Elektrobit

EB tresos Solutions for Essentials

Documentation

product release 8.8.7

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Table of Contents

1. Overview of EB tresos Solutions for Essentials

This document includes the following chapters:

- Chapter 2, “Supported features“: provides the list of supported features and their descriptions.
- Chapter 3, “Step by step for your project“: provides information to get familiar with the product and step-by-step instructions.
- Chapter 5, “Best practices“: provides a list of best practices and tips to avoid the most common errors.

Disclaimer: The content of this document might include information that is not related to your product order. To learn what components are part of your product order, refer to the Components list of your delivery on EB command.

2. Supported features

This chapter provides an overview of the components and features of EB tresos Solutions for Essentials.

- Section 2.1, “Product details”: contains an overview of the product.
- Section 2.2.1, “Startup Package Application CAN”: contains an overview of the Startup Package Application CAN.
- Section 2.2.2, “Startup Package Application FlexRay”: contains an overview of the Startup Package Application FlexRay.
- Section 2.2.3, “Startup Package Application IP”: contains an overview of the Startup Package Application IP.
- Section 2.2.4, “Common features”: contains an overview of the common features.

2.1. Product details

The architecture consists of basic software modules and additional Software Components (SWCs) according to the AUTOSAR standard. Some parts are not always present as it depends on the licenses and services. Refer to *EB_tresos_Solutions_for_Essentials_Product_Description.pdf* for details.

- **EB tresos AutoCore Generic** : hardware-independent basic software modules.
- **EB tresos Autocore OS**: operating system, ported and qualified for one dedicated microcontroller unit and derivative. For more information, refer to the corresponding product description.
- **MCAL and hardware-dependent modules**: hardware-dependent basic software modules for the ordered architectures and derivatives.
- **EB tresos Modules for Essentials**: AUTOSAR-compliant software component for the ordered architecture(s): Enhanced Testability Service (ETS).

2.2. Feature details

2.2.1. Startup Package Application CAN

2.2.1.1. Module usage

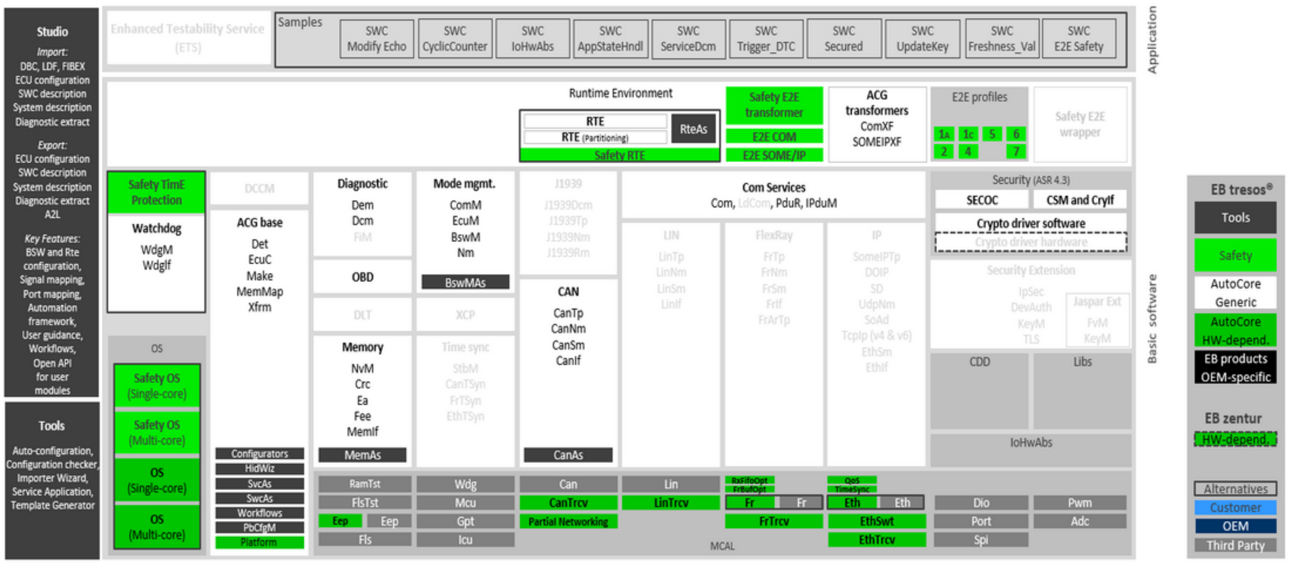


Figure 2.1. Module usage in Startup Package Application CAN

2.2.1.2. Software components

The SWC composition consists of the following components:

- SWC_CyclicCounter
- SWC_ModifyEcho
- SWC_IoHwAbs
- SWC_AppStateHandler
- ServiceSwComponentDcm
- SWC_Trigger_DTC
- SWC_Secured

- SWC_UpdateKey
- SWC_Freshness_Value
- SWC_E2ESafety

2.2.1.3. CAN/CAN FD communication

Startup Package Application CAN supports CAN and CAN FD communication with standard (11b) and extended (29b) message IDs.

ID	Type	Length	Direction	Description
0x100	CAN	1 byte	Rx	Contains Pdu_CounterIn at byte 0
0x110	CAN	1 byte	Tx	Contains Pdu_CounterOut at byte 0
0x00000101	CAN FD	64 byte	Rx	Contains Pdu_CounterInFd at byte 0
0x00000111	CAN FD	64 byte	Tx	Contains Pdu_CounterOutFd at byte 0
0x200-0x250	CAN	8 byte	Rx	NM Message
0x251	CAN	8 byte	Tx	NM message
0x18DAEBF1	CAN	8 byte	Rx	UDS physical request
0x18DAEBF2	CAN FD	64 byte	Rx	UDS physical request
0x18DBFEF1	CAN	8 byte	Rx	UDS functional request
0x18DBFEF2	CAN FD	64 byte	Rx	UDS functional request
0x18DAF1EB	CAN	8 byte	Tx	UDS physical response
0x18DAF2EB	CAN FD	64 byte	Tx	UDS physical response
0x7DF	CAN	8 byte	Rx	OBD functional request
0x7E0	CAN	8 byte	Rx	OBD physical request
0x7E8	CAN	8 bye	Tx	OBD physical response
0x119	CAN	8 byte	Rx	SecOc Request Message
0x11D	CAN	8 byte	Tx	SecOc Response Message
0x118	CAN	8 byte	Rx	E2E Request Message
0x11C	CAN	8 byte	Tx	E2E Response Message

Table 2.1. Supported CAN messages

Baudrate is 500 kBaud for CAN communication, and 2 MBaud for CANFD communication.

An NM message must be sent to the ECU to wake it up. After the start-up, the ECU displays:

- Two application counters that are increased periodically every second. The counters start at default value 0 after flashing the ECU, or the previously saved value for the one sent on CAN communication if the ECU performs a sleep wakeup sequence.
- A Dio channel (usually connected to one of the available board LED lights) that is toggled periodically every second.
- One CAN message Pdu_CounterOut that is sent periodically from the ECU every second, and which contains 1 payload byte containing current counter value.
- One CAN message Pdu_CounterIn that can be accepted by the ECU.
- Support for application counter set/reset in Pdu_CounterOut(Pdu_CounterIn with payload value less than 0xF0), pseudo development error (Pdu_CounterIn with payload value equal to 0xF1), and ECU shutdown (Pdu_CounterIn with payload value equal to 0xFF) commands.
- One CAN FD message Pdu_CounterOutFd that is sent periodically from the ECU every second and which contains 1 payload byte containing current counter value.
- One CAN FD message Pdu_CounterInFd that can be accepted by the ECU.
- Support for application counter set/reset in Pdu_CounterOutFd, Pdu_CounterInFd payload value.

2.2.1.4. Network Management

Network management over CAN with partial networking is supported. The supported NM messages are listed in Section 2.2.1.3, “CAN/CAN FD communication”.

An NM message consists of:

- NM node identifier with 1 byte lenght at position 0.
- NM Control Bit Vector with 1 byte lenght at position 1.
- Partial network info with 6 byte length at position 2.

The ECU node identifier is 0xEB.

The ECU is part of the following Partial Network Clusters:

- 0; contains Rx Id 0x100, Tx Id 0x110.
- 1; contains Rx Id 0x00000101, Tx Id 0x00000111.

To wake up and keep the ECU awake, the Nm message with a valid PNC enabled must be sent periodically. Refer to AUTOSAR documentation for details on CAN network management.

An example of the message payload to keep the ECU awake is: `0x56 0x51 0x03 0x00 0x00 0x00 0x00 0x00`. This enables both supported PNCs.

2.2.1.5. Diagnostics

2.2.1.5.1. Unified Diagnostic Services (UDS)

UDS over CAN/CAN FD is implemented with 0xEB as ECU address for TP. Two testers are supported, 0xF1 over CAN and 0xF2 over CAN FD. Physical and functional addressing is supported.

2.2.1.5.2. On-Board Diagnostics (OBD)

OBD over CAN is implemented, physical and functional addressing is supported.

2.2.1.6. Secure On-Board communication over CAN (SecOC)

Secure Onboard Communication provides protection on bus level. Secured PDU includes the FV (Freshness Value) and the MAC (Message Authentication Code) that ensure the integrity of the signal.

The ECU receives a secured PDU on CAN with the ID 0x119 and responds with a secured PDU with the ID 0x11D.

The secured PDU has the following attributes:

- Authentification Algorithm = CMAC/AES-128. This parameter defines the authentication algorithm used for authentication verification.
- Authentic PDU = 3 bytes. This parameter defines the length of the authentic PDU.
- Message Authentication Code = 4 bytes. This parameter defines the length in bits of the authentication code.
- Truncated Freshness Value Length = 1 byte. This parameter defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU.

2.2.1.7. End To End protection over CAN (E2E)

The E2E uses the following safety mechanisms:

- Cyclic redundancy check (CRC: A 16-bit CRC is explicitly sent with polynomial in normal form 0x1021 with an initial value 0xFFFF and a final XOR-value 0x0000. It starts with the bit offset 0.
- Sequence counter/alive counter: An 8-bit sequence number is explicitly sent and incremented at every transmission request. The Sequence counter starts with bit offset 16.

- Data In/Data Out:An application counter that is increased periodically every second. The counter starts at default value 0 after flashing the ECU, and it can be set to a certain value. It starts with bit offset 24.
- Data ID: A system-wide unique 16-bit data ID is implicitly sent for every port data element. The Data Id used for the transmition is 0x11C and for reception 0x118.

Byte 0 + Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**
CRC	Sequence Counter	Data Out/Data In	0xFF	0xFF	0xFF	0xFF

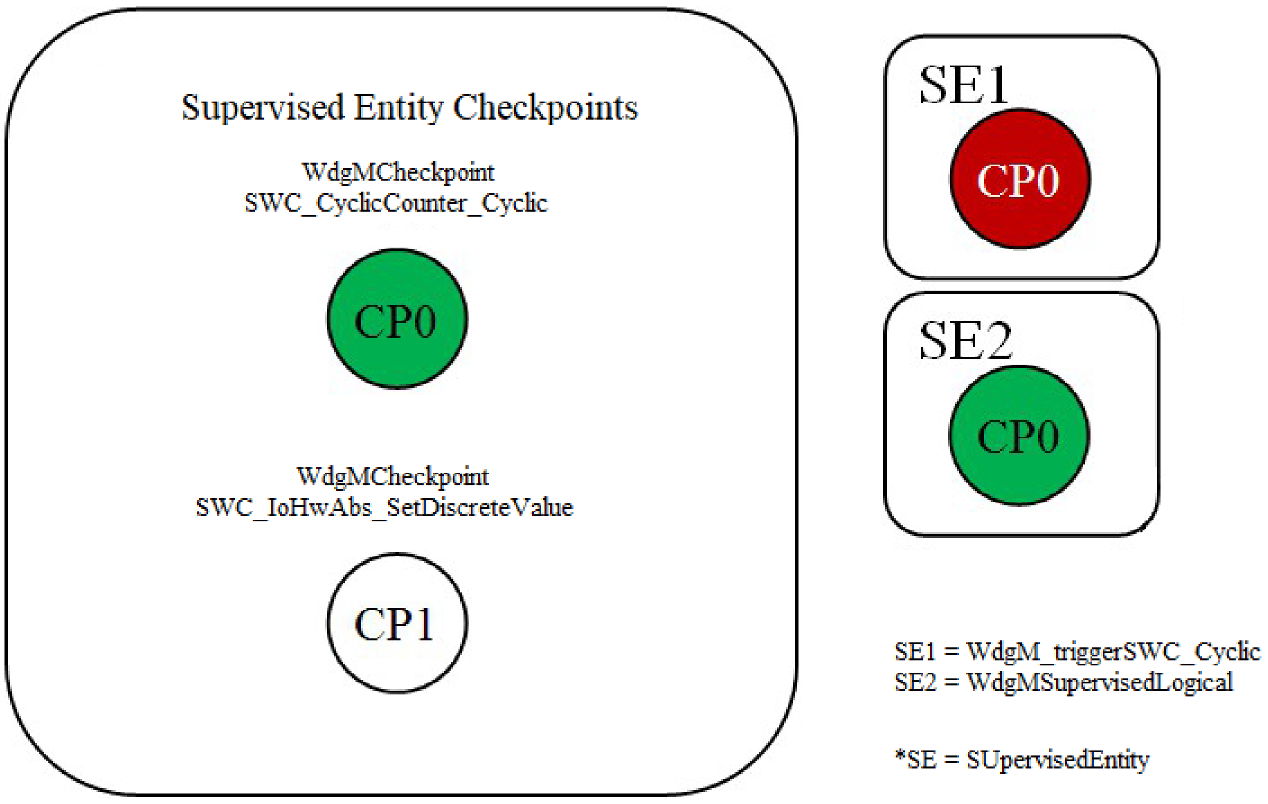
Table 2.2. E2E message layout

2.2.1.8. Safety Time and Execution Protection (TimEMP)

In the TimEPM, the safety mechanism consists of adding a Checkpoint (CP) to the control flow of a SW-C, BSW or CDD under the observation and evaluation of this Checkpoint at run time for a specified reference. In the TimEPM, the SW-C, BSW or CDD under observation is called Supervised Entity (SE). The TimEPM implementation from Solution for Essentials provides the following safety mechanisms:

- Alive Supervision is part of Temporal Prog Flow Monitoring. Alive Supervision verifies if the expected number of Alive Indications that are reported by the Checkpoint of a Supervised Entity are within specified limits. The TimEPM periodically checks if a Supervised Entity is not run too frequently or too rarely.
- Logical Supervision verifies if the configured Graphs are executed. This means that every time a Checkpoint of a Supervised Entity is reported, the TimEPM verifies if this Checkpoint is an authorized successor of the previously reported Checkpoint.
- Deadline Supervision is part of Temporal Prog Flow Monitoring and verifies the evaluation of the timing of transitions between the start and the end of the 5 ms SchMComTask_5ms with respect to a minimum and maximum allowed timing. The TIMEPM checks if the measured time duration is within the configured timing constraint.It verifies evaluation of the timing of transitions between the start and the end of the 5 ms SchMComTask_5ms.

Two Checkpoints (0_SWC_CyclicCounter_Cyclic and 1_SWC_IoHwAbs_SetDiscreteValue) were configured. In the application code, the two Checkpoints are called in the corresponding functions SWC_CyclicCounter_Cyclic and SWC_IoHwAbs_SetDiscreteValue. Logical Supervision is configured for WDGM_NORMAL_OPERATION mode. The switch to NORMAL_OPERATION mode is done by Supervision Callouts with the function Supervisor_WdgM_GetExpectedWdgMModeCallout().


Supervised Entity Checkpoints
WdgMCheckpoint
SWC_CyclicCounter_Cyclic
CP0
WdgMCheckpoint
SWC_IoHwAbs_SetDiscreteValue
CP1
SE1
CP0
SE2
CP0
SE1 = WdgM_triggerSWC_Cyclic
SE2 = WdgMSupervisedLogical
*SE = SUpervisedEntity

Supervision Callouts (only available with TimE license) were enabled inside the WdgM plugin.

2.2.2. Startup Package Application FlexRay

2.2.2.1. Module usage

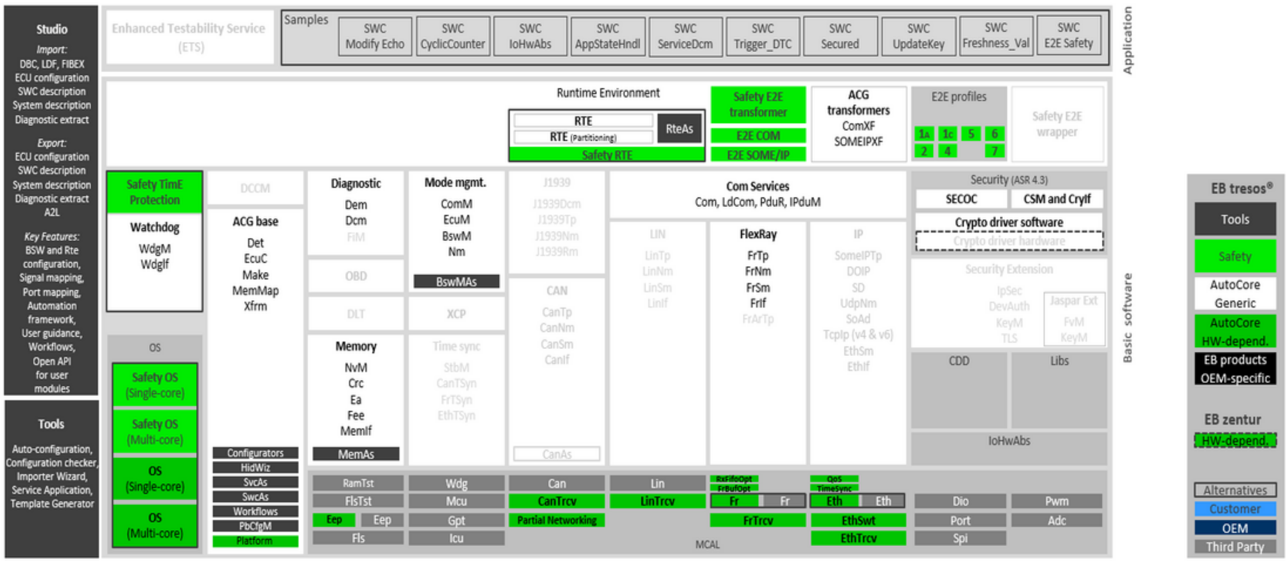


Figure 2.2. Module usage in Startup Package Application FlexRay

2.2.2.2. Software components

The SWC composition consists of the following components:

- SWC_CyclicCounter
- SWC_ModifyEcho
- SWC_IoHwAbs
- SWC_AppStateHandler
- ServiceSwComponentDcm
- SWC_Trigger_DTC
- SWC_Secured
- SWC_UpdateKey
- SWC_Freshness_Value
- SWC_E2ESafety

2.2.2.3. Hardware requirements

If the TC38XQ is used, for the FrTrcv driver to work properly, it is necessary that the board PIN 10.2 is connected to the STBN input of the TLE9221SX transceiver. For this purpose, an additional connection must be soldered to the board as shown in the diagram below.

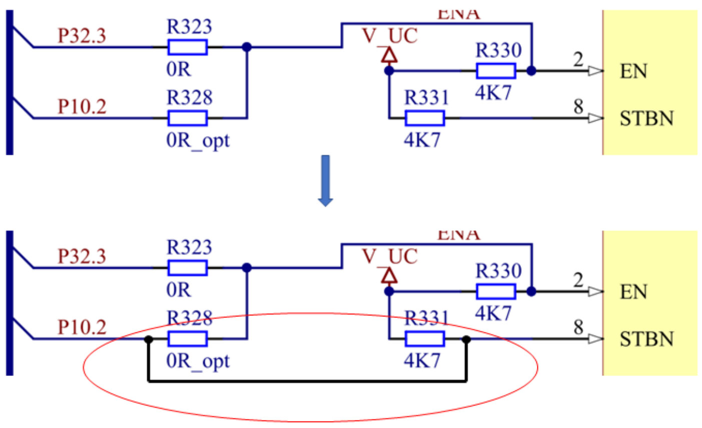


Figure 2.3. Additional connection for TC38XQ

2.2.2.4. FlexRay communication

The FlexRay application is not supported on WINDOWS(WIN32X86)

Startup Package Application FlexRay supports FlexRay communication with static and dynamic slot IDs.

The following messages are supported:

Slot ID	Base Cycle	Cycle repetition	Direction	Description
1	0	1	Rx	Contains Pdu_CounterIn at byte 0
10	0	1	Tx	Contains Pdu_CounterOut at byte 0
93	7	8	Rx	NM Message
93	8	16	Tx	NM message
200	0	1	Rx	UDS functional request
201	0	1	Rx	UDS physical request
202	0	1	Tx	UDS physical response
300	0	1	Rx	SecOc Request Message
301	0	1	Tx	SecOc Response Message

Slot ID	Base Cycle	Cycle repetition	Direction	Description
330	0	1	Rx	E2E Request Message
331	0	1	Tx	E2E Response Message

Table 2.3. Overview of FlexRay messages

After a startup, the ECU shows:

- An application counter that is increased periodically every second. The counter starts at default value 0 after flashing the ECU, or the previously saved value if the ECU performs a sleep wakeup sequence.
- A Dio channel (usually connected to one of the available board LED lights) that is toggled periodically every second.
- One FlexRay frame, that is sent periodically from the ECU every second, and that contains the current counter value in Pdu_CounterOut at byte 0.
- One FlexRay frame, which can be accepted by the ECU containing the Pdu_CounterIn at byte 0.
- Support for application counter set/reset (Pdu_CounterIn with payload value less than 0xF0), pseudo development error (Pdu_CounterIn with payload value equal to 0xF1), and ECU shut-down (Pdu_CounterIn with payload value equal to 0xFF) commands.

