**Architecture Design**

**Version: 0.0.15**

**[Author](app:ds:author):**

**Reviewer:**

**Approver:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Revision History** | | | | | |
| **Date** | **PIC** | **Version** | **Description** | **Reviewer**  **Data** | **Approver**  **Data** |
| 2022/5/31 | Xinrun.wang | 0.0.1 | File initialize |  |  |
| 2022/7/26 | Jian.Jiang | 0.0.2 | 1. Added chapter 5.8.3.Data Flow  2. Add captions to pictures |  |  |
| 2022/7/27 | Xinrun.wang | 0.0.3 | 1. Added [SA\_R2\_290] for MemoryMap  2. Add captions to pictures  3. Modify format for numbering |  |  |
| 2022/7/28 | Xinrun.wang | 0.0.4 | 1. Added [SA\_R2\_291], [SA\_R2\_293], [SA\_R2\_294], [SA\_R2\_295], [SA\_R2\_296], [SA\_R2\_297], [SA\_R2\_298],  2. Deleted[SA\_R2\_276], [SA\_R2\_265],  [SA\_R2\_256], [SA\_R2\_255], [SA\_R2\_253]  3. Add data flow section for some modules. |  |  |
| 2022/7/29 | Xinrun.wang | 0.0.5 | 1. Added numbering for control flow diagram  2. Deleted [SA\_R2\_297] |  |  |
| 2022/7/29 | Xinrun.wang | 0.0.6 | 1. Added [SA\_R2\_297], [SA\_R2\_299],  [SA\_R2\_300]-[SA\_R2\_306],  [SA\_R2\_308]-[SA\_R2\_312], [SA\_R2\_314],  [SA\_R2\_315], [SA\_R2\_316] for specific API definition. |  |  |
| 2022/7/29 | Tao.li | 0.0.7 | 1.Added [SA\_R2\_307], [SA\_R2\_313] for CDS's dataflow and queue's Dataflow |  |  |
| 2022/8/3 | Xinrun.wang | 0.0.8 | Modification for specific API definition:  1. Deleted [SA\_R2\_137], [SA\_R2\_179],  [SA\_R2\_204], [SA\_R2\_149], [SA\_R2\_232], [SA\_R2\_240]   1. Added [SA\_R2\_330]-[SA\_R2\_349],   [SA\_R2\_317]-[SA\_R2\_329],[SA\_R2\_350]   1. Modified [SA\_R2\_033],[SA\_R2\_084] |  |  |
| 2022/8/4 | zhiqiang.huang | 0.0.9 | Modification for specific API definition:  Modified [SA\_R2\_330]~[SA\_R2\_332] |  |  |
| 2022/8/9 | Xinrun.wang | 0.0.10 | 1. Correct numbering for 5.7.4.x  2. Added [SA\_R2\_353], [SA\_R2\_354], [SA\_R2\_336]  3. Deleted [SA\_R2\_169] |  |  |
| 2022/8/9 | qinchun.yang | 0.0.11 | 1.SA\_R2\_052 and SA\_R2\_053 repeat,delete SA\_R2\_053;  2.delete SA\_R2\_055 and SA\_R2\_056; |  |  |
| 2022/8/9 | Jian.Jiang | 0.0.12 | Add SA\_R2\_355 ~ SA\_R2\_362 |  |  |
| 2022/8/10 | Tao.li | 0.0.13 | Add SA\_R2\_351 |  |  |
| 2022/8/15 | Xinrun.Wang | 0.0.14 | Modified based on review:  1. Modify Reference file version  2. Modify Description for 3.3  3. Modify DataType define  4. Move SA\_R2\_037  5. Correct figure numbering. |  |  |
| 2022/8/18 | Jian.Jiang | 0.0.15 | Modified based on review:   1. SA\_R2\_134(1) |  |  |
| 2022/09/05 | qinchun.yang | 0.0.16 | Improve SA\_R2\_054 requirement description |  |  |
| 2022/09/06 | Zhengfei.li | 0.0.17 | 1. Updating SA\_R2\_016 Description  2. Delete SA\_R2\_073 |  |  |

目录

[1. General Information 4](#_Toc110415429)

[1.1. Purpose 4](#_Toc110415430)

[1.2. Intended Audiance 4](#_Toc110415431)

[1.3. Reference 4](#_Toc110415432)

[1.4. Abbreviations & Terms 5](#_Toc110415433)

[2. Operational environment & Constrains 10](#_Toc110415434)

[3. Software Architecture Overview 10](#_Toc110415435)

[3.1. File Structure 10](#_Toc110415436)

[3.2. Software Architecture 10](#_Toc110415437)

[3.3. Function Description 11](#_Toc110415438)

[3.4. Data Structure Define 13](#_Toc110415439)

[3.5. Data Types Define 13](#_Toc110415440)

[4. Dynamic Behavior Design 15](#_Toc110415441)

[4.1. State Mechanism 15](#_Toc110415442)

[5. RTE 16](#_Toc110415443)

[5.1. Function Description 16](#_Toc110415444)

[5.2. S-R Communciation 16](#_Toc110415445)

[5.3. Inter-Partition Communication 30](#_Toc110415446)

[5.4. C-S Communication 37](#_Toc110415447)

[5.5. Mode Management 44](#_Toc110415448)

[5.6. Internal/External Trigger 59](#_Toc110415449)

[5.7. Internal Behavior 64](#_Toc110415450)

[5.8. Data Transformation 77](#_Toc110415451)

[5.9. Calibration 97](#_Toc110415452)

[5.10. Initialization and Finalization 106](#_Toc110415453)

[5.11. Anxiliary Function 108](#_Toc110415454)

[5.12. Validation Criteria 112](#_Toc110415455)

[6. Memory Map Design 112](#_Toc110415456)

# General Information

## Purpose

This document describes the architecture and function of ORIENTAINS RTE that based on the specification AUTOSAR RTE and ISO26262 to satisfy the top-level requirements presented in the ‘PH-PRD-QC-002-2022\_SoftwareRequirementSpecification\_RTE.doc’.

## Intended Audiance

This document is written for architecture designer, requirement developer, software designer and developer, and testing developers.

## Reference

1. Virtual Functional Bus(4.2.2)

AUTOSAR\_EXP\_VFB

[2] Software Component Template(4.2.2)

AUTOSAR\_TPS\_SoftwareComponentTemplate

[3] Specification of Communication(4.2.2)

AUTOSAR\_SWS\_COM

1. Specification of RTE(4.2.2)

AUTOSAR\_SWS\_RTE

1. Requirements on Runtime Environment(4.2.2)

AUTOSAR\_SRS\_RTE

1. Specification of OS(4.2.2)

AUTOSAR\_SWS\_OS

[7] Specification of Module E2E Transformer(4.2.2)

AUTOSAR\_SWS\_E2ETransformer

[8] Specification of SW-C End-to-End Communication Protection Library(4.2.2/R19-11)

AUTOSAR\_SWS\_E2ELibrary

1. Specification of COM Based Transformer(4.2.2)

AUTOSAR\_SWS\_COMBasedTransformer

1. Specification of ECU State Manager(4.2.2)

AUTOSAR\_SWS\_ECUStateManager

[11] ASAM\_MCD-2MC\_DataSpecifcation

ASAM\_MCD-2MC\_DataSpecifcation\_V1.6.pdf.

## Abbreviations & Terms

### Abbreviations

| Abbreviations | Description |
| --- | --- |
| IRV | Inter Runnable Variables |
| SWC | Software Component |
| RTE | Runtime Environment |
| E2E | End to End Protection |
| COM | Communication |
| ECUM | ECU State Manager |
| S-R | Sender Receiver |
| C-S | Client Server |
| API | Application Interface |
| Os | Operating System |
| E2EXF | End to End Protection Transformer |
| CRC | Cyclic Redundancy Check |
| COMXF | Com based Transformer |
| ASAM | Association for Standardization of Automation and Measuring Systems |

### Terms

| Terms | Description |
| --- | --- |
| application mode manager | An application mode manager is an AUTOSAR softwarecomponent that provides the service of switching modes. The modes of an application mode manager do not have to be standardized. |
| client | A client is defined as one ClientServerOperation in one  RPortPrototype of one Software Component instance. For the definition of the client neither the number of ServerCallPoints nor RunnableEntity accesses to the ServerCallPoint are relevant. A Software Component instance can appear as several clients to the same server if it defines ServerCallPoints for several PortPrototypes of the same PortInterface’s ClientServerOperation. |
| data semantic | When data is distributed, the last received value is of interest  (last-is-best semantics). Therefore the software implementation  policy, stated in the swImplPolicy attribute of the SwDataDefProps, shouldn’t be ’queued’. |
| event semantic | When events are distributed the whole history of received events  is of interest, hence they must be queued on receiver side. Therefore the software implementation policy, stated in the swImplPolicy attribute of the SwDataDefProps, will have the value  ’queued’(corresponding to event distribution with a queue). |
| inter-ECU communication | The communication between ECUs, typically using COM is called  inter-ECU communication in this document. |
| inter-partition communication | The communication within one ECU but between different partitions, represented by different OS applications, is called interpartition communication in this document. It typically involves the use of OS mechanisms like IOC or trusted function calls. The partitions can be located on different cores or use different memory sections of the ECU. |
| intra-ECU communication | The communication within one ECU is called intra-ECU communication in this document. It is a super set of interpartition communication and intra-partition communication. |
| intra-partition communication | The communication within one partition of one ECU is called  intra-partition communication. In this case, RTE can make use of internal buffers and queues for communication. |
| mode disabling | When a ‘mode disabling’ is active, RTE disables the start of mode disabling dependent ExecutableEntitys. The ‘mode disabling’ is active during the mode that is referenced in the mode disabling dependency and during the transitions that enter and leave this mode. |
| mode disabling dependency | A RTEEvent that starts a RunnableEntity can contain a disabledMode association which references a ModeDeclaration. This association is called mode disabling dependency in this document. |
| mode machine instance | The instances of mode machines or ModeDeclarationGroups are  defined by the ModeDeclarationGroupPrototypes of the mode  managers. Since a mode switch is not executed instantaneously, The RTE has to maintain it’s own states. For each mode manager’s ModeDeclarationGroupPrototype, RTE has one state machine. This state machine is called mode machine instance. For all mode users  of the same mode manager’s ModeDeclarationGroupPrototype. |
| mode manager | Entering and leaving modes is initiated by a mode manager. A  mode manager is a software component that provides a  p-port typed by a ModeSwitchInterface ModeDeclarationGroupPrototype in the role providedModeGroup. |
| ModeSwitchAck ExecutableEntity | A RunnableEntity that is triggered by a ModeSwitchedAckEvent connected to the mode manager’s ModeDeclarationGroupPrototype. It is called ModeSwitchAck ExecutableEntity. |
| mode switch notification | The communication of a mode switch from the mode manager  to the mode user using either the ModeSwitchInterface  or providedModeGroup and requiredModeGroup ModeDeclarationGroupPrototypes is called mode switch notification. |
| mode switch port | The port for receiving (or sending) a mode switch notification. For  this purpose, a mode switch port is typed by a ModeSwitchInterface. |
| mode user | An AUTOSAR SW-C Module that depends on modes is called a mode user. The dependency can occur through a SwcModeSwitchEvent, a ModeAccessPoint for a provided/required mode switch port, or a accessedModeGroup for a  providedModeGroup/requiredModeGroup ModeDeclarationGroupPrototype. |
| runnable activation | The activation of a runnable is linked to the RTEEvent that leads to the execution of the runnable. It is defined as the incident that is referred to by the RTEEvent.  E. g., for a timing event, the corresponding runnable is activated,  when the timer expires, and for a data received event, the runnable is activated when the data is received by the RTE. |
| runnable start | A runnable is started by the calling the C-function that implements the runnable from within a started task. |
| server | A server is defined as one RunnableEntity which is the target of an OperationInvokedEvent. Call serialization is on activation of RunnableEntity. |
| server runnable | A server that is triggered by an OperationInvokedEvent. It has a mixed behavior between a runnable and a function call. In certain situations, RTE can implement the client server communication as a simple function call. |
| trigger sink | A trigger sink relies on the activation of Runnable Entities if a particular Trigger is raised. A trigger sink has a dedicated require trigger port(s) or /and requiredTrigger Trigger(s) to communicate to the trigger source(s). |
| trigger source | A trigger source administrate the particular Trigger and informs the RTE if the Trigger is raised. A trigger source has dedicated provide trigger port(s) or /and releasedTrigger Trigger(s) to communicate to the trigger sink(s). |
| triggered runnable | A Runnable Entity that is triggered at least by one ExternalTriggerOccurredEvent or InternalTriggerOccurredEvent. In particular cases, the Trigger Event Communication or the Inter Runnable Triggering is implemented by RTE as a direct function call of the triggered runnable by the triggering runnable. |

# Operational environment & Constrains

Environment refer to ‘PH-PRD-QC-002-2022\_RiskAnalysis.xlsx’, sheet: Pre-define, Assumption of environment.

# Software Architecture Overview

## File Structure

[SA\_R2\_001]:

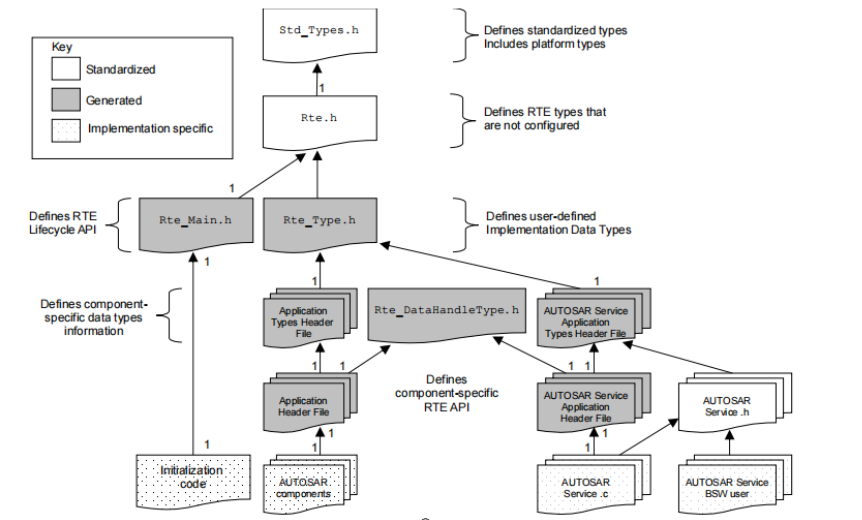


Figure 3‑1 RTE file contains structure

Figure 3-1 illustrates the structure of the RTE source and header files. It includes RTE standard headers and RTE-generated headers, and can see the relationship between RTE headers, application headers, lifecycle headers, and AUTOSAR software components in the diagram.

## Software Architecture

[SA\_R2\_002]:

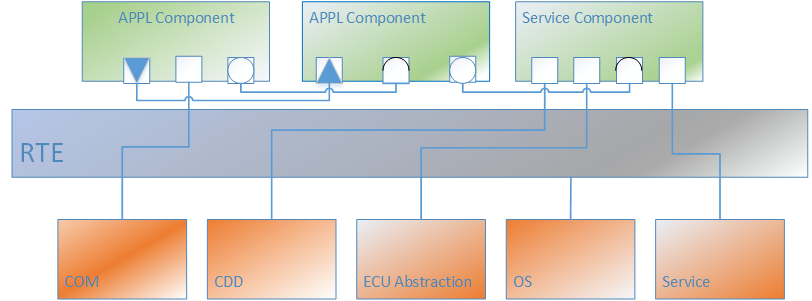


Figure 3‑2 RTE Software Architecture

RTE软件架构如上图所示。从图上可以看到RTE作为隔绝上层应用和底层软件的中间模块，起到了承上启下的作用。COM模块负责收发外部信号，一方面是将从PDUR收到的信号通过RTE函数封装后传递给上层应用模块（APPL Component）;一方面是将上层应用模块的信号通过RTE函数封装后传递给COM。IO模块将IO函数接口封装后经过RTE供上层应用模块调用,其中包含了DIO/ADC/PWM。OS为RTE的任务调度提供支持，同时RTE根据OS的不同分区实现IOC核间通讯功能。

Service模块将底层模块进行封装供上层进行调用，其中包含的底层模块有：DCM, DEM, NVM, FIM,CSM, KEYM, BSWM, ECUM, WDGM, COMM, STBM。

## Function Description

The Run-Time Environment (RTE) is at the heart of the AUTOSAR ECU architecture. The RTE is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB). The RTE provides the infrastructure services that enable communication to occur between AUTOSAR software-components as well as acting as the means by which AUTOSAR software-components access basic software modules including the OS and communication service.

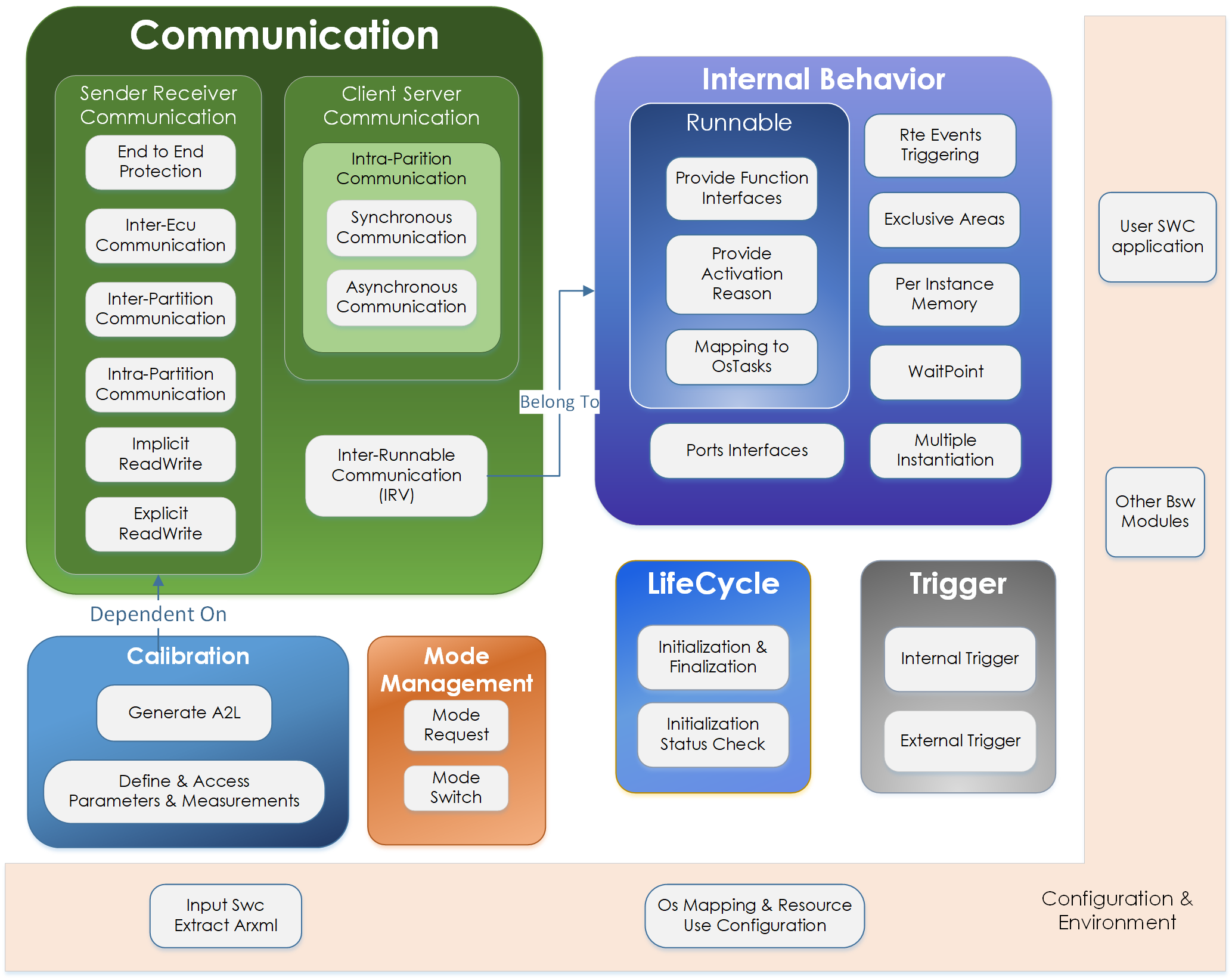


Figure 3‑3 Rte Functionalities

The RTE encompasses both the variable elements of the system infrastructure that arise from the different mappings of components to ECUs as well as standardized RTE services. In principle, the RTE can be logically divided into several sub-parts realizing:

• The communication between software components: Sender-Receiver communication either between ECUs or inside an ECU that allows data transmitting and receiving, Client-Server communication inside an ECU that allows function calling, and inter-runnable communication using inter-runnable variables which is the data exchange between runnables inside a software component.

• The internal behavior of a software component: It is the core of RTE, which manages runnable functionalities, RTE events which activate runnables, exclusive area, activation reason, task mapping, per instance memory, multiple instantiation, port interfaces and waitpoints.

• Calibration: the process of adjusting an ECU SW to fulfill its tasks to control physical processes respectively to fit it to special project needs or environments

• Mode Management: It allows users to define modes and do mode request which switches modes to notify their users, runnables can be triggered by modes.

• RTE Life Cycle Management: Some of the RTE modules require initialization of global variables that are necessary to fulfill the functionalities, thus RTE is allowed to function normally only after proper initialization. Before termination, deinitilization shall be done and RTE shall stop operating after that.

## Data Structure Define

|  |  |
| --- | --- |
| Name： | Rte\_Inst\_<cts> |
| Type： | Structure |
| Range： | Custom implementation |
| Describe： | where <cts> is the component type symbol of the AtomicSwComponentType. The Rte\_Instance data type defines the handle used to access instance specific information from the component data structure. |

|  |  |
| --- | --- |
| Name： | Rte\_[Byps]\_CDS\_ |
| Type： | Structure |
| Range： | /\* Data Handle Section\*/  /\* Per-instance Memory Handles Section\*/ |
| Describe： | Rte\_[Byps]\_CDS\_ where <CDS> is the component type symbol of the AtomicSwComponentType. [Byps] is an optionnal infix used when component wrapper method for bypass support is enabled for the related software component type  Note: Data Handle Section shall be a pointer to a data element with or without status and extended status. Per-instance Memory Handles Section The data type of each Per-instance Memory Handle shall be a pointer to the type of the per instance memory that is defined in the Application Header file |

## Data Types Define

|  |  |
| --- | --- |
| Name： | Boolean |
| Type： | boolean |
| Range： | TRUE/FALSE |
| Describe： | Boolean basic implementation type |

|  |  |
| --- | --- |
| Name： | Float |
| Type： | float32 |
| Range： | -- |
| Describe： | Float basic implementation type |

|  |  |
| --- | --- |
| Name： | Float32 |
| Type： | float32 |
| Range： | -- |
| Describe： | Float32 basic implementation type |

|  |  |
| --- | --- |
| Name： | Float64 |
| Type： | float64 |
| Range： | -- |
| Describe： | Float64 basic implementation type |

|  |  |
| --- | --- |
| Name： | SInt16 |
| Type： | sint16 |
| Range： | -32768- 32767 |
| Describe： | SInt16 basic implementation type |

|  |  |
| --- | --- |
| Name： | SInt32 |
| Type： | sint32 |
| Range： | -2147483648- 2147483647 |
| Describe： | SInt32 basic implementation type |

|  |  |
| --- | --- |
| Name： | SInt8 |
| Type： | sint8 |
| Range： | -128-127 |
| Describe： | SInt16 basic implementation type |

|  |  |
| --- | --- |
| Name： | UInt8 |
| Type： | uint8 |
| Range： | 0-255 |
| Describe： | UInt8 basic implementation type |

|  |  |
| --- | --- |
| Name： | UInt16 |
| Type： | uint16 |
| Range： | 0-65535 |
| Describe： | UInt16 basic implementation type |

|  |  |
| --- | --- |
| Name： | UInt32 |
| Type： | uint32 |
| Range： | 0-4294967295 |
| Describe： | UInt32 basic implementation type |

|  |  |
| --- | --- |
| Name： | UInt64 |
| Type： | uint64 |
| Range： | 0-18446744073709551615 |
| Describe： | UInt64 basic implementation type |

|  |  |
| --- | --- |
| Name： | Std\_ReturnType |
| Type： | uint8 |
| Range： | 0-255 |
| Describe： | The Std\_ReturnType defines the "‘status"’ and "‘error values"’ returned by API functions |

|  |  |
| --- | --- |
| Name： | Application Error（No fixed format） |
| Type： | uint8 |
| Range： | 1-63 |
| Describe： | In client server communication, the server may return any value within the application error range |

|  |  |
| --- | --- |
| Name： | Predefined Error Code |
| Type： | Enumeration uint8 |
| Range： | RTE\_E\_OK 0  RTE\_E\_INVALID 1  RTE\_E\_COM\_STOPPED 128  RTE\_E\_TIMEOUT 129  RTE\_E\_LIMIT 130  RTE\_E\_NO\_DATA 131  RTE\_E\_TRANSMIT\_ACK 132  RTE\_E\_NEVER\_RECEIVED 133  RTE\_E\_UNCONNECTED 134  RTE\_E\_IN\_EXCLUSIVE\_AREA 135  RTE\_E\_SEG\_FAULT 136  RTE\_E\_OUT\_OF\_RANGE 137  RTE\_E\_SERIALIZATION\_ ERROR 138  RTE\_E\_HARD\_TRANSFORMER\_ ERROR 138  RTE\_E\_SERIALIZATION\_ LIMIT 139  RTE\_E\_TRANSFORMER\_ LIMIT 139  RTE\_E\_SOFT\_TRANSFORMER\_ ERROR 140  RTE\_E\_COM\_BUSY 140  RTE\_E\_LOST\_DATA 64  RTE\_E\_MAX\_AGE\_EXCEEDED 64 |
| Describe： | Define standard error and status |

|  |  |
| --- | --- |
| Name： | Rte\_TransformerError |
| Type： | uint8 |
| Range： | 0-255 |
| Describe： | The Rte\_TransformerErrorCode represents a transformer error in the context of a certain transformer chain. |

|  |  |
| --- | --- |
| Name： | Rte\_ModeType\_<prefix>\_<ModeDeclarationgroup> |
| Type： | Implementation Data Type |
| Range： | Custom implementation |
| Describe： | An Rte\_ModeType is used to hold the identifiers for the ModeDeclarations of a ModeDeclarationGroup. |

|  |  |
| --- | --- |
| Name： | Enumeration Data Types |
| Type： | Implementation Data Type |
| Range： | Custom implementation |
| Describe： | Defined by the customer for use in SWC compute methods |

# Dynamic Behavior Design

## State Mechanism

[SA\_R2\_003]:

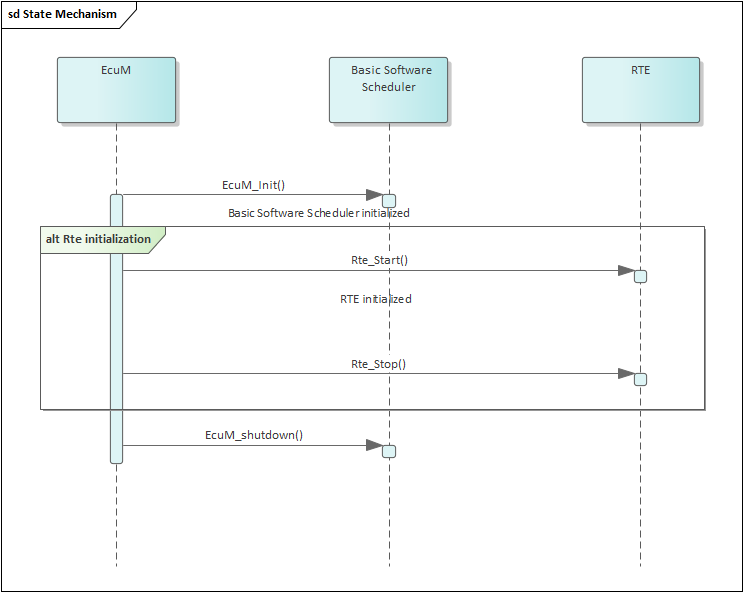
**

Figure 4‑1 Nested life cycle of RTE and Basic Software Scheduler

当ECU上电开始初始化，ECUM调用ECUM\_Init()函数执行Basic Software Scheduler Initialized。基础软件模块初始化完毕后由ECUM调用Rte\_Start()函数执行RTE初始化，完成对RTE的初始化任务。初始化后运行实体才可以被激活，RTE接口才可以被调用。下电时RTE先调用Rte\_Stop()函数释放RTE分配的所有系统和通信资源，而后ECUM调用ECM\_shutdown()执行Basic Software Scheduler Initialized的下电任务。

# RTE

## Function Description

[SA\_R2\_353]:

RTE作为上层和底层的衔接，总体实现了下述功能：

* RTE实现ECU内SWC通讯以及ECU间总线信号与SWC的通讯传递。其通讯形式包含S-R通讯（隐式与显式），C-S通讯（异步调用与同步调用）。在连接这些通讯时RTE会返回通讯状态告知上层软件。
* RTE实现对Runnable的管理功能，将Runnable映射到OS TASK上。通过OS的调度实现RTE任务的执行。
* RTE支持多核间以及跨分区通讯
* RTE encapsulates basic software services as a complete software component based on Autosar requirement so that upper layer application could use the Autosar standard interface to utilize basic software. The fowlloing basic software modules support encapsulation: DCM, DEM, NVM, FIM, CSM, KEYM, BSWM, ECUM, WDGM, COMM, STBM.
* RTE实现标定功能并为测量量和观测量自动生成对应的A2L文件
* RTE支持模式管理功能，实现应用模块与BSWM、ECUM等模块的交互

[SA\_R2\_037]:

The RTE generator shall support a mode in which the generated code is independent of the micro controller. Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02734

## S-R Communciation

### Function Description

Sender-receiver communication involves the transmission and reception of signals consisting of atomic data elements that are sent by one component and received by one or more components. A sender-receiver interface can contain multiple data elements. Sender-receiver communication is one-way - any reply sent by the receiver is sent as a separate sender-receiver communication

RTE 支持显示和隐式通信

[SA\_R2\_004]: The RTE shall support 'explicit' and 'implicit' data reception and transmission. refers to 'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_06011]

显示通信就是runnable直接读写全局变量

[SA\_R2\_005]: Explicit communication, as the name implies, means direct communication, and the code generates a global variable that Runnable can read and write directly；refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06011]

隐士通信将全局变量复制到runnable的局部变量中，操作局部变量

[SA\_R2\_006]: Implicit communication, also known as cache calls, involves copying the global variable into a Runnable local variable, manipulating the local variable, and finally copying the local variable into the global variable. When Runnable operates on local variables, the value of the global variable does not change.refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06011]

SR支持1:n,n:1通信

[SA\_R2\_007]: Sender-receiver communication is not restricted to communication connections between a single sender and a single receiver. Instead, sender receiver communication connection can have multiple senders (’n:1’ communication) or multiple receivers (’1:m’ communication).

