

AUTOSAR

Fundamentals

V21.0.0 | 2021-08-17

Agenda

► **Overview and Objectives**

AUTOSAR Application

AUTOSAR RTE

AUTOSAR BSW

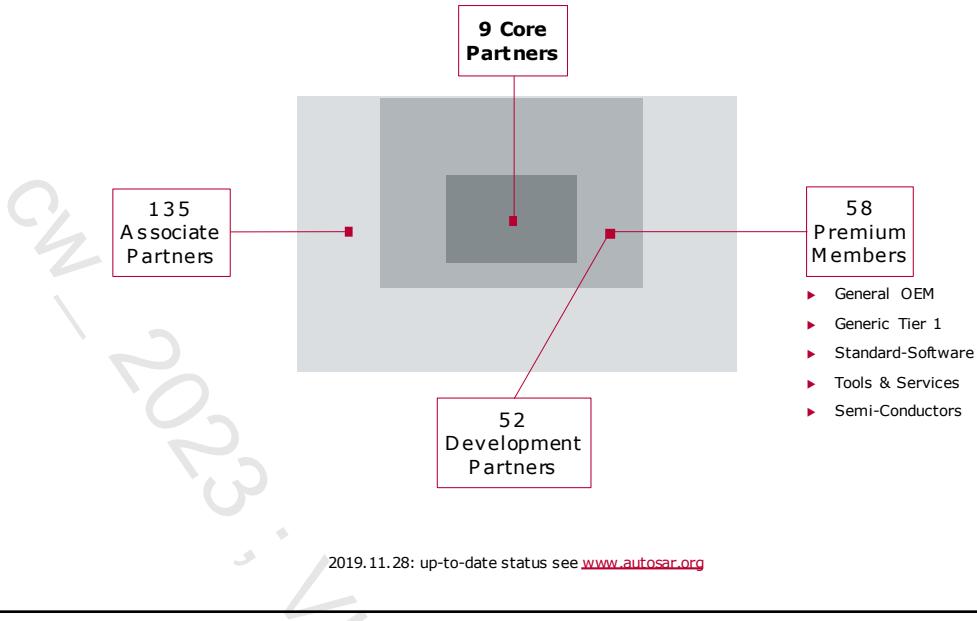
AUTOSAR Methodology

AUTOSAR in Practice

Appendix



AUTOSAR Partnership



3

- ▶ The AUTOSAR development partnership was founded in 2003 with the goal of establishing an open standard for electrical/electronic architecture in the automotive industry. It is a worldwide development partnership of automotive OEMs, suppliers and other companies in the software, semiconductor and electronics industries.
- ▶ Core Partner: These nine companies have brought together their respective areas of expertise to define an automotive open system architecture standard to support the needs of future in-car applications.

	<i>premium members</i>	<i>development members</i>	<i>associate members</i>	<i>attendees</i>
Right to use the AUTOSAR technology, with a free-of-charge license and royalty-free for automotive applications	■	■	■	□
Access to current information and specifications	■	■	■	■
Leadership of working groups	■	□	□	□
Cooperation in working groups	■	■	□	■
Access to AUTOSAR related IP of all other AUTOSAR members free of charge	■	■	■	□
Right to vote on Partnership issues	■	□	□	□
has to assign staff with such skills as it is required in the participating working group.	■	■	□	□
contributes work package related Intellectual Properties	■	■	□	□
Administration fee per year (EUR)	17500	-	10000	-

AUTOSAR Partnership

Core Partners:



58 Premium Members:

Advanced Driver Information Technology	actia	APTIV	APTIV	HONDA	Huawei	HYUNDAI MOTOR GROUP	IAV	NXP	Panasonic	RENAULT	RENESAS
ADT Corporation	AVL	Aptiv	APTIV	Honda Motor	Huawei Technologies Co., Ltd.	Hyundai	IAV	NXP Semiconductors B.V.	Panasonic Corporation	Renault SAS	RENESAS Electronics Corp.
ARC CORE	OAI	Autoliv	Baidu	Intel	iTK	CETIC	SCSK	sodius™	ST Microelectronics	SUMITOMO ELECTRIC INDUSTRIES, LTD.	SUMITOMO ELECTRIC INDUSTRIES, LTD.
CEA	CHRYSLER	Deloitte	dSPACE	ixia	JTEKT	KPIT	TATA CONSULTANCY SERVICES	TATA ELXSI	TATA MOTORS	Tata Electronics	ThyssenKrupp
EE	EGL	ETAS	Fraunhofer	LEAR	CLUXOFT	MathWorks	Velox	VECTOR ➤	VOLVO	VOLVO	VOLVO Car Corporation
Great Wall	Green Hills Software	HCL	HELLF	Hitachi	NEC	NISSAN Motor Company	NVIDIA	WIND	ZF Friedrichshafen AG	ZF Friedrichshafen AG	ZF Friedrichshafen AG

+ [52 Development Partners](#) + [135 Associate Partners](#)up-to-date status see
www.autosar.org

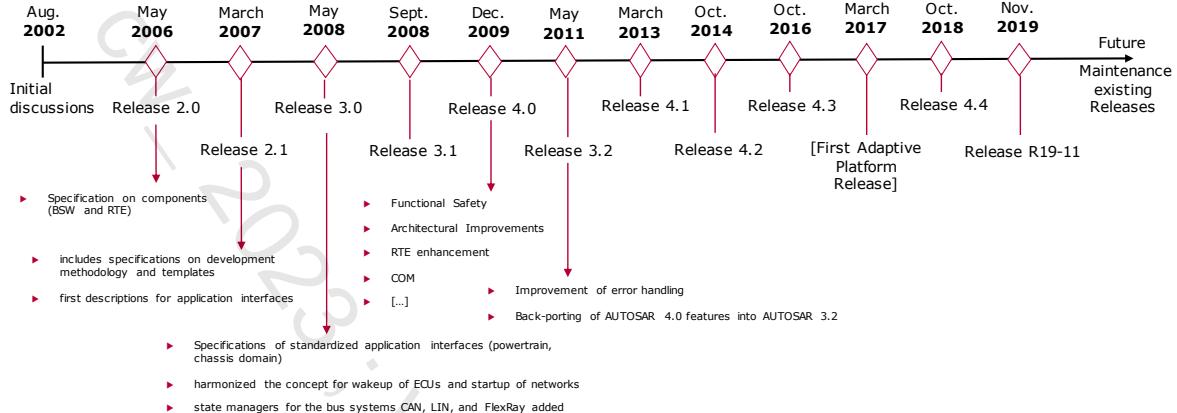
4

- The AUTOSAR development partnership was founded in 2003 with the goal of establishing an open standard for electrical/electronic architecture in the automotive industry. It is a worldwide development partnership of automotive OEMs, suppliers and other companies in the software, semiconductor and electronics industries.
- Core Partner: These nine companies have brought together their respective areas of expertise to define an automotive open system architecture standard to support the needs of future in-car applications.

▶

	premium members	development members	associate members	attendees
Right to use the AUTOSAR technology, with a free-of-charge license and royalty-free for automotive applications	■	■	■	□
Access to current information and specifications	■	■	■	■
Leadership of working groups	■	□	□	□
Cooperation in working groups	■	■	□	■
Access to AUTOSAR related IP of all other AUTOSAR members free of charge	■	■	■	□
Right to vote on Partnership issues	■	□	□	□
has to assign staff with such skills as it is required in the participating working group.	■	■	□	□
contributes work package related Intellectual Properties	■	■	□	□
Administration fee per year (EUR)	17500	-	10000	-

History



5

► AUTOSAR 2.0

- ▶ specifications on components (BSW and RTE)

► AUTOSAR 2.1

- ▶ includes specifications on development methodology and templates
- ▶ first descriptions for application interfaces (body and interior electronics)

► AUTOSAR 3.0

- ▶ > 20 compositions from the body, powertrain, and chassis domain
- ▶ Specifications of standardized application interfaces (powertrain, chassis domain)
- ▶ harmonized the concept for wakeup of ECUs and startup of networks
- ▶ state managers for the bus systems CAN, LIN, and FlexRay added

► AUTOSAR 3.1

- ▶ OBDII (new revisions of the DCM, DEM, FIM specifications,...)

► AUTOSAR 4.0 Rev1 (2009-12)

- ▶ Functional Safety (Memory Partitioning, Time Determinism, Program Flow Monitoring, E2E, BSWM Defensive Behavior, Dual µC, E-Gas Monitoring)
- ▶ Architectural Improvements (Error Handling, Multi Core, Boot loader Interaction)
- ▶ RTE enhancement (Triggered Events, Integrity and Scaling at Ports, API Enhancement)
- ▶ COM (LIN 2.1 Spec, FR 3.0 Spec, FlexRay ISO TP, XCP for ASR, Large Data Types, TCP/IP + Dolp, J1939Tp, TTCAN)
- ▶ Functional (NM coordination, ASR Scheduler)
- ▶ Conformance Test specifications (CT Specs, only Rev1 and Rev2, but not >Rev2)

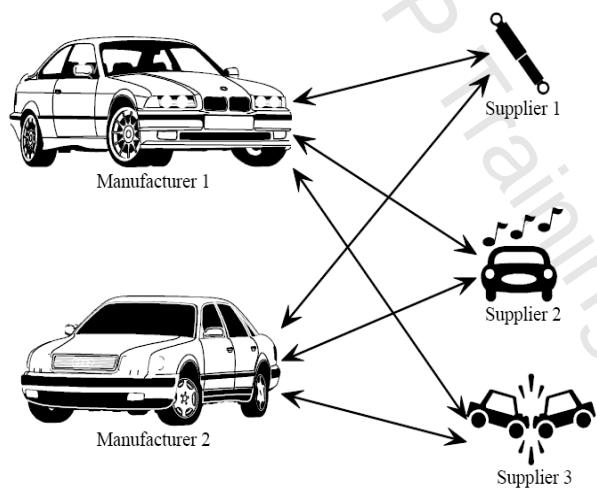
- ▶ Detailed overview: [#132. Appendix – Concepts of AUTOSAR 4.x](#)
- ▶ AUTOSAR 3.2 Rev1 (2011-05)
 - ▶ Partial Networking
 - ▶ Robustness Features (state manager modules)
 - ▶ Improvement of error handling (e.g. production vs. development errors)
 - ▶ Back-porting of AUTOSAR 4.0 features into AUTOSAR 3.2
 - > Parts of the Safety Concept (E2E communication protection)
 - > Extended CDD Concept
 - > BSW Mode Manager
 - > FlexRay ISO TP

The Situation Today

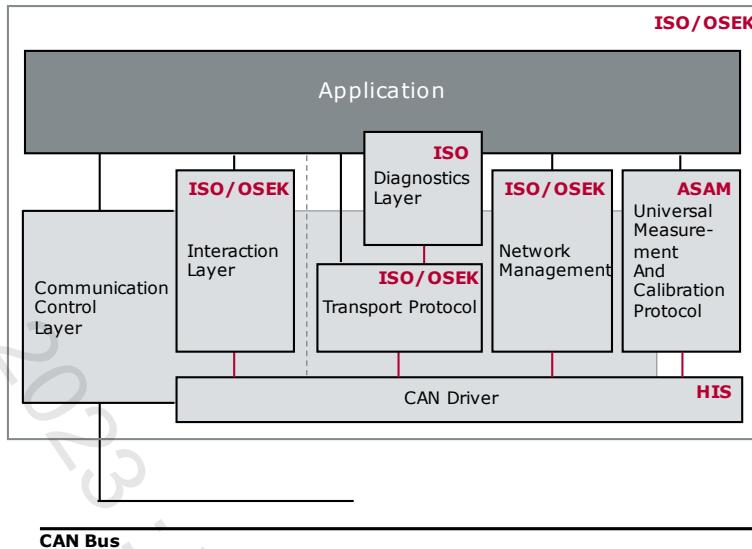
- ▶ Growing E/E complexity
- ▶ Quantity of functions implemented in **software** rising very steeply (e.g. in the field of driver assistance systems)
- ▶ Many different **hardware platforms** are used
 - ▶ Embedded systems traditionally do not support full **hardware abstraction**
 - ▶ Limited **modularity** of the software
 - ▶ Poor **reusability**: Software must often be rewritten from scratch when hardware (processor type) is changed
- ▶ **Life cycle shortfall**: The life of cars is usually longer than that of their ECUs (need for sustainable spare parts, especially software for maintenance reasons)
- ▶ **Variability**: Suppliers must support a large variety of OEMs and vehicle platform variants with their software

6

- ▶ The automotive industry is faced with great challenges in upcoming years. Today in-vehicle electronics have already reached a level of complexity that can hardly be mastered. Nonetheless, customers of the OEMs want even greater diversity in variants and equipment features. Moreover, automotive OEMs also need to observe criteria that are generally not even experienced by the end customer. Some examples of these non-functional requirements are diagnostic capability, availability and upgradeability. For suppliers the situation is complicated as well. Not only do they need to master the model diversity of one OEM; they need to satisfy the requirements of a number of different customers. Additionally suppliers must align themselves with the different product creation processes of different customers. This affects the development process, the electronic architecture, and last but not least the tool environment of the companies.



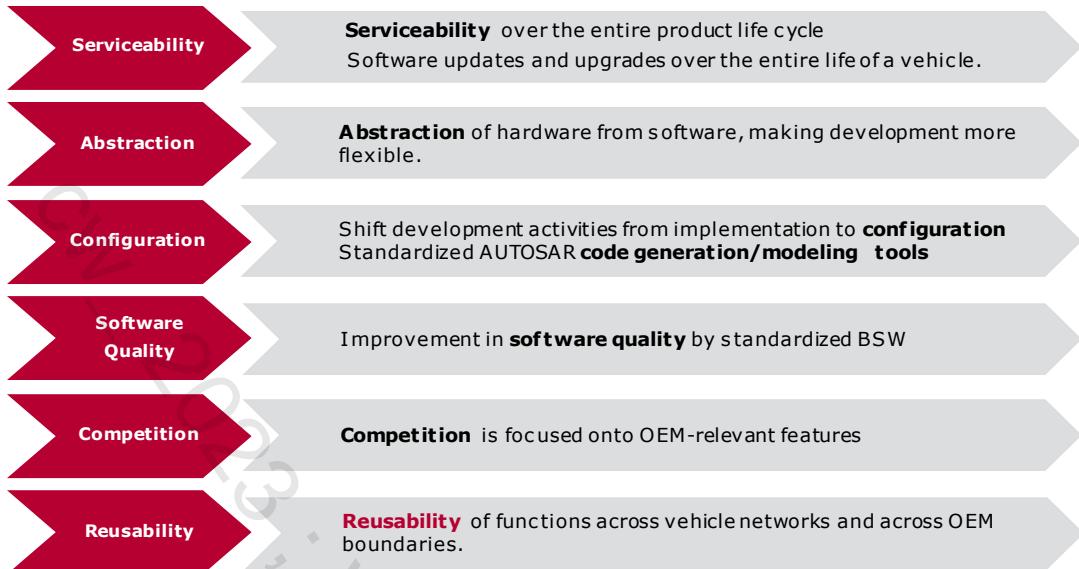
The Situation Before AUTOSAR



7

- ▶ HIS Herstellerinitiative Software [“OEM Software Initiative”]
- ▶ ASAM Association for Standardization of Automation and Measuring Systems
- ▶ ISO International Organization for Standardization
- ▶ OSEK Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug

Objectives

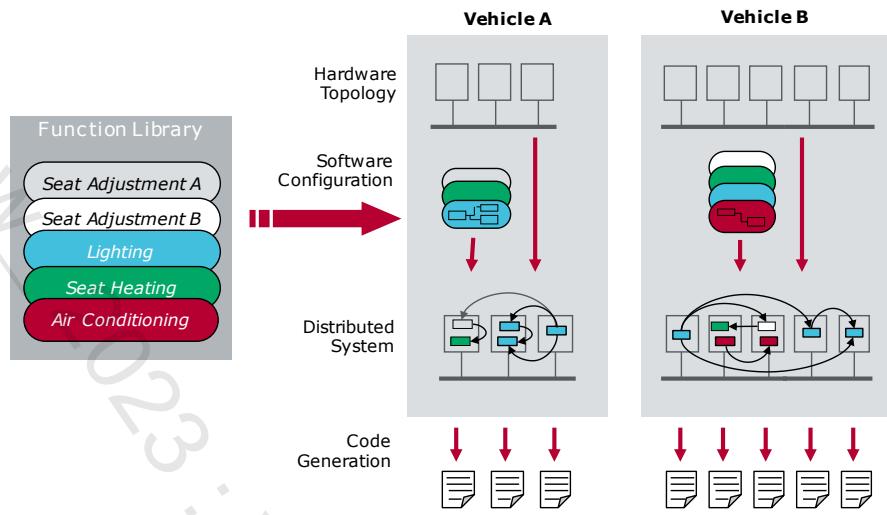


8

- Software quality is improved by uniform exchange formats in XML over the entire development process and related standardized AUTOSAR code generation and modeling tools.

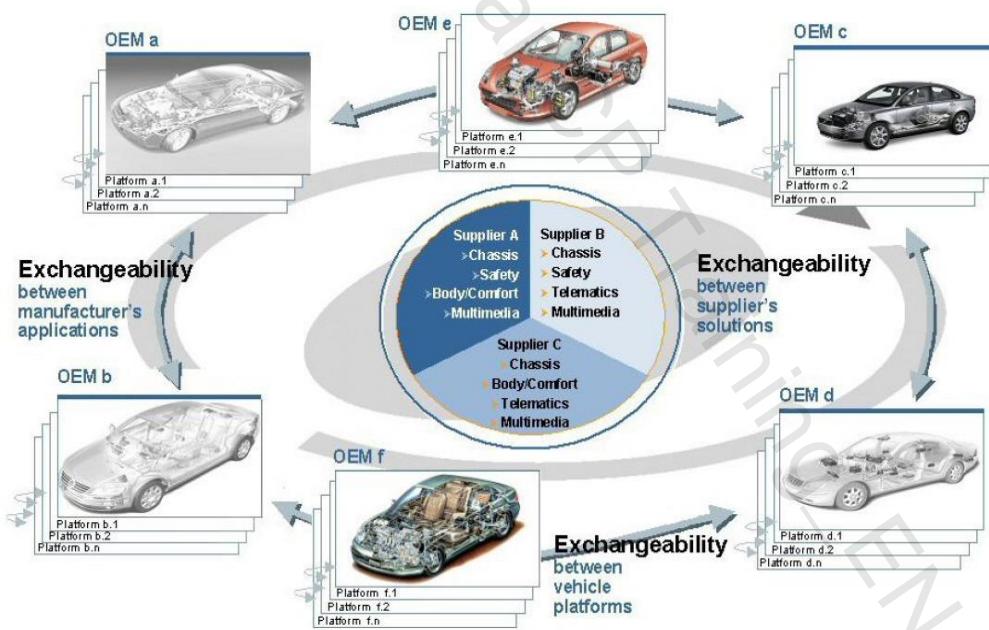
AUTOSAR Objectives

- **Reusability** of functions over different vehicles.



9

- Reusability of functions beyond the vehicle network is a key characteristic of AUTOSAR. One reason this is possible is the uniform and standardized description of functions and the system. A configurable “middleware” is also needed for abstraction of the basic software: The RTE (Runtime Environment). Specific code can then be generated for each ECU.
- Furthermore, functions can be moved across OEM boundaries as long as the standardized AUTOSAR application interfaces are being used.



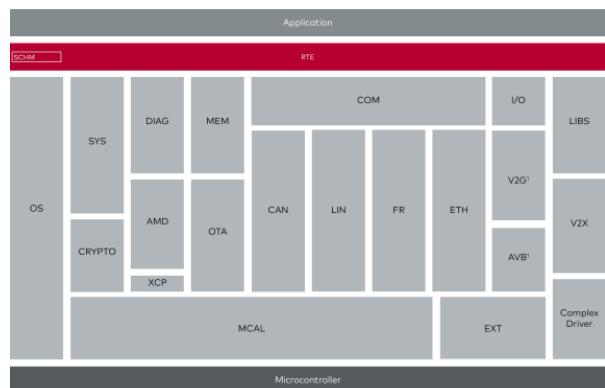
Objectives



AUTOmotive Open System **AR**chitecture

► Standardization

- ▶ of interfaces
- ▶ of exchange formats
- ▶ of methodology



- ▶ Implementation and standardization of fundamental system functions as the cross-OEM “**basic software stack**”.
- ▶ **Integration** of functional modules from different suppliers.
- ▶ **Scalability** to different vehicle and platform variants.

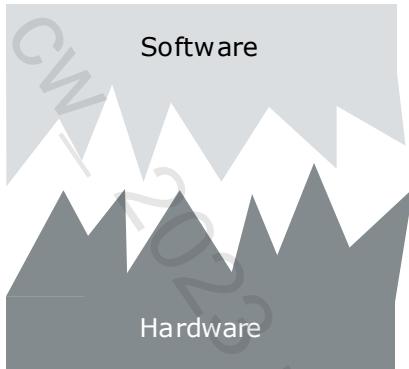
10

- ▶ Standardization of interfaces between the application and basic software enables abstraction of the application from the hardware and smooth integration of functional modules.

AUTOSAR Slogan

The AUTOSAR statement:

"Cooperate on standards – compete on implementation"



Application Software

AUTOSAR

Hardware

Standardized
Methodology

HW-specific
(ECUs)

11

- ▶ AUTOSAR wants to emphasize that many standards are being created with its slogan “Cooperate on standards, compete on implementation”. Since every CAN control module has a CAN driver, it makes little sense for each OEM to have its own implementation, since this does not yield any competitive advantages. Costs are saved by everyone using a standardized CAN driver. In contrast, competition is desired in application-related areas, since special implementations drive innovation and reduce costs here.
- ▶ In the past, the hardware and software of an ECU were often tightly interwoven. ECUs usually had to be redeveloped from scratch whenever there were additional features, new controllers, different OEMs or modified architectures. The approach of the AUTOSAR Initiative is to use modular components. ECU suppliers can, for example, develop application components that can be reused on the projects of different OEMs. In the future, it will be easy for OEMs to port functions over vehicle platforms without necessarily having to choose the same ECU to execute a certain function.

Agenda

Overview and Objectives

► **AUTOSAR Application**

AUTOSAR RTE

AUTOSAR BSW

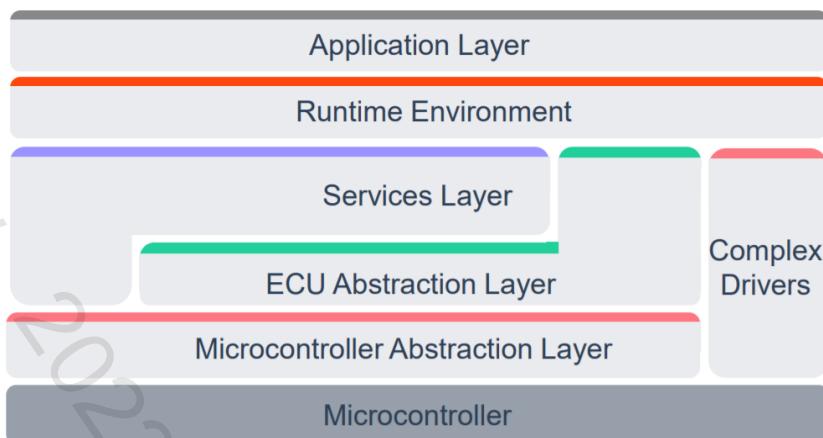
AUTOSAR Methodology

AUTOSAR in Practice

Appendix



Architecture – Overview of Software Layers

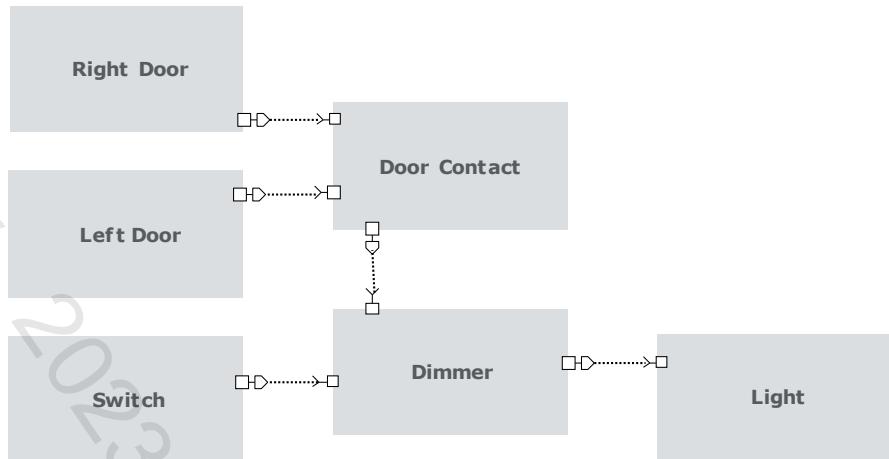


Source: <http://www.autosar.org>

13

- ▶ The AUTOSAR Architecture distinguishes on the highest abstraction level between three software layers: Application, Runtime Environment and Basic Software which run on a Microcontroller.
- ▶ The AUTOSAR Basic Software is further divided in the layers: Services, ECU Abstraction, Microcontroller Abstraction and Complex Drivers.

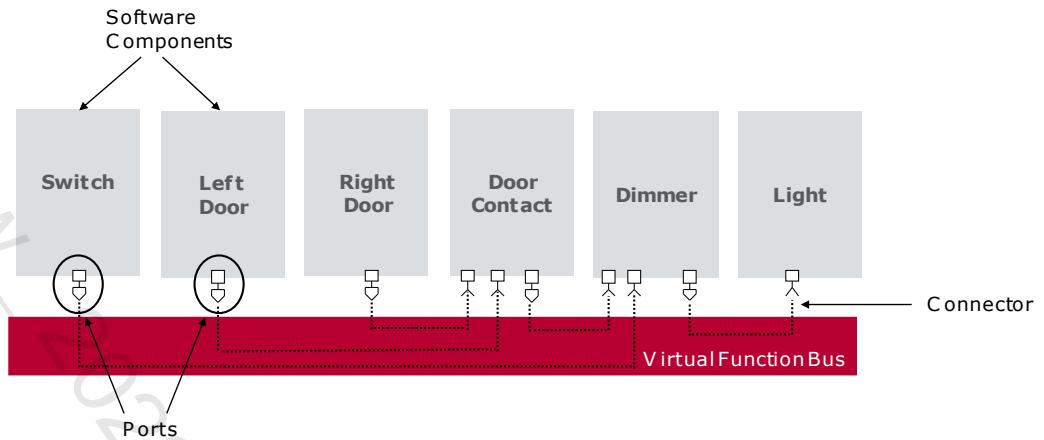
Components View for Lighting Control



14

- ▶ At the beginning, a system is only described by its functionality. Sub-functionalities are distributed to components. These components transfer information to other components utilizing defined interfaces.

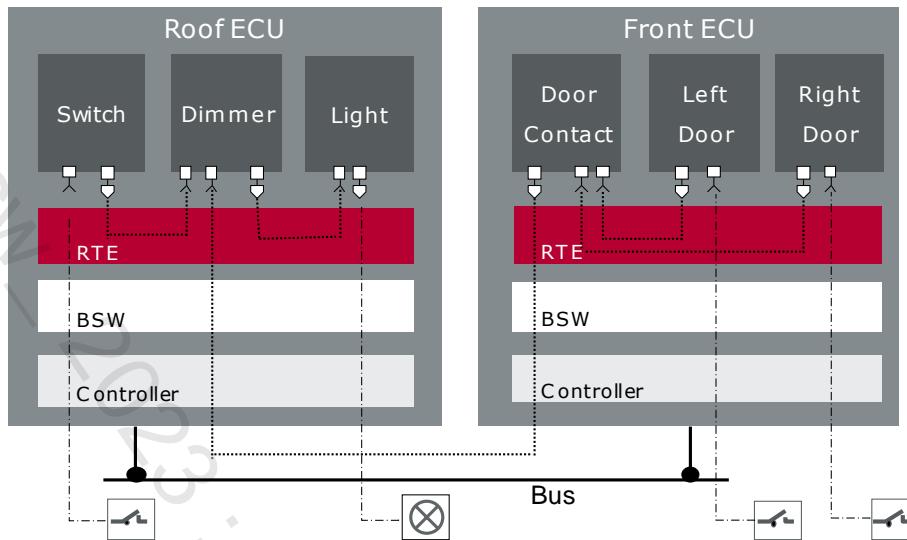
Communication



15

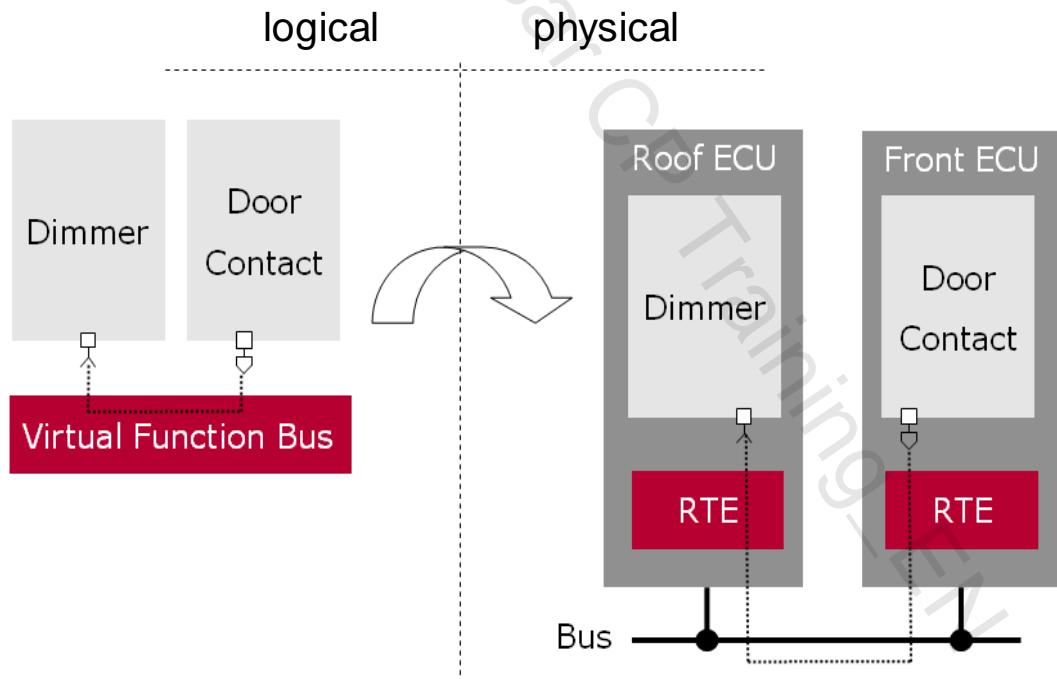
- ▶ The “Virtual Function Bus“ symbolizes the communication between the components. At this point in time, allocations of components to ECUs are still undefined. Only a logical perspective exists. The VFB represents communication within an ECU as well as between ECUs. The application has no knowledge of the underlying technologies. This makes it possible for the application software to be developed and utilized independent of the hardware.

Distribution of Components



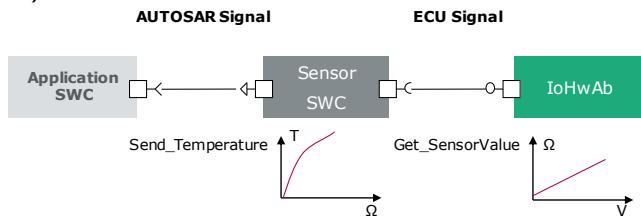
16

- After all components have been chosen and defined, they are distributed to the relevant ECUs (mapping of software components to ECUs).
- The VFB (Virtual Function Bus) is implemented with the help of an ECU-specific RTE (Run Time Environment). The RTE organizes the communication between the individual components and – with the help of the operating system – handles execution of the components. The RTE is scalable and is generated statically for each ECU.

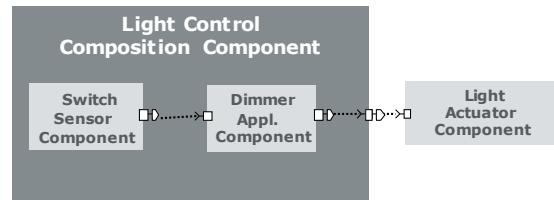


Types of Software Components

- ▶ Atomic component (cannot be further subdivided)
 - ▶ Application
 - > Implementation of the algorithm
 - ▶ Sensor/actuator
 - > Preparation of I/O data for the application
 - > Component is bound to ECU
(constraint for mapping!)



- ▶ Composition
 - ▶ logical organization or encapsulation of SWCs and/or Composition.



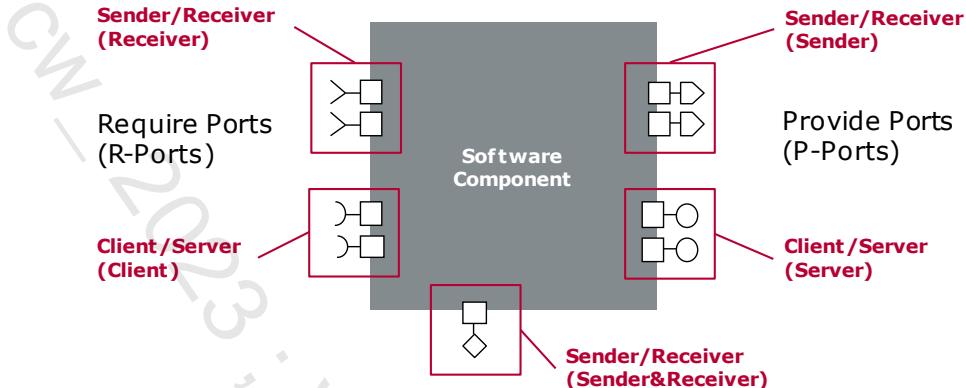
17

- ▶ The AUTOSAR software component (SWC) is an atomic software code fragment and part of the application. SWCs are independent of the infrastructure and can be mapped to any ECU.
- ▶ The sensor/actuator software components are a special type of atomic components and represent dependency on a special sensor or actuator.

