busy waiting.

Using "loops" or "nop instructions" is a poor and critical design, because time intervals implemented in such a way are dependent on:

- CPU speed
- Pipeline effects
- Cache effects
- Access time to memory (bus width, wait states, ...)
- Interruption by Interrupt Service Routines
- Compiler version, compiler options, compiler optimizations

2 Acronyms, abbreviations and terms

Only a few acronyms and abbreviations are listed here which are helpful to understand this document or which have a local scope. Further information can be found in the official [2, AUTOSAR glossary].

Acronym / Abbreviation:	Description:
nop	No Operation

Table 2.1: Acronyms and abbreviations

The terms defined in the table below have a local scope within this document.

Term:	Description:
GPT Predef Timer	A GPT Predef Timer is a free running up counter provided by the GPT driver [1]. Which GPT Predef Timer(s) are available depends on hardware (clock, hardware timers, prescaler, width of timer register, ...) and configuration. A GPT Predef Timer has predefined physical time unit and range.
Time Service Predef Timer	A Time Service Predef Timer is a free running up counter with predefined physical time unit and range. The hardware timer functionality is based on the corresponding GPT Predef Timer. The user uses timer instances to interact with the PreDef Timers.
Timer instance	A timer instance is a configured object (of a Time Service Predef Timer) and is used as handle in APIs. The user can configure any number of instances, they are completely independently of each other.
Reference time	The reference time is a time value stored for each timer instance. It represents the "start" time of the timer instance.

Table 2.2: Terms

3 Related documentation

3.1 Input documents & related standards and norms

[1] Specification of GPT Driver
AUTOSAR_CP_SWS_GPTDriver

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS BSW General], which is also valid for Tm.

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

4 Constraints and assumptions

4.1 Assumptions

The Time Service module is using hardware timers which may be impacted when the ECU is in SLEEP mode. This means that it depends on the hardware if a timer still counts in such a situation or if it stops while the ECU is in SLEEP. As the Time Service module itself cannot detect this it is assumed that the users of the module are aware of such situations. As a consequence it is assumed that users don't measure time across SLEEP periods and that they reset the Time Service timers after a SLEEP.

4.2 Limitations

Functionality is based on HW timers which are not perhaps available

The functionality of the Time Service module is based on hardware timers (GPT Predef Timers) provided by the GPT Driver [1]. Which GPT Predef Timer(s) can be enabled depends on clock and available timer hardware (prescaler, width of timer register). It is recommended to enable all GPT Predef Timers to ensure compatibility of time-based functionality for all platforms.

4.3 Applicability to car domains

No restrictions.

5 Dependencies to other modules

This section describes the relations to other modules.

The Time Service module has dependencies to the following other AUTOSAR modules:

GPT:

The functionality of the Time Service module is based on so called "GPT Predef Timers". A GPT Predef Timer is a free running up counter provided by the [1, GPT driver].

6 Requirements Tracing

The following tables reference the requirements specified in AUTOSAR SRS documents and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_Tm_91000] [SWS_Tm_91001]
[SRS_BSW_00312]	Shared code shall be reentrant	[SWS_Tm_00020]
[SRS_BSW_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Tm_00008] [SWS_Tm_00012] [SWS_Tm_00016] [SWS_Tm_00018] [SWS_Tm_00021] [SWS_Tm_00037] [SWS_Tm_00068] [SWS_Tm_00077] [SWS_Tm_00082]
[SRS_BSW_00337]	Classification of development errors	[SWS_Tm_00030]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_Tm_91001]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Tm_00031]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_Tm_91001]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Tm_00008] [SWS_Tm_00012] [SWS_Tm_00066] [SWS_Tm_91002] [SWS_Tm_91003]
[SRS_BSW_00386]	The BSW shall specify the configuration and conditions for detecting an error	[SWS_Tm_00068] [SWS_Tm_00082]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_Tm_91001]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Tm_91001]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Tm_00036]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_Tm_91001]
[SRS_BSW_00452]	Classification of runtime errors	[SWS_Tm_00064]
[SRS_Tm_00001]	Different types of Predef Timers shall be supported by the Time Service module	[SWS_Tm_91002] [SWS_Tm_91003] [SWS_Tm_91004] [SWS_Tm_91005] [SWS_Tm_91006]
[SRS_Tm_00002]	The GPT Predef Timers shall be used as time base for the Predef Timers of the Time Service module	[SWS_Tm_00001] [SWS_Tm_00057]
[SRS_Tm_00004]	The Time Service module shall provide a synchronous service to reset a timer instance	[SWS_Tm_00006] [SWS_Tm_00063] [SWS_Tm_91002]

△

Requirement	Description	Satisfied by
[SRS_Tm_00005]	The Time Service module shall provide a synchronous service to get the time span	[SWS_Tm_00009] [SWS_Tm_00010] [SWS_Tm_00063] [SWS_Tm_00065] [SWS_Tm_00069] [SWS_Tm_91003]
[SRS_Tm_00006]	The Time Service module shall provide a synchronous service to shift the reference time of a timer instance	[SWS_Tm_00013] [SWS_Tm_00014] [SWS_Tm_00063] [SWS_Tm_91004] [SWS_Tm_CONSTR_00002]
[SRS_Tm_00007]	The Time Service module shall provide a synchronous service to synchronize two timer instances	[SWS_Tm_00019] [SWS_Tm_00063] [SWS_Tm_91005]
[SRS_Tm_00008]	The Time Service module shall provide a synchronous service with tick duration 1μs to perform busy waiting by polling	[SWS_Tm_00022] [SWS_Tm_00023] [SWS_Tm_00024] [SWS_Tm_00070] [SWS_Tm_91006]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 TM Predef Timers

7.1.1 Background

This functionality of the Time Service module is based on so called "GPT Predef Timers", see [1, SWS GPTDriver].

7.1.2 Time Service Predef Timers

A Time Service Predef Timer is based on the corresponding GPT Predef Timer. The Time Service module assumes that the GPT is configured in a way suitable for the Time Service.

The following resolutions are available:

Type of Time Service Predef Timer	Tick duration	Maximum tick value	Number of bits	Maximum time span (circa values)
TM_PREDEF_TIMER_1US_16BIT	1 μs	65535	16 bit	65 ms
TM_PREDEF_TIMER_1US_24BIT		16777215	24 bit	16 s
TM_PREDEF_TIMER_1US_32BIT		4294967295	32 bit	71 minutes
TM_PREDEF_TIMER_100US_32BIT	100 μs	4294967295	32 bit	4.9 days

Table 7.1: Characteristics of Time Service Predef Timers

Users of Time Service require a timer instance to access the functionality offered in Time Service module. A timer instance has to be configured (via `TmPreDefTimerInstance`) before it can be used.

The Time Service module offers a set of API services which are available to its users. The following services are offered for timer instances:

- `Tm_ResetTimer`
- `Tm_GetTimeSpan`
- `Tm_ShiftTimer`
- `Tm_SyncTimer`

Additionally the service `Tm_BusyWait1us` is available for short waiting times.