RTE支持初始化

[SA\_R2\_008]: For each data element in an interface specified with data semantics, the RTE shall support the initValue attribute.refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06009]

当数据原型具有复合数据类型时，RTE必须对数据进行编排(不使用数据转换器)

[SA\_R2\_009]: When a data prototype has a composite data type the RTE must marshall the data(Without Data Transformer).Refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02557]

RTE创建接收队列为每个dataelement swImplPolicy = queued

[SA\_R2\_010]: The RTE shall implement a receive queue for each event-like data element (swImplPolicy = queued) of a receive port. Refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02521]

RTE 应禁用对分区本地通信的超时监视，与是否存在 aliveTimeout 无关。

[SA\_R2\_011]: The RTE shall have time-out monitoring disabled for communications local to the partition, independently of the presence of aliveTimeout.Refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_05021]

数据元素无效

[SA\_R2\_012]: The communication infrastructure shall provide means to set a data element to invalid and also indicate an invalid data element to the receiving software components. This functionality is called “data element invalidation” refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_05024]

接收数据的范围检查

[SA\_R2\_013]: The RTE shall implement a range check in the receiving path of a particular component if the handleOutOfRange is defined at the ReceiverComSpec and has any value other than none. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08028]

RTE应支持在非队列的接收端支持基于值得过滤

[SA\_R2\_014]: The RTE shall provide value-based filters on the receiver side of unqueued S/R-Communication as specified in the SW-C template.refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_05503]

DataSendCompletedEvent 发送确认事件

[SA\_R2\_015]: As a general rule, the acknowledgment events DataWriteCompletedEvent and DataSendCompletedEvent shall be raised immediately after the sending to all receivers has been performed and in case of Inter-ECU communication all acknowledgments from COM have been received.refers to 'AUTOSAR\_SWS\_RTE'

[SWS\_Rte\_08017,SWS\_Rte\_08018,SWS\_Rte\_08020,SWS\_Rte\_08021]

RTE应支持S-R通信中的所有类型的数据元素，包括原始数据、结构体和数组

[SA\_R2\_016]: RTE shall have support all types of data element including primitive, struct, and array in S-R Communication.

RTE支持ECU间的数据保护

[SA\_R2\_017]: RTE should support data protection for SR communication between ECUs

RTE应该能够支持S-R通信的无效值替换

[SA\_R2\_018]: RTE should be able to support invalid value replacement for S-R communication

RTE应能支持ecu之间产生S-R通信超时Callout功能

[SA\_R2\_019]: RTE should be able to support the generation of S-R communication timeout Callout function between ECUs

RTE应能支持S-R通信的数据一致性保护

[SA\_R2\_020]: RTE should be able to support data consistency protection of S-R communication

RTE支持队列长度检查

[SA\_R2\_021]: RTE should support queue length detection of S-R communication

如果SR通信接口没有收到数据,RTE API应该返回RTE\_NEVER\_RECEIVED

[SA\_R2\_022]: If the SR communication interface never receives data, the RTE API returns the status of RTE\_NEVER\_RECEIVED.

[SA\_R2\_350]: RTE shall provide User Manual that user ensure array wouldn’t exceed the limit

### Control Flow

[SA\_R2\_023]:

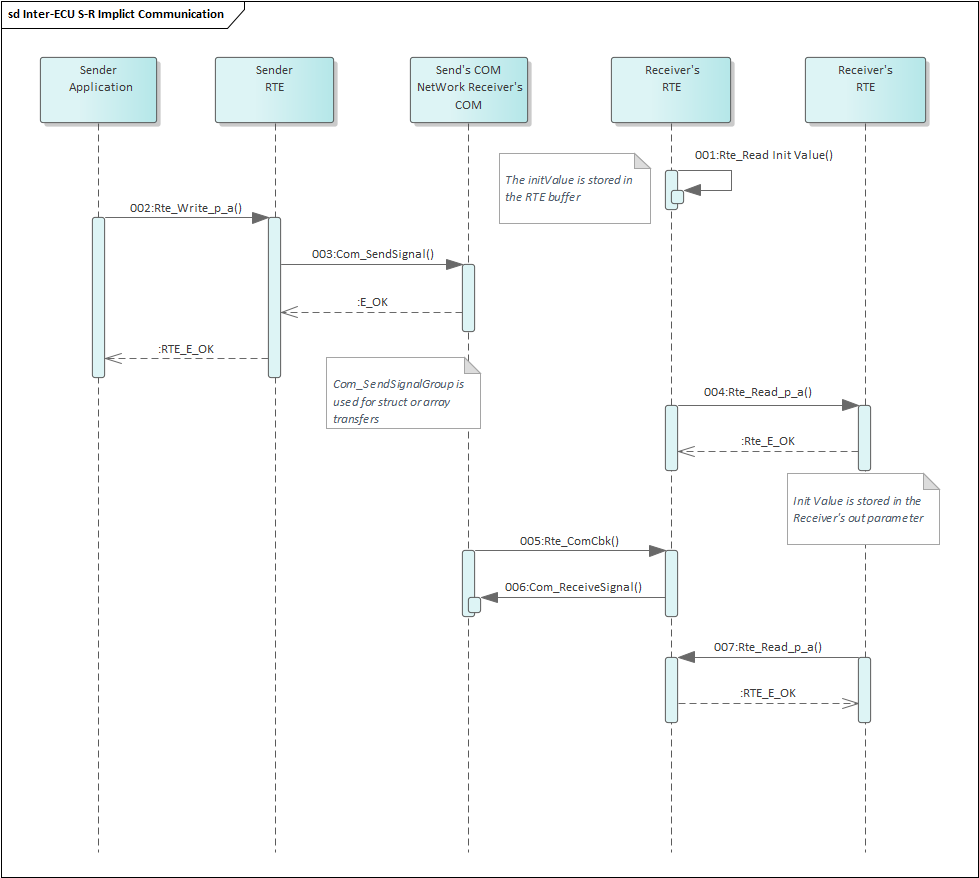


Figure 5‑1 ECU间SR显式通讯示意图

图式为ECU间SR显式通讯示意图。当Sender Application发送一个信号时调用Rte\_Write\_p\_a()函数，而后RTE调用Com\_SendSignal()将信号发出到总线。接收到端的总线COM收到信号后会调用Rte\_ComCbk()函数，接收端的RTE调用Com\_ReceiveSignal()将信号值保存在全局变量中。在接收端未收到更新的值前，接收端的应用始终以初始值呈现。接收端收到外部通讯值更新后，接收端再次调用Rte\_Read\_p\_a()函数即可得到储存在全局变量中的信号值。值得注意的是如果收发的是结构体或者是数组的则需要调用Com\_SendSignalGroup发送以及调用Com\_ReceiveSignalGroup接收；

[SA\_R2\_024]:

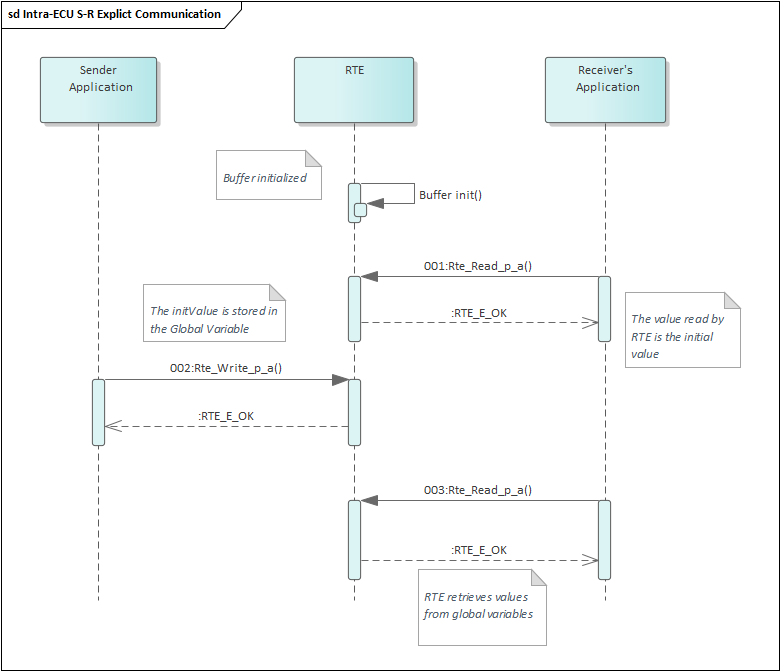


Figure 5‑2 ECU内部SR显式通讯序列图

图示为ECU内部SR通讯。当SWC Sender发送信号时调用Rte\_Write\_p\_a()，将信号值赋值到全局变量中，RTE回复给发送端传输状态。当全局变量中的值未被更新过时，接收端SWC读取到的值为初始值。当全局变量中的值被更新过后，接收端SWC再次调用Rte\_Read\_p\_a()获取全局变量中的值。值得注意的是如果接收端配了滤波，则接收时可根据滤波算法过滤。如果信号被滤，则接收到得值为上一时刻值。

[SA\_R2\_025]:

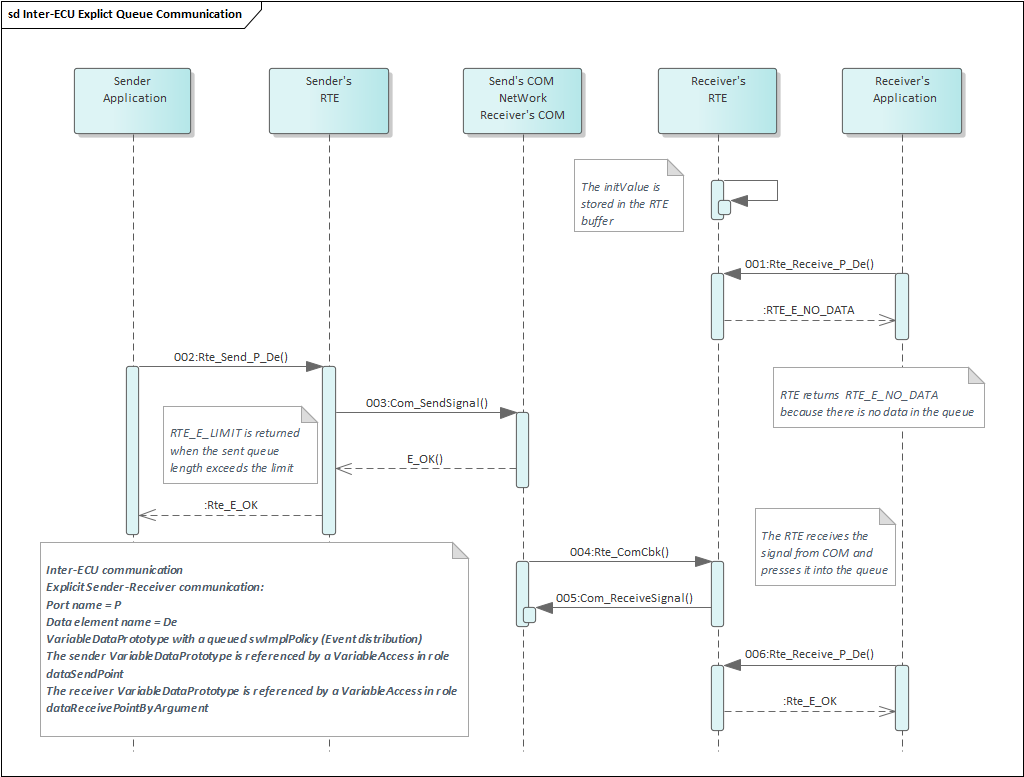


Figure 5‑3 ECU间显式队列通讯序列图

图为ECU间显式队列通讯序列图。与非列队通讯不同的是：VariableDataPrototype's swImplPolicy should be set to queued.当发送端发送信号时调用Rte\_Send\_P\_De()，将信号发送给发送端的RTE。Rte\_Send\_P\_De返回状态给应用端。若发送的信号长度已经超过了队列长度，则RTE返回RTE\_E\_LIMIT。RTE调用Com\_SendSignal()将信号发送到总线；接收端收到信号后调用Rte\_ComCbk()函数通知RTE，RTE调用Com\_ReceiveSignal()接收信号，将信号压入队列中。值得注意的是如果压入的信号过多。则接受的时候会返回RTE\_E\_LOST\_DATA。接收端应用端第一次调用Rte\_Receive\_P\_De()函数时因为队列中没有数据，因此读到的是初始值，同时RTE返回RTE\_E\_NO\_DATA;当队列中有数据时Rte\_Receive\_P\_De()将会从队列中读取数据并返回Rte\_E\_OK

[SA\_R2\_026]:

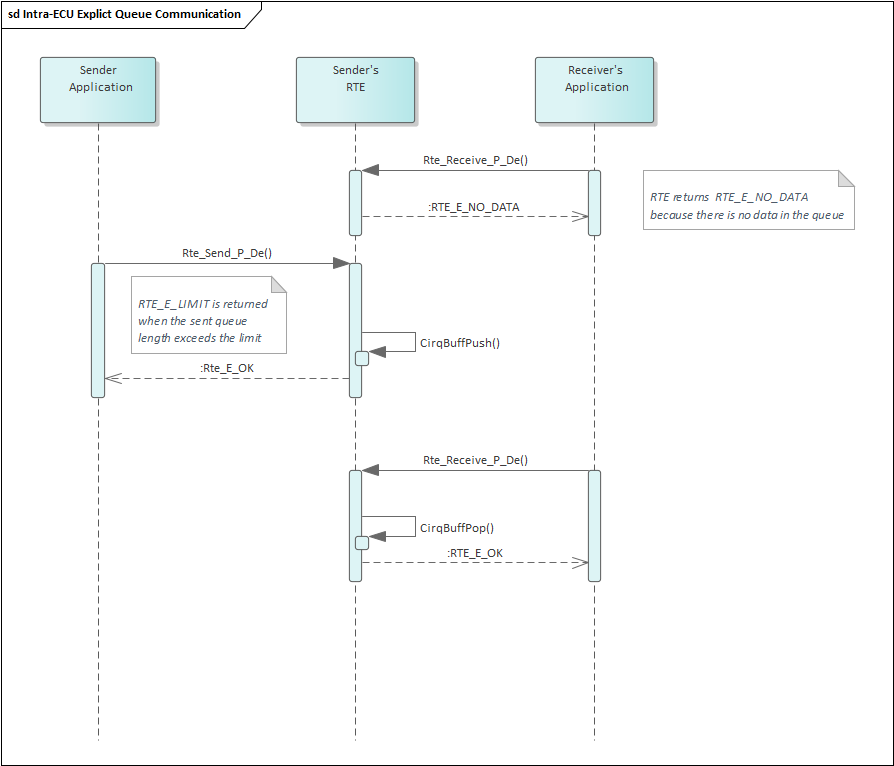


Figure 5‑4 ECU内显式队列通讯序列图

图为ECU内显式队列通讯序列图，当队列中还没有数据时由接收端调用Rte\_receive函数，函数返回RTE\_E\_NO\_DATA说明此时队列中没有数据。当有数据需要发送到队列中时，发送段调用Rte\_Send函数，函数中执行CirqBuffPush将数据压入队列中。如果压入的数据超过队列长度，则发送端将会返回RTE\_E\_LIMIT。当队列中有数据时，接收端调用Rte\_receive函数通过CirqBuffPop将数据从队列中取出。

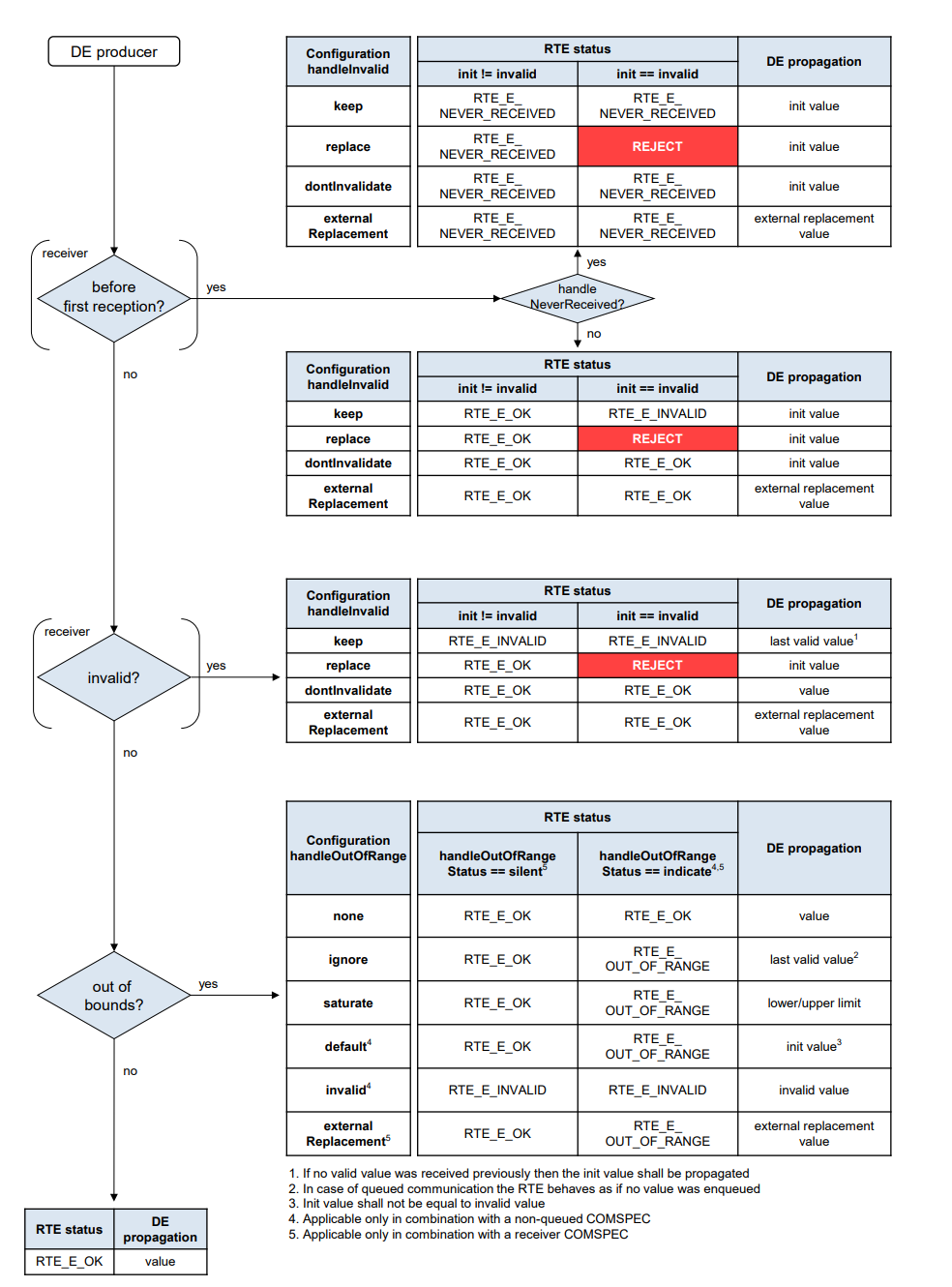


Figure 5‑5范围检查和有效性检查流程、返回值以及返回状态图

图为信号的范围检查和有效性检查流程图。

如果配置了handleNeverReceived，在第一次收到信号前如果初始值是有效值则RTE函数Rte\_Read或者Rte\_IRead函数回RTE\_NeverReceived（HandInvalid配置的是replace，初始值不允许配置成无效值），Rte\_Read或者Rte\_IRead函数对应收到的值为初始值（HandInvalid配置的是externalReplacement除外）。如果初始值是无效值，同上。

如果未配置handleNeverReceived，在第一次收到信号前如果初始值是有效值则RTE函数Rte\_Read或者Rte\_IRead函数回RTE\_E\_OK，Rte\_Read或者Rte\_IRead函数对应收到的值为初始值（HandInvalid配置的是externalReplacement除外）。如初始值是无效值，HandInvalid配置的是keep则返回RTE\_E\_INVALID其余返回RTE\_E\_OK（HandInvalid配置的是externalReplacement除外）

当传输的信号时如果信号是无效值且HandleInvalidConfig配置为Keep,则Rte\_Read或者Rte\_IRead函数回RTE\_E\_INVALID，读取的信号值为last valid value；当传输信号时如果信号是无效值且HandleInvalidConfig配置为replace,则Rte\_Read或者Rte\_IRead函数回RTE\_E\_OK，读取的信号值为init value（HandInvalid配置的是replace，初始值不允许配置成无效值）;当传输信号时如果信号是无效值且HandleInvalidConfig配置为dontInvalidate,则Rte\_Read或者Rte\_IRead函数回RTE\_E\_OK，读取的信号值为读到的值；当传输信号时如果信号是无效值且HandleInvalidConfig配置为externalReplacement,则Rte\_Read或者Rte\_IRead函数回RTE\_E\_OK，读取的信号值为external Replacement value

如果传输的信号是有效的则需要检查是否超过边界：如果信号超过边界且handleOutOfRangeConfig配置的是none,RTE返回状态为RTE\_E\_OK,传输的信号值即为收到的信号；如果信号超过边界且handleOutOfRangeConfig配置的是ignore,RTE返回(HandleOutOfRangeStatus == silent 为RTE\_E\_OK;HandleOutOfRangeStatus == indicate 为RTE\_E\_OUT\_OF\_RANGE),传输值为last valid value;如果信号超过边界且handleOutOfRangeConfig配置的saturate,RTE返回(HandleOutOfRangeStatus == silent 为RTE\_E\_OK;HandleOutOfRangeStatus == indicate 为RTE\_E\_OUT\_OF\_RANGE),传输值为low/upper value;如果信号超过边界且handleOutOfRangeConfig配置的default,RTE返回(HandleOutOfRangeStatus == silent 为RTE\_E\_OK;HandleOutOfRangeStatus == indicate 为RTE\_E\_OUT\_OF\_RANGE),传输值为init value;如果信号超过边界且handleOutOfRangeConfig配置的invalid,RTE返回(HandleOutOfRangeStatus == silent 为RTE\_E\_OK;HandleOutOfRangeStatus == indicate 为RTE\_E\_INVALID),传输值为invalid value;如果信号超过边界且handleOutOfRangeConfig配置的externalReplacement,RTE返回(HandleOutOfRangeStatus == silent 为RTE\_E\_OK;HandleOutOfRangeStatus == indicate 为RTE\_E\_OUT\_OF\_RANGE),传输值为external Replacement value

如果收到的信号是有效的且没有out of range，则Rte\_Read或者Rte\_IRead函数返回RTE\_E\_OK

值得注意的是，如果HandInvalid配置的是externalReplacement,则当收到的信号无论是无效值或是未收到信号或是超出边界值的情况，接收数据都将被替换为设置好的外部值。

[SA\_R2\_027]:

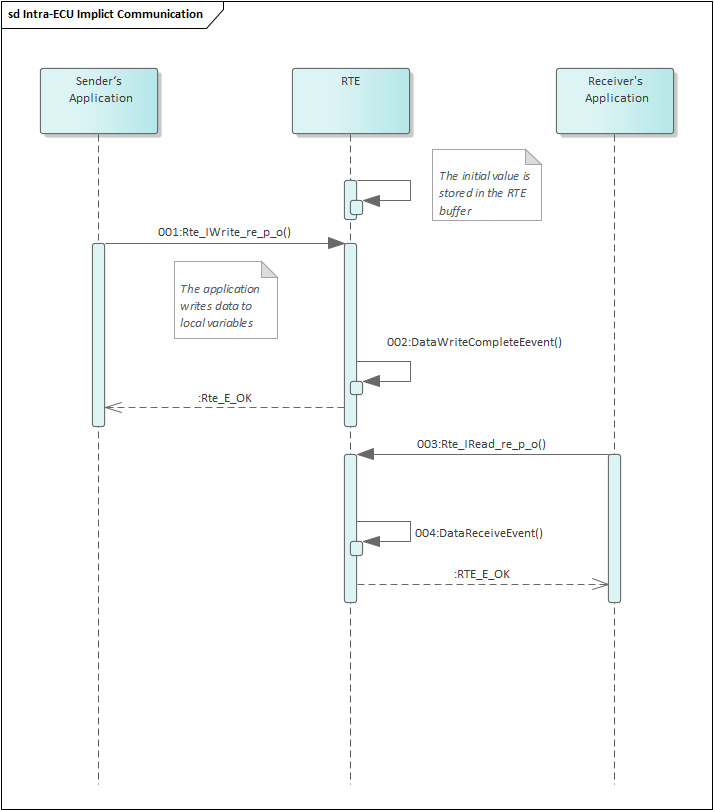


Figure 5‑6 Implicit SR Intra-ECU Commuciation with DataWriteCompletedEvent&DataReceiveEvent

在ECU内部通讯中如果配置了隐式收发以及DataWriteCompletedEvent&DataReceiveEvent，那么在完成信号的读写任务后就将激活Event。具体流程如下：当Send Application需要发送信号时调用Rte\_IWrite\_re\_p\_o()函数，将数据写入RTE的局部变量中。当Runnable的RTE任务结束前，RTE会将局部变量中的数据写入到全局变量中供数据接收方读取。发送方写完数据后将激活对应的DataWriteTASK。接收端Runnbale RTE在运行开始时将全局变量中的数据读到局部变量中，Receiver's Application调用Rte\_IRead\_re\_p\_o()将局部变量中的值读取，在接收到对应的信号后将触发DataReceiveEvent。

[SA\_R2\_028]:

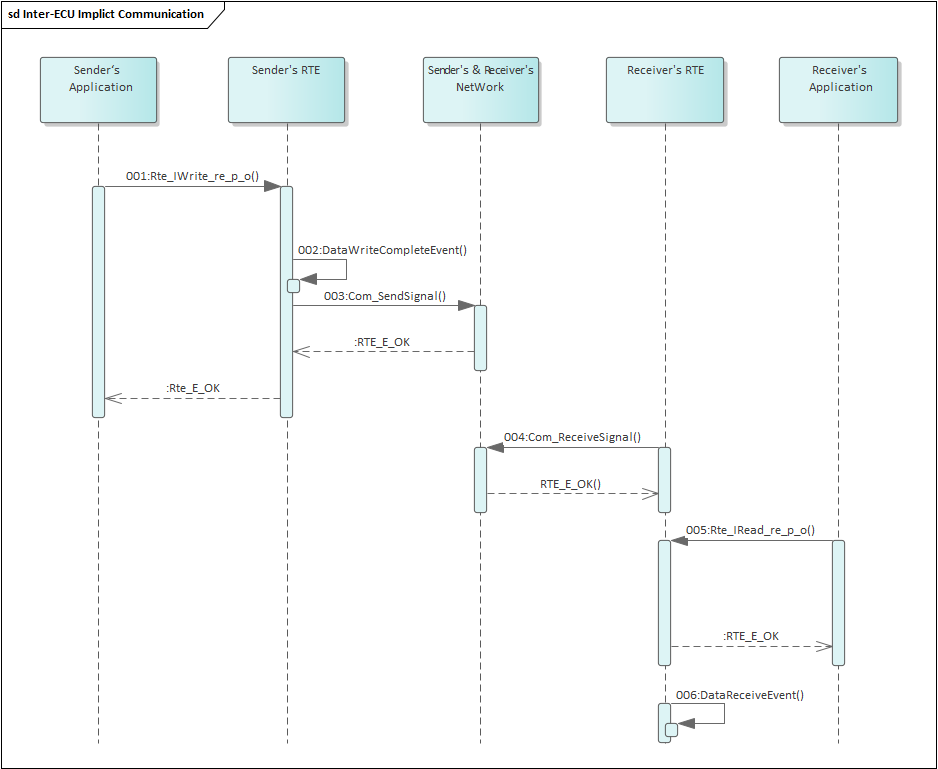


Figure 5‑7 Implicit SR Inter-ECU Commuciation with DataWriteCompletedEvent&DataReceiveEvent