Types of Ports

- ▶ Software components have Ports
 - ▶ Interface to other components for communication
 - ▶ Contents: Data elements (S/R) and operations (C/S)

- ▶ AUTOSAR 4: Provide Require Ports (PR-Ports):



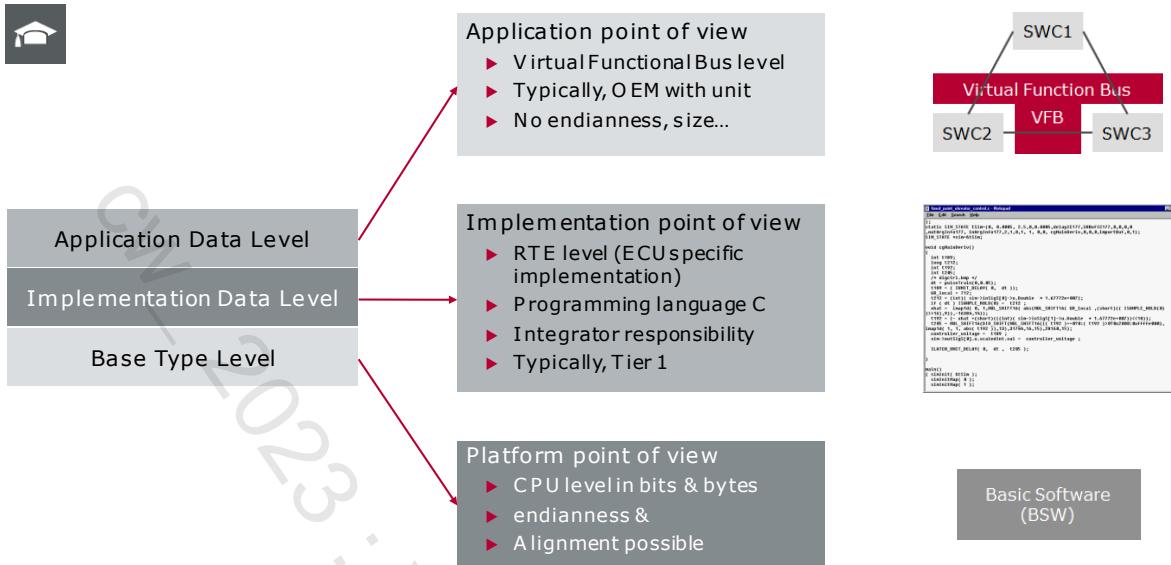
18

- ▶ Ports
 - ▶ Interfaces between software components are called ports.
 - ▶ Ports are either Sender/Receiver ports or Client/Server ports. Sender/Receiver ports contain data elements that are assigned to network signals.
 - ▶ Client/Server ports contain operations with arguments.
 - ▶ Ports are joined by connectors.

- ▶ Sender-Receiver communication
 - ▶ Sender-Receiver communication is used for asynchronous data transmission. This means that a sender provides data to one or more receivers asynchronously. The sender neither expects any data nor does it receive any. The sender does not know the number of receivers. This principle corresponds to the familiar signal communication between ECUs.

- ▶ Client-Server communication
 - ▶ An alternative method for communication between software components is Client-Server communication. The Server offers services which are requested by the Client. An AUTOSAR software component may be implemented as a Client, as a Server or as Client and Server.

AUTOSAR Data Types



19

► Application data types

Application data types define all data attributes that are needed from the application point of view. It is used for exchanging data between software components or between a software component and a measurement and calibration tool. For this level also the communication for a complete virtual function bus can be specified.

► Implementation data types

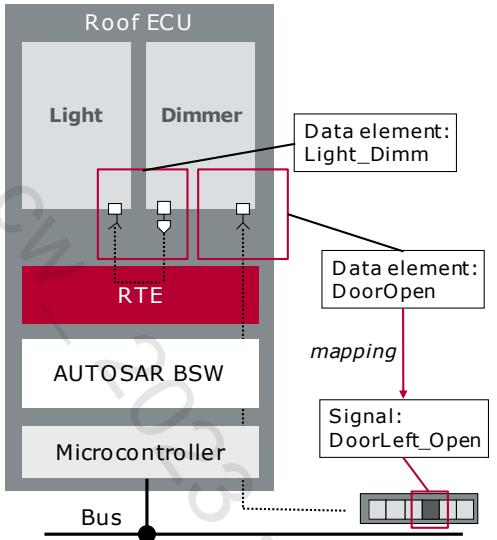
Implementation data types enable a formal definition of types from the implementation point of view. These types conceptually represent the source code of a programming language like C. They are used for interface descriptions and data within the basic software. The data description on this level is also input for the RTE generator.

► Base data types

Base data types describe the primitive elements in terms of bits and bytes. They are also the base for the implementation data. The attributes for base data types can be defined either platform independent or platform specific. The description of base types is required for the generation of the RTE.

(for more information, see AUTOSAR 4.X Software Component Template)

Sender-Receiver



- ▶ Transport of data
- ▶ A port may contain multiple data elements.
- ▶ A data element is “mapped” to a signal if the information goes over the bus.
- ▶ Simple data types (integer, float) and complex types (array, record).
- ▶ Communication: 1:n or n:1

▶ Example receiver call:
`Rte_Read_Door_DoorOpen()`

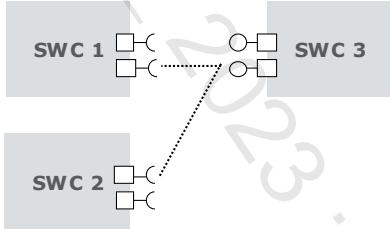
20

- ▶ As a communication interface, AUTOSAR provides “Ports” that may contain multiple data elements. These data (simple data types like float, integer etc. or complex structures) are supported by AUTOSAR.
- ▶ In inter-ECU communication the data associated with a port must be mapped to a network signal.

Client-Server

- ▶ Serving of operations
- ▶ Communication: 1:1 or n:1
- ▶ Synchronous or asynchronous
- ▶ Interface with multiple operations.
- ▶ Operations can be called individually.
- ▶ Inter and Intra ECU communication possible.

- ▶ Example call from Client:

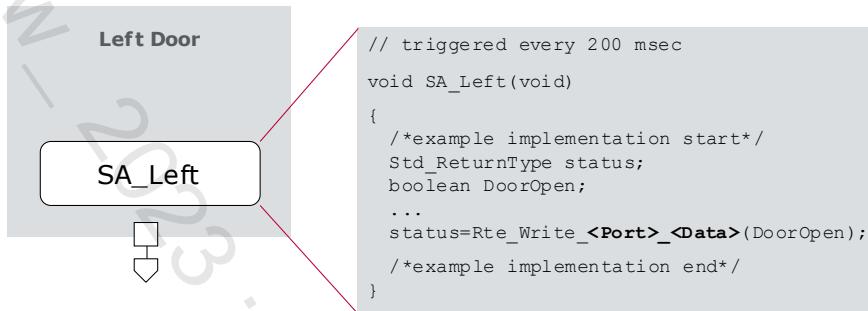


21

- ▶ A distinction is made between synchronous and asynchronous calls.
- ▶ Synchronous: The Client is blocked throughout the duration of the call and retains the result in the returning call. Sequential program flow.
- ▶ Asynchronous: The Client does not block during the call and can execute other operations, while the Server executes the requested operation in parallel.
- ▶ Inter ECU communication: RPCs “Remote Procedure Calls” / RMI “Remote Method Invocation”

Runnables

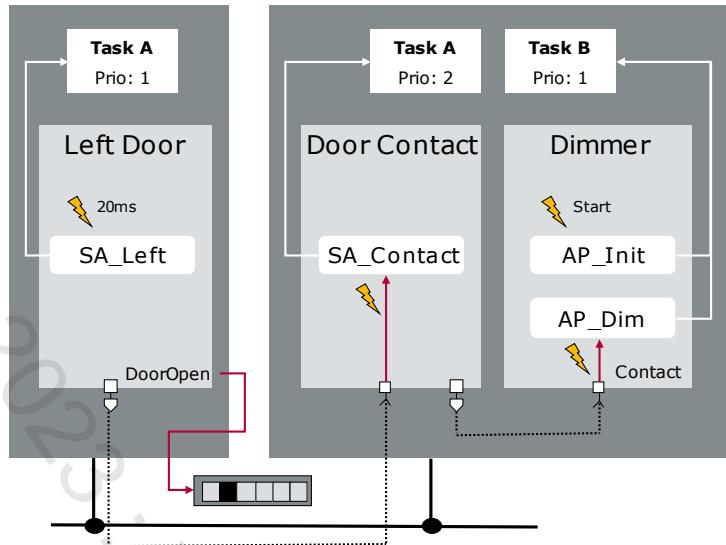
- ▶ Atomic Software components contain
 - ▶ Runnable entities (Runnables) as part of their internal behavior
 - > Function containing the actual implementation.
 - > Function is called by RTE if the corresponding trigger occurs
 - > e.g. when data is received a timer expires or by an operation call



22

- ▶ A runnable entity is the executable unit of a software component
 - ▶ Corresponds to a C-function
 - > void Runnable (void)
- ▶ Software components contain one or more runnable entities (“runnables”).
- ▶ Runnables might be executed when RTE Events occur (e.g. data are received or a periodic activation is due).

Quick Reference to AUTOSAR



23

- ▶ Quick reference to AUTOSAR
 - ▶ SWC, the smallest logical unit of the application
 - ▶ Ports (precisely: Port Prototypes), the communication interfaces to other SWCs
 - ▶ Data elements, the contents of the ports (only for Sender/Receiver Communication)
 - ▶ Runnables, the executable code units of the application
 - ▶ SWC mapping, assignment of SWCs to ECUs
 - ▶ Data mapping, assignment of data elements of the application to bus signals (from the communication matrix)
 - ▶ Task mapping, assignment of runnables to tasks of the operating system
 - ▶ Target platform, configuration of the basic software for the target platform
 - ▶ Code generation, generation of executable codes for ECU

Exercise

Exercise 1:

Outline the functionality of "interior light" utilizing the AUTOSAR modeling objects in a software design. Only use software components that communicate via ports (Sender/Receiver).

- ▶ Model a separate software component for each sensor (Door + Switch) and actuator (Light).
- ▶ An application software component should interpret the status of doors and pass it to the actuator components.
- ▶ Implement a shutoff delay and dimming behavior in another application software component.

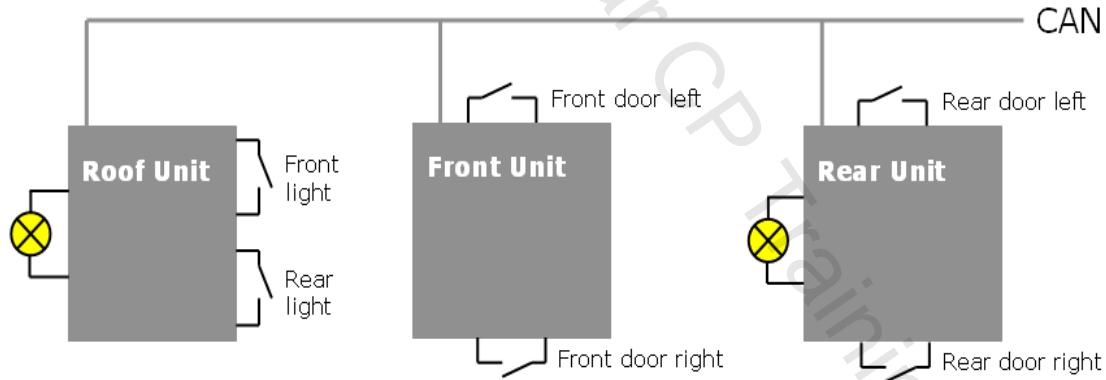
Exercise 2:

There are three ECUs (rear, front and roof) which shall realize the functionality of "interior light":

- ▶ Map each SWCs from Exercise 1 to one of the ECUs.
- ▶ Which information is exchanged between the ECUs?

24

- ▶ Functional description of interior lighting control:
 - ▶ Two separate interior lights for the front and rear seats
 - ▶ If at least one of the four doors is opened, both interior lights are activated
 - ▶ After the last door has been closed both interior lights are deactivated

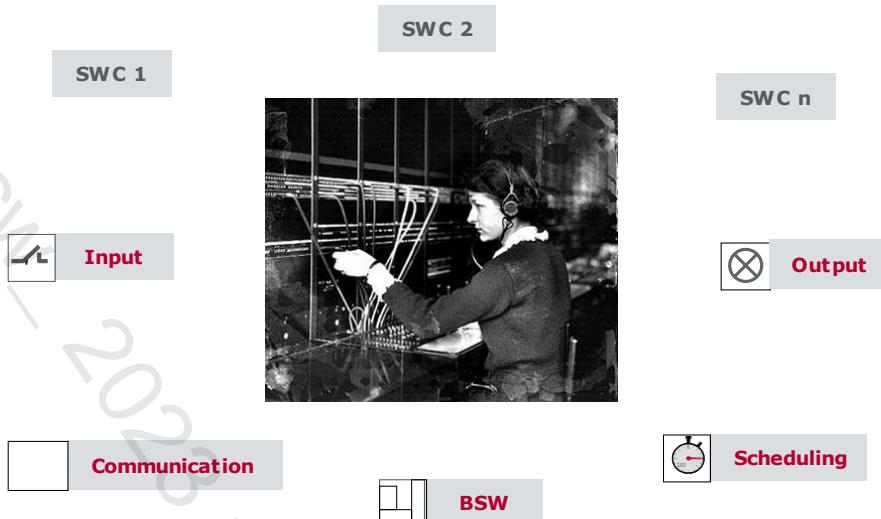


Agenda

- Overview and Objectives
- AUTOSAR Application
- ▶ **AUTOSAR RTE**
- AUTOSAR BSW
- AUTOSAR Methodology
- AUTOSAR in Practice
- Appendix

2023_VH_Autosar CP Training_EN

RTE in the Role of a Switchboard Operator

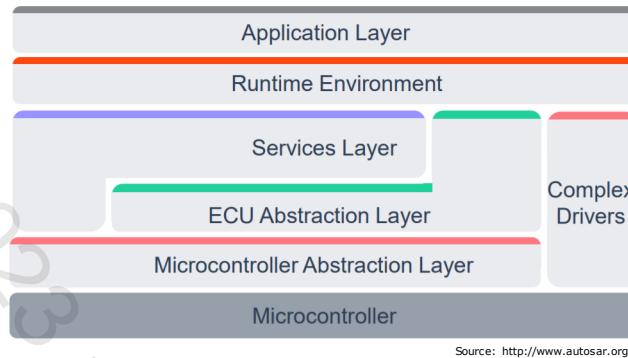


26

- ▶ The RTE (Run Time Environment) is the implementation of the VFB (Virtual Function Bus) and acts as a switchboard between the application, basic software and hardware.

Architecture

- ▶ Task
 - ▶ Components independent of ECU mapping.
- ▶ Functionality
 - ▶ Middleware providing communication services (intra / inter ECU).

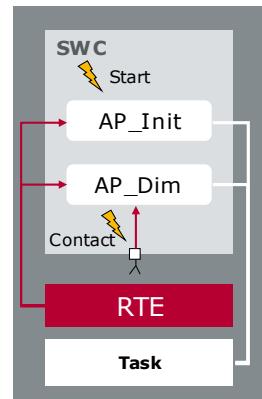


27

- ▶ The Runtime Environment (RTE) as middleware integrates individual applications with the basic software. It organizes the communication and data exchange between the two and handles execution of software functions (runnables). The intermediate RTE is needed to develop application software that is independent of the hardware to be used later. Since all interfaces lying between the described layers are defined precisely, the application software no longer needs to know how the layers below it operate. Communication involves sending only certain parameters to the next layer by passing them to the defined interfaces (described in the model by ports). In the functional description in the model, it does not matter whether the port represents a connection within an ECU or whether a connection with another ECU is necessary. If the design process requires distribution of software components to separate ECUs, then this is not defined until a later point in time.

RTE as Runtime Environment for Runnables

- ▶ RTE is configured (e.g. mapping Runnables to OS tasks).
- ▶ Triggering of Runnables via RTE events.
- ▶ Includes executable code units for the tasks with calls of the Runnables.
- ▶ Configures parts of the OS (tasks, events, alarms).
- ▶ Implements communication between components.
- ▶ RTE is different for each ECU and is customized to requirements of the SWC.
- ▶ RTE abstracts the OS and prevents SWC from directly accessing OS and BSW.
- ▶ RTE does not have any runtime component.



28

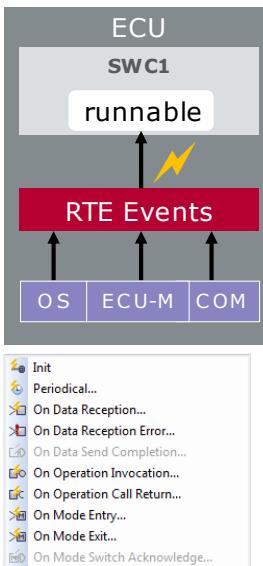
- ▶ The RTE as a runtime environment enables applications that are platform-independent (i.e. independent of the hardware and basic software). The RTE is regenerated for each ECU after software components have been mapped to the specific ECU. A new mapping or change to the interfaces of the SW components makes it absolutely necessary to regenerate the RTE.

RTE as Runtime Environment for Runnables

- ▶ Triggering of Runnables, wake-up from waitpoints
 - ▶ Init
 - ▶ TimingEvent
 - ▶ DataReceivedEvent (S/R)
 - ▶ DataReceiveErrorEvent (S/R)
 - ▶ DataSendCompletedEvent (S/R)
 - ▶ OperationInvokedEvent (C/S)
 - ▶ AsynchronousServerCallReturnsEvent (C/S)

- ▶ ModeSwitchEvent
- ▶ ModeSwitchAckEvent

- ▶ ExternalTriggerOccurredEvent
- ▶ InternalTriggerOccurredEvent



29

Syntax:

- ▶ Runnable triggered by any event except C/S communication events
 - ▶ `void <RunnableName>([IN Rte_Instance instance])`

- ▶ Runnable triggered by AsynchronousServerCallReturnsEvent or OperationInvokedEvent (C/S)
 - ▶ `{void|Std_ReturnType} <Runnable>([IN Rte_Instance inst]{,paramlist}*)`

Examples:

- ▶ Runnable triggered by any event except communication events
 - ▶ `void runnable (Rte_Instance self)`

- ▶ Server runnable triggered by OperationInvokedEvent, with 2 IN parameters and 1 OUT parameter
 - ▶ `void LightServer (Rte_Instance self, UInt8 keyValue,
Boolean active, ResultTypeRef result)`

- ▶ All types of RTE events can be used to either:
 - ▶ Activate a runnable;
 - ▶ Cause a wakeup at a specific waitpoint (e.g. Client waits for response of Server)

Since AUTOSAR 4.x:

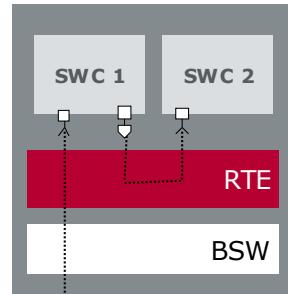
ExternalTriggerOccurredEvent - trigger from a different SWC (inter SWC triggering)

InternalTriggerOccurredEvent - inter runnable triggering

BackgroundEvent – low priority triggering if no other trigger released

RTE as Communication Interface

- ▶ RTE as **communication mechanism** among SWCs and to the BSW.
 - ▶ RTE acts as implementation of the VFB
 - ▶ Sender-Receiver
 - ▶ Client-Server
 - ▶ Intra-ECU and Inter-ECU (via COM)
 - ▶ RTE implements callbacks of AR-COM

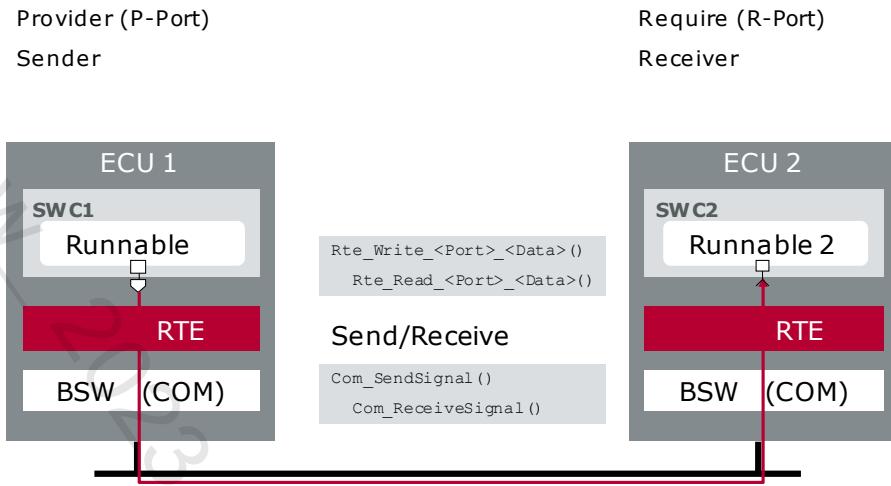


- ▶ Other features:
 - ▶ Mechanisms for data consistency (inter SWC, intra SWC)
 - ▶ Support of primitive and complex data (records)
 - ▶ Multiple instantiation of SWC types

30

- ▶ Besides the runtime environment of the runnables, another primary task is communication of the software components (SWC) among one another (inter-ECU and intra-ECU communication) and between the software components and the basic software.

Sender/Receiver Communication: Inter-ECU



31

- ▶ Example of a runnable with writing of a data element:

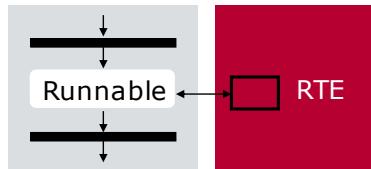
```
// triggered every 20 msec
void AP_Dimmer()
{
    Std_ReturnType status;
    boolean Door_IsOpen;
    static uint8 dimmer = 0;

    status = Rte_Read_AnyDoor_Open_DoorOpen(&Door_IsOpen);
    if (TRUE == Door_IsOpen) {
        if (DIMMER_MAX > dimmer) {
            dimmer++;
        }
    }
    status = Rte_Write_DimmLight_DimmLight(dimmer);
}
```

Sender/Receiver Communication: Direct

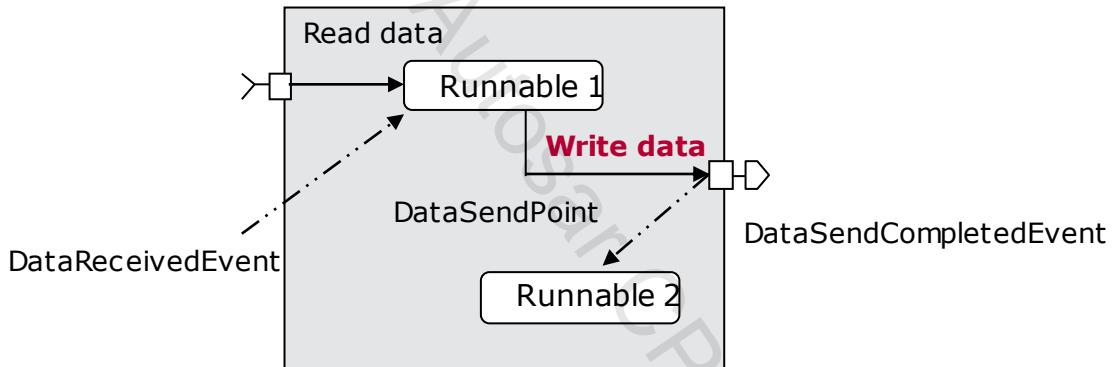
- ▶ “**last-is-best semantic**” (isQueued=false)
 - ▶ RTE uses “direct access” for data buffer
 - ▶ 1:n communication
 - ▶ Init values as default

```
Std_ReturnType Rte_Read_<p>_<d> (OUT <DataType> *data)
Std_ReturnType Rte_Write_<p>_<d> (IN <DataType> data)
```



32

- ▶ Data is written immediately at “DataSendPoint” during execution of the runnable.

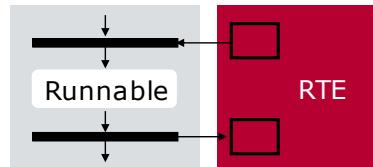


- ▶ In AUTOSAR 4.x: The Read data can also be obtained in the return value for explicit access:
 - ▶ <return> Rte_DRead_<p>_<o> ([IN Rte_Instance <instance>])

Sender/Receiver Communication: Buffered

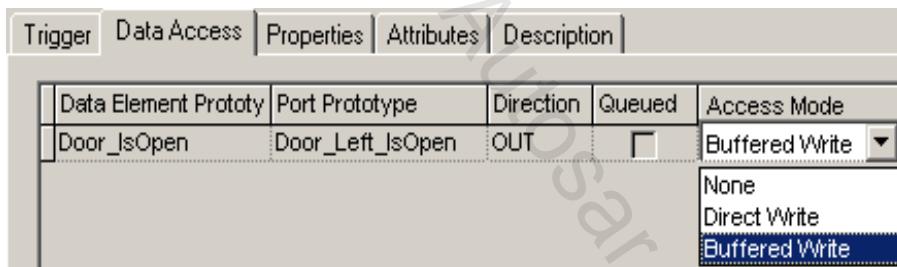
- ▶ “**last-is-best semantic**” (isQueued=false)
 - ▶ RTE generates copy before executing the Runnable
 - ▶ After execution of the Runnable the RTE copies the data to a global buffer.
 - ▶ Values are not changed during execution

```
<DataType> Rte_IRead_<re>_<p>_<d> (void)
void Rte_IWrite_<re>_<p>_<d> (IN <DataType> data)
```

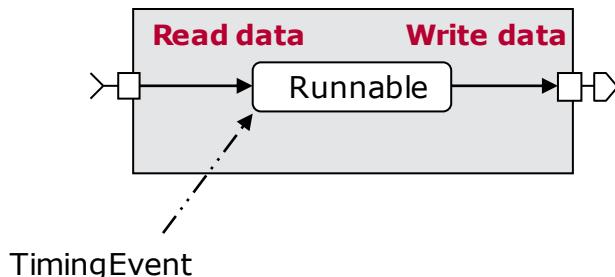


33

- ▶ DaVinci option for “last is best semantic” and “buffered communication”.



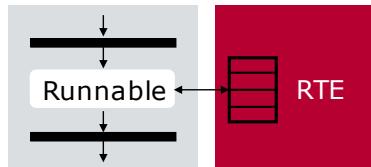
- ▶ Writing of data does not occur until the runnable has ended



Sender/Receiver Communication: Queued

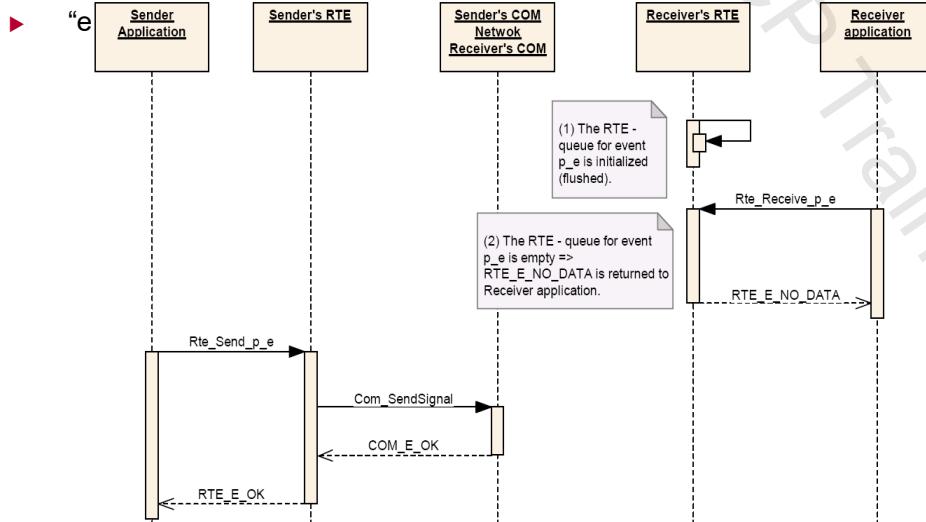
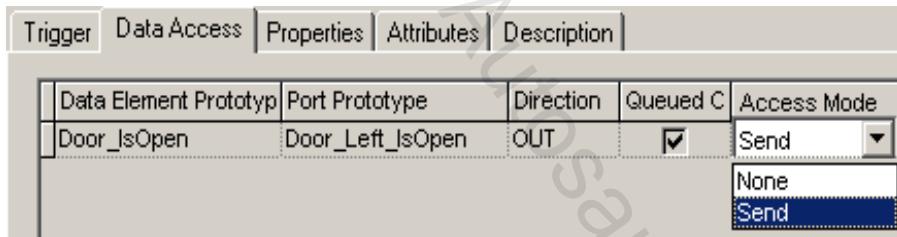
- ▶ “event semantic” (isQueued=true)
- ▶ “Polling receive” or “waiting receive”
- ▶ RTE reads from a specific receive queue
- ▶ “Waiting receive” with timeout handling

```
Std_ReturnType Rte_Receive_<p>_<d> (OUT <DataType>* data)
Std_ReturnType Rte_Send_<p>_<d> (IN <DataType> data)
```



34

- ▶ DaVinci option for “event semantic” and “queued communication”



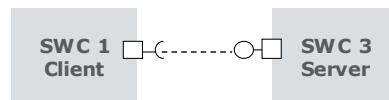
Sender/Receiver Communication

- ▶ „**data element invalidation**“
 - ▶ Use case: Sensor delivers invalid values
 - ▶ Only for „isQueued=false“ communication
 - ▶ realized by „invalid value“
 - ▶ Set invalid values on the sender's side
 - > `Rte_Invalidate_<p>_<d>()`
 - ▶ Reaction on invalidated data elements on the receiver's side
 - > Evaluate the `Std_ReturnType`. Result would be `RTE_E_INVALID`
 - > direct communication: Return value of `Rte_Read_<p>_<d>()`
 - > buffered communication: `Rte_IStatus_<re>_<p>_<d>()`
 - > Activate Runnable via
 - > `DataReceiveErrorEvent`

Client/Server Communication

- ▶ Communication
 - ▶ n:1
- ▶ Server call
 - ▶ Client calls Server operation
 - ▶ Operation is implemented as Runnable of the Server-SWC
 - ▶ Synchronous / asynchronous calls
 - ▶ Server Runnables run in the
 - > Task context
 - > Context of the Client (direct function call)

```
Std_ReturnType Rte_Call_<p>_<o>([IN | IN/OUT | OUT <param_1>], ...
[IN | IN/OUT | OUT <param_n>])
```



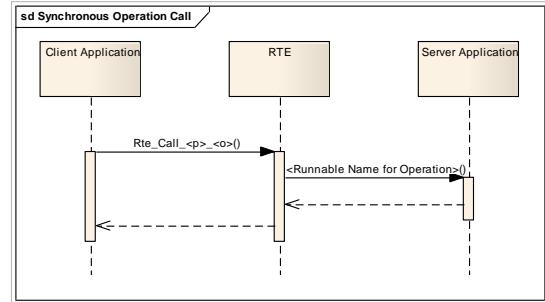
36

- ▶ Client/Server communication that extends beyond ECUs makes a new type of protocol necessary. In contrast to this, a simple means of communication, e.g. by CAN, suffices for Sender/Receiver communication. Here a mapping is made of a port's data elements to a network signal. In Client/Server communication the mapping of the operation is to the protocol.
- ▶ If the Server runnable is running in the context of the Client, then it is not assigned to any task, but is called directly instead (direct function call).
- ▶ If the Server runnable is running in the task context, then a task switch is made when the Client runnable is assigned to another task. In this case, the RTE handles the saving of data.

Client/Server Communication

► Synchronous

- ▶ Wait for response of Server (Client is blocked)
- ▶ Results of Server via "OUT" parameter of
Rte_Call_<p>_<o>



► Server Runnable

```
Std_ReturnType GetTime(uint32 *hour, uint32 *minute, *uint32 second)
```

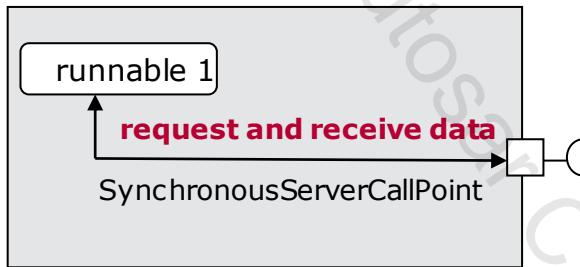
► RTE Client API

```
Std_ReturnType Rte_Call_<Port>_GetTime(uint32 *hour,
                                         uint32 *minute, uint32 *second)
```

37

► Example of synchronous Clients:

- ▶ Runnable 1 as Client sends its data synchronously. That is, it waits for the response of the Server.



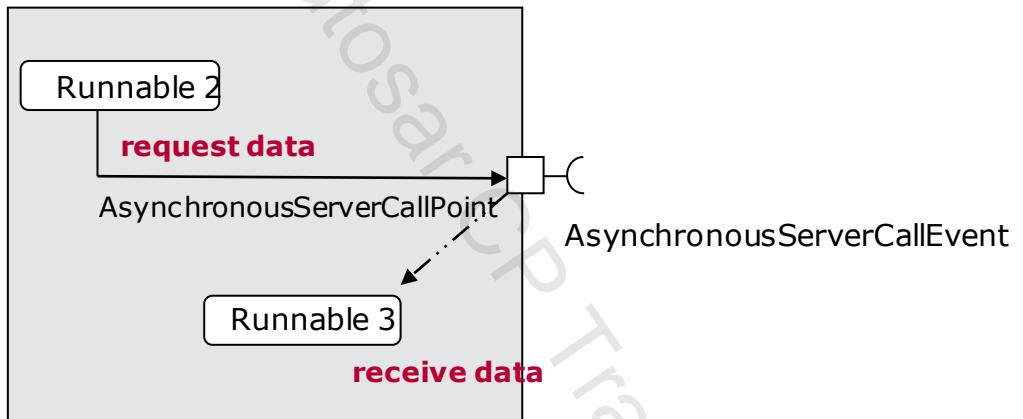
Client/Server Communication

- ▶ Asynchronous
 - ▶ Client is not blocked (does not wait for results)
 - ▶ Client gets Server results by call of Rte_Result...
 - > Polling or waiting
 - > Timeout handling
- ▶ RTE Client API


```
Std_ReturnType Rte_Result_<p>_<o>([ IN/OUT | OUT <param_1>],...  
[IN/OUT | OUT <param_n>])
```
- ▶ Alternative:
 - > RTE activates Client Runnables as soon as results are available

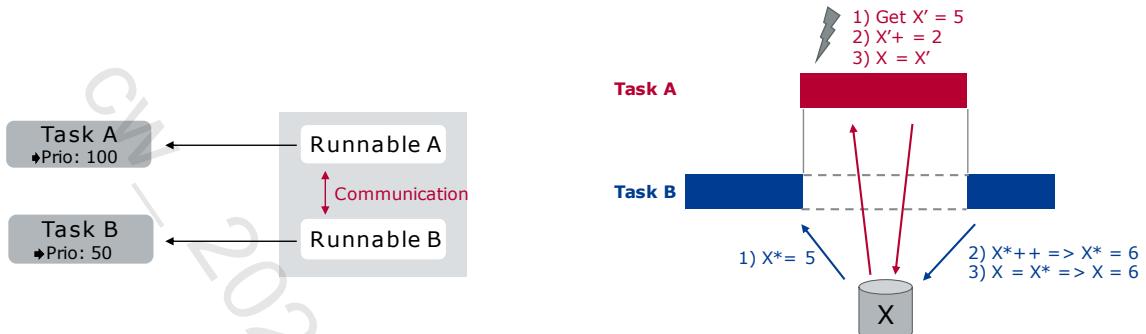
38

- ▶ Example of asynchronous Clients:
 - ▶ Runnable 2 as Client sends its data asynchronously. Receiving of the data can, for example, be performed by runnable 3 via AsynchronousServerCallEvent.