2.2.2.5. Network Management

The Startup Package Application FlexRay supports network management over FlexRay without partial networking. The supported NM messages are listed in Section 2.2.2.4, “FlexRay communication”.

An NM message consists of the following:

- NM node identifier with 1 byte length at position 0.
- NM Control Bit Vector with 1 byte length at position 1.

The ECU node identifier is 0xEB.

For details on FlexRay network management, refer to the AUTOSAR documentation.

2.2.2.6. Diagnostics

Diagnostics according to ISO10681-2:2010 are implemented with 0xEB as ECU address for TP. The supported tester is 0x7F. Physical and functional addressing is supported.

For a description of the supported services, refer to Section 2.2.4.1.1, “Unified Diagnostic Services (UDS)”.

The supported diagnostic messages are listed in Section 2.2.2.4, “FlexRay communication”.

2.2.2.7. Secure On-Board communication over FlexRay (SecOC)

Secure Onboard Communication provides protection on bus level. Secured PDU includes the FV (Freshness Value) and the MAC (Message Authentication Code) that ensure the integrity of the signal.

The ECU receives a secured PDU on FlexRay with the Slot ID 300 (base cycle 0, cycle repetition 1) and responds with a secured PDU with the Slot ID 301(base cycle 0, cycle repetition 1).

The secured PDU has the following attributes:

- Authentification Algorithm = CMAC/AES-128. This parameter defines the authentication algorithm used for authentication verification.
- Authentic PDU = 9 bytes. This parameter defines the length of the authentic PDU.
- Message Authentication Code = 4 bytes. This parameter defines the length in bits of the authentication code.
- Truncated Freshness Value Length = 1 byte. This parameter defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU.

2.2.2.8. End To End protection over FlexRay (E2E)

The E2E uses the following safety mechanisms:

- Cyclic redundancy check (CRC: A 16-bit CRC is explicitly sent with polynomial in normal form 0x1021 with an initial value 0xFFFF and a final XOR-value 0x0000. It starts with the bit offset 0.
- Sequence counter/alive counter: An 8-bit sequence number is explicitly sent and incremented at every transmission request. The Sequence counter starts with bit offset 16.
- Data In/Data Out: An application counter that is increased periodically every second. The counter starts at default value 0 after flashing the ECU, and it can be set to a certain value. It starts with bit offset 24.
- Data ID: A system-wide unique 16-bit data ID is implicitly sent for every port data element. The Data Id used for the transmition is 0x14B and for reception 0x14A.

The layout of the message is as follows:

Byte 0 + Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**
CRC	Sequence Counter	Data Out/Data In	0xFF	0xFF	0xFF	0xFF

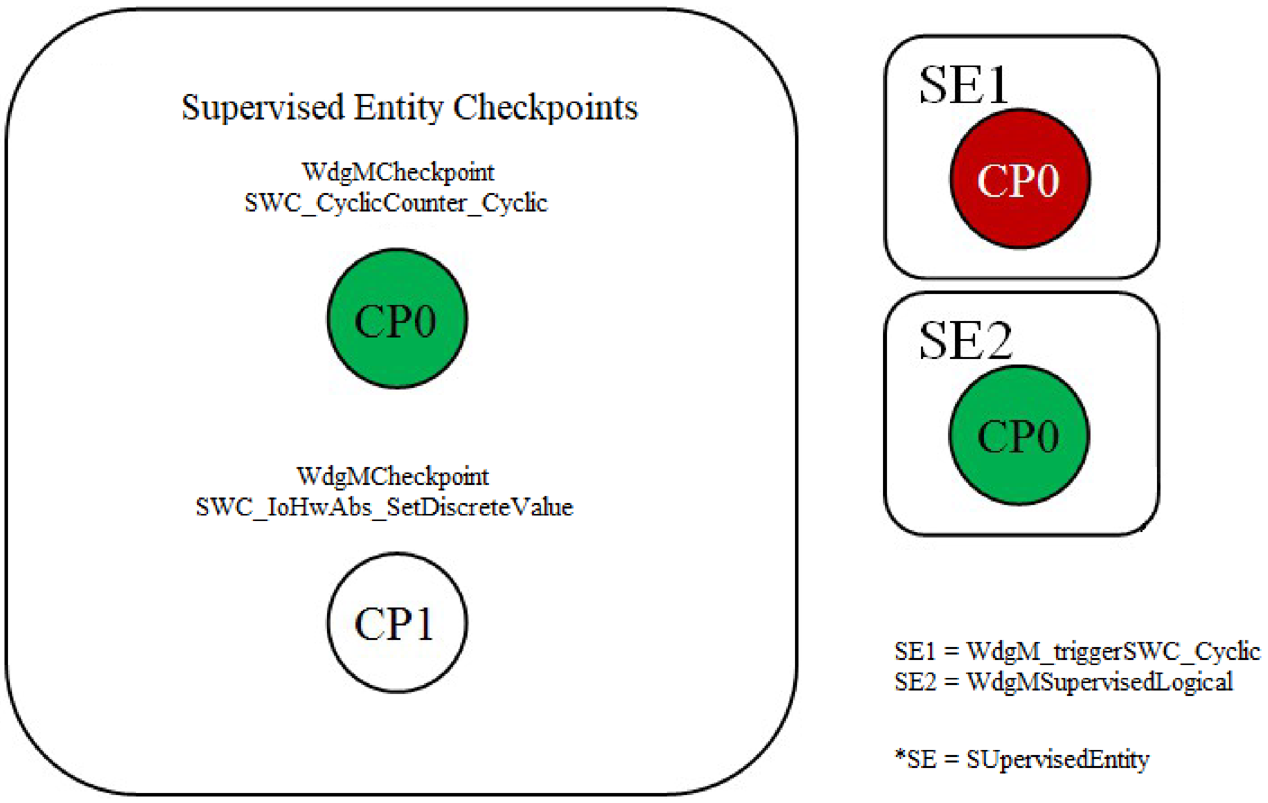
Table 2.4. E2E message layout

2.2.2.9. Safety Time and Execution Protection (TimEMP)

In the TimEPM, the safety mechanism consists of adding a Checkpoint (CP) to the control flow of a SW-C, BSW or CDD under the observation and evaluation of this Checkpoint at run time for a specified reference. In the TimEPM, the SW-C, BSW or CDD under observation is called Supervised Entity (SE). The TimEPM implementation from Solution for Essentials provides the following safety mechanisms:

- Alive Supervision is part of Temporal Prog Flow Monitoring. Alive Supervision verifies if the expected number of Alive Indications that are reported by the Checkpoint of a Supervised Entity are within specified limits. The TimEPM periodically checks if a Supervised Entity is not run too frequently or too rarely.
- Logical Supervision verifies if the configured Graphs are executed. This means that every time a Checkpoint of a Supervised Entity is reported, the TimEPM verifies if this Checkpoint is an authorized successor of the previously reported Checkpoint.

Two Checkpoints (0_SWC_CyclicCounter_Cyclic and 1_SWC_IoHwAbs_SetDiscreteValue) were configured. In the application code, the two Checkpoints are called in the corresponding functions SWC_CyclicCounter_Cyclic and SWC_IoHwAbs_SetDiscreteValue. Logical Supervision is configured for WDGM_NORMAL_OPERATION mode. The switch to NORMAL_OPERATION mode is done by Supervision Callouts with the function Supervisor_WdgM_GetExpectedWdgMModeCallout().


Supervised Entity Checkpoints
WdgMCheckpoint
SWC_CyclicCounter_Cyclic
CP0
WdgMCheckpoint
SWC_IoHwAbs_SetDiscreteValue
CP1
SE1
CP0
SE2
CP0
SE1 = WdgM_triggerSWC_Cyclic
SE2 = WdgMSupervisedLogical
*SE = SUpervisedEntity

Supervision Callouts (only available with TimE license) were enabled inside the WdgM plugin. This modification has an impact on WdgM main function that will not be mapped inside the Rte Event. In order to work, a new function (WdgM_MainFunctionTriggering) was created with a call periodicity of 10ms that calls the WdgM_-MainFunction.

2.2.3. Startup Package Application IP

2.2.3.1. Module usage

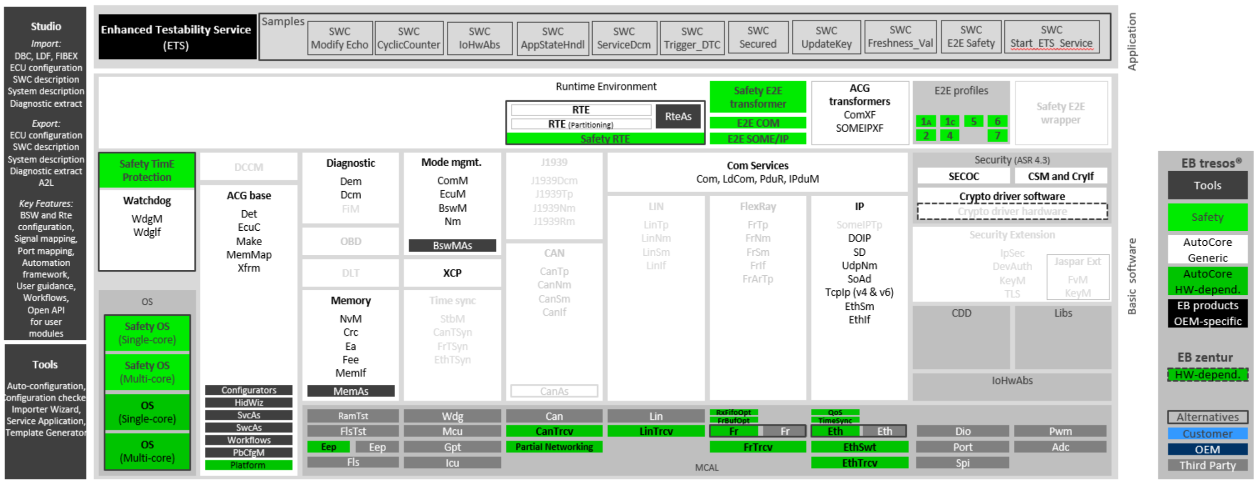


Figure 2.4. Module usage in Startup Package Application IP

2.2.3.2. Software components

The SWC composition consists of the following components:

- SWC_CyclicCounter
- SWC_ModifyEcho
- SWC_IoHwAbs
- SWC_AppStateHandler
- ServiceSwComponentDcm
- SWC_Trigger_DTC
- SWC_Secured

- SWC_UpdateKey
- SWC_Freshness_Value
- SWC_SomeIp_TransportProtocol
- SWC_SomeIp_TLV
- SWC_SomeIp_TLV_ArrayMember
- SWC_Start_ETS_Service

2.2.3.3. UDP and TCP communication

The Startup Package Application IP supports UDP and TCP communication on IPv6 or IPv4 over Ethernet with a speed of 100 Mbps.

Startup Package Application IP is delivered with IPv6 enabled, the IP version can be changed to IPv4 by executing Full_sequence_with_splittables_Ipv4 multitask wizard, or IPv6 configuration can be restored by execution of Full_sequence_with_splittables multitask wizard.

- On IPv6 the static unicast TcpIp address fd53:7cb8:383:eb03:0:0:0:104, and multicast TcpIp ff02:0:0:0:0:0:0:1 addresses are supported.
- On IPv4 the static unicast TcpIp address 192.168.88.73, the multicast TcpIp address 235.2.3.5 and broadcast TcpIp 255.255.255.255 addresses are supported.
- Network management is supported by the ECU. It sends out an NM message on the multicast IP address on port 20000, and receives NM messages on the ECU IP address on port 20000.

After startup, the ECU offers:

- An application counter that is increased periodically every second. The counter starts at default value 0.
- A Dio channel usually connected to one of the available board LED lights, and which is toggled periodically every second.
- Support for application counter set/reset (Pdu_CounterIn with payload value less than 0xF0), pseudo development error (Pdu_CounterIn with payload value equal to 0xF1), and ECU shut-down (Pdu_CounterIn with payload value equal to 0xFF) commands.

2.2.3.4. Diagnostics

DoIp according to ISO13400-2:2011 is implemented. One tester with address 0x0EF8 is supported. Physical addressing is supported on target address 0x1148 and functional addressing on 0xEEF8 .

For a description of the supported services, see Section 2.2.4.1.1, “Unified Diagnostic Services (UDS)”.

2.2.3.5. Secure On-Board communication over IP (SecOC)

Secure Onboard Communication provides protection on bus level. Secured PDU includes the FV (Freshness Value) and the MAC (Message Authentication Code) that ensure the integrity of the signal. The ECU receives a secured PDU on UDP port 7100 and respond with a secured PDU on UDP port 7100. The secured PDU has the following attributes:

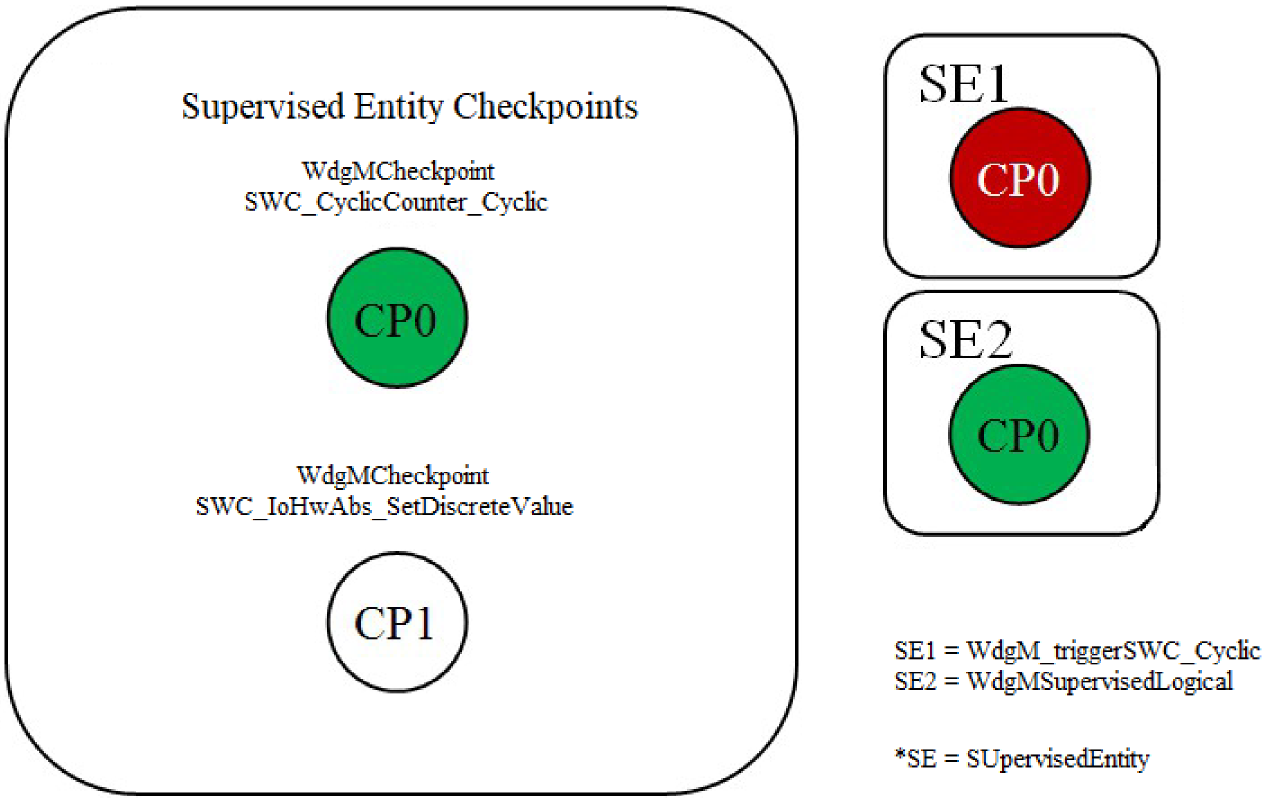
- Authentification Algorithm = CMAC/AES-128. This parameter defines the authentication algorithm used for authentication verification.
- Authentic PDU = 9 bytes. This parameter defines the length of the authentic PDU (8 bytes represent the SomeIp header + 1 byte represents the data that is sent).
- Message Authentication Code = 4 bytes. This parameter defines the length in bits of the authentication code.
- Truncated Freshness Value Length = 1 byte. This parameter defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU.

2.2.3.6. Safety Time and Execution Protection (TimEMP)

In the TimEPM, the safety mechanism consists of adding a Checkpoint (CP) to the control flow of a SW-C, BSW or CDD under the observation and evaluation of this Checkpoint at run time for a specified reference. In the TimEPM, the SW-C, BSW or CDD under observation is called Supervised Entity (SE). The TimEPM implementation from Solution for Essentials provides the following safety mechanisms:

- Alive Supervision is part of Temporal Prog Flow Monitoring. Alive Supervision verifies if the expected number of Alive Indications that are reported by the Checkpoint of a Supervised Entity are within specified limits. The TimEPM periodically checks if a Supervised Entity is not run too frequently or too rarely.
- Logical Supervision verifies if the configured Graphs are executed. This means that every time a Checkpoint of a Supervised Entity is reported, the TimEPM verifies if this Checkpoint is an authorized successor of the previously reported Checkpoint.

Two Checkpoints (0_SWC_CyclicCounter_Cyclic and 1_SWC_IoHwAbs_SetDiscreteValue) were configured. In the application code, the two Checkpoints are called in the corresponding functions SWC_CyclicCounter_Cyclic and SWC_IoHwAbs_SetDiscreteValue. Logical Supervision is configured for WDGM_NORMAL_OPERATION mode. The switch to NORMAL_OPERATION mode is done by Supervision Callouts with the function Supervisor_WdgM_GetExpectedWdgMModeCallout().


Supervised Entity Checkpoints
WdgMCheckpoint
SWC_CyclicCounter_Cyclic
CP0
WdgMCheckpoint
SWC_IoHwAbs_SetDiscreteValue
CP1
SE1
CP0
SE2
CP0
SE1 = WdgM_triggerSWC_Cyclic
SE2 = WdgMSupervisedLogical
*SE = SUpervisedEntity

Supervision Callouts (only available with TimE license) were enabled inside the WdgM plugin. This modification has an impact on WdgM main function that will not be mapped inside the Rte Event. In order to work, a new function (WdgM_MainFunctionTriggering) was created with a call periodicity of 10ms that calls the WdgM_-MainFunction.

2.2.3.7. SOME/IP

The SOME/IP supports UDP communication on IPv6 or IPv4 over Ethernet.

2.2.3.7.1. SOME/IP Service Discovery

The Service Discovery Protocol communicates the functional entities called services in the in-vehicle communication and controls the send behavior of event messages. This enables to send only event messages to receivers requiring them (Publish/Subscribe). The solution is also known as SOME/IP-SD (Scalable service-Oriented MiddlewarE over IP - Service Discovery).

After startup, the ECU offers:

- One server service where the incremented counter Pdu_CounterOut can be read.

- One client service subscription if an offer is detected. With this service, the internal counter Pdu_CounterIn can be written.
- Service discovery is supported by the ECU. It provides a server service with ID 0xEB on UDP port 50000, and supports a client service with ID 0xEC on UDP port 60000.

2.2.3.7.2. SOME/IP TP

- The task of the SOME/IP TP module is to segment SOME/IP packets that do not fit into one single UDP packet. On the reception side, it re-assembles the received SOME/IP segments.
- The ECU re-assembles the SOME/IP segments on UDP port 42000 with message ID 0xAB000C, and respond UDP port 42000 with the same segments that were received. The maximum Npdu length is set on 3488 and Spdu length on 6000.

2.2.3.7.3. SOME/IP TLV

The extended serialization for data structures in SOME/IP with tag/length/value encoding (TLV) provides an efficient way for transferring large data structures without mapping each element of the structure to a signal.

- The SOME/IP TLV encoding provide two members:
  - Optional member on UDP port 43000 with message ID 0xAB010A and respond on the port 42000 with message ID 0xAB010C, and the same payload that was received.
  - Array member on UDP port 43000 with message ID 0xAB010B and respond on the port 42000 with message ID 0xAB010D, and the same payload that was received.
- SOME/IP TLV layout message:

SOME/IP Header								**TLV Extention**	
Message ID	Length	Request ID	Session ID	Protocol version	Interface version	Message type	Return Code	TAG	Length, Value
4 bytes	4 bytes	2 bytes	2 bytes	1 byte	1 byte	1 byte	1 byte	2 bytes	Payload

Table 2.5. SOME/IP TLV layout message

2.2.4. Common features

- Section 2.2.4.1, “Diagnostic communication protocols”: provides a list of diagnostic communication protocols.
- Section 2.2.4.2, “CPU Load measurement”: provides information about the CPU Load.