在ECU间通讯中如果配置了隐式收发以及DataWriteCompletedEvent&DataReceiveEvent，那么在完成信号的读写任务后就将激活Event。具体流程如下：当应用发送端需要发送信号时调用Rte\_IWrite\_re\_p\_0()函数，将信号值写入对应的BUFF中后触发DataWriteCompleteEvent对应的Runnable。发送端RTE任务调用Com\_SendSignal()函数将信号发送到总线上。接收端在收到信号后在RTE\_ComCbk中调用Com\_ReceiveSignal()函数将值写入对应的RTE Buffer中等待上层调用。上层通过调用Rte\_IRead\_re\_p\_o()函数将buff中的值传递给上层

[SA\_R2\_029]:

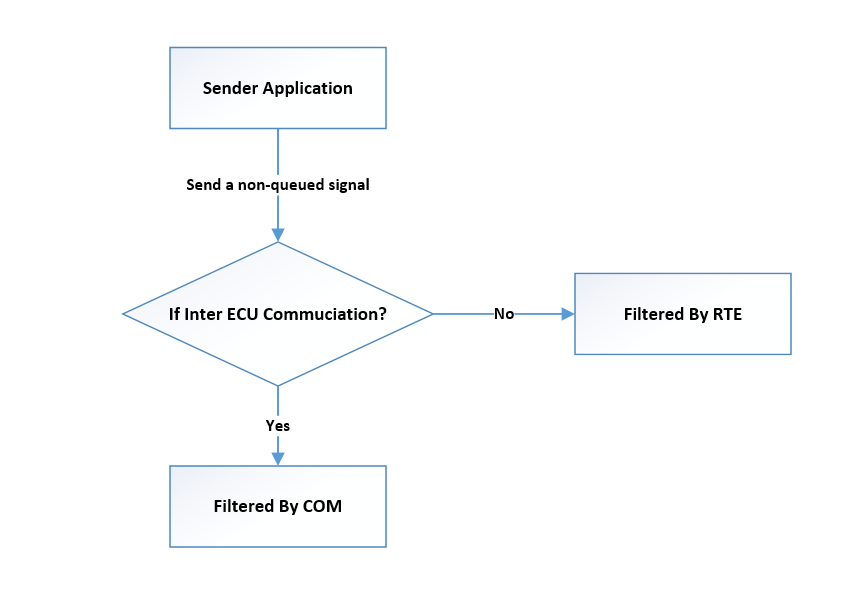


Figure 5‑8 Filter Processing principle

The picture shows the principle of Filter processing for non-queue received signals.The RTE shall provide value-based filters on the receiver side of unqueued S/R-Communication.For inter-ECU communication, the filter implementation is performed/done by the COM module. For intra-ECU and inter-Partition communication, the RTE shall perform the filtering itself. refers to 'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_05503, SWS\_Rte\_05500]

[SA\_R2\_030]:

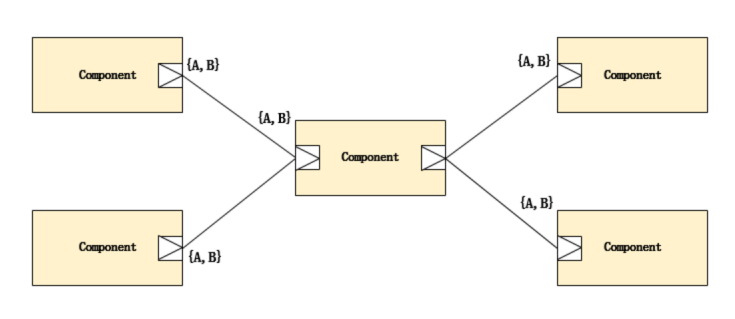


Figure 5‑9 SR通讯1:N 和 N:1通讯

图为SR通讯中一对多和多对一示意图。RTE支持多对一以及一对多的通讯功能。

### Data Flow

#### Implicit Communication

[SA\_R2\_031]：

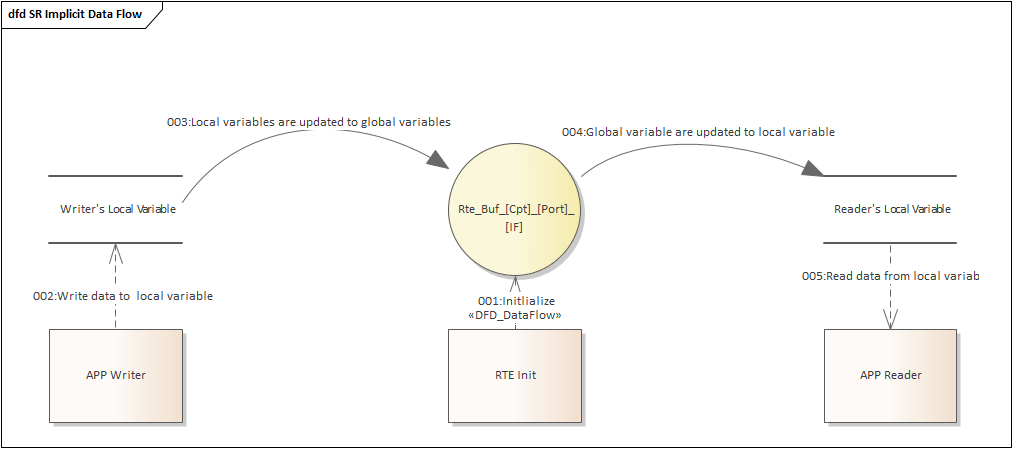


Figure 5‑10 Implicit Communication Data Flow

1. Write data to local variable. -1

2. Local variable are updated to global variables.-2

3. Global variable are updated to local variable. -3

4. Read data from local variable.-4

#### Explicit Communication

[SA\_R2\_032]：

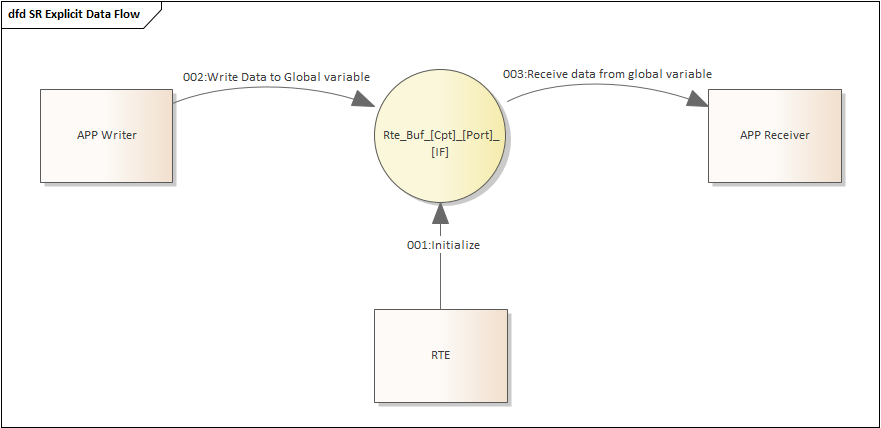


Figure 5‑11 Explicit Communication Data Flow

1. Write data to local variable. -1

2. Local variable are updated to global variables.-2

3. Global variable are updated to local variable. -3

### External Interface Definition

#### Rte\_Write

[SA\_R2\_033]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Write | | |
| Function prototype: | Std\_ReturnType Rte\_Write\_<p>\_<o> (  [IN Rte\_Instance <instance>],  IN <data>,  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrant | | |
| Input parameters: | Instance: instance of cds  data: pass the data element | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | Rte\_TransformerError transformerError : transformer error | | |
| Return value: | Std\_ReturnType：  RTE\_E\_OK、RTE\_E\_COM\_STOPPED、 RTE\_E\_SEG\_FAULT、 RTE\_E\_COM\_BUSY、RTE\_E\_HARD\_TRANSFORMER\_ERROR、RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Initiate an “explicit” sender-receiver transmission of data elements  with “data” semantic (swImplPolicy different from queued). | | |
| Precautions | None | | |

#### Rte\_Send

[SA\_R2\_317]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Send | | |
| Function prototype: | Std\_ReturnType Rte\_Send\_<p>\_<o> (  [IN Rte\_Instance <instance>],  IN <data>,  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds  data: pass the data element | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、RTE\_E\_COM\_STOPPED、RTE\_E\_LIMIT、  RTE\_E\_SEG\_FAULT、RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Initiate an “explicit” sender-receiver transmission of data elements with “event” semantic (swImplPolicy equal to queued). | | |
| Precautions | None | | |

#### Rte\_Invalidate

[SA\_R2\_318]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Invalidate | | |
| Function prototype: | Std\_ReturnType Rte\_Invalidate\_<p>\_<o> (  [IN Rte\_Instance ],  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、RTE\_E\_COM\_STOPPED、  RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Invalidate a data element for an “explicit” sender-receiver transmission. | | |
| Precautions | None | | |

#### Rte\_Read

[SA\_R2\_319]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Read | | |
| Function prototype: | Std\_ReturnType Rte\_Read\_<p>\_<o> (  [IN Rte\_Instance ],  OUT <data>,  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error  data:back the received data. | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、 RTE\_E\_INVALID、  RTE\_E\_MAX\_AGE\_EXCEEDED、 RTE\_E\_NEVER\_RECEIVED、  RTE\_E\_OUT\_OF\_RANGE、RTE\_E\_COM\_BUSY、RTE\_E\_COM\_STOPPED、  RTE\_E\_UNCONNECTED、  RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Performs an “explicit” read on a sender-receiver communication data element with “data” semantics (swImplPolicy != queued). | | |
| Precautions | None | | |

#### Rte\_Receive

[SA\_R2\_320]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Receive | | |
| Function prototype: | Std\_ReturnType Rte\_Receive\_<p>\_<o> (  [IN Rte\_Instance ],  OUT <data>,  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error  data:back the received data. | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、RTE\_E\_NO\_DATA、 RTE\_E\_INVALID、  RTE\_E\_TIMEOUT、RTE\_E\_LOST\_DATA、  RTE\_E\_IN\_EXCLUSIVE\_AREA、 RTE\_E\_UNCONNECTED、  RTE\_E\_HARD\_TRANSFORMER\_ERROR、RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Performs an “explicit” read on a sender-receiver communication data  element with “event” semantics (swImplPolicy = queued). | | |
| Precautions | None | | |

#### Rte\_IRead

[SA\_R2\_321]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IRead | | |
| Function prototype: | <return> Rte\_IRead\_<re>\_<p>\_<o> ([IN Rte\_Instance]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | return: according the applicable ImplementationDataType | | |
| Functional Overview: | Provide read access to the VariableDataPrototype referenced by VariableAccess in the dataReadAccess role. | | |
| Precautions | None | | |

#### Rte\_IWrite

[SA\_R2\_322]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IWrite | | |
| Function prototype: | void Rte\_IWrite\_<re>\_<p>\_<o> ([IN RTE\_Instance], IN <data>) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds  data:pass the data element to write | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void | | |
| Functional Overview: | Provide write access to the VariableDataPrototypes referenced by VariableAccesses in the dataWriteAccess role. | | |
| Precautions | None | | |

#### Rte\_IWriteRef

[SA\_R2\_323]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IWriteRef | | |
| Function prototype: | <return reference> Rte\_IWriteRef\_<re>\_<p>\_<o> ([IN RTE\_Instance]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | return reference: according the applicable ImplementationDataType | | |
| Functional Overview: | Provide a reference to the VariableDataPrototype referenced by a VariableAccess in the dataWriteAccess role | | |
| Precautions | None | | |

#### Rte\_IInvalidate

[SA\_R2\_324]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IInvalidate | | |
| Function prototype: | void Rte\_IInvalidate\_<re>\_<p>\_<o> (  [IN Rte\_Instance <instance>],  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error | | |
| Return value: | None | | |
| Functional Overview: | Invalidate a VariableDataPrototype referenced by a VariableAccess in the dataWriteAccess role. | | |
| Precautions | None | | |

#### Rte\_IStatus

[SA\_R2\_325]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IStatus | | |
| Function prototype: | Std\_ReturnType Rte\_IStatus\_<re>\_<p>\_<o> (  [IN Rte\_Instance],  [OUT Rte\_TransformerError transformerError]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | transformerError:transformer error | | |
| Return value: | Std\_ReturnType:  RTE\_E\_OK、RTE\_E\_INVALID、RTE\_E\_MAX\_AGE\_EXCEEDED、  RTE\_E\_NEVER\_RECEIVED、RTE\_E\_OUT\_OF\_RANGE、  RTE\_E\_COM\_STOPPED、RTE\_E\_UNCONNECTED、  RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR | | |
| Functional Overview: | Provide the error status of a VariableDataPrototype referenced by a VariableAccess in the dataReadAccess role. | | |
| Precautions | None | | |

#### Rte\_Feedback

[SA\_R2\_326]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Feedback | | |
| Function prototype: | Std\_ReturnType Rte\_Feedback\_<p>\_<o> ( [IN Rte\_Instance ]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType:  RTE\_E\_NO\_DATA、RTE\_E\_COM\_STOPPED、RTE\_E\_TIMEOUT、  RTE\_E\_TRANSMIT\_ACK、RTE\_E\_UNCONNECTED、RTE\_E\_IN\_EXCLUSIVE\_AREA、RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR、 | | |
| Functional Overview: | Provide access to acknowledgement notifications for explicit senderreceiver communication and to pass error notification to senders. | | |
| Precautions | None | | |

#### Rte\_IFeedback

[SA\_R2\_327]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IFeedback | | |
| Function prototype: | Std\_ReturnType Rte\_IFeedback\_<re>\_<p>\_<o> ( [IN RTE\_Instance ]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType:  RTE\_E\_NO\_DATA、RTE\_E\_COM\_STOPPED、RTE\_E\_TIMEOUT、 RTE\_E\_TRANSMIT\_ACK、RTE\_E\_UNCONNECTED、RTE\_E\_HARD\_TRANSFORMER\_ERROR、  RTE\_E\_SOFT\_TRANSFORMER\_ERROR、 | | |
| Functional Overview: | Provide access to acknowledgement notifications for implicit sender  receiver communication and to pass error notification to senders. | | |
| Precautions | None | | |

#### Rte\_IsUpdated

[SA\_R2\_328]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IsUpdated | | |
| Function prototype: | boolean Rte\_IsUpdated\_<p>\_<o> ( [IN RTE\_Instance ]) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | None Reentrancy | | |
| Input parameters: | Instance: instance of cds | Range: | None |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType:TRUE、FALSE | | |
| Functional Overview: | Provide access to the update flag for an explicit receiver.refers to 'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_07390] | | |
| Precautions | None | | |

## Inter-Partition Communication

### Function Description

[SA\_R2\_034]:Since the partitions may be separated by core boundaries or memory boundaries and since the partitions can be stopped and restarted independently.

分区可以由内核边界或内存边界分隔，并且分区可以独立停止和重新启动；

[SA\_R2\_035]:Inter partition communication uses the connectionless communication paradigm. Both parties in the communication cannot obtain the communication status through the port in time. Therefore, RTE needs to add a communication timeout detection function for inter-partition communication.

分区间通信不会通过端口，因此通信双方无法及时通过端口获取通信状态，因此RTE需要针对inter-partition通信添加通信超时检测功能；

[SA\_R2\_036]:It can not be expected that the RTE generated wrapper code running in one partition can directly access the memory objects assigned to the RTE part of another partition.

位于两个分区间的收发双方不能直接去访问另外一个分区中的内存；

[SA\_R2\_038]:The IOC offers communication of data to another core or between memory protected partitions with guarantee of data consistency.RTE can realize inter-partition communication through the IOC mechanism provided by the OS;

RTE可通过OS提供的IOC机制实现跨分区通信；



Figure 5‑12

[SA\_R2\_039]:The IOC API is not reentrant.RTE shall prevent concurrent access to the same IOC API from different ExecutableEntity execution-instances.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02737

IOC接口为不可重入类型，RTE需要保证多个Runnable不能同时并发访问相同的IOC接口

[SA\_R2\_040]:The IOC will use the appropriate mechanism to communicate between the partitions, whether it requires :

1. communicating with another core,
2. communicating with a partition with a different level of trust,
3. or communicating with another memory partition.

IOC使用3种场景：1.跨核通信；2.不同信任等级的分区间通信；3.内存保护打开下的跨分区通信；

[SA\_R2\_041]:The RTE shall not execute ExecutableEntitys in the context of IOC callbacks.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02736

RTE不能够在IOC的回调函数中去调用执行实体

[SA\_R2\_042]:The RTE Generator shall use a trusted function to call an ExecutableEntity of a RteSwComponentInstance of a different partition than the ExecutableEntity’s call context (OsTask, OsIsr or RTE API) when both partitions are on the same core.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_08903

当两个分区在同一个核上时，RTE 应使用可信函数来调用与 ExecutableEntity 的调用上下文（OsTask、OsIsr 或 RTE API）不同的分区的 RteSwComponentInstance 的 ExecutableEntity。

[SA\_R2\_043]:Direct start of an ExecutableEntity execution-instance by the mean of a trusted function shall only be used for the start of an ExecutableEntity in a Trusted Partition.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07606

Trusted function implement a function call from a trusted or untrusted to a trusted partition on the same core.

通过可信函数直接启动 ExecutableEntity 执行实例只能用于启动可信分区中的 ExecutableEntity。

[SA\_R2\_044]:RTE has to ensure, that the OS does not kill an RTE-generated task due to stopping or restarting a partition while this task is executing a function call to BSW or to the software component of another partition when this call is not a pure function.

RTE 必须确保OS不会因为停止或重新启动分区而终止 RTE 生成的任务（该任务正在执行对 BSW 或另一个分区的软件组件的函数调用）。

[SA\_R2\_045]:In a partitioned system that supports stop or restart of partitions, the RTE shall not use a direct function call (without use of OS call trusted function) from a task of an untrusted partition to BSW or to the SW-C of another partition.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02761

在支持停止或重启的分区中，RTE 不应使用从不受信任分区的任务到 BSW 或另一个分区的 SW-C 的直接函数调用（仅能使用OS 提供的可信函数）。

[SA\_R2\_046]:When a SW-C in an untrusted partition receives (OUT parameter) or provides (IN parameter with composite data type) an ArgumentDataPrototype or VariableDataPrototype, it hands over a pointer to a memory object to an RTE API. The RTE shall only forward this pointer to a trusted SW-C after it has checked that the whole memory object is owned by the caller’s partition.(eg.call CheckTaskMemoryAccess).Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02752

当不可信分区中的 SW-C 接收（OUT 参数）或提供（具有复合数据类型的 IN 参数）ArgumentDataPrototype 或 VariableDataPrototype 时，它会将指向内存对象的指针移交给 RTE API。 RTE 只有在检查了整个内存对象由调用者的分区拥有后，才应将此指针转发给受信任的 SW-C。

[SA\_R2\_047]:When a SW-C in an untrusted partition passes an ArgumentDataPrototype or VariableDataPrototype, as a reference type to a SW-C in a trusted partition (DATA\_REFERENCE as an IN parameter), the RTE shall only check that the caller’s partition owns the start address of the referenced memory.(eg.call CheckTaskMemoryAccess).Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02753

当不可信任分区中的 SW-C 传递 ArgumentDataPrototype 或 VariableDataPrototype 作为对受信任分区中 SW-C 的引用类型（DATA\_REFERENCE 作为 IN 参数）时，RTE 应仅检查调用者的分区是否拥有引用的内存。

[SA\_R2\_048]:The RTE Generator shall reject configurations where the handle TerminationAndRestart attribute of a SW-C is not set to canBeTerminatedAndRestarted and this SW-C is mapped on a Partition with the PartitionCanBeRestarted parameter set to TRUE.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07610

如果一个SWC所在分区的PartitionCanBeRestarted配置为TRUE，代表此分区可以被重启。那么SWC内部行为中handleTerminationAndRestart属性必须配置为canBeTerminatedAndRestarted 。否则RTE工具需要报错

[SA\_R2\_049]:Inter-Partition communication,the RTE shall perform the filtering depend on configuration attribute

(NonqueuedReceiverComSpec->filter).Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05500

Inter-Partition需要对接收到的无效值进行过滤

[SA\_R2\_050]:For inter-partition transmission of data elements, the RTE shall perform the fan-out to each receiver.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_06024

inter-partition通信支持1：n

[SA\_R2\_051]:For inter-partition communication with implicit dataWriteAccess the DataWriteCompletedEvent shall be fired if and only if a task/ISR2 terminates and the write-back copy actions to the global RTE-buffer are completed. In addition the execution of the data write operations at the data receiver partitions must have taken place. Thereby the return status of the IOC for the different write operations can be neglected. The transmission status shall be RTE\_E\_TRANSMIT\_ACK and can be collected with Rte\_IFeedback API.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_08020

inter-partition隐式通信中，当且仅当任务/ISR2 终止并且对全局 RTE 缓冲区的回写复制操作完成时，才会触发 DataWriteCompletedEvent

[SA\_R2\_052]:For inter-partition communication with explicit dataSendPoint the DataSendCompletedEvent shall be fired if and only if the sending to all receivers has been performed and the execution of the data write operations at the data receiver partitions have taken place. Thereby the return status of the IOC for the different write operations can be neglected. The transmission status shall be RTE\_E\_TRANSMIT\_ACK and can be collected with Rte\_Feedback API.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_08021

inter-partition显示通信中，当且仅当所有接收着在其分区上的数据写入操作时，才会触发 DataSendCompletedEvent事件

[SA\_R2\_054]If all the following conditions are satisfied:

• the server runnable’s property canBeInvokedConcurrently is set to TRUE

• the client and server execute in different partitions, i.e. inter-partition Client-Server communication

• the ServerCallPoint is Synchronous

• the OperationInvokedEvent is not mapped to an OsTask

the RTE Generator shall implement the Client-Server communication as a trusted function call.

跨分区C/S通信通过可信函数调用方式实现的情况

[SA\_R2\_057]:RTE\_E\_TIMEOUT–(Inter-ECU and InterPartition only) A timeout notification was received from COM or IOC before any error notification.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07637

需要支持超时检测

### Control Flow

Inter-Partiton通信可以通过两种方式实现：1.基于OS提供的IOC功能；2.基于OS提供的可信函数功能；

1. 基于IOC实现Inter-partition通信：

Last-is-best communication（非队列方式）：

[SA\_R2\_058]:

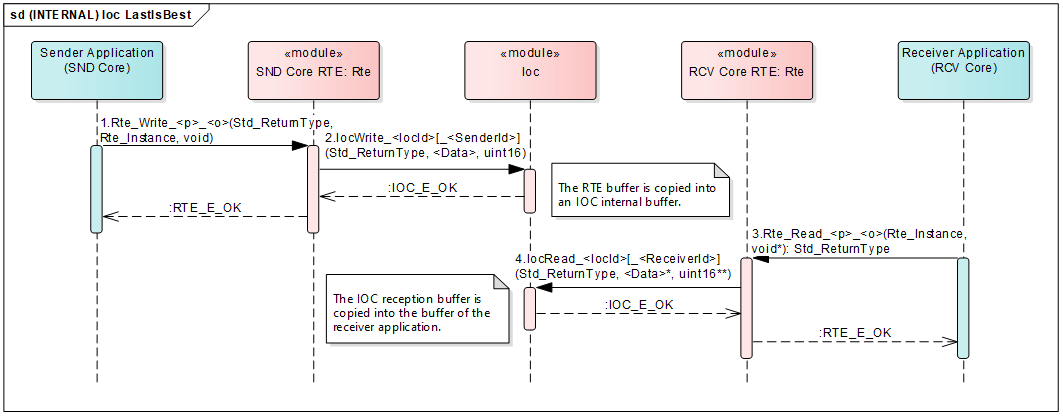


Figure 5‑13

如图所示，当inter-partition以非队列方式进行通信时，通信双发基于显示S/R通信（data semantic）方式，发送方会在Rte\_Write函数中去调用IocWrite接口把数据写到IOC的内部缓存区，接收方通过在Rte\_Read中调用IocRead获取通信数据。

Queued communication with pull callback（队列带回调通知）：

[SA\_R2\_059]:

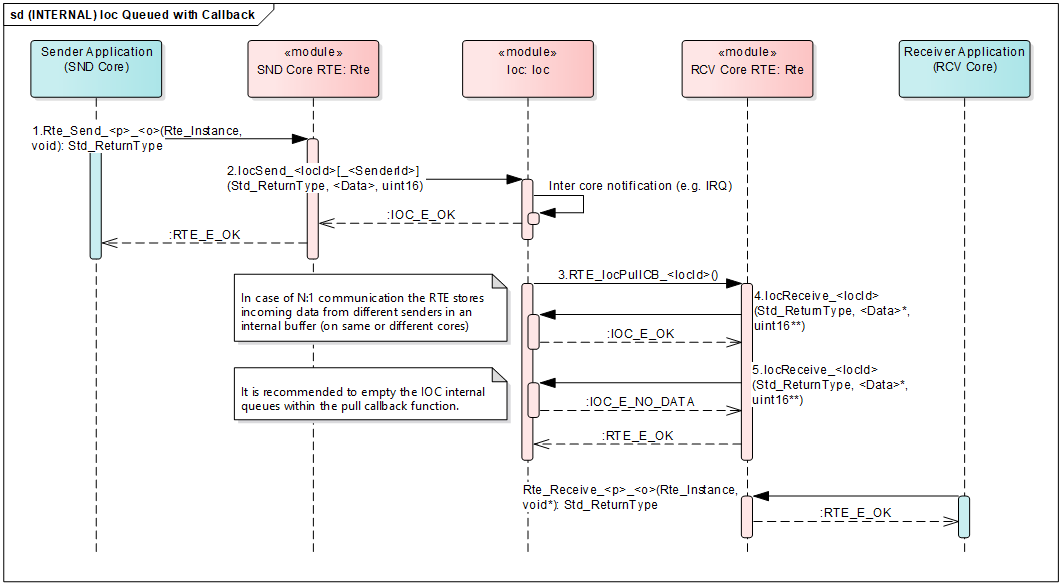


Figure 5‑14

如图所示，当inter-partition以队列（配置回调通知函数）进行通信时，通信双发基于显示S/R通信（event semantic）方式，会在Rte\_Send函数中去调用IocSend接口把数据写到IOC的内部缓存区，并且发送方会以回调函数方式通知到接收方。接收方收到通知后，通过在Rte\_Receive中调用IocReceive获取通信数据。

Queued communication without pull callback（队列不带回调通知）：

[SA\_R2\_060]:

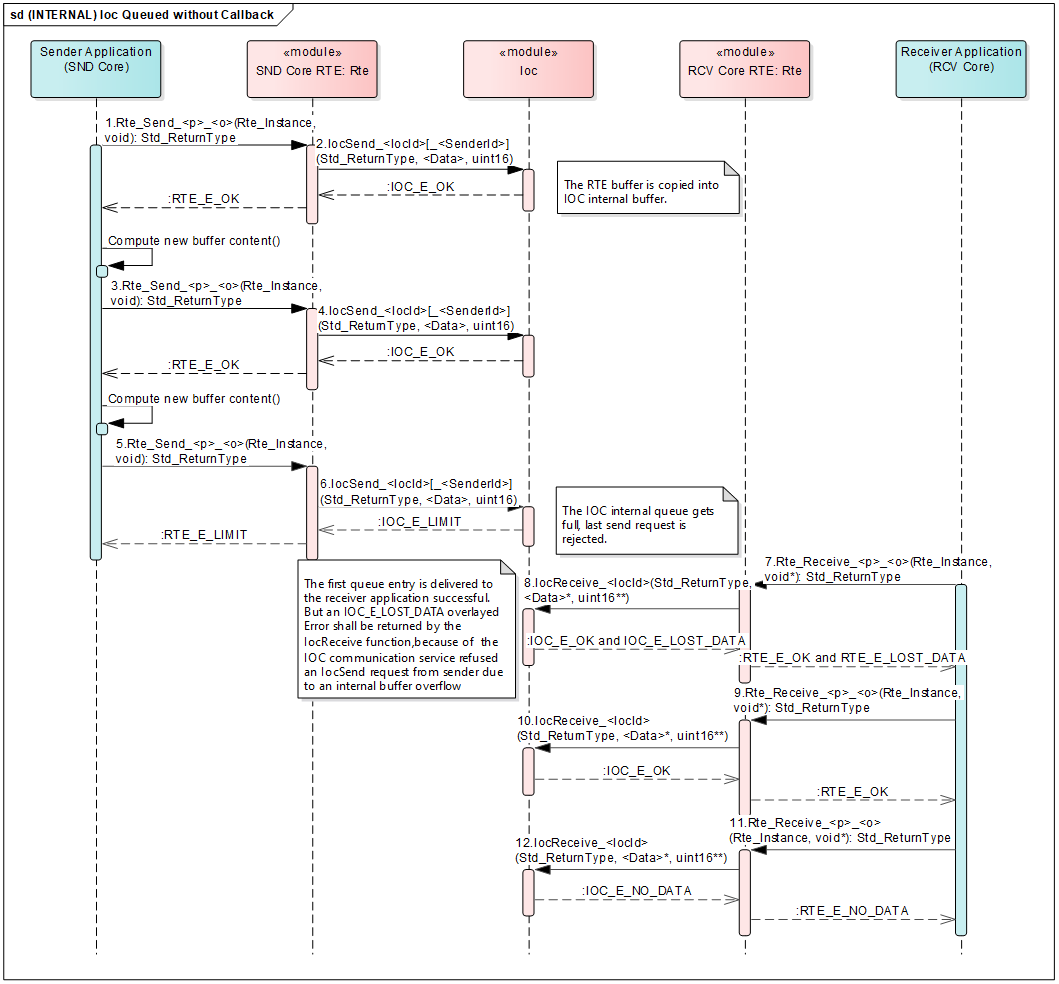


Figure 5‑15

如图所示，当inter-partition以队列（未配置回调通知函数）进行通信时，通信双发基于显示S/R通信（event semantic）方式，会在Rte\_Send函数中去调用IocSend接口把数据写到IOC的内部缓存区。接收方会周期去调用Rte\_Receive->IocReceive获取通信数据。