Intra-SWC Communication

- ▶ **Problem:** Communication between Runnables within **one** SWC, which runs potentially on **different tasks**.



- ▶ **Goal:** Data consistency (atomic access on task level).

39

- ▶ Inter Runnable Variables (IRVs)
 - ▶ Variable that allows the communication between two runnables of the same component
 - ▶ Interrupt-safe access API provided from the RTE
- ▶ Exclusive Areas (EAs)
 - ▶ Allows to specify atomic sections for your implementation
 - ▶ Configurable implementation: e.g. interrupt lock, OS resource..
- ▶ Per Instance Memory (PIM)
 - ▶ For multiply instantiated Components; “a multi instantiation compatible version of a static variable”.
 - ▶ In combination with EAs, you obtain similar functionality like with the IRVs while obtaining the possibility to freely define the used data type. (typedef)

Intra-SWC Communication Solution

Solution:

Exclusive Areas (EAs)

Enclosed statements are atomic

```
Rte_Enter_<name>();  
/* protected statements  
 */  
Rte_Exit_<name>();
```

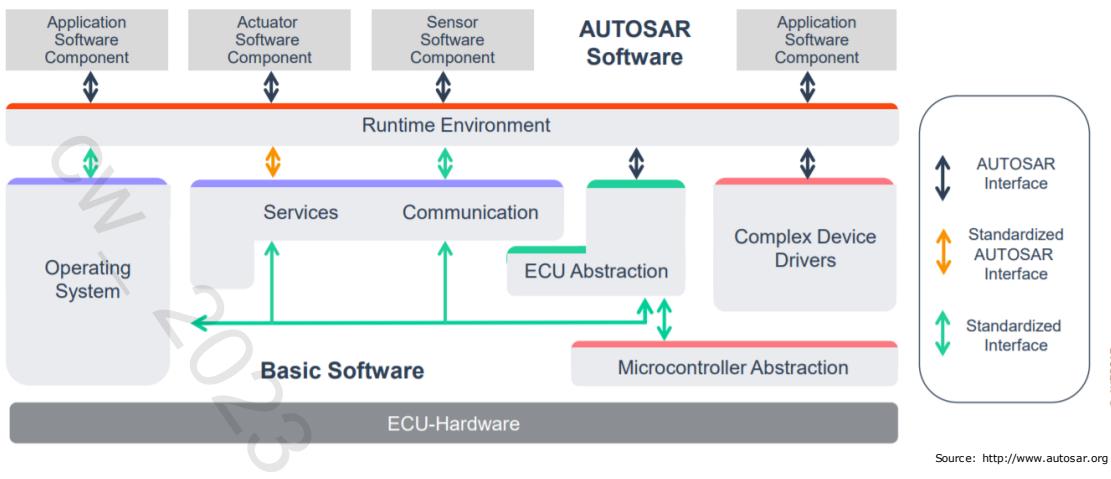
Note: There are no problems in intra- and inter-communication of *different* SWCs, since they are handled by the RTE.

Inter-Runnable variables (IRVs)

Only access to variable is atomic

```
Rte_IrvWrite_<re>_<name>  
or Rte_IrvRead_<re>_<name>
```

Interfaces

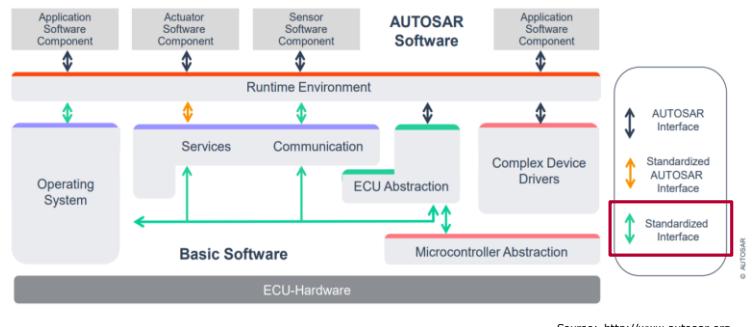


41

- ▶ Three AUTOSAR interface types are described in the diagram above:
 - ▶ AUTOSAR interface
 - ▶ Standardized interface
 - ▶ Standardized AUTOSAR interface
- ▶ Furthermore, standardized basic software components are described as well as ECU-specific basic software components:
 - ▶ Standardized basic software components
 - > Services
 - > Diagnostics, NVRAM, flash, memory management
 - > Communication
 - > Communication framework (CAN, LIN, FlexRay), I/O management, network management
 - > Operating system
 - > Based on OSEK OS (ISO 17356-3)
 - > Microcontroller abstraction
 - > Digital I/O, analog/digital converter, EEPROM, etc.
 - ▶ ECU-specific basic software
 - > ECU abstraction
 - > Complex device driver
 - > For resource-critical applications

Standardized Interface

- ▶ Simple C-API
- ▶ OS and COM are only available indirectly via the RTE
- ▶ Has no ports
- ▶ OS: RTE implements the tasks and uses the OS API
 - ▶ Schedule(), WaitEvent()
- ▶ COM: RTE implements the callbacks of COM and uses their API
 - ▶ Com_SendSignal()
- ▶ EcuM: Rte_Start(), Rte_Stop()

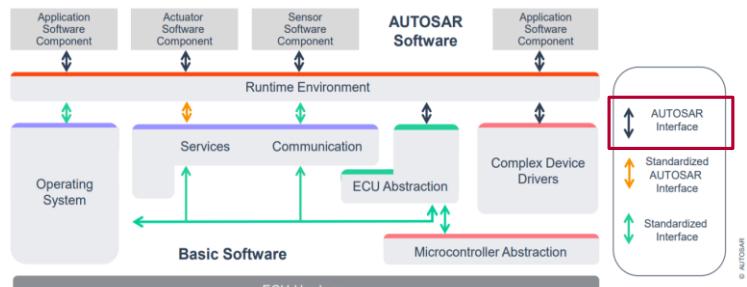


42

- ▶ Standardized interfaces are interfaces whose semantics are prescribed by the AUTOSAR standard and are implemented as direct C-APIs for optimization reasons.

AUTOSAR Interface

- ▶ I/O interfaces of hardware are described as in SWC
 - ▶ Ports
 - ▶ Runnables
- ▶ ECU Abstraction = "Firmware"
 - ▶ Since access here is exclusively via sensor/actuator SWC
- ▶ API function:
 - ▶ S/R -> Rte_Write_<p>_<d>
 - > Rte_Write_DimmLight_DimmValue (value)
 - ▶ C/S -> Rte_Call_<p>_<o>

Source: <http://www.autosar.org>

43

- ▶ AUTOSAR interfaces are interfaces that are described by generic AUTOSAR means and whose semantics are not prescribed by the AUTOSAR standard. These interfaces are used by the software components and ECU Abstraction. Ports are used for communication.

Standardized AUTOSAR Interface

- Are standardized and not ECU specific (e.g. ECU Abstraction)

- RTE -> BSW: Here one speaks of "Service Ports"

- API function
 - Communication mode for special SWC

- SWC → RTE

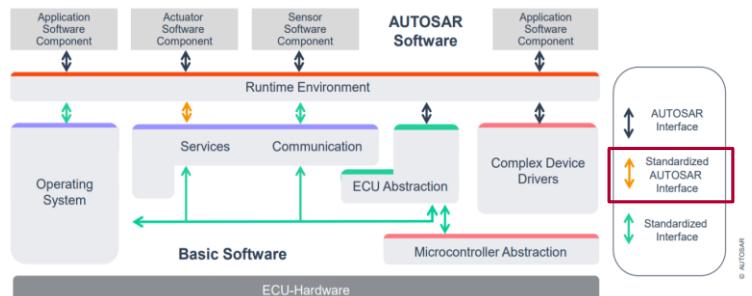
```
Rte_Call_<portPrototype>_<operation>()
```

Source: <http://www.autosar.org>

- <Operation> DEM

```
SetEventStatus (In Dem_EventStatusType  
status)
```

44



- Standardized AUTOSAR interfaces are interfaces that are described by generic AUTOSAR means and whose semantics are prescribed by the AUTOSAR standard. Standardized AUTOSAR interfaces are used exclusively for the services of the basic software.

Agenda

Overview and Objectives

AUTOSAR Application

AUTOSAR RTE

► **AUTOSAR BSW**

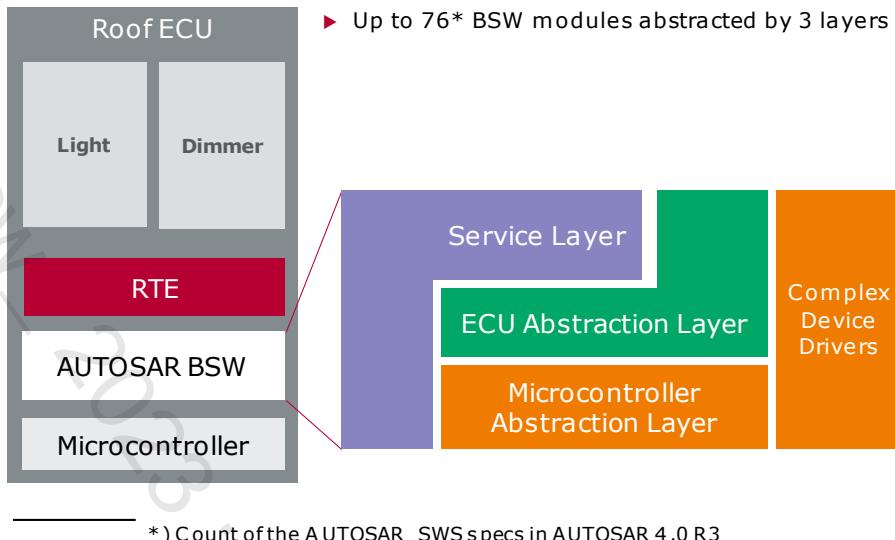
AUTOSAR Methodology

AUTOSAR in Practice

Appendix



Layered View: Simplified

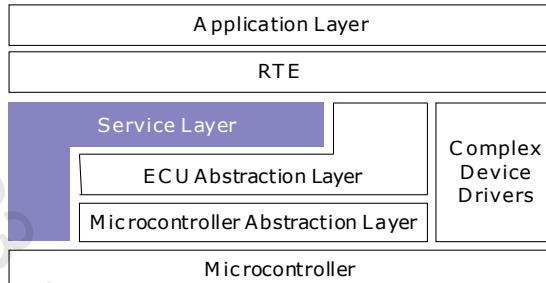


46

- ▶ In AUTOSAR all components of an ECU were abstracted and subdivided into application software, basic software and the hardware layer. The lowermost layer is the hardware layer. Abstracted in it are all hardware features of the microcontroller. Missing hardware features are compensated by suitable software modules at this point. This layer is called the MCAL (Microcontroller Abstraction Layer), and it contains the microcontroller driver, memory driver, communication and I/O drivers.
- ▶ Above the MCAL is the abstraction layer of the ECU. This layer abstracts all base components located in the ECU. The drivers for external peripheral components are located here as well.
- ▶ The next higher layer, the Services Layer, is largely independent of the hardware and provides various types of background services such as memory management, network and bus communication services.
- ▶ Special importance is given to the RTE, as the fourth layer: It connects the components of the application to the basic software by regulating data exchanges and interactions between them.

Service Layer

- ▶ Task
 - ▶ Services for application
- ▶ Functionality
 - ▶ Diagnostics, NVRAM Management, Watchdog Manager, Communication, OS, Schedule Manager, ECU state management, Com Channel Management

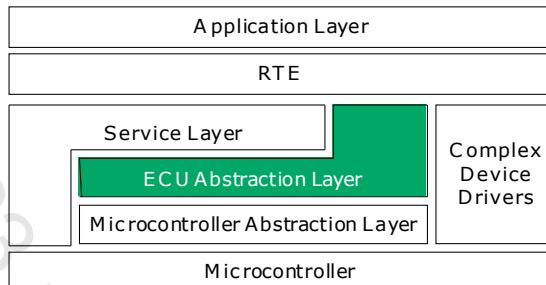


47

- ▶ The Service Layer is the uppermost layer of the basic software. While I/O hardware access is organized by the ECU Abstraction Layer, the Service Layer offers the following services:
 - ▶ Operating system services
 - ▶ Vehicle network communication and management services
 - ▶ Memory services (NVRAM management)
 - ▶ Diagnostic Services (including UDS communication and error memory)
 - ▶ ECU state management
- ▶ The task of the Service Layer is to provide basic services to the application and basic software modules.

ECU Abstraction Layer

- ▶ Task
 - ▶ Make higher levels independent of ECU hardware
- ▶ Functionality
 - ▶ Driver for external devices
 - ▶ Interfaces for internal and external periphery (I/O, Memory, Watchdog(s) and communication)

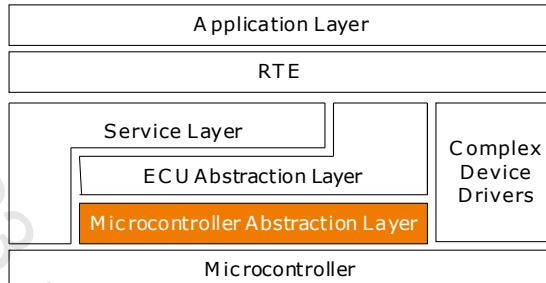


48

- ▶ The ECU Abstraction Layer provides an API (programming interface) for devices, which is independent of their location (μ C internal/external).
- ▶ The task of the ECU Abstraction Layer is to make higher layers independent of the ECU hardware layout.

Microcontroller Abstraction Layer

- ▶ Task
 - ▶ Make higher layers independent of microcontroller.
- ▶ Functionality
 - ▶ Drivers with direct access to internal periphery of µC.
 - ▶ Memory-mapped devices external to µC.

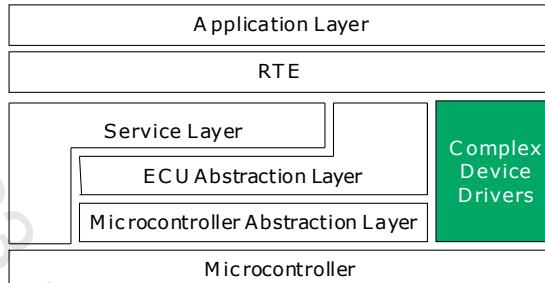


49

- ▶ The Microcontroller Abstraction Layer (MCAL) is the lowermost software layer of the basic software. This layer contains drivers for direct access to the microcontrollers, internal peripherals and memory mapped microcontrollers of external devices.
- ▶ The task of the Microcontroller Abstraction Layer is to make higher layers independent of the microcontroller.

Complex Device Drivers

- ▶ Task
 - ▶ Offer functionality for complex sensors and actuators.
- ▶ Functionality
 - ▶ Direct access to resources for critical applications.
 - ▶ Examples: Injection control, battery management system.



50

The so-called Complex Device Drivers represent a special case in the Layered Software Architecture of AUTOSAR. They control special sensors and actuators by direct microcontroller access, e.g. those sensors and actuators subject to special timing conditions or the integration of non-AUTOSAR technologies. One key concept to follow strict timing requirements is the possible usage of interrupts. (which is not allowed above the RTE layer). Complex Drivers may have interfaces to modules of the BSW stack or to SWCs.

1. Make existing BSW modules use the CDD:

- ▶ AUTOSAR BSW module must be able to configure its interface for the usage of the APIs of the CDD.
Example: PduR: A CDD may implement a new communication system like Bluetooth or USB connections. The PduR configuration can take this into account by calling e.g. CDDIf_Transmit() etc.

2. Make the CDD use existing BSW modules: If the BSW module offers the interfaces, which are suitable for the usage by a CDD, consider to verify:

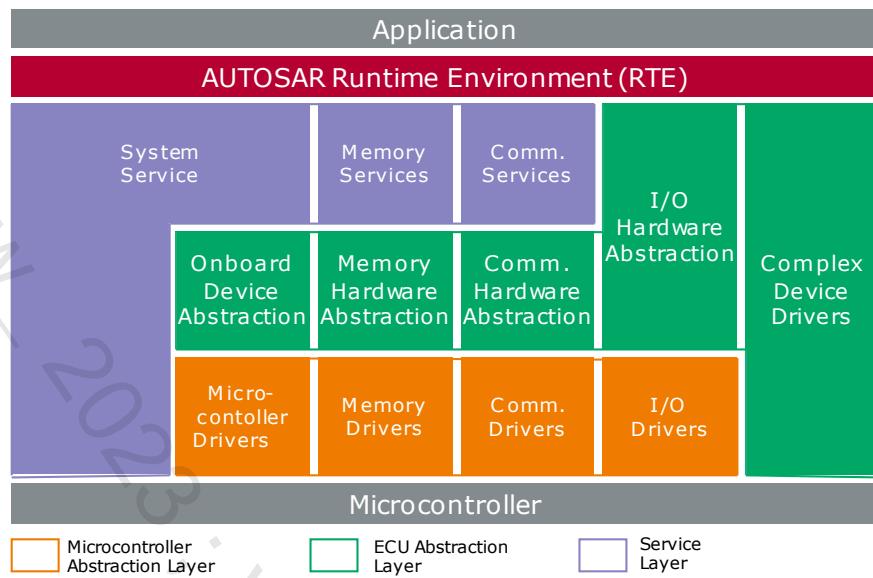
- ▶ No concurrent access to the BSW by several users which modify the BSW module FSM granted?
- ▶ re-entrancy of the offered interfaces given?
- ▶ Configurable Callbacks available?

In general, it is possible to access the following modules:

- ▶ Service layer/Ecu Abstraction Layer modules: OS (used OS objects are not used by a module of the layered software architecture), NVRAM, WDGM, PDUR, BusIf, NM Interface, ComM Upper layer API, BSWM, DET, DEM, DLT
- ▶ MCAL modules: SPI, GPT, I/O drivers (Take care of the lack of re-entrancy or parallel access for the same channels e.g. for ADC, DIO, etc. Re-entrancy only for disjunctive channels.)

It is necessary to check if the respective function is marked as being re-entrant. Example: “_Init()” functions are usually not re-entrant and should only be called once and by the ECU State Manager.

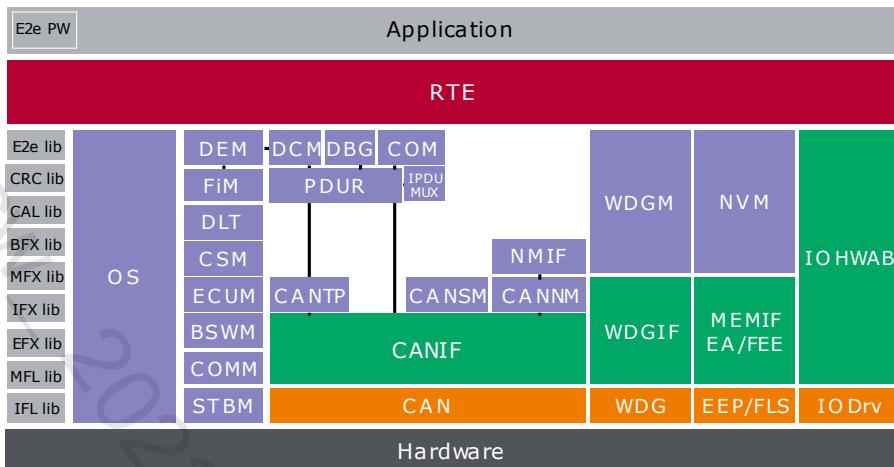
Layered View: Detailed



51

- ▶ Overall, the AUTOSAR basic software consists of approx. 50 modules. They can be subdivided into 3 logical layers:
 - ▶ **AUTOSAR Service Layer**
 - > Task:
 - > Provide services to application
 - > Functionality:
 - > Diagnostics, NVRAM Management, Watchdog Manager,
 - > Communication, OS, Schedule Manager,
 - > ECU state management, Com Channel Management
 - ▶ **AUTOSAR ECU Abstraction Layer**
 - > Task:
 - > Make higher layers independent of ECU hardware
 - > Functionality:
 - > Drivers for external devices
 - > Interfaces for internal I/O and periphery
 - ▶ **AUTOSAR Microcontroller Abstraction Layer**
 - > Task:
 - > Make higher layers independent of microcontroller
 - > Functionality:
 - > Drivers with direct access to µC-internal periphery
 - > Memory mapped µC external devices

AUTOSAR Software Architecture for CAN ECUs



- ▶ High level abstraction
- ▶ Many basic software modules

52

- ▶ The AUTOSAR basic software architecture is organized horizontally in different clusters (areas of function of the basic software) and organized vertically in different layers of abstraction.
- ▶ AUTOSAR defines a new architecture that accepts proven elements, but also differs in key points. One example of an AUTOSAR architecture on the CAN bus is depicted below. If one compares the two architectures, primarily the following aspects become apparent:
 - ▶ Various standardized basic software modules become defined, e.g. for mode management (Ecum, ComM, BswM, CanSm) and hardware abstraction.
 - ▶ Individual basic software modules are split into several sub layers, e.g. CAN drivers (CAN-IF, CAN-DRV) and network management (NmIf, CanNm).
 - ▶ The application's access to the basic software is via the AUTOSAR Run-Time Environment (RTE). This represents a significant change compared to previous application development, in which a direct interface existed to the basic software components and the hardware.
 - ▶ Functions that - for historical reasons - were arranged in different domain-specific layers converge (e.g. OSEK-COM and FTCOM converge in AUTOSAR COM).

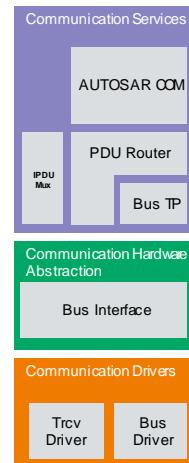
New modules introduced with AUTOSAR 4.x (partly already in AUTOSAR 3.2):

BSWM Basic Software Mode Manager, **STBM** Synchronous Time Base Manager, **CSM** Crypto Service Manager
DBG The Debugging module supports debugging of the AUTOSAR BSW. It interfaces to ECU internal modules and to an external host system via communication .

DLT The module Diagnostic Log and Trace supports logging and tracing of applications. It collects user defined log messages and converts them into a standardized format.

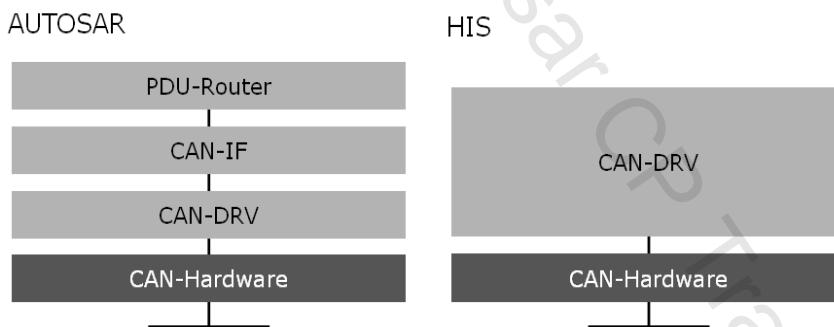
Communication

- ▶ COM
 - ▶ Signal interface
 - ▶ Setup/analysis of PDUs
 - ▶ Signal gateway
- ▶ PDU Router
 - ▶ Abstraction of the bus system relative to higher layers
 - ▶ PDU gateway
- ▶ IPDU Multiplexer (optional)
 - ▶ same CAN ID but different signal layout/content
- ▶ Transport Protocol
 - ▶ Segmented data transmission



53

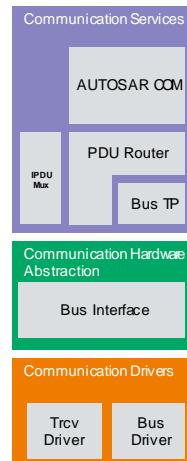
- ▶ The most important reasons for developing a new CAN driver: To separate the driver into a generic, hardware-independent part and a hardware-specific part, and to reduce the complexity of the HIS CAN driver that evolved due to its history.
- ▶ The AUTOSAR CAN driver consists of a hardware-specific driver (CAN-DRV) and the hardware-independent CAN interface (CAN-IF). Some functionalities of the HIS driver are covered by the PDU Router, e.g. distribution of data to higher layers.



- ▶ The PDU Router is responsible for routing (including multi-cast routing) of messages to the same type of bus system (CAN – CAN) or different bus systems (e.g. CAN – FlexRay) and between service layer modules (COM/DCM) and transport protocol modules.
- ▶ The maximum size of PDUs is limited by the underlying communication interface. 0-8 Bytes for CAN and LIN, 0-254 Bytes for FlexRay. Currently, the transport protocol is only used by the Diagnostic Communication Manager (DCM).

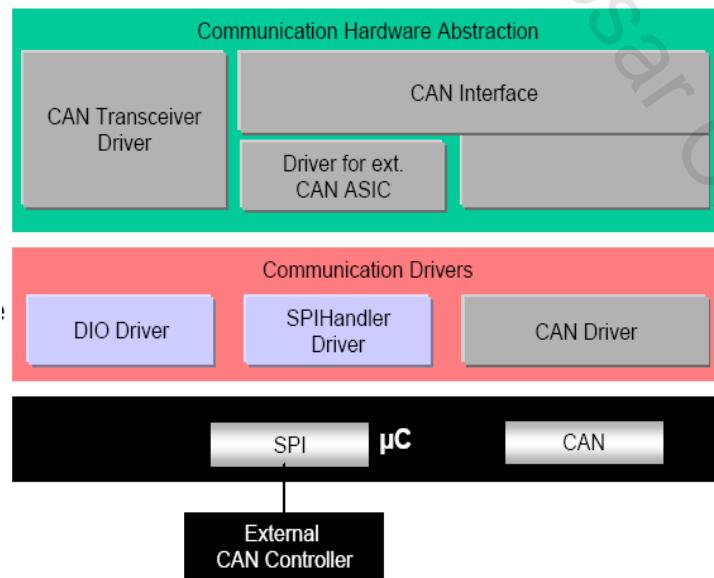
Communication

- ▶ Interface
 - ▶ HW independent, bus-typical functions
 - > Queuing
 - > Assembly of data link layer PDUS/frames (with FlexRay)
 - > Tx/Rx triggering on time-triggered buses (LIN, FlexRay)
- ▶ Driver
 - ▶ HW operation
 - > Initialization, filling of buffer, ISR implementation
- ▶ Transceiver
 - ▶ Operation of hardware line driver circuits



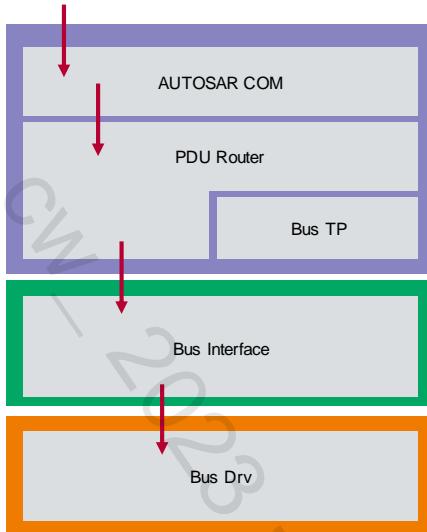
54

- ▶ CAN example with on-board and on-chip CAN controller
- ▶ An ECU has a microcontroller with one internal (on-chip) CAN channel and two other on-board CAN controllers. The on-board CAN controllers are connected to the microcontroller via SPI.
- ▶ The CAN interface provides generic API for CAN communication that is independent of the number of CAN controllers and their locations (on-chip; on-board).



- ▶ Graphics from AUTOSAR_TechnicalOverview.pdf (www.autosar.org)

Communication: Sending Controlled by COM

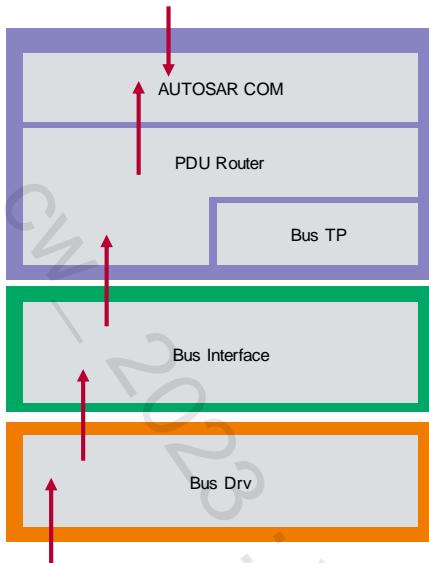


- ▶ e.g. CAN (spontaneous + cyclic), FlexRay (dynamic segment)
- ▶ Writing a signal
 - ▶ Is written to PDU buffer
- ▶ PDU is passed to the PDU Router either immediately or periodically
 - ▶ Each PDU is identified by an ID
- ▶ PDU Router recognizes bus system and passes an associated Interface
 - ▶ Interface recognizes relevant channel and writes PDU to driver or queue
- ▶ Driver sends the message as quickly as possible while observing priorities

55

- ▶ Example of spontaneous sending of application data by the layers of the basic software. While data is only exchanged over the RTE in intra-ECU communication, the components of the basic software are needed in inter-ECU communication.

Communication: Receiving

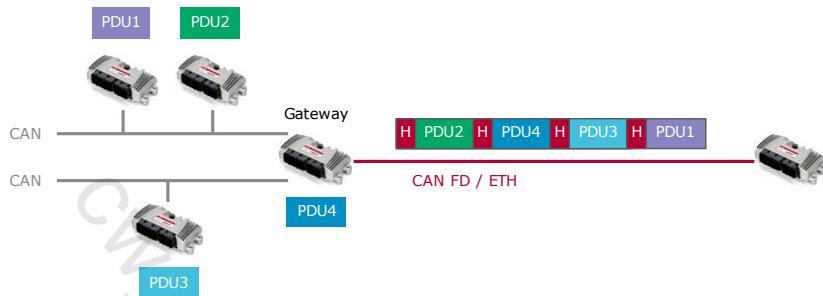


- ▶ Interrupt or polling, here interrupt
- ▶ Receipt of a frame
- ▶ Rx interrupt, processed by driver
- ▶ Data are passed to the interface by RxIndication
- ▶ Transfer to PDU router
- ▶ Transfer to COM
 - ▶ Indication to RTE, if SWCs are used with Data Reception Trigger
 - ▶ Else stored in buffer
- ▶ Signals are read by RTE

56

- ▶ Example of receiving application data through the layers of the basic software.

Container PDU Concept (AUTOSAR 4.2.1)



ECU (e.g. gateway) collects PDUs and transmits them in one CAN FD / ETH frame

- ▶ The order of PDUs within the frame can dynamically change (e.g. if one PDU needn't be sent)
- ▶ Requires additional header information
- ▶ Different frame triggering maybe necessary

Container PDU Concept (AUTOSAR 4.2.1)

- ▶ A Container PDU can aggregate several Contained PDUs
 - ▶ Reduces network protocol overhead (at least for Ethernet)
 - ▶ Increases flexibility
- ▶ Is applied for communication via
 - ▶ Ethernet (SOAD) and
 - ▶ CAN FD (IPDUM)
- ▶ Comparable concept to FlexRay but
 - ▶ for FlexRay, the position of the PDU in the frame is static
 - ▶ for a Container PDU, the position of the PDU in the container is dynamic
- ▶ SOAD and IPDUM interpret a Container via PDU Headers consisting of
 - ▶ Id
 - ▶ DLC

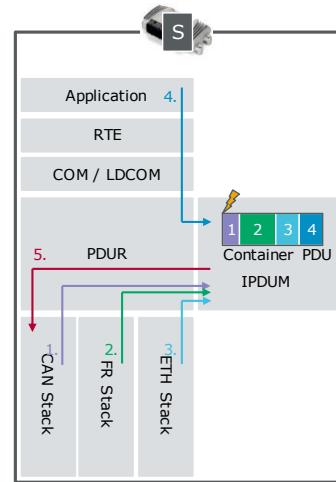


58

- ▶ Multiple-PDU-to-Container (Container PDU) (IPDUM)
- ▶ PDU Header option (SOAD)

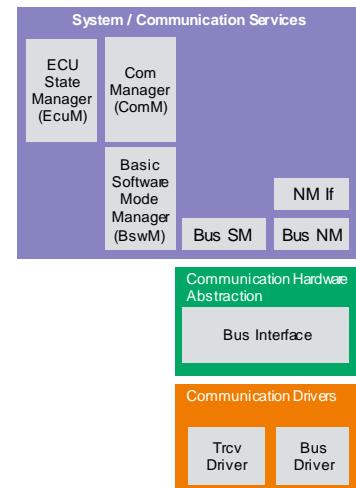
Container PDU Concept (AUTOSAR 4.2.1)

- ▶ Realized in the IPDU Multiplexer (IPDUM)
 - ▶ Network independent
 - ▶ Mapping of
 - > "Contained PDU"-to-"Container PDU"
- ▶ Container PDU (frame) triggering
 - ▶ Container PDU
 - > threshold exceeded
 - > timeout
 - ▶ Contained PDU
 - > timeout
 - > High-priority
 - > First Contained PDU trigger
- ▶ Header formats
 - ▶ 4 bytes header
 - > 3 bytes ID, 1 byte length
 - ▶ 8 bytes header
 - > 4 bytes ID, 4 bytes length



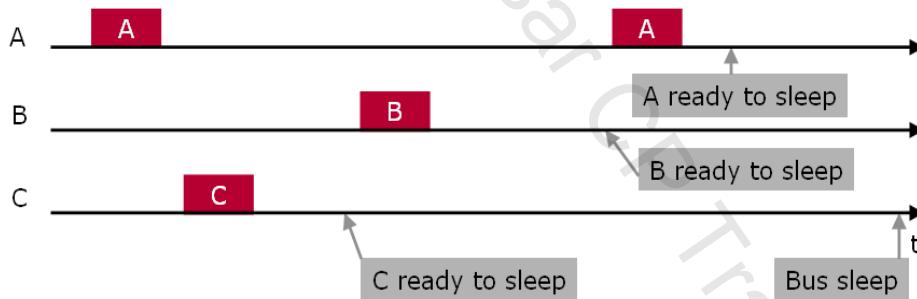
Mode Management

- ▶ ECU State Manager (EcuM)
 - ▶ ECU state machine
 - ▶ Centralizes and validates wake-up events.
 - ▶ Coordination of start-up/shutdown
- ▶ Communication Manager (ComM)
 - ▶ Abstracted bus state machine
 - ▶ Startup/shutdown of communication
- ▶ Network Management (Nm If, Bus NM)
 - ▶ Keep the bus awake, coordinate shutdown
- ▶ Bus State Manager (Bus SM)
 - ▶ Switch transceiver and controller state
- ▶ Basic Software Mode Manager (BswM)
 - ▶ Rule based system to control BSW behavior (e.g. switch IPDU Groups in COM)



60

- ▶ AUTOSAR NM:
 - ▶ Periodic messages while ECU is in normal mode or bus segment is awake
 - ▶ Broadcast principle
 - ▶ Optimization for bus load reduction (only 2 ECUs in the bus segment send NM messages, all others listen)



- ▶ The figure depicts the functionality of AUTOSAR NM, which is based on a broadcast principle, i.e. each node that needs the bus periodically sends its own NM message. If no NM messages appear on the bus for a specific duration of time, the connected ECUs assume the network can be released. To reduce bus load to a minimum and avoid bursts, AUTOSAR NM offers a special bus load reduction algorithm, in which only a 2 nodes participate actively in the NM message traffic.

