Document Title	Specification of Time Service
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	624
Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History

Date	Release	Changed by	Change Description
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 the "configuration variants" from 10.2.1 Variants • Added line "Supported Config Variants" to the table of the 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
2015-07-31	4.2.2	AUTOSAR Release Management	• Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes

Document Change History

Date	Release	Changed by	Change Description
2013-03-15	4.1.1	AUTOSAR Administration	• 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 work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

Table of 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

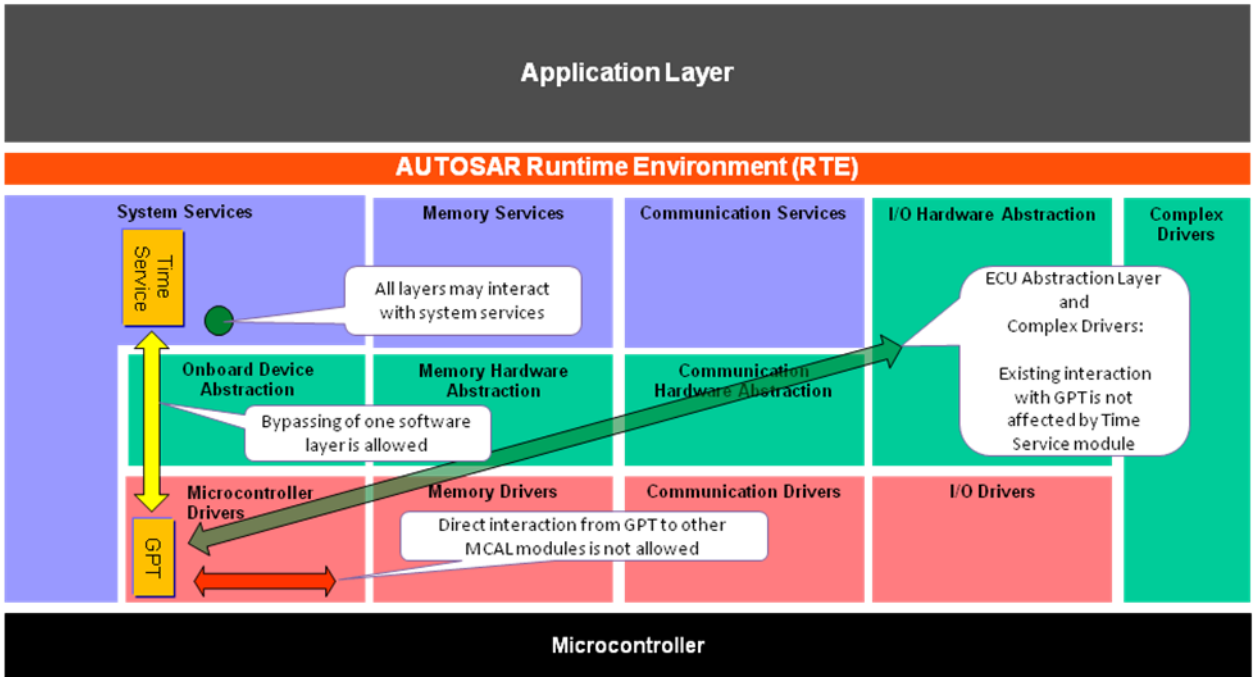

Figure 1 – Architectural overview

The Time Service module does not use and distribute all features of the GPT driver. The Time Service module is not the top of a “Timer Stack”.

Several “timer types” - 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). 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.

The following Time Service Predef Timers are defined:
- Tm_PredefTimer1us16bitType
- Tm_PredefTimer1us24bitType
- Tm_PredefTimer1us32bitType
- Tm_PredefTimer100us32bitType

If a user wants to implement a time-based functionality, no user specific configuration of the Time Service module is necessary. The user can instantiate any timers (only limited by available memory) and can use the timer instances completely independently. So, hardware timers are reused.

The following time based services are provided ("…" means: extension on the left side):
- Tm_ResetTimer…
- Tm_GetTimeSpan…
- Tm_ShiftTimer…
- Tm_SyncTimer…
- Tm_BusyWait…

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

The time services can be used in:
- Initialization phase
- 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 AUTOSAR glossary [8].

Acronym / Abbreviation	***Description***
nop	No Operation

Table 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. 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. For each Predef Timer a set of API services is provided by the Time Service module. The user can instantiate any timers (only limited by available memory) and can use the instances completely independently of each other.
Timer instance	A timer instance is a data object of an API data type `Tm_PredefTimer…bitType`, this means it is an instantiation of a Time Service Predef Timer on user software level. The user can instantiate any timers (only limited by available memory). The timer instances can be used completely independently of each other by methodes provided as API services.
Reference time	The reference time is a time value stored for each timer instance. It's an implementation specific element of the API data types `Tm_PredefTimer…bitType`.

Table 2: Terms

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[5] Specification of Default Error Tracer,
AUTOSAR_SWS_ DefaultErrorTracer.pdf

[6] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[7] Requirements on Time Service,
AUTOSAR_SRS_TimeService.pdf

[8] Glossary,
AUTOSAR_TR_Glossary.pdf

[9] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[10] General Specification of Basic Software Modules,
AUTOSAR_SWS_BSWGeneral.pdf

[11] Specification of GPT Driver,
AUTOSAR_SWS_GPTDriver.pdf

3.2 Related standards and norms

[12] IEC 7498-1 The Basic Model, IEC Norm, 1994

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [10] (SWS BSW General), which is also valid for Time Service.

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

4 Constraints and assumptions

4.1 Assumptions

No assumptions.

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.
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.

No Standardized AUTOSAR Interfaces
In this specification no Standardized AUTOSAR Interfaces are defined. This means the services of the Time Service module are not accessible by AUTOSAR Software Components (SW-Cs) which are located above the RTE. In a further step (future AUTOSAR release/revision) the Standardized AUTOSAR Interfaces may be added to the specification.

Multi Partition Support
Because the Time Service module uses the GPT module to get the current time of a hardware timer both modules should run on the same BSW partition. If the Time Service module is used in systems with distributed BSW (e.g. in multi-core systems) it's recommended to have a functional cluster with a Time Service and GPT module in each BSW partition to prevent inter-partition communication.
A master/satellite approach with GPT and Time Service master in one BSW partition and Time service satellite in another BSW partition seams not appropriate due to performance reasons.

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 GPT driver, see [11] (SWS GPT Driver).

6 Requirements traceability

This chapter refers to input requirements specified in the SRS documents (Software Requirements Specifications) that are applicable for this software module.

The table below lists links to specification items of the SWS document, which satisfy the input requirements. Only functional requirements are referenced.