2.2.4.1. Diagnostic communication protocols

The EB tresos Solutions for Essentials implement the following diagnostic communication protocols:

- Section 2.2.4.1.1, "Unified Diagnostic Services (UDS)": Unified Diagnostic Services
- Section 2.2.4.1.2, "On-Board Diagnostics (OBD)": On-Board Diagnostics

2.2.4.1.1. Unified Diagnostic Services (UDS)

The EB tresos Solutions for Essentials implement UDS according to ISO 14229-1:2013 over different transport protocols.

The following services are supported:

- Section 2.2.4.1.1.1.1, "DiagnosticSessionControl (0x10) service"
- Section 2.2.4.1.1.1.2, "ECUReset (0x11) service"
- Section 2.2.4.1.1.1.3, "SecurityAccess (0x27) service"
- Section 2.2.4.1.1.1.4, "CommunicationControl (0x28) service"
- Section 2.2.4.1.1.1.5, "TesterPresent (0x3E) service"
- Section 2.2.4.1.1.1.6, "ControlDTCSetting (0x85) service"
- Section 2.2.4.1.1.1.7, "ResponseOnEvent (0x86) service"
- Section 2.2.4.1.1.1.8, "ReadDataByIdentifier (0x22) service"
- Section 2.2.4.1.1.1.9, "WriteDataByIdentifier (0x2E) service"
- Section 2.2.4.1.1.1.10, "ClearDiagnosticInformation (0x14) service"
- Section 2.2.4.1.1.1.11, "ReadDTCInformation (0x19) service"
- Section 2.2.4.1.1.1.12, "RoutineControl (0x31) service"

For a description of the supported DIDs, refer to Section 2.2.4.1.1.2, "Diagnostic DID definitions".

For a description of the supported routines, refer to Section 2.2.4.1.1.3, "Diagnostic Routine definitions".

2.2.4.1.1.1. Diagnostic Services

2.2.4.1.1.1.1. DiagnosticSessionControl (0x10) service

Name	ID	Security	Session
Default	0x01		
Programming	0x02		

Name	ID	Security	Session
Extended	0x03		

Table 2.6. DiagnosticSessionControl subservice configuration

2.2.4.1.1.1.2. ECUReset (0x11) service

Name	ID	Security	Session
hardReset	0x01		

Table 2.7. ECUReset subservice configuration

2.2.4.1.1.1.3. SecurityAccess (0x27) service

Name	ID	Security	Session
Level1_Seed_requestSeed	0x01		Extended, Programming
Level1_Seed_sendKey	0x02		Extended, Programming
Level3_Seed_requestSeed	0x03		Extended
Level3_Seed_sendKey	0x04		Extended

Table 2.8. SecurityAccess subservice configuration

2.2.4.1.1.1.4. CommunicationControl (0x28) service

Name	ID	Security	Session
enableRxAndTx	0x00		Extended, Programming
enableRxAndDisableTx	0x01		Extended, Programming
disableRxAndEnableTx	0x02		Extended, Programming
disableRxAndTx	0x03		Extended, Programming

Table 2.9. CommunicationControl subservice configuration

2.2.4.1.1.1.5. TesterPresent (0x3E) service

Name	ID	Security	Session
zeroSubFunction	0x00		Default, Extended

Table 2.10. TesterPresent subservice configuration

2.2.4.1.1.1.6. ControlDTCSetting (0x85) service

Name	ID	Security	Session
on	0x01		Extended, Programming
off	0x02		Extended, Programming

Table 2.11. ControlDTCSetting subservice configuration

2.2.4.1.1.1.7. ResponseOnEvent (0x86) service

Name	ID	Security	Session
ISOSAEReserved	0x04		Default, Extended
vehicleManufacturerSpecific	0x40		Default, Extended
vehicleManufacturerSpecific	0x45		Default, Extended

Table 2.12. ResponseOnEvent subservice configuration

Name	ID	Service To respond
DcmDspRoeEvent	0x00	0xF186 DcmDspDid_ActiveDiagnosticInformation

Table 2.13. ResponseOnEvent type records.

2.2.4.1.1.1.8. ReadDataByIdentifier (0x22) service

Name	ID	Security	Session
DcmDspDid_ActiveDiagnosticInformation	0xf186		
DcmDspDid_NVM_TEST_BLOCK	0xf191		
DcmDspDid_VehicleIdentificationNumber_Block	0xf190		
DcmDspDid_CpuLoad	0xf200		

Table 2.14. Supported Data Identifier for read

For the DID definitions, refer to Section 2.2.4.1.1.2, “Diagnostic DID definitions”.

2.2.4.1.1.1.9. WriteDataByIdentifier (0x2E) service

Name	ID	Security	Session
DcmDspDid_NVM_TEST_BLOCK	0xf190		

Table 2.15. Supported Data Identifier for write

For the DID definitions, refer to Section 2.2.4.1.1.2, “Diagnostic DID definitions”.

2.2.4.1.1.1.10. ClearDiagnosticInformation (0x14) service

Name	ID	Security	Session
none			Default, Extended

Table 2.16. ClearDiagnosticInformation service configuration

2.2.4.1.1.1.11. ReadDTCInformation (0x19) service

Name	ID	Security	Session
reportNumberOfDTCByStatus-Mask	0x01		Default, Extended
reportDTCByStatusMask	0x02		Default, Extended
reportDTCSnapshotRecordBy-DTCNumber	0x04		Default, Extended
reportDTCExtDataRecordByDTC-Number	0x06		Default, Extended
reportSupportedDTC	0x0A		Default, Extended
reportDTCFaultDetectionCounter	0x14		Default, Extended
reportUserDefMemoryDTCBySta-tusMask	0x17		
reportUserDefMemoryDTCSnap-shotRecordByDTCNumber	0x18		
reportUserDefMemoryDTCExt-DataRecordByDTCNumber	0x19		

Table 2.17. ReadDTCInformation subservice configuration

2.2.4.1.1.1.12. RoutineControl (0x31) service

Name	ID	Security	Session
DIAGNOSTIC_LOOPBACK	0x0100		Extended
DIAGNOSTIC_TRIGGER_DTC	0x0101		Extended

Table 2.18. RoutineControl service configuration

2.2.4.1.1.2. Diagnostic DID definitions

Name	ID	Data length	Description
DcmDspDid_ActiveDiagnosticInformation	0xf186	1 byte	Contains the active diagnostic of the ECU. Coding is according to session definition in Table 2.6, “DiagnosticSessionControl subservice configuration”
DcmDspDid_NVM_TEST_BLOCK	0xf191	17 byte	Contains 17 bytes which are stored in the NVM.
DcmDspDid_VehicleIdentification-Number_Block	0xf190	17 byte	Vehicle Identification Number.
DcmDspDid_CpuLoad	0xf200	30 byte	Contains the CPU Load and Start-up time measurement

Table 2.19. Supported DID definitions

2.2.4.1.1.3. Diagnostic Routine definitions

Name	ID	Data length	Description
DIAGNOSTIC_LOOP-BACK	0x0100	Start Routine input signal: Variable, max 1024 byte Start Routine output signal: Variable, max 1024 byte	Returns the data which it receives.
DIAGNOSTIC_TRIG-GER_DTC	0x0101	Start Routine input signal: 1 byte, not used Start Routine output signal: 1 byte, not used	Set event to trigger DTC 0x000001

Table 2.20. Routine definitions

2.2.4.1.2. On-Board Diagnostics (OBD)

The EB tresos Solutions for Essentials implements OBD according to ISO-15031-5-2015 over different transport protocols.

The following services are supported:

- Request current powertrain diagnostic data (0x01)

- Request powertrain freeze frame data (0x02)
- Request emission-related diagnostic trouble codes (0x03)
- Clear/reset emission-related diagnostic information (0x04)
- Request emission-related diagnostic trouble codes detected during current or last completed driving cycle (0x07)
- Request vehicle information (0x09)
- Request emission-related diagnostic trouble codes with permanent status (0x0A)

2.2.4.1.2.1. PID definitions

ID	Data length	Description	Supported in Services
0x01	4 byte	Implemented in DEM according to Autosar.	0x01, 0x02
0x03	4 byte	Dummy implementation 0x00 is returned in each byte.	0x02
0x41	4 byte	Implemented in DEM according to Autosar.	0x01

Table 2.21. Supported PID definitions

2.2.4.1.2.2. Infotype definitions

ID	Data length	Description	Supported in Services
0x01	1 byte	Number of messages returned by Infotype 0x02.	0x09
0x02	17 byte	Vehicle Identification Number.	0x09

Table 2.22. Supported Infotype definitions

2.2.4.2. CPU Load measurement

The CPU Load measurement in not supported on WINDOWS(WIN32X86)

The CPU Load measurement provides the CPU currentLoad and peakLoad: the time spent in the idle thread is idle time and any other time is busy time. The measured average load for the last 1-second window is stored in CPU Load current, and the highest load is stored in CPU load peak.

You can read this value via diagnostics using DID 0xF200.

CPU Load measurements	Measurement location in payload of DID 0xF200	Unit measurement
CPU Load current	byte 1	%
CPU Load peak	byte 2	%

Table 2.23. CPU load

2.2.4.2.1. Potential failure

If you have 100% load - for example a low priority background task that never terminates and never waits for an event - the idle task never runs and the data structures is not initialized.

If you have almost 100% load but the system manages to get occasionally to the idle thread, the CPU Load might show a stale value.

2.2.4.2.2. Safety

WARNING ⚠

Do not use this software in a production system

The CPU Load measurement software is provided "as it is". It is intended as a development aid.

Even though it runs in a thread, it requires certain privileges to enable and disable interrupts. If it spends an excessive amount of time with interrupts disabled (as it could happen if control returns to the idle thread after a long interval of busy time, for example), the interrupts might remain disabled for a long time and interfere with the normal operation of the system.

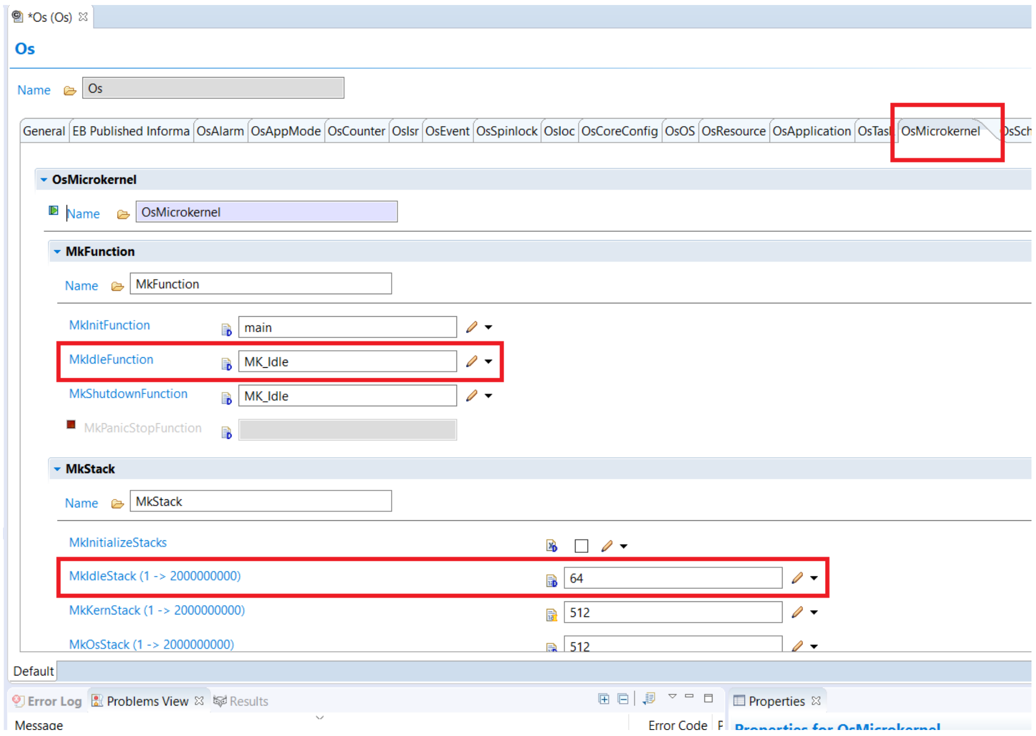
2.2.4.2.3. Removing CPU Load measurement

Step 1
Open EB tresos Studio.

Step 2
Open the OS and select the **OsMicrokernel** tab.

Step 3
In **MkFunction** field, change MkIdleFunction to MK_Idle.

Step 4
In **MkStack** field, set the MkIdleStack value to 64 for a default configuration.



Os

Name: Os

OsMicrokernel

Name: OsMicrokernel

MkFunction

Name: MkFunction

MkInitFunction: main

MkIdleFunction: MK_Idle

MkShutdownFunction: MK_Idle

MkPanicStopFunction:

MkStack

Name: MkStack

MkInitializeStacks: ☐

MkIdleStack (1 -> 2000000000): 64

MkKernStack (1 -> 2000000000): 512

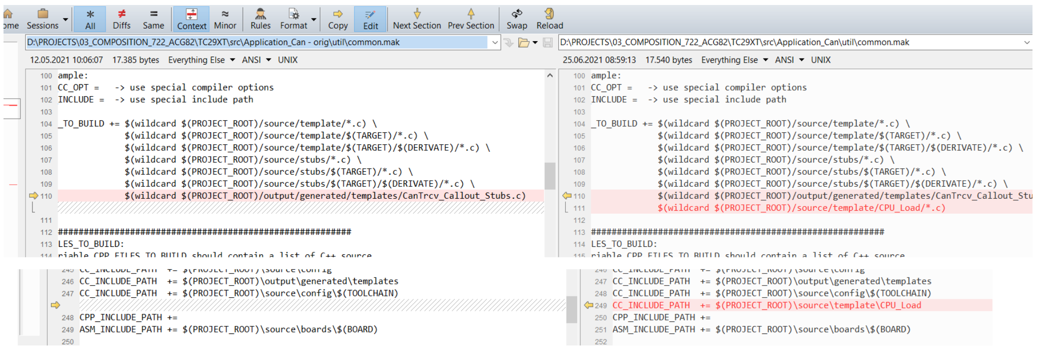
MkOsStack (1 -> 2000000000): 512

Properties for OsMicrokernel

Figure 2.5.

Step 5
Remove the CPU_Load folder from `source\template`.

Step 6
Update the `common.mak` file from the util folder so that the `CPU_Load` file is not taken into account.



D:\PROJECTS\03_COMPOSITION_722_ACG82\TC29XT\src\Application_Can - orig\util\common.mak

12.05.2021 10:06:07 17.385 bytes Everything Else ANSI UNIX

```
ample:
CC_OPT =   -> use special compiler options
INCLUDE =  -> use special include path

_TO_BUILD += $(wildcard $(PROJECT_ROOT)/source/template/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/template/$(TARGET)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/template/$(TARGET)/$(DERIVATE)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/$(TARGET)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/$(TARGET)/$(DERIVATE)/*.c) \
             $(wildcard $(PROJECT_ROOT)/output/generated/templates/CanTrcv_Callout_Stubs.c)


##########################################################
LES_TO_BUILD:
riable CPP_FILES_TO_BUILD should contain a list of C++ source
```

```
CC_INCLUDE_PATH  += $(PROJECT_ROOT)\output\generated\templates
CC_INCLUDE_PATH  += $(PROJECT_ROOT)\source\config\$(TOOLCHAIN)

CPP_INCLUDE_PATH +=
ASM_INCLUDE_PATH += $(PROJECT_ROOT)\source\boards\$(BOARD)
```

D:\PROJECTS\03_COMPOSITION_722_ACG82\TC29XT\src\Application_Can\util\common.mak

25.06.2021 08:59:13 17.540 bytes Everything Else ANSI UNIX

```
ample:
CC_OPT =   -> use special compiler options
INCLUDE =  -> use special include path

_TO_BUILD += $(wildcard $(PROJECT_ROOT)/source/template/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/template/$(TARGET)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/template/$(TARGET)/$(DERIVATE)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/$(TARGET)/*.c) \
             $(wildcard $(PROJECT_ROOT)/source/stubs/$(TARGET)/$(DERIVATE)/*.c) \
             $(wildcard $(PROJECT_ROOT)/output/generated/templates/CanTrcv_Callout_Stu
             $(wildcard $(PROJECT_ROOT)/source/template/CPU_Load/*.c)

##########################################################
LES_TO_BUILD:
riable CPP_FILES_TO_BUILD should contain a list of C++ source
```

```
CC_INCLUDE_PATH  += $(PROJECT_ROOT)\output\generated\templates
CC_INCLUDE_PATH  += $(PROJECT_ROOT)\source\config\$(TOOLCHAIN)
CC_INCLUDE_PATH  += $(PROJECT_ROOT)\source\template\CPU_Load
CPP_INCLUDE_PATH +=
ASM_INCLUDE_PATH += $(PROJECT_ROOT)\source\boards\$(BOARD)
```

Figure 2.6.

2.2.4.3. Startup time measurement

The Startup time provides time stamp measurement when the functions listed in the following table are called.

Function for which time stamp is measured	Measurement location in payload	Unit measurement
Eb_Intgr_BswM_Hook_OnStartupTwoA	bytes 3 to 6	microseconds
Eb_Intgr_BswM_Hook_OnStartupTwoB	bytes 7 to 10	microseconds
Eb_Intgr_BswM_Hook_OnRunTwo	bytes 11 to 14	microseconds
Eb_Intgr_BswM_Hook_OnPostRun	bytes 15 to 18	microseconds
Eb_Intgr_BswM_Hook_OnPrpShutdown	bytes 19 to 22	microseconds
Eb_Intgr_BswM_Hook_OnGoOffOneA	bytes 23 to 26	microseconds
Eb_Intgr_BswM_Hook_OnGoOffOneB	bytes 27 to 30	microseconds

Table 2.24. Startup Time Call Functions

The time stamps value can be read out via diagnostics using DID 0xF200. Each value is using 4 bytes starting with byte 3 from the payload.

Based on this time stamp, the time passed betwen two hook function calls can be calculated. The value of the time stamp is measured in microseconds.

3. Step by step for your project

EB tresos Solutions for Essentials consists of basic software modules and additional Software Components (SWCs) according to the AUTOSAR standard. The Software Components descriptions and source files, and the system description files have a similar behavior in all projects. The difference between the projects is based on the type of underlying communication.

EB tresos Solutions for Essentials provides a configuration example of the AUTOSAR stack provided by EB. The product is composed of EB tresos Studio projects that you can build, load into the ECU target, and execute. The workflow is delivered in EB tresos Studio configuration project `.workflows\Workflow.xml` that you can adapt to your needs.

We recommend that you have a good knowledge of AUTOSAR, EB tresos Studio and EB tresos AutoCore before you perform the steps described in this chapter.

For an introduction to the EB tresos Studio workflows:

- Use the tutorials available on our website (*Tutorials* tab).
- Refer to EB tresos Studio user guide, chapter *Workflows view* available in `$TRESOS_BASE/doc/2.0_-EB_tresos_Studio/2.1_Studio_documentation_users_guide.pdf`.

3.1. Setting up and building your project

This section explains how to setup and build the Startup Package Application project. The same procedures are applied to setup and build the Startup Package Bootloader.

3.1.1. Prerequisites

1. Install the correct version of the required toolchain.
   - Compiler toolchain
   - Debugger toolchain
2. Install EB tresos Studio license.
3. Install the workspace: the Startup Package Application workspace is delivered as an uip package and is installed with the EB tresos Studio installer.
4. Configure the compiler path: open `templates/<project name>/util/launch_cfg.bat`, and change the value of `TOOLPATH_COMPILER` to the installation location of your compiler.

3.1.2. Executing batch files for the project

Batch files are available to execute automatically some project steps. The files are in the `templates` folder. Some batch files have the `prj` in their name to represent a project on which they apply.

- __Start_Tresos.bat: starts EB tresos Studio and automatically imports the projects if not previously imported.
- `prj`_01_RunSysRestoreWizard.bat: performs a complete system update based on the supplementary files, it uses `Execute multiple tasks Unattended Wizards`.
- `prj`_02_Generate.bat: generates tresos application.
- `prj`_03_Clean.bat: cleans the project using the build environment.
- `prj`_04_Build.bat: compiles the project using the build environment.
- `prj`_05_Debug.bat: starts the configured debugger with the project loaded inside.

3.1.3. Importing the project

You must import the project into your EB tresos Studio workspace (for example `C:/EB/tresos`).

Step 1
To import the Startup Package Application, go to `tresos_gui.exe`, and double-click it to open EB tresos Studio.

Step 2
In the **File** menu, select **Import**.

Step 3
In the **Import** window, select **Existing Projects into Workspace**, and then select **Next**.

Step 4
Select **Select root directory**, and browse to `templates/<StartupPackage>`.

Step 5
Select one Startup Package Application, and then select **OK**.

Step 6
In the **Import Projects** window, select **Copy projects into workspace** to copy the project into the default workspace.

Step 7
Select **Finish**.

In the **Project Explorer** view, you can:

- Open the project by double-clicking on the project name.
- Open the configuration by clicking .

3.1.4. Changing compilers for the project

You can change compilers if your architecture supports multiple compilers.