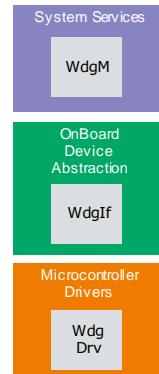
Watchdog

- ▶ Concept
 - ▶ supervise execution of application software
 - ▶ trigger watchdog / no not trigger it any more

- ▶ WdgM
 - ▶ manages alive indications from all supervised entities (Runnables of application)
 - ▶ Derives Global Supervision Status
 - ▶ triggers WdgIf if Runnables behave like expected

- ▶ WdgIf
 - ▶ triggers Hardware Watchdog(s)

- ▶ WdgDrv
 - ▶ Internal or external Watchdog driver



61

AUTOSAR 3.x

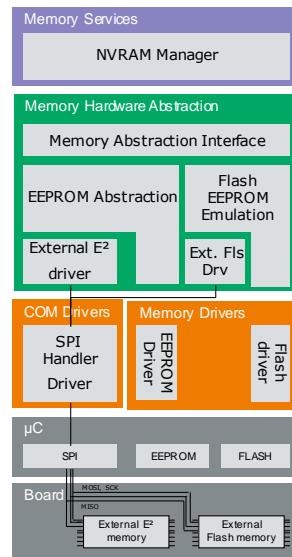
- does not support windowed watchdogs
- Only offers alive supervision, yields only a binary verdict on the operational state of the application: dead or alive.

AUTOSAR 4.x

- Supports windowed watchdogs
- WdgM is no longer responsible to trigger Watchdog directly. Instead the WdgM reports only the status to the WdgIf. The WdgDrv does the actual triggering of the Watchdog Hardware in dependency to the status reported by the WdgM
- Additionally to alive supervision: Check pointing with program flow monitoring and transition timeout supervision. This is a very fine grained supervision of the control flow in the application, combinational and also temporal.
- Enables ISO 26262 Functional Safety applications to be implemented in AUTOSAR. (additionally, an ASIL D compliant Wdg Stack implementation would be required!)

Memory Services

- ▶ Abstracts the locations of internal (on-chip) and external (on-board) memory devices.
- ▶ Same mechanisms for internal and external EEPROM or Flash possible
- ▶ Architectural subdivision into
 - ▶ Service Module: NVRAM manager
 - ▶ ECU Abstraction: Memory abstraction interface
 - ▶ MCAL Layer: Memory drivers



62

- ▶ Example of Memory Abstraction Interface:

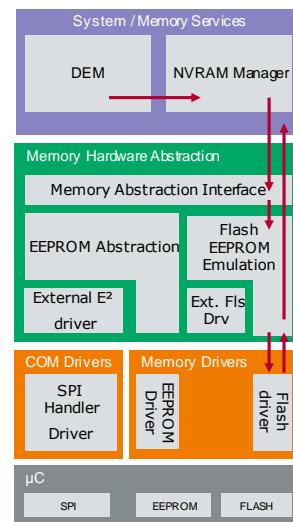
```
Std_ReturnType MemIf_Write
(
    uint8 DeviceIndex,
    uint8 BlockNumber,
    uint8 *DataBufferPtr
)
```

Example of EEPROM Abstraction:

```
Std_ReturnValue Ea_Write
(
    uint8 BlockNumber,
    uint8 *DataBufferPtr
)
```

Memory Services

- ▶ Writing diagnostic snapshot data to NVRAM
- ▶ Write the data as block with ID.
- ▶ Write to specific address.
 - > Memory type(Flash/EEPROM) unknown in NVRAM Manager
- ▶ It is known from the configuration to which memory abstraction the write access should be passed.
- ▶ Access is passed to the driver for internal memory or the driver for external memory.
- ▶ The driver is called periodically to actually write to the memory.
- ▶ Calls the callback of the abstraction when it is finished.
- ▶ Abstraction notifies the NVRAM Manager.

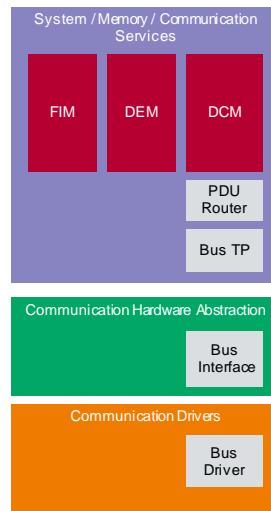


63

- ▶ The NVRAM Manager converts data blocks to memory addresses. The Memory Abstraction Interface converts addresses to the memory system. EEPROM Abstraction or Flash EEPROM Emulation knows whether internal or external memory is being used and addresses the correct driver. Since the EEPROM or Flash is slow compared to program execution, the memory is written piecewise and waits until the write process has ended. Writing occurs in the MainFunction of the driver, which is called periodically. After the entire block has been written an JobEndNotification is passed from the driver to the Abstraction and then directly to the NVRAM Manager.

Diagnostics

- ▶ Diagnostic Communication Manager (DCM)
 - ▶ Implements the diagnostic communication protocol.
- ▶ Diagnostic Event Manager (DEM)
 - ▶ Error memory
 - > Saves diagnostic events
- ▶ Function Inhibition Manager (FIM)
 - ▶ Application functionality is executed conditionally.
 - > Depending on error memory contents (diagnostic events)

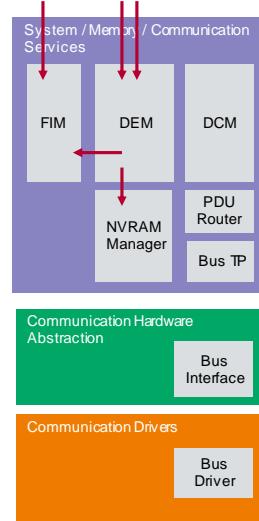


64

- ▶ DCM:
 - ▶ Implements UDS protocol
 - ▶ Is bus independent
- ▶ DEM:
 - ▶ Saves diagnostic events from BSW and application
- ▶ FIM:
 - ▶ Gets current failure status from DEM
 - ▶ Sets inhibition status for a configurable number of DEM events
 - ▶ Inhibition status is polled by application (Fim_GetFunctionPermission)

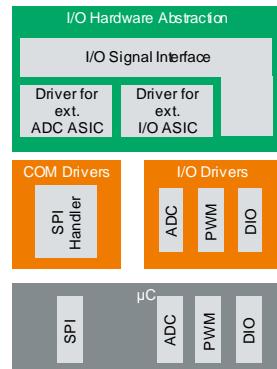
Diagnostics

- ▶ Error state detection and saving in error memory.
 - ▶ Snapshot is saved
 - ▶ Event associated with error is set.
 - ▶ Associated NVRAM block is written.
- ▶ Shutdown of a function due to error memory entries.
 - ▶ FIM is notified that certain events are set in DEM.
 - ▶ In case of error, a SWC is inhibited by Polling by the SWC.

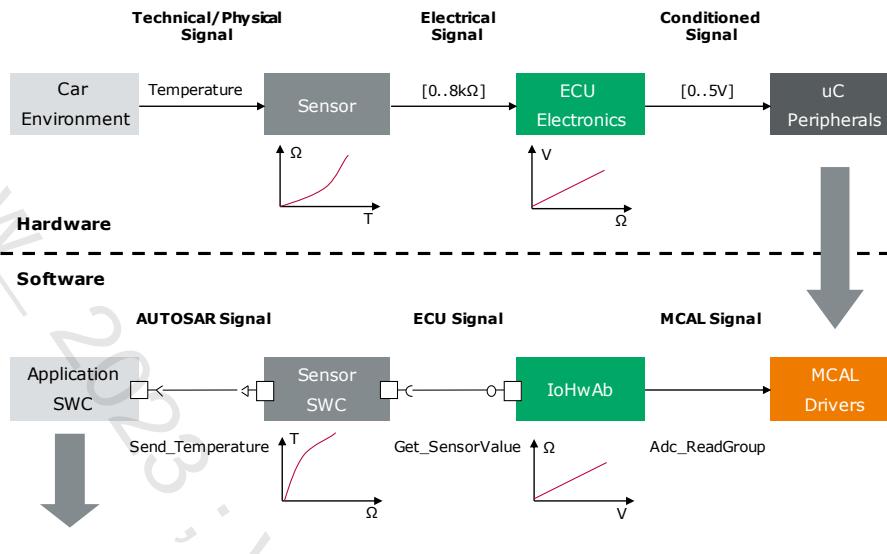


Hardware I/O

- ▶ Abstraction of location of I/O peripheral devices (on-chip / on-board) and layout
- ▶ No abstraction of sensor/actuator
 - ▶ I/O signal interface
 - > Conditioning of the read values of inputs.
 - > e.g. debouncing of signals
 - ▶ Drivers
 - > ADC, DIO, PWM



Hardware I/O

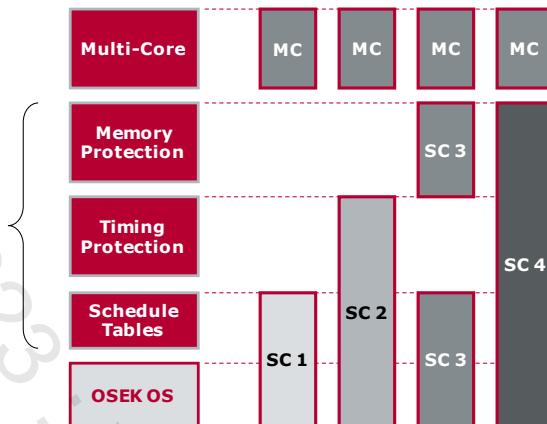


67

- ▶ Conversion of a physical value (e.g. temperature, door status, etc.)
- ▶ Into an electrical signal (e.g. resistance)
- ▶ The signal is adapted by the electronics (ECU electronics) so that the microcontroller can interpret the signal (e.g. voltage)
- ▶ The microcontroller reads in the signal (e.g. via an analog to digital input) and makes it available to “ECU Abstraction” in the form of a software value
- ▶ “ECU Abstraction” makes the temperature available to the RTE.
- ▶ The purpose of the Sensor SWC is to condition or filter data on the application level. It can then pass the data to the other applications.

New Functionality in Scalability Classes

- AUTOSAR **extends** the OSEK/VDX Operating System standard
- AUTOSAR OS add-ons are segmented into Scalability Classes (SCs)
- Multi-Core support is optional and independent of Scalability Class



68

Memory Protection

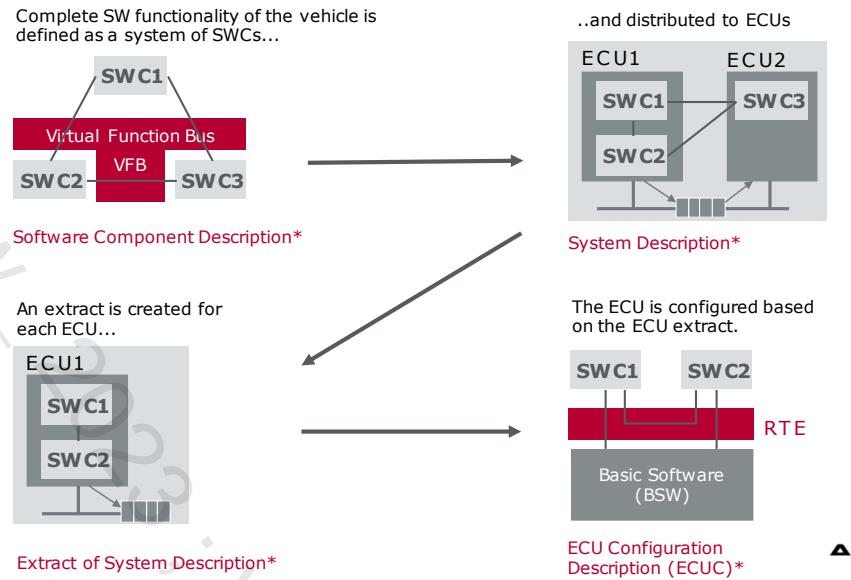
The hardware must contain a Memory Protection Unit (MPU) to support this feature.

Agenda

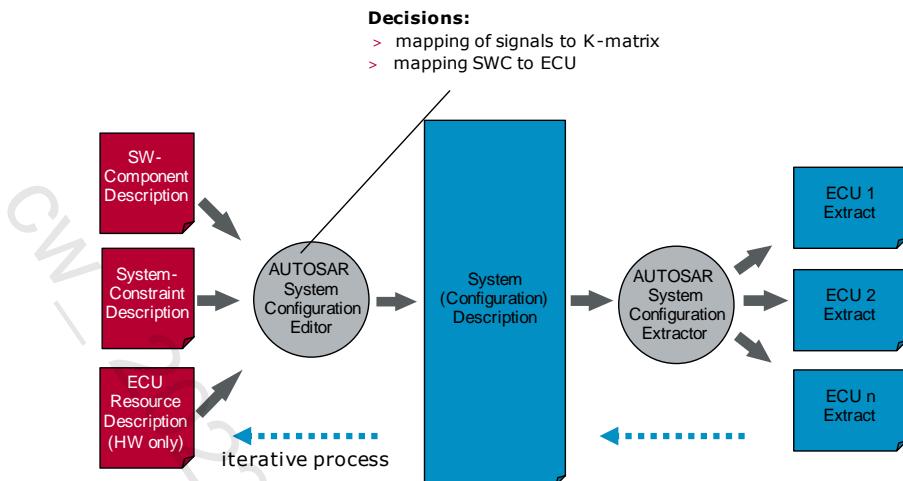
- Overview and Objectives
- AUTOSAR Application
- AUTOSAR RTE
- AUTOSAR BSW
- **AUTOSAR Methodology**
- AUTOSAR in Practice
- Appendix

– 2023 – VH Autosar CP Training_EN

AUTOSAR Workflow

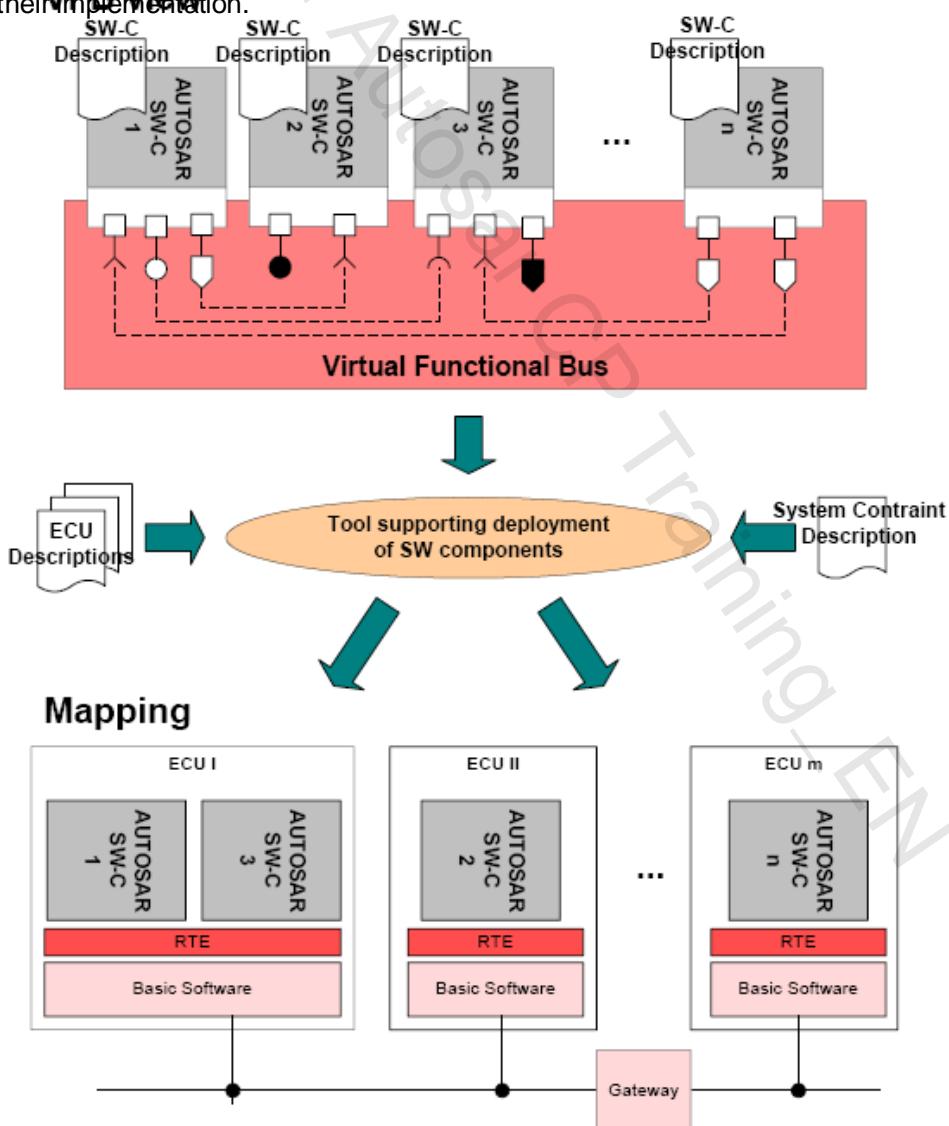


System Description



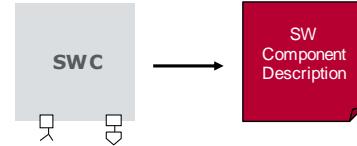
71

- ▶ Configuration of the “System Configuration Description” is based on the description of the SW components and not on their implementation.



Software Component Description

- ▶ The AUTOSAR “Software Component Description” contains the following information:
 - ▶ Data and operations which are part of the component.
 - ▶ Resource needs of the component (memory, CPU time, etc.).
 - ▶ Information relating to the specific implementation (repetition rate).
 - ▶ Component interfaces.
- ▶ Structure and format are described by the “Software Component Template”.



72

- ▶ The Software Component Description contains detailed information on the ports, interfaces and connections between software components.
- ▶ The parts of the descriptions used in the methodology are prescribed by the AUTOSAR Meta Model, which was created by the AUTOSAR development partnership in the modeling language UML (Unified Modeling Language). Several hundred UML classes with relevant attributes and associations describe all information needed by the AUTOSAR methodology. From the meta model so-called “templates” can be generated, which form the framework for the three input descriptions. The memory format of the “templates” and that of the “descriptions” are also objects of standardization, in order to assure properly functional exchanges of descriptions between an OEM and suppliers. The XML language is used for this purpose. Thanks to the central positioning of the meta model as a starting point for “templates, “descriptions” and memory formats, consistent terminology is achieved within AUTOSAR, inconsistencies are avoided and maintainability is improved.

ECU Description

- ▶ The AUTOSAR “ECU Resource Description” contains the following information:
 - ▶ Description of the hardware being used
 - ▶ Sensor, actuator
 - ▶ Memory
 - ▶ Processor
 - ▶ Communications periphery
 - ▶ Pin assignments
 - ▶ The structure and format are described by the “ECU Resource Template”.

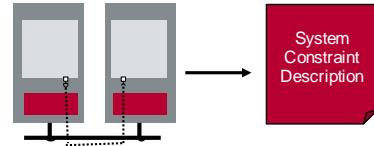


73

- ▶ The ECU Resource Description describes the available hardware resources.

System Description

- ▶ The AUTOSAR “System Constraint Description” contains the following information:
 - ▶ Information on network topologies
 - ▶ Limitations (“Constraints”)
 - ▶ Protocols
 - ▶ K-matrix
 - ▶ Baud rate and timing parameters
- ▶ Structure and format are described by the “System Template”.

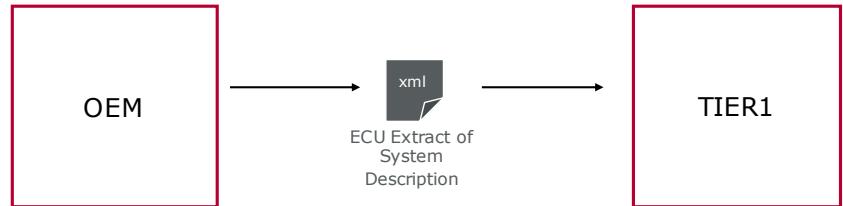


74

- ▶ The System Constraint Description describes the conditions of an existing or planned network architecture.

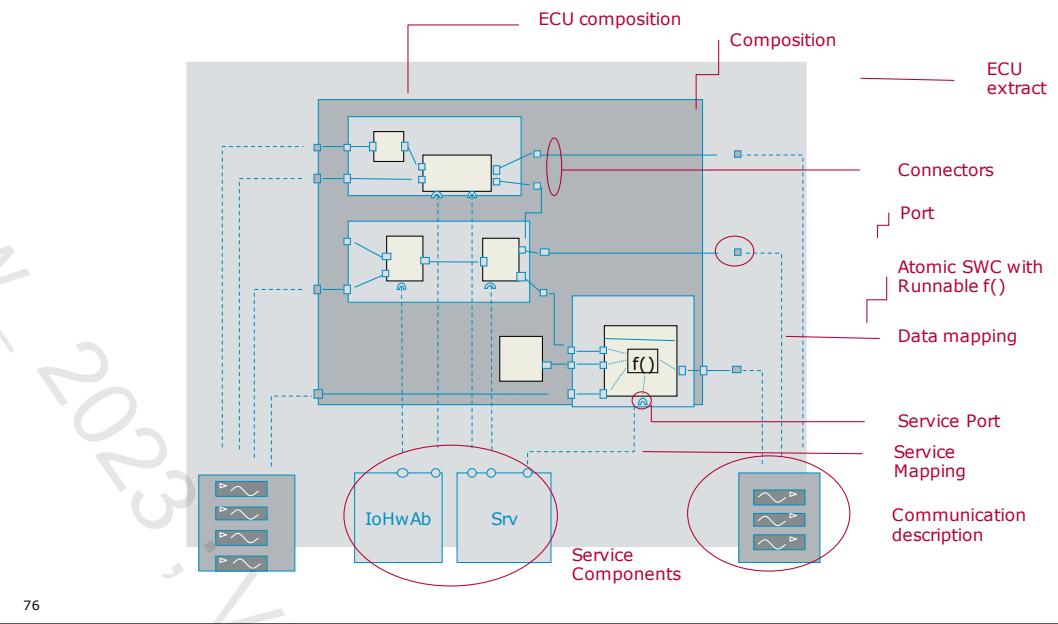
AUTOSAR Workflow with OEM and TIER1

- ▶ Interface between OEM* and TIER1*: ECU extract of System Description (ECUEX)
- ▶ OEM creates ECUEX based on vehicle system design
- ▶ TIER1 configures AUTOSAR ECU based on ECUEX



*Note: "OEM" and "TIER1" may also be organizational units within one company ("System Responsible" vs. "ECU responsible")

Content of ECUEX



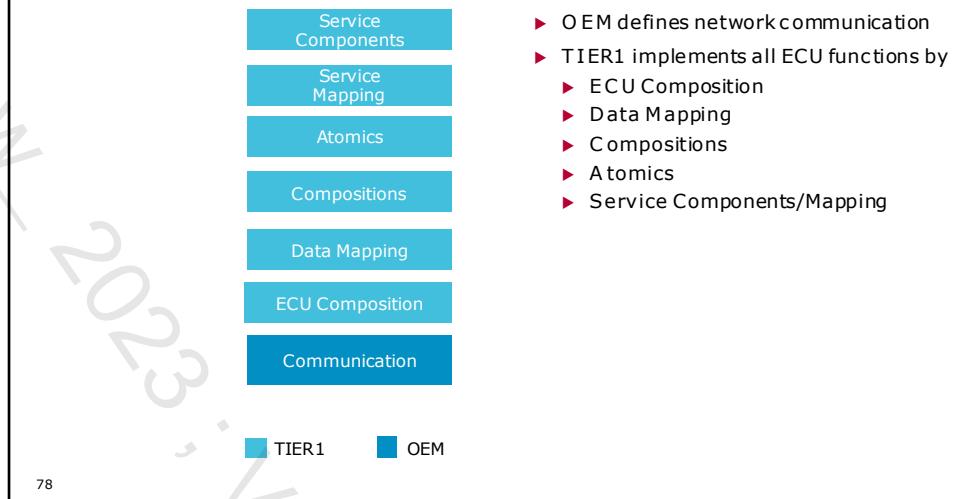
Content of ECUEX

- ▶ Communication description of the ECU
 - ▶ Tx-/Rx-Frames, IDs, network signals
 - ▶ Send types
 - ▶ PDUs
- ▶ SWCs, Ports (grouped within a central ECU composition)
 - ▶ Atomic SWCs, Runnables
 - ▶ Compositions
- ▶ Port Interfaces, Data Types, Constants
- ▶ Connectors between the SWCs
- ▶ Data Mapping (connecting SWCs with network signals)
- ▶ Service Components (providing interfaces of the actually used BSW services for the SWCs – need to match with the ECU Configuration)
- ▶ Service Mapping (connecting SWCs with Service Components)

AUTOSAR XML format allows also ECUEX files with only a subset of this content!

Use Case 1

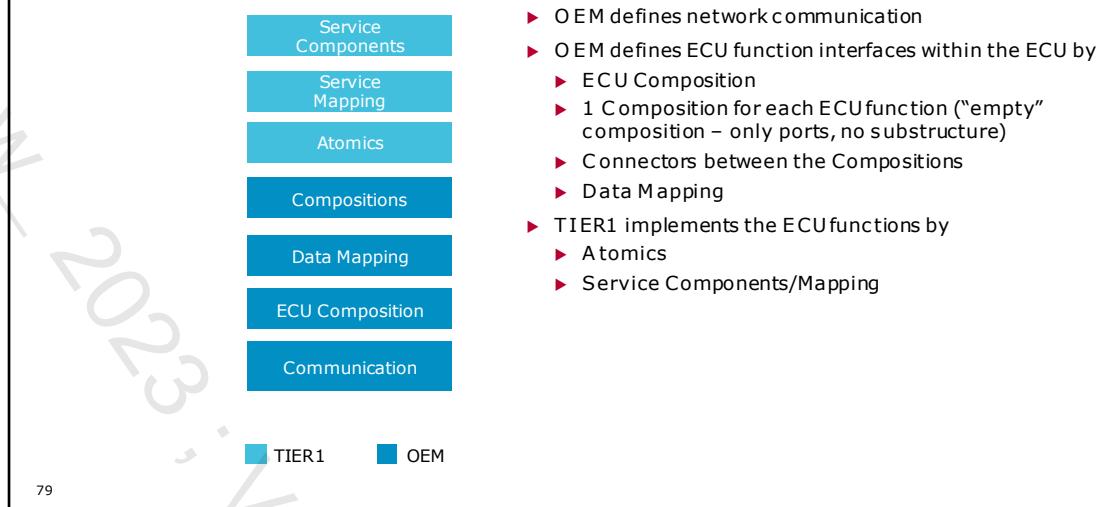
- ▶ Use Case 1: Network-oriented approach (same process as today, only with new AUTOSAR XML format instead of DBC/FIBEX/LDF).



- ▶ Challenges
 - ▶ Typically, an OEM cannot provide the full content of the ECUEX because
 - > Some content is only known after ECU integration (e.g. Service Components)
 - > Some content is not in the responsibility of the OEM (e.g. SW implementation details)
 - > The respective process supporting AUTOSAR workflows is not fully established at the OEM
 - ▶ TIER1 must complete the ECUEX
 - ▶ Iterations must be supported
 - > TIER1 work must not be lost when getting update of ECUEX from OEM
 - ▶ Several use cases possible
 - > Some examples are shown in the next slides
- ▶ Experience
 - ▶ Ideal situation: clearly separated responsibility regarding the content of the ECUEX
 - > No conflicts due to concurrent modification of same content by OEM and TIER1
 - ▶ Realistic: TIER1 requires flexibility to make modifications within content created by OEM
 - > Add TIER1 standard SW, which is not in responsibility of OEM
 - > Additional ports (Service Ports, Application Ports)
 - > Additional top-level components
 - > Define additional networks, which are not in responsibility of OEM
 - > e.g. private CAN within the ECU, LIN to sensor cluster
 - > Fix issues in the OEM content to avoid delays in development

Use Case 2

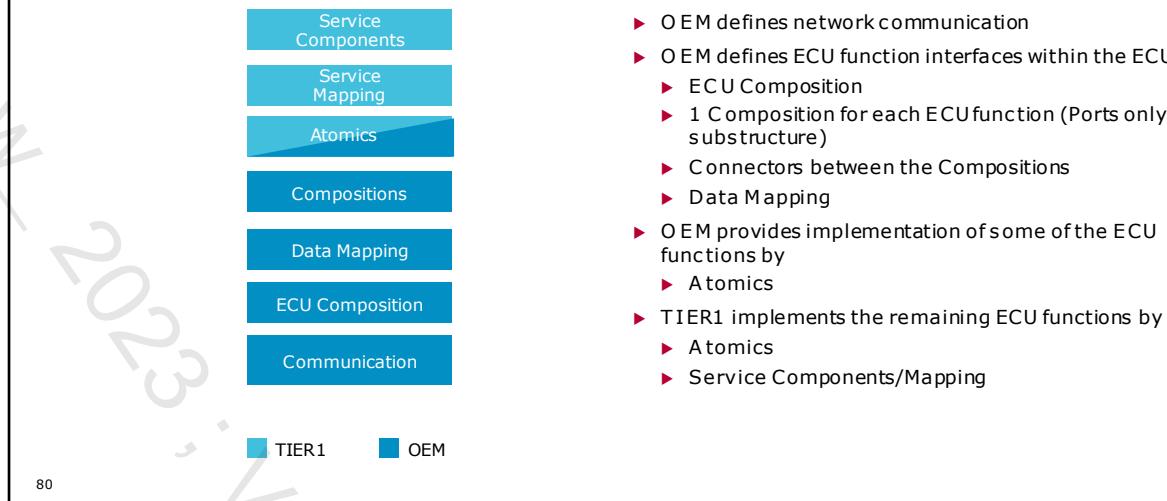
- ▶ Use Case 2: Functional system design approach, top-level architecture of ECU functional SW specified by OEM.



79

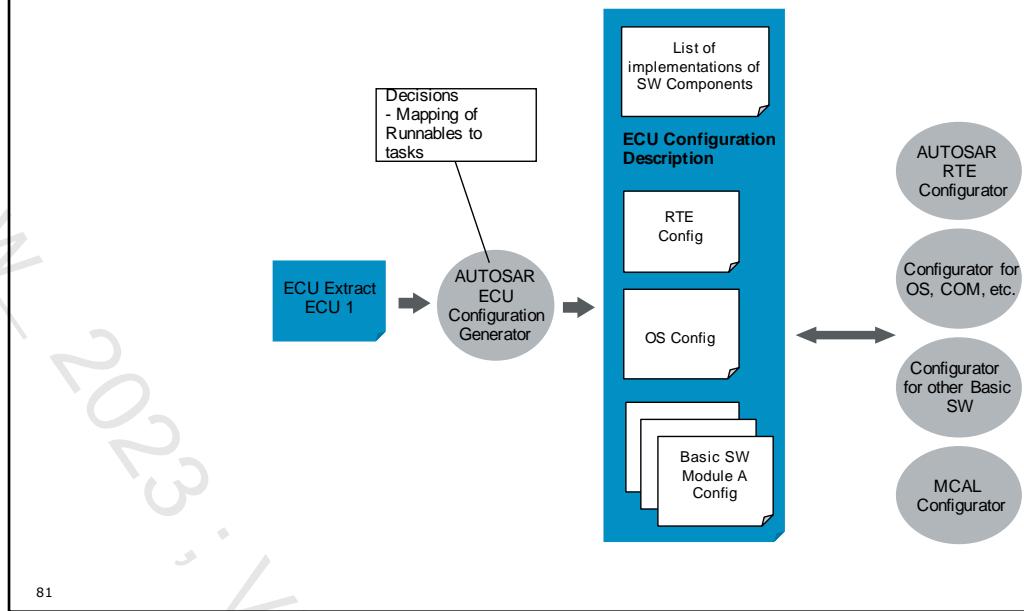
Use Case 3

- ▶ Use Case 3: Functional system design approach, top-level architecture of ECU functional SW specified by OEM, SW partially implemented by OEM.