以队列方式进行通信时，需要配置其队列长度（RTE以及IOC都需要配置，需要保证其队列长度一致），如上图中，配置的队列长度为2，那么前两次调用Rte\_Send时均能正常返回IOC\_E\_OK，第三次调用时，即会回复IOC\_E\_LIMIT指示队列长度已满；接收方在获取数据时，第一次调用Rte\_Receive时能够正常得到数据值，但是返回值会回复RTE\_E\_LOST\_DATA，用于指示最近一次Rte\_Send写入数据失败。第二次调用Rte\_Receive时能够正常得到数据值以及返回值，第三次调用时，即会返回RTE\_E\_NO\_DATA表示无数据可读。

1. 基于Trusted Functions实现Inter-partition通信：

[SA\_R2\_061]:

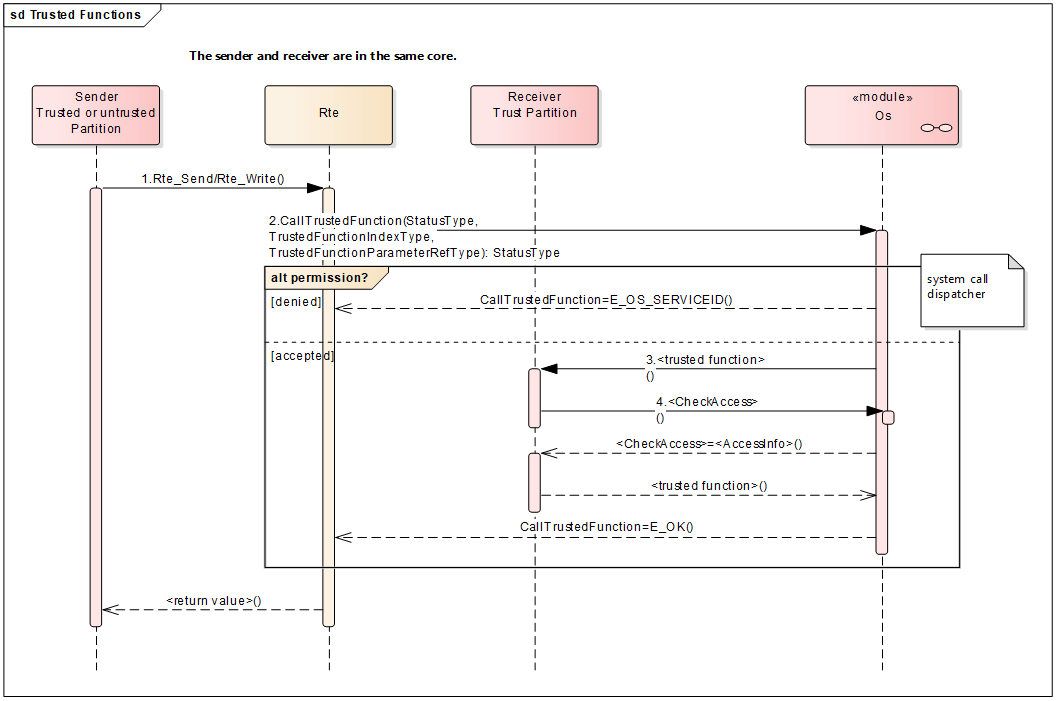


Figure 5‑16

如图所示，当发送方以及接收方处于同一个核的不同分区，且接收方处于可信分区时，inter-partition通信可以通过OS提供的可信函数来实现，那么发送方直接在Rte\_Write/Rte\_Send中直接调用CallTrustdFunction即可以达到双方通信的目的。

### Data Flow

[SA\_R2\_298]:

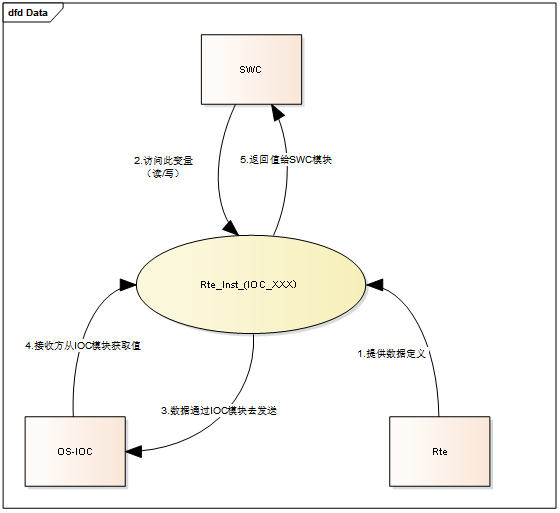


Figure 5‑17 IOC内部变量Rte\_Inst\_(IOC\_XXX）

### External Interface Definition

参考S/R通信章节。

## C-S Communication

[Client-server communication involves two entities, the client which is the requirer of a service and the server that provides the service. The client initiates the communication, requesting that the server performs a service, transferring a parameter set if necessary. The server, in the form of the RTE, waits for incoming communication requests from a client, performs the requested service and dispatches a response to the client’s request. So, the direction of initiation is used to categorize whether a AUTOSAR software-component is a client or a server. A single component can be both a client and a server depending on the software realization.] refers to 'AUTOSAR\_SWS\_RTE' 4.3.2.1. RTE is responsible for performing the whole invocation process by providing APIs. These APIs use os mechanism involving tasks, event and ticks to perform function calls. Currently our RTE supports intra-ECU C-S Communication.

### Function Description

[SA\_R2\_062]: [The RTE shall support multiple-client-single-server ("n:1") client-server (function invocation) communication. Individual clients are independent - there is no coordination of requests between clients. Single-client-multiple-server ("1:n") communication is not required.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00029].RTE应支持多客户端-单服务器(“n:1”)客户端-服务器(函数调用)通信。单个客户端是独立的——客户端之间不需要协调请求，(1:n)通信不支持

[SA\_R2\_063]: [The RTE shall support at most one asynchronous call at a time from a single operation in a required port categorized by a client-server interface (i.e. there can only be one outstanding request per "AsynchronousServerCallPoint"). Note that a single client can simultaneously have multiple outstanding requests provided each is to different server operations.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00079].RTE每次最多支持一个来自客户端-服务器接口分类的端口中单个操作的异步调用(即每个“AsynchronousServerCallPoint”只能有一个未完成的请求)。请注意，单个客户端可以同时有多个未完成的请求，只要每个请求针对不同的服务器操作。

[SA\_R2\_064]: The RTE shall support synchronous server call for client-server communication.

[SA\_R2\_065]: [The RTE shall support serialized and non-serialized execution of Server Runnable Entities. The RTE shall complete the processing of one serialized service request before it accepts and dispatches the next request for that server.

The serialization is applied on the service level, so one server can handle multiple service calls concurrently (this implies that the service’s Runnable Entities are mapped to different tasks and there is no shared data between them).

If serialization is supported by a server and how big the actual queue shall be configurable.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00033].

RTE应该支持服务器可运行实体的序列化和非序列化执行。RTE应完成一个序列化服务请求的处理，然后接受并向该服务器发送下一个请求。

序列化应用于服务级别，因此一台服务器可以并发处理多个服务调用(这意味着服务的可运行实体被映射到不同的任务，它们之间没有共享数据)。

如果服务器支持序列化，那么实际队列的大小是可配置的

[SA\_R2\_066]: [Unless a server call is implemented as a direct function call, the RTE shall buffer a request on the server side in a first-in-first-out queue] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_04515].

除非一个服务器调用是作为一个直接的函数调用来实现的，RTE应该在服务器端以一个先进先出的队列缓冲一个请求

[SA\_R2\_067]: [Unless a server call is implemented as a direct function call, RTE shall keep the response on the client side in a queue with queue length 1.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02528].

[SA\_R2\_068]: [The RTE shall use the queue of requests to call serialise access to a server.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02530].

RTE应使用请求队列调用对服务器的序列化访问

[SA\_R2\_069]: [The RTE’s implementation of the client-server communication shall ensure that a service result is dispatched to the correct client if more than one client uses a service.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_04516].

当多个客户端使用一个服务时，RTE实现的客户端-服务器通信应确保服务结果被分派到正确的客户端

[SA\_R2\_070]: [If all the following conditions are satisfied:

• the server runnable’s property canBeInvokedConcurrently is set to TRUE

• the client and server execute in the same partition, i.e. intra-partition Client-Server communication

• the ServerCallPoint is Synchronous

• the OperationInvokedEvent is not mapped to an OsTask

the RTE Generator shall implement the Client-Server communication as a direct function call.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07409].

[SA\_R2\_071]: [The RTE shall support simultaneous invocation requests of a server operation.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_04520].

RTE应支持服务器操作的同时调用请求

[SA\_R2\_072]: [The RTE shall ensure that the RunnableEntity implementing a server operation has completed the processing of a request before it begins processing the next request.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_04522].RTE应确保实现服务器操作的RunnableEntity在开始处理下一个请求之前已经完成了对请求的处理

[SA\_R2\_074]: [The RTE shall ensure that timeout monitoring is performed for client-server communication, regardless of the receive mode for the result.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03763].超时检测

[SA\_R2\_075]: [For each asynchronous invocation of an operation prototype only one AsynchronousServerCallReturnsEvent shall be passed to the client component by the RTE. The AsynchronousServerCallReturnsEvent shall indicate either that the transmission was successful or that the transmission was not successful.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03765].对于每个操作原型的异步调用，RTE只需要将一个AsynchronousServerCallReturnsEvent传递给客户端组件。AsynchronousServerCallReturnsEvent将指示传输是否成功

[SA\_R2\_076]: [The status information about the success or failure of the asynchronous server invocation shall be available as the return value of the RTE API call to retrieve the result.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03766].

关于异步服务器调用成功或失败的状态信息应作为检索结果的RTE API调用的返回值

[SA\_R2\_077]: [When invoking the runnable entity specified for an OperationInvokedEvent, the RTE shall include the port-defined argument values between the instance handle (if it is included) and the operation-specific parameters, in the order they are given in the Software Component Template Specification.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01360].

当调用为OperationInvokedEvent指定的可运行实体时，RTE应按照软件组件模板规范中给出的顺序，在实例句柄(如果包含的话)和特定于操作的参数之间包含端口定义的参数值。

[SA\_R2\_078]: RTE shall have support all types of arguments including IN, INOUT, and OUT in C-S operation.

[SA\_R2\_079]: RTE should support C-S error call API detection.

[SA\_R2\_080]: RTE should be able to support queue length detection of C-S communication

### Control Flow

[SA\_R2\_081]:

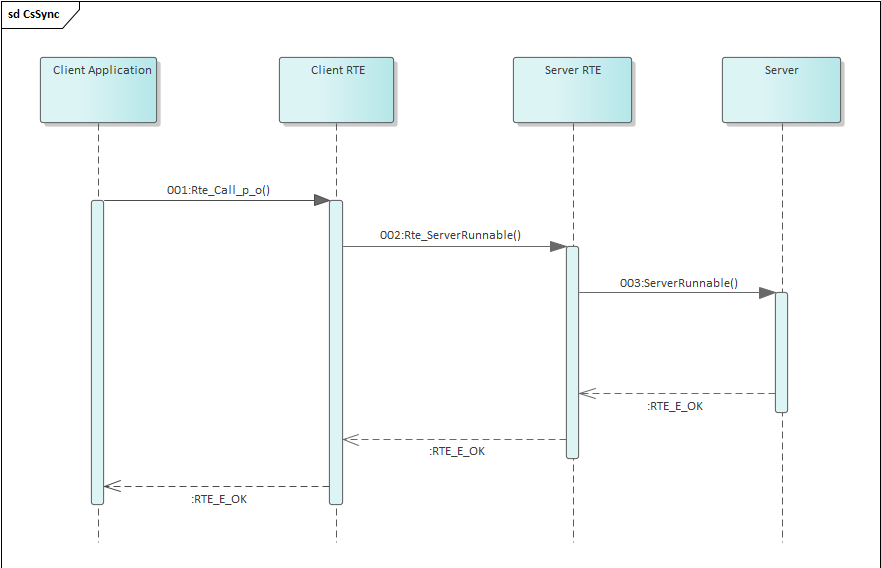


Figure 5‑18 The Daigram describe a CS communication of Synchronous.

图示为ECU内同步的CS通信示意图。当Client Application 发起一个请求服务器则调用Rte\_Call\_p\_o()函数，而后RTE通过调用Rte\_ServerRunnable()去调用Server端的ServerRunnable()函数，值得注意的是，这时Client会一直阻塞等待Server端处理完返回。

[SA\_R2\_082]:

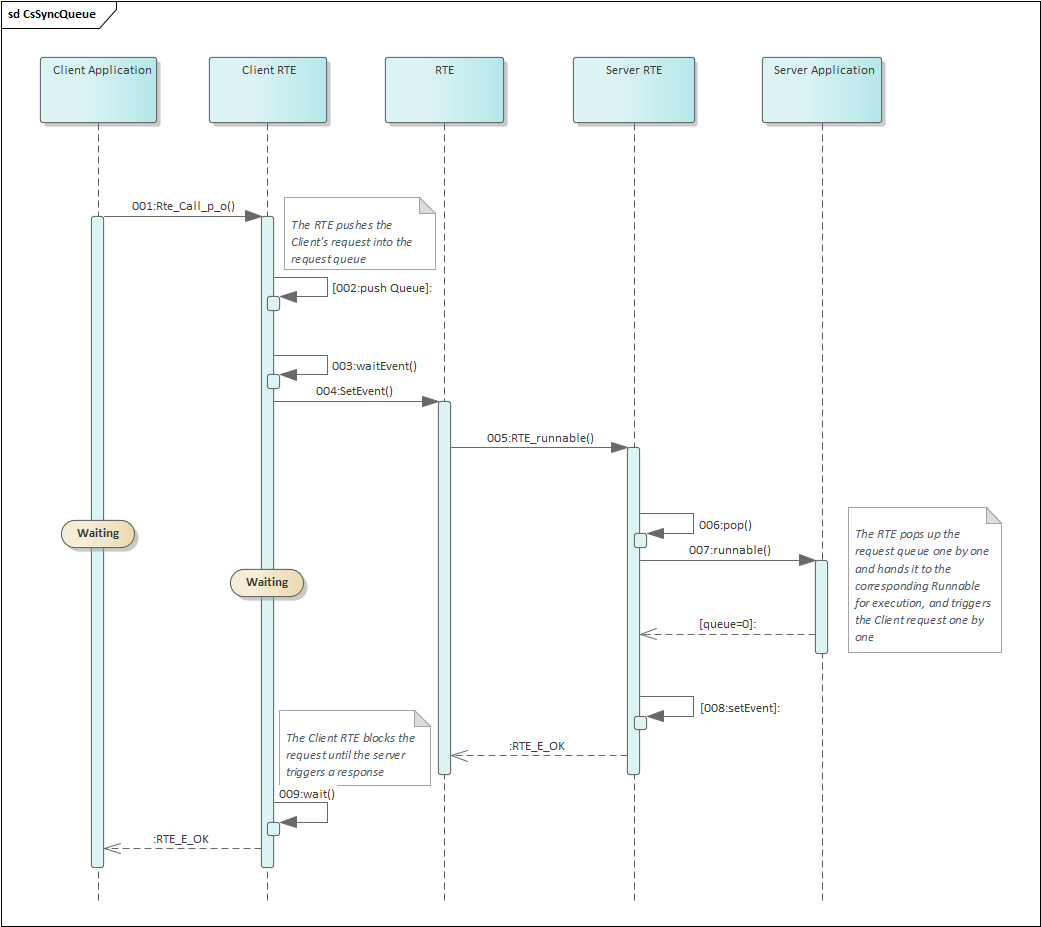


Figure 5‑19 The Daigram describe a CS communication of Synchronous about queue.

图示为ECU内的同步队列的CS通信示意图。当Client请求服务器时则调用Rte\_Call\_p\_o()函数，然后Client RTE将客户端的请求压入到维护的队列中并触发RTE waitEvent等待的事件和任务，然后Client开始阻塞等待响应事件触发，RTE阻塞被触发后则将对应请求的队列进行处理，将队列中的元素挨个弹出执行知道队列元素为空，并在每个请求处理完后触发对应的Client RTE，队列处理完后RTE将继续处于阻塞状态等待触发。Client RTE被触发后阻塞就会结束，然后返回给Client RTE\_E\_OK，这时一个同步的队列请求结束。

[SA\_R2\_083]:

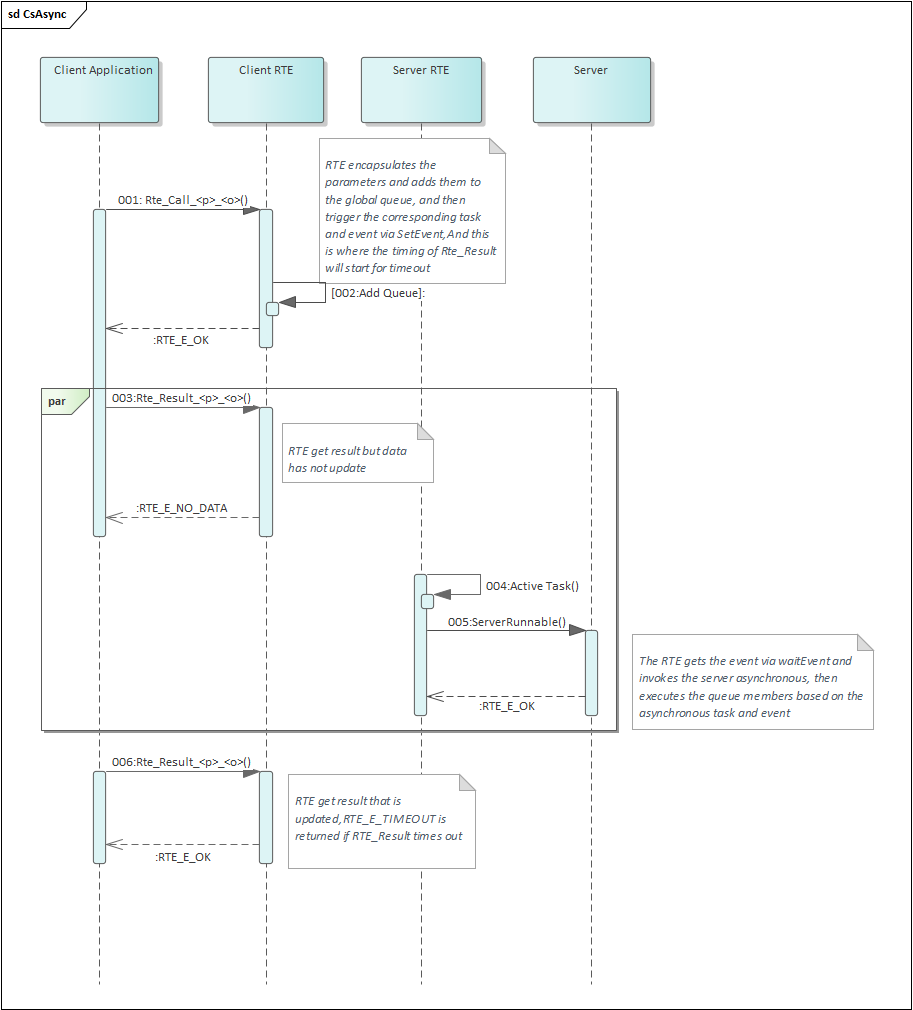


Figure 5‑20 The Diagram describe a CS communication of Asynchronous.

图示为ECU内的异步CS通信示意图。当Client Application请求服务器时则调用Rte\_Call\_p\_o()函数，而后RTE将client的请求添加到全局的异步请求队列中(此时RTE会对队列长度进行检查，超出队列可维护长度则返回RTE\_E\_LIMIT)，压入到队列后，RTE会通过SetEvent调用激活Server端的task任务，RTE获取到队列后，根据task和event去触发服务器异步调用。值得注意的是当Server还没处理结束时，Client端通过Rte\_Result\_<p>\_<o>()获取结果时，RTE将返回Client端RTE\_E\_NO\_DATA，当Server端处理完时，Client端通过Rte\_Result\_<p>\_<o>()获取结果将返回RTE\_E\_OK，如果获取超时则返回RTE\_E\_TIMEOUT。

### Data Flow

[SA\_R2\_296]:

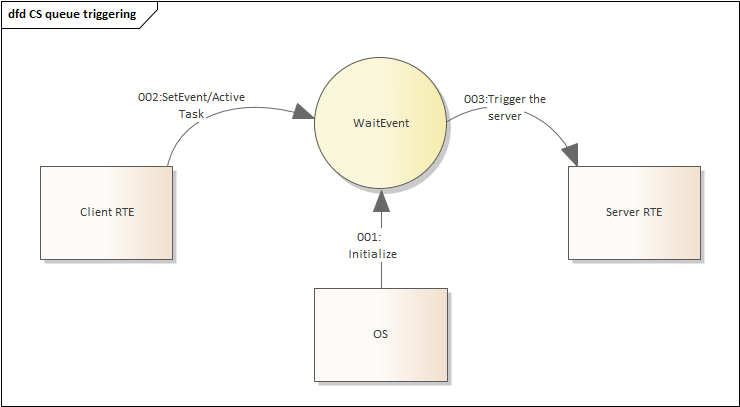


Figure 5‑21 Data flow diagram describing a CS communication of Asynchronous.

As figure 5-20 shows, Client RTE setEvent or activate task when calling server, which triggers the server to execute and provide the result.

For queued communication data flow, refer to 5.11.3

### External Interface Definition

#### Rte\_Call

[SA\_R2\_084]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Call | | |
| Function prototype: | Std\_ReturnType Rte\_Call\_<p>\_<o>(  [IN Rte\_Instance <instance>],  [IN|IN/OUT|OUT] <data\_1>...  [IN|IN/OUT|OUT] <data\_n>) | | |
| Synchronous/Asynchronous: | Asynchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | Instance:cds的instance  data\_n: <data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | Range: | Pointing to Static configuration |
| Input and output parameters: | data\_n: <data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | | |
| Output parameters: | transformerError:transformer error  data\_n:<data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、RTE\_E\_LIMIT、RTE\_E\_TRANSFORMER\_LIMIT、RTE\_E\_HARD\_TRANSFORMER\_ERROR、RTE\_E\_SOFT\_TRANSFORMER\_ERROR、  RTE\_E\_COM\_STOPPED、RTE\_E\_TIMEOUT、RTE\_E\_UNCONNECTED、  RTE\_E\_IN\_EXCLUSIVE\_AREA、RTE\_E\_SEG\_FAULT、RTE\_E\_COM\_BUSY | | |
| Functional Overview: | Initiate a client-server communication. | | |
| Precautions | None | | |

#### Rte\_Result

[SA\_R2\_329]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Result | | |
| Function prototype: | Std\_ReturnType Rte\_Result\_<p>\_<o>( [IN Rte\_Instance <instance>],  [IN/OUT|OUT <param 1>]...  [IN/OUT|OUT <param n>]) | | |
| Synchronous/Asynchronous: | Asynchronous | | |
| Is it reentrant: | Reentrancy | | |
| Input parameters: | Instance:cds的instance  data\_n: <data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | Range: | Pointing to Static configuration |
| Input and output parameters: | data\_n: <data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | | |
| Output parameters: | transformerError:transformer error  data\_n:<data\_1>...<data\_n> are the parameters configured in SWC C-S Interface. | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK、RTE\_E\_LIMIT、RTE\_E\_TRANSFORMER\_LIMIT、RTE\_E\_HARD\_TRANSFORMER\_ERROR、RTE\_E\_SOFT\_TRANSFORMER\_ERROR、  RTE\_E\_COM\_STOPPED、RTE\_E\_TIMEOUT、RTE\_E\_UNCONNECTED、  RTE\_E\_IN\_EXCLUSIVE\_AREA、RTE\_E\_SEG\_FAULT、RTE\_E\_COM\_BUSY | | |
| Functional Overview: | Get the result of an asynchronous client-server call | | |
| Precautions | None | | |

## Mode Management

Modes can be used to trigger runnables: The SwcInternalBehavior of the AUTOSAR SW-C can define a SwcModeSwitchEvent referencing the required ModeDeclaration.This SwcModeSwitchEvent can then be used as trigger for a RunnableEntity. SwcModeSwitchEvent carry an attribute ModeActivationKind which can be exit , entry , or transition .

A RunnableEntity that is triggered by a SwcModeSwitchEvent with ModeActivationKind exit is triggered on exiting the mode. For simplicity it will be called on-exit ExecutableEntity.

An on-transition ExecutableEntity is triggered by a SwcModeSwitchEvent with ModeActivationKind ‘transition’ and will be executed during the transition between two modes.

An on-entry ExecutableEntity is triggered by a SwcModeSwitchEvent with ModeActivationKind ‘entry’ and will be executed when the mode is entered.

An RTEEvent that starts a ExecutableEntity can contain a mode disabling dependency.

The RTE will take the actions necessary to switch between the modes. This includes the termination and execution of several ExecutableEntities from all mode users that are connected to the same ModeDeclarationGroupPrototype of the mode manager. To do so, the RTE needs a state machine to keep track of the currently active modes and transitions initiated by the mode manager. The RTE s mode machine is called mode machine instance. There is exactly one mode machine instance for each ModeDeclarationGroupPrototype of a mode manager s provide mode switch port respectively providedModeGroup ModeDeclarationGroupPrototype.

Entering and leaving modes is initiated by a mode manager. A mode manager might be a basic software module, for example the Basic Software Mode Manager (BswM), the communication manager (ComM), or the ECU state manager (EcuM). The mode manager may also be an AUTOSAR SW-C. In this case, it is called an application mode manager.

It is the responsibility of the mode manager to advance the RTE s mode machine instance by sending mode switch notifications to the mode users. The mode switch notifications are implemented by a non blocking API. So, the mode switch notifications alone provide only a loose coupling between the state machine of the mode manager and the mode machine instance of the RTE. To prevent, that the mode machine instance lags behind and the states of the mode manager and the RTE get out of phase, the mode manager can use acknowledgment feedback for the mode switch notification. RTE can be configured to send an acknowledgment of the mode switch notification to the mode manager when the requested transition is completed.

### Function Description

[SA\_R2\_085]: An Basic Software Module (mode user) has to define a requiredModeGroup ModeDeclarationGroupPrototype. The ModeDeclarationGroup referred by these ModeDeclarationGroupPrototype contains the required modes.

[SA\_R2\_086]: The ModeDeclarations can be used in two ways by the mode user:

1. Modes can be used to trigger runnables: The SwcInternalBehavior of the AUTOSAR SW-C can define a SwcModeSwitchEvent referencing the required ModeDeclaration. This SwcModeSwitchEvent can then be used as trigger for a RunnableEntity. Both SwcModeSwitchEvent carry an attribute ModeActivationKind which can be exit , entry , or transition .

2. An RTEEvent that starts a ExecutableEntity can contain a mode disabling dependency.

[SA\_R2\_087]: [If a RunnableEntity r is referenced with startOnEvent by an RTEEvent e that has a mode disabling dependency on a mode m, then RTE shall not activate runnable r on any occurrence of e while the mode m is active.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02503].

[SA\_R2\_088]: [The existence of a mode disabling dependency shall not instruct the RTE to kill a running runnable at a mode switch.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02504].

[SA\_R2\_089]: The RTE needs a state machine to keep track of the currently active modes and transitions initiated by the mode manager. The RTE s mode machine is called mode machine instance.

[SA\_R2\_090]: There is exactly one mode machine instance for each ModeDeclarationGroupPrototype of a mode manager’s provide mode switch port respectively providedModeGroup ModeDeclarationGroupPrototype.

[SA\_R2\_091]: The mode manager advances the mode machine instance of the RTE by sending mode switch notifications to mode users using a non-blocking API (Rte\_Switch).

[SA\_R2\_092]: To prevent, that the mode machine instance lags behind and the states of the mode manager and the RTE get out of phase, RTE can be configured to send an acknowledgment of the mode switch notification to the mode manager when the requested transition is completed.

1. Use a looped RunnableEntit to poll for acknowledgments using Rte\_SwitchAck.

2. Polls for acknowledgments from RunnableEntity initiated by ModeSwitchedAckEvent.

[SA\_R2\_093]:Mode managers as well as mode users can use the Rte\_Mode API to read the current active mode from the RTE's perspective. If the mode machine instance is otherwise in transition, provide the values of the previous mode and the next mode.