NOTE

There is no compiler to configure for EB tresos WinCore.
EB tresos WinCore is delivered with the MinGW development environment, which is automatically installed when you build the Startup Package Application. The MinGW development environment also contains a compiler. You only need to follow these instructions to configure a different compiler.

Step 1
Locate the project folder in the workspace directory.

Step 2
Open `util/launch_cfg.bat` in a text editor.

Step 3
Set the variable `TOOLCHAIN` to the name of the toolchain. For example `set TOOLCHAIN=dcc`.

Step 4
Set the environment variable `TOOLPATH_COMPILER` to the actual installation path of your compiler. For example: `set TOOLPATH_COMPILER=C:/WindRiver/diab/5.5.1.0`.

Step 5
Save the file.

3.1.5. Building the project

Step 1
Ensure EB tresos Studio is not running.

Step 2
In the `<StartupPackage>/util` directory, double-click `launch.bat`. The first startup of `launch.bat` takes some time.

Step 3
Type `make generate`, and hit the **Enter** key to generate the project.

Step 4
Type `make` (or `make -j -O` for compiling in parallel), and hit the **Enter** key to build the application.

Step 5
If you work with EB tresos WinCore, the MinGW development environment is being installed with the build of the Startup Package Application.

The resulting binary file is in the `<StartupPackage>/output/bin` directory.

3.1.6. Testing the project

3.1.6.1. Flashing the projects to target HW

Use the Lauterbach debugger or any other flash tool to perform the download to the target.

Step 1
Connect the Lauterbach debugger to the target and start the Trace32.

Step 2
Execute the debugger script `<StartupPackage>\util\flash.cmm` to load the corresponding Startup Package to the target.

Step 3
Select **Go** in the debugger window. The Application starts by default.

If a Bootloader is available for the given application, the Bootloader is loaded automatically before the application.

Step 4
Select **Go** in the debugger window. The Bootloader starts by default.

It is recommended to verify the basic functionality of the application after the first download to the target.

3.1.6.2. Smoke test for Bootloader Can

Programming session test.

- Send a CAN message with ID **18DAEBF1h** and payload value **0x02 0x10 0x02 0x00 0x00 0x00 0x00 0x00** to jump in Bootloader.
- An acknowledge response returns from ECU with CAN ID 18DAF1EBh and payload value **0x06 0x50 0x02 0x00 0x32 0x01 0xF4 0x00**.

3.1.6.3. Smoke test for Application Can

After the start-up, the ECU shows:

- A Dio channel that is toggled periodically every second. This channel is usually connected to one of the available board LED lights.
- An application counter that is periodically increased every second is received from ECU with CAN ID 110h for CAN. Check the value is incremented every second.
- Reset the counter value received with CAN ID **110h** by sending a message with CAN ID **100h** with payload value less than 0xF0. For example, send **0x00** and the counter will restart from 0x00.

- An application counter that is increased periodically every second is received from ECU with CAN ID **00000111h for CAN FD**. Check the value is increment every second.
- The application counters reception is needed to keep the **ECU awake**: to do so, a CAN NM message must be sent to enable Partial Network Clusters.
- For example, the message **0x56 0x51 0x03 0x00 0x00 0x00 0x00 0x00** must be sent to the ECU every **200ms** with CAN ID **0x210.** This enables both supported networks for CAN and CAN FD.
- Enable CAN network and disable CAN FD network by sending the message **0x56 0x51 0x01 0x00 0x00 0x00 0x00 0x00**. As the result, the counter from CAN FD stops refreshing, and only CAN counter is updated.
- Check if ECU falls asleep when CAN NM message does not send anything anymore. The counter received with CAN ID 110h and CAN ID 00000111h will stop refreshing.

3.1.6.4. Testing basic functionalities for Application Can

3.1.6.4.1. Session control (Default session)

Step 1
Send a message with ID **18DAEBF1h** and payload value **0x02 10 01 00 00 00 00 00**.

Step 2
A response is **received** with ID **18DAF1EBh** and payload value **0x06 50 01 00 14 00 C8 00**.

3.1.6.4.2. Read DID (Read VIN)

Using CAN: this represents the first frame. To get the rest of the response, transport protocol frames must be sent.

Step 1
Send a message with ID **18DAEBF1h** and payload value **0x03 22 F1 90 00 00 00 00** .

Step 2
A response is received with ID **18DAF1EBh** and payload value **0x10 14 62 F1 90 AB AB AB**.

Using CAN FD

Step 1
Send a message with ID **18DAEBF2h** and payload value **0x03 22 F1 90 00 00 00 00** .

Step 2
A response is received with ID **18DAF2EBh** and payload value **0x00 14 62 F1 90 AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB AB 00 00**.

3.1.6.5. Smoke test for Application Fr

The Startup Application FlexRay supports FlexRay communication with static and dynamic slot IDs.

After the start-up, the ECU shows:

- A Dio channel (usually connected to one of the available board LED lights) that is toggled periodically every second.
- An application counter is received from ECU in slot ID10 in byte 0 of the FlexRay frame, and increased periodically every second.
- Reset the counter value by sending a new value stored Slot ID 1 Byte 0 of FlexRay frame. For example, send **0x00** and the counter will restart from 0x00.

3.1.6.6. Testing basic functionalities for Application Fr

3.1.6.6.1. Session control (default session)

Step 1
Send a message with Slot ID **200** and payload value **0 127 0 223 64 2 0 2 16 1**.

Step 2
A response is received in Slot ID **202** and payload **0 128 0 127 64 6 0 6 80 1 0 20 0 200**.

3.1.6.6.2. Read DID (Read VIN)

Step 1
Send a message with Slot ID **200** and payload value **0 127 0 223 64 3 0 3 34 241 144**.

Step 2
A response is received in Slot ID **202** and payload **0 128 0 127 64 20 0 20 98 241 144 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171**.

3.1.6.7. Smoke test for Application Eth

The StartUp Application Eth supports UDP and TCP communication on IPv6 or IPv4 over Ethernet with a speed of 100 Mbps. The StartUp Application Eth is delivered with IPv6 enabled.

After the start-up, the ECU shows:

- A Dio channel (usually connected to one of the available board LED lights) that is toggled periodically every second.
- SOME/IP Service Discovery Protocol offers are received from ECU.

3.1.6.8. Testing basic functionalities for Application Eth

Eth ITA tests must be run for testing the functionality.

4. EB tresos modules for Essentials

The ETS (Enhanced Testability Service) module is used to test that the software chain from Ethernet port up to SOME/IP Transformer correctly work.

4.1. ETS (Enhanced Testability Service)

4.1.1. Background information

The protocol parts addressed by the Enhanced Testability Service include:

- SOME/IP Stack
- Service Discovery
- Serialization
- Remote Procedure Call

The ETS module enables different categories of tests, for example when used in component testing scenarios for devices under test (DUTs):

- Positive tests (testing using valid messages)
- Negative tests (testing error handling)
- Load testing
- Regression testing

The external tester sends various types tests to the ETS module, and the ETS module responds back in a specified way (echoing back parameters, notifying different events, get and set fields or parameters, triggers timers etc.).

The ETS module also manages specific service requests (SubscribeEventgroup, client server calls, check supported byte order), and passes them forward. There should always be an expected result in ETS response to the Tester data that are sent.

Figure 4.1, “ETS interactions” shows the module interactions in the system.

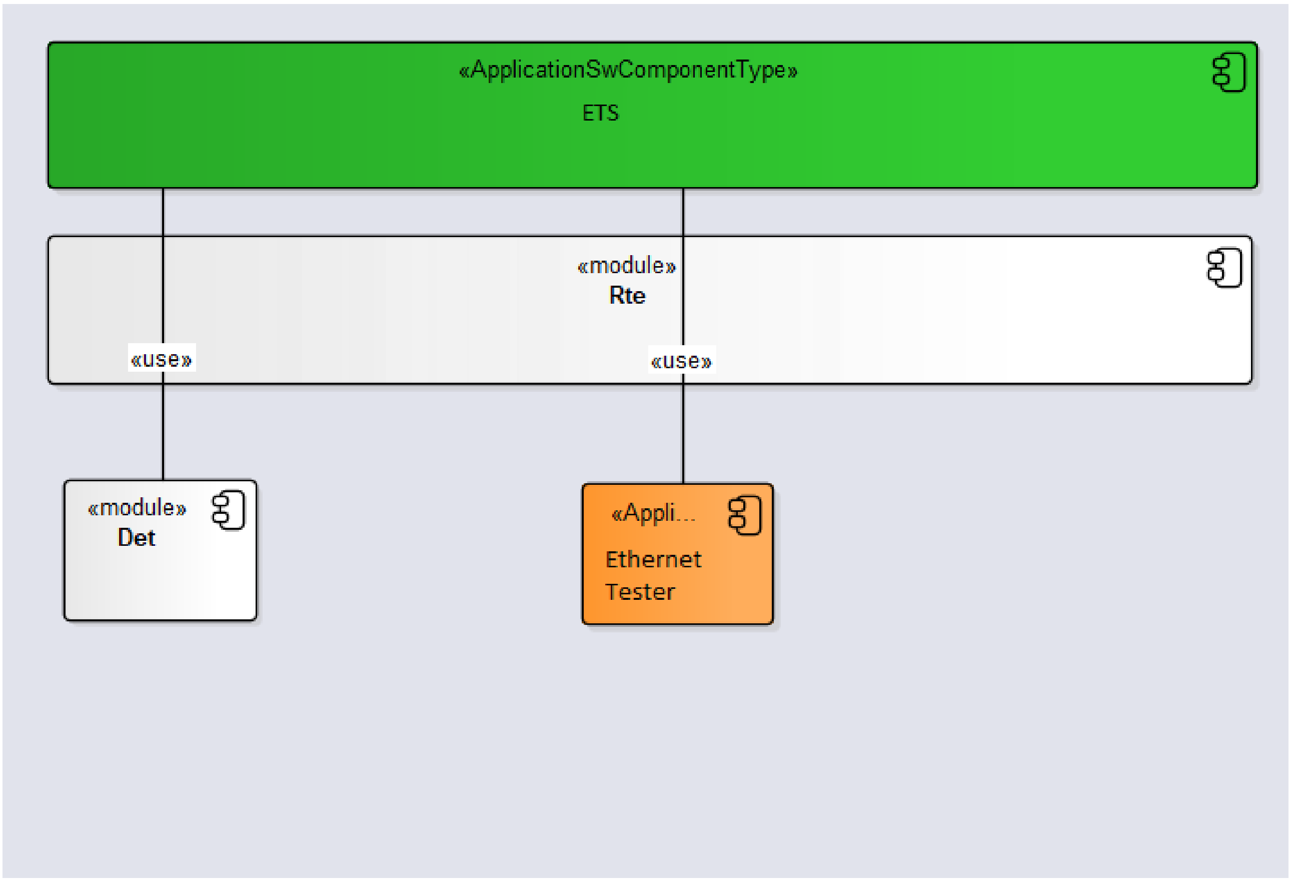


Figure 4.1. ETS interactions

4.1.2. Module dependencies

4.1.2.1. Interfaces between ETS and ETS tester

This section lists the interfaces between ETS and the ETS tester. The `subscribe to eventgroups`, `client service activate`, and `deactivate` tasks are also managed:

- When the Ethernet Tester calls the `subscribe eventgroup` function, the ETS receives a timer period through RTE interface and starts the timer with subscribe client event, start and duration parameters.
- When the Ethernet Tester calls the `client service activate` function, the ETS receives the start value through RTE interface and starts the timer with activate client event and start parameters.

The following service-related functions are available:

- `check byte order`: the Ethernet Tester checks that the ETS target platform uses little or big endian.

- `suspend interface`: the Ethernet Tester sets the ETS client service interface to wait a certain time (using the duration and start parameters).
- `reset interface`: the Ethernet Tester initializes again the interface parameters by running the `initalize` function.

The module also contains `get` and `set` testfields interface functions that the Ethernet Tester can directly call via a client-server interface. These functions are available from the interface listing.

The ETS errors are managed through DET RTE error interface. All modules have unique id values and when activated, the errors can be traced to a specific module.

4.1.2.1.1. Data mappings

The C/S Interfaces are mapped from the COM Stack to the SWC.

4.1.2.1.1.1. Service client-server interfaces

```
ClientServerInterface ETS_CheckByteOrder
{

  ETS_CheckByteOrder
  (
    IN uint8 summandUINT8
    IN uint16 summandUINT16
    OUT uint32 sum
  )
}
```

```
ClientServerInterface ETS_ClientServiceCallEchoUINT8Array
{

  ETS_ClientServiceCallEchoUINT8Array
  (
    INOUT ETS_Uint8Array uINT8ARRAY
  )
}
```

```
ClientServerInterface ETS_ClientServiceGetLastValueOfEventTCP
{

  ETS_ClientServiceGetLastValueOfEventTCP
  (
    OUT uint8 lastValue
  )
}



ClientServerInterface ETS_ClientServiceGetLastValueOfEventUDPMulticast
{

  ETS_ClientServiceGetLastValueOfEventUDPMulticast
  (
    OUT uint8 lastValue
  )
}



ClientServerInterface ETS_ClientServiceGetLastValueOfEventUDPUnicast
{

  ETS_ClientServiceGetLastValueOfEventUDPUnicast
  (
    OUT uint8 lastValue
  )
}



ClientServerInterface ETS_TestFieldUINT8Reliable
{

  getFieldTestFieldUINT8Reliable
  (
    OUT uint8 Getter
  )

  setFieldTestFieldUINT8Reliable
  (
    INOUT uint8 Setter
  )
}



ClientServerInterface ETS_TestFieldUINT8
{

  getFieldTestFieldUINT8
  (
    OUT uint8 Getter
  )

  setFieldTestFieldUINT8
  (
    INOUT uint8 Setter
  )
}
```

```
ClientServerInterface ETS_TestFieldUINT8Array
{

  getFieldTestFieldUINT8Array
  (
    OUT ETS_Uint8Array Getter
  )

  setFieldTestFieldUINT8Array
  (
    INOUT ETS_Uint8Array Setter
  )
}
```

4.1.2.1.1.2. Service sender-receiver interfaces

```
SenderReceiverInterface ETS_ClientServiceActivate
{
  uint8 start
}
```

```
SenderReceiverInterface ETS_ClientServiceDeactivate
{
  uint8 start
}
```

```
SenderReceiverInterface ETS_ClientServiceSubscribeEventgroup
{
  ETS_TimePeriod clientServiceSubscribeEventgroup
}


SenderReceiverInterface ETS_ModeRequest_SD_ClientService
{
  ETS_SD_ClientServiceModeType requestedMode
}


SenderReceiverInterface ETS_ModeRequest_SD_ConsumeEventGroup
{
  ETS_SD_ConsumeEventGroupModeType requestedMode
}


SenderReceiverInterface ETS_ModeRequest_SD_ServerService
{
  ETS_SD_ServerServiceModeType requestedMode
}


SenderReceiverInterface ETS_SuspendInterface
{
  ETS_TimePeriod suspendInterface
}


SenderReceiverInterface ETS_TestEventUINT8
{
  ETS_Uint8Value TestEventUINT8
}


SenderReceiverInterface ETS_TestEventUINT8Array
{
  ETS_Uint8Array TestEventUINT8Array
}


SenderReceiverInterface ETS_TestEventUINT8E2E
{
  ETS_Uint8Value TestEventUINT8E2E
}
```

```
SenderReceiverInterface ETS_TestEventUINT8Multicast
{
  ETS_Uint8Value TestEventUINT8Multicast
}



SenderReceiverInterface ETS_TestEventUINT8Reliable
{
  ETS_Uint8Value TestEventUINT8Reliable
}



SenderReceiverInterface ETS_TriggerEventUINT8
{
  ETS_TriggerEventType triggerEventUINT8
}



SenderReceiverInterface ETS_TriggerEventUINT8Array
{
  ETS_TriggerEventType triggerEventUINT8Array
}



SenderReceiverInterface ETS_TriggerEventUINT8E2E
{
  ETS_TriggerEventType triggerEventUINT8E2E
}



SenderReceiverInterface ETS_TriggerEventUINT8Multicast
{
  ETS_TriggerEventType triggerEventUINT8Multicast
}



SenderReceiverInterface ETS_TriggerEventUINT8Reliable
{
  ETS_TriggerEventType triggerEventUINT8Reliable
}
```

4.1.2.1.1.3. Service ModeSwitch interfaces

```
ModeSwitchInterface ETS_SwitchPort_CurrentMode
{
  ETS_CurrentMode CurrentMode
}
```

4.1.2.1.1.4. Echo client-server interfaces

```
ClientServerInterface ETS_EchoBitfields
{

  EchoBitfields
  (
    IN ETS_Bitfield_uint8 bitfield8_in
    IN ETS_Bitfield_uint16 bitfield16_in
    IN ETS_Bitfield_uint32 bitfield32_in
    OUT ETS_Bitfield_uint8_return bitfield8_out
    OUT ETS_Bitfield_uint16_return bitfield16_out
    OUT ETS_Bitfield_uint32_return bitfield32_out
  )
}
```

```
ClientServerInterface ETS_EchoCommonDatatypes
{

  EchoCommonDatatypes
  (
    IN boolean bOOLEAN_in
    IN uint8 uINT8_in
    IN uint16 uINT16_in
    IN uint32 uINT32_in
    IN sint8 iNT8_in
    IN sint16 iNT16_in
    IN sint32 iNT32_in
    IN float32 fLOAT32_in
    IN float64 fLOAT64_in
    OUT float64 fLOAT64_out
    OUT float32 fLOAT32_out
    OUT sint32 iNT32_out
    OUT sint16 iNT16_out
    OUT sint8 iNT8_out
    OUT uint32 uINT32_out
    OUT uint16 uINT16_out
    OUT uint8 uINT8_out
    OUT boolean bOOLEAN_out
  )
}



ClientServerInterface ETS_EchoENUM
{

  EchoENUM
  (
    IN ETS_ENUM ENUMValue
    OUT ETS_ENUM ENUMReturnValue
  )
}



ClientServerInterface ETS_EchoFLOAT64
{

  EchoFLOAT64
  (
    IN float64 float64Value
    OUT float64 float64ReturnValue
  )
}



ClientServerInterface ETS_EchoINT64
{

  EchoINT64
  (
    IN sint64 int64Value
    OUT sint64 int64ReturnValue
  )
}



ClientServerInterface ETS_EchoINT8
{

  EchoINT8
  (
    IN sint8 Int8Value
    OUT sint8 Int8ReturnValue
  )
}



ClientServerInterface ETS_EchoSTRUCTSimple
{

  EchoSTRUCTSimple
  (
    IN ETS_SimpleStructArray structElement
    OUT ETS_SimpleStructArray structReturnElement
  )
}



ClientServerInterface ETS_EchoStaticUINT8Array
{

  EchoStaticUINT8Array
  (
    IN ETS_StaticUint8Array ES_uINT8Array
    OUT ETS_StaticUint8Array ES_uINT8ArrayReturnValue
  )
}



ClientServerInterface ETS_EchoTYPEDEF
{

  EchoTYPEDEF
  (
    IN uint8 typeDefElement
    OUT uint8 typeDefreturnElement
  )
}



ClientServerInterface ETS_EchoUINT64
{

  EchoUINT64
  (
    IN uint64 Uint64Value
    OUT uint64 Uint64ReturnValue
  )
}



ClientServerInterface ETS_EchoUINT8
{

  EchoUINT8
  (
    IN uint8 Uint8Value
    OUT uint8 Uint8ReturnValue
  )
}



ClientServerInterface ETS_EchoUINT8Array
{

  EchoUINT8Array
  (
    INOUT ETS_Uint8Array uint8Array
  )
}



ClientServerInterface ETS_EchoUINT8Array16Bitlength
{

  EchoUINT8Array16Bitlength
  (
    INOUT ETS_Uint8Array E_uINT8Array
  )
}



ClientServerInterface ETS_EchoUINT8Array2Dim
{

  EchoUINT8Array2Dim
  (
    IN ETS_TwoDimUint8Array uINT8Array_2D
    OUT ETS_TwoDimUint8Array uINT8ArrayReturnValue_2D
  )
}



ClientServerInterface ETS_EchoUINT8Array8BitLength
{

  EchoUINT8Array8BitLength
  (
    INOUT ETS_Uint8Array uINT8Array_BL
  )
}



ClientServerInterface ETS_EchoUINT8ArrayMinSize
{

  EchoUINT8ArrayMinSize
  (
    IN ETS_Uint8Array uINT8Array_MS
    OUT ETS_Uint8Array uINT8ArrayReturnValue_MS
  )
}



ClientServerInterface ETS_EchoUINT8E2E
{

  EchoUINT8E2E
  (
    INOUT uint32 cRCId
    INOUT uint16 alive
    INOUT uint32 cRC
    INOUT uint8 uINT8Value
  )
}



ClientServerInterface ETS_EchoUINT8RELIABLE
{

  EchoUINT8RELIABLE
  (
    IN uint8 Uint8Value
    OUT uint8 Uint8ReturnValue
  )
}



ClientServerInterface ETS_EchoUNION
{

  EchoUNION
  (
    IN ETS_UNION uINT8Union
    OUT ETS_UNION uINT8UnionReturnValue
  )
}



ClientServerInterface ETS_EchoUTF16FIXED
{

  EchoUTF16FIXED
  (
    IN ETS_UTF16FixedArray uINT16Array
    OUT ETS_UTF16FixedArray uINT16ArrayReturnValue
  )
}



ClientServerInterface ETS_EchoUTF8FIXED
{

  EchoUTF8FIXED
  (
    IN ETS_UTF8FixedArray uINT8Array_FX
    OUT ETS_UTF8FixedArray uINT8ArrayReturnValue
  )
}
```

4.1.2.1.1.5. Fields client-server interfaces

```
ClientServerInterface ETS_InterfaceVersion
{
  getFieldInterfaceVersion
  (
    OUT ETS_VersionType Getter
  )
}
```

```
SenderReceiverInterface ETS_NotifyFieldInterfaceVersion
{
  ETS_VersionType Notifier
}
```

```
SenderReceiverInterface ETS_NotifyFieldTestFieldUINT8
{
  uint8 Notifier
}
```

```
SenderReceiverInterface ETS_NotifyFieldTestFieldUINT8Array
{
  ETS_Uint8Array Notifier
}
```

```
SenderReceiverInterface ETS_NotifyFieldTestFieldUINT8Reliable
{
  uint8 Notifier
}
```

4.1.2.1.2. Project specific interfaces

```
ClientServerInterface ETS_Pst<ProjectSpecificTest>
{

  ETS_Cbk_<ProjectSpecificTest>
  (
  )
}
```

4.1.2.1.3. Interface to Det (optional)

```
ClientServerInterface DetService
{
  PossibleError
  {
    E_OK = 0
    E_NOT_OK = 1
  }

  ReportError
  (
    IN uint8 InstanceId
    IN uint8 ApiId
    IN uint8 ErrorId
    ERR { E_OK, E_NOT_OK }
  )
}
```

4.1.2.2. Ports between ETS and Ethernet tester and Det modules

This section lists the ports between ETS and Ethernet Tester and Det modules.