Requirement	Description	Satisfied by
SRS_BSW_00005	Modules of the µC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Tm_00059
SRS_BSW_00006	The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Tm_00059
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	SWS_Tm_00059
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Tm_00059
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_Tm_00059
SRS_BSW_00159	All modules of the AUTOSAR Basic Software shall support a tool based configuration	SWS_Tm_00059
SRS_BSW_00160	Configuration files of AUTOSAR Basic SW module shall be readable for human beings	SWS_Tm_00059
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Tm_00059
SRS_BSW_00162	The AUTOSAR Basic Software shall provide a hardware abstraction layer	SWS_Tm_00059
SRS_BSW_00167	All AUTOSAR Basic Software Modules shall provide configuration rules	SWS_Tm_00059

	and constraints to enable plausibility checks	
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Tm_00059
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Tm_00059
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Tm_00059
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Tm_00059
SRS_BSW_00307	Global variables naming convention	SWS_Tm_00059
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Tm_00059
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Tm_00059
SRS_BSW_00312	Shared code shall be reentrant	SWS_Tm_00007, SWS_Tm_00011, SWS_Tm_00017, SWS_Tm_00020, SWS_Tm_00025
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_Tm_00059
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
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_Tm_00059
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Tm_00059
SRS_BSW_00330	It shall be allowed to use	SWS_Tm_00059

	macros instead of functions where source code is used and runtime is critical	
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_Tm_00059
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_Tm_00059
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Tm_00059
SRS_BSW_00335	Status values naming convention	SWS_Tm_00059
SRS_BSW_00337	Classification of development errors	SWS_Tm_00030
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Tm_00059
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Tm_00059
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Tm_00059
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Tm_00059
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_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_Tm_00059
SRS_BSW_00357	For success/failure of an API call a standard return type shall be defined	SWS_Tm_00059
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_Tm_00059
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	SWS_Tm_00059
SRS_BSW_00361	All mappings of not	SWS_Tm_00059

	standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	
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_00038, SWS_Tm_00039, SWS_Tm_00043, SWS_Tm_00044, SWS_Tm_00048, SWS_Tm_00049, SWS_Tm_00053, SWS_Tm_00054, SWS_Tm_00066
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	SWS_Tm_00059
SRS_BSW_00377	A Basic Software Module can return a module specific types	SWS_Tm_00059
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Tm_00059
SRS_BSW_00398	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	SWS_Tm_00059
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_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_Tm_00059
SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_Tm_00059
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Tm_00059
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Tm_00059
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_Tm_00059
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Tm_00059
SRS_BSW_00424	BSW module main	SWS_Tm_00059

	processing functions shall not be allowed to enter a wait state	
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_Tm_00059
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Tm_00059
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Tm_00059
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_Tm_00059
SRS_BSW_00429	Access to OS is restricted	SWS_Tm_00059
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_Tm_00059
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Tm_00059
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_Tm_00059
SRS_BSW_00439	Enable BSW modules to handle interrupts	SWS_Tm_00059
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_Tm_00059
SRS_Tm_00001	Different types of Predef Timers shall be supported by the Time Service module	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
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_00002, SWS_Tm_00003, SWS_Tm_00004, SWS_Tm_00005, SWS_Tm_00057
SRS_Tm_00003	The Time Service module shall make it possible to configure which Predef Timers are enabled	SWS_Tm_00026, SWS_Tm_00027
SRS_Tm_00004	The Time Service module shall provide a synchronous service to reset a timer instance	SWS_Tm_00038, SWS_Tm_00043, SWS_Tm_00048, SWS_Tm_00053
SRS_Tm_00005	The Time Service module shall provide a synchronous service to get the time span	SWS_Tm_00009, SWS_Tm_00039, SWS_Tm_00044, SWS_Tm_00049, SWS_Tm_00054
SRS_Tm_00006	The Time Service module shall provide a synchronous service to shift the reference time of a timer instance	SWS_Tm_00006, SWS_Tm_00013, SWS_Tm_00040, SWS_Tm_00045, SWS_Tm_00050, SWS_Tm_00055
SRS_Tm_00007	The Time Service module shall provide a synchronous service to synchronize two timer instances	SWS_Tm_00019, SWS_Tm_00041, SWS_Tm_00046, SWS_Tm_00051, SWS_Tm_00056
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_00042, SWS_Tm_00047, SWS_Tm_00052

7 Functional specification

7.1 General behavior

7.1.1 GPT Predef Timers

The functionality of the Time Service module is based on so called “GPT Predef Timers”, see [11] (SWS GPT Driver).

7.1.2 Time Service Predef Timers

A Time Service Predef Timer is based on the corresponding GPT Predef Timer.

For each Time Service Predef Timer a data type is defined.

Data type name of Time Service Predef Timer	Tick duration	Maximum tick value	Number of bits	Maximum time span (circa values)
Tm_PredefTimer1us16bitType	1 µs	65535	16 bit	65 ms
Tm_PredefTimer1us24bitType		16777215	24 bit	16 s
Tm_PredefTimer1us32bitType		4294967295	32 bit	71 minutes
Tm_PredefTimer100us32bitType	100 µs	4294967295	32 bit	4.9 days

Table 3 - Characteristics of Time Service Predef Timers

A timer instance can be created by defining a data object (RAM data) of a “Time Service Predef Timer data type”, for example:

```
Tm_PredefTimer1us32bitType Timer1;   /* Define timer instance */
```

The data type (and so the timer instance) contains a so called “reference time”. This reference time is necessary for some API services.

The detailed definition of the data types is out of scope of this specification, because the structure element(s) shall not be used outside the Time Service module.

Example for data type Tm_PredefTimer1us32bitType:

```
typedef struct
{
   uint32 ui32RefTime;  /* Reference time of the timer */
} Tm_PredefTimer1us32bitType;
```

Each Time Service Predef Timer has its own set of API services, due to performance reasons, especially for the 1µs timers. The services provide “simple” functionalitiy like a stopwatch:

- ResetTimer
- GetTimeSpan

- ShiftTimer
- SyncTime
- BusyWait (only for 1µs timers)

Each service has at least one parameter (e.g. TimerPtr), which is a pointer to a timer instance defined on user software level.

The service names are built of two parts:
- Part1: what to do, e.g. `Tm_ResetTimer`
- Part2: which Predef Timer type is used, e.g. `1us32bit`

Example of service name: `Tm_ResetTimer1us32bit`

[SWS_Tm_00001] ⌈ The Time Service module shall use the GPT driver service `Gpt_GetPredefTimerValue` to get the current time value for the desired Predef Timer. ⌋ (SRS_Tm_00002)

[SWS_Tm_00002] ⌈ The "`1us16bit`" functions shall use the Timer GPT_PREDEF_TIMER_1US_16BIT as time base if a time base is needed. ⌋ (SRS_Tm_00002)

An example for a "`1us16bit`" function is: `Tm_ResetTimer1us16bit`

[SWS_Tm_00003] ⌈ The "`1us24bit`" functions shall use the Timer GPT_PREDEF_TIMER_1US_24BIT as time base if a time base is needed. ⌋ (SRS_Tm_00002)

[SWS_Tm_00004] ⌈ The "`1us32bit`" functions shall use the Timer GPT_PREDEF_TIMER_1US_32BIT as time base if a time base is needed. ⌋ (SRS_Tm_00002)

[SWS_Tm_00005] ⌈ The "`100us32bit`" functions shall use the Timer GPT_PREDEF_TIMER_100US_32BIT as time base if a time base is needed. ⌋ (SRS_Tm_00002)

7.1.3 Maximal measurable time span

This chapter has to be considered on user software level.

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 GetTimeSpan functions, see SWS_Tm_00010.