[SA\_R2\_094]: [The RTE generator shall use the synchronous mode switching procedure if at least one mode user of the mode machine instance does not support the asynchronous mode switch behavior. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07150].

[SA\_R2\_281]: [The RTE generator shall apply the asynchronous mode switch behavior, if all mode users support the asynchronous mode switch behavior and if it is configured for the related mode machine instance. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07151].

[SA\_R2\_095]: [The RTE generator shall support invocation of on-entry ExecutableEntitys, on-transition ExecutableEntitys, on-exit ExecutableEntitys and ModeSwitchAck ExecutableEntitys via direct function call, if all following conditions are fulfilled:

• if the asynchronous mode switch behavior is configured.

• the on-entry ExecutableEntitys, on-transition ExecutableEntitys, on-exit ExecutableEntitys and ModeSwitchAck ExecutableEntitys do not define a minimum start distance.

• the mode manager and mode user are in the same Partition.

] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07173].

[SA\_R2\_096]:[The RTE generator shall reject configurations with on-entry, on-transition, or on-exit ExecutableEntity s of the same core local mode user group that are mapped to different tasks in case of synchronous mode switching procedure. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02662].

[SA\_R2\_097]: [Within the mode manager s Rte\_Switch API call to indicate a mode switch, one of the following shall be done:

1.If the corresponding mode machine instance is in a transition, and the queue for mode switch notifications is full, Rte\_Switch shall return an error immediately.

2.If the corresponding mode machine instance is in a transition, and the queue for mode switch notifications is not full, the mode switch notification shall be queued.

3.If the mode machine instance is not in a transition, Rte\_Switch shall initiate the transition. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02667].

[SA\_R2\_098]: [During a transition of a mode machine instance each applicable of the steps:

1. At the beginning of a transition of a mode machine instance, the RTE shall activate the mode disablings of the next mode, if any mode disabling dependencys for that mode are defined. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02661].

2. If any mode disabling dependencys for the next mode are defined, the RTE shall wait until the newly disabled RunnableEntitys are terminated, in case of synchronous mode switching procedure. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07152].

3. RTE shall execute the on-exit ExecutableEntitys of the previous mode. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02562].

4. If any on-exit ExecutableEntity is configured the RTE shall wait after its execution until all on-exit ExecutableEntitys are terminated in case of synchronous mode switching procedure. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07153].

5. RTE shall execute the on-transition ExecutableEntitys configured for the transition from previous mode to next mode. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02707].

6. If any on-transition ExecutableEntity is con- figured, the RTE shall wait after its execution until all on-transition ExecutableEntitys are terminated in case of synchronous mode switching procedure. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02708].

7. RTE shall execute the on-entry ExecutableEntitys of the next mode. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02564].

8. If any on-entry ExecutableEntity is configured the RTE shall wait after its execution until all on-entry ExecutableEntitys are terminated in case of synchronous mode switching procedure. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07154].

9. The RTE shall deactivate the previous mode disablings and only keep the mode disablings of the next mode. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02563].

10. At the end of the transition, RTE shall trigger the ModeSwitchedAckEvents connected to the mode manager s ModeDeclarationGroupPrototype.

] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02665].

Note: shall be executed in the order as listed for a core local mode user group. If a step is not applicable, the order of the remaining steps shall be unchanged.

[SA\_R2\_099]: [If the next mode and the previous mode of a transition are the same, the transition shall still be executed. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02669].

[SA\_R2\_100]: [For common mode machine instances the on-entry Runnable Entities of the current active mode are executed during the initialization of the RTE if the common mode machine instance is not in transition. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07582].

[SA\_R2\_101]: [A common mode machine instances is not allowed to enter transition phase during the RTE initialization if the common mode machine instances has on-entry Runnable Entities, on-transition Runnable Entities or on-exit Runnable Entities. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07583].

[SA\_R2\_102]: [Mode switches shall be communicated via RTE by ModeDeclarationGroupPrototypes of a ModeSwitchInterface. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02549].

Note: The mode switch ports of the mode manager and the mode user are of the type of a ModeSwitchInterface.

[SA\_R2\_103]: [A mode switch shall be notified asynchronously as indicated by the use of a ModeSwitchInterface. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02508].

[SA\_R2\_104]: [A ModeSwitchInterface shall support 1:n communication. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02566].

[SA\_R2\_105]: [The connection of providedModeGroup and requiredModeGroup ModeDeclarationGroupPrototype shall support 1:n communication. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07539].

[SA\_R2\_106]: [RTE shall not support connections with multiple senders (n:1 communication) of mode switch notifications connected to the same receiver. The RTE generator shall reject configurations with multiple senders of mode switch notifications connected to the same receiver. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02670].

[SA\_R2\_107]: [A mode switch shall be notified with event semantics, i.e., the mode switch notifications shall be buffered by RTE to which the mode machine instance is assigned. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02624].

[SA\_R2\_108]: [The RTE shall implement a receive queue for the mode switch notifications of each mode machine instance. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02718].

[SA\_R2\_109]: [The mode switch notification shall be written to the end of the queue and read (consuming) from the front of the queue (i.e. the queue is first-in-first-out). ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02719].

[SA\_R2\_110]: [If a new mode switch notification is received when the queue is already filled, the RTE shall discard the received notification. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02720].

[SA\_R2\_111]: [RTE shall dequeue a mode switch notification, when the mode switch is completed. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02721].

[SA\_R2\_112]: [The RTE generator shall reject a queueLength attribute of an ModeSwitchSenderComSpec with a queue length 0. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02723].

Note: In case of a queue length of 1, RTE will reject new mode switch notifications during the mode transition.

[SA\_R2\_113]: [A ModeSwitchInterface shall only indicate the next mode of the transition. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02567].

[SA\_R2\_114]: [A providedModeGroup ModeDeclarationGroupPrototype shall only indicate the next mode of the transition. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07541].

[SA\_R2\_115]: [The RTE shall keep track of the active modes of a mode manager s ModeDeclarationGroupPrototypes (mode machine instances) which is assigned to the RTE. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02546].

[SA\_R2\_116]: [If acknowledgment is enabled for a provided ModeDeclarationGroupPrototype and a ModeSwitchedAckEvent references a RunnableEntity as well as the ModeDeclarationGroupPrototype, the RunnableEntity shall be activated when the mode switch acknowledgment occurs or when the RTE detects that the partition to which the mode users are mapped was stopped or restarted. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02679].

[SA\_R2\_117]: [If ModeSwitchedAckRequest or BswModeSwitchAckRequest with a timeout greater than zero is specified, the RTE shall ensure that timeout monitoring is performed, regardless of the receive mode of the acknowledgment.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07056].

[SA\_R2\_118]: [Regardless of an occurred timeout during a mode transition the RTE shall complete the transition of a mode machine instance.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07060].

[SA\_R2\_119]: [The status information about the success or failure of the mode transition shall be buffered with last-is-best semantics. When a new mode switch notification is sent or when the mode switch notification was completed after a timeout, the status information is overwritten. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07058].

[SA\_R2\_120]: [If the timeout value of the ModeSwitchedAckRequest or BswModeSwitchAckRequest is 0, no timeout monitoring shall be performed. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07059].

[SA\_R2\_121]: [The RTE Generator shall take the modeManagerErrorBehavior from the ModeDeclarationGroup typing the ModeDeclarationGroupPrototype in the ModeSwitchInterface of the PPortPrototype/PRPortPrototype. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06794].

[SA\_R2\_122]: [The RTE shall execute the error reactions in case the partition of the mode users gets terminated in following order:

1. [The RTE shall clear all mode switch notifications in the queue when all partitions of the mode usersS are terminated. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06772].
2. [The RTE shall activate RunnableEntitys triggered by a SwcModeManagerErrorEvent when all partitions of the mode usersS are terminated. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06773].
3. [If ModeSwitchedAckRequest is specified, the RTE shall detect a timeout when mode users partitions are terminated during an ongoing transition. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06774].
4. The following steps select the appropriate:
   1. If the attribute modeManagerErrorBehavior.errorReactionPolicy is set to lastMode the mode machine instance stays in the last mode before the termination of the mode users. If the partition of the mode users gets terminated during an ongoing transition the last mode is the next mode of the transition.
   2. If the attribute modeManagerErrorBehavior.errorReactionPolicy is set to defaultMode the RTE shall enqueue the mode defined by modeManagerErrorBehavior.defaultMode to the mode switch notification queue.
   3. If the attribute modeManagerErrorBehavior is not defined the RTE shall enqueue the mode defined by initialMode to the mode switch notification queue. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06775, SWS\_Rte\_06776, SWS\_Rte\_06777].

] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06778].

[SA\_R2\_123]: [RTE shall support the enqueueing of new mode switch requests during the restart of the mode user’s partition by the mode manager after the call of Rte\_PartitionRestarting. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06779].

[SA\_R2\_124]: [When the partition with the mode users is restarted (after call of Rte\_PartitionRestart), RTE shall dequeue queued mode switch notifications. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06780].

[SA\_R2\_125]: Mode machine instances can remain in the last mode, which can be used to remain "as is" until the mode manager is restarted.

[SA\_R2\_126]: [The RTE Generator shall take the modeUserErrorBehavior from the ModeDeclarationGroup typing the ModeDeclarationGroupPrototype in the ModeSwitchInterface of the PPortPrototype/PRPortPrototype. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06795].

[SA\_R2\_127]: [The RTE shall execute the error reactions in case the partition of the mode manager gets terminated in the following order:

1. If the partition of the mode manager gets terminated during an ongoing transition, the RTE shall complete the transition. If the partition of the mode manager gets terminated during an ongoing transition, the RTE shall skip the mode switch acknowledgment. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06785, SWS\_Rte\_06786]
2. The RTE shall clear all mode switch notifications in the queue when the partition of the mode manager gets terminated and after an ongoing transition is completed. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06787]
3. The following steps select the appropriate:
   1. If the attribute modeUserErrorBehavior.errorReactionPolicy is set to lastMode the mode machine instance stays in the last mode before the termination of the mode manager. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06788].
   2. If the attribute modeUserErrorBehavior.errorReactionPolicy is set to defaultMode the RTE shall enqueue the mode defined by modeUserErrorBehavior.defaultMode to the mode switch notification queue. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06789].
   3. If the attribute modeUserErrorBehavior is not defined the RTE shall enqueue the mode defined by initialMode to the mode switch notification queue. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06790].

] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06791].

[SA\_R2\_128]: [The RTE shall dequeue queued mode switch notifications and execute them regardless whether the partition with the mode manager is terminated, restarting or restarted. Thereby the restart of the mode manager s partition shall not abort the ongoing transition of a mode machine instance. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06792].

[SA\_R2\_129]: [The RTE shall activate RunnableEntitys triggered by a SwcModeManagerErrorEvent when the partition of the mode manager is restarted. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06793].

[SA\_R2\_130]: [If a ModeDeclaration of a mode user is mapped to a single ModeDeclaration of a mode manager the related mode of the mode user is entered or exit when the mapped mode of the mode manager is entered or exit. ] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08511].

[SA\_R2\_131]: RTE为每个 mode machine instance分别提供一套接口。在同一个mode machine instance中，模式切换使用同一个API切换模式。

[SA\_R2\_291]: RTE shall initiate the transition to the initial modes of each mode machine instance belonging to the RTE during Rte\_Start if the on-entry Runnable Entities for the initialMode are not mapped to any RteInitializationRunnableBatch container.

### Control Flow

[SA\_R2\_132]:

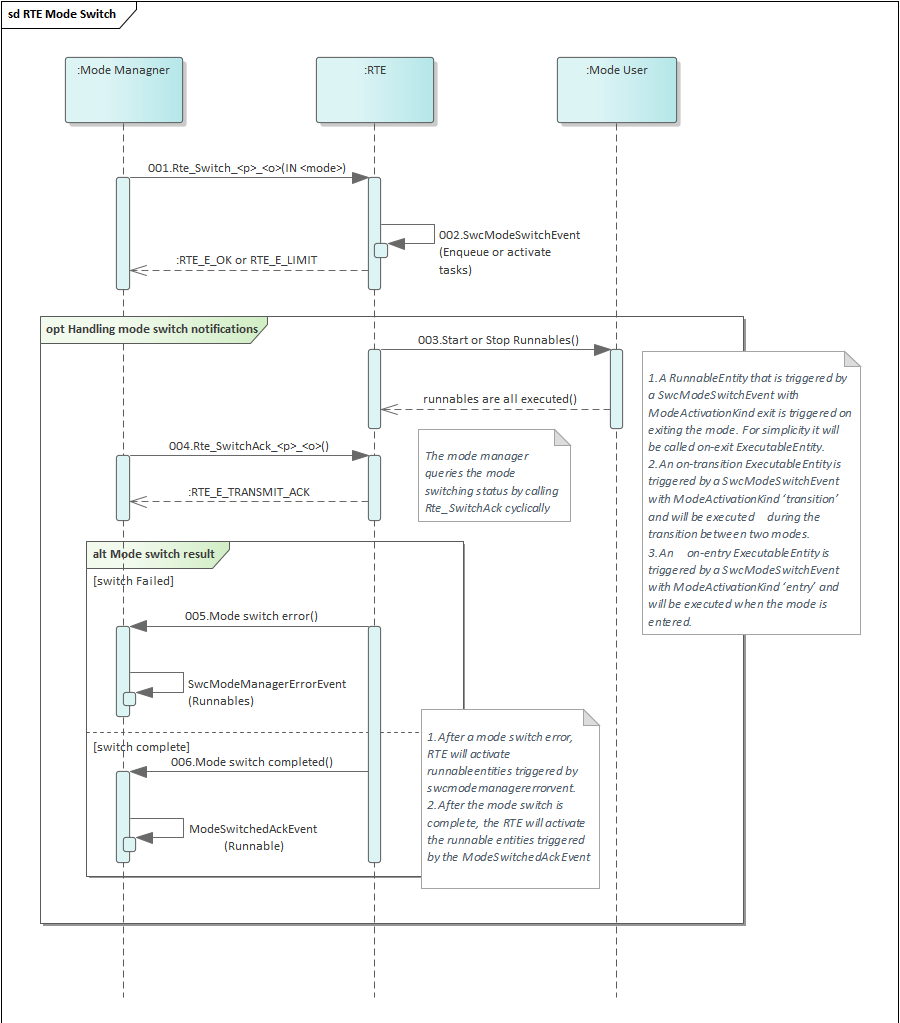


Figure 5‑23模式切换示意图

图示为模式切换示意图。当Mode Managner切换模式时调用Rte\_Switch\_<p>\_<o>()函数，而后RTE会执行响应的模式切换，接着RTE通知模式用户执行响应的Runnables, 在此期间Mode Managner可以调用Rte\_SwitchAck\_<p>\_<o>()函数查询当前模式切换处于什么阶段，当模式切换完成，该函数会返回RTE\_E\_TRANSMIT\_ACK。当模式切换期间，如果发生错误，RTE会触发SwcModeManagerErrorEvent相关联的Runnable。模式切换完成会触发ModeSwitchedAckEvent相关联的Runnable。

[SA\_R2\_133]:

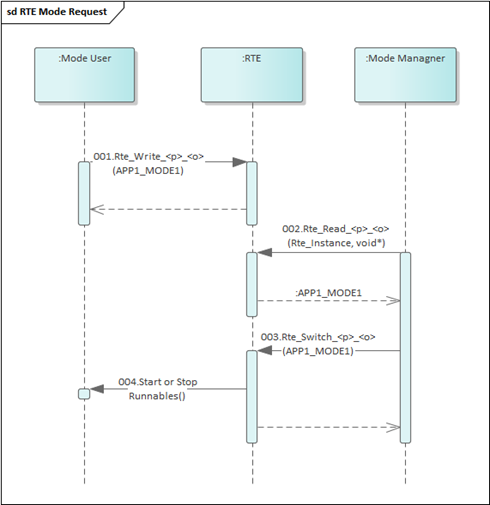


Figure 5‑24模式请求示意图

图示为模式请求示意图。Mode User请求模式APP1\_MODE1时调用Rte\_Write\_<p>\_<o>()函数，Mode Managner调用Rte\_Read\_<p>\_<o>()函数获取到模式APP1\_MODE，然后Mode Managner调用Rte\_Switch\_<p>\_<o>()函数切换模式，RTE收到模式切换信息后通知 Mode User执行相应的Runnables。

[SA\_R2\_134]:

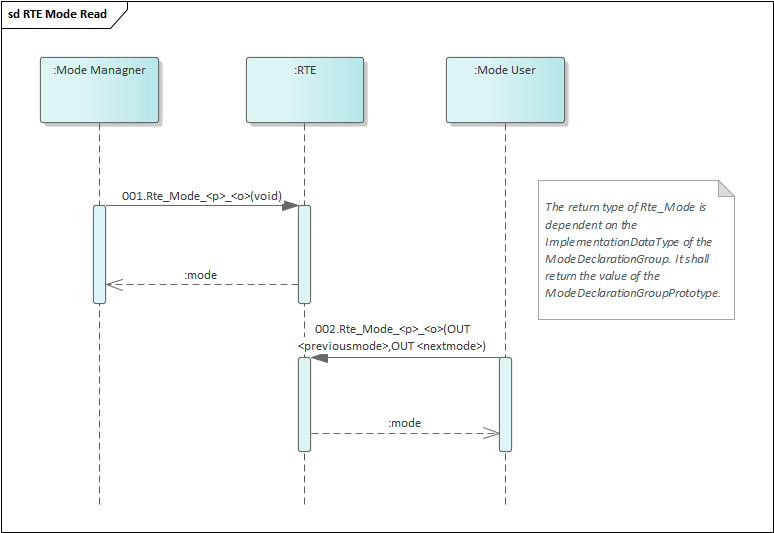


Figure 5‑25模式状态读取示意图

图示为模式状态读取示意图。Mode Managner读取当前模式通过调用函数Rte\_Mode\_<p>\_<o>()，RTE向Mode Managner返回当前模式值。Mode User读取当前模式通过调用增强版函数Rte\_Mode\_<p>\_<o>()(说明：当处于转换中时，该函数会提供上一个模式previousmode，下一个模式nextmode供用户查询)，RTE向Mode User返回previousmode，nextmode以及mode。注意：模式管理者和模式用户都可以使用增强版Rte\_Mode\_<p>\_<o>()。

Compared with normal Rte\_mode, enhanced Rte\_mode additionally provides the values of the previous and the next mode if the mode machine instance is in transition .

[SA\_R2\_135]:

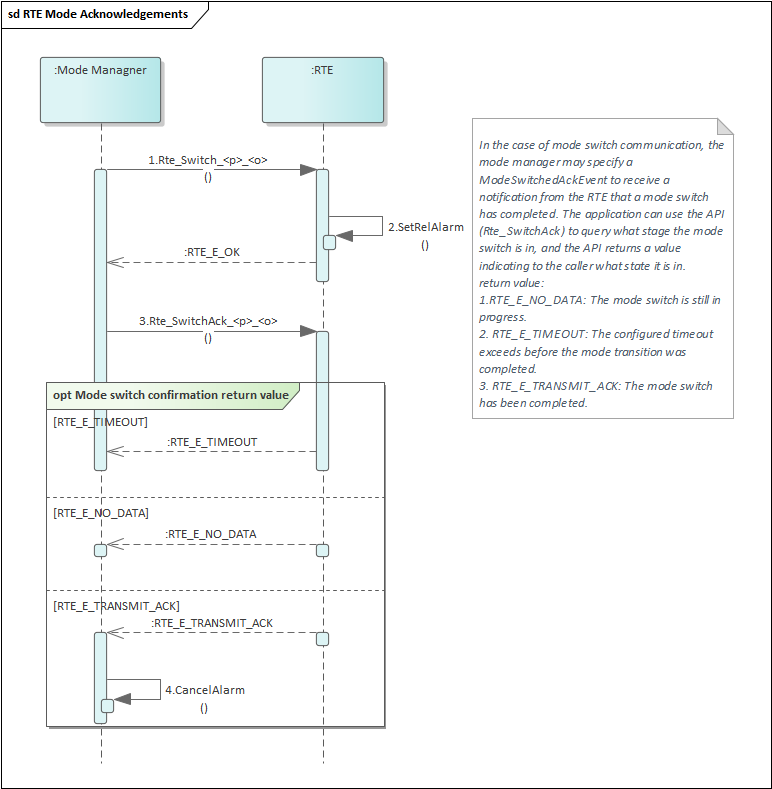


Figure 5‑26模式切换确认示意图

图示为模式切换确认示意图。在Mode Managner在调用Rte\_Switch\_<p>\_<o>函数发起模式切换的时候，RTE会使用OS提供的SetRelAlarm函数启动超时检测，然后Mode Managner使用函数Rte\_SwitchAck\_<p>\_<o>获取当前模式切换处于何种状态中，当RTE向Mode Managner返回RTE\_E\_TIMEOUT时，说明当前模式切换超过用户配置的检测时间；当RTE向Mode Managner返回RTE\_E\_NO\_DATA时，说明当前模式切换仍在进行中；当RTE向Mode Manager返回RTE\_E\_TRANSMIT\_ACK时，说明当前模式切换已经完成。

### Data Flow

[SA\_R2\_136]:

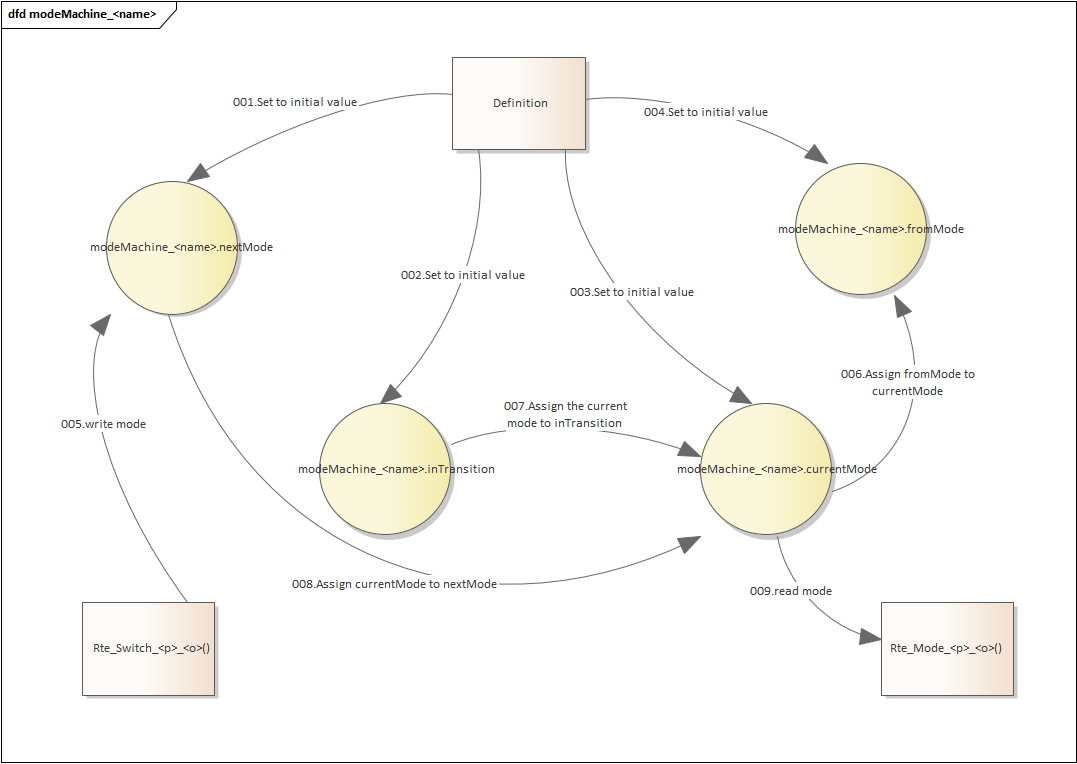


Figure 5‑27 Data Flow Diagaram for modeMachine\_<name>

如图5-21所示，currentMode、nextMode、fromMode、inTransition的初始值通过SWC配置输入。Rte\_Switch\_<p>\_<o>函数可以更改nextMode的值，与此同时，fromMode先被赋值为currentMode，currentMode被赋值为inTransition, 一个模式切换完成后，currentMode被赋值为nextMode。通过函数Rte\_Mode\_<p>\_<o>读取currentMode的值。

### External Interface Definition

#### 5.5.4.1 Rte\_Switch

[SA\_R2\_333]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Switch | | |
| Function prototype: | Std\_ReturnType Rte\_Switch\_<p>\_<o>(IN <mode>) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | mode | Range: | The IN parameter <mode> is passed by value according to the ImplementationDataType on which the ModeDeclarationGroup is mapped. The type name shall be equal to the shortName of the ImplementationDataType. |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType：RTE\_E\_OK RTE\_E\_LIMIT | | |
| Functional Overview: | Initiate a mode switch. The Rte\_Switch API call is used for explicit sending of a mode switch notification. | | |
| Precautions | Note that the mode switch might be discarded when the queue is full and a mode transition is in progress.  Rte\_Switch is restricted to ECU local communication. | | |

#### 5.5.4.2 Rte\_SwitchAck

[SA\_R2\_334]:

|  |  |
| --- | --- |
| Function name: | Rte\_SwitchAck |
| Function prototype: | Std\_ReturnType Rte\_SwitchAck\_<p>\_<o>(void) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | None |
| Return value: | Std\_ReturnType：RTE\_E\_NO\_DATA  RTE\_E\_TIMEOUT RTE\_E\_TRANSMIT\_ACK |
| Functional Overview: | Provide access to mode switch completed acknowledgements and error notifications to mode managers. The Rte\_SwitchAck API takes no parameters other than the instance handle the return value is used to indicate the acknowledgement status to the caller. |
| Precautions | None |

#### 5.5.4.3 Rte\_Mode

[SA\_R2\_335]:

|  |  |
| --- | --- |
| Function name: | Rte\_Mode |
| Function prototype: | <return> Rte\_Mode\_<p>\_<o>(void) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | None |
| Return value: | The return type of Rte\_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the shortName of the ImplementationDataType.  During a transition of the mode machine instance, Rte\_Mode shall return RTE\_TRANSITION\_<ModeDeclarationGroup>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup.  When the mode machine instance is in a defined mode, Rte\_Mode shall return RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration. |
| Functional Overview: | The Rte\_Mode API tells the AUTOSAR software-component which mode of a ModeDeclarationGroup of a given port is currently active. |
| Precautions | The RTE shall support calls of Rte\_Mode after initialization of the Basic Software Scheduler but before the RTE is initialized. |

#### 5.5.4.4 Enhanced Rte\_Mode

[SA\_R2\_336]:

|  |  |
| --- | --- |
| Function name: | Rte\_Mode |
| Function prototype: | <return> Rte\_Mode\_<p>\_<o>(  OUT <previousmode>,  OUT <nextmode>) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | During a transition of a mode machine instance, <previousmode> shall contain the value of the RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration> of the mode being left, <nextmode> shall contain the RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration> of the mode being entered;  When the mode machine instance is in a defined mode, <previousmode> shall contain the value of the RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration>, <nextmode> shall contain the RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration>. |
| Return value: | The return type of Rte\_Mode is dependent on the ImplementationDataType of the ModeDeclarationGroup. It shall return the value of the ModeDeclarationGroupPrototype. The type name shall be equal to the shortName of the ImplementationDataType.  During a transition of the mode machine instance, Rte\_Mode shall return RTE\_TRANSITION\_<ModeDeclarationGroup>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup.  When the mode machine instance is in a defined mode, Rte\_Mode shall return RTE\_MODE\_<ModeDeclarationGroup>\_<ModeDeclaration>, where <ModeDeclarationGroup> is the short name of the ModeDeclarationGroup and <ModeDeclaration> is the short name of the currently active ModeDeclaration. |
| Functional Overview: | The existence of a ModeAccessPoint given that the attribute enhancedModeApi of the ModeSwitchSenderComSpec resp. ModeSwitchReceiverComSpec is set to true shall result in the generation of a Rte\_Mode API.  The Rte\_Mode API tells the AUTOSAR software-component which mode of a ModeDeclarationGroup of a given port is currently active. During mode transitions, i.e. during the execution of runnables that are triggered on exiting one mode or on entering the next mode, overlapping mode disablings of two modes are active. In this case, the Rte\_Mode will return RTE\_TRANSITION\_<ModeDeclarationGroup>. The parameter <previousmode> than contains the mode currently being left,the parameter <nextmode> the mode being entered. |
| Precautions | None |

## Internal/External Trigger

Trigger Events allow software components to request the activation of runnables. They can happen either through port to trigger runnables in other software components, called as an external trigger event, or inside the sofware component to triger its own runnables, called as an internal trigger event. [With the mechanism of the trigger event communication a software component acting as a trigger source is able to request the activation of Runnable Entities of connected trigger sinks.] refers to 'AUTOSAR\_SWS\_RTE' 4.5.1.1

### Function Description

[SA\_R2\_138]: [The RTE shall support the communication of External Trigger events from one trigger source to multiple trigger sinks ("1:n")] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00162]

[SA\_R2\_139]: [The RTE generator shall support invocation of triggered ExecutableEntitys via OS Task.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07213]

[SA\_R2\_140]: [The RTE generator shall support invocation of triggered ExecutableEntitys via direct function call, if no queuing for the trigger source is configured] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07214]

[SA\_R2\_141]: [The RTE shall support InterRunnableTriggering.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00163]

[SA\_R2\_142]: [The RTE shall support for Inter Runnable Triggering that triggered runnables entities are invoked via OS Task activation.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07221]

[SA\_R2\_143]: [The RTE shall support for Inter Runnable Triggering that triggered runnables are invoked via direct function call if no queuing for the InternalTriggeringPointis configured.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07224]

[SA\_R2\_144]: [External and internal trigger event communication shall support queuing the number of triggers issued by a trigger source.]

refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00235]

### Control Flow

[SA\_R2\_145]:

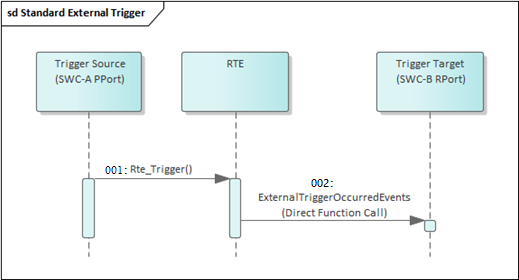


Figure 5‑28 Standard External Trigger Control Flow

As figure 5-28 shows, external trigger happens between two SWCs through a trigger port. RTE is responsible for triggering an event by a direct function call if SwImplPolicy of the trigger is set to standard.