[SWS_Tm_00001]

Upstream requirements: SRS_Tm_00002

⌈The Time Service module shall use the GPT driver service `Gpt_GetPredefTimer-Value` with the related resolution and width of the timers to get the current time value for the desired timer instance.⌋

7.1.3 Timing aspects to consider

This chapter contains several aspects of the provided Time Service functionality which has to be considered on user software level. Chapter 7.1.4.6 contains additional hints for users of `Tm_BusyWait1us`

7.1.3.1 Maximal measurable time span

The measurable time span is restricted to the maximum value of the corresponding GPT Predef Timer. A wrap-around of a timer is handled by the `Tm_GetTimeSpan` function, see [SWS_Tm_00010].

The diagram "Free running up counter" below shows the general behavior of a free running up counter provided by the GPT driver [1]. The services `Tm_ResetTimer` and `Tm_GetTimeSpan` are used to measure three time spans, as example.

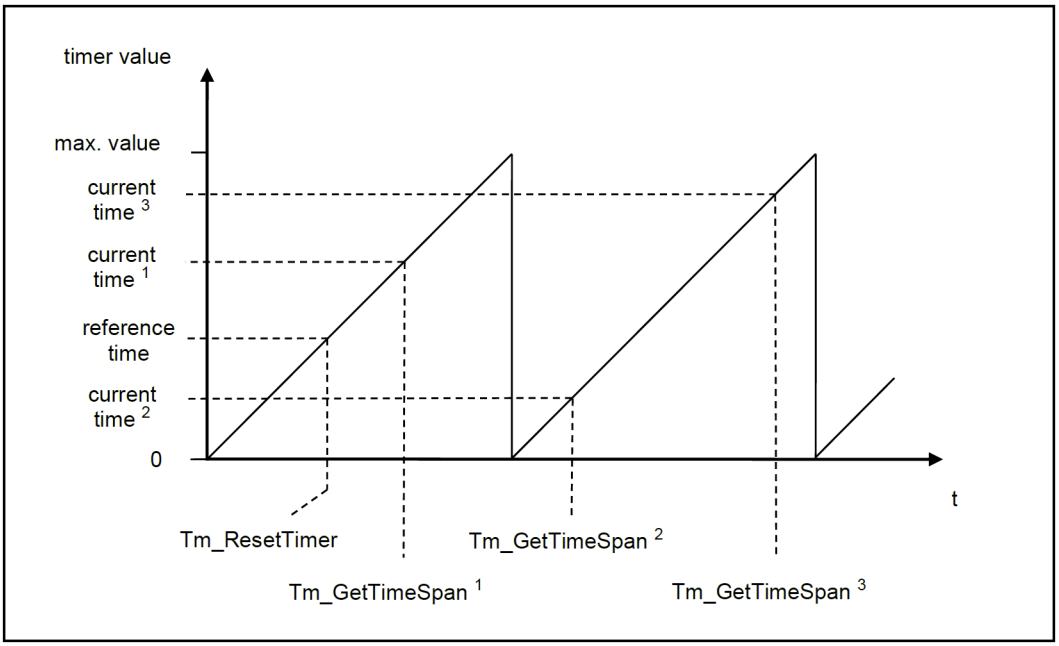

Figure 7.1: Free running up counter

By calling `Tm_ResetTimer` the current time of the related GPT Predef Timer is stored as a reference time in the timer instance. For details see chapter 7.1.4.1.

By calling Tm_GetTimeSpan the time difference between the current time and the reference time is calculated and delivered. For details see chapter 7.1.4.2.

For:

- Tm_GetTimeSpan[1]
- Tm_GetTimeSpan[2]

the time span will be calculated correctly.

For:

- Tm_GetTimeSpan[3]

it is not possible to calculate the correct time span, because the maximum time span is exceeded. It is not possible for the Time Service module to detect such an exceeding. This is not a fault of this specification, it's a logical consequence caused by the technical principle. See also "Unintentional behavior of Tm_BusyWait1us services" in chapter 7.1.4.6.

To ensure correct behavior under every possible circumstance, the user of the Tm_GetTimeSpan service has to check:

- which Predef Timer is required/sufficient
- the task scheduling
- whether an interrupt or resource lock is necessary on user software level
- whether the user software is tolerant of such problems

7.1.3.2 Time quantization error

The theory of quantization error has to be considered at using/interpretation of the values delivered by the Tm_GetTimeSpan function.

The value delivered by a Tm_GetTimeSpan function has an accuracy of +/- 1 tick.

For example:

Value delivered by Tm_GetTimeSpan function		Real time minimum			Real time maximum			Comment
Value	Tick duration							
1	μs	nearly	0	μs	nearly	2	μs	See figure 7.2
3400	μs	nearly	3399	μs	nearly	3401	μs	
56	100μs	nearly	5500	μs	nearly	5700	μs	