80

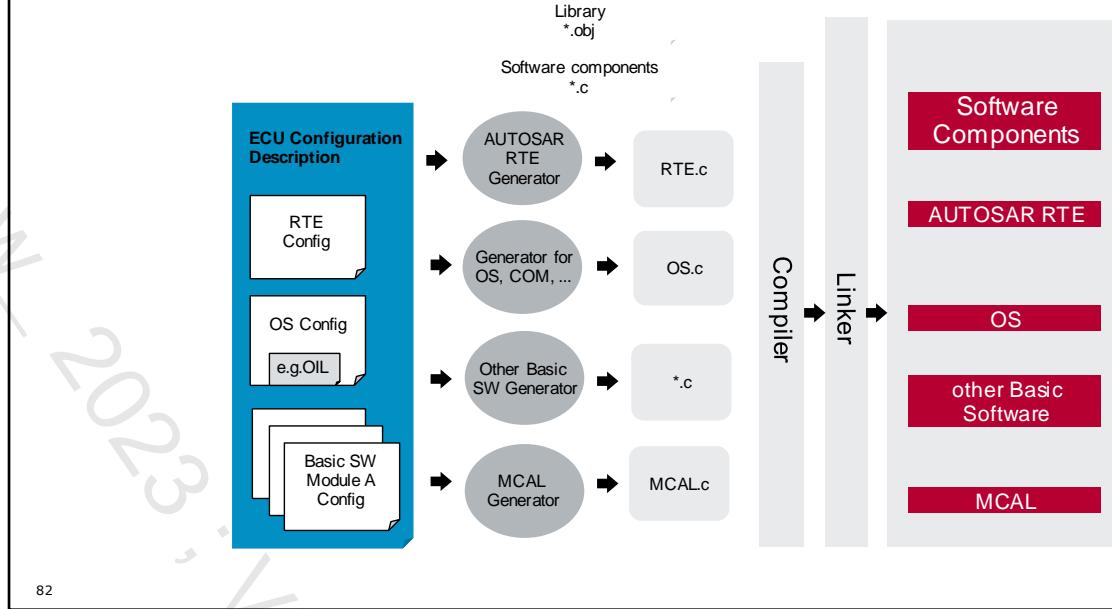
ECU Configuration



81

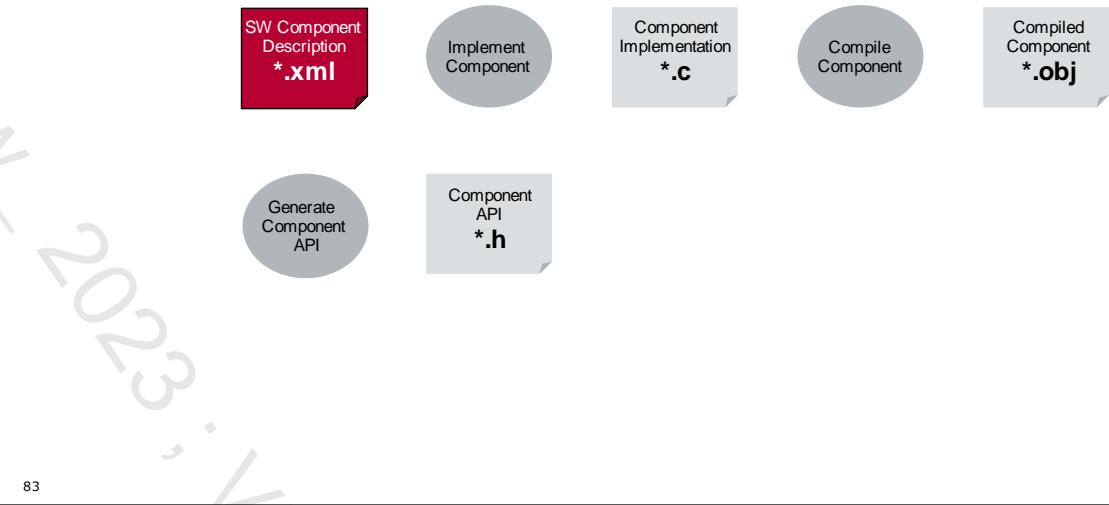
- In the next step, the information of the ECU Description is used to configure the local basic software of the ECU. Other ECU design decisions are also made (e.g. OS Scheduling).

Executable ECU



- The complete ECU Configuration Description is used to generate the basic software and integrate it with the application. Various generators from different producers may be used in this process, which results in a binary for the relevant ECU.

Component Development Process



1. Creation of the component description (runnables, events, data elements). No description of functional behavior.
2. Generation of API components declarations related to RTE communication for header.
3. Functional development (programming and testing) of the component is nearly independent of the overall system. Besides producing the C code, a component description is also produced, which contains information on the implementation (compiler settings and optimizations).
4. The object file is generated from the API header and C code. There is also a component description containing information on the build process (linker settings, etc.).

Agenda

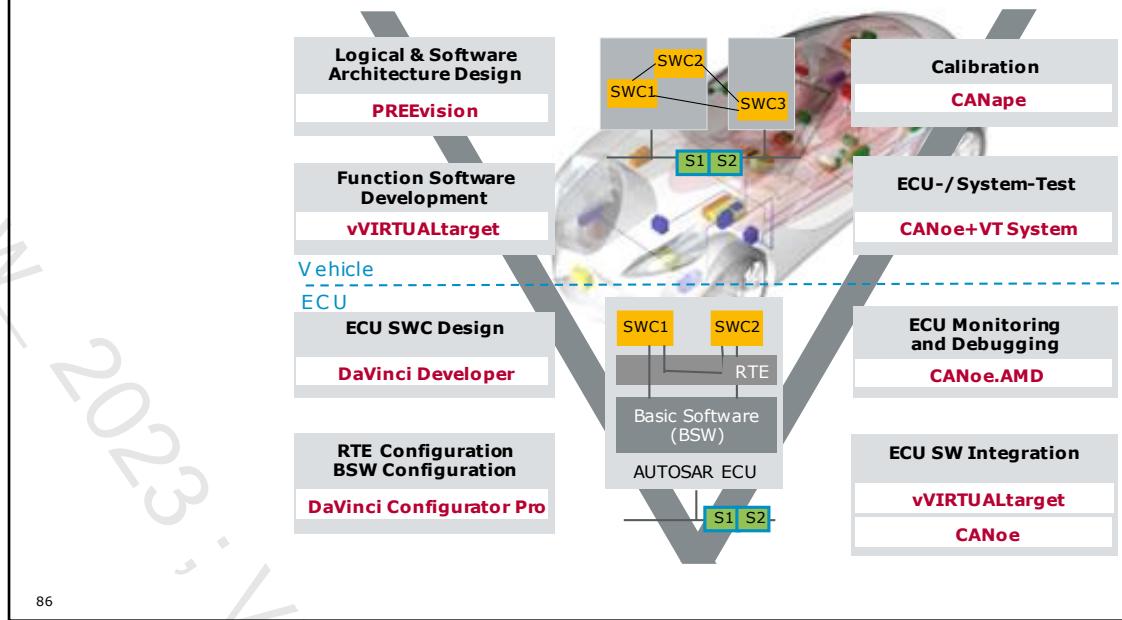
- Overview and Objectives
- AUTOSAR Application
- AUTOSAR RTE
- AUTOSAR BSW
- AUTOSAR Methodology
- **AUTOSAR in Practice**
- Appendix

2023_VH_Autosar CP Training_EN

Exchange Formats

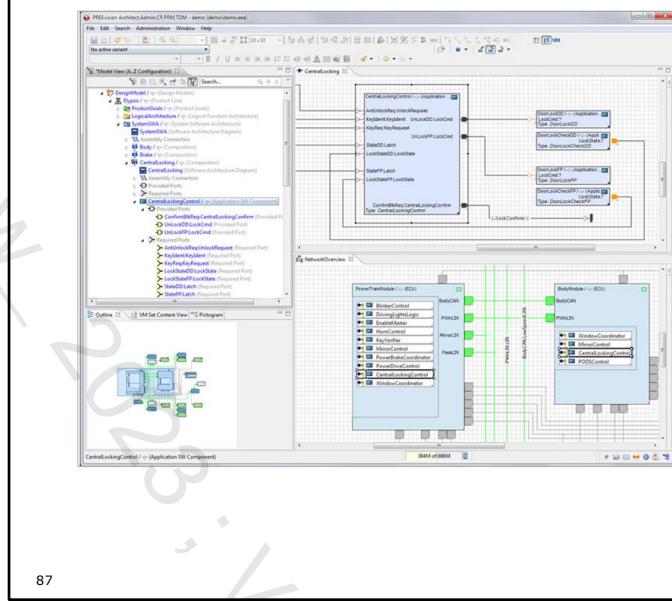
- ▶ System Extract / ECU extract of System Description (SYSD/ECUEX)
 - ▶ AUTOSAR XML format
 - ▶ Contains communication data of one ECU → used instead of DBC, LDF or FIBEX, for config CAN, LIN and FR stack
 - ▶ May contain AUTOSAR SWCs
 - ▶ This format is created by System Design tools, e.g. PREEvision, or in-house database system OEM
- ▶ Basic Software Module Description (BSWMD)
 - ▶ AUTOSAR XML format
 - ▶ Contains a detailed description of all available parameters of the BSW module. Configuration use this information to build up a configuration (ECUC)
- ▶ ECU Configuration Description (ECUC)
 - ▶ AUTOSAR XML format
 - ▶ Contains the module configurations (i.e. the concrete parameter values) of all BSW module RTE
- ▶ Software Component Description (SWC description)

AUTOSAR Tool Chain

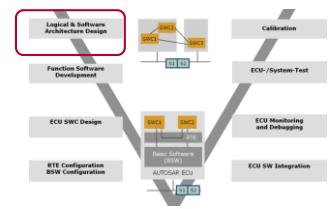


86

PREEvision System Design

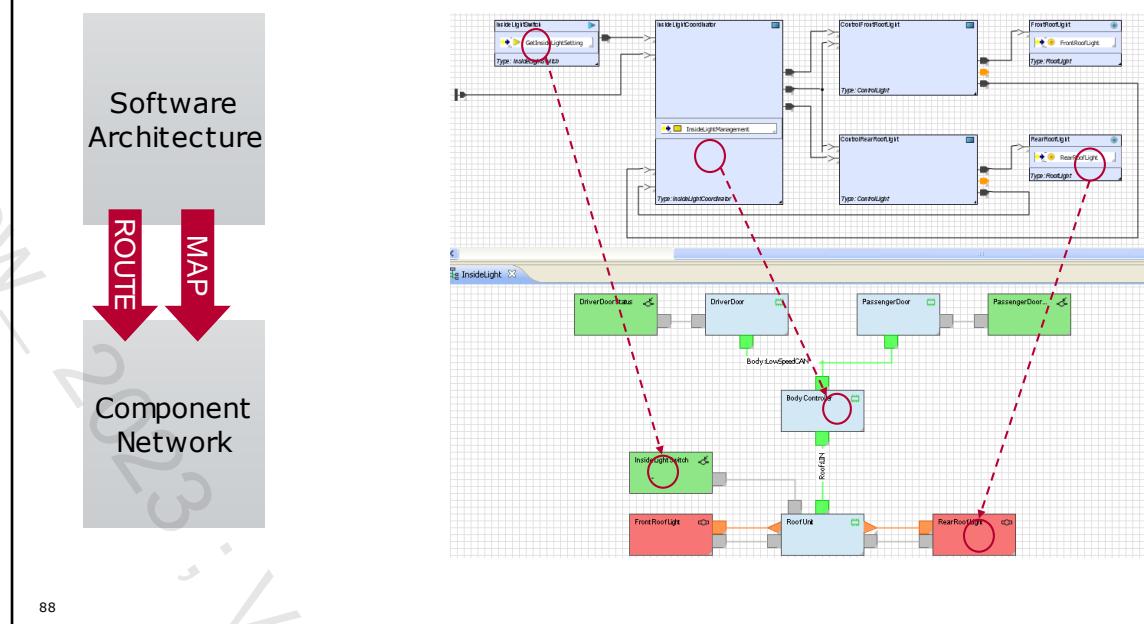


87



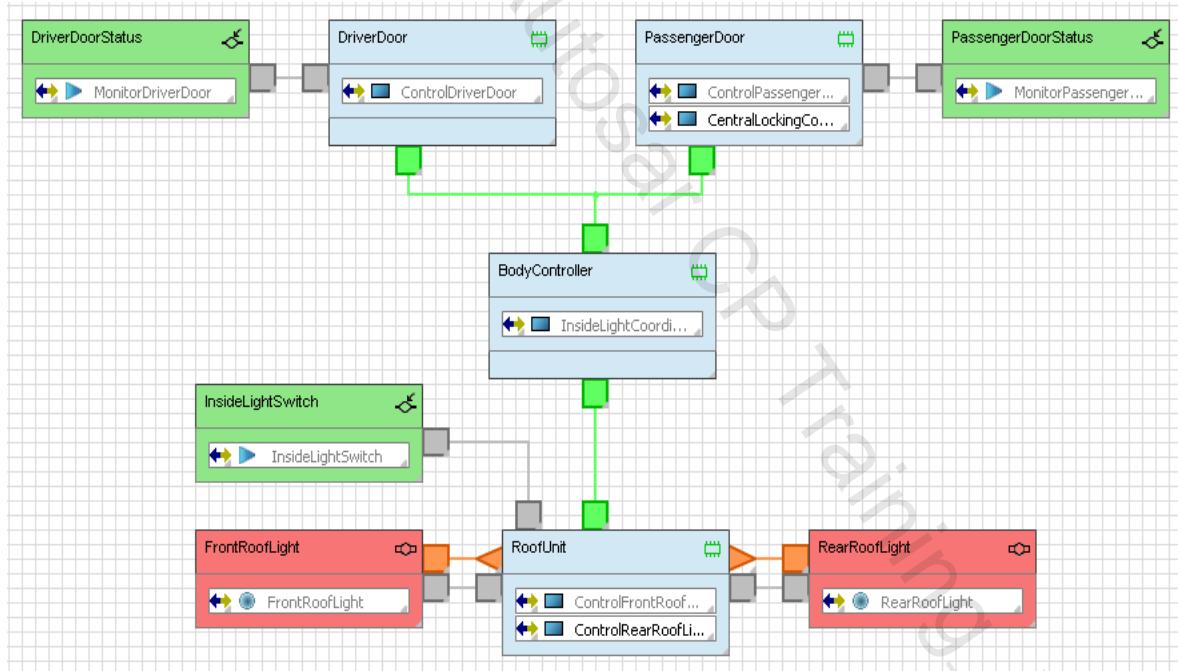
- ▶ System design of vehicles or ECU
- ▶ Software components
- ▶ Network communication
- ▶ Mapping
- ▶ Supports also various architectures beyond AUTOSAR, e.g.
 - ▶ Requirements
 - ▶ Electrical Wiring Harness

PREEvision System Design

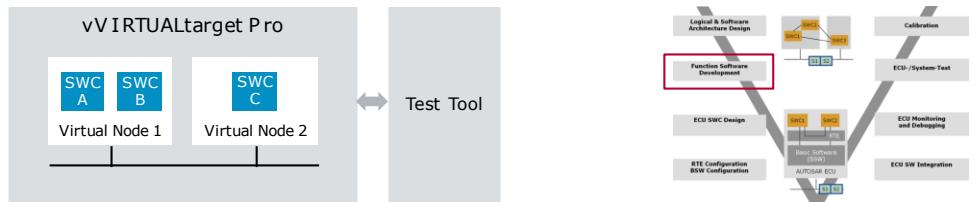


88

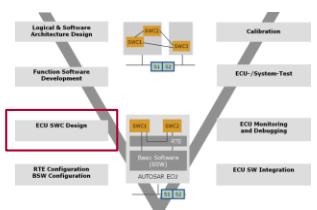
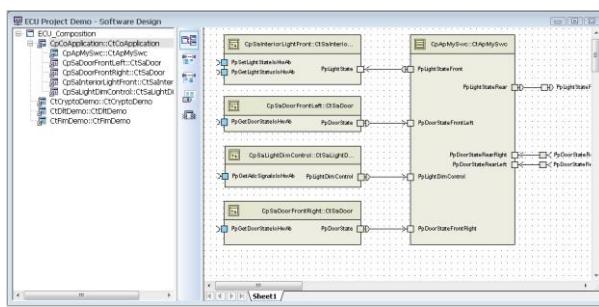
- Having mapped all Software the Network looks like this:



vVIRTUALtarget Pro

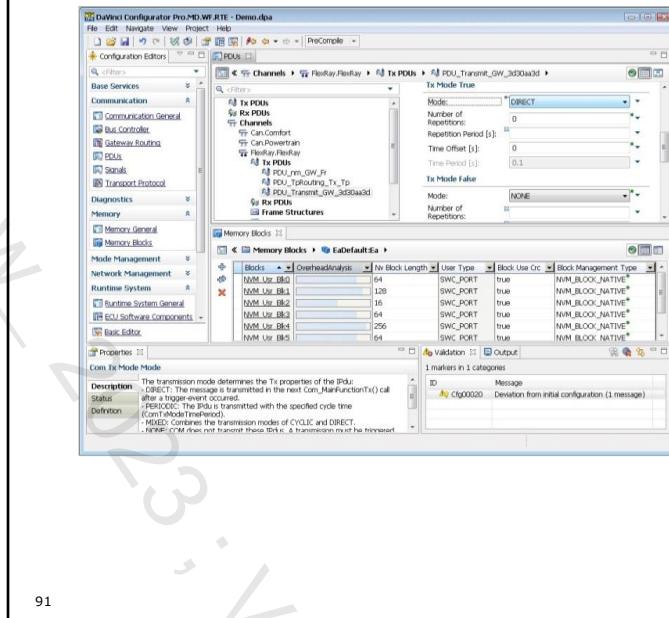


- ▶ Validation of distributed functions
 - ▶ Run SWCs in a virtual environment
 - ▶ Realistic simulation of AUTOSAR ECU behavior and network communication
- ▶ CANoe as test tool
- ▶ API to connect other test tools

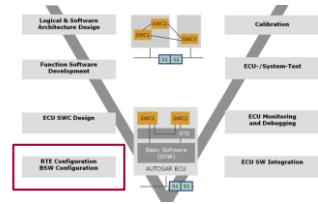


- ▶ Creation of SWC descriptions with graphical or table-based editors
- ▶ Definition of SWC internal behavior (Runnable entities)
- ▶ Consistency check of the SWCs
- ▶ Data exchange analysis

DaVinci Configurator Pro



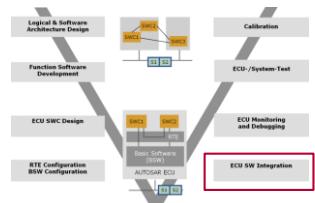
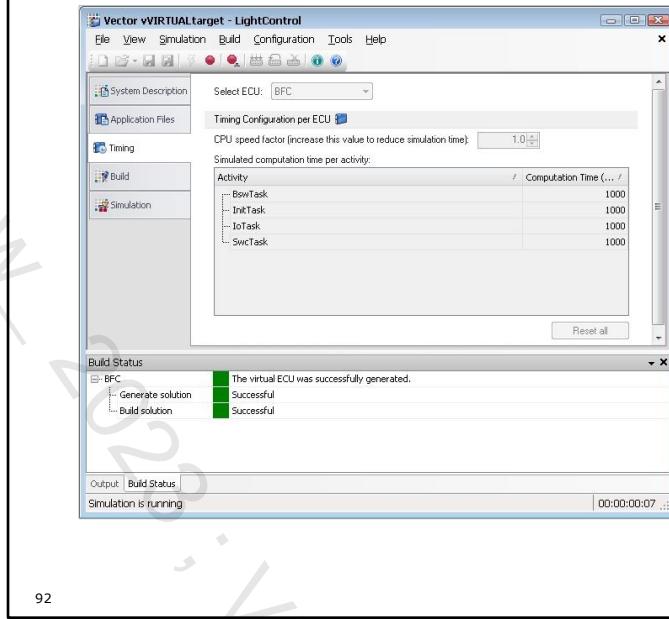
91



▶ Creation of ECU configurations

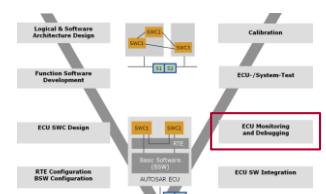
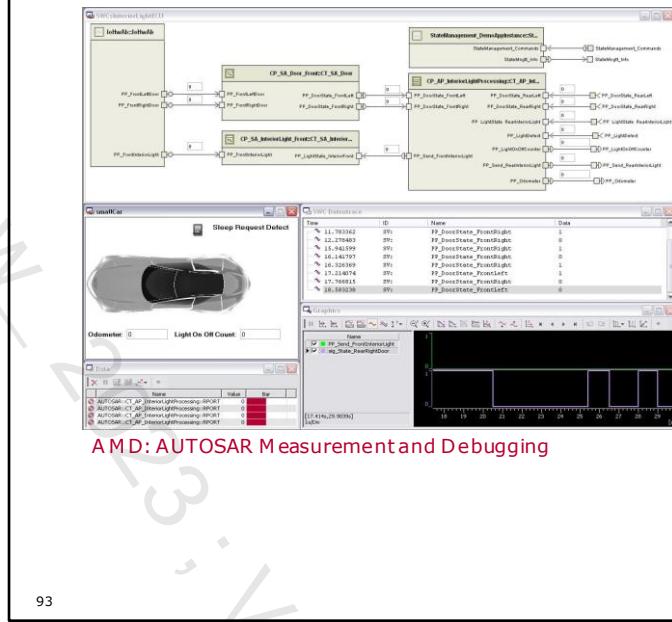
- ▶ MICROSAR BSW and RTE
- ▶ Third party BSW (MCALs)
- ▶ Validation function incl. "click-to..."
- ▶ Use case specific editors and ass. functions
- ▶ Smart project update function
- ▶ Generation of configurable part o... and RTE

DaVinci Configurator Pro.VTT + vVIRTUALtarget



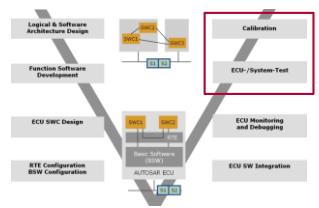
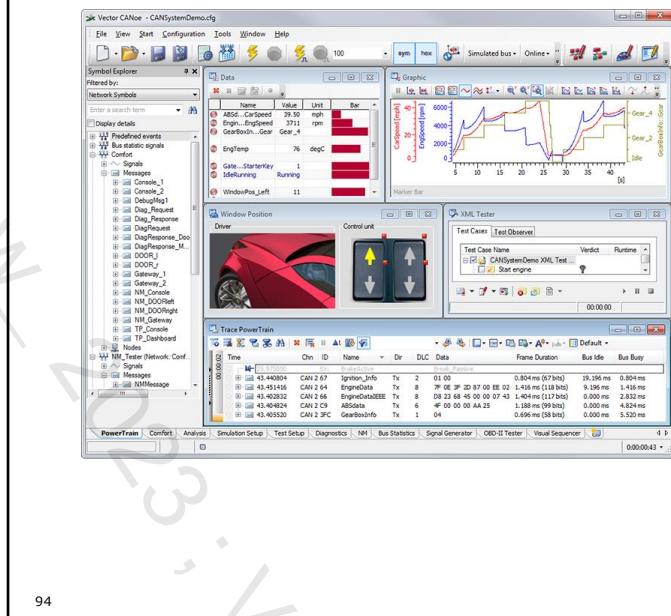
- ▶ Integration of ECU SW in a virtual environment
- ▶ Single-source ECU configuration for real and virtual target
- ▶ Execution of ECU code on the component vVIRTUALtarget
- ▶ CANoe or other tools as test tool
- ▶ Debugging of ECU code on the component vVIRTUALtarget

CANoe.AMD



- ▶ Real-time access to ECU status
- ▶ Runtime measurement of SWC co
- ▶ Notification of ECU internal errors
- ▶ XCP-based data access by CANOE

CANoe/CANape



- ▶ ECU Test by stimulation and measurement of ECU interfaces
 - ▶ Bus Communication
 - ▶ I/O
- ▶ Calibration of ECU parameters

Agenda

Overview and Objectives

AUTOSAR Application

AUTOSAR RTE

AUTOSAR BSW

AUTOSAR Methodology

AUTOSAR in Practice

► Appendix



Documents for Reference

- ▶ **Introduction**
 - ▶ AUTOSAR_TechnicalOverview.pdf
 - ▶ AUTOSAR_LayeredSoftwareArchitecture.pdf
- ▶ **Methodology**
 - ▶ AUTOSAR_Methodology.pdf
- ▶ **RTE**
 - ▶ AUTOSAR_SWS_RTE.pdf
 - ▶ AUTOSAR_SoftwareComponentTemplate.pdf
- ▶ **BSW**
 - ▶ AUTOSAR_BasicSoftwareModules.pdf
- ▶ For additional information go to:
 - ▶ <http://www.autosar.org> or <http://www.vector.com>

Appendix

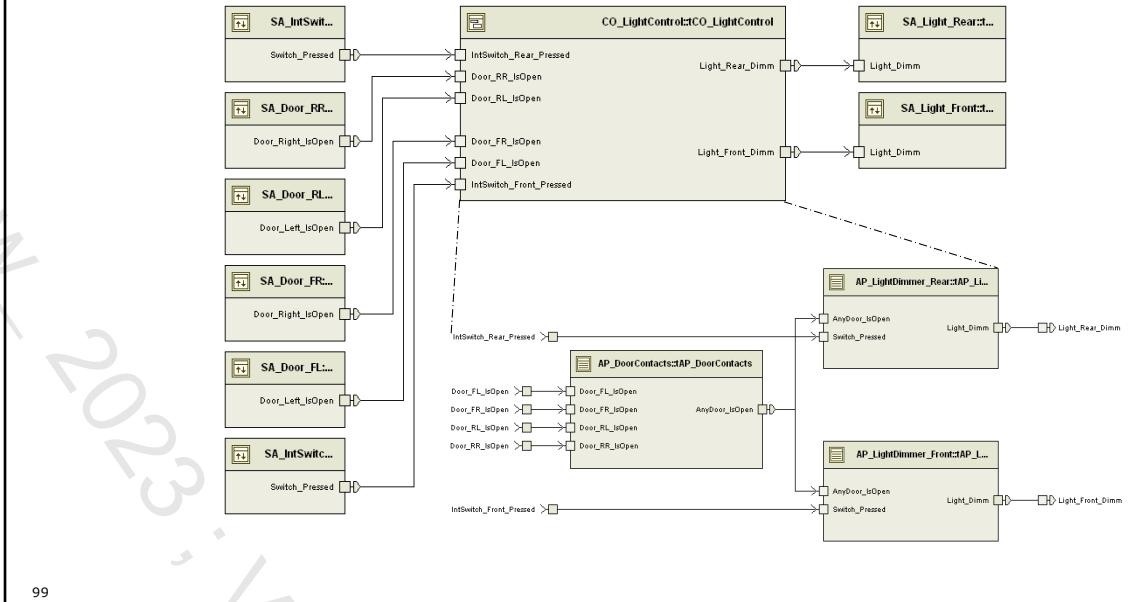
List of Acronyms/Abbreviations 1

- ▶ API Application Programming Interface
- ▶ ASAM MCD Association for Standardization of Automation - and Measuring Systems Measurement, Calibration and Diagnostics
- ▶ AUTOSAR Automotive Open System Architecture
- ▶ BSW Basic Software
- ▶ Bus Bus stands for CAN, FlexRay, LIN, Ethernet or TTCAN
- ▶ CAL Communication Abstraction Layer
- ▶ CAN Controller Area Network
- ▶ CCP CAN Calibration Protocol
- ▶ COM Communication Layer
- ▶ ComM Communication Manager
- ▶ CPU Central Processing Unit
- ▶ ECU Electronic Control Unit
- ▶ EcuM ECU Manager
- ▶ HAL Hardware Abstraction Layer

List of Acronyms/Abbreviations 2

- ▶ HIS Hersteller Initiative Software ["OEM Initiative Software"]
- ▶ I-PDU Interaction Layer Protocol Data Unit
- ▶ MCAL MicroController Abstraction Layer
- ▶ NM Network Management
- ▶ OEM Original Equipment Manufacturer
- ▶ OS Operating System
- ▶ PDU Router Protocol Data Unit Router
- ▶ RTE Runtime Environment
- ▶ SPAL Standard Peripheral Abstraction Layer
- ▶ SW Software
- ▶ SWC Software Component
- ▶ TP Transport Protocol
- ▶ VFB Virtual Functional Bus
- ▶ XCP Universal Measurement and Calibration Protocol
- ▶ XML Extensible Markup Language

Solution to Exercise 1



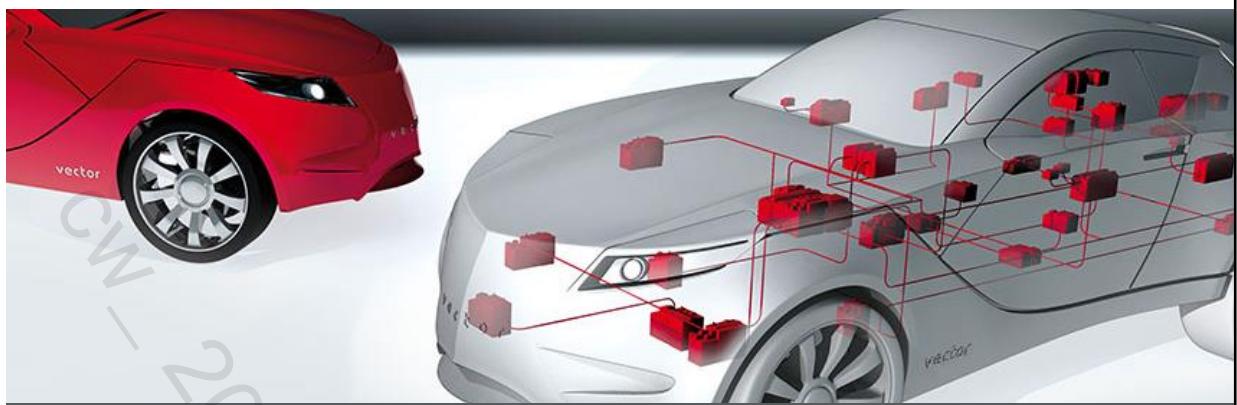
For more information about Vector
and our products please visit

www.vector.com

Author:
Vector Academy
Vector Germany

© 2021. Vector Informatik GmbH. All rights reserved. Any distribution or copying is subject to prior written approval by Vector. V1.0.0 | 2023-03-28





AUTOSAR in Practice

Introduction

V1.0.0 | 2020-01-20

Agenda

► **Introduction**

Workflows

Coming up next

Prerequisite for this training

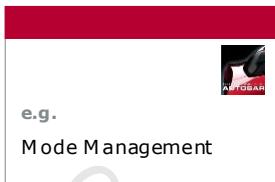


AUTOSAR Seminar
Basics and Theory



AUTOSAR in Practice
Practice and Experience

How does this training work?



Training Slides

- ▶ Agenda
- ▶ Introduction
- ▶ Operating System
- ▶ Software Components
- ▶ I/O
- ▶ Communication
- ▶ Mode Management
- ▶ Memory
- ▶ Diagnostics
- ▶ Busses (for reference only)

Slides are divided into

- ▶ Theory
- ▶ Exercise description

THEORY

Theory

EXERCISE

Exercise
Description

Recap: What we already know

...from AUTOSAR fundamental training or from other sources

- ▶ Software Components (SWCs)
 - ▶ Compositions
 - ▶ Atomics
- ▶ Architecture
 - ▶ Hierarchy of SWCs
- ▶ Methodology
 - ▶ Workflow between OEM and Tier 1
- ▶ Exchange Data Formats
 - ▶ ARXML schema
 - ▶ ECUEX, ECUC, BSWMD, and SWC descriptions
- ▶ RTE
 - ▶ Middleware which makes use of OS and BSW and abstracts communication
- ▶ BSW
 - ▶ Functional clusters, layered SW Architecture
- ▶ Tooling has a vital role
 - ▶ Work can only be accomplished with AUTOSAR Tools



What is coming up in the next three days?