NOTE The ASSOCIATED INTERFACE name to be connected is derived from the customer arxml files.

DIRECTION	PORT NAME	TARGET MODULE	ASSOCIATED INTERFACE	OPTIONALITY
RequirePort	Det	BswM	DetService	Yes
ProvidePort	ETS_CheckByteOrder	Ethernet Tester	ETS_CheckByteOrder	Yes
ProvidePort	ETS_ClientServiceCallEchoUINT8Array	Ethernet Tester	ETS_ClientServiceCallEchoUINT8Array	Yes
ProvidePort	ETS_ClientServiceGetLastValueOfEventTCP	Ethernet Tester	ETS_ClientServiceGetLastValueOfEventTCP	Yes
ProvidePort	ETS_ClientServiceGetLastValue-	Ethernet Tester	ETS_ClientServiceGetLastValue-	Yes

DIRECTION	PORT NAME	TARGET MODULE	ASSOCIATED IN-TERFACE	OPTIONALITY
	OfEventUDPMulti-cast		OfEventUDPMulti-cast	
ProvidePort	ETS_ClientSer-viceGetLastValue-OfEventUDPUnicast	Ethernet Tester	ETS_ClientSer-viceGetLastValue-OfEventUDPUnicast	Yes
ProvidePort	ETS_EchoBitfields	Ethernet Tester	ETS_EchoBitfields	Yes
ProvidePort	ETS_EchoCommon-Datatypes	Ethernet Tester	ETS_EchoCommon-Datatypes	Yes
ProvidePort	ETS_EchoENUM	Ethernet Tester	ETS_EchoENUM	Yes
ProvidePort	ETS_EchoFLOAT64	Ethernet Tester	ETS_EchoFLOAT64	Yes
ProvidePort	ETS_EchoINT64	Ethernet Tester	ETS_EchoINT64	Yes
ProvidePort	ETS_EchoINT8	Ethernet Tester	ETS_EchoINT8	Yes
ProvidePort	ETS_EchoSTRUC-TSimple	Ethernet Tester	ETS_EchoSTRUC-TSimple	Yes
ProvidePort	ETS_-EchoStaticUINT8Array	Ethernet Tester	ETS_-EchoStaticUINT8Array	Yes
ProvidePort	ETS_EchoTYPE-DEF	Ethernet Tester	ETS_EchoTYPE-DEF	Yes
ProvidePort	ETS_EchoUINT64	Ethernet Tester	ETS_EchoUINT64	Yes
ProvidePort	ETS_EchoUINT8	Ethernet Tester	ETS_EchoUINT8	Yes
ProvidePort	ETS_-EchoUINT8Array	Ethernet Tester	ETS_-EchoUINT8Array	Yes
ProvidePort	ETS_-EchoUINT8Array16Bitlength	Ethernet Tester	ETS_-EchoUINT8Array16Bitlength	Yes
ProvidePort	ETS_-EchoUINT8Array2Dim	Ethernet Tester	ETS_-EchoUINT8Array2Dim	Yes
ProvidePort	ETS_-EchoUINT8Array8BitLength	Ethernet Tester	ETS_-EchoUINT8Array8BitLength	Yes
ProvidePort	ETS_-EchoUINT8ArrayMinSize	Ethernet Tester	ETS_-EchoUINT8ArrayMinSize	Yes
ProvidePort	ETS_-EchoUINT8E2E	Ethernet Tester	ETS_-EchoUINT8E2E	Yes
ProvidePort	ETS_-EchoUINT8RELIABLE	Ethernet Tester	ETS_-EchoUINT8RELIABLE	Yes

DIRECTION	PORT NAME	TARGET MODULE	ASSOCIATED INTERFACE	OPTIONALITY
ProvidePort	ETS_EchoUNION	Ethernet Tester	ETS_EchoUNION	Yes
ProvidePort	ETS_EchoUTF16FIXED	Ethernet Tester	ETS_EchoUTF16FIXED	Yes
ProvidePort	ETS_EchoUTF8FIXED	Ethernet Tester	ETS_EchoUTF8FIXED	Yes
ProvidePort	ETS_InterfaceVersion	Ethernet Tester	ETS_InterfaceVersion	Yes
ProvidePort	ETS_NotifyFieldTestFieldUINT8	Ethernet Tester	ETS_NotifyFieldTestFieldUINT8	Yes
ProvidePort	ETS_NotifyFieldTestFieldUINT8Array	Ethernet Tester	ETS_NotifyFieldTestFieldUINT8Array	Yes
ProvidePort	ETS_NotifyFieldTestFieldUINT8Reliable	Ethernet Tester	ETS_NotifyFieldTestFieldUINT8Reliable	Yes
ProvidePort	ETS_NotifyFieldsInterfaceVersion	Ethernet Tester	ETS_NotifyFieldsInterfaceVersion	Yes
ProvidePort	ETS_SD_ClientServiceRequest	Ethernet Tester	ETS_ModeRequest_SD_ClientService	Yes
ProvidePort	ETS_SD_ConsumedEventGroupRequest	Ethernet Tester	ETS_ModeRequest_SD_ConsumeEventGroup	Yes
ProvidePort	ETS_SD_ServerServiceRequest	Ethernet Tester	ETS_ModeRequest_SD_ServerService	Yes
ProvidePort	ETS_TestEventUINT8	Ethernet Tester	ETS_TestEventUINT8	Yes
ProvidePort	ETS_TestEventUINT8Array	Ethernet Tester	ETS_TestEventUINT8Array	Yes
ProvidePort	ETS_TestEventUINT8E2E	Ethernet Tester	ETS_TestEventUINT8E2E	Yes
ProvidePort	ETS_TestEventUINT8Multicast	Ethernet Tester	ETS_TestEventUINT8Multicast	Yes
ProvidePort	ETS_TestEventUINT8Reliable	Ethernet Tester	ETS_TestEventUINT8Reliable	Yes

DIRECTION	PORT NAME	TARGET MODULE	ASSOCIATED INTERFACE	OPTIONALITY
ProvidePort	ETS_TestFieldUINT8	Ethernet Tester	ETS_TestFieldUINT8	Yes
ProvidePort	ETS_TestFieldUINT8Array	Ethernet Tester	ETS_TestFieldUINT8Array	Yes
ProvidePort	ETS_TestFieldUINT8Reliable	Ethernet Tester	ETS_TestFieldUINT8Reliable	Yes
ProvidePort	ETS_Pst<ProjectSpecificTest>	Ethernet Tester	ETS_Pst<ProjectSpecificTest>	Yes
RequirePort	ETS_ClientServiceActivate	Ethernet Tester	ETS_ClientServiceActivate	Yes
RequirePort	ETS_ClientServiceDeactivate	Ethernet Tester	ETS_ClientServiceDeactivate	Yes
RequirePort	ETS_ClientServiceSubscribeEventgroup	Ethernet Tester	ETS_ClientServiceSubscribeEventgroup	Yes
RequirePort	ETS_EchoUINT8ArrayClient	Ethernet Tester	ETS_EchoUINT8Array	Yes
RequirePort	ETS_ResetInterface	Ethernet Tester	ETS_ResetInterface	Yes
RequirePort	ETS_SuspendInterface	Ethernet Tester	ETS_SuspendInterface	Yes
RequirePort	ETS_TestEventUINT8ArrayClient	Ethernet Tester	ETS_TestEventUINT8Array	Yes
RequirePort	ETS_TestEventUINT8Client	Ethernet Tester	ETS_TestEventUINT8	Yes
RequirePort	ETS_TestEventUINT8E2EClient	Ethernet Tester	ETS_TestEventUINT8E2E	Yes
RequirePort	ETS_TestEventUINT8MulticastClient	Ethernet Tester	ETS_TestEventUINT8Multicast	Yes
RequirePort	ETS_TestEventUINT8ReliableClient	Ethernet Tester	ETS_TestEventUINT8Reliable	Yes
RequirePort	ETS_TriggerEventUINT8	Ethernet Tester	ETS_TriggerEventUINT8	Yes
RequirePort	ETS_TriggerEventUINT8Array	Ethernet Tester	ETS_TriggerEventUINT8Array	Yes

DIRECTION	PORT NAME	TARGET MODULE	ASSOCIATED INTERFACE	OPTIONALITY
RequirePort	ETS_-TriggerEventUINT8E2E	Ethernet Tester	ETS_-TriggerEventUINT8E2E	Yes
RequirePort	ETS_-TriggerEventUINT8Multicast	Ethernet Tester	ETS_-TriggerEventUINT8Multicast	Yes
RequirePort	ETS_-TriggerEventUINT8Reliable	Ethernet Tester	ETS_-TriggerEventUINT8Reliable	Yes
RequirePort	currentMode	Ethernet Tester	ETS_Switch-Port_CurrentMode	Yes

Table 4.1. List of available ports

Details on optional ports: **Yes**: the ports are present when configuration option of each port has been enabled.

4.1.3. Tester interactions

4.1.3.1. Tester service interactions

The Service module contains the main function of the ETS module. It manages the event check whenever it is called, and contains some parts of trigger and test events functionality. All initializations are performed here.

When the Ethernet Tester calls the `client service activate` function, the ETS receives the start value through RTE interface and starts the timer with activate client event and start parameters.

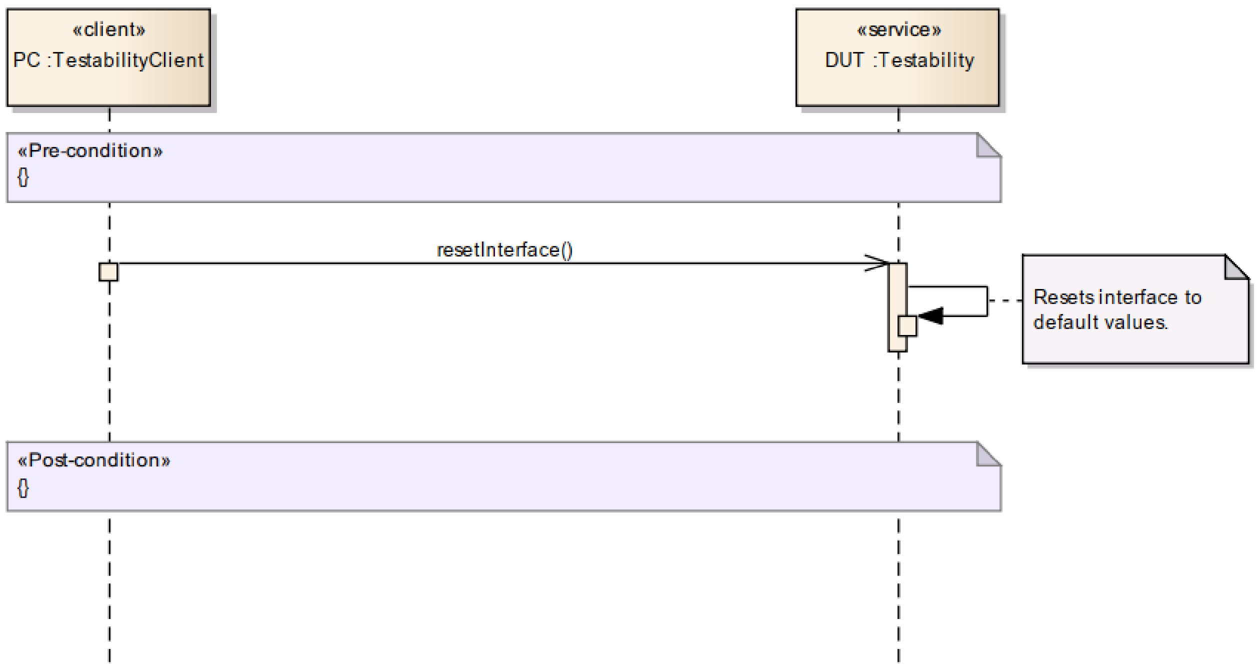


Figure 4.2. ETS Service Example

4.1.3.2. Tester trigger test event interactions

The ETS receives trigger test events from the ethernet testing software components (Ethernet Tester). Event reporting is managed via RTE 'DataReceivedEvent' mechanism to ensure a prompt storage of the event.

The ETS ensures that the type of event is received, and that it does the required action (such as sending specific stored integer value periodically). The receiving notification of the event is sent to Ethernet Tester if the event is related to setting or getting stored fields handling.

The test client server connections are bound to events and are activated by sending subscribe message via RTE and deactivated by sending deactivate to RTE. This is done by receiving activate client event. Event reset interface initializes the software.

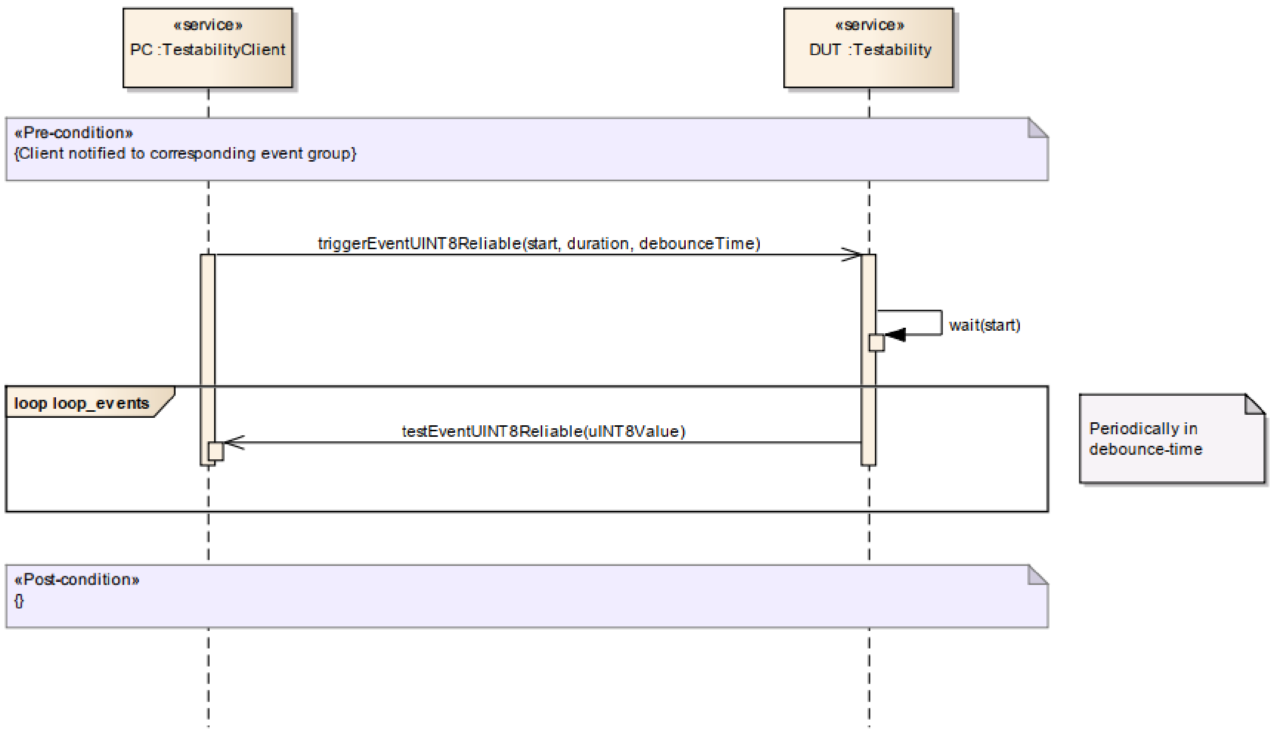

«service»
PC :TestabilityClient
«service»
DUT :Testability
«Pre-condition»
{Client notified to corresponding event group}
triggerEventUINT8Reliable(start, duration, debounceTime)
wait(start)
loop loop_events
testEventUINT8Reliable(uINT8Value)
Periodically in debounce-time
«Post-condition»
{}

Figure 4.3. ETS Trigger Event Example

4.1.3.3. Tester echo interactions

The ETS receives echo test from the Ethernet testing software components (Ethernet Tester). Echo tests are implemented as direct client-server interfaces. When a specific ETS echo function is called with specific parameters, the function copies the input parameter values to the output parameter values, and the testing software components (Ethernet Tester) can verify that data is correctly looped.

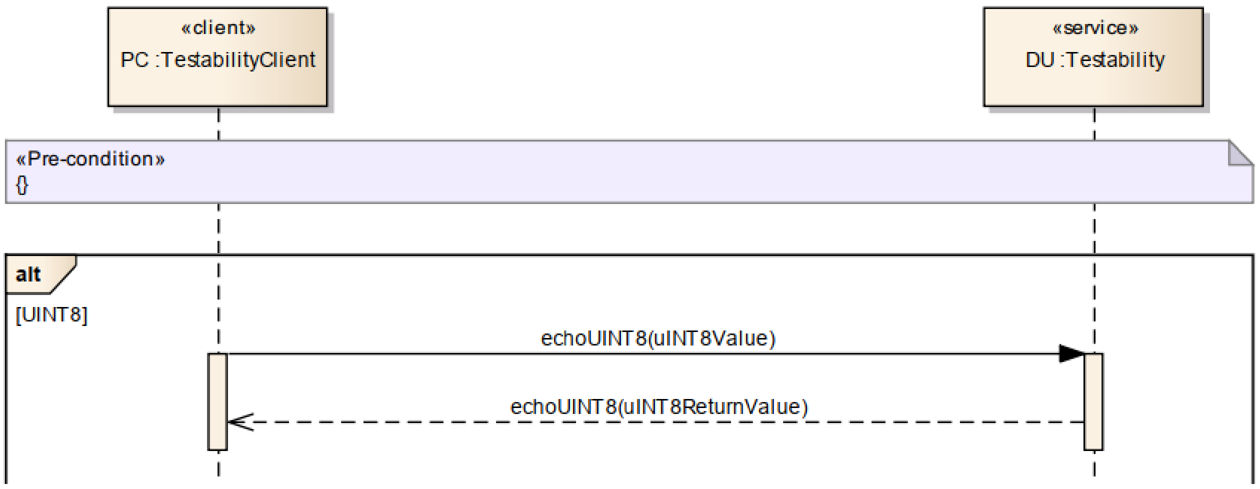


Figure 4.4. ETS echo example

There are different kinds of echo functions such as for common data types, different integers and floats, different kind of arrays, enums, typedef, structures.

4.1.3.4. Tester fields interactions

The ETS receives `get` or `set` field requests from the Ethernet testing software components (Ethernet Tester). The tests are implemented as direct client-server interfaces.

When specific ETS `set` field function is called with specific parameters, the function sets the parameter given data to stored value where it can be read later. A notification message is then sent to Ethernet Tester.

When specific ETS `get` field function is called, the function reads stored value and adds it. As a return parameter, Ethernet Tester gets the stored value from ETS.

Only the `set` field sends interface version value in the notification message. The get field interface version reads the stored value, and sends it directly. As a return parameter, Ethernet Tester gets the stored value from ETS.