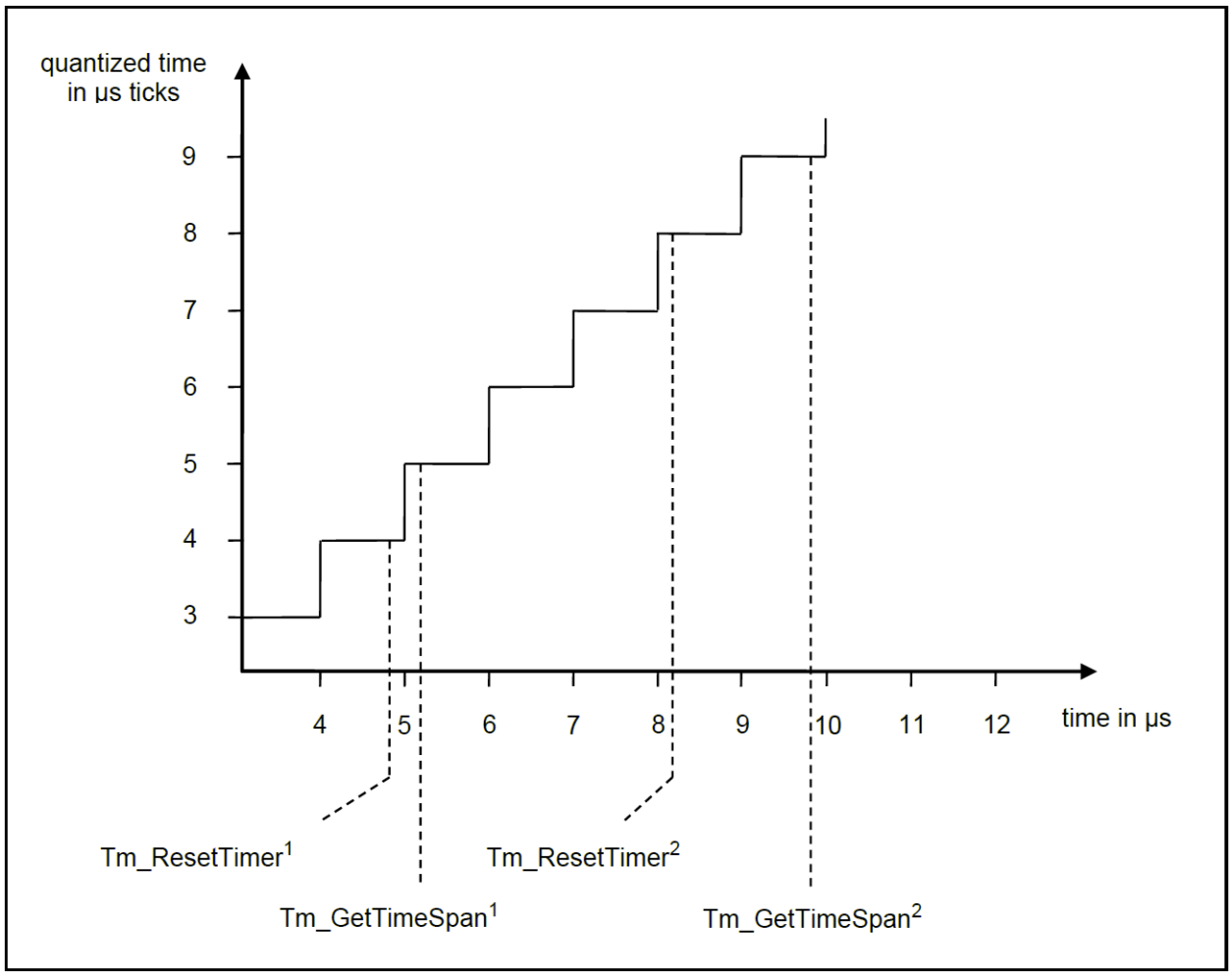

Figure 7.2: Time quantization example diagram

In the example diagram above both calls of `Tm_GetTimeSpan` ([1] and [2]) deliver the value 1, this means 1μs.

Depending on points in time the calls of `Tm_ResetTimer` and `Tm_GetTimeSpan` occur, the real time span can be in a range nearly 0μs to nearly 2μs.

If a `Tm_GetTimeSpan` function is used to check a minimum time, e.g. for:

- Timeout supervision
- Busy waiting

n+1 ticks must be observed by user software to ensure that an interval of at least n ticks has passed, see also [SWS_Tm_00024].

For busy waiting please use the `Tm_BusyWait1us` service, see chapter 7.1.4.5.

7.1.3.3 Execution times of services / measurement of short time spans

If short time spans shall be measured on user software level, the execution times of the Tm services and the underlying GPT driver services shall be short enough related to the time spans to be measured.

The execution times are dependent on:

- Implementation
- CPU speed
- Realization of related GPT Predef Timer, see chapter GPT Predef Timer in [1, SWS GPT Driver]

The user has to check whether the execution times are sufficient for his use case.

7.1.4 API Services

The `Tm_GetTimeSpan`, `Tm_ShiftTimer` and `Tm_SyncTimer` services require timer instance(s) as input. It is assumed that these timer instance(s) do have a valid reference time. This means there was a previous call to `Tm_ResetTimer` which used the timer instance.

[SWS_Tm_00068] Development error when refernce time is missing

Upstream requirements: SRS_BSW_00386, SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If a timer instance provided by the user in a call to `Tm_GetTimeSpan`, `Tm_ShiftTimer` or `Tm_SyncTimer` does not contain a reference time the called function shall raise the development error `TM_E_STATE`.⌋

The `Tm_GetTimeSpan` and the `Tm_ShiftTimer` services provide or require a timer value. For both services the type of the timer value is fixed to a 32 bit type (`uint32`). If predef timers are used with a smaller width (16bit or 24bit), the "unused" part of the timer value shall always be 0.

Example: If a timer instance is using a 16bit pre def timer then values of the timer are always in the range of `0x00000000` and `0x0000FFFF`.

[SWS_Tm_CONSTR_00002] Assumption on provided timer value

Upstream requirements: SRS_Tm_00006

⌈When `Tm_ShiftTimer` is called with a timer instance using a 16bit (or 24bit) the Time Service module assumes that the value of the parameter TimeValue is in the range of `0x0000` to `0xFFFF` (or `0x000000` to `0xFFFFFF`).⌋

[SWS_Tm_00069] Range of timer values

Upstream requirements: SRS_Tm_00005

⌈For timer instance using a 16bit (or 24bit) the `Tm_GetTimeSpan` will always return a timer value between `0x00000000` and `0x0000FFFF` (or `0x00000000` and `0x00FFFFFF`).⌋

7.1.4.1 Service ResetTimer

The service `Tm_ResetTimer` resets a timer instance from user point of view.

[SWS_Tm_00006]

Upstream requirements: SRS_Tm_00004

⌈The `Tm_ResetTimer` function shall reset the provided timer instance. This means, the reference time of the timer instance shall be set to the current time of the related GPT Predef Timer.⌋

7.1.4.2 Service GetTimeSpan

The service `Tm_GetTimeSpan` returns the current timer value of the timer instance.

[SWS_Tm_00009]

Upstream requirements: SRS_Tm_00005

⌈The `Tm_GetTimeSpan` function shall calculate and deliver the time difference between the current time and the reference time of the timer instance.⌋

Note: The restriction of maximal measurable time span has to be considered on user software level, see chapter 7.1.3.1.

Note: Because the `Tm_GetTimeSpan` function deliver time differences as integer values, the theory of quantization error has to be considered on user software level at using/interpretation of the values, see chapter 7.1.3.2.

[SWS_Tm_00010]

Upstream requirements: SRS_Tm_00005

⌈The `Tm_GetTimeSpan` function shall perform proper wrap-around handling at subtraction (current time - reference time), if value of current time is less than value of reference time.⌋

7.1.4.3 Service ShiftTimer

[SWS_Tm_00013]

Upstream requirements: SRS_Tm_00006

⌈The `Tm_ShiftTimer` function shall shift the reference time of the timer instance. This means, the provided timer value shall be added to the reference time of the timer instance.⌋

[SWS_Tm_00014]

Upstream requirements: SRS_Tm_00006

⌈The `Tm_ShiftTimer` function shall perform proper wrap-around handling at adding (reference time + TimeValue), if the sum is greater than the maximum value of the timer.⌋

7.1.4.4 Service SyncTimer

[SWS_Tm_00019]

Upstream requirements: SRS_Tm_00007

⌈The `Tm_SyncTimer` function shall synchronize two timer instances. This means, the reference time of the destination timer instance shall be set to the reference time of the source timer instance.⌋

7.1.4.5 Service BusyWait

The service `Tm_BusyWait1us` performs busy waiting (active waiting) by polling with a guaranteed minimum waiting time. The `Tm_BusyWait1us` service should be used instead of own implementations on user software level to avoid risks of bad implementations.