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

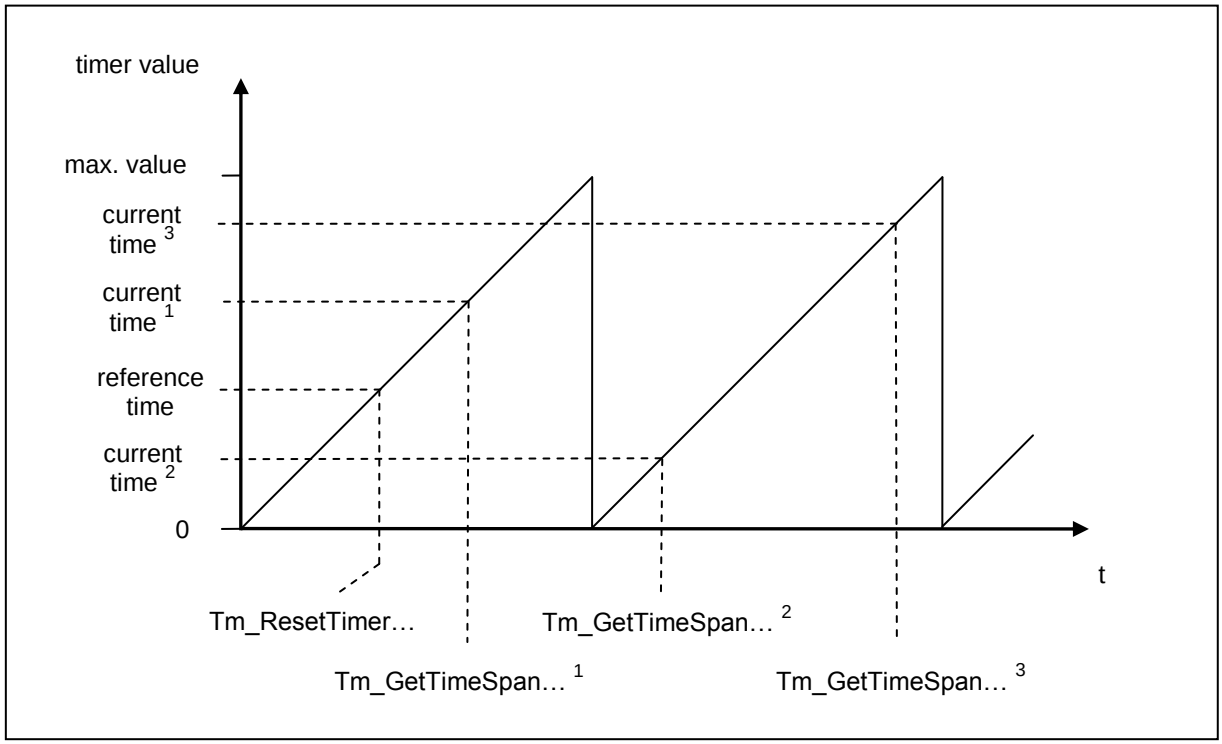

Figure 2 – Free running up counter

By calling `Tm_ResetTimer…` the current time of the related GPT Predef Timer is stored as a reference time. For details see chapter 7.1.6.

By calling `Tm_GetTimeSpan…` the time difference between the current time and the reference time is calculated and delivered. For details see chapter 7.1.7.

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 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 behaviour of BusyWait services" in chapter 7.1.10.1.

To ensure correct behavior under every possible circumstance, the user of the 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

7.1.4 Time quantization error

This chapter has to be considered on user software level.

The theory of quantization error has to be considered at using/interpretation of the values delivered by the GetTimeSpan functions.
The value delivered by a GetTimeSpan function has an accuracy of +/- 1 tick.

For example:

Value delivered by GetTimeSpan function		Real time minimum	Real time maximum	Comment
Value	Tick duration			
1	µs	nearly 0 µs	nearly 2 µs	See figure Time quantization example diagram
3400	µs	nearly 3399 µs	nearly 3401 µs	
56	100µs	nearly 5500 µs	nearly 5700 µs	

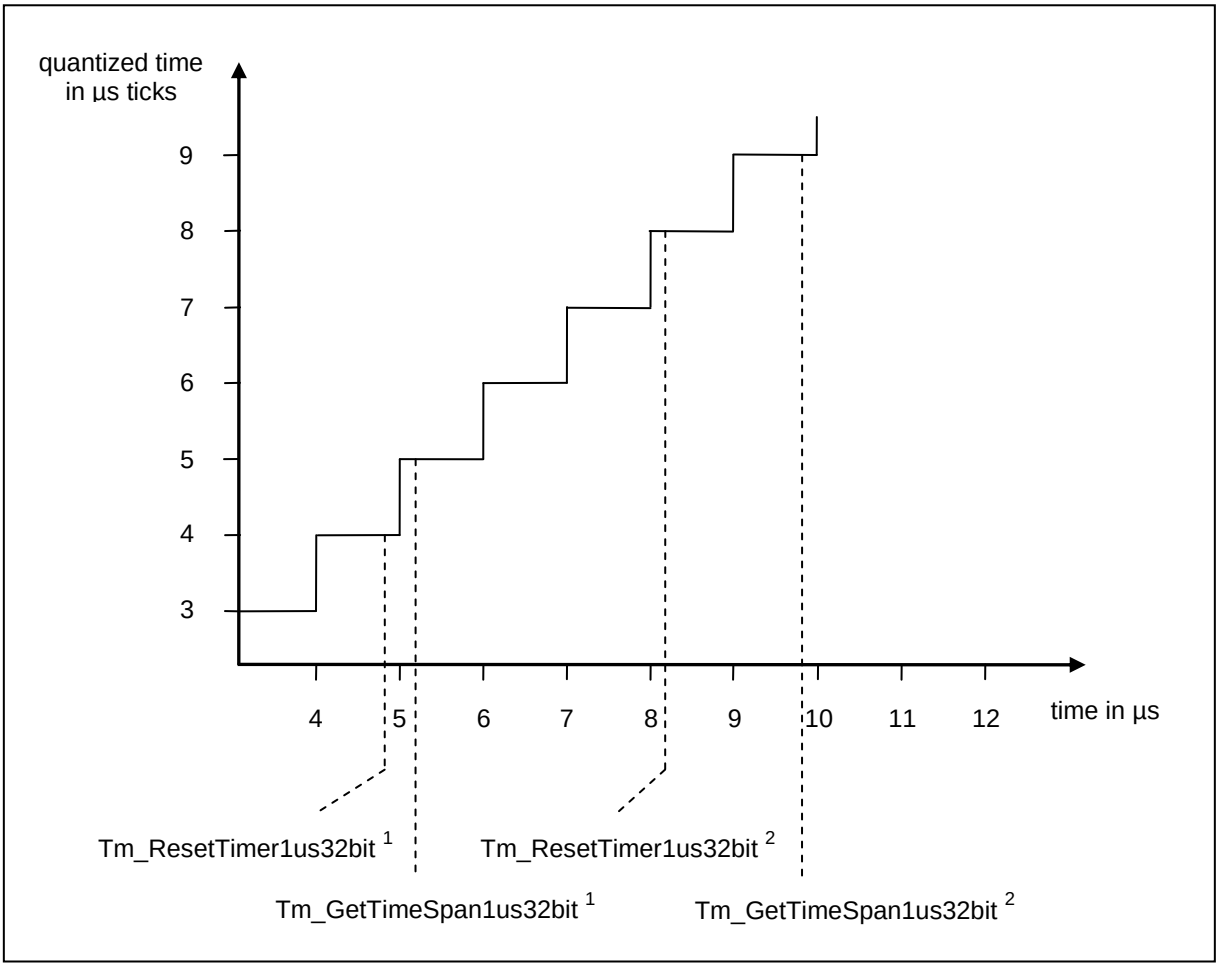

Figure 3 – Time quantization example diagram

In the example diagram above both calls of `Tm_GetTimeSpan1us32bit` ([1] and [2]) deliver the value 1, this means 1µs.