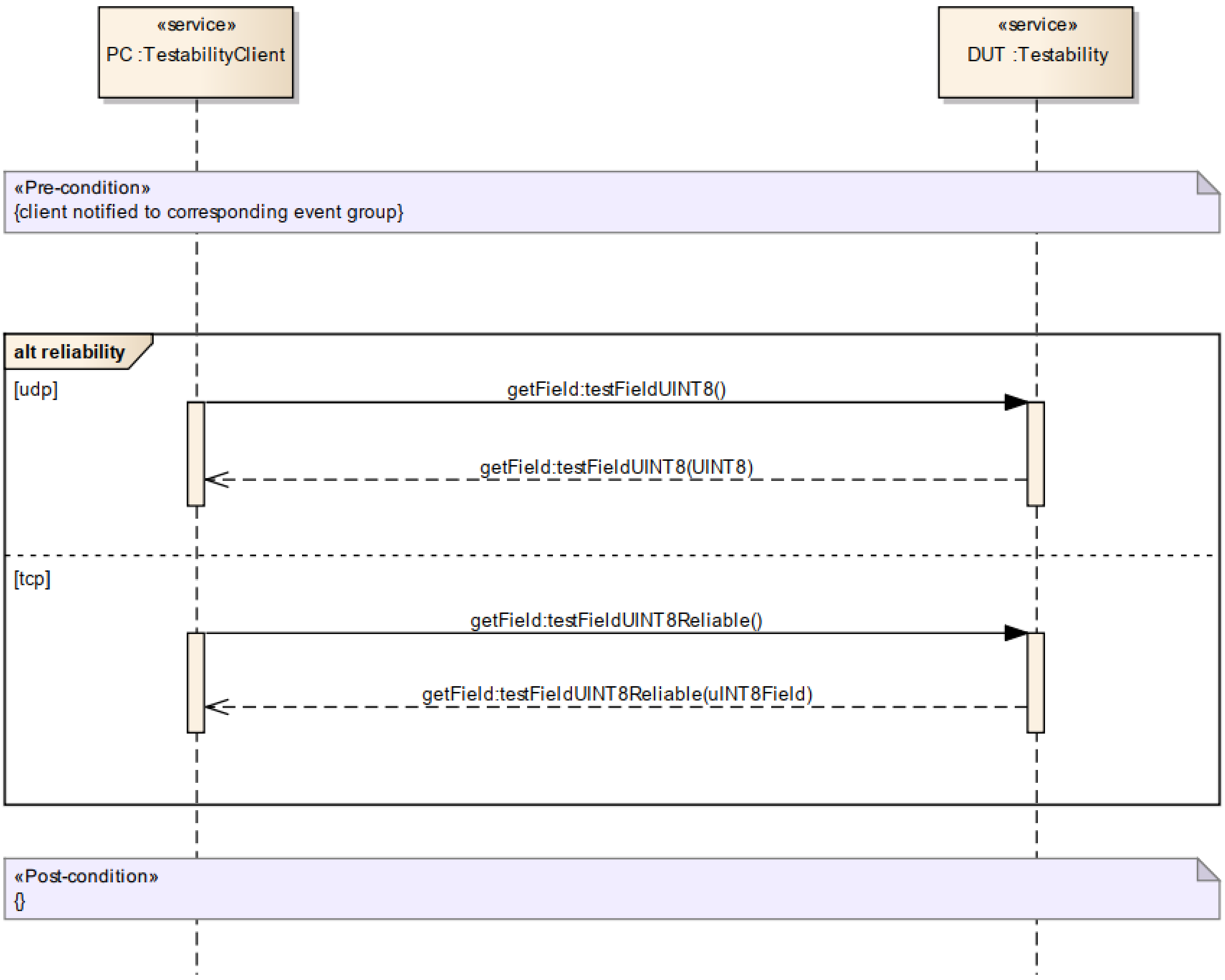

«service»
PC :TestabilityClient
«service»
DUT :Testability
«Pre-condition»
{client notified to corresponding event group}
alt reliability
[udp]
getField:testFieldUINT8()
getField:testFieldUINT8(UINT8)
[tcp]
getField:testFieldUINT8Reliable()
getField:testFieldUINT8Reliable(uINT8Field)
«Post-condition»
{}

Figure 4.5. ETS fields example

4.1.4. Configuring ETS module

This section presents the configuration views for ETS, and it describes the individual configuration parameters.

4.1.4.1. Software component description

A software component description (SWCD) is needed for the final integration of the ETS into the complete system model. The description is an arxml file to generate with EB tresos Studio after completing the configuration of the ETS.

4.1.4.2. General tab

Use the **General** tab to:

- Adjust the main function period. Default value is 20ms.
- Activate or deactivate DevErrorDetect (DET) error reporting by checking the selection box.

4.1.4.3. Enhanced Testability Service tab

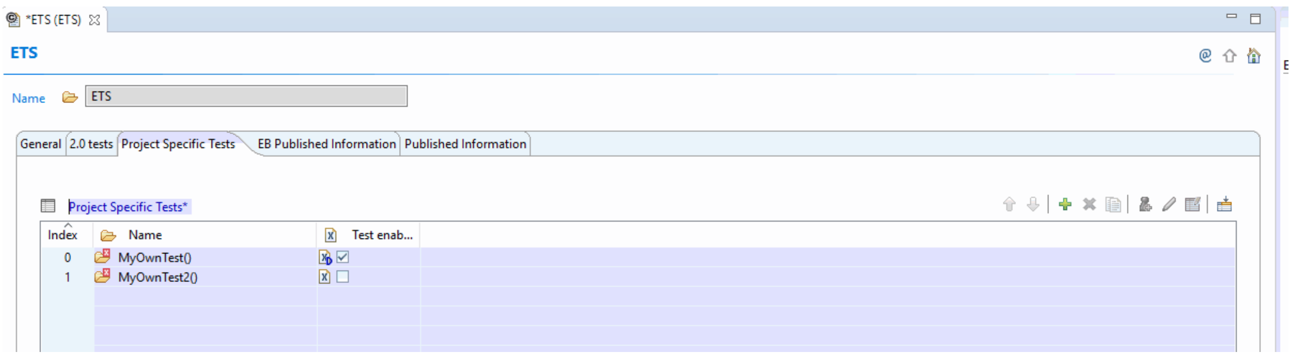
- Echo tests: select echo tests individually. All tests are selected by default.
- Single service tests: select client interaction tests individually. All tests are selected by default. The tests are related to service handling functionalities.
- Single event tests: select individual event tests individually. All tests are selected by default. The tests are related to trigger envents functionalities.
- Events and fields tests: select the events and fields tests individually. All tests are selected by default. The tests are related to test events and test fields functionalities. InterfaceVersion and timerEventUINT8E2E tests are also available.

4.1.4.4. Project-specific tests tab

To modify existings tests or create totally new special tests, you can list the configurations and check on/off each test individually.

Step 1
Select the **Add** icon to add a test.

Step 2
Rename the test.
The test is generated and the empty skeleton of function is created to test area template.



Name: ETS

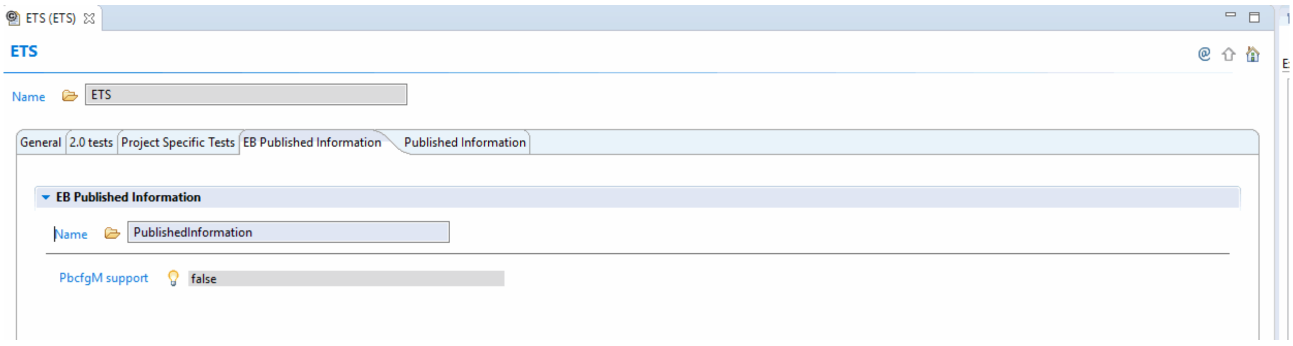
Index	Name	Test enab...
0	MyOwnTest()	✓
1	MyOwnTest2()	☐

Figure 4.6. Project specific tests example

Step 3
Use the functions as a base for you own test.

4.1.4.5. EB PublishedInformation tab

EB PublishedInformation default value is false.



ETS (ETS)

ETS

Name: ETS

General | 2.0 tests | Project Specific Tests | EB Published Information | Published Information

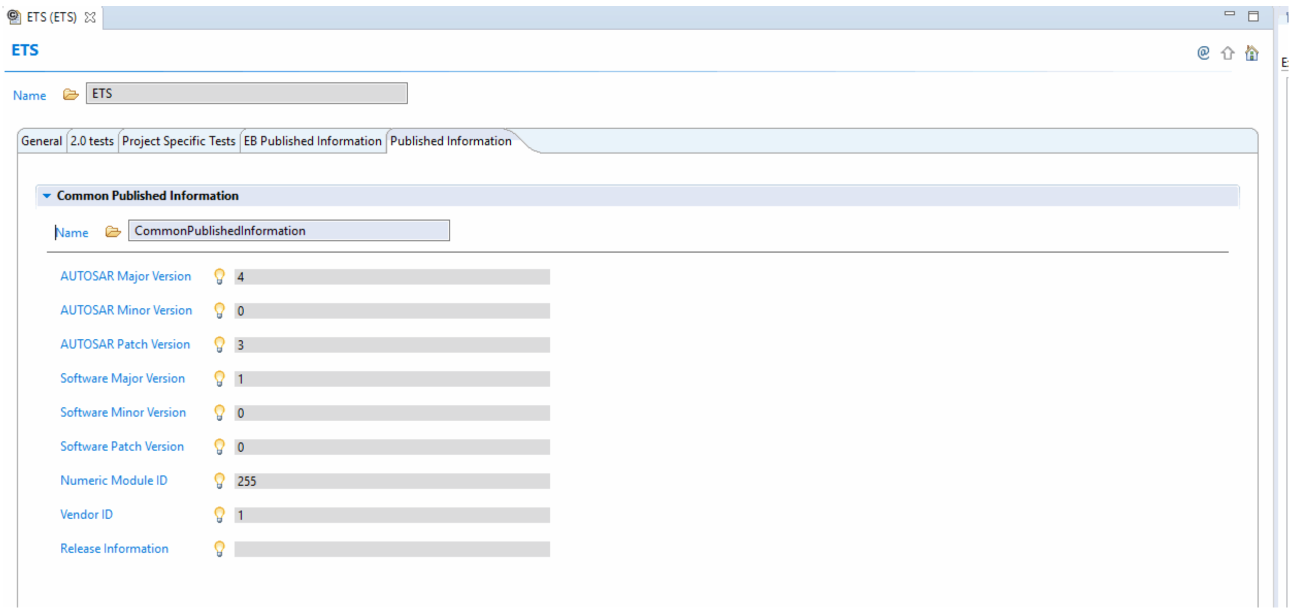
EB Published Information

Name: PublishedInformation

PbcfgM support: false

Figure 4.7. EB PublishedInformation

4.1.4.6. PublishedInformation tab



ETS (ETS)

ETS

Name: ETS

General | 2.0 tests | Project Specific Tests | EB Published Information | Published Information

Common Published Information

Name: CommonPublishedInformation

AUTOSAR Major Version	4
AUTOSAR Minor Version	0
AUTOSAR Patch Version	3
Software Major Version	1
Software Minor Version	0
Software Patch Version	0
Numeric Module ID	255
Vendor ID	1
Release Information	

Figure 4.8. CommonPublishedInformation

4.1.4.7. Product specific key checks

- SOME/IP signals to be checked with tags DATA-TRANSFORMATIONS, TRANSFORMATION-TECHNOLOGY.

- Client server to signal mappings and sender receiver to signal mappings to be checked with tags CLIENT-SERVER-TO-SIGNAL-MAPPING, SENDER-RECEIVER-TO-SIGNAL-MAPPING.
- Signal I-PDU should contain transfer property.

4.1.4.8. Connections

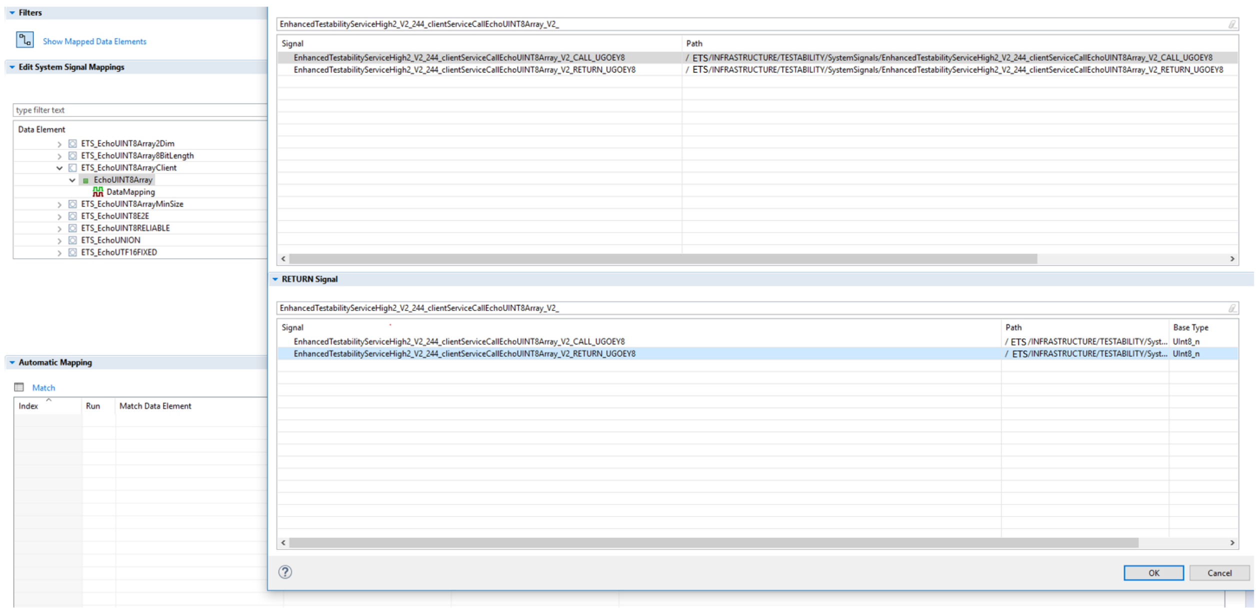
IMPORTANT Do not map signals to wrong ports. A keyword can be the interface name, and most of ETS configuration mapping information could be retrieved from Interface mappings in Symphony.

Step 1
Create ETS swc prototype as part of CPU instance. ETS contains the client server interfaces with the same configurations as in product and ports related to CSI and these signals can map to ports.

Step 2
Select the interface, and then search the P port and R port of the selected interface.

Step 3
Map the signals to the ports tasking reference from the product. In case of missing ports or missing interfaces, report to the ETS author.

You can map the signals from the CS – Signal mapping. For example: CSO: ETS_-ClientServiceCallEchoUINT8Array, P Port: ETS_ClientServiceCallEchoUINT8Array, R port: ETS_-EchoUINT8ArrayClient



Data Element
- ETS_EchoUINT8Array2Dim
- ETS_EchoUINT8Array8BitLength
- ETS_EchoUINT8ArrayClient
  - EchoUINT8Array
    - DataMapping
- ETS_EchoUINT8ArrayMinSize
- ETS_EchoUINT8E2E
- ETS_EchoUINT8RELIABLE
- ETS_EchoUNION
- ETS_EchoUTF16FIXED

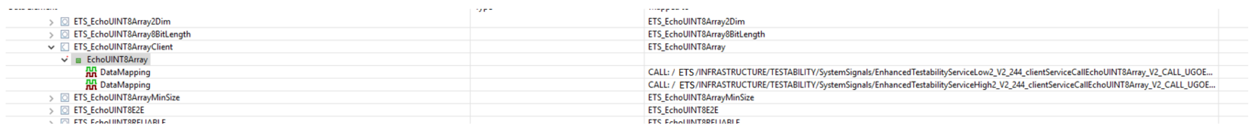
EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_

Signal	Path
EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_CALL_UGOEY8	/ ETS/INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_CALL_UGOEY8
EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_RETURN_UGOEY8	/ ETS/INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_RETURN_UGOEY8

EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_

Signal	Path	Base Type
EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_CALL_UGOEY8	/ ETS /INFRASTRUCTURE/TESTABILITY/Syst...	UInt8_n
EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_RETURN_UGOEY8	/ ETS/INFRASTRUCTURE/TESTABILITY/Syst...	UInt8_n

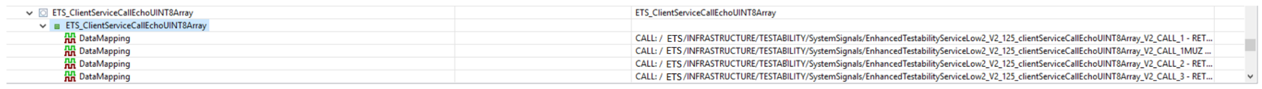
Figure 4.9. R port signal mapping



ETS_EchoUINT8Array2Dim		ETS_EchoUINT8Array2Dim
ETS_EchoUINT8Array8BitLength		ETS_EchoUINT8Array8BitLength
ETS_EchoUINT8ArrayClient		ETS_EchoUINT8Array
EchoUINT8Array		
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceLow2_V2_244_clientServiceCallEchoUINT8Array_V2_CALL_UGOE...
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceHigh2_V2_244_clientServiceCallEchoUINT8Array_V2_CALL_UGOE...
ETS_EchoUINT8ArrayMinSize		ETS_EchoUINT8ArrayMinSize
ETS_EchoUINT8E2E		ETS_EchoUINT8E2E

Figure 4.10. R port signal mappings

Similarly map signal of P port



ETS_ClientServiceCallEchoUINT8Array		ETS_ClientServiceCallEchoUINT8Array
ETS_ClientServiceCallEchoUINT8Array		
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceLow2_V2_125_clientServiceCallEchoUINT8Array_V2_CALL_1 - RET...
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceLow2_V2_125_clientServiceCallEchoUINT8Array_V2_CALL_1MUZ ...
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceLow2_V2_125_clientServiceCallEchoUINT8Array_V2_CALL_2 - RET...
DataMapping		CALL: / ETS /INFRASTRUCTURE/TESTABILITY/SystemSignals/EnhancedTestabilityServiceLow2_V2_125_clientServiceCallEchoUINT8Array_V2_CALL_3 - RET...

Figure 4.11. Signal mappings of P port

The sender receiver interfaces are used to test Service discovery: methods, events and fields (Getter, Setter, Notifier).

Step 4

Ensure that you map the signals related to SRI.

- clientServiceActivate (searches all signals in product and map)
- clientServiceDeactivate (searches all signals in product and map)
- clientServiceSubscribeEventgroup (searches all signals in product and map)
- suspendInterface (searches all signals in product and map)

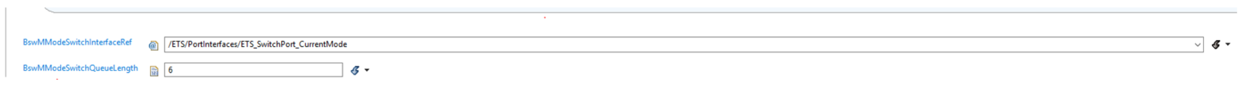
4.1.4.9. Configuring BSWM

It is recommended to configure ETS initialization as the last StartuptwoB actionlist action.

4.1.4.9.1. BSWM – ETS connection

For notification (server service availability), requests (client), switching modes (ETS mode), connect ETS Ports currentMode, ETS_SD_ClientServiceRequest, ETS_SD_ConsumedEventGroupRequest, ETS_SD_-ServerServiceRequest To related BswM ports Related Bswm Ports are generated if BSWM SWITCH port, Mode request ports are configured in BSWM()

4.1.4.9.2. BSWM switch port

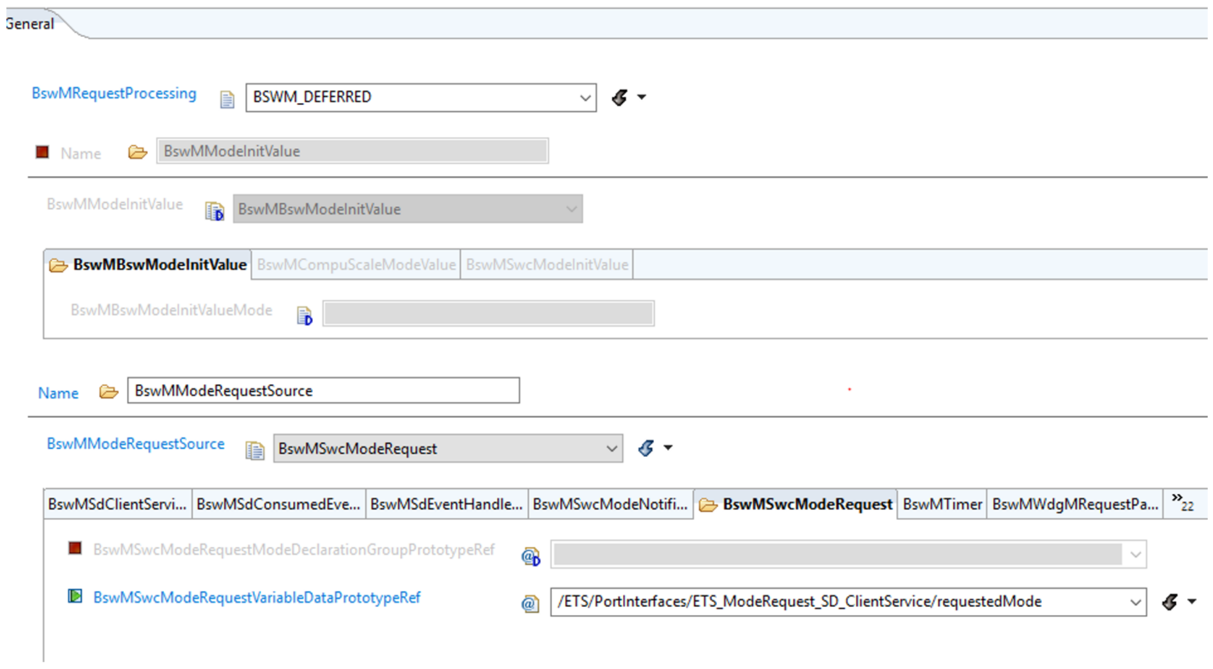


BswMModeSwitchInterfaceRef	/ETS/PortInterfaces/ETS_SwitchPort_CurrentMode
BswMModeSwitchQueueLength	6

Figure 4.12.