Risks may be:

- minimum waiting time is not guaranteed
- "loops" or "nop instructions" are used instead of hardware timers, see chapter 1.1

Note: The specification of the `Tm_BusyWait1us` function considers the theory of quantization error, see chapter 7.1.3.2.

Note: Because the Tm_BusyWait1us service is based on polling, the user of the Tm_BusyWait1us service is responsible for avoiding unintentional behavior, see chapter 7.1.4.6.

The service is available with tick duration of 1μs. The waiting time is restricted to 8 bits (255μs) to prevent long time blocking of code execution.

[SWS_Tm_00022]

Upstream requirements: SRS_Tm_00008

⌈The Tm_BusyWait1us function shall perform busy waiting for the minimum time passed by the parameter WaitingTimeMin.⌋

[SWS_Tm_00023]

Upstream requirements: SRS_Tm_00008

⌈The Tm_BusyWait1us function shall not disable the interrupts. This means the real waiting time may be greater than the desired waiting time.⌋

[SWS_Tm_00024]

Upstream requirements: SRS_Tm_00008

⌈The Tm_BusyWait1us function shall guarantee the minimum waiting. This means, n+1 ticks must be observed to ensure that an interval of at least n ticks have passed.⌋

[SWS_Tm_00070] Resolution of busy waiting

Upstream requirements: SRS_Tm_00008

⌈The Tm_BusyWait1us function shall always use a PreDef Timer with 1μs resolution and 32bit width.⌋

7.1.4.6 Unintentional behavior of BusyWait services

This chapter has to be considered on user software level.

Because the Tm_BusyWait1us service is based on polling, the user of Tm_BusyWait1us service is responsible for avoiding unintentional behavior.

Unintentional behavior can occur when an ongoing call of Tm_BusyWait1us is preempted or interrupted. Here is a possible scenario:

1. A Task A calls Tm_BusyWait1us with waiting time of 10μs. The function reads the current value of the timer. The current value is 0.
2. After 2μs the scheduler preempts Task A, and another Task is executed.

3. Task A stays preempted for a long time. In the meantime the timer wraps around and the value is again 0
4. Shortly afterwards Task A continues. The next value which is read in the function `Tm_BusyWait1us` is 3
5. For `Tm_BusyWait1us` it looks like that just 1μs has passed. So it still loops until the timer reaches 10.

The waiting time in the above example is in the end extremely long. By using the service `Tm_BusyWait1us` a problem as described above can only occur, if a task which calls the busy wait function is preempted for more than 71 minutes. See also [SWS_Tm_00070].

To ensure correct behavior under every possible circumstance, the user of the `Tm_BusyWait1us` service has to check:

- the task scheduling
- whether an interrupt or resource lock is necessary on user software level
- whether the user software is tolerant of such problems

7.1.5 Configuration of Predef Timers

The Time Service module requires that timer instances are configured before they can be used. See `TmPreDefTimerInstance` for details. For each timer instance the type of the underlying timer has to be provided (see `TmPreDefTimerType`). Additionally it has to be specified if the timer instance is used on application level (access via ports) or just from other modules via its C-API. This can be specified in `TmTimerUser`.

There is a restriction regarding the availability of port(s) and timer types: For ports only PreDef Timers of type `TM_PREDEF_TIMER_1US_32BIT` are allowed.

[SWS_Tm_CONSTR_00001] Service interface always use 32bit wide 1us resolution ⌈The Time Service module only support ports for configured timer instances of type `TM_PREDEF_TIMER_1US_32BIT`.⌋

[SWS_Tm_00071] Configuration check ⌈If a configuration contains a `TmPreDefTimerInstance` where the `TmTimerUser` equals `PORT` and the `TmPreDefTimerType` is not equal `TM_PREDEF_TIMER_1US_32BIT` then the generator tool of the Time Service module shall report an error.⌋

7.1.6 Sample code of use cases

This chapter contains example code of use cases in addition to the use cases described in chapter 1.1.

7.1.6.1 Time measurement

Sometimes execution time of code shall be measured.

Sample code:

```
#include "Os.h"
#include "Tm.h"

/* TmConf_TmPreDefTimerInstance_TimerIsr1 = Name of the configured timer
    instance with a 1us resolution */
/* TmConf_TmPreDefTimerInstance_TimerTask100us = Name of the configured
    timer instance with a 1us resolution */

uint32 RunTimeIsr1us; /* Gross runtime of Isr1 */
uint32 RunTimeTask100us; /* Gross runtime of Task100ms */

ISR(Isr1) {
  (void)Tm_ResetTimer(TmConf_TmPreDefTimerInstance_TimerIsr1);
  /* Code */
  (void)Tm_GetTimeSpan(TmConf_TmPreDefTimerInstance_TimerIsr1, &
   RunTimeIsr1us);
}

TASK(Task100ms) {
  (void)Tm_ResetTimer(TmConf_TmPreDefTimerInstance_TimerTask100us);
  /* Code */
  (void)Tm_GetTimeSpan(TmConf_TmPreDefTimerInstance_TimerTask100us, &
   RunTimeTask100us);
  (void)TerminateTask();
}
```

7.1.6.2 time-based state machine

By implementing a time-based state machine it is possible to realize time-based functionality nearly independently from the cycle time of the calling task.

Sample code:

```
#include "Os.h"
#include "Tm.h"

#define MY_INIT 0
#define MY_WAIT1 1
#define MY_WAIT2 2

```

```
/* TmConf_TmPreDefTimerInstance_Timer = Name of configured timer instance.
   */

uint8_least State = MY_INIT;

TASK(Task5ms) {
  uint32 WaitingTime1_us = 500000u; /* 500ms */
  uint32 WaitingTime2_us = 250000u; /* 250ms */

  switch (State) {
    case MY_INIT: {
      (void)Tm_ResetTimer(TmConf_TmPreDefTimerInstance_Timer);
      State = MY_WAIT1;
      break;
    }
    case MY_WAIT1: {
      uint32 Time_us;
      (void)Tm_GetTimeSpan(TmConf_TmPreDefTimerInstance_Timer, &Time_us);
      if (Time_us >= WaitingTime1_us) {
        /* Action ... */
        Tm_ShiftTimer(TmConf_TmPreDefTimerInstance_Timer, WaitingTime1_us);
        State = MY_WAIT2;
      }
      break;
    }
    case MY_WAIT2: {
      uint32 Time_us;
      (void)Tm_GetTimeSpan(TmConf_TmPreDefTimerInstance_Timer, &Time_us);
      if (Time_us >= WaitingTime2_us) {
        /* Action ... */
        Tm_ShiftTimer(TmConf_TmPreDefTimerInstance_Timer, WaitingTime2_us);
        State = MY_WAIT1;
      }
      break;
    }
  }
  (void)TerminateTask();
}
```

7.1.6.3 Timeout supervision

In case of hardware accessing MCAL driver, sometimes it is necessary that a hardware reaction is expected within certain but short time frame.