- ▶ Acquire working experience with the DaVinci Tools (Developer, Configurator Pro)
 - ▶ Configure the BSW with Configurator Pro
 - ▶ Set up Software Components (SWCs)
 - ▶ Assemble SWCs in a software design
 - ▶ Generate embedded C code for an ECU
 - ▶ Generate and debug the RTE
 - ▶ Program Runnables
 - ▶ Simulate the result via CANoe

[Let's start >>](#)

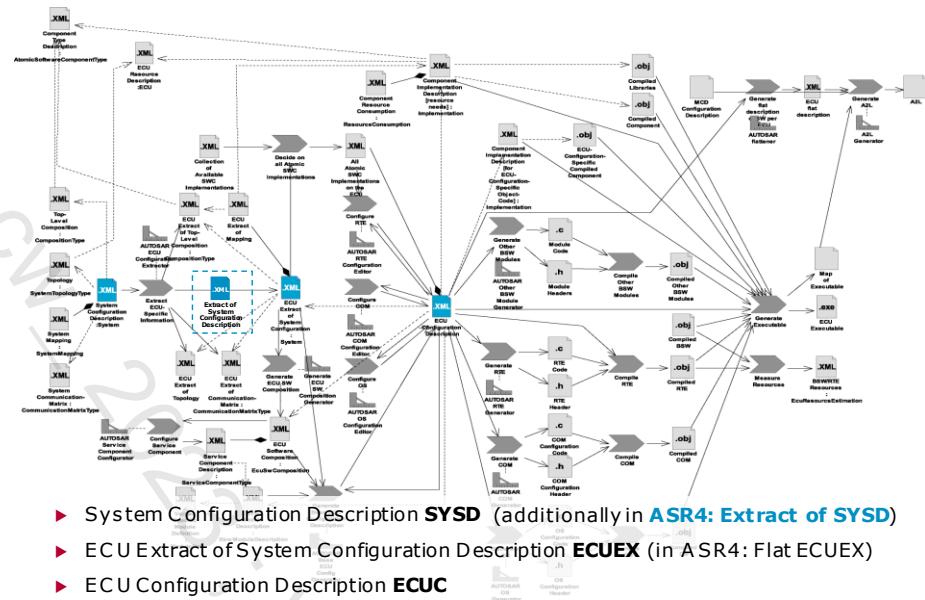
Agenda

Introduction

► **Workflows**

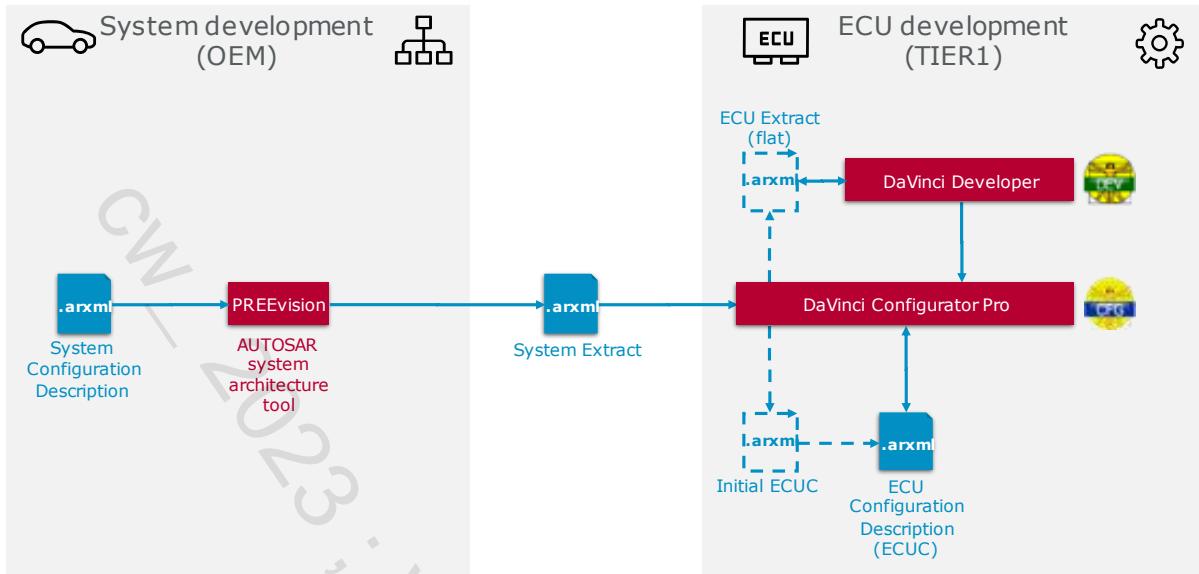
Coming up next

The AUTOSAR Methodology

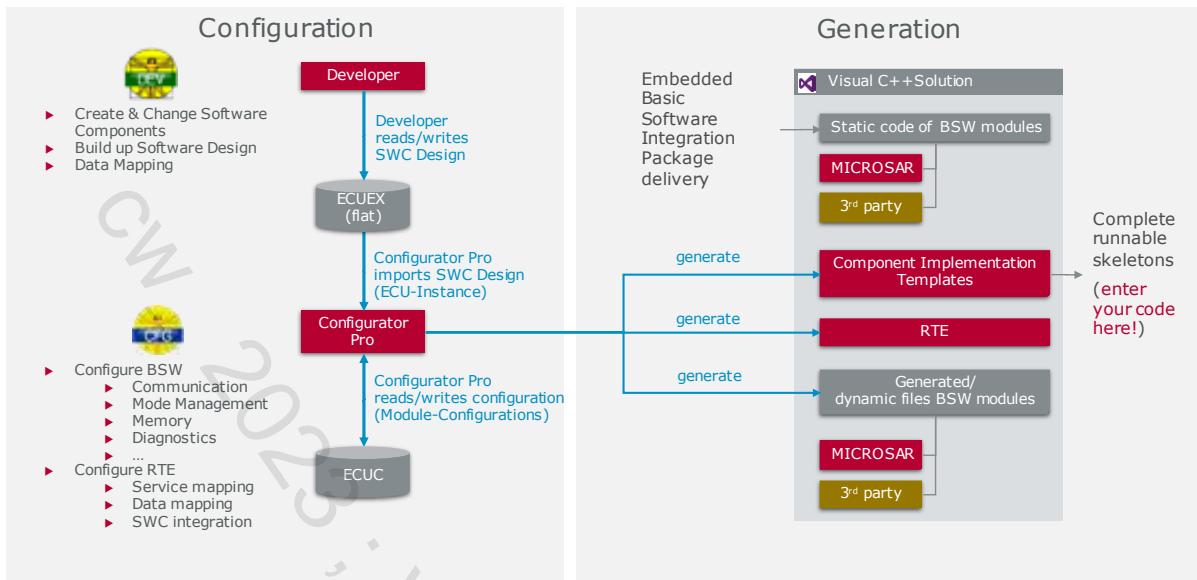


- ▶ System Configuration Description **SYSD** (additionally in ASR4: Extract of SYSD)
- ▶ ECU Extract of System Configuration Description **ECUEx** (in ASR4: Flat ECUEx)
- ▶ ECU Configuration Description **ECUC**

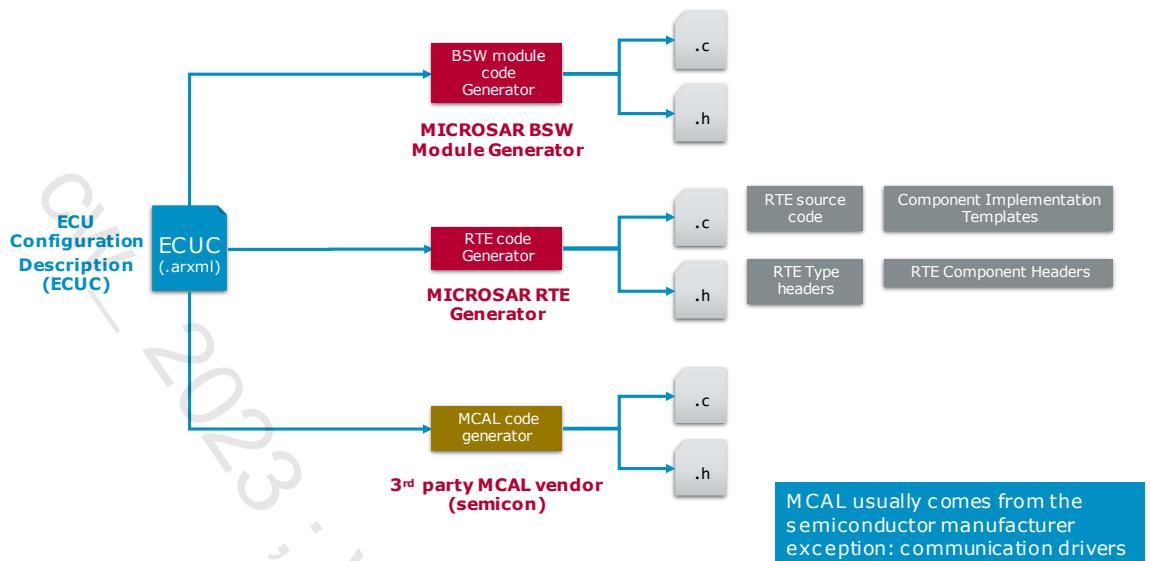
Division of work between OEM and TIER1



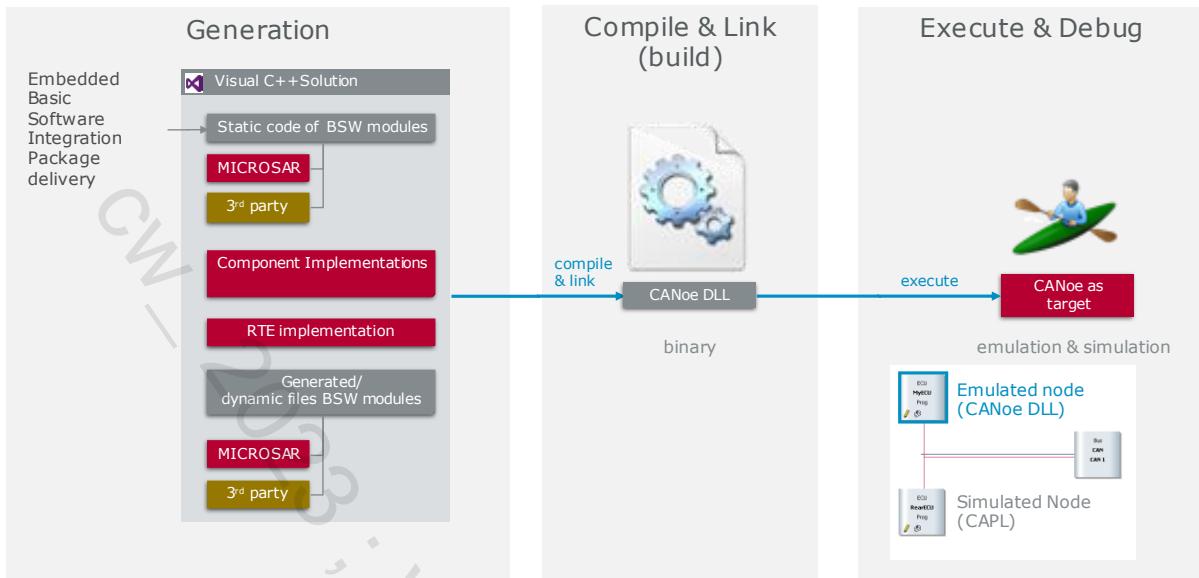
ECU configuration and generation of configuration dependent code



Embedded Software Code Generation

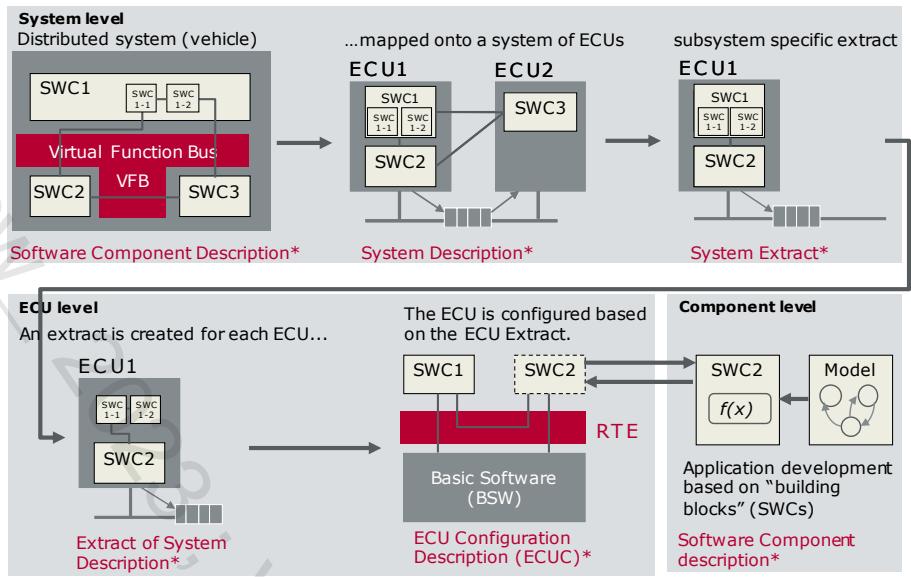


build, execute and debug



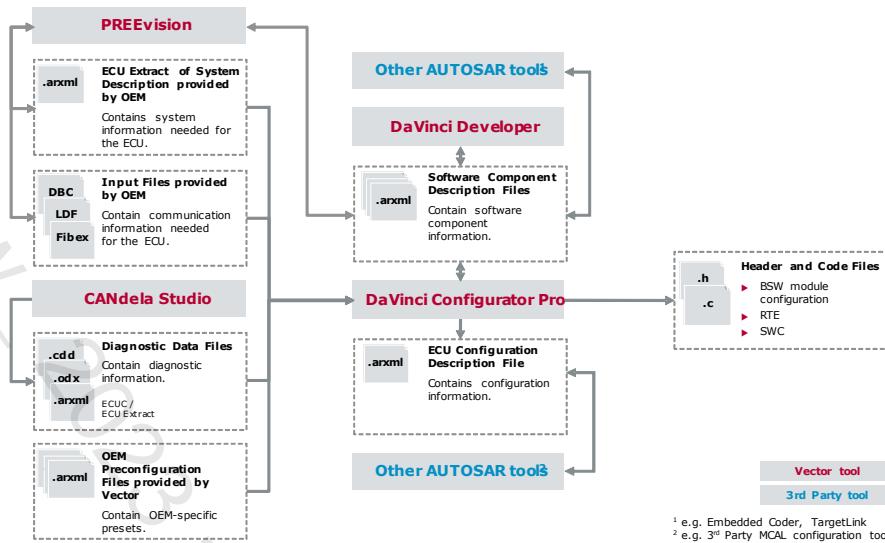
12

AUTOSAR methodology in a nutshell



13

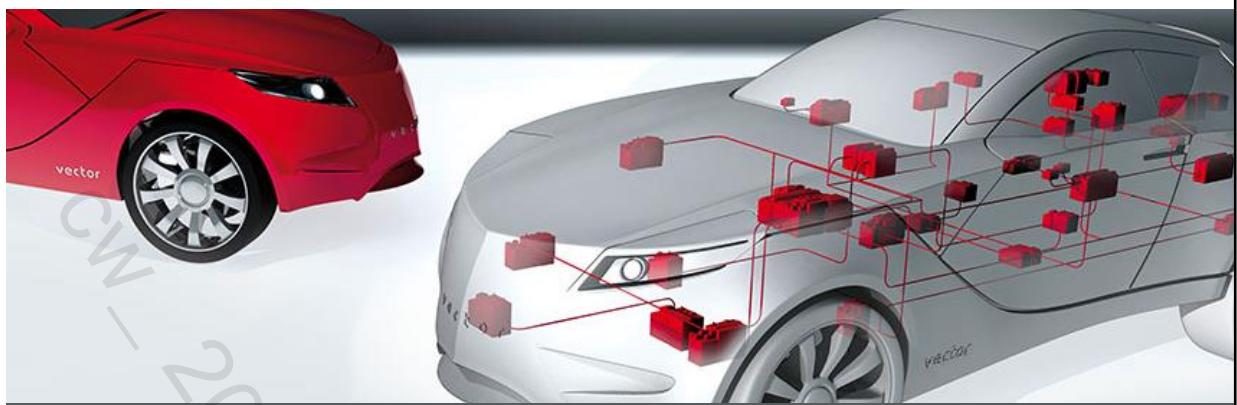
Complete MICROSAR Workflow according to AUTOSAR 4.x



Introduction

Operating System

CW_2023_VH Autosar CP Training_EN



AUTOSAR in Practice

Operating System

V1.0.0 | 2020-01-20

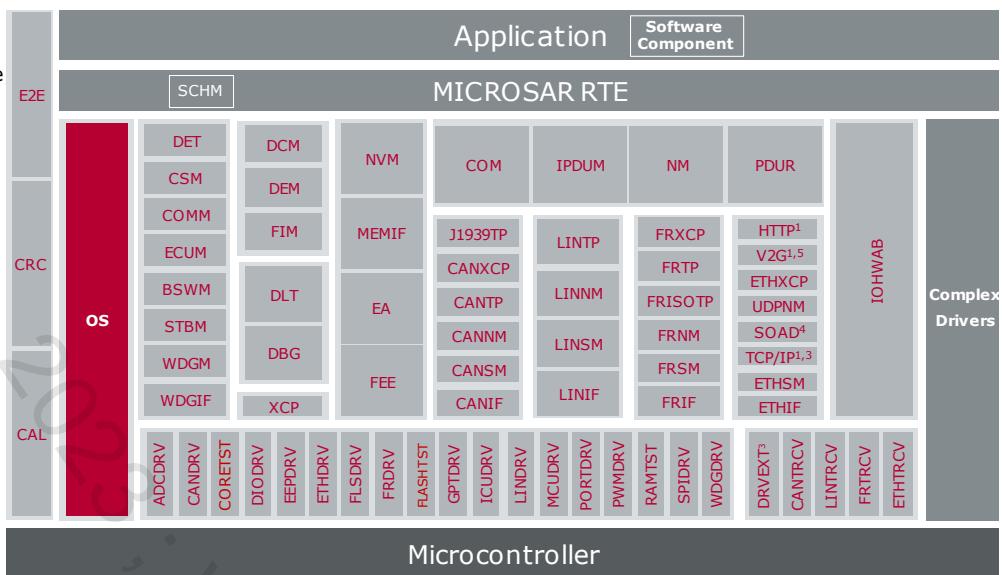
Agenda

► Overview

- Tasks
- Alarms
- OS counter
- Interrupts
- IOCs
- OS Resources
- Illustration
- Scalability of the OS
- Tool example
- Coming up next

MICROSAR OS

- ▶ Static operating system
- ▶ Fully-preemptive event triggered
- ▶ Scalable in functionality and size
- ▶ Elements
 - ▶ Tasks
 - ▶ Events
 - ▶ Alarms
 - ▶ Interrupts
 - ▶ IOCs
 - ▶ Resources



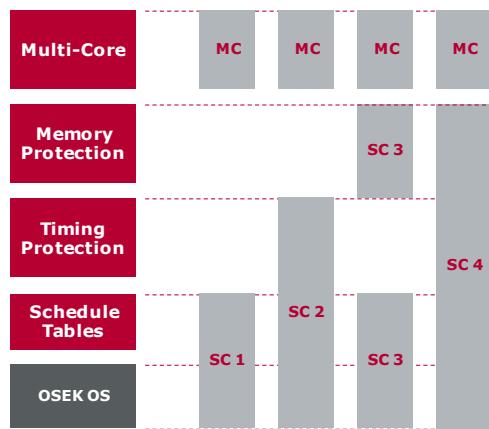
New functionality in Scalability Classes

- ▶ AUTOSAR **extends** the OSEK/VDX Operating System standard
- ▶ AUTOSAR Workflow supported (configuration not any more over OSEK Implementation Language)
- ▶ Additional functionality
- ▶ AUTOSAR OS add-ons are segmented into Scalability Classes (SCs)
 - ▶ Exception
 - > Multi-Core support is independent of the used Scalability Class



Memory Protection

The hardware must contain a Memory Protection Unit (MPU) to support this feature.



Agenda

Overview

► **Tasks**

Alarms

OS counter

Interrupts

IOCs

OS Resources

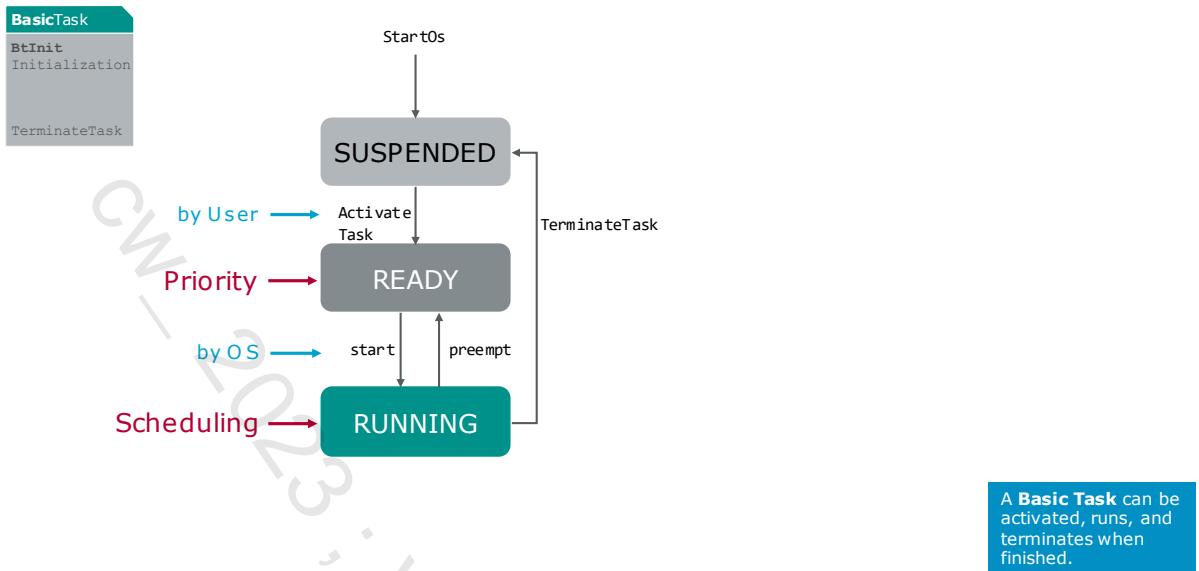
Illustration

Scalability of the OS

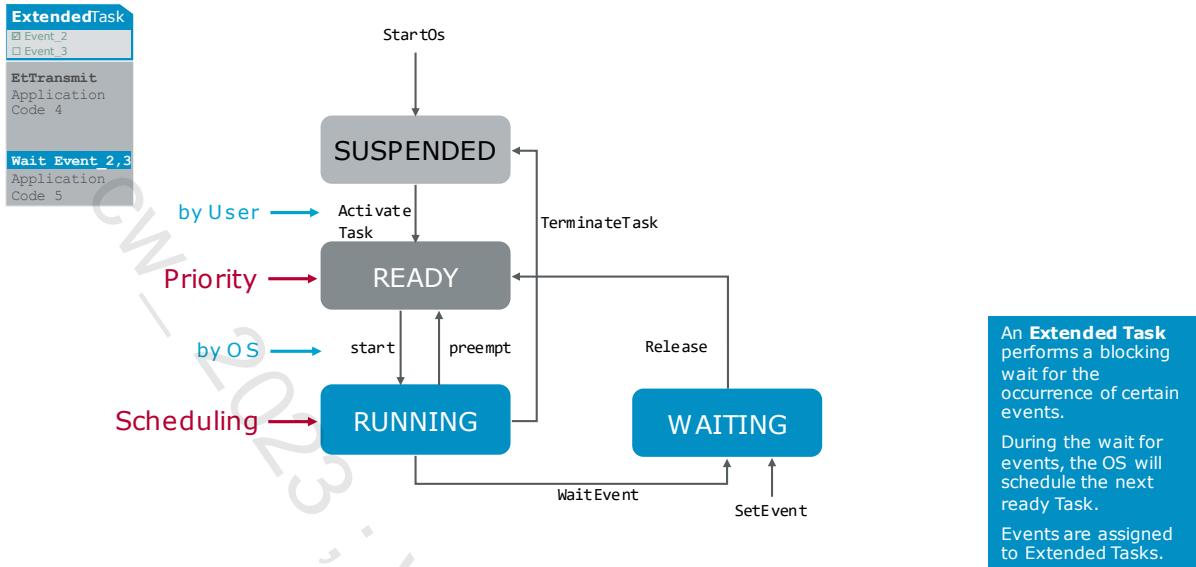
Tool example

Coming up next

Basic Tasks



Extended Tasks and Events



t is a context

“context” of a thread of execution (task/ISR) consists of

A set of processor core registers and their contents

- ▶ Processor Status Word
- ▶ Program Counter
- ▶ Interrupt Enable Flags
- ▶ Stack Pointer
- ▶ ...

MPU settings

CPU mode

preemptable

Stack memory and its current content

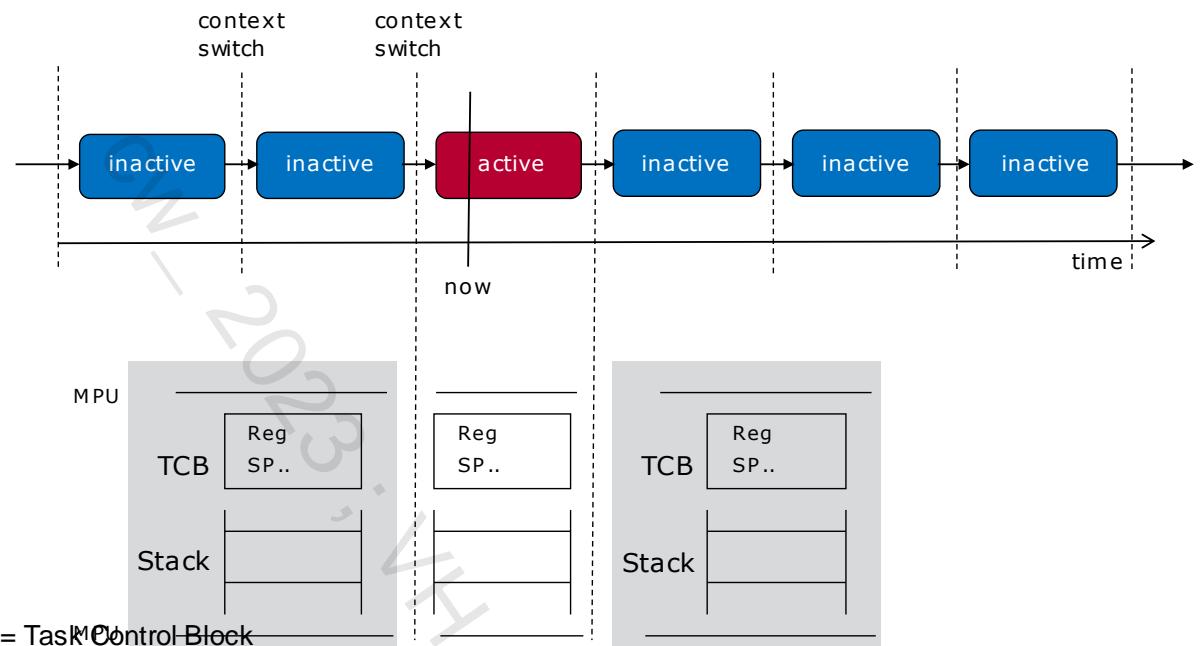
At a time, one processor core has at most one active context

Active context can only be changed by the allocated program

Active context is invariant



Text



Context switch

- ▶ A context switch occurs in several situations
 - ▶ Task switch
 - ▶ ISR entry
 - ▶ Call of OS services
 - ▶ Call of trusted functions
 - ▶ Hook routines

Agenda

Overview

Tasks

► **Alarms**

OS counter

Interrupts

IOCs

OS Resources

Illustration

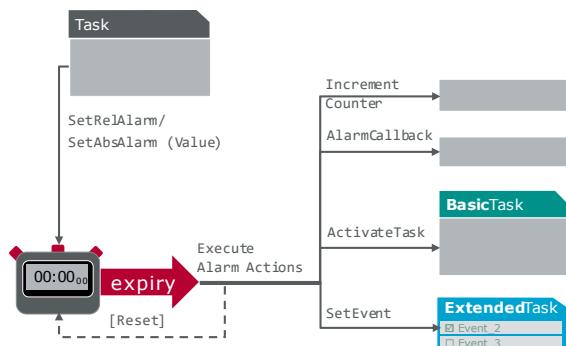
Scalability of the OS

Tool example

Coming up next

Alarms

- ▶ Alarms can be set to absolute or relative points of the system timer
- ▶ The time to which the alarms are set is an argument to the Alarm API
- ▶ You can
 - ▶ set alarms
 - > Dynamically over API
 - > Statically over configuration (AUTOSTART)
 - ▶ cancel alarms
- ▶ Once an alarm expires
 - ▶ An alarm action defines what happens
 - > Activate a task
 - > Set an event
 - > Call a callback function
 - > Increment an OS counter



If you set a series of alarms, the system timer may increase in the midst of setting the alarms of the series. This results in different starting times of the alarms. You can prevent this if

- ▶ a so called "High Resolution Timer" is used
- ▶ schedule tables are used

Agenda

Overview

Tasks

Alarms

► OS counter

Interrupts

IOCs

OS Resources

Illustration

Scalability of the OS

Tool example

Coming up next

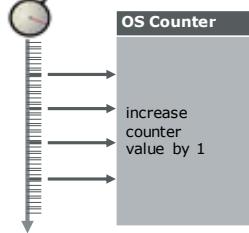
OS counter/ PIT or High resolution timer

OS counters are used to provide a time base "System Timer" for the operating system

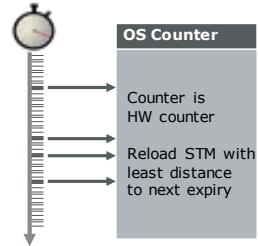
► Counter Driver (HARDWARE)

- PIT Periodical Interrupt Timer
 - > No HRT possible: OsDriverHighResolution = FALSE
 - > Equidistant periodic ticks
 - > Increment counter by a tick
 - > High precision results in a high interrupt frequency
- STM System Timer Module
 - > HRT supported: OsDriverHighResolution is TRUE: HRT supported
 - > FALSE: STM is working periodic like a PIT
 - > High precision is possible
 - > A periodic ticks
 - > Increment counter by last interval duration
 - > Program timer to next expiry
 - > High precision possible with low interrupt frequency

PIT (no HRT possible)

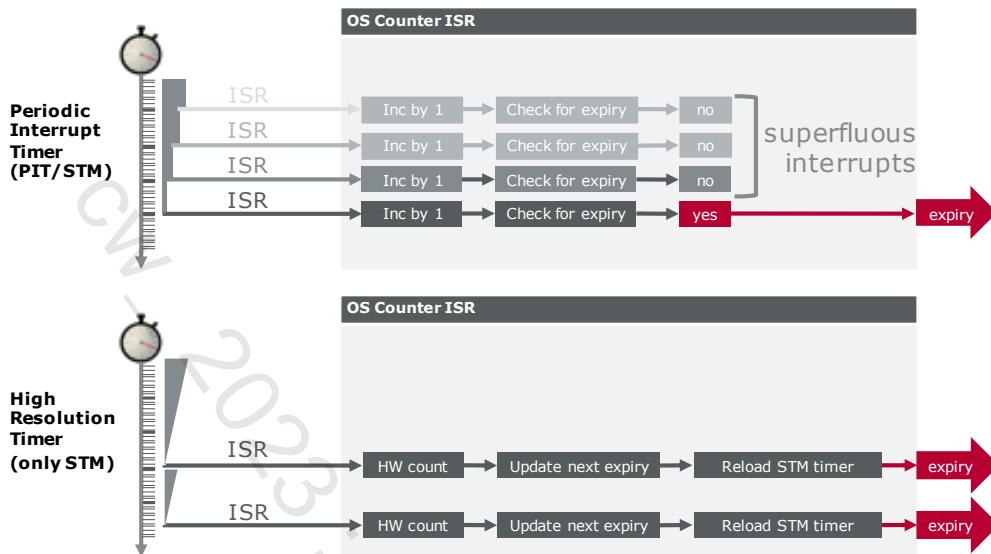


STM (used as HRT)



	PIT	HRT
IRQ	periodical	On demand
Precision (Alarms, schedule tables)	Multiples of OsSecondsPerTick	Any time the timer hardware can provide
Interrupt Load	Constant equally distributed	No equally distributed, bursts possible

Periodical behavior vs High Resolution Timer

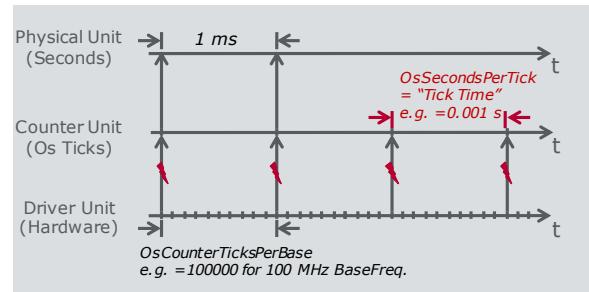


OS time base parameters

- ▶ Time base
 - ▶ $OsSecondsPerTick [s]$ resolution for SW counter increments
- ▶ Counter increments each $OsSecondsPerTick$
- ▶ The OS provides macros to convert between OS ticks and seconds
- ▶ Counter Type can be
 - ▶ HARDWARE
 - ▶ Have to select a driver (PIT/STM...)
 - ▶ Hardware advances OS counter over interrupts
 - ▶ SOFTWARE
 - ▶ Can use the IncrementCounter API to advance the System Time.
- ▶ PIT:
 - ▶ In every tick, the counter increments by 1
- ▶ HRT:
 - ▶ TicksPerBase = 1 → OS Counter increments by HW Base

Illustration of the OC counter configuration

$$OsCounterTicksPerBase = OsSecondsPerTick[s] \cdot BaseFreq[Hz]$$



$OsSecondsPerTick * OsCounterTicksPerBase = 1x$ Tick core code 8.

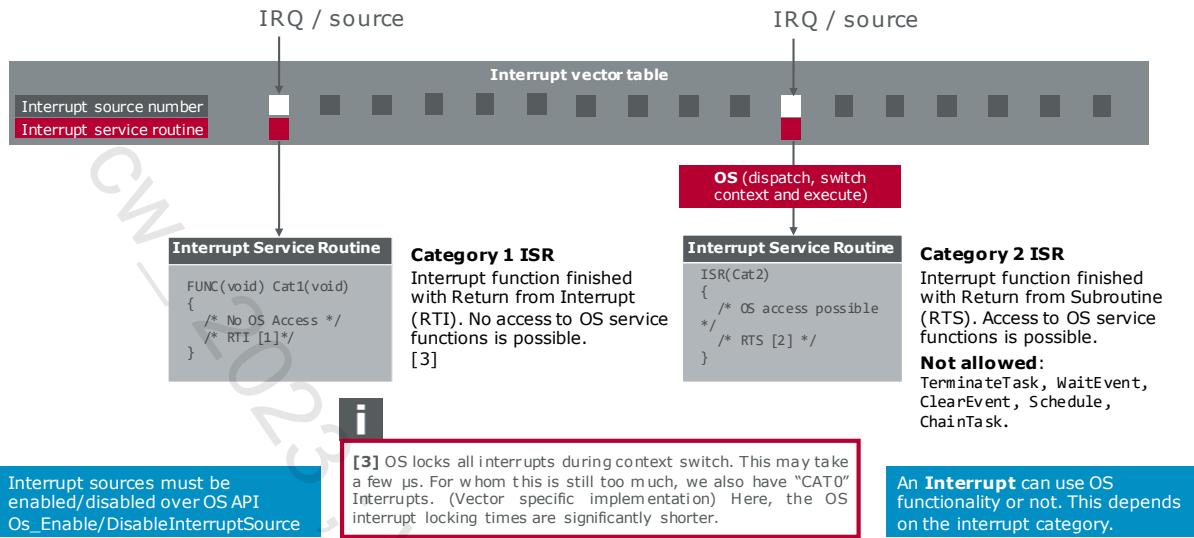
seconsperTick core code 9: hw ist jeder tick.... OsCounterTicksPerBase reload value:
 $SecondsPerTick/OsCounterTicksPerbase = HW\ base$

Timerauflösung → 0.001 ossecondsperTick → hw timer 100mhz : ticksperbase = 100000

Agenda

- Overview
- Tasks
- Alarms
- OS counter
- ▶ **Interrupts**
- IOCs
- OS Resources
- Illustration
- Scalability of the OS
- Tool example
- Coming up next

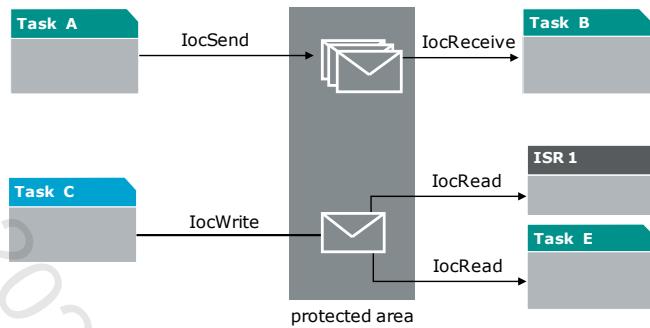
Interrupt categories



Impact of the Interrupt category on the ECU software

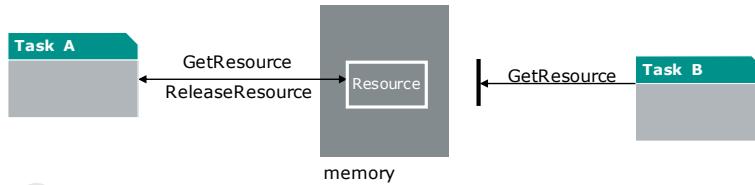
properties	CAT1 ISR	CAT2 ISR
Context	Runs in any context currently active	Runs in its own dedicated context
Stack usage	You'll need to reserve a safety margin of RAM for all CAT1 ISRs on all stacks of all tasks per core. Inefficient RAM usage.	You'll be able to decide on the amount of stack per ISR. Efficient RAM usage possible.
Memory protection	Stack overflow can lead to a protection trap inside all OS applications on the core. Memory corruption may remain undetected	Both stack overflow and memory corruption can lead to protection trap inside its OS application. Memory corruption attempts are detectable
Processor mode	Runs regardless of supervisor/user mode. Unintended access to some processor registers / peripherals possible.	Runs in the processor mode defined by the containing OS application. No way to affect the processor registers /peripherals in an unintended way.
impact		
Safety projects (mixed ASIL): requirements for ISRs	In safety scenarios with memory protection, all CAT1 ISRs must be implemented according to the highest ASIL in the system. Non ASIL ISRs must be qualified. If this is not done, memory corruption can happen and remain undetected.	CAT2 ISRs can be used for all ASILs/QM. Implementation according to the respective ASIL. Memory corruption can be detected.
How to decouple from ISR into TASK context	Can not be used for decoupling from ISR to TASK context using OS mechanisms. No task can be triggered. Only flags work. (DEFERRED operation)	Can be used for IMMEDIATE decoupling from ISR to TASK using OS mechanisms. Tasks can be triggered. That is the way the RTE works.