[SA\_R2\_146]:

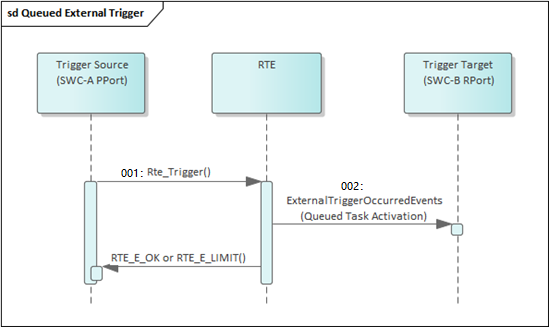


Figure 5‑29 Queued External Trigger Control Flow

As figure 5-29 shows, external trigger happens between two SWCs through a trigger port. RTE is responsible for triggering an event by a queued OS task activation if SwImplPolicy of the trigger is set to queued.

[SA\_R2\_147]:

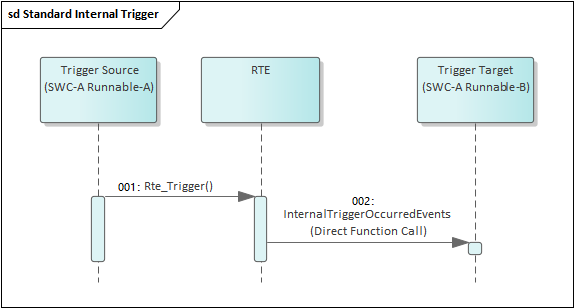


Figure 5‑30 Standard Internal Trigger Control Flow

As figure 5-30 shows, Internal trigger happens between two runnables in the same SWC through internal behaviour. RTE is responsible for triggering an event by a direct function call if SwImplPolicy of the trigger is set to standard.

[SA\_R2\_148]:

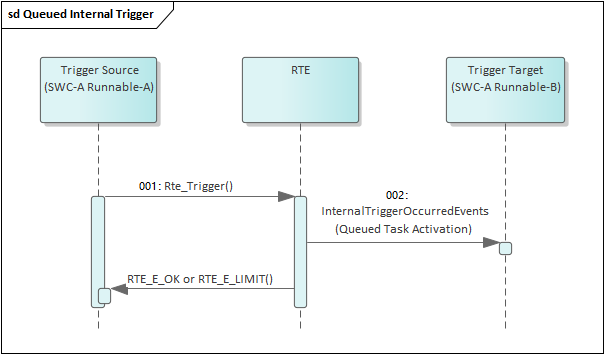


Figure 5‑31 Queued Internal Trigger Control Flow

As figure 5-31 shows, Internal trigger happens between two runnables in the same SWC through internal behaviour. RTE is responsible for triggering an event by a queued OS task activation if SwImplPolicy of the trigger is set to queued.

### Data Flow

N/A

### External Interface Definition

#### Rte\_Trigger

[SA\_R2\_297]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Trigger\_<p>\_<o> | | |
| Function prototype: | signature without queuing support:  void Rte\_Trigger\_<p>\_<o>()  signature with queuing support:  Std\_ReturnType Rte\_Trigger\_<p>\_<o>() | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void/Std\_ReturnType | | |
| Functional Overview: | Raise an external trigger of a trigger port. Where <p> is the port name and <o> the Trigger within the trigger interface categorizing the port. If supporting queuing, return value is neccessary to notify whether reaching queue's limit. [The existence of a ExternalTriggeringPoint shall result in the generation of a Rte\_Trigger API.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07201] | | |
| Precautions | None | | |

[SA\_R2\_299]: Rte\_Trigger should have a return value with Std\_ReturnType when supporting queued external trigger, which shall be RTE\_E\_LIMIT when reaching the max queue length, and be RTE\_E\_OK otherwise.

#### Rte\_IrTrigger

[SA\_R2\_300]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrTrigger\_<p>\_<o> | | |
| Function prototype: | signature without queuing support:  void Rte\_IrTrigger\_<p>\_<o>()  signature with queuing support:  Std\_ReturnType Rte\_IrTrigger\_<p>\_<o>() | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | None | 值域：  Scope： | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void/Std\_ReturnType | | |
| Functional Overview: | Raise an internal trigger to activate Runnable entities of the same software component instance. Where <re> is the name of the runnable entity the API might be used in and <o> is the name of the InternalTriggeringPoint. If supporting queuing, return value is neccessary to notify whether reaching queue's limit. [The existence of a InternalTriggeringPoint shall result in the generation of a Rte\_IrTrigger API.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07204] | | |
| Precautions | None | | |

[SA\_R2\_301]: Rte\_IrTrigger should have a return value with Std\_ReturnType when supporting queued internal trigger, which shall be RTE\_E\_LIMIT when reaching the max queue length, and be RTE\_E\_OK otherwise.

## Internal Behavior

Internal behavior is the core of software components. It is mainly responsible for managing the functionalities of runnables as well as their implementation and relationship with RTE events since every runnable must be triggered by at least one RTE event. It also builds up connection with port prototypes. For instance, if an SWC has an require port with an S-R interface, then it can have a DataReceivedEvent that triggers a runnable which has a read access to the data element of the interface, or Alternatively, it can have a TimingEvent that periodically triggers a runnable to frequently read the value of the data element. Beside that, internal behavior also supports configuring the exclusive area, portAPIOption, dataType mapping and so on for the software component.

### Function Description

[SA\_R2\_150]: [The RTE shall support multiple Runnable Entities in one Software Component type.] refers to 'AUTOSAR\_SRS\_RTE [SRS\_Rte\_00031]

[SA\_R2\_151]: [The RTE shall start/resume a Runnable Entity according to the RTEEvents to which it is linked.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00072]

[SA\_R2\_152]: [The RTE shall allow the configuration of a debounce start time of Runnable Entities to avoid the same Runnable Entity being executed shortly after each other.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00160]

[SA\_R2\_153]: [The RTE shall support one or more mechanism for ensuring data consistency within an Application Software Component instance.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00032]

[SA\_R2\_154]: [The RTE shall support exclusive areas where a Runnable Entity or a Basic Software Schedulable Entity is declared as "running inside" the Exclusive Area] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00046]

[SA\_R2\_155]: [The RTE shall provide an API to access the data consistency mechanism(s).] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00115]

[SA\_R2\_157]: Users shall make sure the application runnable leave the exclusive area in time to avoid long time holding of the resource.

[SA\_R2\_158]: [The RTE shall support InterRunnableVariables. A Software Component shall be able to declare one or more

InterRunnableVariables used for data consistency purposes. An InterRunnableVariable is use when several Runnable Entities of the same Software Component instance access the same data item. InterRunnableVariables are used to store data item copies to avoid concurrent Runnable Entity accesses to the one original data item.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00142]

[SA\_R2\_160]: The read and write of interRunnableVariables can be done both explicitly or implicitly. Data is directly written or read if it is explicit; while IRV data is copied to a temporary variable to be read before runnable starts, and the IRV data is written from the temporary variable to IRV data after the runnable terminates if it is implicit..

[SA\_R2\_161]: [The RTE shall support the time recurrent activation of Runnable Entities. The applicable time period shall be definable by the software component type and be overwriteable per component instance.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00237]

[SA\_R2\_162]: [The mechanism of "port-defined argument values", as defined in the AUTOSAR Software Component Template , has to be supported.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00152]

[SA\_R2\_163]: [It is possible to request the activation of one Executable Entity by several Events. It shall be possible to identify the activating Event during the execution of the activated Executable Entity.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00238]

[SA\_R2\_164]: The RTE shall support automatic configuration of RteSwComponentInstance and related Os Task based on time period of timing events.

[SA\_R2\_165]: The RTE shall support graphical user interface which allows users to map RTE runnables in software components to related os tasks.

[SA\_R2\_166]: The RTE shall support automatic configuration of Os events and extended Os tasks to realize task sleep and wakeup so as to support queued synchronous C-S server calls, asynchronous C-S server calls and implementation of waitpoints.

[SA\_R2\_167]: The RTE shall provide function signature and declaration of runnables in software components in application header file.

[SA\_R2\_168]: The RTE shall generate correct function signature and declaration based on the configuration of C-S interface such as number and type of parameters, type of return values, etc.

[SA\_R2\_170]: [The RTE shall provide per-instance memory to Application Software Components, where each Application Software Component instance has its own copy of memory, not shared with other instances of the same Application Software Component type.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00013]

[SA\_R2\_171]: [The RTE generator shall instantiate each per-instance memory section of a software component according to the attributes given in its software component description.] refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00077]

[SA\_R2\_172]: [The RTE shall support the Runnable Entity categories 1a, 1b and 2 Runnable Entity category:

• 1a) The Runnable Entity is only allowed to use implicit reading (DataReadAccess) and writing (DataWriteAcess). A category 1a Runnable Entity cannot block and cannot use explicit read/write.

• 1b) The Runnable Entity can use explicit reading and writing (DataReadAccess). A category 1b Runnable Entity cannot block. Implicit read/write is also allowed.

• 2) The Runnable Entity may use explicit reading/writing including blocking behavior.]

refers to 'AUTOSAR\_SRS\_RTE' [SRS\_Rte\_00134]

[SA\_R2\_173]: Users shall make sure the application runnable finishes on time to avoid infinite blocking.

[SA\_R2\_354]: RTE shall timeout monitoring for waitpoints.

### Control Flow

[SA\_R2\_174]:

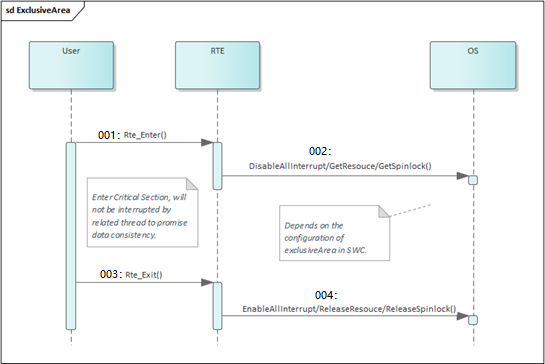


Figure 5‑32 ExclusiveArea Sequence Diagram

As figure 5-25 shows, after calling Rte\_Enter, Rte will call Os interface to enter critical section, there are three ways: disabling interrupt, using resource, and using spinlock, which depends on the actual configuration. The user remains in the exclusive area until calling Rte\_Exit, which will cause Rte to leave the critical section also by calling related Os interfaces.

[SA\_R2\_175]:

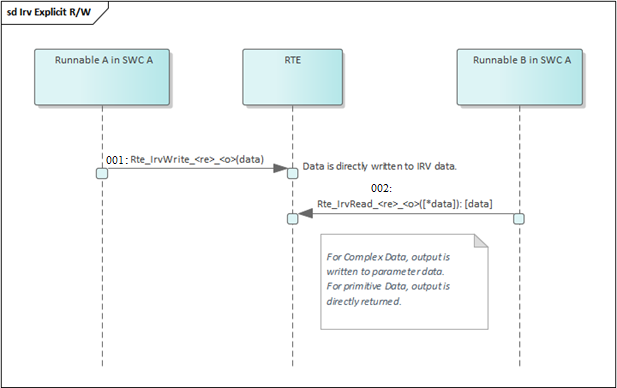


Figure 5‑33 Explicit IRV R/W Sequence Diagram

As figure 5-26 shows, IRV communication occurs between two runnables in the same SWC. After runnable A calling Rte\_IrvWrite, data is directly written to IRV. Then runnable B calls Rte\_IrvRead to read the updated value of the IRV.

[SA\_R2\_176]:

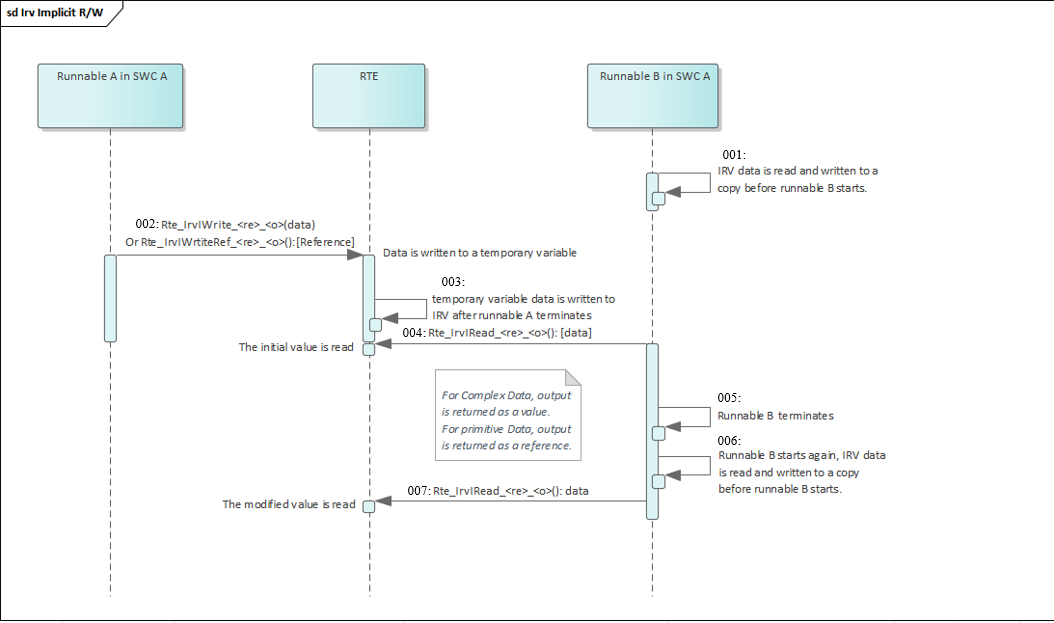


Figure 5‑34 Implicit IRV R/W Sequence Diagram

As figure 5-27 shows, IRV communication occurs between two runnables in the same SWC. Runnable B starts first and IRV data is read to a temporary variable. Then runnable A starts calls Rte\_IrvIWrite or Rte\_IrvIWriteRef to write using reference, data is directly written to a temporary variable. Then runnable B calls Rte\_IrvIRead, the unmodified value is read since runnable B starts before runnable A and a copy of IRV is read which remains unmodified before runnable B terminates. After that, runnable B terminates and restarts and calls Rte\_IrvIRead again, now the modified value is read since runnable A has already terminated as temporary variable data is written to global IRV now.

### Data Flow

[SA\_R2\_177]:

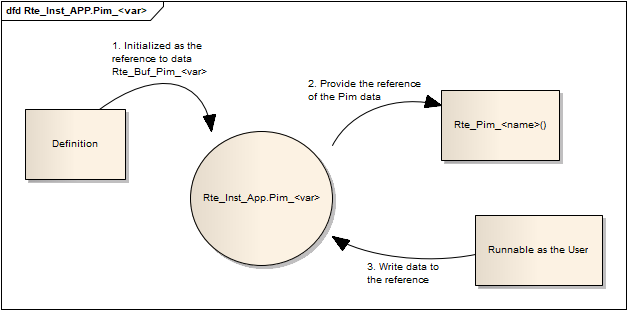


Figure 5‑35 Data Flow Diagram for Per Instance Memory

As figure 5-35 shows, Per Instance Memory Section is initialized to be a pointer of the data buffer which has length based on the SWC configuration input. After calling Rte\_Pim, this reference is provided to the application and can be written and read by the user runnable.

[SA\_R2\_178]:

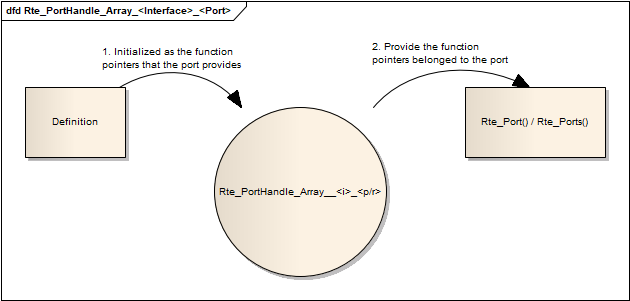


Figure 5‑36 Data Flow Diagram for Port Handle Section

As figure 5-36 shows, Port Handle Section is initialized to be an array of function pointers that reference to the port interfaces, for example, Rte\_Write for SR Provided port. After calling Rte\_Port, this reference is provided to the application and user runnable can indiretcly call port interface API through this reference array.

### External Interface Definition

#### <runnableEntity>

[SA\_R2\_347]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | <runnableEntity> | | |
| Function prototype: | [<void|Std\_ReturnType> <name>(  [IN Rte\_Instance <instance>],  [IN Rte\_ActivatingEvent\_<name> <activation>])]  refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01126] | | |
| Synchronous/Asynchronous: | Based on configuration | | |
| Is it reentrant: | Based on configuration | | |
| Input parameters: | <instance> | Range: | *--* |
| <activation> | *--* |
| Based on configuration | *--* |
| Input and output parameters: | Based on configuration | | |
| Output parameters: | Based on configuration | | |
| Return value: | void/Std\_ReturnType | | |
| Functional Overview: | The definition of all runnable entities, whatever the RTEEvent that triggers their execution, follows the same basic form. Where <name> is the symbol describing the runnable’s entry point. Parameter <instance> exists if supporting multiple instantiation. The usage of Rte\_ActivatingEvent is optional and defined if the provide activating event feature is enabled. Return value Type depends on whether the C-S interface has possible error or not. The interface exits for every runnable. | | |
| Precautions: | None | | |

#### Rte\_Enter

[SA\_R2\_302]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Enter\_<EA> | | |
| Function prototype: | void Rte\_Enter\_<EA>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void | | |
| Functional Overview: | Enter an exclusive area, where <name> is the exclusive area name. [An Rte\_Enter API shall be created for each ExclusiveArea that is declared and which has an canEnterExclusiveArea association.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01307] | | |
| Precautions: | None | | |

[SA\_R2\_156]: [The Rte\_Enter and Rte\_Exit API may only be called nested if different exclusive areas are invoked; in this case exclusive areas shall exited in the reverse order they were entered]. refers to 'AUTOSAR\_SWS\_RTE' [constr\_9029]

#### Rte\_Exit

[SA\_R2\_303]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Exit\_<EA> | | |
| Function prototype: | void Rte\_Exit\_<EA>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void | | |
| Functional Overview: | Leave an exclusive area, where <name> is the exclusive area name. [An Rte\_Exit API shall be created for each ExclusiveArea that is declared and which has an canEnterExclusiveArea association.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01308] | | |
| Precautions: | None | | |

#### Rte\_IrvIread

[SA\_R2\_304]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrvIread\_<re>\_<o> | | |
| Function prototype: | <return> Rte\_IrvIRead\_<re>\_<o>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: |  |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | <return>: provide access to the data value of the IRV, its type denpends on the implementationDataType of the IRV. | | |
| Functional Overview: | Provide read access to the InterRunnableVariables with implicit behavior of an AUTOSAR SW-C, where <re> is the name of the runnable entity the API might be used in, <o> is the name of the VariableDataPrototype in role implicitInterRunnableVariable. Parameter <instance> exists if supporting multiple instantiation. <return> value provide access to the data value of the IRV, its type denpends on the implementationDataType of the IRV. [An Rte\_IrvIRead API shall be created for each VariableAccess in role readLocalVariable to an implicitInterRunnableVariable.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01303] | | |
| Precautions: | None | | |

[SA\_R2\_159]: User application and Rte shall check the validity of the pointer to data array when using interface to read or write

InterRunnableVariables and correctly use the pointer based on the data array length.

#### Rte\_IrvIWrite

[SA\_R2\_305]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrvIWrite\_<re>\_<o> | | |
| Function prototype: | void Rte\_IrvIWrite\_<re>\_<o>([IN <data>]) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | <data>:written to the data value of the IRV, its type denpends on the implementationDataType of the IRV. | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | void | | |
| Functional Overview: | Provide write access to the InterRunnableVariables with implicit behavior of an AUTOSAR SW-C, where <re> is the name of the runnable entity the API might be used in, <o> is the name of the VariableDataPrototype in role implicitInterRunnableVariable. Parameter <instance> exists if supporting multiple instantiation. Parameter <data> value is written to the data value of the IRV, its type denpends on the implementationDataType of the IRV. [An Rte\_IrvIWrite API shall be created for each VariableAccess in role writtenLocalVariable to an implicitInterRunnableVariable.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01304] | | |
| Precautions: | None | | |

#### Rte\_IrvIWriteRef

[SA\_R2\_306]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrvIWriteRef\_<re>\_<o> | | |
| Function prototype: | <return reference> Rte\_IrvIWrite\_<re>\_<o>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | <return reference>: the pointer to the data value of the IRV, its type denpends on the implementationDataType of the IRV. | | |
| Functional Overview: | Provide a reference to the VariableDataPrototype defined with the implicitInterRunnableVariable role referenced by a VariableAccess in the writtenLocalVariable role, where <re> is the name of the runnable entity the API might be used in, <o> is the name of the VariableDataPrototype in role implicitInterRunnableVariable. Parameter <instance> exists if supporting multiple instantiation. <return reference> is the pointer to the data value of the IRV, its type denpends on the implementationDataType of the IRV. [An Rte\_IrvIWriteRef API shall be created for each VariableAccess in role writtenLocalVariable to an implicitInterRunnableVariable.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_06208] | | |
| Precautions: | None | | |

#### Rte\_IrvRead

[SA\_R2\_308]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrvRead\_<re>\_<o> | | |
| Function prototype: | <return> Rte\_IrvRead\_<re>\_<o>(void)  or for complex data type:  void Rte\_IrvRead\_<re>\_<o>(OUT <data>) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | [<data>: the data reference pointing to the data] exists only for complex data type. | | |
| Return value: | void/<return>:the return value which represents the data value of the IRV | | |
| Functional Overview: | Provide read access to the InterRunnableVariables with explicit behavior of an AUTOSAR SW-C, where <re> is the name of the runnable entity the API might be used in, <o> is the name of the VariableDataPrototype in role implicitInterRunnableVariable. Parameter <instance> exists if supporting multiple instantiation. For primitive IRV data type, <return> is the return value which represents the data value of the IRV. And for complex IRV data type such as struct, the value of the data is provided as an out parameter <data>. [An Rte\_IrvRead API shall be created for each read InterRunnableVariable using explicit access.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01305] | | |
| Precautions: | None | | |

[SA\_R2\_309]: Rte\_IrvRead directly return the value of InterRunnableVariables as the return value for primitive data type and output value as the output parameter for complex data types.

#### Rte\_IrvWrite

[SA\_R2\_310]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_IrvWrite\_<re>\_<o> | | |
| Function prototype: | void Rte\_IrvWrite\_<re>\_<o>(IN <data>) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | <data>:written to the data value of the IRV, its type denpends on the implementationDataType of the IRV. | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | None | | |
| Functional Overview: | Provide write access to the InterRunnableVariables with explicit behavior of an AUTOSAR SW-C, where <re> is the name of the runnable entity the API might be used in, <o> is the name of the VariableDataPrototype in role implicitInterRunnableVariable. Parameter <instance> exists if supporting multiple instantiation. The IN parameter <data> is passed by value or reference according to whether ImplementationDataType is primitive or complex. [An Rte\_IrvWrite API shall be created for each written InterRunnableVariable using explicit access.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01306] | | |
| Precautions: | None | | |

#### Rte\_Ports

[SA\_R2\_311]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Ports\_<i>\_<R/P> | | |
| Function prototype: | Rte\_PortHandle\_<i>\_<R/P> Rte\_Ports\_<i>\_<R/P>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Rte\_PortHandle\_<i>\_<R/P> | | |
| Functional Overview: | Provide an array of the ports of a given interface type and a given provide / require usage that can be accessed by the indirect API, Where here <i> is the port interface name and ‘P’ or ‘R’ are literals to indicate provide or require ports respectively. [An Rte\_Ports API shall be created for each interface type and usage by a port in at least one PreCompileTime variant when the indirectAPI attribute of that port is set to true.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02613] | | |
| Precautions: | None | | |

[SA\_R2\_312]:Rte\_Ports return an array of the ports, user application shall respect the length of the array when use it to avoid index out of bound, Rte\_NPorts provide the length of the array.

#### Rte\_NPorts

[SA\_R2\_314]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_NPorts\_<i>\_<R/P> | | |
| Function prototype: | uint8 Rte\_NPorts\_<i>\_<R/P>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | uint8 | | |
| Functional Overview: | Provide the number of ports of a given interface type and provide / require usage that can be accessed through the indirect API. Where here <i> is the port interface name and ‘P’ or ‘R’ are literals to indicate provide or require ports respectively. [An Rte\_NPorts API shall be created for each interface type and usage by a port in at least one PreCompileTime variant when the indirectAPI attribute of the port is set to true.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02615] | | |
| Precautions: | None | | |

#### Rte\_Port

[SA\_R2\_315]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Port\_<p> | | |
| Function prototype: | Rte\_PortHandle\_<i>\_<R/P> Rte\_Port\_<p>(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Rte\_PortHandle\_<i>\_<R/P> | | |
| Functional Overview: | Provide access to the port data structure for a single port of a particular software component instance. This allows a software component to extract a sub-group of ports characterized by the same interface in order to iterate over this sub-group. where <i> is the port interface name and <p> is the name of the port. [An Rte\_Port API shall be created for each port of an AUTOSAR SW-C, for which the indirectAPI attribute is set to true.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_01355] | | |
| Precautions: | None | | |

[SA\_R2\_316]:Rte\_Port return a pointer to the interface of a single port, user application shall directly access to the reference for the function and treat it as an array of size 1 to avoid index out of bound.

## Data Transformation

Transformers enable AUTOSAR systems to use a data transformation mechanism to linearize and transform data. They can be concatenated to transformer chains and are executed by the RTE for inter-ECU communication which is configured to be transformed. The input of the first transformer in the chain gets the data from the RTE. Each following transformer uses the output of the preceding transformer as input. All transformers following the first one then have a generic signature with just a byte array as IN and OUT parameter. Such an architecture could be used to design systems, where you can flexibly add functionality like safety or security protection to a serialized stream. The execution of the transformers and the necessary buffer handling is coordinated by the RTE.

### Function Description

[SA\_R2\_180]:[ The E2E transformer defined in this document shall be used as a transformer if

1. the attribute protocol of the TransformationTechnology is set to E2E

2. and the attribute version of the TransformationTechnology is set to 1.0.0

3. and the attribute transformerClass of the TransformationTechnology is set to safety] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00161]

[SA\_R2\_181]: [The configuration options in EndToEndTransformationComSpecProps shall have precedence over the options in EndToEndTransformationDescription and EndToEndTransformationISignalProps.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00134]

[SA\_R2\_182]: [If configuration option EndToEndTransformationComSpecProps.disableEndToEndCheck is set for a given <transformerId>, then E2E Transformer shall skip the invocation of the E2E Library – it shall only perform buffer processing (e.g. copying from inputBuffer to buffer). Return value shall be E\_OK.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00154]

[SA\_R2\_183]: [The E2E Transformer shall generate the following data structure, to store the configuration of E2E Transformer module:

E2EXf\_ConfigStruct\_<v>(of type E2EXf\_ConfigType)]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00011]

[SA\_R2\_184]: [The E2E Transformer shall derive the required number of independent state data resources of types E2E\_PXXProtectStateType, E2E\_PXXCheckStateType, and E2E\_SMCheckStateType to perform E2E Protection within the E2E Transformer module from the number of E2E-protected data uniquely identified with <transformerId>, protected by profile XX.]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00125]

[SA\_R2\_185]: [The E2E Transformer shall derive the required number of independent statically initialized configuration objects of types E2E\_PXXConfigType and E2E\_SMConfigType to perform E2E Protection within the E2E Transformer, from: 1. the number of E2E-protected data uniquely identified with <transformerId>, protected by profile XX, and 2. the number of configuration variants.]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00126]

[SA\_R2\_186]: [In post-build-selectable variant, E2EXf\_Init() shall check that Config pointer (received as function parameter) points to one of the configuration variants E2EXf\_ConfigStruct\_<v>.If it is equal, then E2EXf\_Init() shall select the passed configuration variant, and it shall set the module initialization state to TRUE.]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00024]

[SA\_R2\_187]: [The E2EXf\_Init() function shall initialize all external state data resources managed by E2E transformer (see SWS\_E2EXf\_00125) as follows:

- Initialization of state data resources of type E2E\_PXXProtectStateType by

calling corresponding E2E\_PXXProtectInit() methods,

- Initialization of state data resources of type E2E\_PXXCheckStateType by

calling corresponding E2E\_PXXCheckInit() methods,

- Initialization of state data resources of type E2E\_SMCheckStateType by

calling corresponding E2E\_SMCheckInit() methods.]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00021]

[SA\_R2\_188]: [The E2EXf\_Init() function shall initialize all internal state data resources of E2E transformer.]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00158]

[SA\_R2\_189]: [In case of post-build configuration, E2EXf\_Init() function shall store the information about the selected configuration.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00159]

[SA\_R2\_190]: [The E2E Transformer shall maintain a boolean information (Initialization state) that is only set to TRUE, if the module has been successfully initialized via a call to E2EXf\_Init(). Otherwise, it is set to FALSE. ]refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00130]

[SA\_R2\_191]: [In case of deinitialization (invocation of E2EXf\_DeInit()), the module initialization state shall be set to FALSE.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00132]

[SA\_R2\_192]: [If the E2E Transformer has not been correctly initialized (which means that E2EXf\_Init() was not successfully called before), then all generated E2E Transformer APIs shall immediately return E\_SAFETY\_HARD\_RUNTIMEERROR.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00133]

[SA\_R2\_193]: [E2EXf\_DeInit() shall check shall set the module initialization state to FALSE.] refers to 'AUTOSAR\_SWS\_E2ETransformer' [SWS\_E2EXf\_00148]

[SA\_R2\_194]: E2E Transformer functions work using in-place processing or out-of-place processing. This is configured by binary setting BufferProperties.inPlace. In-place means that one buffer is used by a transformer both as input and as output. In-place processing has a performance advantage (less copying, less buffers).Out-ofplace means that there is one input buffer and a separate output buffer.