Sample code:

```
#include "Register.h"
#include "Tm.h"

/* TmConf_TmPreDefTimerInstance_Timer1 = Name of configured timer instance
   */

uint16 StatusRegisterBit0;
uint32 TimeElapsed_us;
```

```
8
9  void SampleFunction(void) {
10   (void)Tm_ResetTimer(TmConf_TmPreDefTimerInstance_Timer1);
11   do {
12     StatusRegisterBit0 = HW_STATUS_REG & 0x0001u;
13     (void)Tm_GetTimeSpan(TmConf_TmPreDefTimerInstance_Timer1, &
       TimeElapsed_us);
14   } while ( (StatusRegisterBit0 != 0x0001u) /* Wait until bit 0 is set*/
15             && (TimeElapsed_us <= 40) /* Timeout 40us */);
16 }
```

7.1.6.4 Busy waiting

In case of hardware accessing MCAL driver, sometimes it is necessary that a certain but short time frame shall elapse.

Sample code:

```
1  #include "Tm.h"
2
3  Std_ReturnType CanTrcv_SetOpMode(uint8 Transceiver, CanIf_TrcvModeType
       OpMode) {
4    /* Code */
5    switch(OpMode) {
6      case CANIF_TRCV_MODE_NORMAL: {
7        /* Code */
8        break;
9      }
10     case CANIF_TRCV_MODE_SLEEP: {
11       /* Code */
12       SetPinEnableHigh();
13       /* Busy waiting: 50us (for TJA1054: at least 50us) */
14       (void)Tm_BusyWait1us(50);
15       SetPinEnableLow();
16       /* Code */
17       break;
18     }
19     case CANIF_TRCV_MODE_STANDBY: {
20       /* Code */
21       break;
22     }
23   }
24   /* Code */
25 }
```

7.2 Version check

Please refer to chapter "Version Check" in [3, SWS BSW General].

7.3 Error classification

[SWS_Tm_00063]

Upstream requirements: SRS_Tm_00004, SRS_Tm_00005, SRS_Tm_00006, SRS_Tm_00007

⌈When an error occurs the corresponding Time Service function shall return without any action, unless it is specified for the specific function differently/more in detail.⌋

7.3.1 Development Errors

[SWS_Tm_00028] Definiton of development errors in module Tm ⌈

Type of error	*Related error code*	*Error value*
API parameter checking: invalid pointer	TM_E_PARAM_POINTER	0x01
API parameter checking: invalid value	TM_E_PARAM_VALUE	0x02
API parameter checking: incompatible timer instances for Tm_SyncTimer	TM_E_PARAM_SYNC	0x03
A provided timer instance was not initialized via Tm_ResetTimer	TM_E_STATE	0x04

⌋

[SWS_Tm_00030]

Upstream requirements: SRS_BSW_00337

⌈Additional errors that are detected because of specific implementation shall be added in the specific implementation specification. The classification and enumeration shall be compatible to the errors listed.⌋

7.3.2 Runtime Errors

[SWS_Tm_00067] Definiton of runtime errors in module Tm ⌈

Type of error	*Related error code*	*Error value*
Access to underlying hardware timer failed	TM_E_HARDWARE_TIMER	0x03

⌋

[SWS_Tm_00064]

Upstream requirements: SRS_BSW_00452

⌈If the underlying GPT driver service returns `E_NOT_OK`, the functions `Tm_ResetTimer`, `Tm_GetTimeSpan` and `Tm_BusyWait1us` shall raise the runtime error `TM_E_HARDWARE_TIMER`.⌋

7.3.3 Production Errors

No production errors are defined for the Time Service module.

7.3.4 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

[SWS_Tm_00031] Definition of imported datatypes of module Tm

Upstream requirements: SRS_BSW_00348

⌈

Module	*Header File*	*Imported Type*
Gpt	Gpt.h	Gpt_PredefTimerType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋

8.2 Type Definitions

8.2.1 Tm_ConfigType

[SWS_Tm_91000] Definition of datatype Tm_ConfigType

Upstream requirements: SRS_BSW_00101

⌈

Name	Tm_ConfigType	
Kind	Structure	
Elements	implementation specific	
	Type	–
	Comment	The contents of the initialization data structure are implementation specific
Description	This type contains the implementation-specific configuration structure.	
Available via	Tm.h	

⌋

8.3 Function definitions

8.3.1 Tm_Init

[SWS_Tm_91001] Definition of API function Tm_Init

Upstream requirements: SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_-00101, SRS_BSW_00358, SRS_BSW_00414

⌈

Service Name	Tm_Init	

Syntax	`void Tm_Init (`
	`  const Tm_ConfigType* config`
	`)`	

Service ID [hex]	0x02	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	config	Pointer to the Time Service module's configuration dat
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	This service initializes the Time Service module. Is shall be called before other functional APIs are used.	
Available via	Tm.h	

⌋

8.3.2 Tm_GetVersionInfo

[SWS_Tm_00036] Definition of API function Tm_GetVersionInfo

Upstream requirements: SRS_BSW_00407

⌈

Service Name	Tm_GetVersionInfo	

Syntax	`void Tm_GetVersionInfo (`
	`  Std_VersionInfoType* VersionInfoPtr`
	`)`	

Service ID [hex]	0x1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	VersionInfoPtr	Pointer to where to store the version information of this module.
Return value	None	
Description	Returns the version information of this module.	

▽

△

Available via	Tm.h

⌋

[SWS_Tm_00037]

Upstream requirements: SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the parameter VersionInfoPtr is a null pointer, the function `Tm_GetVersionInfo` shall raise the error `TM_E_PARAM_POINTER`.⌋

8.3.3 Tm_ResetTimer

[SWS_Tm_91002] Definition of API function Tm_ResetTimer

Upstream requirements: SRS_Tm_00001, SRS_Tm_00004, SRS_BSW_00369

⌈

Service Name	Tm_ResetTimer	
Syntax	`Std_ReturnType Tm_ResetTimer ( uint32 timerId )`	
Service ID [hex]	0x03	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same timerId	
Parameters (in)	timerId	A timer instance handle.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The underlying GPT driver service has returned `E_OK`. `E_NOT_OK`: The underlying GPT driver service has not returned `E_NOT_OK`.
Description	Reset a timer instance by setting the reference time to the current value of the related PreDef Timer.	
Available via	Tm.h	

⌋

[SWS_Tm_00008]

Upstream requirements: SRS_BSW_00369, SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the parameter timerId is not a valid timer instance, the `Tm_ResetTimer` functions shall raise the development error `TM_E_PARAM_VALUE`.⌋