Inter OS Application Communicator (IOC)



An **IOC** can have several writers and several readers. Therefore IOCs support n:m data transfer across several OS applications.

Deadlock prevention using OS Resources



A OS **Resource** can be used to protect against concurrent access by at least two concurrent tasks without using Interrupt locks. Priority Ceiling Protocol.

Deadlock prevention using OS Resources

i

Resource management coordinates the access to shared resources:

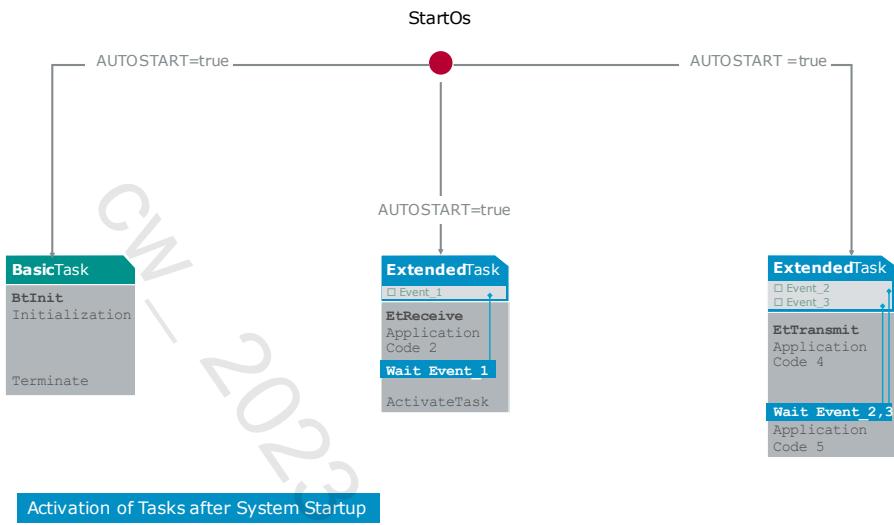
- ▶ memory (RAM)
- ▶ hardware components
- ▶ application program sequences (critical sections)
- ▶ elements of the operating system (Scheduler)

OSEK/VDX and AUTOSAR OS guarantee that

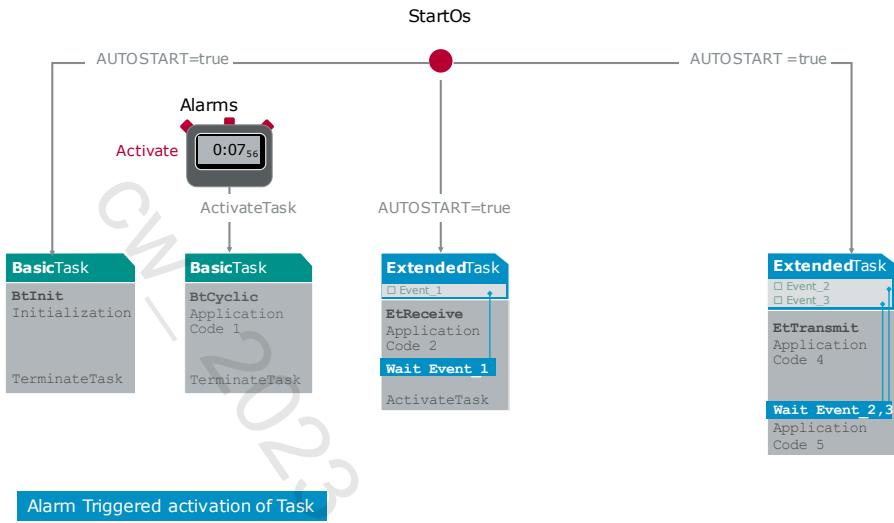
- ▶ a resource is never occupied by two users simultaneously
- ▶ priority inversion is excluded
- ▶ deadlocks are prevented
- ▶ access to a resource never ends in a wait state

Used for Exclusive Areas in AUTOSAR!

Interaction of OS Elements



Interaction of OS Elements



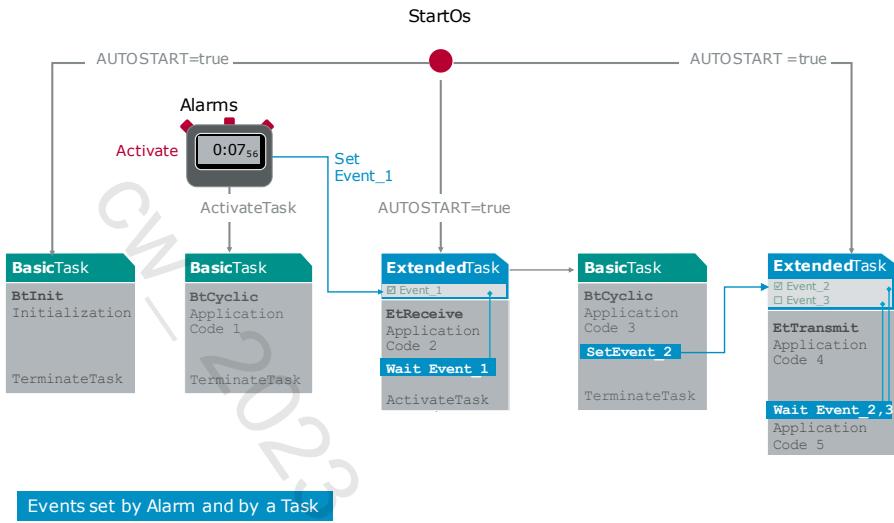
Alarm Triggered activation of Task



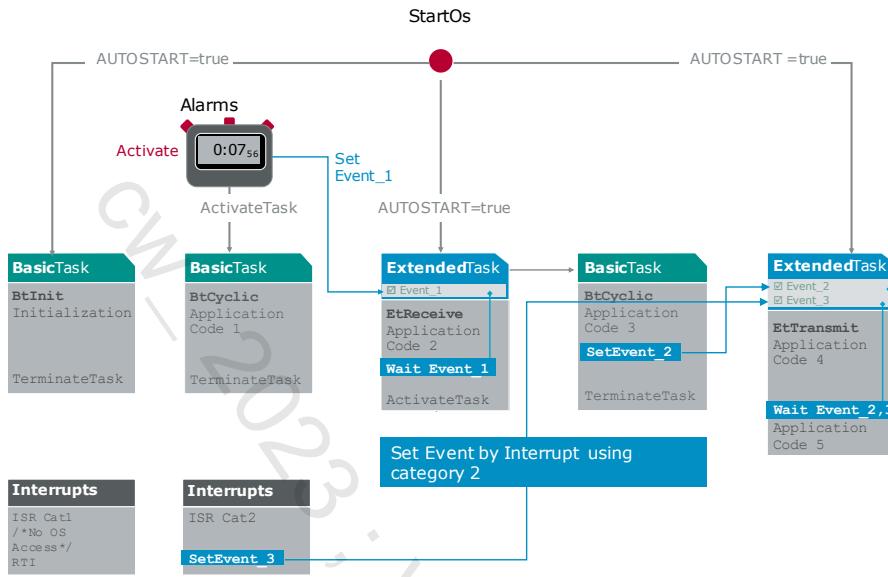
Alarm Activation

- An Alarm can be activated via:
- ▶ System start
 - ▶ User code

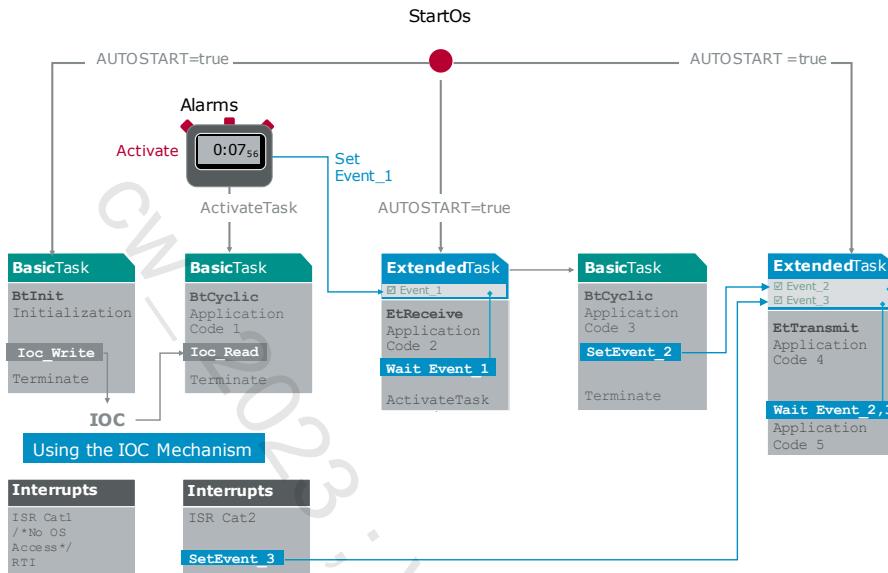
Interaction of OS Elements



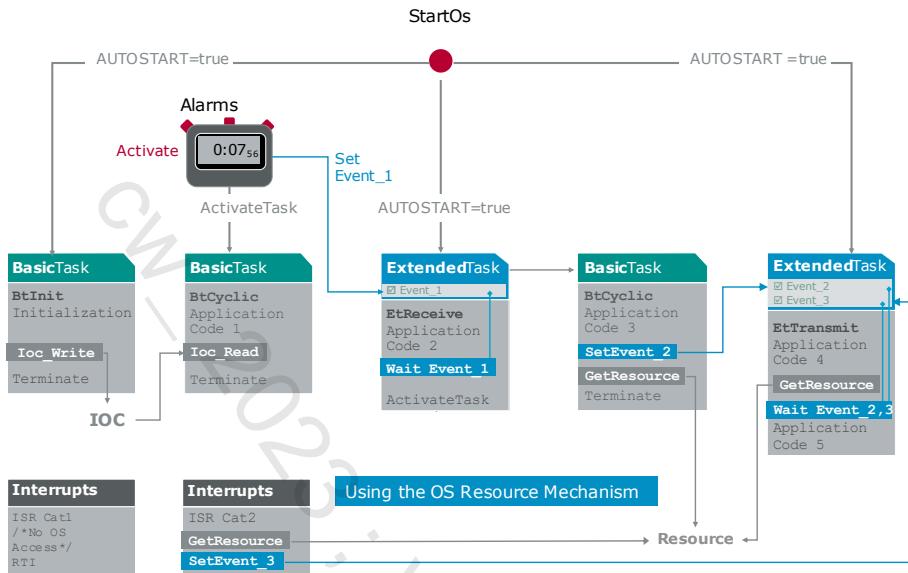
Interaction of OS Elements



Interaction of OS Elements



Interaction of OS Elements

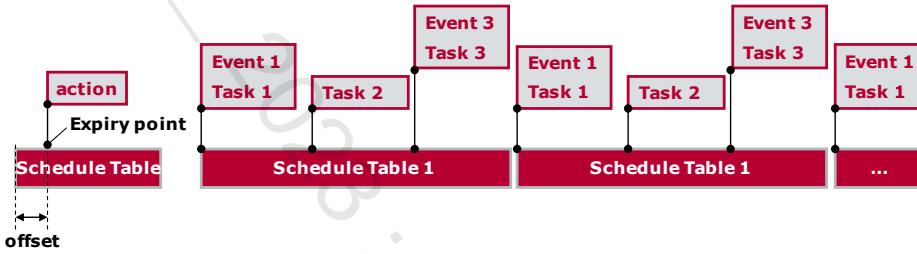


Agenda

- Overview
- Tasks
- Alarms
- OS counter
- Interrupts
- IOCs
- OS Resources
- Illustration
- ▶ **Scalability of the OS**
- Tool example
- Coming up next

SC1: Schedule Tables

- ▶ Definition of sequences of actions with a fixed time distance
- ▶ Can be executed **once** or **repeatedly**
- ▶ Can be autostarted or activated by the application
- ▶ Local task execution can be synchronized to a global time base
 - ▶ E.g., to the time base of a FlexRay bus (cycle start)
- ▶ Multiple schedule tables can be active at the same time



Each schedule has so called **Expiry Points** which occur within the **schedule table duration**.

Expiry Points are used to cause "**actions**" as soon as they occur.

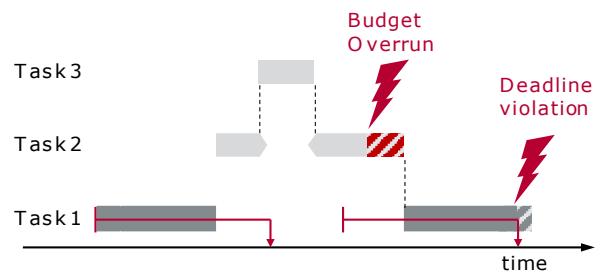
An action is either a **task activation** or a **SetEvent()** operation or both.

Expiry Points are positioned relative to the start of the schedule.

Each Expiry Point is signified within the schedule table by an individual "**offset**".

SC2: Timing Protection

- ▶ Execution budgets are assigned to tasks and monitored
- ▶ If the budget is exceeded the Protection hook is called
- ▶ Similarly, inter-arrival-times and resource locking times are monitored
- ▶ Timing Protection vs. Watchdog
 - ▶ Timing protection does not detect deadline violations!
 - ▶ Detects causes of deadline violations earlier than watchdog
 - ▶ Timing protection is limited to tasks / ISR 2 (ISRs of type 1 bypass the timing protection)



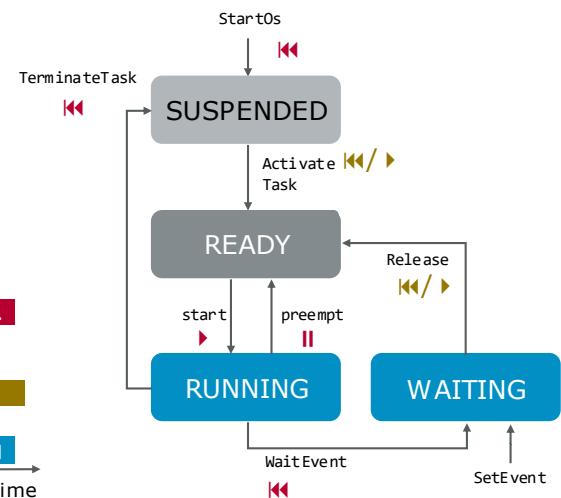
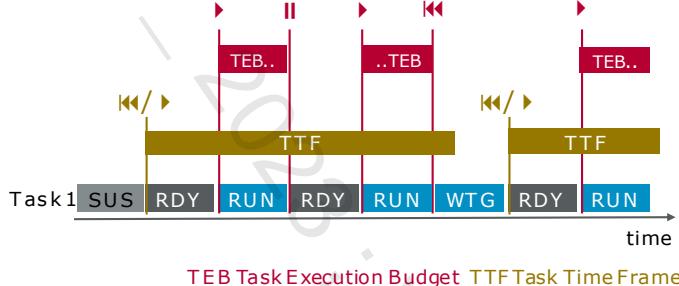
SC2: Timing protection (execution time / inter arrival time)

OsTaskExecutionBudget (TEB)

- ▶ maximum allowed execution time of the task
- ▶ To detect too long execution times

OsTaskTimeFrame (TTF)

- ▶ The minimum inter-arrival time between activations and/or releases of a task
- ▶ To detect too frequent activations



TEB Task Execution Budget TTF Task Time Frame

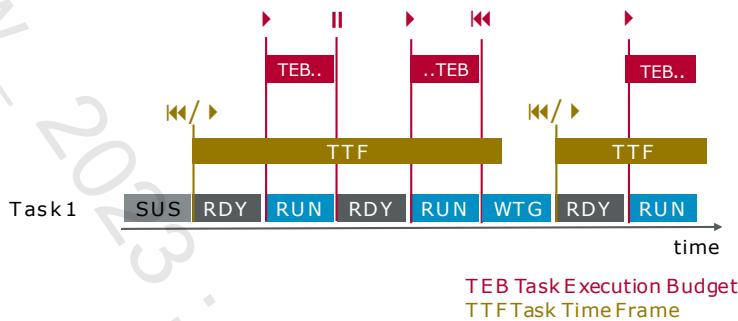
SC2: Timing protection (execution time / inter arrival time)

OsTaskExecutionBudget (TEB)

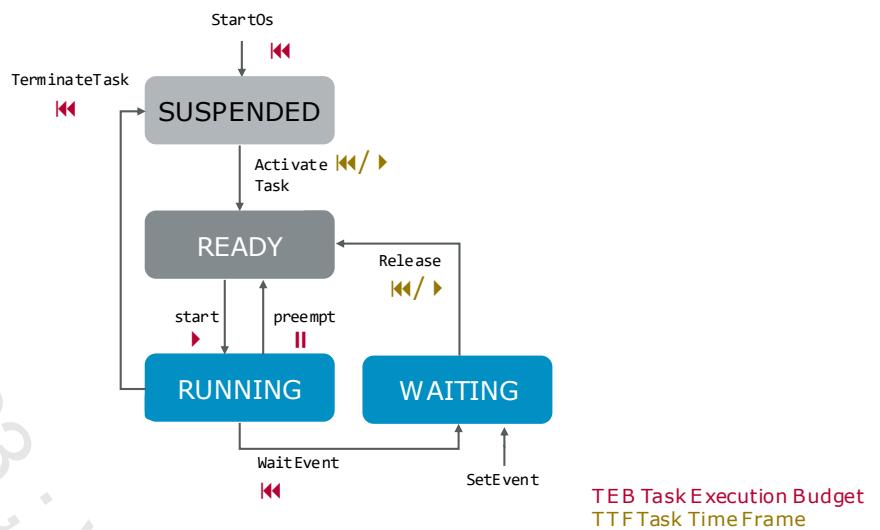
- ▶ maximum allowed execution time of the task
- ▶ To detect too long execution times

OsTaskTimeFrame (TTF)

- ▶ The minimum inter-arrival time between activations and/or releases of a task
- ▶ To detect too frequent activations



SC2: Task Execution Budget / Time Frame



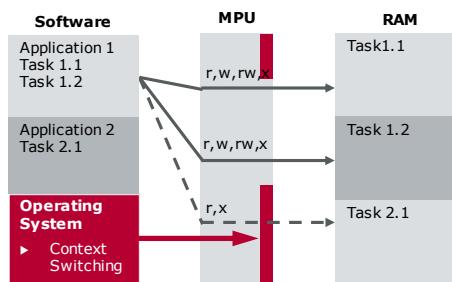
SC3: Memory Protection

Motivation and background

- ▶ Easy integration of applications
 - > Faulty code of one application has no influence on others
- ▶ Hardware support needed (MPU)
- ▶ Stack protection
 - ▶ Immediate detection of stack overflow
 - ▶ Constituent parts of an OS-Application
- ▶ Data protection
 - ▶ Private data sections of OS-Applications are shared by all its tasks and ISRs
- ▶ Code protection
 - ▶ Protect commonly used code (e.g. shared libraries) to prevent memory, timing or service violation.

SC3: OS-Applications

- ▶ Collection of cohesive functional units
 - ▶ Tasks, ISRs, alarms, hooks, etc.
- ▶ AUTOSAR OS shares systems resources between OS-Applications
- ▶ All objects belonging to the same OS-Application can access each other
- ▶ Access between OS-Applications may be granted in configuration
- ▶ Classes of OS-Applications
 - ▶ **Trusted**
 - > Allowed to run **without** protection
 - > Unrestricted access to memory / APIs
 - ▶ **Non-Trusted**
 - > Run only with protection
 - > Only limited access to memory / APIs



bility of the OS

Application

ferences and implements an *EcucPartition*

manages access to the µC resources

Runtime (CPU core)

Memory access

ollection of OS objects which shall use these
ources

Tasks

Interrupts

Alarms

Events

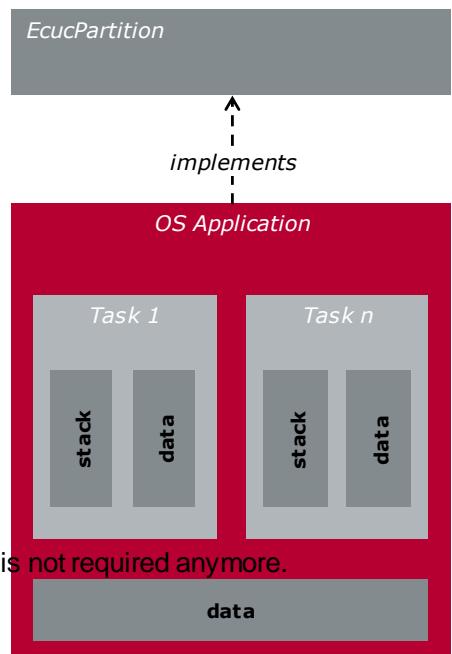
Hooks...

OS Application

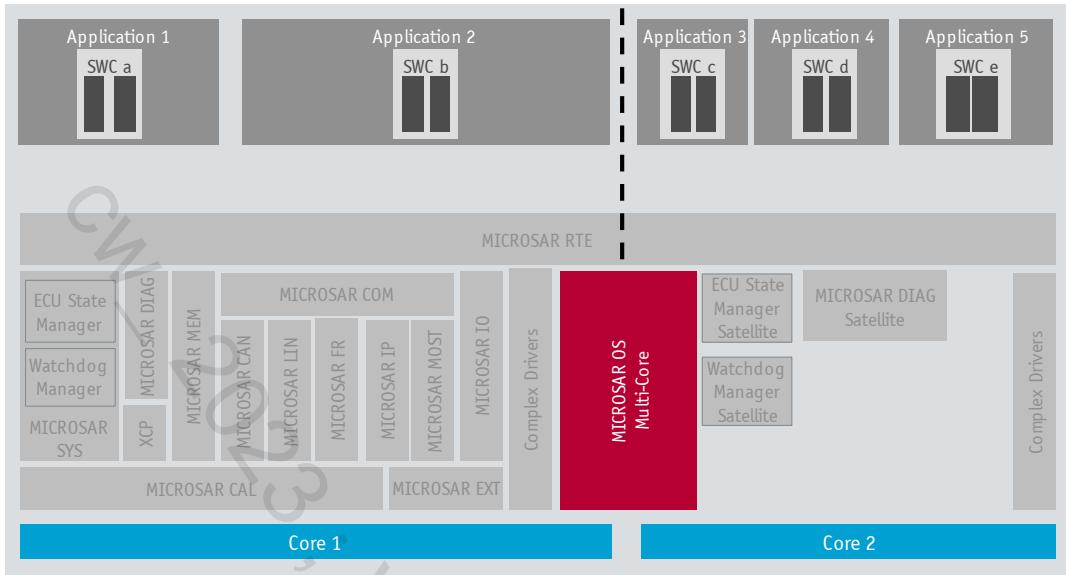
summarizes all elements having access to

each other

has ~~data regions that the OS partition has~~ task data region is not required anymore.



Multi-core operating system



Multi-core operating system



IOC: Inter-OS-Application Communicator

For CDDs (Complex Device Drivers) and Multi-Core:

- ▶ BSW resides on the Master core (Core0). If the CDD needs to access standardized interfaces of the BSW, it needs to reside on the same core.
- ▶ In case a CDD resides on a Slave core (Core1), it can use the normal port mechanisms to access AUTOSAR interfaces and standardized AUTOSAR interfaces. This involves the RTE, which uses the IOC mechanism of the operating system to transfer requests to the master core.
- ▶ Proxy solution: However, if the CDD needs to access standardized interfaces of the BSW and does not reside on the master core, a stub part of the CDD needs to be implemented on the master core, and communication needs to be managed locally in the CDD using the IOC mechanism of the operating system. Additionally, in this case the initialization part of the CDD also needs to reside in the stub part on the master core.

Agenda

- Overview
- Tasks
- Alarms
- OS counter
- Interrupts
- IOCs
- OS Resources
- Illustration
- Scalability of the OS
- ▶ **Tool example**
- Coming up next

OS in Configurator Pro

The screenshot shows the OS Configuration interface with a table of tasks. A context menu is open over the row for 'My_Task'. The menu items are:

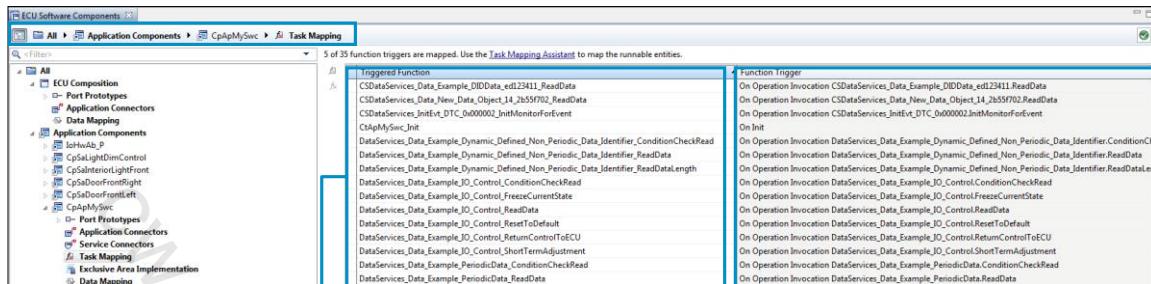
- Set task Priority
- Non-preemptive or fully-preemptive?
- Number of Activations (for Basic Task)
- Set Task Type

OsTasks	Task Prior.	Task Schedule	Task Stack Size [Byte]	Task Activation	Task Type
IdleTask_OsCore	4294967...	FULL	1024	*	AUTO
Init_Task	4	NON	1024	*	BASIC
My_Task	2	NON	1024	*	EXTENDED
SchM_Task	6	FULL	1024	*	EXTENDED

The RTE generator decides automatically how a task must be treated
(by the Function Trigger to Task mapping)

- ▶ Application task, BSW Scheduler task, non-RTE task

Runnables in DaVinci Configurator Pro



Runnables appear
as "Triggered
Functions"

Trigger
conditions

Please note the Access Point cannot be changed or viewed in Configurator. The Trigger also cannot be changed in Configurator. These are design aspects and have to be changed in Developer.

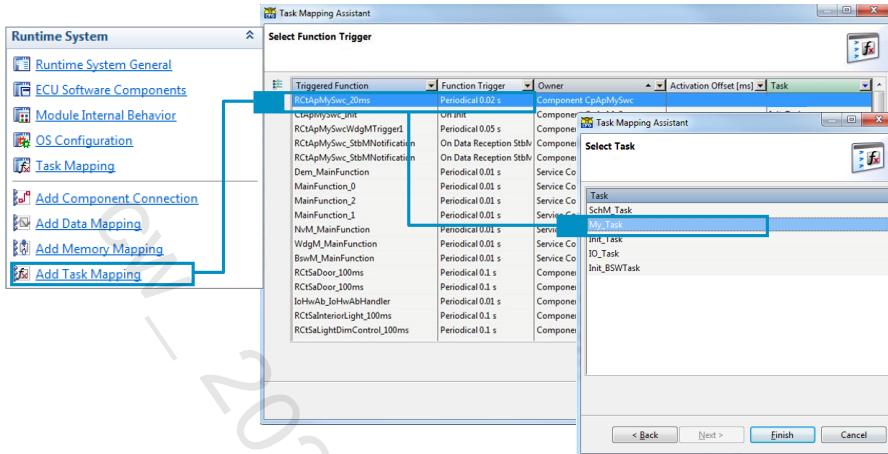
Task mapping using the ECU Software Components view

The screenshot shows the ECU Software Components view in a tool interface. On the left, a navigation tree includes Runtime System General, ECU Software Components (selected), Module Internal Behavior, OS Configuration, Task Mapping, Add Component Connection, Add Data Mapping, Add Memory Mapping, and Add Task Mapping. The main area displays a table of function triggers mapped to tasks. A context menu is open over a 'Service Component' node in the tree, and a 'Select Reference Target' dialog is overlaid. The dialog lists targets such as <NONE>, IdleTask_OsCore, Init_Task, IO_Task, My_Task, and SchM_Task. A callout box points to the 'My_Task' entry.

Triggered Function	Owner	Task	Position
RCApMySwcCode	Component CpApMySwc	My_Task	0
RCApMySwcInit	Component CpApMySwc	My_Task	2
RCApMySwcComm_ModeChange_FULL_COMM_Exit	Component CpApMySwc	My_Task	3
RCApMySwcComm_ModeChange_FULL_COMM_Entry	Component CpApMySwc	My_Task	4
RCApMySwcPostRunCode	Component CpApMySwc	My_Task	4

You can map Runnables to Tasks on a per-SWC approach.

Task mapping using the Task Mapping Assistant



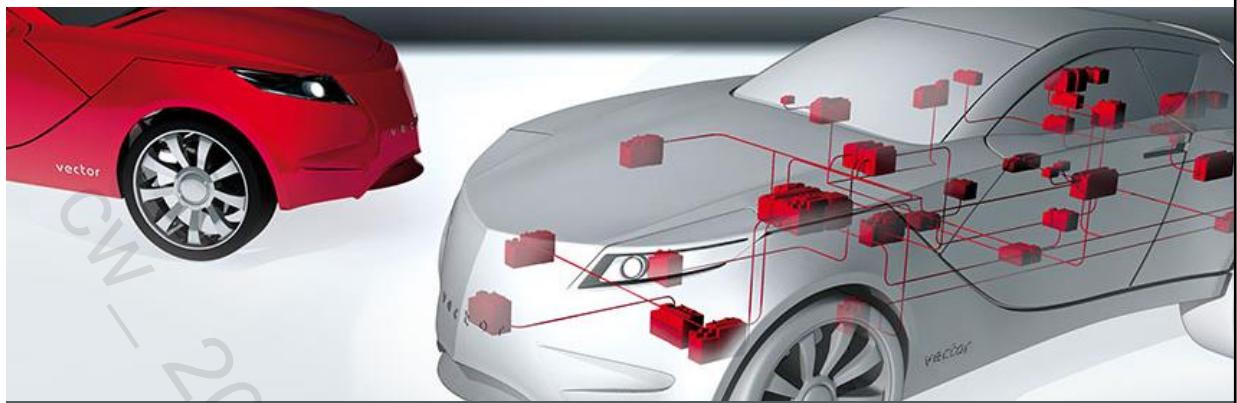
Columns in the Task Mapping Assistant have auto-filter functionality. This makes it easy to map Runnables globally instead for just one SWC.

For example, you can map several Runnables in one step to a specific Task. The Assistant will choose the Position in the Task automatically.

Operating System

Software Components

CW_2023_VH_Autosar CP Training_EN



AUTOSAR in Practice

Software Components

V1.0.0 | 2020-01-20

Agenda

► **Software Components**

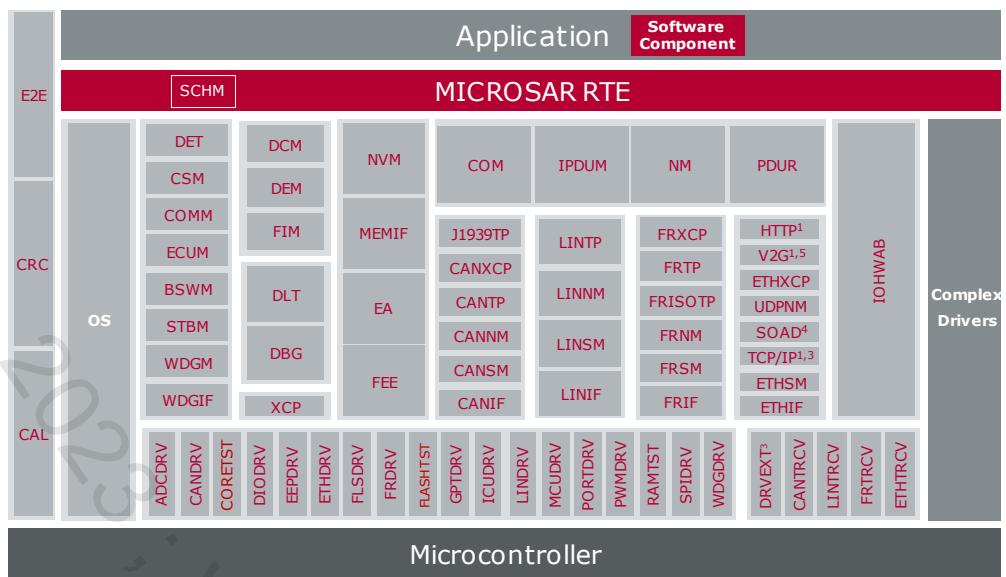
Packages

Data Types

Coming up next

Elements for Software Design

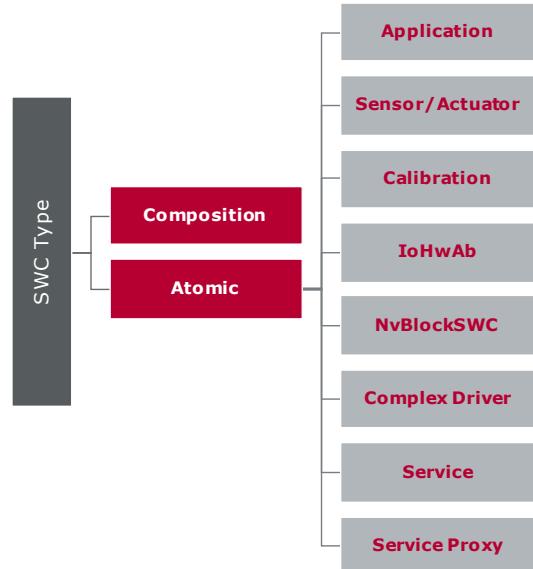
- ▶ Software Components (SWCs) require an RTE
- ▶ RTE runs the SWCs
- ▶ RTE runs the BSW modules over SCHM



Overview on the kinds of SWCs

Software Component Types (SWC)

- ▶ Composition
 - > has Port Prototypes
 - > Does not have any internal behavior
 - > Does not result in executable code
 - > Is placeholder, container for other SWCs
- ▶ Atomic
 - > Has Port Prototypes
 - > Internal behavior exists (Runnables, ...)
 - > Results in executable code
 - > Cannot be divided any further
- ▶ In our tools, we distinguish between
 - ▶ Application Component Types
 - > Constituting Parts of the application
 - > information exchange over AUTOSAR Interface
 - ▶ Service Component Types
 - > Representation of the service layer modules
 - > Services of the BSW be accessed by the application (Standardized AUTOSAR Interface)



Overview on the kinds of SWCs



Compositions can be considered only the hull without an own internal behavior, but with a defined interface. They serve as placeholders for further software components which could in turn be also compositions but in the end, there must be atomic software components.