[SA\_R2\_195]: The data protection function is mainly achieved by calling E2EXf\_<transformerId>(). If the initialization state is True and the input buffer is not empty and the length of the input data is correct, E2E\_PXXProtect() in E2EL will be called to calculate the CRC check value according to the corresponding algorithm in the call CRC. Add the E2E header containing counter and CRC to the data and print it.

[SA\_R2\_196]: Data verification is performed by calling E2EXf\_inv\_<transformerId>(). If the initialization state is True, and the input buffer is not empty and the input data length is correct, E2E\_PXXCheck() in E2EL will be called to extract the counter and CRC of E2E header, and the CRC check value will be calculated according to the corresponding algorithm in calling CRC. Evaluate counter and CRC received, judge verification result, judge state machine result based on current and historical verification result, and return the result to the user.

[SA\_R2\_197]: The E2EXf module supports development error detection by calling the Det module interface.

[SA\_R2\_198]: The ComXf transformer shall only be used as the topmost transformer (first) in a transformer chain. The ComXf transformer serializes structured data into a linear form. Therefore it can only be used as the first transformer on the sending side and the last transformer on the receiving side.

[SA\_R2\_199]: ComXf data element serialization converts input data elements into uint8-byte aligned serialized data (conversion rules follow the configuration of Signal Group Array in COM).

[SA\_R2\_200]: ComXf deserialization of data elements converts input Uint8-byte aligned data into the original data element (conversion rules follow the configuration of Signal Group Array in COM) and passes it to the RTE layer.

[SA\_R2\_201]: The ComXf module supports development error detection by calling the Det module interface.

[SA\_R2\_355]: The RTE shall evaluate the return codes of transformers. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08539].

[SA\_R2\_356]: The RTE shall continue with the execution of a transformer chain if a transformer returns a soft error. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08540].

[SA\_R2\_357]: If a transformer which doesn t transform the request of a client server communication on the server side (i.e., a transformer that either transforms the request of a client server communication on the client side or transforms the response of a client server communication or transforms an sender receiver communication) returns a hard error, the Rte shall notify this hard error to the runnable which called the RTE API that triggered the transformer execution. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08558].

[SA\_R2\_358]: If no transformer in the transformer chain returned a hard error and at least one transformer returned a soft error, the Rte shall notify the first soft error (in transformer execution order) to the SWC. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08559].

[SA\_R2\_359]: For VariableAccesses in the dataReadAccess role the RTE shall execute data transformation after reception of the data from the Com stack and before start of the runnable/coherency group. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08570].

[SA\_R2\_360]: For VariableAccesses in the dataWriteAccess role the RTE shall execute data transformation after termination of the runnable/coherency group and before handing the data over to the Com stack. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08571].

[SA\_R2\_361]: The RTE on sender side shall execute the transformers of the chain in order. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08799].

[SA\_R2\_362]: The RTE on receiver side shall execute the retransformers of the chain in reverse order. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_08588].

### Control Flow

[SA\_R2\_202]:

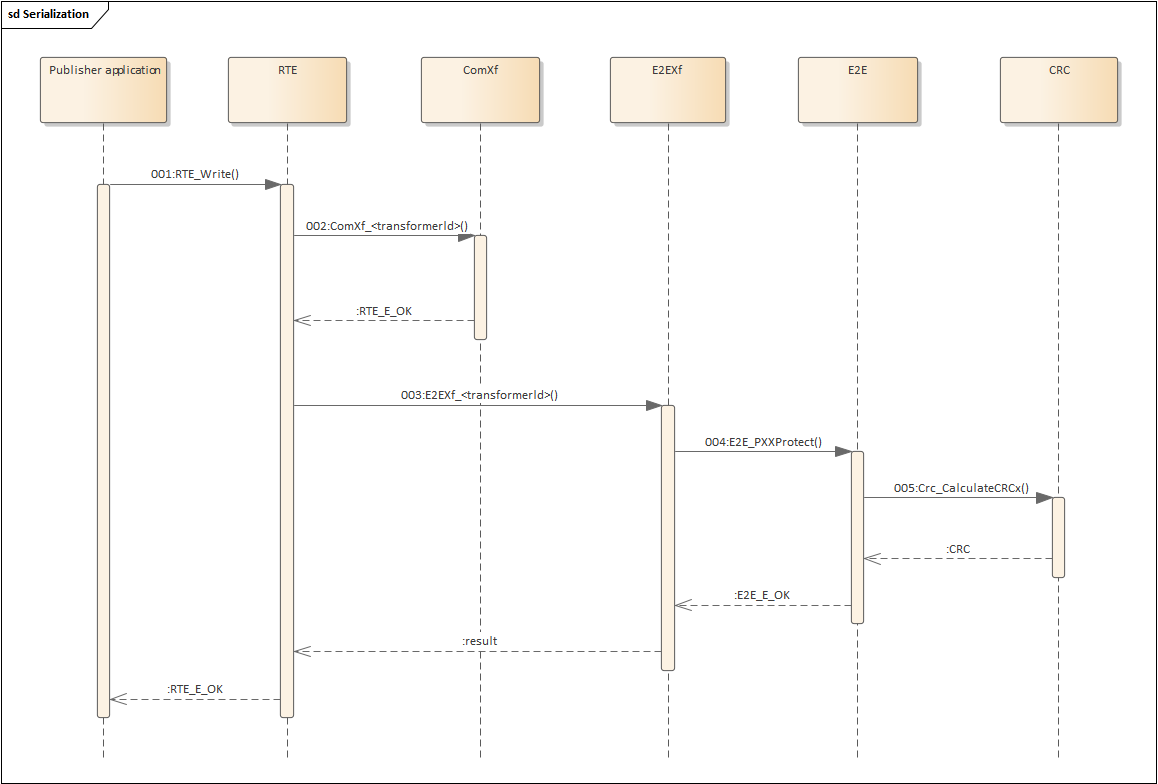


Figure 5‑37 E2EXf Data protection Control Flow

Step 1: Rte\_Write is called when the Publisher application sends a signal.

Step 2: If a signal group is configured, the user needs to call ComXf\_<transformerId> to serialize the signal group.

Step 3: If E2E data protection is required, user shall call E2EXf\_<transformerId> to protect data by E2EXf with needing to know E2E internal details.

Step 4: E2EXf prepares input parameters such as Config and State for E2E\_PXXProtect and calls E2E\_PXXProtect.

Step 5: E2EL invokes its corresponding CRC algorithm.

[SA\_R2\_203]:

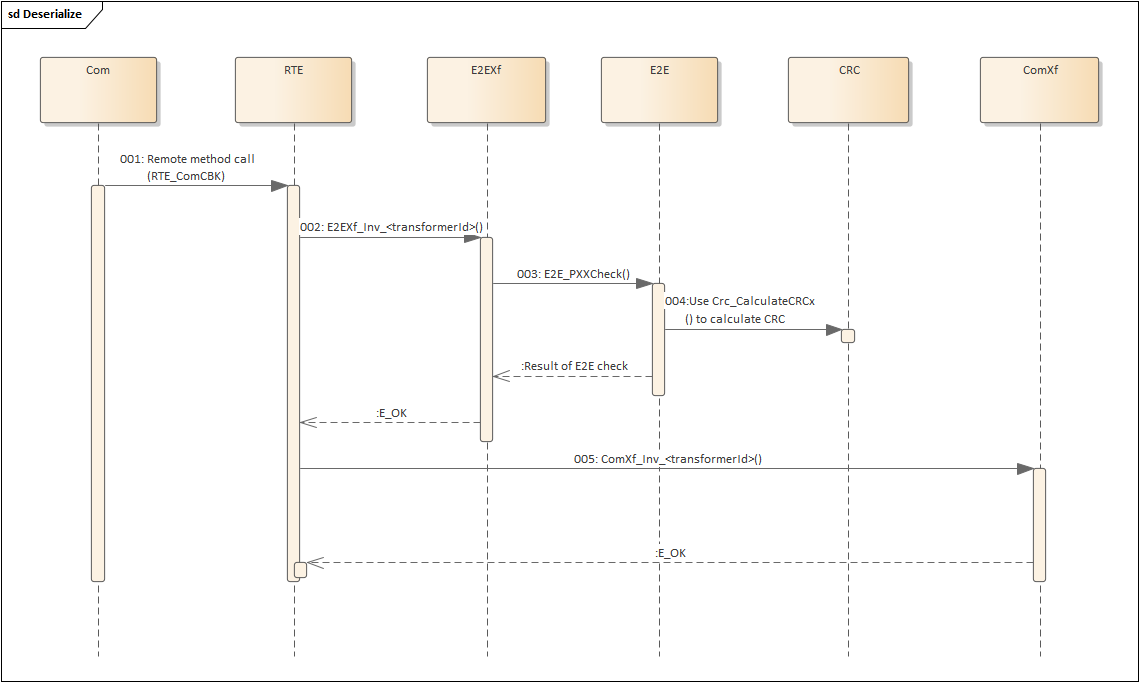


Figure 5‑38 E2EXf Data check Control Flow

Step 1: When the receiver receives the signal, it will call Rte\_ComCBK and hand over the received data to RTE for processing.

Step 2: If E2E data check is required, user shall call E2EXf\_Inv\_<transformerId> to check data by E2EXf with needing to know E2E internal details.

Step 3: E2EXf prepares input parameters such as Config and State for E2E\_PXXCheck and calls E2E\_PXXCheck.

Step 4: E2EL tests CRC results.

Step 5: Get actual data by calling ComXf\_Inv\_<transformerId>.

### Data Flow

[SA\_R2\_282]:

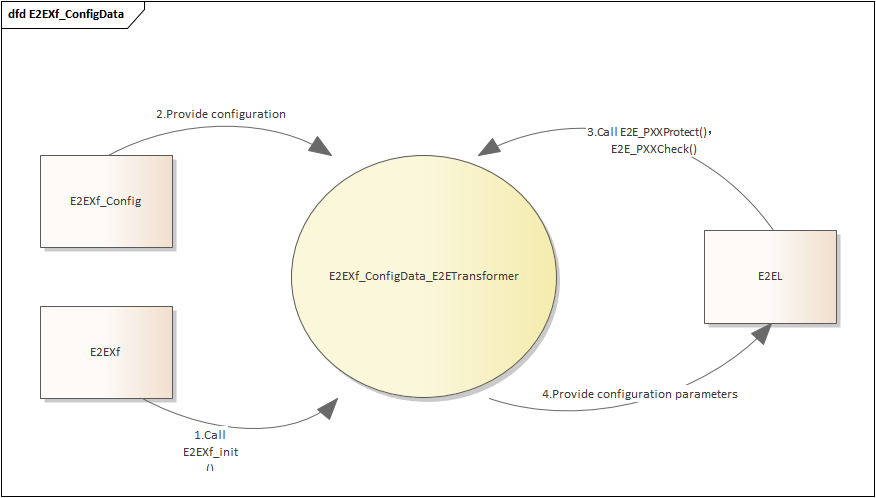


Figure 5‑39 Data Flow Diagram for E2EXf\_ConfigData

As Figure 5‑39 shows, E2EXf\_Config provides configuration for E2EXf, E2EXf calls E2EXf\_Init to provide configuration parameters for E2EL, E2EL calls E2E\_PXXProtect and E2E\_PXXCheck uses the configuration provided by E2EXf.

[SA\_R2\_283]:

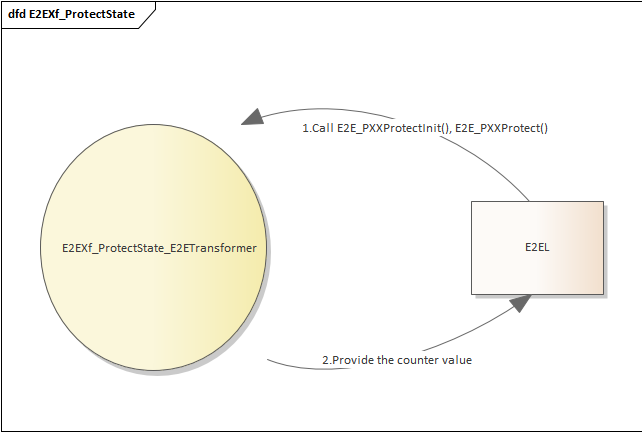


Figure 5‑40 Data Flow Diagram for E2EXf\_ProtectState

As Figure 5‑40 shows, E2EL calls E2E\_PXXProtectInit and E2E\_PXXProtect writes E2EXf\_ProtectState\_E2ETransformer, which provides count value for E2EL.

[SA\_R2\_284]:

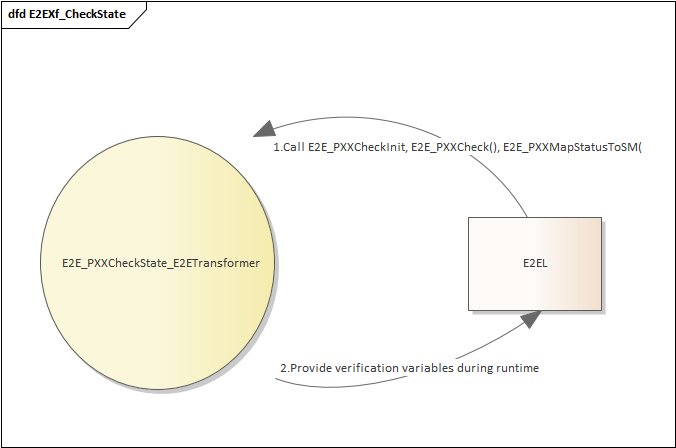


Figure 5‑41 Data Flow Diagram for E2EXf\_CheckState

As Figure 5‑41 shows, E2EL calls E2E\_PXXCheckInit and E2E\_PXXCheck writes E2EXf\_CheckState\_E2ETransformer, E2EL calls E2E\_PXXMapStatusToSM to determine the current state.

[SA\_R2\_285]:

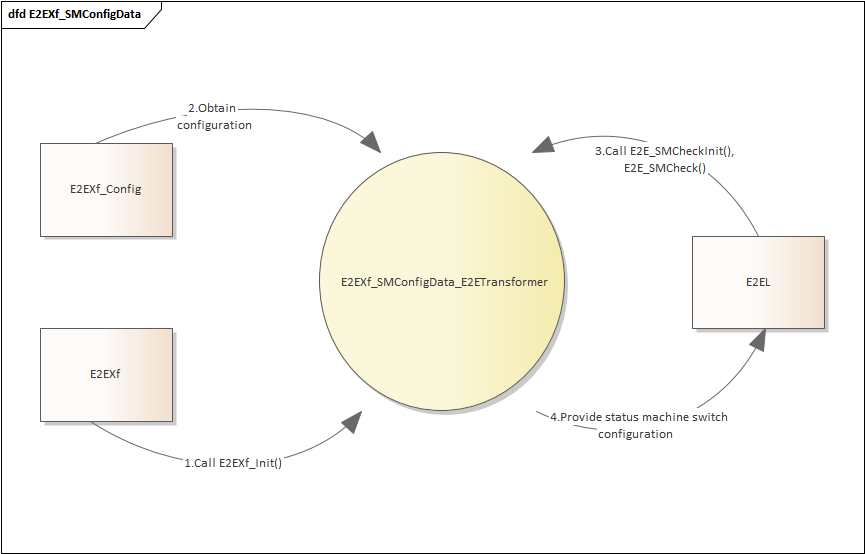


Figure 5‑42 Data Flow Diagram for E2EXf\_SMConfigData

As Figure 5‑42 shows，E2EXf calls E2EXf\_Init to provide configuration for E2E\_SMCheckInit, and E2EL calls E2E\_SMCheck to obtain state machine transition configuration

[SA\_R2\_286]:

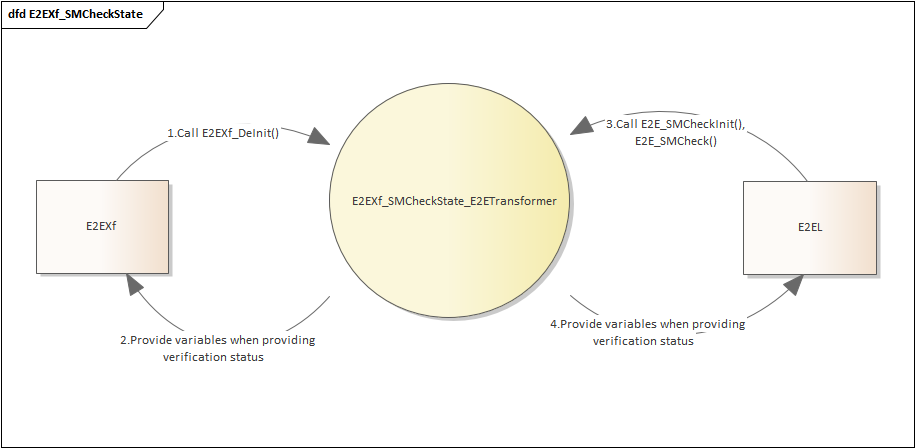


Figure 5‑43 Data Flow Diagram for E2EXf\_SMCheckState

As Figure 5‑43 shows，E2EXf calls E2EXf\_DeInit to set E2EXf\_SMCheckState\_E2ETransformer.SMState to E2E\_SM\_DEINIT, E2EXf\_SMCheckState\_E2ETransformer provides verification time variables for E2EXf, and E2EL writes and reads the verification state variables by calling E2E\_SMCheckInit and E2E\_SMCheck.

[SA\_R2\_287]:

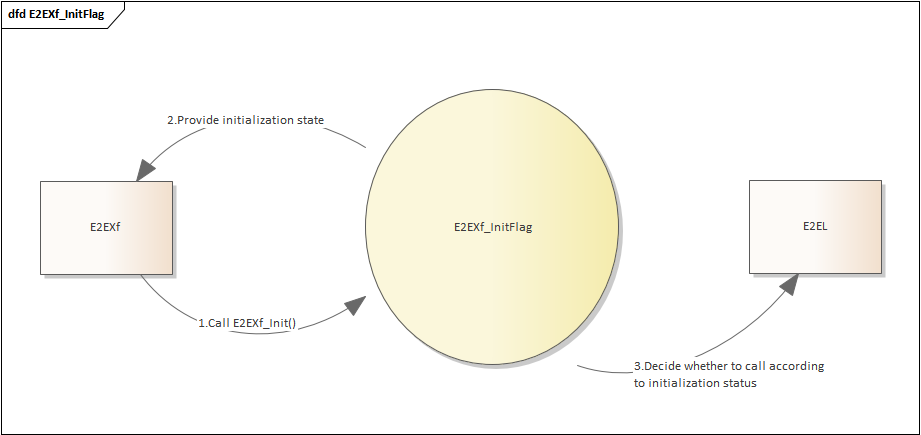


Figure 5‑44 Data Flow Diagram for E2EXf\_InitFlag

As Figure 5‑44 shows, E2EXf calls E2EXf\_Init to set E2EXf\_InitFlag to 1, E2EXf judges the initialization state by reading E2EXf\_InitFlag, and judges whether to call E2EL according to the initialization state.

[SA\_R2\_288]:

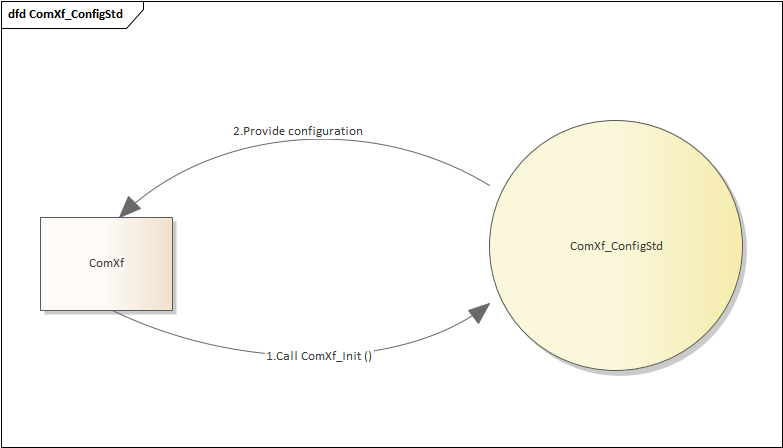


Figure 5‑45 Data Flow Diagram for ComXf\_ConfigStd

As Figure 5‑45 shows, ComXf calls ComXf\_Init to initialize ComXf\_ConfigStd, and ComXf\_ConfigStd provides configuration for ComXf.

[SA\_R2\_289]:

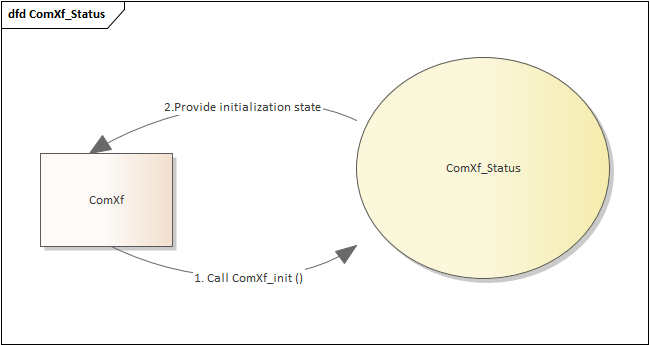


Figure 5‑46 Data Flow Diagram for ComXf\_Status

As Figure 5‑46 shows, ComXf calls ComXf\_Init to initialize ComXf\_Status, and ComXf\_Status provides ComXf with the initialization status.

### External Interface Definition

#### 5.8.4.1 E2EXf\_<transformerId>

[SA\_R2\_337]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | E2EXf\_<transformerId> | | |
| Function prototype: | uint8 E2EXf\_<transformerId> (  uint8\* buffer,  uint32\* bufferLength,  [const uint8\* inputBuffer],  uint32 inputBufferLength  ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | inputBuffer | Range: | Determine if it is a null pointer |
| inputBufferLength | Range: | Determine whether the length is less than EndToEndTransformationDescription.upperHeaderBitsToShift/8u |
| Input and output parameters: | buffer: | Range: | Determine if it is a null pointer |
| Output parameters: | bufferLength | Range: | Determine if it is a null pointer |
| Return value: | uint8：0x00(E\_OK) 0XFF(E\_SAFETY\_HARD\_RUNTIMEERROR) | | |
| Functional Overview: | Protects the array/buffer to be transmitted, using the in-place transformation. | | |
| Precautions | None | | |

#### 5.8.4.2 E2EXf\_Inv\_<transformerId>

[SA\_R2\_338]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | E2EXf\_Inv\_<transformerId> | | |
| Function prototype: | uint8 E2EXf\_<transformerId> (  uint8\* buffer,  uint32\* bufferLength,  [const uint8\* inputBuffer],  uint32 inputBufferLength  ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | inputBuffer | Range: | Determine if it is a null pointer |
| inputBufferLength | Range: | Determine whether the length is less than EndToEndTransformationDescription.upperHeaderBitsToShift/8u |
| Input and output parameters: | buffer | Range: | Determine if it is a null pointer |
| Output parameters: | bufferLength | Range: | Determine if it is a null pointer |
| Return value: | uint8：  0x00 (E\_OK)  0x01 (E\_SAFETY\_VALID\_REP)  0x02 (E\_SAFETY\_VALID\_SEQ)  0x03 (E\_SAFETY\_VALID\_ERR)  0x05 (E\_SAFETY\_VALID\_NND)  0x20 (E\_SAFETY\_NODATA\_OK)  0x21 (E\_SAFETY\_NODATA\_REP)  0x22 (E\_SAFETY\_NODATA\_SEQ)  0x23 (E\_SAFETY\_NODATA\_ERR)  0x25 (E\_SAFETY\_NODATA\_NND)  0x30 (E\_SAFETY\_INIT\_OK)  0x31 (E\_SAFETY\_INIT\_REP)  0x32 (E\_SAFETY\_INIT\_SEQ)  0x33 (E\_SAFETY\_INIT\_ERR)  0x35 (E\_SAFETY\_INIT\_NND)  0x40 (E\_SAFETY\_INVALID\_OK)  0x41 (E\_SAFETY\_INVALID\_REP)  0x42 (E\_SAFETY\_INVALID\_SEQ)  0x43 (E\_SAFETY\_INVALID\_ERR)  0x45 (E\_SAFETY\_INVALID\_NND)  0x77 (E\_SAFETY\_SOFT\_RUNTIMEERROR) | | |
| Functional Overview: | Checks the received data. If the data can be used by the caller, then the function returns E\_OK. | | |
| Precautions | None | | |

#### 5.8.4.3 E2EXf\_Init

[SA\_R2\_339]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | E2EXf\_Init | | |
| Function prototype: | void E2EXf\_Init (const E2EXf\_ConfigType\* config) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | config | Range: | Pointer to static configuration. |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | None | | |
| Functional Overview: | Initializes the state of the E2E Transformer. The main part of it is the initialization of the E2E library state structures, which is done by calling all init-functions from E2E library. | | |
| Precautions | None | | |

#### 5.8.4.4 E2EXf\_DeInit

[SA\_R2\_340]:

|  |  |
| --- | --- |
| Function name: | E2EXf\_DeInit |
| Function prototype: | void E2EXf\_DeInit (void) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | None |
| Return value: | None |
| Functional Overview: | Deinitializes the E2E transformer. |
| Precautions | None |

#### 5.8.4.5 E2EXf\_GetVersionInfo

[SA\_R2\_341]:

|  |  |
| --- | --- |
| Function name: | E2EXf\_GetVersionInfo |
| Function prototype: | void E2EXf\_GetVersionInfo (Std\_VersionInfoType\* versioninfo) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | versioninfo: Pointer to where to store the version information of this module. |
| Return value: | None |
| Functional Overview: | Returns the version information of this module. |
| Precautions | None |

#### 5.8.4.6 ComXf\_<transformerId>

[SA\_R2\_342]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function name: | ComXf\_Inv\_<transformerId> | | | | |
| Function prototype: | uint8 ComXf\_<transformerId> (  uint8\* buffer,  uint32\* bufferLength,  <paramtype> dataElement  ) | | | | |
| Synchronous/Asynchronous: | Synchronous | | | | |
| Is it reentrant: | Reentrant | | | | |
| Input parameters: | dataElement | | Range: | | Determine if it is a null pointer |
| Input and output parameters: | None | | | | |
| Output parameters: | buffer | Range: | | Determine if it is a null pointer | |
| bufferLength | Range: | | Determine if it is a null pointer | |
| Return value: | uint8：0x00(E\_OK) 0X81(E\_SER\_GENERIC\_ERROR) | | | | |
| Functional Overview: | This function transforms a Sender/Receiver communication using the serialization of COM Based Transformer. It takes the data element as input and outputs a uint8 array containing the serialized data. | | | | |
| Precautions | None | | | | |

#### 5.8.4.7 ComXf\_Inv\_<transformerId>

[SA\_R2\_343]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | ComXf\_Inv\_<transformerId> | | |
| Function prototype: | uint8 ComXf\_Inv\_<transformerId> (  const uint8\* buffer,  uint32 bufferLength,  <type>\* dataElement  ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | Buffer | Range: | Determine if it is a null pointer |
| BufferLength | Range: | 不为0且大于SignalGroupLen |
| Input and output parameters: | None | | |
| Output parameters: | dataElement | Range: | Determine if it is a null pointer |
| Return value: | uint8：  0x00 (E\_OK)  0x01 (E\_NO\_DATA)  0x02 (E\_SER\_GENERIC\_ERROR) | | |
| Functional Overview: | This function deserializes a Sender/Receiver communication using the deserialization of COM Based Transformer. It takes the uint8 array containing the serialized data as input and outputs the original data element which will be passed to the Rte. | | |
| Precautions | None | | |

#### 5.8.4.8 ComXf\_Init

[SA\_R2\_344]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | ComXf\_Init | | |
| Function prototype: | void ComXf\_Init (const ComXf\_ConfigType\* config) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | config | Range: | Pointer to static configuration. |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | None | | |
| Functional Overview: | This service initializes the transformer for the further processing. | | |
| Precautions | None | | |

#### 5.8.4.9 ComXf\_DeInit

[SA\_R2\_345]:

|  |  |
| --- | --- |
| Function name: | ComXf\_DeInit |
| Function prototype: | void ComXf\_DeInit (void) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | None |
| Return value: | None |
| Functional Overview: | This service deinitializes the transformer. |
| Precautions | None |

#### 5.8.4.10 ComXf\_GetVersionInfo

[SA\_R2\_346]:

|  |  |
| --- | --- |
| Function name: | ComXf\_GetVersionInfo |
| Function prototype: | void ComXf\_GetVersionInfo (Std\_VersionInfoType\* versioninfo) |
| Synchronous/Asynchronous: | Synchronous |
| Is it reentrant: | Reentrant |
| Input parameters: | None |
| Input and output parameters: | None |
| Output parameters: | versioninfo: Pointer to where to store the version information of this module. |
| Return value: | None |
| Functional Overview: | This service returns the version information of the called transformer module. |
| Precautions | None |

## Calibration

Calibration is the process of adjusting an ECU SW to fulfill its tasks to control physical processes respectively to fit it to special project needs or environments. To do this two different mechanisms are required and have to be distinguished:

1. Measurement

Measure what’s going on in the ECU e.g. by monitoring communication data (Inter-ECU, Inter-Partition, Intra-partition, Intra-SWC). There are several ways to get the monitor data out of the ECU onto external visualization and interpretation tools.