8.3.4 Tm_GetTimeSpan

[SWS_Tm_91003] Definition of API function Tm_GetTimeSpan

Upstream requirements: SRS_Tm_00001, SRS_Tm_00005, SRS_BSW_00369

⌈

Service Name	Tm_GetTimeSpan	
Syntax	`Std_ReturnType Tm_GetTimeSpan ( uint32 timerId, uint32* timeSpan )`	
Service ID [hex]	0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	timerId	A timer instance handle.
Parameters (inout)	None	
Parameters (out)	timeSpan	Pointer to location where the time will be stored.
Return value	Std_ReturnType	`E_OK`: The underlying GPT driver service has returned `E_OK`. `E_NOT_OK`: The underlying GPT driver service has not returned `E_OK`.
Description	Delivers the (relative) value of the time since the last reset (or shifting) of the timer instance.	
Available via	Tm.h	

⌋

[SWS_Tm_00082] Unknown timer

Upstream requirements: SRS_BSW_00386, SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the parameter timerId is not a valid timer instance, the `Tm_GetTimeSpan` functions shall raise the development error `TM_E_PARAM_VALUE`.⌋

[SWS_Tm_00012]

Upstream requirements: SRS_BSW_00369, SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the pointer parameter (timeSpan) is a null pointer, the `Tm_GetTimeSpan` functions shall raise the development error `TM_E_PARAM_POINTER`.⌋

[SWS_Tm_00065]

Upstream requirements: SRS_Tm_00005

⌈When `Tm_GetTimeSpan` returns `E_NOT_OK` it shall set the value for the timeSpan to "0".⌋

8.3.5 Tm_ShiftTimer

[SWS_Tm_91004] Definition of API function Tm_ShiftTimer

Upstream requirements: SRS_Tm_00001, SRS_Tm_00006

⌈

Service Name	Tm_ShiftTimer	
Syntax	`void Tm_ShiftTimer ( uint32 timerId, uint32 timeValue )`	
Service ID [hex]	0x05	
Sync/Async	Synchronous	
Reentrancy	Reentrant but not for the same timer instance	
Parameters (in)	timerId	A timer instance handle.
	timeValue	Time value in in units (e.g. µs), the reference time has to be shifted.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Shifts the reference time of the timer instance.	
Available via	Tm.h	

⌋

[SWS_Tm_00018]

Upstream requirements: SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the timerId is not a valid timer instance, the `Tm_ShiftTimer` functions shall raise the development error `TM_E_PARAM_VALUE`.⌋

[SWS_Tm_00016]

Upstream requirements: SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the value of the parameter timeValue is greater than the maximum allowed of the base GPT predef timer (`0xFFFFFF` for 24bit timers and `0xFFFF` for 16bit timers), the `Tm_ShiftTimer` functions shall raise the development error `TM_E_PARAM_VALUE`.⌋

Note: A shift by "0" is considered a valid request although no real adjustments are performed.

8.3.6 Tm_SyncTimer

[SWS_Tm_91005] Definition of API function Tm_SyncTimer

Upstream requirements: SRS_Tm_00001, SRS_Tm_00007

⌈

Service Name	Tm_SyncTimer	
Syntax	`void Tm_SyncTimer ( uint32 timerIdSrc, uint32 timerIdDst )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant Conditionally reentrant (see [SWS_Tm_00020])	
Parameters (in)	timerIdSrc	Source timer instance defined by the user.
	timerIdDst	Destination timer instance defined by the user.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Synchronizes two timer instances.	
Available via	Tm.h	

⌋

[SWS_Tm_00021]

Upstream requirements: SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If a parameter is not a valid timer instance, the `Tm_SyncTimer` functions shall raise the development error `TM_E_PARAM_VALUE`.⌋

[SWS_Tm_00077] Resolution mismatch

Upstream requirements: SRS_BSW_00323

⌈If development error detection for the Time Service module is enabled: If the resolution or the width of the two timer instances are not equal , the `Tm_SyncTimer` functions shall raise the development error `TM_E_PARAM_SYNC`.⌋

[SWS_Tm_00020]

Upstream requirements: SRS_BSW_00312

⌈The SyncTime functions shall be reentrant, if the timer instances used in concurrent calls are different.⌋

8.3.7 Tm_BusyWait1us

[SWS_Tm_91006] Definition of API function Tm_BusyWait1us

Upstream requirements: SRS_Tm_00001, SRS_Tm_00008

⌈

Service Name	Tm_BusyWait1us	
Syntax	`Std_ReturnType Tm_BusyWait1us ( uint8 WaitingTimeMin )`	
Service ID [hex]	0x15	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	WaitingTimeMin	Minimum waiting time in microseconds.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: The underlying GPT driver service has returned `E_OK`. `E_NOT_OK`: The underlying GPT driver service has not returned `E_OK`.
Description	Performs busy waiting by polling with a guaranteed minimum waiting time.	
Available via	Tm.h	

⌋

Note: Because the BusyWait service is based on polling, the user of the BusyWait service is responsible for avoiding unintentional behavior, see chapter 7.1.4.5 Service BusyWait.

[SWS_Tm_00066]

Upstream requirements: SRS_BSW_00369

⌈When an error is detected, the BusyWait functions shall return `E_NOT_OK` and shall abort "waiting" immediately.⌋

The [SWS_Tm_00066] means that if an error is returned the minimal waiting time is (most likely) not reached.

8.4 Call-back Notifications

None.

8.5 Scheduled functions

None.

8.6 Expected Interfaces

In this chapter all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all interfaces, which are required to fulfill the core functionality of the module.

[SWS_Tm_00057] Definition of mandatory interfaces required by module Tm

Upstream requirements: SRS_Tm_00002

⌈

API Function	*Header File*	*Description*
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.
Gpt_GetPredefTimerValue	Gpt.h	Delivers the current value of the desired GPT Predef Timer.

⌋

8.6.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_Tm_00060] Definition of optional interfaces requested by module Tm ⌈

API Function	*Header File*	*Description*
Det_ReportError	Det.h	Service to report development errors.

⌋

8.6.3 Configurable Interfaces

In this chapter all interfaces are listed where the target function could be configured. The Time Service module does not have such functions.

None.

8.7 Service Interfaces

8.7.1 Provided Ports of Tm

[SWS_Tm_91007] Definition of Port TmPreDefTimer_{Name} provided by module Tm ⌈

Name	TmPreDefTimer_{Name}		
Kind	ProvidedPort	***Interface***	TmPreDefTimerService
Description	–		
Port Defined Argument Value(s)	***Type***	uint32	
	Value	{ecuc(Tm/TmPreDefTimerInstance/TmPreDefTimerInstanceId)}	
Variation	{ecuc(Tm/TmPreDefTimerInstance/TmTimerUser) == PORT AND Name = {ecuc(Tm/TmPreDefTimerInstance.SHORT-NAME)}		

⌋

8.7.2 Client-Server-Interfaces

The offered ClientServerInterface provides functionality to reset, get and shift a timer. The C API additionally offers `Tm_SyncTimer` and `Tm_BusyWait1us` which are not part of the ClientServerInterface. The reasons are: SWCs should not really perform busy waiting and for synchronization the SWC would require access to the generated symbols of the instance Ids which is currently not possible for the application.