4.1.4.9.3. Mode request Ports

ETS_SD_ClientServiceModeRequest:



General

BswMRequestProcessing: BSWM_DEFERRED

Name: BswMModeInitValue

BswMModeInitValue: BswMBswModeInitValue

BswMBswModeInitValue | BswMCompuScaleModeValue | BswMSwcModeInitValue

BswMBswModeInitValueMode:

Name: BswMModeRequestSource

BswMModeRequestSource: BswMSwcModeRequest

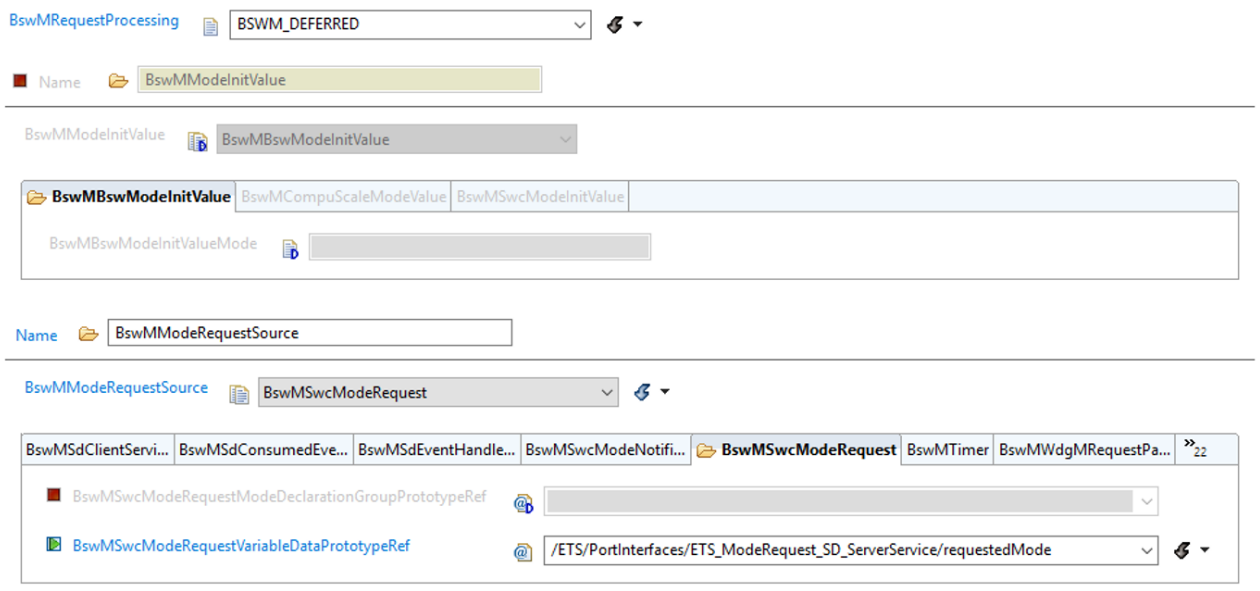
BswMSdClientServi... | BswMSdConsumedEve... | BswMSdEventHandle... | BswMSwcModeNotifi... | BswMSwcModeRequest | BswMTimer | BswMWdgMRequestPa... | »22

BswMSwcModeRequestModeDeclarationGroupPrototypeRef:

BswMSwcModeRequestVariableDataPrototypeRef: /ETS/PortInterfaces/ETS_ModeRequest_SD_ClientService/requestedMode

Figure 4.13. Client service mode request port

ETS_SD_Server serviceModeRequest:



BswMRequestProcessing: BSWM_DEFERRED

Name: BswMModeInitValue

BswMModeInitValue: BswMBswModeInitValue

BswMBswModeInitValue | BswMCompuScaleModeValue | BswMSwcModeInitValue

BswMBswModeInitValueMode:

Name: BswMModeRequestSource

BswMModeRequestSource: BswMSwcModeRequest

BswMSdClientServi... | BswMSdConsumedEve... | BswMSdEventHandle... | BswMSwcModeNotifi... | BswMSwcModeRequest | BswMTimer | BswMWdgMRequestPa... | »22

BswMSwcModeRequestModeDeclarationGroupPrototypeRef:

BswMSwcModeRequestVariableDataPrototypeRef: /ETS/PortInterfaces/ETS_ModeRequest_SD_ServerService/requestedMode

Figure 4.14. Server service mode request port

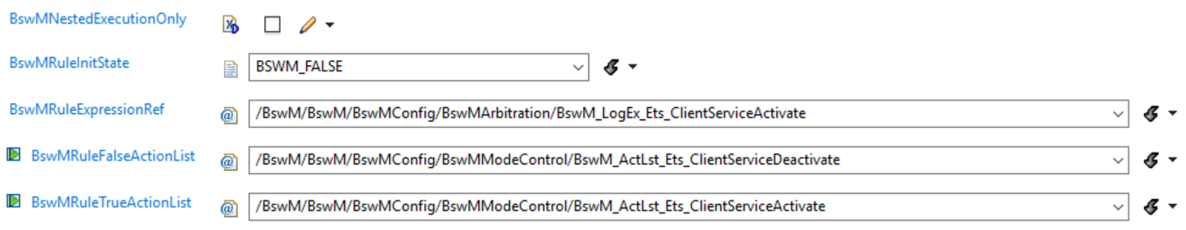
4.1.4.9.4. Mode conditions

Create mode conditions for ETS_ClientServiceActivate, ETS_ServerServiceActivate, ConsumedEventGroup. Each equals to ETS_SD_CLIENT_SERVICE_REQUESTED, ETS_SD_SERVER_SERVICE_AVAILABLE, ETS_SD_CONSUMED_EVENTGROUP_REQUESTED respectively.

4.1.4.9.5. Logical expression, rule and action

Sequence: if the signal is received to request the client service state to ETS SWC, ETS sends a notification. Based on this notification, BswM must perform the action to request for the service.

- For client services: activate and deactivate client services.



BswMNestedExecutionOnly	☐
BswMRuleInitState	BSWM_FALSE
BswMRuleExpressionRef	/BswM/BswM/BswMConfig/BswMArbitration/BswM_LogEx_Ets_ClientServiceActivate
BswMRuleFalseActionList	/BswM/BswM/BswMConfig/BswMModeControl/BswM_ActLst_Ets_ClientServiceDeactivate
BswMRuleTrueActionList	/BswM/BswM/BswMConfig/BswMModeControl/BswM_ActLst_Ets_ClientServiceActivate

Figure 4.15. ETS Client service rule

  Figure 4.15, “ETS Client service rule” configures the available action to BswMCLientServiceModeRequest with service state requested and Reference to client service.

  True action to “Requested”, False action to “Released” .

- For consumed event groups: configure the available action BswMSdConsumedEventGroupMod-eRequest with BSWM_SD_CONSUMED_EVENTGROUP_REQUESTED for true action and BSWM_-SD_CONSUMED_EVENTGROUP_RELEASE for false action.

  NOTE
  Do this for each Event group in the service.

- For server services: configure the available action to BswMSdServerServiceModeRequest with service state BSWM_SD_SERVER_SERVICE_AVAILABLE for true action and BSWM_SD_SERV-ER_SERVICE_DOWN for false action.

5. Best practices

- Section 5.1, “ARXML files”: provides recommendations for ARXML files.
- Section 5.2, “Multi-core usage”: explains how to use Multi-core.
- Section 5.3, “Memory protection usage”: explains how to use Memory-Protection.

5.1. ARXML files

5.1.1. Required ARXML files

- With a new importer, ensure that you have all the required ARXML files in the output generated folder once you populated the configuration in the modules.
- Ensure that the ARXML files that were imported are the files after patching the tools and not the files from the OEM.
- Check that you use the file patched by Tools for OEM for import.

5.1.2. Recommended configuration

With a new importer:

- Use a recommended configuration available for Tools for OEM to populate the argument values for which the patches need to be applied.
- Configure the Diag Extract/ECU extract in the Tools for OEM, and generate the patch file.
- Ensure that the required patches are used, and that these required patches are described orderly in the recommended configuration.
- The recommended configuration is not automatically updated during the release cycle of the Tools for OEM, and once the module is added, the recommended configuration cannot be updated anymore. To update the additional recommended configuration parameters, right-click and select **Module Configurations**, then add newly a module.

5.1.3. ECU configuration wizard

- When loading a patch file in tresos, some modules may not be configured. The configuration updates propagation does not happen automatically. Select **Create ECU Configuration** and run the ECU configuration wizard.
- If the modules do not get imported after executing the ECU configuration wizard, go to **Select configurations to import**, and ensure that the modules to be imported are enabled.

5.1.4. Error log

If you have errors during import, remember to check the **Error log** and read the error messages.

5.1.5. Temporary file

If the temporary file is not automatically created, restart tresos to create it.

5.1.6. ARXML file not well-formed

- If the error description is not helpful, select **Windows** > **References**> **EB tresos Studio** > **Developer features** > Enable **Show stack traces for errors** to get a detailed stack trace per error.
- In **Im-and exporter** > **All models** tab, check that the file is not well-formed.

5.1.7. Content not correctly modeled

To ensure that at least the required content is already modeled in the ARXML files, go to **Im-and Exporter** > **All Models** > **Enable Validate against XML schema**.

5.1.8. Content of several versions

To use a mix-up of versions with ARXML files, (for example 4.2.1 and 4.3.0), ensure that in **Im-and Exporter** > **All Models** tab > **Auto detect for each file** option is selected.

5.1.9. Multiple importers

Once you configured the files to be used in your new importer, and that you added additional information, in the case you use multiple importers:

- If you want both contents to coexist: in the **System Model Import** tab, select **Merge content of selected files into existing model**.
- If you want to overwrite the content: in the **System Model Import** tab, select **Overwrite existing model with content from selected files**.

5.2. Multi-core usage

Multicore configuration enables to distribute the CPU load to different cores.The example covers the use case of distributed SWCs and basic software modules. SWC Cyclic Counter is mapped to a different core, mode management stack modules BswM and EcuM are distributed over the cores to ensure synchronized mode changes across multiple cores.

EcuM:

- Master/slave approach
- Master and slave synchronize internally

BswM:

- Independent instances per partition
- Controlled via core-local EcuM instances
- Synchronization among BswM instances via Mode request ports

OS:

For details of multicore support by OS, refer to *Multicore User guide* chapter in *3.1_AutoCoreOS_Documentation.pdf* available in tresos installation folder.

- Multi-core support is based on OsApplications.
- OsApplication is a group of Os objects.
- OS Applications are assigned to cores, multiple OsApplications can be mapped to the same core.
- The assignment of Os Objects is done during configuration time.

A webinar about Multi-Core software design for AURIX in combination with EB tresos product line is available.

5.2.1. Updating System Description Importer

You must extend the **SystemExtract_SystemModel** importer with the port prototypes needed for core synchronization.

Step 1
Open the **Create, manage and run im- and exporters** dialog.

Step 2
Select the **SystemExtract_SystemModel** and click the **All Models** tab.

Step 3
On the **All Models** page:

Step 3.1
To add the file, select **Add**, and browse to `Project root"/system/BswMModeMulticore.arxml`, or select **New** to enter the path `system/BswMModeMulticore.arxml`.

Step 3.2
Click **OK**.

Step 3.3
Execute multiple task **SystemDescriptionImport_Run** to update the system model.

5.2.2. EcuM

Step 1
Enable **EcuMEnableMulticore**.

Step 2
Set **EcuMMasterCoreId** to 0.

5.2.3. OS

Set the number of cores in **OsNumberOfCores** to 2.

5.2.3.1. OsApplication

Step 1
Create an additional **OsApplication**.

Step 2
Enable **OsApplicationCoreAssignment** for both OsApplications.

Step 3
Set the core assignment.

5.2.3.2. OsCounter

You must create a new OSCounter for the schedule table on the second core.

Step 1
Rename the **HwCounter** to **HwCounter_C0**.

Step 2
Duplicate **HwCounter_C0** to **HwCounter_C1**.

Step 3
In **HwCounter_C1**, change OsCounterTricoreTimer to a timer that is not used (for example STM0_T1).

5.2.3.3. OSTasks

New tasks are needed for core initialization, execution of Rte events, mode switching, and a cyclic task to execute the main function of Bsw. It is recommended to leave the task naming for the unmodified existing core to enable the auto-assignment of the main function.

The tasks for the second core are:

- **Init_Task_C1**: for core initialization, must have the highest priority and must be on autostart.
- **Rte_Time_Task_C1**: to run the cyclic events of the SWC that runs on the second core.
- **SchMDiagStateTask_20ms_C1**: running the EcuM and BswM main functions.
- **TASK_RTE_ModeSwitchEvent**: for mode switch events on second core.
- **Rte_Event_Task_C1**

The BSW_Com task is needed for Rte to handle the interpretation operation. As the cyclic events of the SWC is executed on the second core, you can delete the Rte_Time_Task.

5.2.3.4. OsArtefactMapping

You must map each Os artefact to the belonging OsApplication.

Step 1
Open the **OsApplication**.

Step 2
Map the artefacts on the corresponding tabs.

Step 3
Map the following artefacts to the newly created OsApplication that belongs to core 1:

- OSAppAlarmRef: AlarmIncrementRteCounter_C1

- OSAppCounterRef: HwCounter_C1, Rte_Counter_C1
- OSTaskRef: Init_Task_C1, Rte_Event_Task_C1, Rte_Time_Task_C1, SchMDiagStateTask_20ms_C1, RTE_ModeSwitchEvent_Task_C1

Step 4
Map the OSTaskRef: BSW_Com artefact to the OsApplication that belongs to core 0.

5.2.4. BswM

BswM supports multicore functionality with a module instance for each core. Each BswM instance must have its own configuration set. The mode request port definitions needed by BswM to communicate between the BswM instances are provided in an arxml file, and the files are already imported.

5.2.4.1. Changes for Core 0 BswM configuration

Starting Rte on both cores from BswM and sending a mode change notification from C0 to C1

Step 1
In the **BswMSwitchPort** tab > **BswMConfig_C0**, create a new BswMSwitchPort.

Step 2
Name it **BswMSwitchPort_SchM**.

Step 3
Set **BswMModeSwitchInterfaceRef** to **BwwMMode_CrossCore**.

Step 4
In the **Configuration** tab, rename the existing **BswMConfig** to **BswMConfig_C0** for consistent naming.

Step 5
In the **General** tab > **BswMConfig_C0**, enable **BswMPartitionRef** and select /Os/Os/C0.

Step 1
In the **BswMAction** tab > **BswMConfig_C0**, create a new BswMAction.

Step 2
Name it **BswM_Act_SchMSwitch_Startup**.

Step 3
Set **BswMAvailableActions** to **BwwMSchMSwitch**.

Step 4
Set **BswMSchMSwitchPortRef** to **BswMSwitchPort_SchM**.

Step 5
Set **BswMSchMSwitchedMode** to **EB_INTGR_BSWM_CROSS_CORE_STARTUP_TWO_A**.

Step 6
In **BswMDataTypeMappingSetRef** tab > **BswMConfig_C0**, create a new **BswMDataTypeMappingSetRef** and set it to **BswMModeMapping_CrossCore**.

Procedure 5.1. New BswMSwitchPort is required to send a mode change notification from C0 to C.

Step 1
In **BswMConfig_C0** > **BswMActionList**, open **BswM_ActLst_StartupTwoA**.

Step 2
In the **BswMActionListItem** tab, create a new entry.

Step 3
Right-click the newly created item, and select **Move element up** to place it immediately after the item that references BswM_Act_RteStart.

Step 4
Set **BswMActionListItemRef** to **BswM_Act_SchMSwitch_Startup**.

Step 5
Trigger **BswM_Act_SchMSwitch_Startup** from an action list.

Procedure 5.2. A new BswMAction is required to switch the mode for C1.

5.2.4.2. Changing Core 1 BswM configuration

The change notification on C1 must be received and the Rte must be restarted.

Step 1
In the **BswMModeRequestPort** tab > **BswMConfig_C1**, create a new **BswMModeRequestPort**.

Step 2
Name it **BswM_ModeReqPort_EcuState_CrossCore**.

Step 3
Set **BswMRequestProcessing** to **BSWM_DEFERRED**.

Step 4
Set **BswMModeRequestSource** to **BswMBswMModeNotification**.

Step 5
Set **BswMBswModeDeclarationGroupPrototypeRef** to **CurrentCrossCoreMode**.

Step 6
In the **Configuration** tab, create a new **BswMConfig** and rename it **BswMConfig_C1**.

Step 7
In the **General** tab > **BswMConfig_C1**, enable **BswMPartitionRef** and select /Os/Os/C1.

Step 8
In **BswMConfig_C1**, enable **BswMDataTypeMappingSets**.

A new BswMModeRequestPort is required to receive a mode change notification from C0.

Step 1
In the **BswMDataTypeMappingSetRef** tab > **BswMConfig_C1**, create a new **BswMDataTypeMappingSetRef**, and set it to **BswMModeMapping_CrossCore**.

Step 2
In the **BswMAction** tab > **BswMConfig_C1**, create a new BswMAction.

Step 3
Name it **BswM_Act_RteStart**.

Step 4
Set **BswMAvailableActions** to **BswMUserCallout**.

Step 5
Add **Rte_Start()** in **BswMUserCalloutFunction**.

A new BswMAction is required to start Rte for C1.

Step 1
In **BswMActionList** > **BswMConfig_C1**, create a new item and name it **BswM_ActLst_StartupTwoA**.

Step 2
In the **General** tab, set **BswMActionListExecution** to **BSWM_TRIGGER**.

Step 3
In the **BswMActionListItem** tab, create a new entry.

Step 4
Set **BswMActionListItemRef** to **BswM_Act_RteStart**.

You must trigger BswM_Act_RteStart from an action list.

Step 1
In the **BswMModeCondition** tab > **BswMConfig_C1**, create a new item and name it **BswM_ModeCond_StartupTwoA**.

Step 2
Set **BswMConditionType** to **BSWM_EQUALS**.

Step 3
Set **BswMConditionMode** to **BswM_ModeReqPort_EcuState_CrossCore**.

Step 4
Set **BswMConditionValue** to **BswM_ModeDeclaration**.

Step 5
Set **BswMModeValueRef** to **EB_INTGR_BSWM_CROSS_CORE_STARTUP_TWO_A**.

A BswM_ModeCondition is required to detect the mode switch from core C0.

Step 1
In the **BswMLogicalExpression** tab > **BswMConfig_C1**, create a new item and name it **BswM_LogEx_StartupTwoA**.

Step 2
In the **BswMArgumentRef** tab, create a new entry.

Step 3
Set the newly created **BswMArgumentRef** to **BswMArbitration/BswM_ModeCond_StartupTwoA**.

Mode arbitration requires a BswMLogicalExpression

Mode arbitration must be described with a BswMRule.
Step 1
In the **BswMRule** tab > **BswMConfig_C1**, create a new item and name it **BswM_Rule_StartupTwoA**.

Step 2
Set **BswMRuleInitState** to **BSWM_FALSE**.

Step 3
Set **BswMRuleExpressionRef** to **BswM_LogEx_StartupTwoA**.

Step 4
Enable **BswMRuleTrueActionList**, and set it to **BswM_ActLst_StartupTwoA**.

5.2.5. Connecting Software Components with basic software

As the BswM instances were changed, you must map again the used ports in the **Connection** editor by adding the component prototypes. The unvalid references are automatically removed when you open the **Compositions and Connections** editor.

Step 1
In the **Sidebar** view > **System** category, select **Edit Compositions and Connections**.

Step 2
Add **Component Prototypes**.

Step 3
In the **Entity** list, right-click on **TopLevelComposition**, and select **Add Prototypes**.

Step 4
Check the box for the needed component types, **BswM_C0** in our case.

Step 5
Click **OK** to add a component prototype for each selected component type.

Step 6
Expand a prototype to make the assignment, right-click on the port of the prototype, and select **Add Connector**.