Depending on points in time the calls of `Tm_ResetTimer1us32bit` and `Tm_GetTimeSpan1us32bit` occur, the real time span can be in a range nearly 0µs to nearly 2µs.

If a 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 BusyWait services, see chapter 7.1.10

7.1.5 Execution times of services / measurement of short time spans

This chapter has to be considered on user software level.

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 [11] (SWS GPT Driver)

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

7.1.6 Service ResetTimer

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

An example for a ResetTimer function is: `Tm_ResetTimer1us32bit`

[SWS_Tm_00006] ⌈ The ResetTimer functions shall reset the timer instance passed by the parameter `TimerPtr`. This means, the reference time of the timer instance shall be set to the current time of the related GPT Predef Timer. ⌋ (SRS_Tm_00006)

[SWS_Tm_00007] ⌈ The ResetTimer functions shall be reentrant, if the timer instances used in concurrent calls are different. ⌋ (SRS_BSW_00312)

[SWS_Tm_00008] ⌈ If development error detection for the Time Service module is enabled:
If the pointer parameter is a null pointer, the ResetTimer functions shall raise the error `TM_E_PARAM_POINTER` and shall return `E_NOT_OK`. ⌋ (SRS_BSW_00369, SRS_BSW_00323)

7.1.7 Service GetTimeSpan

An example for a GetTimeSpan function is: `Tm_GetTimeSpan1us32bit`

[SWS_Tm_00009] ⌈ The GetTimeSpan functions shall calculate and deliver the time difference between the current time and the reference time of the timer instance. ⌋ (SRS_Tm_00005)

Note:
The restriction of maximal measurable time span has to be considered on user software level, see chapter 7.1.3.

Note:
Because the GetTimeSpan functions 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 7.1.4.

[SWS_Tm_00010] ⌈ The GetTimeSpan functions shall perform proper wrap-around handling at subtraction (current time - reference time), if value of current time is less than value of reference time. ⌋ ()

Hint:
Proper wrap-around handling can be achieved e.g. by following C code:
For 16bit timer:

```
ui16TimeSpan = (uint16)(ui16CurrentTime
               - TimerPtr->ui16RefTime);
```

For 24bit timer:

```
ui32TimeSpan = (uint32)(ui32CurrentTime
               - TimerPtr->ui32RefTime)
                 & (uint32)0x00FFFFFFu;
```

For 32bit timer:

```
ui32TimeSpan = (uint32)(ui32CurrentTime
               - TimerPtr->ui32RefTime;
```

[SWS_Tm_00011] ⌈ The GetTimeSpan functions shall be fully reentrant, this means even for the same timer instance. ⌋ (SRS_BSW_00312)

[SWS_Tm_00012] ⌈ If development error detection for the Time Service module is enabled: If a pointer parameter is a null pointer, the GetTimeSpan functions shall raise the error `TM_E_PARAM_POINTER` and shall return `E_NOT_OK`. ⌋ (SRS_BSW_00369, SRS_BSW_00323)

[SWS_Tm_00065] ⌈ When an error is detected and the parameter `TimeSpanPtr` is not a null pointer, the GetTimeSpan functions shall deliver the time span "`0`". ⌋ ()

Note:
This is to achieve defined (repeatable) behavior on user software level, even if the return value (`E_OK, E_NOT_OK`) is not used.

7.1.8 Service ShiftTimer

An example for a ShiftTimer function is: `Tm_ShiftTimer1us32bit`

[SWS_Tm_00013] ⌈ The ShiftTimer functions shall shift the reference time of the timer instance. This means, the value `TimeValue` shall be added to the reference time of the timer instance. ⌋ (SRS_Tm_00006)

[SWS_Tm_00014] ⌈ The ShiftTimer functions shall perform proper wrap-around handling at adding (reference time + TimeValue), if the sum is greater than the maximum value of the timer. ⌋ ()

Hint:
Proper wrap-around handling can be achieved e.g. by following C code:
For 16bit timer:

```
TimerPtr->ui16RefTime = (uint16)(TimerPtr->ui16RefTime
                        + TimeValue);
```

For 24bit timer:

```
TimerPtr->ui32RefTime = (uint32)(TimerPtr->ui32RefTime
                        + TimeValue) & (uint32)0x00FFFFFFu;
```

For 32bit timer:

```
TimerPtr->ui32RefTime = (uint32)(TimerPtr->ui32RefTime
                        + TimeValue;
```

[SWS_Tm_00015] ⌈ The ShiftTimer functions with range 24bit shall limit the value of the parameter `TimeValue` to 0xFFFFFF. ⌋ ()

[SWS_Tm_00016] ⌈ If development error detection for the Time Service module is enabled: If the value of the parameter `TimeValue` is greater than 0xFFFFFF, the ShiftTimer functions with range 24bit shall raise the error `TM_E_PARAM_VALUE`. ⌋ (SRS_BSW_00323)

[SWS_Tm_00017] ⌈ The ShiftTimer functions shall be reentrant, if the timer instances used in concurrent calls are different. ⌋ (SRS_BSW_00312)

[SWS_Tm_00018] ⌈ If development error detection for the Time Service module is enabled: If the pointer parameter is a null pointer, the ShiftTimer functions shall raise the error `TM_E_PARAM_POINTER`. ⌋ (SRS_BSW_00323)

7.1.9 Service SyncTimer

An example for a “SyncTimer” function is: `Tm_SyncTimer1us32bit`

[SWS_Tm_00019] ⌈ The SyncTimer functions 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. ⌋ (SRS_Tm_00007)

[SWS_Tm_00020] ⌈ The SyncTimer functions shall be reentrant, if the destination timer instances used in concurrent calls are different. ⌋ (SRS_BSW_00312)

[SWS_Tm_00021] ⌈ If development error detection for the Time Service module is enabled: If a pointer parameter is a null pointer, the SyncTimer functions shall raise the error `TM_E_PARAM_POINTER`. ⌋ (SRS_BSW_00323)

7.1.10 Service BusyWait

The service BusyWait performs busy waiting (active waiting) by polling with a guaranteed minimum waiting time. The BusyWait 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 1.1.3

Note:
The specification of the BusyWait functions considers the theory of quantization error, see chapter 7.1.4.

Note:
Because the BusyWait service is based on polling, the user of the BusyWait service is responsible for avoiding unintentional behaviour, see chapter 7.1.10.1.

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

An example for a BusyWait function is: `Tm_BusyWait1us32bit`

[SWS_Tm_00022] ⌈ The BusyWait functions shall perform busy waiting for the minimum time passed by the parameter `WaitingTimeMin`.⌋ (SRS_Tm_00008)

[SWS_Tm_00023] ⌈ The BusyWait functions shall not disable the interrupts. This means the real waiting time may be greater than the desired waiting time. ⌋ (SRS_Tm_00008)

[SWS_Tm_00024] ⌈ The BusyWait functions shall guarantee the minimum waiting. This means, n+1 ticks must be observed to ensure that an interval of at least n ticks has passed. ⌋ (SRS_Tm_00008)

[SWS_Tm_00025] ⌈ The BusyWait functions shall be reentrant. ⌋ (SRS_BSW_00312)

[SWS_Tm_00066] ⌈ When an error is detected, the BusyWait functions shall return `E_NOT_OK` and shall abort "waiting" immediately. ⌋ (SRS_BSW_00369)

7.1.10.1 Unintentional behaviour of BusyWait services

This chapter has to be considered on user software level.