8.7.2.1 TmPreDefTimerService

[SWS_Tm_91008] Definition of ClientServerInterface TmPreDefTimerService ⌈

Name	TmPreDefTimerService		
Comment	–		
IsService	true		
Variation	–		
Possible Errors	0	E_OK	Operation successful
	1	E_NOT_OK	Operation failed

Operation	GetTimeSpan	
Comment	–	
Mapped to API	Tm_GetTimeSpan	
Variation	–	
Parameters	timeSpan	
	Type	uint32

▽

△

	Direction	OUT
	Comment	The current value of the timer.
	Variation	–
Possible Errors	E_OK E_NOT_OK	

Operation	ResetTimer
Comment	This service sets the reference time to the current value of the underlying predef timer
Mapped to API	Tm_ResetTimer
Variation	–
Possible Errors	E_OK E_NOT_OK

Operation	ShiftTimer	
Comment	This service shifts the reference time by the given value.	
Mapped to API	Tm_ShiftTimer	
Variation	–	
Parameters	value	
	Type	uint32
	Direction	IN
	Comment	The value by which the timer is shifted.
	Variation	–
Possible Errors	E_OK	

⌋

9 Sequence diagrams

9.1 Tm Normal Operation

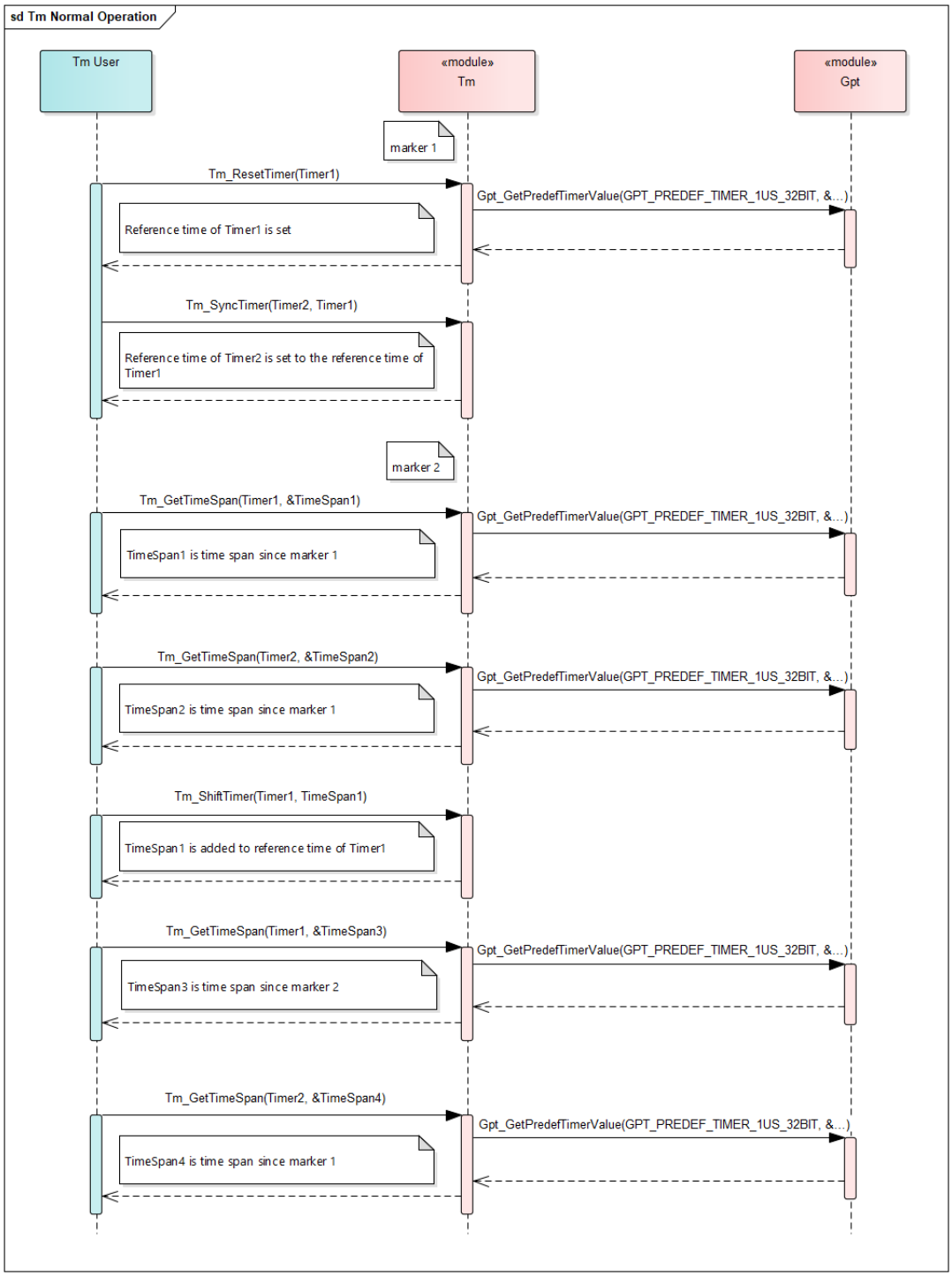

Figure 9.1: Sequence diagram TmNormalOperation

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Tm.

Chapter 10.3 specifies published information of the module Tm.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in [3, SWS BSW General].

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

10.2.1 Tm

[ECUC_Tm_00008] Definition of EcucModuleDef Tm ⌈

Module Name	Tm
Description	Configuration of the Time Service module.
Post-Build Variant Support	false
Supported Config Variants	VARIANT-PRE-COMPILE

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
TmGeneral	1	General configuration of Time Service module.
TmPreDefTimerInstance	0..*	Contains all configurable elements for timer instances

⌋

10.2.2 TmGeneral

[ECUC_Tm_00001] Definition of EcucParamConfContainerDef TmGeneral ⌈

Container Name	TmGeneral
Parent Container	Tm
Description	General configuration of Time Service module.
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
TmDevErrorDetect	1	[ECUC_Tm_00002]
TmVersionInfoApi	1	[ECUC_Tm_00007]

No Included Containers

⌋

[ECUC_Tm_00002] Definition of EcucBooleanParamDef TmDevErrorDetect ⌈

Parameter Name	TmDevErrorDetect		
Parent Container	TmGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Tm_00007] Definition of EcucBooleanParamDef TmVersionInfoApi ⌈

Parameter Name	TmVersionInfoApi		
Parent Container	TmGeneral		
Description	Adds / removes the service Tm_GetVersionInfo() from the code. ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants

▽

△

	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

10.2.3 TmPreDefTimerInstance

[ECUC_Tm_00009] Definition of EcucParamConfContainerDef TmPreDefTimerInstance ⌈

Container Name	TmPreDefTimerInstance
Parent Container	Tm
Description	Contains all configurable elements for timer instances
Post-Build Variant Multiplicity	false
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
TmPreDefTimerInstanceId	1	[ECUC_Tm_00010]
TmPreDefTimerType	1	[ECUC_Tm_00011]
TmTimerUser	1	[ECUC_Tm_00012]