2. Calibration

Based on the measurement data the ECU behavior is modified by changing parameters like runtime SW switches, process controlling data of primitive or composite data type, interpolation curves or interpolation fields. In the following for such parameters the term calibration parameter is used.

### Function Description

**Measurement**

RTE provides measurement support for

1. communication between Ports

Measurable are VariableDataPrototypes of a SenderReceiverInterface used in a PortPrototype (of a SwComponentPrototype) to capture senderreceiver communication or between SwComponentPrototypes.

See figure 5-47, mea\_variableData\_b.

2. communication inside of AUTOSAR SW-Cs

Measurable are arTypedPerInstanceMemory.

See figure 5-47, arTypedPerInstanceMemory\_a.



Figure 5‑47

The information that measurement shall be supported by RTE is defined in applied SwDataDefProps: The value readOnly or readWrite of the property swCalibrationAccess defines that measurement shall be supported, any other value of the property swCalibrationAccess is to be ignored for measurement.

[SA\_R2\_205]: If the swCalibrationAccess of a VariableDataPrototype used in an interface of a sender-receiver port of a SwComponentPrototype is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the instance specific data of the corresponding VariableDataPrototype of the communication can be accessed.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03900.

[SA\_R2\_206]: For a VariableDataPrototype with measurement demand associated with received data of inter-ECU sender-receiver communication the RTE shall provide only one measurement store reference containing the actual received data even if several receiver ports demand measurement.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03974.

[SA\_R2\_207]: For a VariableDataPrototype with measurement demand associated with received data of inter-Partition sender-receiver communication the RTE shall provide only one measurement store reference per partition containing the actual received data even if several receiver ports demand measurement in the Partition.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07344.

[SA\_R2\_208]: If the swCalibrationAccess of a VariableDataPrototype in the role arTypedPerInstanceMemory is set to readOnly or readWrite the RTE generator has to provide one reference to a location in memory where the actual content of the arTypedPerInstanceMemory can be accessed for a specific instantiation of the AUTOSAR SWC.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07160.

[SA\_R2\_209]: RTE generator shall reject configurations where measurement for queued sender-receiver communication is configured.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03950.

**Calibration**

With AUTOSAR, a calibration parameter is instantiated with a ParameterDataPrototype class that aggregates a SwDataDefProps with properties swCalibrationAccess = readWrite and swImplPolicy = standard.

RTE provides the calibration parameter access if they are specified via a ParameterSwComponentType. A ParameterSwComponentType can be defined in order to provide ParameterDataPrototypes (via ports) to other Software Components.

The RTE has to support the allocation of calibration parameters and the access to them for SW using them.

Calibration parameters can be defined in ParameterSwComponentTypes, in AUTOSAR SW-Cs.

a) ParameterSwComponentTypes don’t have an internal behavior but contain ParameterDataPrototypes and serve to provide calibration parameters used commonly by several AUTOSAR SW-Cs. The use case that one or several of the user SW-Cs are instantiated on different ECUs is supported by instantiation of the ParameterSwComponentType on the affected ECUs too. Of course several AUTOSAR SW-Cs allocated on one ECU can commonly access the calibration parameters of ParameterSwComponentTypes too. Also several instances of an AUTOSAR SW-Cs can share the same calibration parameters of a ParameterSwComponentType.

See figure 5-48.



Figure 5‑48

b) Calibration parameters defined in AUTOSAR SW-Cs can only be used inside the SW-C and are not visible to other SW-Cs. Instance individual and common calibration parameters accessible by all instances of an AUTOSAR SW-C are possible.

See figure 5-49.



Figure 5‑49

[SA\_R2\_210]: Several AUTOSAR SW-Cs (and also several instances of AUTOSAR SW-Cs) shall be able to share same calibration parameters defined in ParameterSwComponentTypes.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03958.

[SA\_R2\_211]: The generated RTE shall initialize the memory objects implementing ParameterDataPrototypes in p-ports of ParameterSwComponentTypes according the ValueSpecification of the ParameterProvideComSpec referring the ParameterDataPrototype in the p-port if such ParameterProvideComSpec exists.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07186.

[SA\_R2\_212]: If the SwcInternalBehavior aggregates an ParameterDataPrototype in the role perInstanceParameter the RTE shall support the access to instance specific calibration parameters of the AUTOSAR SW-C.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03959.

[SA\_R2\_213]: If the SwcInternalBehavior aggregates an ParameterDataPrototype in the role sharedParameter the RTE shall create a common access to the shared calibration parameter.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05112.

[SA\_R2\_214]: In case of an unconnected parameter r-port, the RTE shall set the values of the ParameterDataPrototypes of the r-port according to the initValue of the r-port’s ParameterRequireComSpec referring to the ParameterDataPrototype.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_02749.

[SA\_R2\_215]: The RTE generator shall support separation of calibration parameters from ParameterSwComponentTypes, AUTOSAR SW-Cs and Basic Software Modules depending on the ParameterDataPrototype property swAddrMethod.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03907.

[SA\_R2\_216]: The RTE shall allocate the memory for calibration parameters.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03961.

[SA\_R2\_217]: For accesses to a shared ParameterDataPrototype the RTE API shall deliver the same one value independent of the instance the calibration parameter is assigned to.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03962.

[SA\_R2\_218]: For accesses to an instance specific ParameterDataPrototype the RTE API shall deliver a separate calibration parameter value for each instance of a ParameterDataPrototype.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03964.

[SA\_R2\_219]: For an instance specific ParameterDataPrototype the calibration parameter value of each instance of the ParameterDataPrototype shall be stored in a separate memory location.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03965.

[SA\_R2\_220]: The memory section used to store measurement values in shall be the memory sections associated with the swAddrMethod enclosed in the SwDataDefProps of a measurement definition.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03981.

[SA\_R2\_221]: The memory section used to store calibration parameters in shall be the memory sections associated with the swAddrMethod enclosed in the SwDataDefProps of a calibration parameter definition.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_03982.

**Generation of A2L file**

The RTE generator shall support generating the corresponding elements in the A2L file from the identified measurement data and calibration data. In this section the requirements on the RTE Generator are collected which elements shall be provided in the A2L file.

[SA\_R2\_222]: If the swCalibrationAccess of a VariableDataPrototype used in an interface of a sender-receiver port of a SwComponentPrototype is set to readOnly or readWrite and RteMeasurementSupport is set to true the RTE Generator shall create an entry of measurement data in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05120.

[SA\_R2\_223]: If the swCalibrationAccess of a VariableDataPrototype in the role arTypedPerInstanceMemory is set to readOnly or readWrite and RteMeasurementSupport is set to true the RTE Generator shall create an entry of measurement data in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05123.

[SA\_R2\_224]: For each FlatInstanceDescriptor referencing a ParameterDataPrototype instance in a PortPrototype of a ParameterSwComponentType with the swCalibrationAccess set to readOnly or readWrite an entry of calibration shall be created in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05126.

[SA\_R2\_225]: For each FlatInstanceDescriptor referencing a ParameterDataPrototype instance of a AtomicSwComponentType’s SwcInternalBehavior aggregated in the role sharedParameter with the swCalibrationAccess set to readOnly or readWrite an entry of calibration shall be created in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05127.

[SA\_R2\_226]: For each FlatInstanceDescriptor referencing a ParameterDataPrototype instance of a AtomicSwComponentType’s SwcInternalBehavior aggregated in the role perInstanceParameter with the swCalibrationAccess set to readOnly or readWrite an entry of calibration shall be created in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_05128.

[SA\_R2\_227]: For each ParameterDataPrototype of a BswModuleDescription’s BswInternalBehavior aggregated in the role perInstanceParameter with the swCalibrationAccess set to readOnly or readWrite an an entry of calibration shall be created in the A2L file.Refer to 'AUTOSAR\_SWS\_RTE.pdf ' SWS\_Rte\_07097.

**Others**

[SA\_R2\_293]: Measurement and calibration parameters can have initial values and shall be generated for them if defined.

[SA\_R2\_294]: The syntax of A2L files shall follow ASAM rules. For details, please refer to ASAM\_MCD-2MC\_DataSpecifcation\_V1.6.pdf.

[SA\_R2\_295]: The RTE tool should avoid all SWC-to-calibration parameter modifications. However, the calibration can only be modified by the calibration engineer through the host computer.

### Control Flow

[SA\_R2\_228]:



Figure 5‑50 An example of a scenario where the user uses the RTE calibration function.

The user defines measurement and calibration data in the SWC tool. The RTE generator generates data and related interfaces after identifying the measurement and calibration data, and generates elements for the measurement data and calibration data in the A2L file. The user accesses the measurement and calibration data by the generated interface and takes the A2L file as the input of the calibration host computer.

### Data Flow

[SA\_R2\_229]:

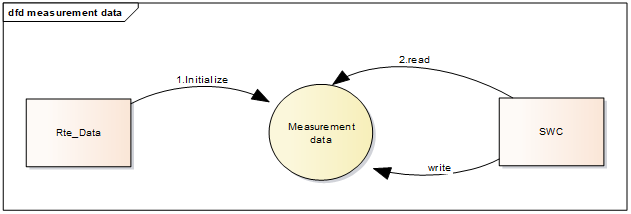


Figure 5‑51 Data Flow Diagaram for measurement data

[SA\_R2\_230]:

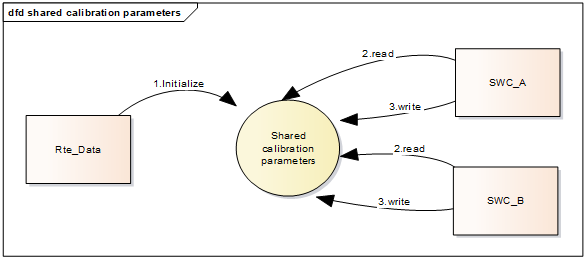


Figure 5‑52 Data Flow Diagaram for shared calibration data

[SA\_R2\_231]:

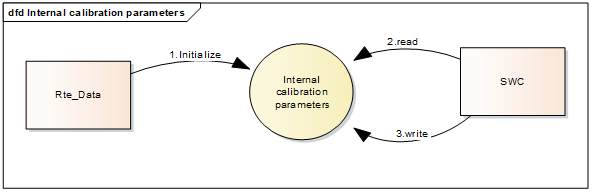


Figure 5‑53 Data Flow Diagaram for internal calibration data

### External Interface Definition

#### Rte\_CData

[SA\_R2\_330]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_CData | | |
| Function prototype: | <DataType> Rte\_CData\_<cp> ( [IN Rte\_Instance instance] )  <DataType> \*Rte\_CData\_<cp> ( [IN Rte\_Instance instance] ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | instance | Range: | NA |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | <DataType>:The Rte\_CData return value provide access to the data value of the  ParameterDataPrototype in the role perInstanceParameter or sharedParameter. | | |
| Functional Overview: | The protection of the transmitted data based on the Profile1 algorithm is realized through this API. Contains check value calculation, counter and DataID processing operations. | | |
| Precautions | None | | |

#### Rte\_Prm

[SA\_R2\_331]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Prm | | |
| Function prototype: | <DataType> Rte\_Prm\_<p>\_<cp> ( [IN Rte\_Instance instance] )  <DataType> \*Rte\_Prm\_<p>\_<cp> ( [IN Rte\_Instance instance] ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | instance | Range: | NA |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType：E2E\_E\_INPUTERR\_NULL E2E\_E\_INPUTERR\_WRONG E2E\_E\_INTERR E2E\_E\_OK | | |
| Functional Overview: | The Rte\_Prm API provides access to the defined parameter within a ParameterSwComponentType. | | |
| Precautions | None | | |

#### Rte\_Pim

[SA\_R2\_332]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Pim | | |
| Function prototype: | <DataType> \*Rte\_Pim\_<n> ( [IN Rte\_Instance instance] ) | | |
| Synchronous/Asynchronous: | Synchronous | | |
| Is it reentrant: | Reentrant | | |
| Input parameters: | ConfigPtr | Range: | Pointer to static configuration. |
| Input and output parameters: | StatePtr：Pointer to port/data communication state.  DataPtr：Pointer to Data to be transmitted. | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType：E2E\_E\_INPUTERR\_NULL E2E\_E\_INPUTERR\_WRONG E2E\_E\_INTERR E2E\_E\_OK | | |
| Functional Overview: | The protection of the transmitted data based on the Profile1 algorithm is realized through this API. Contains check value calculation, counter and DataID processing operations. | | |
| Precautions | None | | |

## Initialization and Finalization

Some of the RTE modules require initialization of global variables that are necessary to fulfill the functionalities, such as the buffering of qeueing C-S communication. Also, timing events that trigger the runnables will be started and activated by RTE initialization and terminated by RTE finalization.

### Function Description

[SA\_R2\_233]: [The RTE shall support initialization and finalization of Application Software Components.

The term "initialization of a component" refers to the phase of a software components life cycle which will be executed before entering the normal operational mode, normally in order to set up an appropriate environment for executing the application.

The term "finalization of a component" refers to the phase of a software components life cycle which will be executed after the normal operational mode, normally in order to reset the operational environment to a determined state.]

refers to 'AUTOSAR\_SWS\_RTE' [SRS\_RTE\_00052].

[SA\_R2\_234]: [The call of Rte\_Start shall start the activation of RunnableEntitys triggered by TimingEvents.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07575]

[SA\_R2\_235]: [The call of Rte\_Start shall start the activation of RunnableEntitys triggered by BackgroundEvents.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07178]

[SA\_R2\_236]: [The RTE shall not activate, start or release RunnableEntitys on a core after Rte\_Stop has been called on this core.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02538]

[SA\_R2\_237]: [RTE shall ignore incoming client server communication requests,before RTE is initialized completely and when it is stopped.] refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_02535]

[SA\_R2\_238]: [Incoming data and events from sender receiver communication shall be ignored, before RTE is initialized completely and when it is stopped.] refers to 'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_02536]

### Control Flow

Refer to 4.1 State Mechanism for details.

### Data Flow

[SA\_R2\_239]:

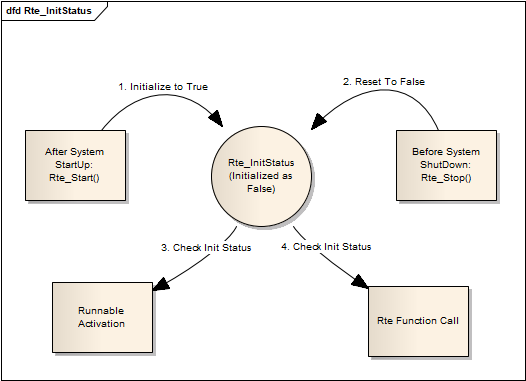


Figure 5‑54 Data Flow Diagaram for Rte\_InitStatus

As figure 5-37 shows, the Rte Initialization status variable is defined as False by definition. It is initialized to True after calling Rte\_Start and reset to False after calling Rte\_Stop. Runnables can only be activated and runnables cannot make any Rte function call while Rte\_InitStatus is False, which means the Rte is not initialized.

### External Interface Definition

#### Rte\_Start

[SA\_R2\_348]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Start | | |
| Function prototype: | Std\_ReturnType Rte\_Start(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType | | |
| Functional Overview: | Rte\_Start is intended to allocate and initialize system resources and communication resources used by the RTE.[The Rte\_Start API is always created.] refers to  'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_01309] | | |
| Precautions | None | | |

#### Rte\_Stop

[SA\_R2\_349]:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: | Rte\_Stop | | |
| Function prototype: | Std\_ReturnType Rte\_Stop(void) | | |
| Synchronous/Asynchronous: | Sync | | |
| Is it reentrant: | Non Reentrant | | |
| Input parameters: | None | Range: | *--* |
| Input and output parameters: | None | | |
| Output parameters: | None | | |
| Return value: | Std\_ReturnType | | |
| Functional Overview: | Rte\_Stop is used to finalize the RTE on the core it is called. This service releases all system and communication resources allocated by the RTE on that core. [The Rte\_Stop API is always created.] refers to 'AUTOSAR\_SWS\_RTE'[SWS\_Rte\_01310] | | |
| Precautions | None | | |

## Anxiliary Function

AUTOSAR supports the connection of an R-port to a P-port with an interface that is not compatible in the sense of the AUTOSAR compatibility rules. In addition, for sender receiver communication it is possible to specify how data elements are represented given that the communication requires the usage of a dedicated communication bus. In these cases the generated RTE has to support the conversion and re-scaling of data. refers to 'AUTOSAR\_SWS\_RTE' 4.3.5

### Function Description

[SA\_R2\_241]: The RTE shall support the conversion of an identical or linear scaled data representation to another identical or linear scaled data representation. In this context, the term "linear scaled data representation" also includes floating-point data representations. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03829].

[SA\_R2\_242]: The RTE shall support the conversion of a texttable data representation (enumeration or bitfield) to another texttable data representation. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03830].

[SA\_R2\_243]: The RTE shall support the conversion of a mixed linear scaled and texttable data representation to another mixed linear scaled and texttable data representation.refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_03855].

[SA\_R2\_244]: The data conversion shall be supported for data types that refer to CompuMethods of category LINEAR, IDENTICAL, SCALE\_LINEAR\_AND\_TEXTTABLE, TEXTTABLE. refers to 'AUTOSAR\_SWS\_RTE' [SWS\_Rte\_07928].

[SA\_R2\_245]: For C/C++ AUTOSAR software-components, the name of the RTE header file shall be Rte.h. [SWS\_Rte\_01157]

[SA\_R2\_246]: The RTE header file shall include the file Std\_Types.h.[SWS\_Rte\_01164]

[SA\_R2\_247]: For C/C++ AUTOSAR software-components, the name of the lifecycle header file shall be Rte\_Main.h.[SWS\_Rte\_01158]

[SA\_R2\_248]: The lifecycle header file shall include the RTE header file.[SWS\_Rte\_01159]

[SA\_R2\_249]: The application header file shall include the Application Types Header File. [SWS\_Rte\_07131]

[SA\_R2\_250]: The application header file shall include the RTE Data Handle Types Header File.[SWS\_Rte\_07924]

[SA\_R2\_251]: The Application Types Header File shall include the RTE Types Header File.[SWS\_Rte\_07127]

[SA\_R2\_252]: The RTE Data Handle Types Header File(Rte\_DataHandleType.h) shall include the RTE Types Header File independent whether this is directly needed or not.[SWS\_Rte\_80009]

[SA\_R2\_254]: The RTE generator shall generate a separate header file containing the prototypes of callback functions.[SWS\_Rte\_03795]

[SA\_R2\_257]: The RTE shall prevent runnable entities of AUTOSAR software components to run in interrupt context.[SWS\_Rte\_03600]

[SA\_R2\_258]: RTE应能为Callback、CALLOUT提供相应的RTE封装函数.

[SA\_R2\_259]: The RTE module shall define the generated functions that will be invoked when an AUTOSAR software-component makes an RTE API call.[SWS\_Rte\_01266]

[SA\_R2\_260]: The generated RTE shall contain an instance of the relevant Component Data Structure for each software-component instance on the ECU for which the RTE is generated.[SWS\_Rte\_03711]

[SA\_R2\_261]: RTE应为每个软件组件实例提供一个标准的命名.[SWS\_RTE\_03712]

[SA\_R2\_262]: If all sections of the Component Data Structure are empty the Component Data Structure shall contain a uint8 with name Rte\_Dummy.[SWS\_Rte\_03724]

[SA\_R2\_263]: RTE应在头文件中加上防止头文件被多次包含的宏条件判断.[SWS\_Rte\_07923]

[SA\_R2\_264]: RTE应拒绝application component type没有关联implementation component type的配置.[SWS\_Rte\_07028]

[SA\_R2\_266]: The API mapping shall be implemented in the application header file.[SWS\_RTE\_01274]

[SA\_R2\_267]: RTE应该为所有的API提供一个标准的命名规范，例如:Rte\_Read等.[SWS\_RTE\_01055]

[SA\_R2\_268]: The Rte provides API data type defines the handle used to access instance specifific information from the component data structure.[SWS\_Rte\_01150]

[SA\_R2\_269]: The component data structure type shall be defifined in the Application Header file.[SWS\_Rte\_07132]

[SA\_R2\_270]: RTE提供内部数据结构一个标准的命名.[SWS\_Rte\_03714]

[SA\_R2\_271]: RTE中所有的全局变量和函数的名称都以Rte\_为前缀.[SWS\_Rte\_01171]

[SA\_R2\_272]: RTE应提供对每个Component实例的标识.[SWS\_RTE\_01012]

[SA\_R2\_273]: RTE应提供端口数据结构和标准的命名.[SWS\_Rte\_03731] [SWS\_Rte\_06522]

[SA\_R2\_274]: The RTE shall not suspend execution while executing a call-back.[SRS\_Rte\_00022]

[SA\_R2\_275]: RTE应该直接提供数据buffer的初始值，避免返回未定义的数值.[SWS\_Rte\_07046]

[SA\_R2\_277]: RTE应提供状态位的生成以供上层获取状态.

[SA\_R2\_292]: The name of the instance handle type shall be defined, using typedef as Rte\_[Byps\_]Instance.[SWS\_Rte\_01150]

[SA\_R2\_307]: RTE shall provide Manual user should ensure that the array length does not exceed the limit.

[SA\_R2\_313]: RTE should provide protection against data inconsistencies in the queue.

[SA\_R2\_351]: RTE生成器应创建RTE类型头文件，包括与输入配置中定义的ImplementationDataTypes以及RTE实现类型相对应的类型声明.[SWS\_Rte\_01160]

### Control Flow

[SA\_R2\_278]:

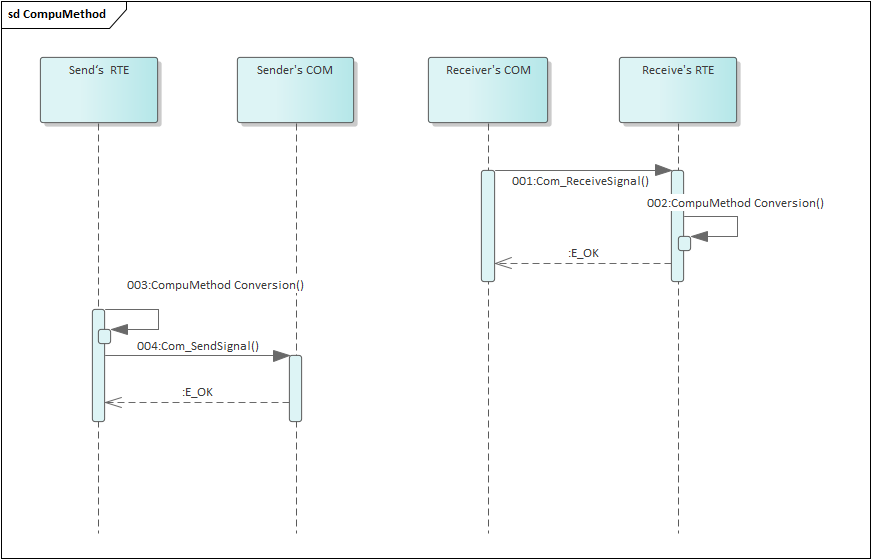


Figure 5‑55 Conversion of Compu Method

Compu Method represents the ability to express the relationship between a physical value and the mathematical representation.对于配置了计算方法的信号发出时，需要在RTE端根据计算方法将数据转换后调用Com\_SendSignal函数发出。对于接收信号，在调用Com\_ReceiveSignal收到信号后需要在RTE使用计算方法转换后发送到总线上。

### Data Flow

[SA\_R2\_279]:

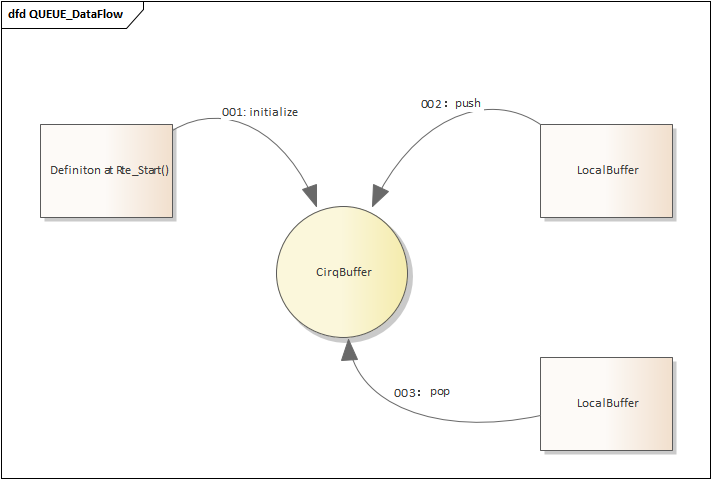


Figure 5‑56 Data Flow Diagaram for CirqBuffer

1. Rte\_Start initialize CirqBuffer
2. Push the local variable into Global CirqBuffer
3. Pop the GlobalCirqBuffer

[SA\_R2\_280]:

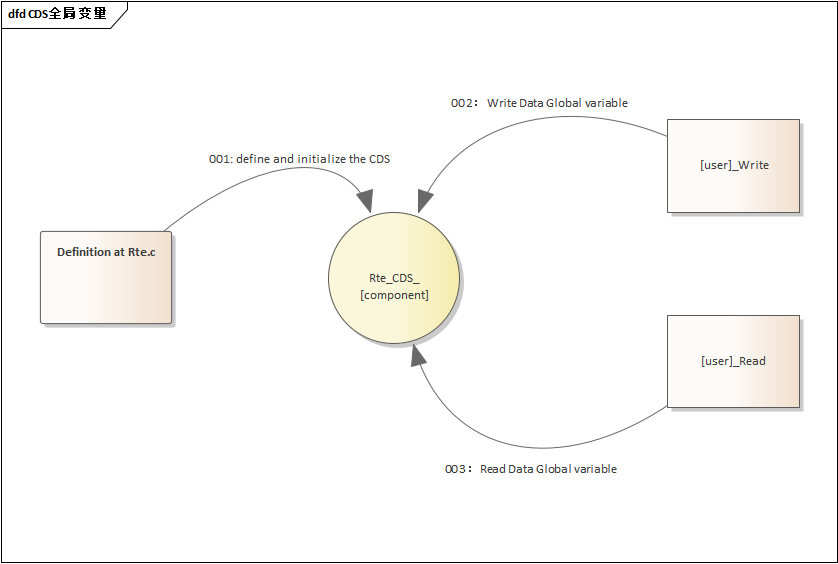


Figure 5‑57 Data Flow Diagaram for CirqBuffer

As figure 5-35 shows,Per Component Data Structure has a Global CDS,The CDS used to access instance specifific information from the component data structure.

### External Interface Definition

N/A

## Validation Criteria

The generated software must pass all chosen Autosar ATS cases to be validated.

# Memory Map Design

[SA\_R2\_290]:

RTE related memory map section assignment is produced automatically based on section type and data type. Although it is automatic, the detail of the section could still be modified by configuring SwAddrMethod in SWC.

RTE supports the following sectionType: VAR, INTERNAL\_VAR, VAR\_SAVED\_ZONE, CONST, CALIB, CONFIG\_DATA, CONST\_PBCFG, CODE, CALLOUT\_CODE, CALLBACK\_CODE, CODE\_FAST.

RTE supports the following SectionInitializationPolicy: NO-INIT, INIT, POWER-ON-INIT, CLEARED,

POWER-ON-CLEARED.

RTE supports the following alignment based on ImplementationDataType: 8, 16, 32(bit number), BOOLEAN, UNSPECIFIED (strcuture, double, float, etc.), PTR(pointer).