Because the BusyWait services are based on polling, the user of a BusyWait service is responsible for avoiding unintentional behaviour.

Example of unintentional behaviour:

Elapsed time in µs	16-bit base timer value in µs	Action
0	0	Task is in state Running Call of service `Tm_BusyWait1us16bit(50); /* Wait for 50us */`
2	2	Task goes in state Ready
21055	21055	Task still in state Ready
65535	65535	Task still in state Ready, wrap-around of timer value with next tick
65536	0	Task still in state Ready
65559	23	Task goes in state Running again. **Problem**: Busy wait service does not return although 65559µs (> 50µs) elapsed since calling.

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

- which Busy wait service is required/sufficient (`Tm_BusyWait1us16bit`, `Tm_BusyWait1us24bit`, `Tm_BusyWait1us32bit`)
- the task scheduling
- whether an interrupt or resource lock is necessary on user software level
- whether the user software is tolerant of such problems

By using the service `Tm_BusyWait1us32bit` a problem as described above can only occur, if a task which calls the busy wait service is preempted (not executed, in state Ready) for more than 71 minutes.

7.1.11 Configuration of API services

The Time Service module allows to configure which Predef Timers are enabled, see configuration parameters in chapter 10.
Example of configuration parameter: `TmEnablePredefTimer1us16bit` .

[SWS_Tm_00026] ⌈ For each Predef Timer enabled by configuration the following set of API services shall be available: ResetTimer, GetTimeSpan, ShiftTimer, SyncTimer. ⌋ (SRS_Tm_00003).

[SWS_Tm_00027] ⌈ For each Predef Timer with tick duration 1µs enabled by configuration the API service BusyWait shall be available. ⌋ (SRS_Tm_00003).

7.2 Module initialization

There is no requirement for an init function (Tm_Init).

No variables (e.g. states) or hardware resources have to be initialized by the Time Service module. All GPT Predef Timers required by the Time Service module (assumed to be configured correct) run automatically whenever possible. This is ensured by the GPT driver, see chapter 7.1.1.

For development error detection, please refer to chapter 7.6.

7.3 Sample code of use cases

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

7.3.1 Time measurement

Sometimes execution time of code shall be measured.

Sample code:

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

Tm_PredefTimer1us24bitType TimerIsr1;       /* Define timer instance */
Tm_PredefTimer1us24bitType TimerTask100ms; /* Define timer instance */
uint32 RunTimeIsr1_us;                      /* Gross runtime of Isr1      */
uint32 RunTimeTask100ms_us;                 /* Gross runtime of Task100ms */

ISR(Isr1)
{
  (void)Tm_ResetTimer1us24bit(&TimerIsr1);

  /* Code */

  (void)Tm_GetTimeSpan1us24bit(&TimerIsr1, &RunTimeIsr1_us);
}

TASK(Task100ms)
{
  (void)Tm_ResetTimer1us24bit(&TimerTask100ms);

  /* Code */

  (void)Tm_GetTimeSpan1us24bit(&TimerTask100ms, &RunTimeTask100ms_us);
  (void)TerminateTask();
}
```

7.3.2 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

uint8_least State = MY_INIT;

TASK(Task5ms)
{
  static Tm_PredefTimer1us24bitType Timer;  /* Define timer instance */
  uint32 WaitingTime1_us = 500000u;         /* 500ms */
  uint32 WaitingTime2_us = 250000u;         /* 250ms */

  switch (State)
  {
    case MY_INIT:
    {
      (void)Tm_ResetTimer1us24bit(&Timer);
      State = MY_WAIT1;
      break;
    }
    case MY_WAIT1:
    {
      uint32 Time_us;
      (void)Tm_GetTimeSpan1us24bit(&Timer, &Time_us);

      if (Time_us >= WaitingTime1_us)
      {
        /* Action  ... */
        Tm_ShiftTimer1us24bit(&Timer, WaitingTime1_us);
        State = MY_WAIT2;
      }
      break;
    }
    case MY_WAIT2:
    {
      uint32 Time_us;
      (void)Tm_GetTimeSpan1us24bit(&Timer, &Time_us);

      if (Time_us >= WaitingTime2_us)
      {
        /* Action  ... */
        Tm_ShiftTimer1us24bit(&Timer, WaitingTime2_us);
        State = MY_WAIT1;
      }
      break;
    }
  }
  (void)TerminateTask();
}
```

7.3.3 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"

Tm_PredefTimer1us32bitType Timer1;    /* Define timer instance */
uint16 StatusRegisterBit0;
uint32 TimeElapsed_us;

void SampleFunction(void)
{
  (void)Tm_ResetTimer1us32bit(&Timer1);

  do
  {
    StatusRegisterBit0 = HW_STATUS_REG & 0x0001u;
    (void)Tm_GetTimeSpan1us32bit(&Timer1, &TimeElapsed_us);

  } while (    (StatusRegisterBit0 != 0x0001u) /* Wait until bit 0 is set*/
            && (TimeElapsed_us <= 40)          /* Timeout 40us */
          );
}
```

7.3.4 Busy waiting

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

Sample code:

```
#include "Tm.h"

Std_ReturnType CanTrcv_SetOpMode(uint8 Transceiver,
                                 CanIf_TrcvModeType OpMode)
{
  /* Code */
  switch(OpMode)
  {
    case CANIF_TRCV_MODE_NORMAL:
    {
      /* Code */
      break;
    }
    case CANIF_TRCV_MODE_SLEEP:
    {
      /* Code */
```