Step 7
Expand a component prototype, and select a port. If the **OK** button is not enabled, the selected port is not compatible.

- Use the buttons at the top of the dialog window to restrict the entities of the tree, and show or hide incompatible ports and already connected ports.
- Use the type filter text to restrict the shown ports by name. Select **OK** to add the connection.

Step 8
Connect each port for **BswM_C0_Prototype** with a compatible port. There should be only one compatible port available.

5.2.6. Rte

5.2.6.1. Implementation selection

To fix the conflicts of the Basic software module instance names of BswM, you must rename them accordingly.

Step 1
Click the **Implementation Selection** tab.

Step 2
In **Basic software module instances**, rename the BSW_BswM that has the Bsw module implementation set to **BswImplementation_0** to **BSW_BswM_C0**.

Step 3
In **Basic software module instances**, rename the **BSW_BswM_1_C1** to **BSW_BswM_C1**.

5.2.6.2. Partition

Rte uses partitions to separate the different SW parts. A communication is needed between the partitions.

Setting communication method between the partitions.

Step 1
Set **Partitioning support** to **SharedMemory**.

Step 2
Enable BSW Os task required.

Step 3
Set BSW Os task to **BSW_Com**.

Step 4
Set a valid number to BSW to send signal queue length and BSW to send signal group queue length. For our use case, 2 should be enough.

Mapping SW component and Bsw instances to the partitions.

Step 1
In **Software components instance partitioning**, change the Os application for SWC_CyclicCounter to C1.

Step 2
In **Software components instance partitioning**, map Os application for BswM_OsApplication_C0_Prototype to C0.

Step 3
In **Basic module instance name**, map the Os application for every entry to C0 except EcuM and BswM with suffix "1".

Step 4
In **Os counter partition assignment**, map the Os application C0 to HwCounter_0 suffix.

Step 5
In **Os counter partition assignment**, map the Os application C1 to HwCounter_1 suffix.

5.2.6.3. Event Mapping

You must map each event to a task on the corresponding partition.

- Map to SchMDiagStateTask_20ms_C1:
  - EcuM_MainFunction
  - BswM_MainFunction
- Unmap from Rte_Time_Task_C0: CyclicEvent...
- Unmap from Rte_Event_Task_C0:
  - ModeSwitchedRunTwo
  - ModeSwitchedPrpShutdown
- Map to TASK_RTE_ModeSwitchEvent_C1: MSE_BswM_ModeReqPort_EcuState_CrossCore_-...
- Map to Rte_Time_Task_C1: CyclicEvent_-...

- Map to Rte_Event_Task_C1:
  - EcuM_UpdateSleepMode
  - ModeSwitchedRunTwo
  - ModeSwitchedPrpShutdown
  - SetCounterOperationInvoked...
- Map to Rte_Event_Task_C0:
  - EV_ReportError_SWC_0
  - EV_SetRamBlockStatus_PersistentCounterValue
  - SetDiscreteValueOperationInvoked
- Map to TASK_RTE_ModeSwitchEvent: All MSE_BswM_ModeReqPort_AppState_-.

If the mapping of tasks to OsApplication is correctly made, an error is generated if events are not mapped on the correct core.

5.2.6.4. BswMModeMapping

As the BswM instances were changed, you must remake the mapping between Dcm and BswM.

- DiagnosticSessionControl
- EcuReset
- CommunicationControl

For the BswM cross core communication, map the following mode groups between BswM_C0 and BswM_C1: BswM_ModeSwitchPort_C0_BswMSwitchPort_SchM on the left to EcuState_CrossCore on the right.

5.2.7. Integration code

Some files and defined names are extended with the OsApplication names because of the multi-core configuration.

- **Include Files**: Rte generates separate source and header files for each partition, used Rte include files must be updated in the integration code.
  - In Eb_Intgr_BswM_UserCallouts.h: #include <Rte_BswM_Type.h> must be changed to #include <Rte_BswM_C0_Type.h>
- **MemMap sections**: separate memory sections are generated for each core. Memory mapping from SWC implementation must be updated based on which core the SWC is located.

- **Init Task**: task names might be changed during the configuration. Check the name for the Init task from C0 and add the implementation for the init task from C1.

  The implementation of the C1 Init task must contain a call to EcuM_StartupTwo() function.

```
TASK(Init_Task_C1) {

EcuM_StartupTwo(); /* Task "Init_Task" has to invoke function "EcuM_StartupTwo". */

(void) TerminateTask(); }
```

- EcuM_CalloutStubs: EcuM provides an additional callout function EcuM_OnCoreSync. The prototype of this function can be found in `\output\generated\templates\EcuM_Callout_Stubs.c`

5.2.8. ARXML files

The memory section names now contain the OsApplication names, and the memory sections from ARXML files must be adapted. Each SWC-IMPLEMENTATION contains MEMORY-SECTION definitions whose SHORT-NAME must be updated based on the generated section names from MemMap. The names are similar to the ones changed in **Integration Code/MemMap** sections.

5.2.9. Memory Mapping

The mapping of variables and code may have an important impact on the performance. Each core might have a flash and a ram memory to which the access is optimized.

5.3. Memory protection usage

Memory protection is used to restrict the write access to memory regions. The Operating System takes care of the memory protection and the following applies:

- Non-trusted OS applications run in a non-privileged processor mode, and are subject to memory protection. To use memory protection, you must configure a number of OS-Application objects and assign your tasks, ISRs and other OS objects to them.
- Each task and ISR belonging to a non-trusted OS-Application have private data with write access. Access by another task or ISR of the same OS-Application or by other (non-trusted) OS-Applications is not permitted.
- Each non-trusted OS-Application has shared data to which all tasks, ISRs and (application-specific) hook functions have write access. Access by other (non-trusted) OS-Applications is not permitted.

- Each executing object has write access on its own stack. Write access by another task or ISR of the same OS-Application or by other (non-trusted) OS-Applications is not permitted.
- When a memory protection violation occurs, the OS calls the ProtectionHook() function that is supplied by the application. This determines what the OS does to the task or ISR that caused the violation.
  - The reactions range from terminating the task or ISR to shutting down the OS (core).
  - The minimum reaction is to terminate the task or ISR.

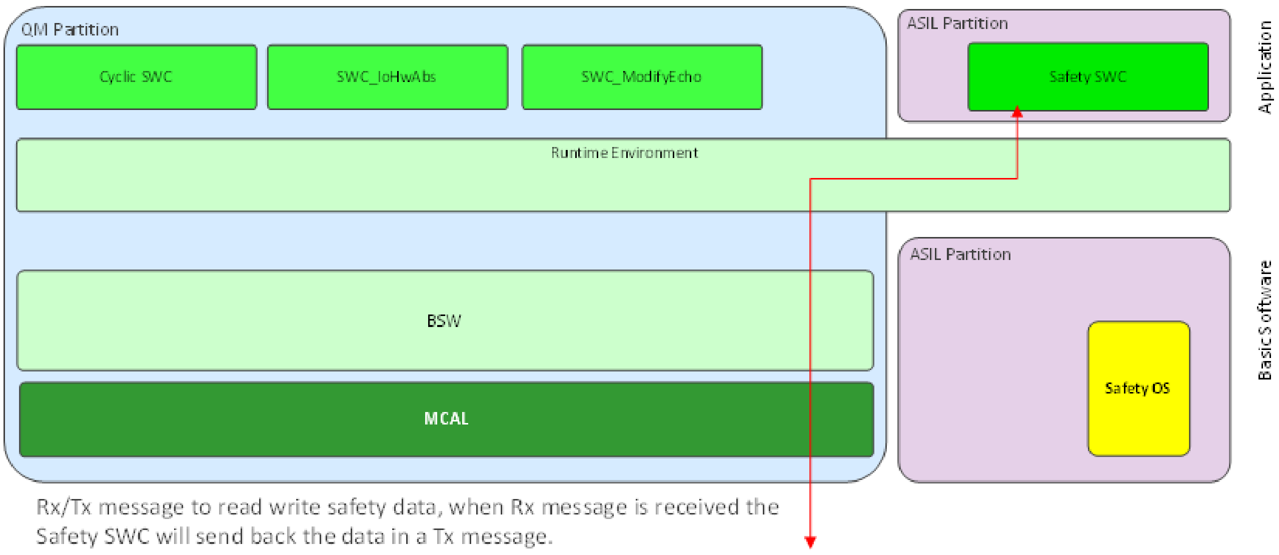


Figure 5.1. Architecture concept to enable the usage of memory protection

Two OS Applications are created: one holds the QM part of the SW and one holds the newly introduced Safety SWC. The Safety SWC data are protected against unwanted modification.

5.3.1. System Configuration update

Step 1
Extend the existing **SystemExtract_SystemModel** importer with the arxml files containing the system description and SWC description.

Step 2
Update the system model.

5.3.1.1. Update System Description Importer

To extend the SystemExtract_SystemModel importer with the arxml files containing the system description and SWC description:

Step 1
Open the **Create, manage and run im- and exporters** window.

Step 2
Select the **SystemExtract_SystemModel**, and click the **All Models** tab.

Step 3
Select **Add** to add a file, and go to `Project root/system/MemoryProtection.arxml`, or select **New** and go to `MemoryProtection.arxml`.

Step 4
Select **Add** to add a file, and go to `Project root/system/SWC_Safety.arxml` or select **New**, and go to `system/SWC_Safety.arxml`.

Step 5
Click **OK**.

5.3.1.2. Running Importer

Execute multiple task **SystemDescriptionImport_Run** to update the system model.

5.3.2. Configuring memory protection

This section explains how to configure the basic software modules.

- **Os**: extends the OS configuration with needed artifacts.
- **BswM**: adapts the startup behavior.

5.3.2.1. Configuring OS module

OsOS tab

Step 1
Set **OsProtection** to ON.

Step 2
Set **OsUseLastError** to True.

Step 3
Set **OsProtectionHook** to True.

Step 4
Create **OsCounter** for the second partition:

Step 4.1
Rename the existing **HwCounter** to **HwCounter_Qm**.

Step 4.2
Duplicate **HwCounter_Qm** to **HwCounter_Safety**.

Step 4.3
In **HwCounter_Safety**, change **OsCounterTricoreTimer** to a timer that is not used, for example STM0_-T1.

Step 4.4
Create **OsTasks** for the second core. New tasks are needed for initialization, execution of Rte events, and inter partition communication.

OsTasks tab

Step 1
Duplicate the existing **Rte_Event_Task** and change its name, for example to Safety_Event to execute the events from Safety SWC.

Step 2
Create a new **BSW_Com** task used by Rte to handle interpreation communication.

- In **BSW_Com**, set **OsTaskPriority** to 49.
- In **BSW_Com**, set **OsStacksize** to 512.

Step 3
Create a new **Rte_Start_Qm** task to start the Rte on the Qm partition.

- In **Rte_Start_Qm**, set **OsTaskPriority** to 51.
- In **Rte_Start_Qm**, set **OsStacksize** to 512.

Step 4
Create a new **Rte_Stop_Qm** task to stop the Rte on the Qm partition.

- In **Rte_Stop_Qm**, set **OsTaskPriority** to 51.
- In **Rte_Stop_Qm**, set **OsStacksize** to 512.

Step 5
Create a new **Rte_Start_Safety** task to start the Rte on the Safety partition.

- In **Rte_Start_Safety**, set **OsTaskPriority** to 51.
- In **Rte_Start_Safety**, set **OsStacksize** to 512.

Step 6
Create a new **Rte_Stop_Safety** task to start the Rte on the Safety partition.

- In **Rte_Stop_Safety**, set **OsTaskPriority** to 51.
- In **Rte_Stop_Safety**, set **OsStacksize** to 512.

OsMikrokernel tab

Step 1
In **MkMemoryProtection**, create a new **MkMemoryRegion Io** to define the IO space.

- In Io, set **MkMemoryRegionInitThreadAccess** to True.
- In Io, set **MkMemoryRegionOsThreadAccess** to True.
- In Io, set **MkMemoryRegionShutdownAccess** to True.
- In Io, set **MkMemoryRegionGlobal** to True.
- In Io, set **MkMemoryRegionShutdownHookAccess** to True.
- In Io, set **MkMemoryRegionKernelAccess** to True.

Step 2
Create a new MkMemoryRegion Fee to define the region needed for Flash emulated EEPROM.

Step 3
Create a new MkMemoryRegion EthBuffer to define the region for the buffers needed by the eth driver.

Step 4
Create a new MkMemoryRegion OsApp_Qm to define the region for Qm Os Application.

- In **OsApp_Qm**, set **MkMemoryRegionInitThreadAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionOsThreadAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionProtHookAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionShutdownAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionIdleThreadAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionErrorHookAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionShutdownHookAccess** to True.
- In **OsApp_Qm**, set **MkMemoryRegionKernelAccess** to True.

Step 5
Create a new **MkMemoryRegion OsApp_Safety** to define the region for Safety Os Application.

OsMikrokernel tab

Step 1
In **MkThreadCustomization**, enable MkInitThreadMode, and set it to SUPERVISOR.

Step 2
Enable **MkShutdownThreadMode**, and set it to SUPERVISOR.

Step 3
Enable **MkShutdownHookMode**, and set it to SUPERVISOR.

OsApplication tab

Step 1
Create a new application, and name it OsApp_Safety.

Step 2
Rename the existing one to OsApp_Qm.

OsApplication

Step 1
Open the **OsApplication**, and map each artefact with the corresponding OsApplication.

You must map the following artefacts to the newly created OsApplication OsApp_Safety:

- OSAppCounterRef: HwCounter_Safety
- OSAppTaskRef: Safety_Event, Rte_Start_Safety, Rte_Stop_Safety
- OsAppMkMemoryRegionRef: Io

You must map the following artefacts to the newly created **OsApplication OsApp_Qm**:

- OSAppCounterRef: HwCounter_Qm
- OSAppTaskRef: BSW_Com. Rte_Start_Qm, Rte_Stop_Qm
- OsAppMkMemoryRegionRef: Io, Fee, EthBuffer

OsApp_Qm must have the right to start the Rte_Start and Rte_Stop tasks of each partition:

- In the **OsTask** tab, open Rte_Start_Safety and in OsTaskAccessingApplication tab, add OsApp_Qm.
- In the **OsTask** tab, open Rte_Stop_Safety and in OsTaskAccessingApplication tab, add OsApp_Qm.
- In the **OsTask** tab, open Rte_Start_Qm and on OsTaskAccessingApplication tab, add OsApp_Qm.
- In the **OsTask** tab, open Rte_Stop_Qm and on OsTaskAccessingApplication tab, add OsApp_Qm.

5.3.2.2. Configuring BswM module

Rte start was moved from BswM action to a partition specific task activation that is done in a callout function.

Step 1
In the **BswM** editor, click the **BswMAction** tab > **BswMConfig**, and open **BswM_Act_RteStart**.

Step 2
Change the **BswMUserCalloutFunction**, to Eb_Intgr_Rte_Start().

5.3.2.3. Connecting Software Components with basic software

To run the **Compositions and Connections editor**, in the **Sidebar** view > **System** category, select **Edit Compositions and Connections**.

Adding Component Prototypes

Step 1
Right-click on **TopLevelComposition** in the **Entity** list, and select **Add Prototypes**.

Step 2
Select the SWC_Safety component types.

Step 3
Click **OK** to add a component prototype for each selected component type.

Adding connections

Step 1
To make the assignment, expand a prototype, right-click on the port of the prototype, and select **Add Connector**.

Step 2
Expand a component prototype, and select a port.

Step 3
Click **OK** to add the connection.

Step 4
Connect each port for SWC_Safety with a compatible port. Only one compatible port should be available.

If the **OK** button is disabled, the selected port is incompatible.

- In the **Add Connector** window, use the buttons to set the tree and to show or hide both incompatible and already connected ports.
- Use the **type filter text** text box to filter the ports by name.

5.3.2.4. Configuring Rte module

Step 1
In the **Rte Editor**, click the **Partitioning** tab, and set **Partitioning support** to **SharedMemory**.

Step 2
Enable BSW Os task required.

Step 3
Set BSW Os task to BSW_Com.

Step 4
Set a valid number to BSW send signal queue length and BSW send signal group queue length. For our use case, 2 is enough.

Step 5
In **Software components instance partitioning** map, set Os application for SWC_Safety to OsApp_Safety.

Step 6
In **Basic module instance name**, map the Os application for every entry to OsApp_Qm.

Step 7
In **Os counter partition assignment**, map HwCounter_Qm to Os application OsApp_Qm.

Step 8
In **Os counter partition assignment**, map HwCounter_Safety to Os application OsApp_Safety.

Step 9
Select the **Event Mapping** tab, and select a task in the **Task** drop-down list box, then select **Map the runnable entity selected above to the task selected below**.

Step 10
Map to **Safety_EventUpdateSafeData**.

5.3.3. Integration Code

This section is an overview of the changes triggered by the memory protection configuration. Some files and some defined names are extended with the OsApplication names because of the use of partitioning.

Name	Description
Include files	Rte generates separate source and header files for each partition, and the included Rte files must be updated in the integration code.
Task activation for Rte start stop	Applies the following changes in the `Eb_Intgr_BswM_UserCallouts.c` file: ▶ Replaces call to Rte_Stop() with task activation for Rte_Stop_Qm and Rte_Stop_Safety. ▶ Creates the Eb_Intgr_Rte_Start function where the Rte_Start_Qm and Rte_Start_Safety tasks are activated. ▶ In Eb_Intgr_BswM_UserCallouts.h, add the function prototype for Eb_Intgr_Rte_Start.

Name	Description
	▶ Creates a new file named Eb_Intgr_Rte_Tasks.c, and implements the tasks for Rte_Stop_Qm, Rte_Stop_Safety, Rte_Start_Qm, Rte_Start_Safety to call the corresponding Rte_Start_xxx, Rte_Stop_xxx function for the partitions.
Protection Hook	A protection hook must be implemented. Creates a new file named Eb_Intgr_Os_-Hooks.c and implements the ProtectionHook function.
SWC Safety	Creates a new file named SWC_Safety.c and implement the SWC_Safe_Da-ta_Received runnable entity.

Linker script generation	The linker script generator must be adapted to define the memory regions by ad-dress or by symbol inside the make files.
	
	▶ In genld-TRICORE-Mk.pl, search for the code section where "symbols for user configured memory region" is handled, and replace:
	
	`$LdBackend->printSymbolDefinition($mr_sym,$dest_adr);`
	
	with *:*
	
	`# check if the memory region is defined by address or by name if((rindex $dest_adr, "0x", 0) ){ $LdBack-end->printSymbolDefinition($mr_sym, "\"$dest_adr\""); } else { $LdBackend->printSymbolDefinition($mr_sym, $dest_-adr); }`
	
	Note: The linker script is generated only if there is no linker script available in the "output\generated\" folder.

Objects assignation to Os Applications	The object files must be assigned to the corresponding OS Applications. This is done in a make file. In the Merged_Makefile.mak from util folder, do the following assignment:
	
	`OBJS_OsApp_XXX:=obj_file_name*\`
	
	`......................`
	
	`obj_file_name*\`
	
	▶ OBJS_OsApp_Qm must contain all the object files from the basic software, and all the SWCs beside SWC_Safety.
	
	▶ OBJS_OsApp_Safety must contain all the object files that are related to SWC_Safety. This includes the generated ones from Rte.
	
	All the object files are successfully mapped if the anon sections from the generat-ed map file are empty.

Name	Description

Memory region address assignment	For each memory region defined inside the OS, you must define the configuration address assignment in a make file.
	
	The following values are valid for **Essentials Solutions** for the **TC29XT** platform. In the **Merged_Makefile.mak** from the util folder, follow this assignment:
	
	▶ Memory region Io:
	MK_MR_START_Io = "MK_RSA_Io=0xC3F88000"
	MK_MR_LIMIT_Io = "MK_RLA_Io=0xFFFFC000"
	▶ Memory region Fee: this memory region defines the area needed for FlashEmulated EEPROM.FlsBaseAddress defines the start address and FlsTotalSize the size of the region.
	MK_MR_START_Fee = "MK_RSA_Fee=0xAF000000"
	MK_MR_LIMIT_Fee = "MK_RLA_Fee=0xAF0C0000"
	▶ Memory region EthBuffer:
	MK_MR_START_EthBuffer = "MK_RSA_EthBuffer=_lc_gb_EthBuffer"
	MK_MR_LIMIT_EthBuffer = "MK_RLA_EthBuffer=_lc_ge_EthBuffer"
	▶ Memory region OsApp_Qm: this memory region is used for the Qm Os Application.
	MK_MR_START_OsApp_Qm = "MK_RSA_OsApp_Qm=_lc_ub_data_C0.OsApp_Qm"
	MK_MR_LIMIT_OsApp_Qm = "MK_RLA_OsApp_Qm=_lc_ue_bss_C0.OsApp_Qm"
	▶ Memory region OsApp_Safety: this memory region is used for the Safety Os Application.
	MK_MR_START_OsApp_Safety = "MK_RSA_OsApp_Safety=_lc_ub_data_C0.OsApp_Safety"
	MK_MR_LIMIT_OsApp_Safety = "MK_RLA_OsApp_Safety=_lc_ue_bss_C0.OsApp_Safety"

Table 5.1.