No Included Containers

⌋

[ECUC_Tm_00010] Definition of EcucIntegerParamDef TmPreDefTimerInstanceId ⌈

Parameter Name	TmPreDefTimerInstanceId		
Parent Container	TmPreDefTimerInstance		
Description	Instance handle of the timer.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 4294967295		
Default value	–		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	

▽

△

Scope / Dependency	scope: local withAuto = true

⌋

[ECUC_Tm_00011] Definition of EcucEnumerationParamDef TmPreDefTimer Type ⌈

Parameter Name	TmPreDefTimerType		
Parent Container	TmPreDefTimerInstance		
Description	Configures the type of PreDef timer used from GPT		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	TM_PREDEF_TIMER_100US_32BIT	predef timer with 100us resolution and 32bit width	
	TM_PREDEF_TIMER_1US_16BIT	predef timer with 1us resolution and 16bit width.	
	TM_PREDEF_TIMER_1US_24BIT	predef timer with 1us resolution and 24bit width.	
	TM_PREDEF_TIMER_1US_32BIT	predef timer with 1us resolution and 32bit width.	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

⌋

[ECUC_Tm_00012] Definition of EcucEnumerationParamDef TmTimerUser ⌈

Parameter Name	TmTimerUser		
Parent Container	TmPreDefTimerInstance		
Description	Specifies if timer instance is used via C-API or via port interface.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CAPI	timer instance is used via C-API	
	PORT	timer instance is used via port interface.	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

⌋

10.3 Published Information

For details refer to the chapter 10.3 “Published Information” in [3, SWS BSW General].

A Not applicable requirements

[SWS_Tm_NA_00059]

Upstream requirements: SRS_BSW_00344, SRS_BSW_00159, SRS_BSW_00167, SRS_BSW_00170, SRS_BSW_00398, SRS_BSW_00416, SRS_BSW_00437, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00005, SRS_BSW_00415, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00160, SRS_BSW_00007, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00373, SRS_BSW_00335, SRS_BSW_00353, SRS_BSW_00328, SRS_BSW_00006, SRS_BSW_00439, SRS_BSW_00357, SRS_BSW_00377, SRS_BSW_00378, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00440, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00009, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00333, SRS_BSW_00321, SRS_BSW_00341

⌈These requirements are not applicable to this specification.⌋

B History of Constraints and Specification Items

B.1 Differences between R23-11 and R24-11

B.1.1 Added Specification Items in R24-11

[ECUC_Tm_00009] [ECUC_Tm_00010] [ECUC_Tm_00011] [ECUC_Tm_00012] [SWS_Tm_00068] [SWS_Tm_00069] [SWS_Tm_00070] [SWS_Tm_00071] [SWS_Tm_00077] [SWS_Tm_00082] [SWS_Tm_91000] [SWS_Tm_91001] [SWS_Tm_91002] [SWS_Tm_91003] [SWS_Tm_91004] [SWS_Tm_91005] [SWS_Tm_91006] [SWS_Tm_91007] [SWS_Tm_91008]

B.1.2 Changed Specification Items in R24-11

[ECUC_Tm_00001] [ECUC_Tm_00008] [SWS_Tm_00001] [SWS_Tm_00006] [SWS_Tm_00008] [SWS_Tm_00009] [SWS_Tm_00010] [SWS_Tm_00012] [SWS_Tm_00013] [SWS_Tm_00014] [SWS_Tm_00016] [SWS_Tm_00018] [SWS_Tm_00019] [SWS_Tm_00020] [SWS_Tm_00021] [SWS_Tm_00022] [SWS_Tm_00023] [SWS_Tm_00024] [SWS_Tm_00028] [SWS_Tm_00064] [SWS_Tm_00065]

B.1.3 Deleted Specification Items in R24-11

[ECUC_Tm_00003] [ECUC_Tm_00004] [ECUC_Tm_00005] [ECUC_Tm_00006] [SWS_Tm_00002] [SWS_Tm_00003] [SWS_Tm_00004] [SWS_Tm_00005] [SWS_Tm_00007] [SWS_Tm_00011] [SWS_Tm_00015] [SWS_Tm_00017] [SWS_Tm_00025] [SWS_Tm_00026] [SWS_Tm_00027] [SWS_Tm_00032] [SWS_Tm_00033] [SWS_Tm_00034] [SWS_Tm_00035] [SWS_Tm_00038] [SWS_Tm_00039] [SWS_Tm_00040] [SWS_Tm_00041] [SWS_Tm_00042] [SWS_Tm_00043] [SWS_Tm_00044] [SWS_Tm_00045] [SWS_Tm_00046] [SWS_Tm_00047] [SWS_Tm_00048] [SWS_Tm_00049] [SWS_Tm_00050] [SWS_Tm_00051] [SWS_Tm_00052] [SWS_Tm_00053] [SWS_Tm_00054] [SWS_Tm_00055] [SWS_Tm_00056]

B.1.4 Added Constraints in R24-11

[SWS_Tm_CONSTR_00001] [SWS_Tm_CONSTR_00002]

B.1.5 Changed Constraints in R24-11

none

B.1.6 Deleted Constraints in R24-11

none

B.2 Differences between R22-11 and R23-11

B.2.1 Added Specification Items in R23-11

none

B.2.2 Changed Specification Items in R23-11

none

B.2.3 Deleted Specification Items in R23-11

none

C Migration from Operating System

In previous releases of AUTOSAR only the Operating System (Os) offered a timer related port interface to software components. This Os interface was replaced by a service interface of the Time Service module (see chapter 8.7). The following hints shall help users which have used the port interface from the Os to migrate their application to use the Tm module:

Availability of ports

The Os offered a port per Counter object. Counter objects are similar to timer instances of the Time Service module. The Tm module offers a port per timer instance, depending on the configuration (see `TmTimerUser`). When migrating an application from using Os ports to Tm it is suggested to replace the used Os ports by a Tm port of a timer instance which is accordingly configured.

Port interfaces

The Os offered a client server interface with two operations: `GetCounterValue` and `GetElapsedValue`. The client server interface of the Tm module provides operations for resetting, getting, shifting and syncing (same functionality as provided by `Tm_ResetTimer`, `Tm_GetTimeSpan`, `Tm_ShiftTimer`, and `Tm_SyncTimer`).

The `GetCounterValue` operation can be replaced by a combination of `ResetTimer`, `ShiftTimer` and `GetTimeSpan` operations. The difference is that the Os function always returns the absolute value of the Os Counter. The Tm operations return always relative values. A solution can be to shift the reference time to "0" so that the relative and absolute values are the same.

The `GetElapsedValue` operation can be replaced by `GetTimeSpan`.