```
      SetPinEnableHigh();
      /* Busy waiting: 50us (for TJA1054: at least 50us) */
      (void)Tm_BusyWait1us32bit(50);
      SetPinEnableLow();
      /* Code */
      break;
    }
    case CANIF_TRCV_MODE_STANDBY:
    {
      /* Code */
      break;
    }
  }
  /* Code */
}
```

7.4 Version check

Please refer to chapter “Version Check” in *SWS_BSWGeneral.*

7.5 Error classification

7.5.1 Development Errors

[SWS_Tm_00028] ⌈ The following errors shall be detectable by the Time Service module depending on its build version (development / production):

Type of error	*Relevance*	*Related error code*	*Value [hex]*
API parameter checking: invalid pointer	Development	TM_E_PARAM_POINTER	0x01
API parameter checking: invalid value	Development	TM_E_PARAM_VALUE	0x02

⌋ ()

[SWS_Tm_00030] ⌈ 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. ⌋ (SRS_BSW_00337)

7.5.2 Runtime Errors

[SWS_Tm_00067]⌈

Type of error	*Relevance*	*Related error code*	*Value [hex]*
Access to underlying hardware timer failed	Runtime	TM_E_HARDWARE_TIMER	0x03

⌋ ()

7.5.3 Transient Faults

There are no transient faults.

7.5.4 Production Errors

No production errors are defined for the Time Service module.

7.5.5 Extended Production Errors

There are no extended production errors.

7.6 Error detection

Please refer to chapter “Error detection” in *SWS_BSWGeneral.*

[SWS_Tm_00063] ⌈ 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. ⌋ ()

[SWS_Tm_00064] ⌈ If the underlying GPT driver service returns `E_NOT_OK`, the functions ResetTimer, GetTimeSpan and BusyWait shall raise the error `TM_E_HARDWARE_TIMER`. ⌋ ()

7.7 Error notification

Please refer to chapter “Error notification” in *SWS_BSWGeneral.*

8 API specification

8.1 Imported types

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

[SWS_Tm_00031] [

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

] (SRS_BSW_00348)

8.2 Type Definitions

8.2.1 Tm_PredefTimer1us16bitType

[SWS_Tm_00032] [

Name:	`Tm_PredefTimer1us16bitType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data type of Time Service Predef Timer 1us16bit. The structure contains the reference time.
Available via:	`Tm.h`

] (SRS_Tm_00001)

8.2.2 Tm_PredefTimer1us24bitType

[SWS_Tm_00033] [

Name:	`Tm_PredefTimer1us24bitType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data type of Time Service Predef Timer 1us24bit. The structure contains the reference time.
Available via:	`Tm.h`

] (SRS_Tm_00001)

8.2.3 Tm_PredefTimer1us32bitType

[SWS_Tm_00034] [

Name:	`Tm_PredefTimer1us32bitType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data type of Time Service Predef Timer 1us32bit. The structure contains the reference time.
Available via:	`Tm.h`

] (SRS_Tm_00001)

8.2.4 Tm_PredefTimer100us32bitType

[SWS_Tm_00035] [

Name:	`Tm_PredefTimer100us32bitType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data type of Time Service Predef Timer 100µs32bit. The structure contains the reference time.
Available via:	`Tm.h`

] (SRS_Tm_00001)

8.3 Function definitions

8.3.1 Tm_GetVersionInfo

[SWS_Tm_00036] [

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`	

] (SRS_BSW_00407)

[SWS_Tm_00037] [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`.] (SRS_BSW_00323)

8.3.2 Tm_ResetTimer1us16bit

[SWS_Tm_00038] [

Service name:	Tm_ResetTimer1us16bit

Syntax:	`Std_ReturnType Tm_ResetTimer1us16bit(`
	`  Tm_PredefTimer1us16bitType* TimerPtr`
	`)`

Service ID[hex]:	0x2
Sync/Async:	Synchronous

Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	TimerPtr	Pointer to a timer instance defined by the user.
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Resets a timer instance (user point of view).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00004, SRS_BSW_00369)

8.3.3 Tm_GetTimeSpan1us16bit

[SWS_Tm_00039] [

Service name:	Tm_GetTimeSpan1us16bit	

Syntax:	`Std_ReturnType Tm_GetTimeSpan1us16bit(`
	`  const Tm_PredefTimer1us16bitType* TimerPtr,`
	`  uint16* TimeSpanPtr`
	`)`	

Service ID[hex]:	0x3	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimeSpanPtr	Pointer to time span destination data in RAM
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Delivers the time difference (current time - reference time).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00005, SRS_BSW_00369)

8.3.4 Tm_ShiftTimer1us16bit

[SWS_Tm_00040] [

Service name:	Tm_ShiftTimer1us16bit	

Syntax:	`void Tm_ShiftTimer1us16bit(`
	`  Tm_PredefTimer1us16bitType* TimerPtr,`
	`  uint16 TimeValue`
	`)`	

Service ID[hex]:	0x4	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	TimeValue	Time value in µs, the reference time has to be shifted.
Parameters (inout):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (out):	None	
Return value:	None	
Description:	Shifts the reference time of the timer instance.	

Available via:	`Tm.h`

] (SRS_Tm_00001, SRS_Tm_00006)

8.3.5 Tm_SyncTimer1us16bit

[SWS_Tm_00041] [

Service name:	Tm_SyncTimer1us16bit	

Syntax:	`void Tm_SyncTimer1us16bit(`
	`  Tm_PredefTimer1us16bitType* TimerDstPtr,`
	`  const Tm_PredefTimer1us16bitType* TimerSrcPtr`
	`)`	

Service ID[hex]:	0x5	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same destination timer instance	
Parameters (in):	TimerSrcPtr	Pointer to the source timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimerDstPtr	Pointer to the destination timer instance defined by the user.
Return value:	None	
Description:	Synchronizes two timer instances.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00007)

8.3.6 Tm_BusyWait1us16bit

[SWS_Tm_00042] [

Service name:	Tm_BusyWait1us16bit	

Syntax:	`Std_ReturnType Tm_BusyWait1us16bit(`
	`  uint8 WaitingTimeMin`
	`)`	

Service ID[hex]:	0x6	
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 and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Performs busy waiting by polling with a guaranteed minimum waiting time.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00008)

Note:
Because the BusyWait service is based on polling, the user of the BusyWait service is responsible for avoiding unintentional behaviour, see chapter 7.1.10 Service BusyWait.

8.3.7 Tm_ResetTimer1us24bit

[SWS_Tm_00043] [

Service name:	Tm_ResetTimer1us24bit	

Syntax:	`Std_ReturnType Tm_ResetTimer1us24bit(`
	`  Tm_PredefTimer1us24bitType* TimerPtr`
	`)`	

Service ID[hex]:	0x7	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	TimerPtr	Pointer to a timer instance defined by the user.
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Resets a timer instance (user point of view).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00004, SRS_BSW_00369)

8.3.8 Tm_GetTimeSpan1us24bit

[SWS_Tm_00044] [

Service name:	Tm_GetTimeSpan1us24bit	

Syntax:	`Std_ReturnType Tm_GetTimeSpan1us24bit(`
	`  const Tm_PredefTimer1us24bitType* TimerPtr,`
	`  uint32* TimeSpanPtr`
	`)`	

Service ID[hex]:	0x8	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimeSpanPtr	Pointer to time span destination data in RAM
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Delivers the time difference (current time - reference time).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00005, SRS_BSW_00369)

8.3.9 Tm_ShiftTimer1us24bit

[SWS_Tm_00045] [

Service name:	Tm_ShiftTimer1us24bit

Syntax:	`void Tm_ShiftTimer1us24bit(`
	`  Tm_PredefTimer1us24bitType* TimerPtr,`
	`  uint32 TimeValue`
	`)`

Service ID[hex]:	0x9	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	TimeValue	Time value in µs, the reference time has to be shifted. Range: 0-0xFFFFFF
Parameters (inout):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (out):	None	
Return value:	None	
Description:	Shifts the reference time of the timer instance.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00006)

8.3.10 Tm_SyncTimer1us24bit

[SWS_Tm_00046] [

Service name:	Tm_SyncTimer1us24bit	

Syntax:	`void Tm_SyncTimer1us24bit(`
	`  Tm_PredefTimer1us24bitType* TimerDstPtr,`
	`  const Tm_PredefTimer1us24bitType* TimerSrcPtr`
	`)`	

Service ID[hex]:	0xa	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same destination timer instance	
Parameters (in):	TimerSrcPtr	Pointer to the source timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimerDstPtr	Pointer to the destination timer instance defined by the user.
Return value:	None	
Description:	Synchronizes two timer instances.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00007)

8.3.11 Tm_BusyWait1us24bit

[SWS_Tm_00047] [

Service name:	Tm_BusyWait1us24bit	

Syntax:	`Std_ReturnType Tm_BusyWait1us24bit(`
	`  uint8 WaitingTimeMin`
	`)`	

Service ID[hex]:	0xb	
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 and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Performs busy waiting by polling with a guaranteed minimum waiting time.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00008)
Note:
Because the BusyWait service is based on polling, the user of the BusyWait service is responsible for avoiding unintentional behaviour, see chapter 7.1.10 Service BusyWait.

8.3.12 Tm_ResetTimer1us32bit

[SWS_Tm_00048] [

Service name:	Tm_ResetTimer1us32bit	
Syntax:	`Std_ReturnType Tm_ResetTimer1us32bit( Tm_PredefTimer1us32bitType* TimerPtr )`	
Service ID[hex]:	0xc	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	TimerPtr	Pointer to a timer instance defined by the user.
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Resets a timer instance (user point of view).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00004, SRS_BSW_00369)

8.3.13 Tm_GetTimeSpan1us32bit

[SWS_Tm_00049] [

Service name:	Tm_GetTimeSpan1us32bit	
Syntax:	`Std_ReturnType Tm_GetTimeSpan1us32bit( const Tm_PredefTimer1us32bitType* TimerPtr, uint32* TimeSpanPtr )`	
Service ID[hex]:	0xd	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimeSpanPtr	Pointer to time span destination data in RAM
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Delivers the time difference (current time - reference time).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00005, SRS_BSW_00369)

8.3.14 Tm_ShiftTimer1us32bit

[SWS_Tm_00050] [

Service name:	Tm_ShiftTimer1us32bit	

Syntax:	`void Tm_ShiftTimer1us32bit(`
	`  Tm_PredefTimer1us32bitType* TimerPtr,`
	`  uint32 TimeValue`
	`)`	

Service ID[hex]:	0xe	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	TimeValue	Time value in µs, the reference time has to be shifted.
Parameters (inout):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (out):	None	
Return value:	None	
Description:	Shifts the reference time of the timer instance.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00006)

8.3.15 Tm_SyncTimer1us32bit

[SWS_Tm_00051] [

Service name:	Tm_SyncTimer1us32bit	

Syntax:	`void Tm_SyncTimer1us32bit(`
	`  Tm_PredefTimer1us32bitType* TimerDstPtr,`
	`  const Tm_PredefTimer1us32bitType* TimerSrcPtr`
	`)`	

Service ID[hex]:	0xf	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same destination timer instance	
Parameters (in):	TimerSrcPtr	Pointer to the source timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimerDstPtr	Pointer to the destination timer instance defined by the user.
Return value:	None	
Description:	Synchronizes two timer instances.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00007)

8.3.16 Tm_BusyWait1us32bit

[SWS_Tm_00052] [

Service name:	Tm_BusyWait1us32bit	

Syntax:	`Std_ReturnType Tm_BusyWait1us32bit(`
	`  uint8 WaitingTimeMin`
	`)`	

Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	WaitingTimeMin	Minimum waiting time in microseconds.
Parameters	None	

(inout):		
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Performs busy waiting by polling with a guaranteed minimum waiting time.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00008)

Note:
Because the BusyWait service is based on polling, the user of the BusyWait service is responsible for avoiding unintentional behaviour, see chapter 7.1.10 Service BusyWait.

8.3.17 Tm_ResetTimer100us32bit

[SWS_Tm_00053] [

Service name:	Tm_ResetTimer100us32bit	

Syntax:	`Std_ReturnType Tm_ResetTimer100us32bit(`
	`  Tm_PredefTimer100us32bitType* TimerPtr`
	`)`	

Service ID[hex]:	0x11	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	TimerPtr	Pointer to a timer instance defined by the user.
Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Resets a timer instance (user point of view).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00004, SRS_BSW_00369)

8.3.18 Tm_GetTimeSpan100us32bit

[SWS_Tm_00054] [

Service name:	Tm_GetTimeSpan100us32bit	

Syntax:	`Std_ReturnType Tm_GetTimeSpan100us32bit(`
	`  const Tm_PredefTimer100us32bitType* TimerPtr,`
	`  uint32* TimeSpanPtr`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimeSpanPtr	Pointer to time span destination data in RAM

Return value:	Std_ReturnType	E_OK: The underlying GPT driver service has returned E_OK and no development error has been detected E_NOT_OK: The underlying GPT driver service has returned E_NOT_OK, or a development error has been detected
Description:	Delivers the time difference (current time - reference time).	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00005, SRS_BSW_00369)

8.3.19 Tm_ShiftTimer100us32bit

[SWS_Tm_00055] [

Service name:	Tm_ShiftTimer100us32bit	

Syntax:	`void Tm_ShiftTimer100us32bit(`
	`  Tm_PredefTimer100us32bitType* TimerPtr,`
	`  uint32 TimeValue`
	`)`	

Service ID[hex]:	0x13	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same timer instance	
Parameters (in):	TimeValue	Time value in unit 100µs, the reference time has to be shifted.
Parameters (inout):	TimerPtr	Pointer to a timer instance defined by the user.
Parameters (out):	None	
Return value:	None	
Description:	Shifts the reference time of the timer instance.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00006)

8.3.20 Tm_SyncTimer100us32bit

[SWS_Tm_00056] [

Service name:	Tm_SyncTimer100us32bit	

Syntax:	`void Tm_SyncTimer100us32bit(`
	`  Tm_PredefTimer100us32bitType* TimerDstPtr,`
	`  const Tm_PredefTimer100us32bitType* TimerSrcPtr`
	`)`	

Service ID[hex]:	0x14	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant but not for the same destination timer instance	
Parameters (in):	TimerSrcPtr	Pointer to the source timer instance defined by the user.
Parameters (inout):	None	
Parameters (out):	TimerDstPtr	Pointer to the destination timer instance defined by the user.
Return value:	None	
Description:	Synchronizes two timer instances.	
Available via:	`Tm.h`	

] (SRS_Tm_00001, SRS_Tm_00007)

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] [

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.

] (SRS_Tm_00002)

8.6.2 Optional Interfaces

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

[SWS_Tm_00060] [

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 target function is usually a call-back function. The names of these kinds of interfaces is not fixed because they are configurable.

None.

9 Sequence diagrams

9.1 Tm Normal Operation

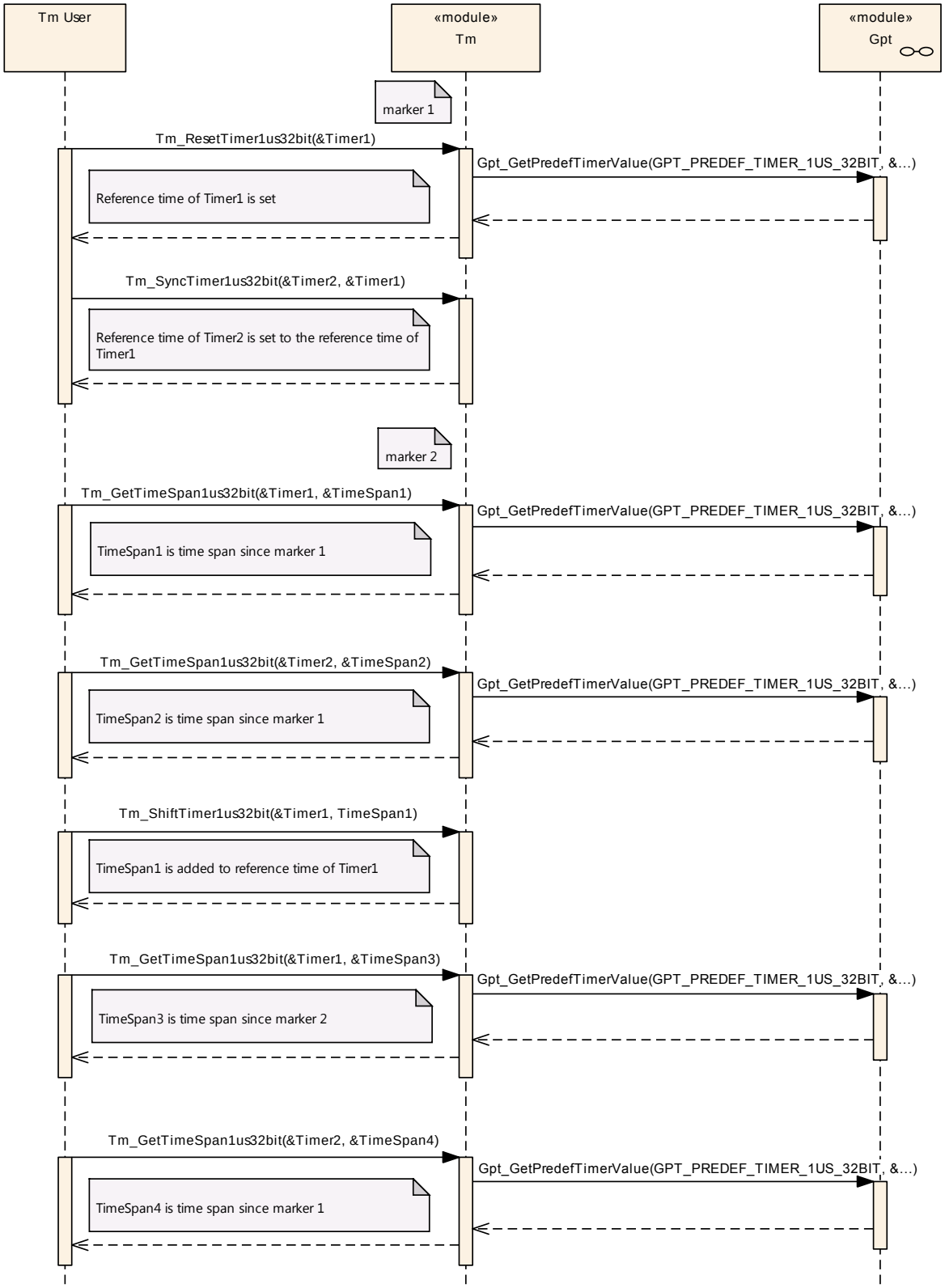

Figure 4 – Sequence diagram “Tm_Normal_Operation”

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 GPT

Chapter 10.3 specifies published information of the module

10.1 How to read this chapter

For details refer to the chapter 10.1 Introduction to configuration specification in SWS_BSWGeneral

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

SWS Item	**ECUC_Tm_00008 :**
Module Name	*Tm*
Module 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.

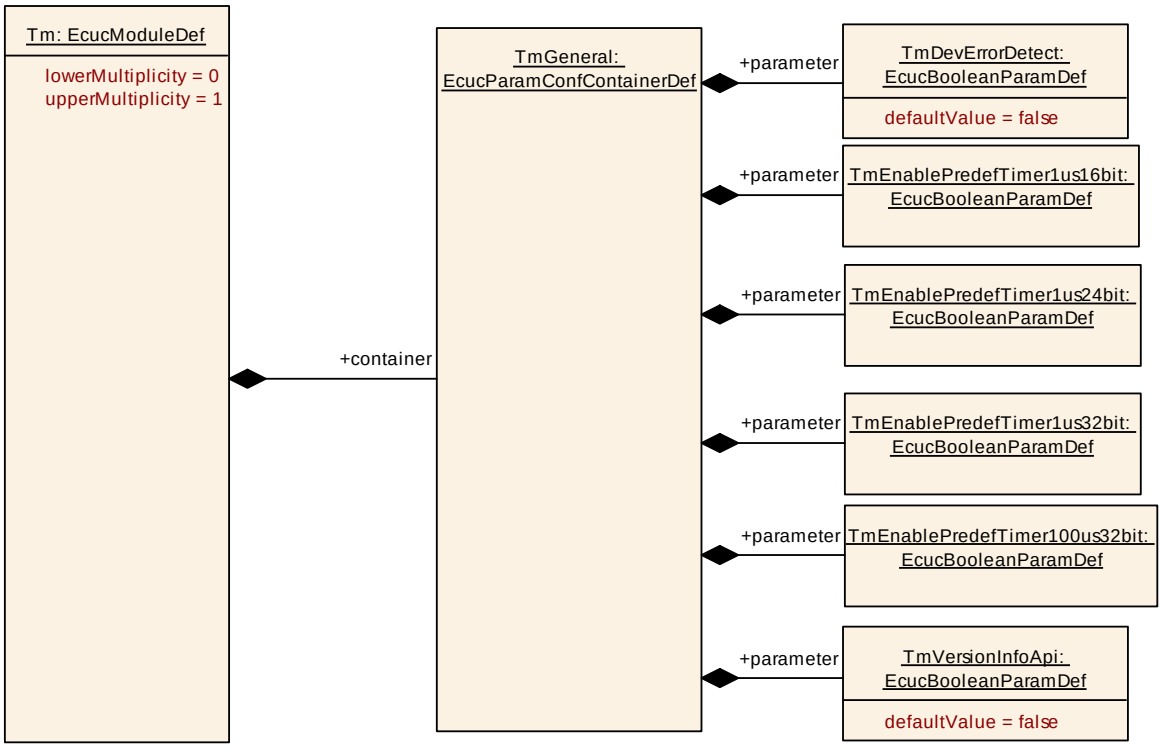

Figure 5 – Configuration Tm

10.2.2 TmGeneral

SWS Item	**ECUC_Tm_00001 :**
Container Name	TmGeneral
Description	General configuration of Time Service module.

Configuration Parameters

SWS Item	**ECUC_Tm_00002 :**		
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		

SWS Item	**ECUC_Tm_00006 :**		
Name	TmEnablePredefTimer100us32bit		
Parent Container	TmGeneral		
Description	Specifies if the Predef Timer 100µs32bit shall be enabled (functionality and set of API services). ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Tm_00003 :**		
Name	TmEnablePredefTimer1us16bit		
Parent Container	TmGeneral		
Description	Specifies if the Predef Timer 1µs16bit shall be enabled (functionality and set of API services). ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Tm_00004 :**		
Name	TmEnablePredefTimer1us24bit		
Parent Container	TmGeneral		
Description	Specifies if the Predef Timer 1µs24bit shall be enabled (functionality and set of API services). ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		

Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Tm_00005 :**		
Name	TmEnablePredefTimer1us32bit		
Parent Container	TmGeneral		
Description	Specifies if the Predef Timer 1µs32bit shall be enabled (functionality and set of API services). ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Tm_00007 :**		
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		

No Included Containers

10.3 Published Information

For details refer to the chapter 10.3 Published Information in SWS_BSWGeneral.

11 Not applicable requirements

[SWS_Tm_00059] ⌈ These requirements are not applicable to this specification.⌋ (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_00361, 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, SRS_BSW_00334)