Atomic software components yield a part of the executable code which exists in the binary. They form the leaves of the hierarchy tree of the software design.

In our tool, we distinguish the following specialized software components:

Application – hardware independent, fully reusable

Sensor/actuator – implementation technically hardware independent, but depending on the existence of a hardware sensor/actuator. Provides characteristics

Calibration – an AUTOSAR term for ROM constant Component which holds and distributes ROM values in the system (calibration parameter)

IoHwAb – IO Hardware abstraction. Resides in the ECU Abstraction layer and abstracts from everything outside the µC on the PCB.

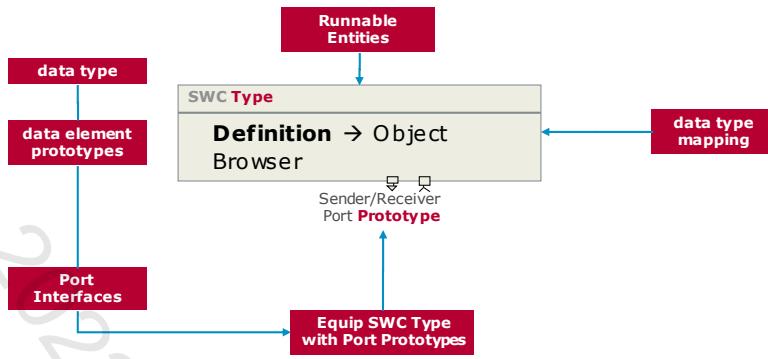
NVBlockSWC – aggregation of NVRAM data prior to writing it down in a NVRAM Block. Distribution of Nv data elements

Complex Driver – a SWC which contains hardware dependent parts of implementation, and also direct Standard Interface calls into the BSW.

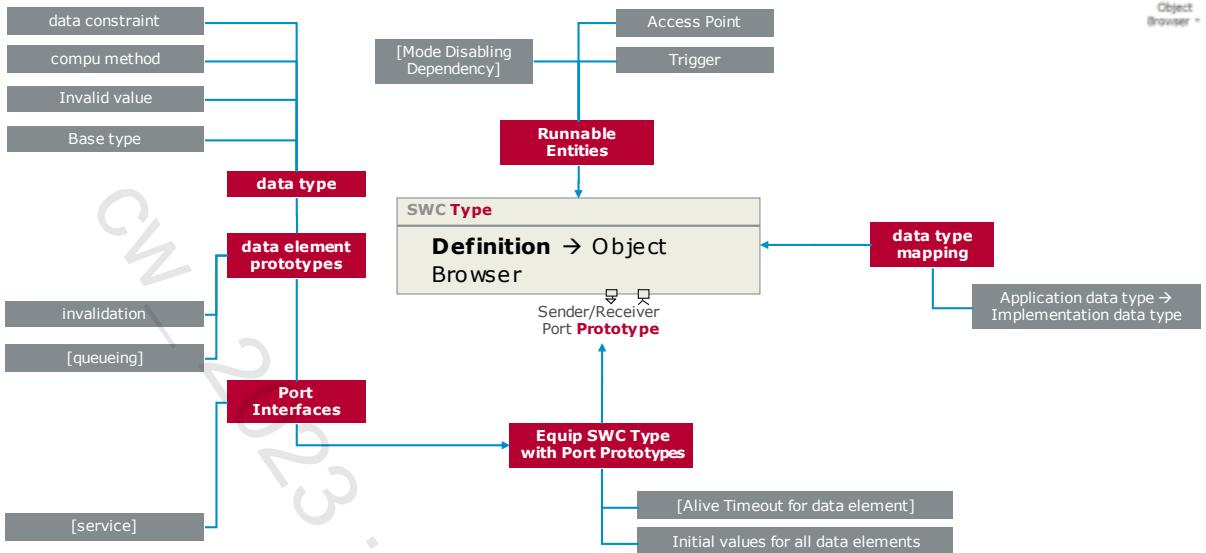
Service – a service SWC which is the representation of a member of the Service Layer in the BSW. A service module offers Standardized ATUTOSAR Interfaces against which the SWCs can connect.

Service Proxy – a stand-in for a basic software service module for multicore scenarios

Definition of an SWC and its elements



Definition of an SWC and its elements



Definition of a new SWC Type in DaVinci Developer

1. Right Click Application Component Types

2. New Component Type comes up

3. Give SWC Type Name

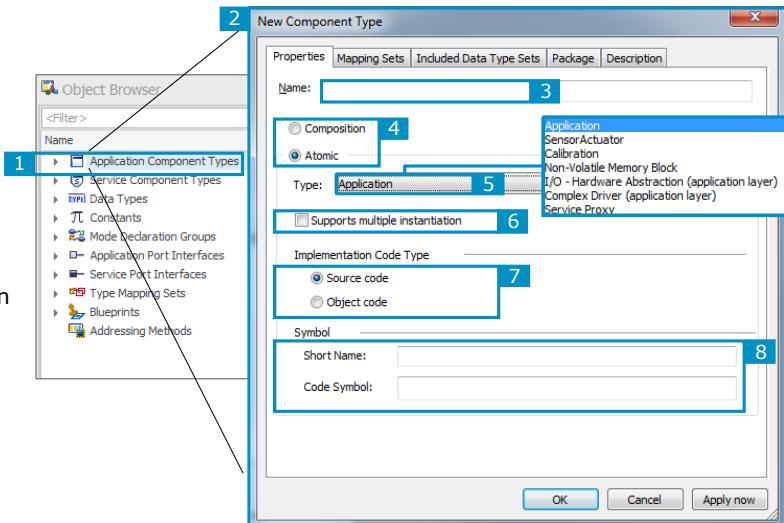
4. Composition or Atomic

5. Type of atomic SWC annotation for intended use of SWC Type

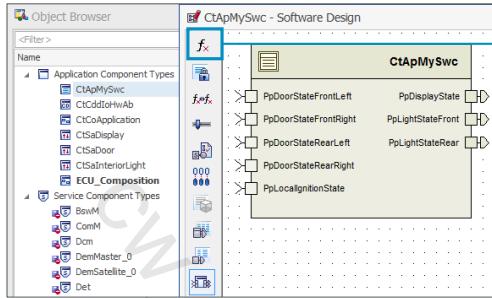
6. Multiple instantiation

7. Source or library implementation

8. Symbol: to be used by the RTE generator instead of the name.
Name: must be identical to Value



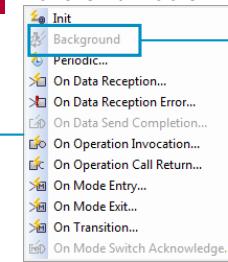
Define Runnable Entities inside an SWC Type



Definition of Runnables

Please take care the Access Point is defined for all Runnables where it is needed (and take care you do not perform the access on C code level without defining it in Developer!)

Definition of a trigger for the execution of the Runnable



Background Trigger:
Execute Runnable in a background task

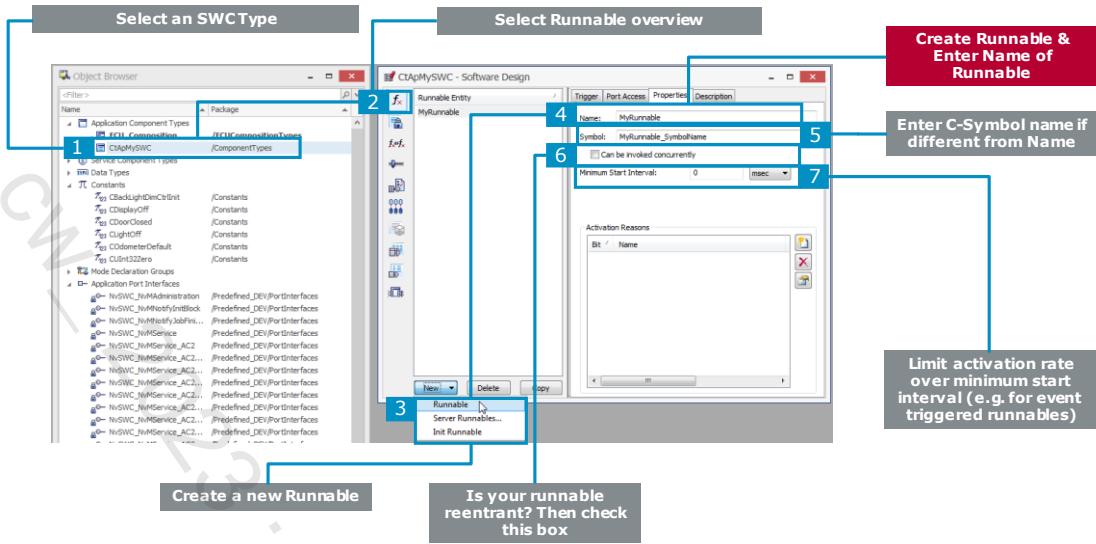
[Mode Disabling Dependency]

Define Access Point in case the Runnable has to use a Port Prototype of the SWC Type

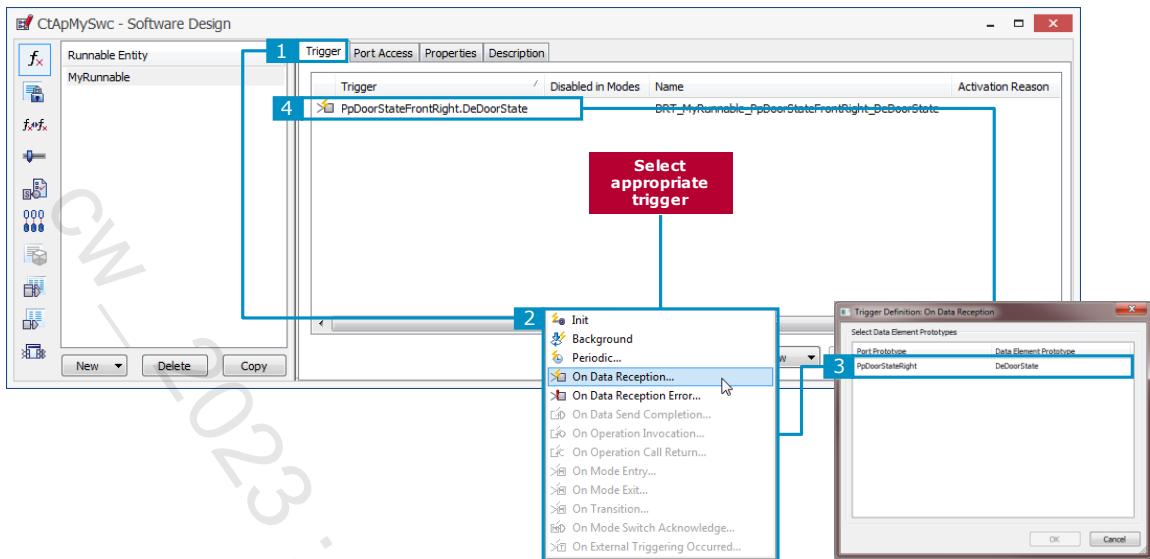
- >□ Receive Data (queued)...
- >□ Read Data (non-queued)...
- Send Data (queued)...
- Write Data (non-queued)...
- >□ Invoke Operations...
- Send Mode Switches...
- >□ Read Received Mode...
- Read Sent Mode...
- Raise External Trigger...

AUTOSAR 4 defines also "Triggered Events". This is yet to come in Vector's software.

Defining Runnable Entities ("Runnables") in DaVinci Developer



Defining Trigger setting for a Runnable in DaVinci Developer



Defining Trigger setting for a Runnable in DaVinci Developer



Triggers

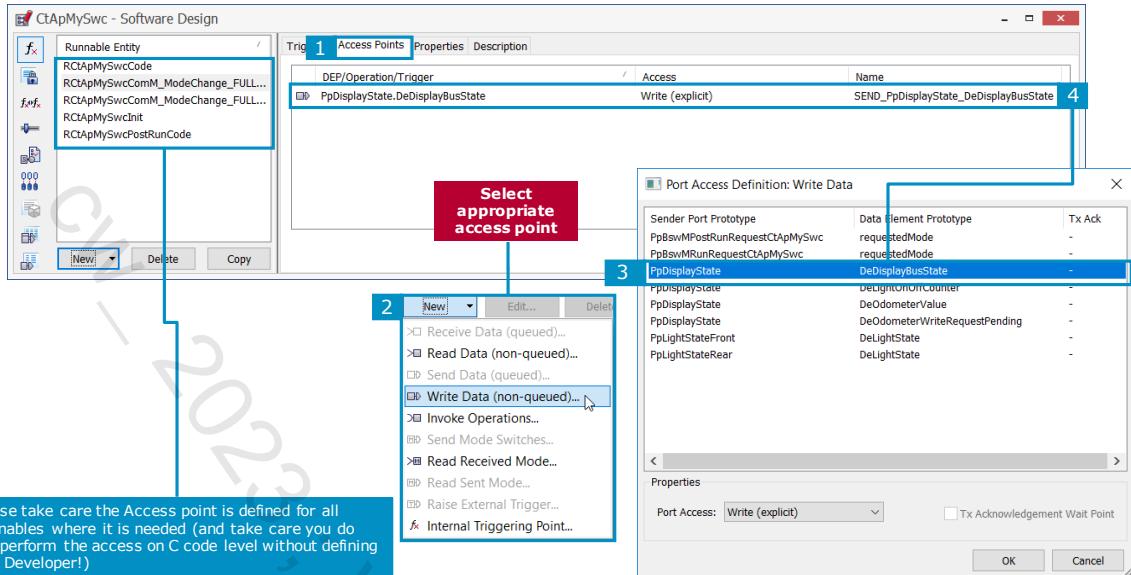
Runnables need triggers. A trigger can be

- ▶ **Periodical** – this is caused by an event which occurs cyclically; here an OS Alarm is configured and a respective Event is set if the Alarm expires.
- ▶ **On Data Reception** – means new data information has been received over a communication bus or from another Software Component
- ▶ **On Data Reception Error** – an error occurred during the reception of data
- ▶ **On Data Send Completion** – the runnable entity is triggered upon completion of a transmission of an output data element (Tx Acknowledge).
- ▶ **On Operation Invocation** – this is needed if you implement server functionality in your SWC and a client component calls a service of your SWC. The runnable which has to provide this service is triggered by the RTE.
- ▶ **On Operation Call Return** – an asynchronous operation has been started and as soon as this operation is finished, the runnable will be called.
- ▶ **On Mode Entry** – Event which occurs when the Mode Manager enters a certain mode (transition)
- ▶ **On Mode Exit** – Event which occurs when the Mode Manager leaves a certain mode (transition)
- ▶ **On Mode Switch Acknowledge** - the runnable entity is triggered upon acknowledgement of a mode switch of a mode P-port prototype

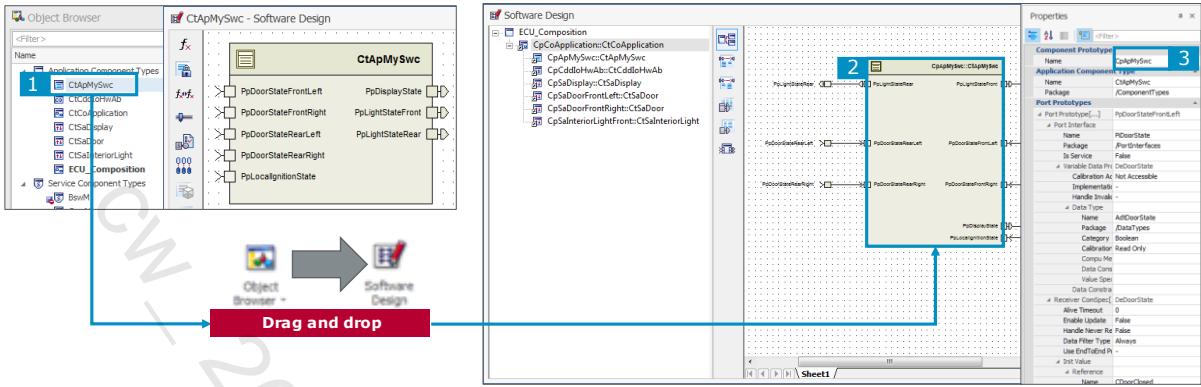
Depending on the number of triggers and internal communication (e.g. queued S/R) of runnables which are mapped to a certain task, the RTE generator makes a decision to create

1. a **Basic Task** or
2. an **Extended task** with wait points, events, and alarms.

Defining Access Point for a Runnable in DaVinci Developer

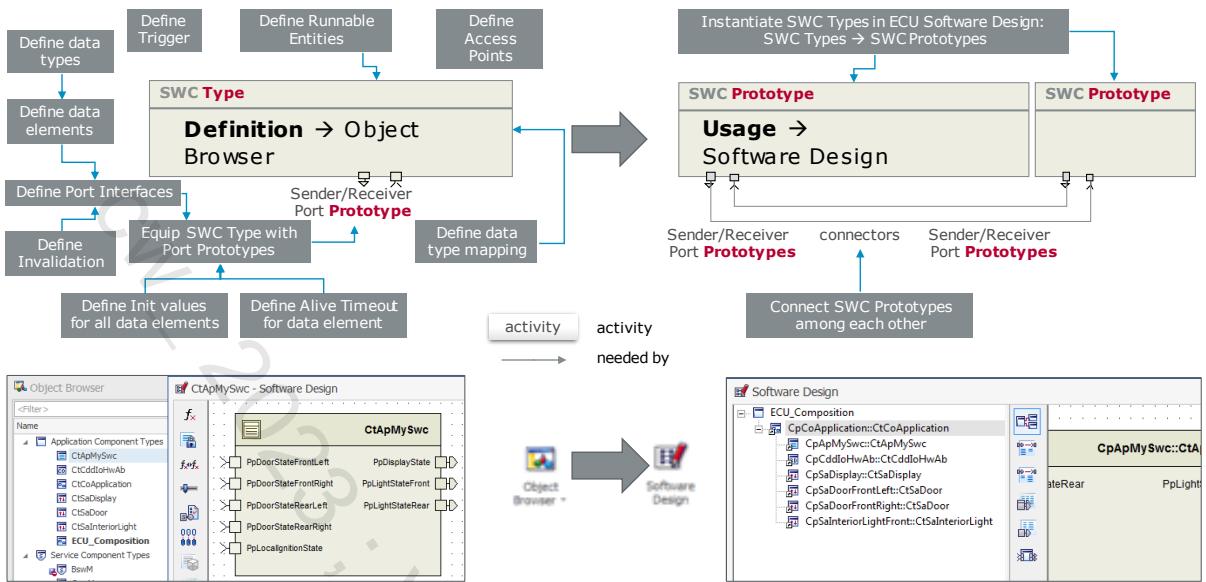


Usage of an SWC Type in the Software Design



1. Dragging and dropping the SWC Type from the Object Browser into the Software Design ...
2. ... instantiates the SWC Type as SWC Prototype
3. You can edit the properties like the prototype name by selecting the SWC. The properties view will update accordingly

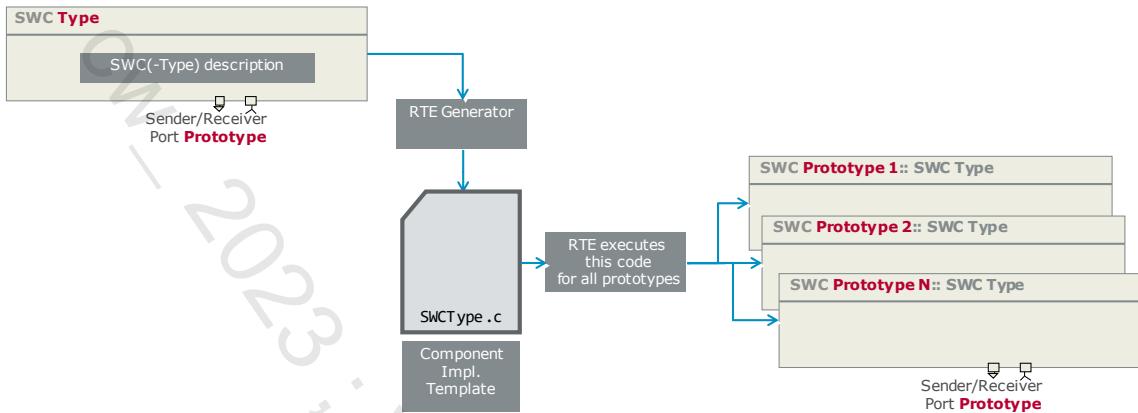
Definition and usage of SWCs and their elements



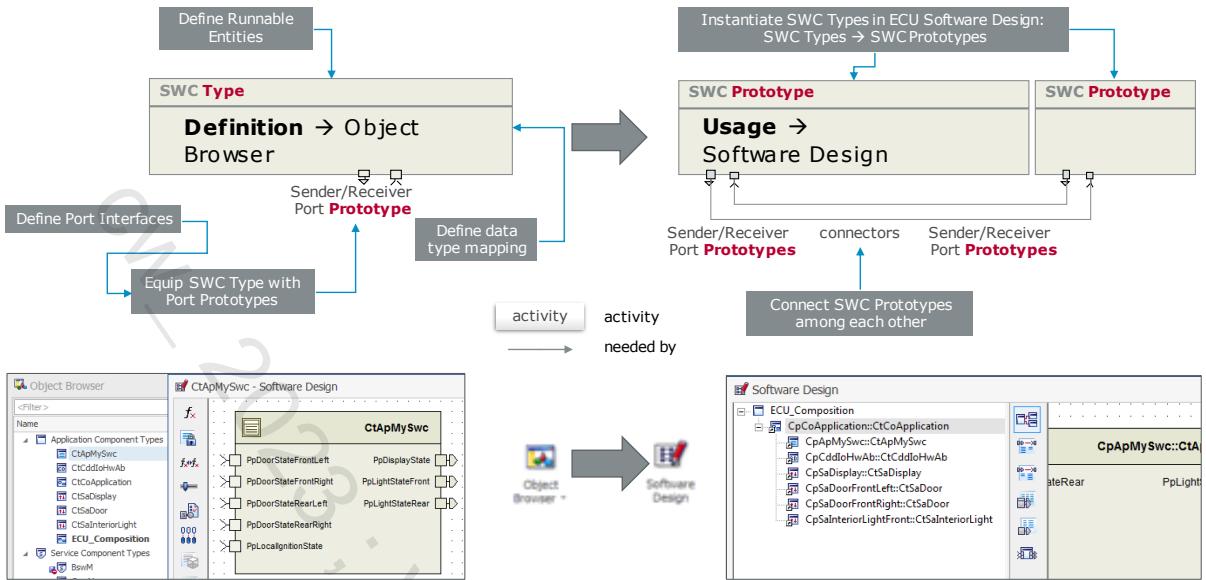
Generation of the implementation template

The implementation (generated SWC Template) is applicable to the Component **Type**

- ▶ Implementation is unique for one Component Type and not for the instantiated SWCs (SWC Prototypes)
- ▶ Component implementation template can be generated

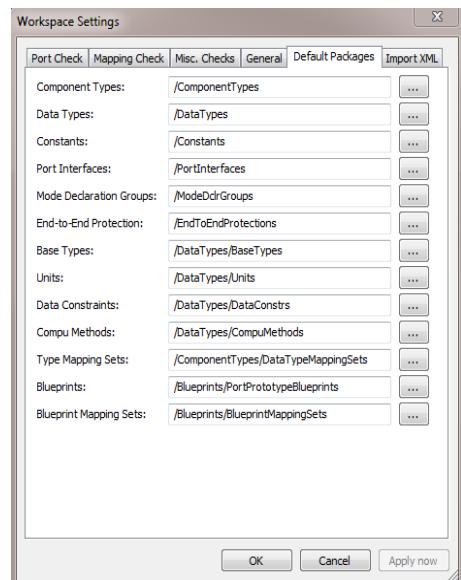


Designing and using Software Components



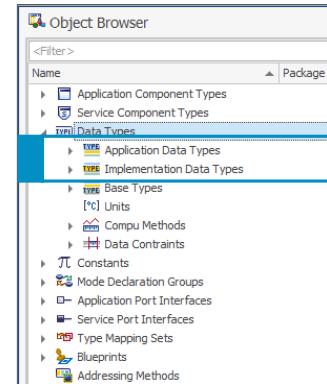
Packages in DaVinci Developer

- ▶ Default Packages Page
- ▶ Select the default package for new objects for all types of library objects
- ▶ This is used when
 - ▶ creating new objects in the Types view of the Object Browser
 - ▶ creating objects automatically, e.g., during data mapping via the "Create Port Prototypes" function
 - ▶ selecting the option "Use Default Packages" during import of an .arxml file



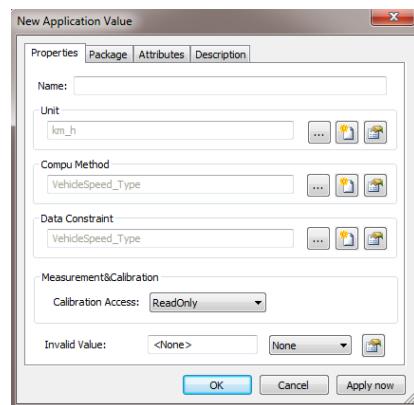
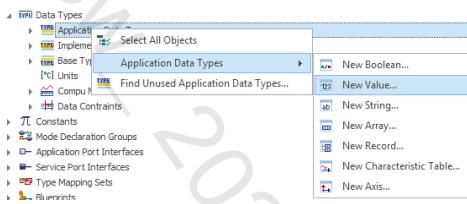
Data Types

- ▶ AUTOSAR distinguishes between Application Data Types (ADT) and Implementation Data Types (IDT)
- ▶ Application Data Types (ADTs) exist on conceptual level
 - ▶ Are abstract data types
 - ▶ Conceptual/algorithmic view
 - ▶ For function developers
 - ▶ Optional usage
- ▶ Implementation Data Types (IDTs) exist on code level
 - ▶ concrete data type
 - ▶ For implementers
 - ▶ Mandatory for code generation



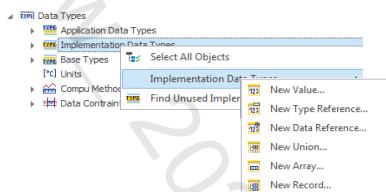
Application Data Types

- ▶ abstract data type, conceptual/algorithmic view
- ▶ Can be used in Port Interfaces
- ▶ for RTE generation mapping onto Implementation Data Type (IDT) required

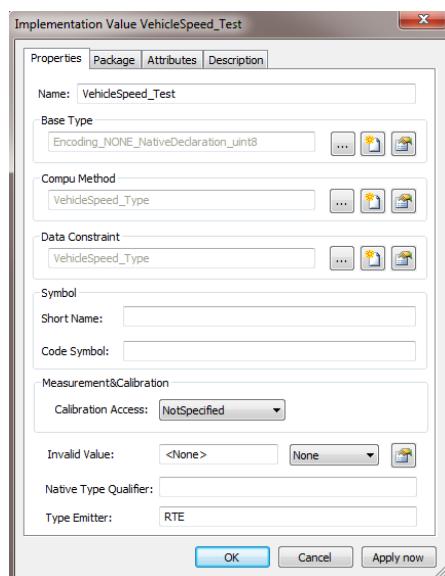


Implementation Data Types

- ▶ include semantics, implementation view
- ▶ Can be used in Port Interfaces
- ▶ IDT references Base Type (if you choose IDT name identical to name of platform type then reference can be omitted)

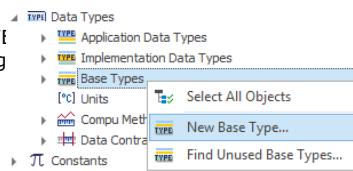


- Could be a complex data type which consists of base types / implementation types
- OR directly references a Base Type / platform type

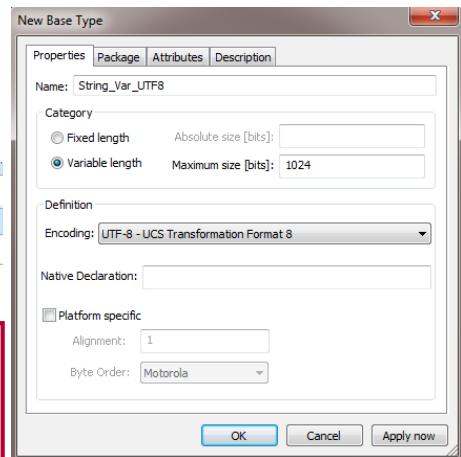


Base Types

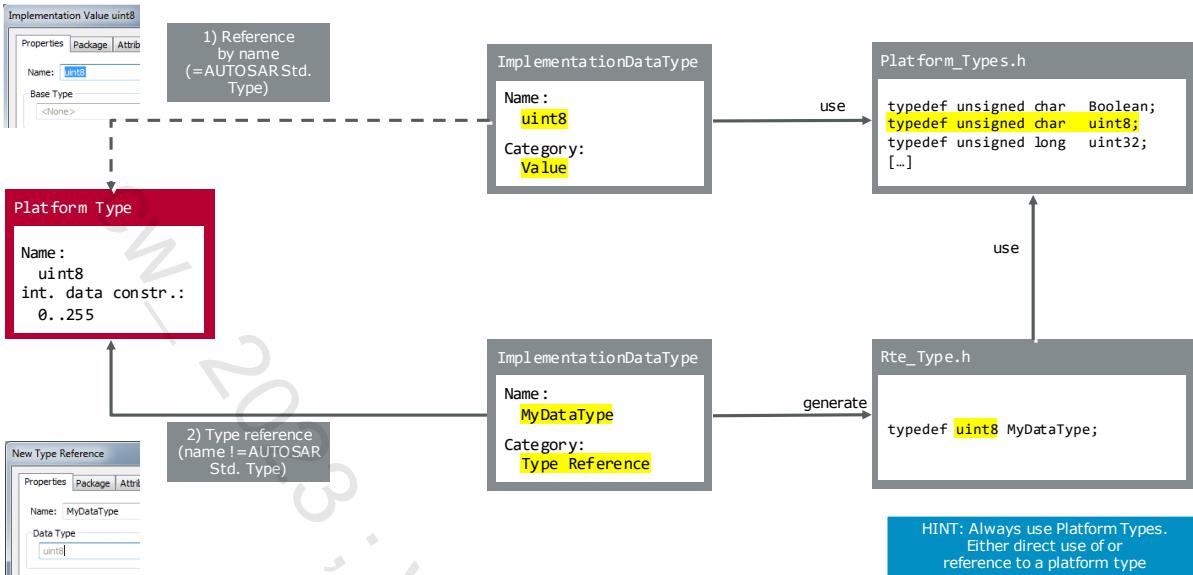
- ▶ Base Type
 - ▶ code-level type
 - ▶ no semantics
- ▶ Allows platform-specific configuration
 - ▶ Not required if implementation type is a known platform type
- ▶ Native Declaration
 - ▶ can be used to influence the RTI Type definition and correct us as user



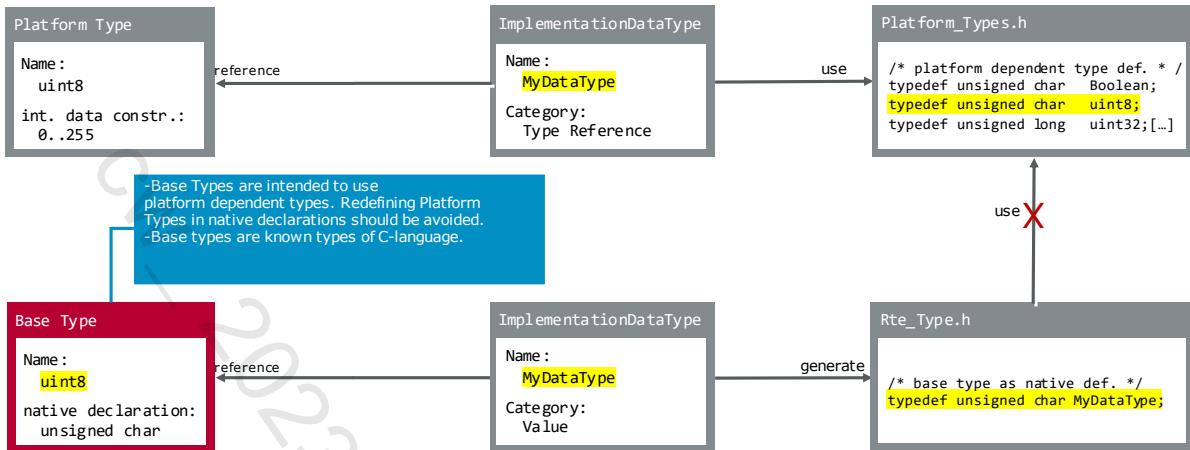
- Platform type corresponds to a Base Type specified for my platform
 - For example, if only 16-bit minimum access is possible, then we have an uint8 platform type which is implemented as uint16
- Platform Types abstract the platform specifics
- Base Types are the well-known C language types



Platform Types create platform independent software

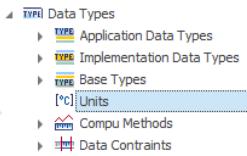


Base Types bypass the Platform Type concept



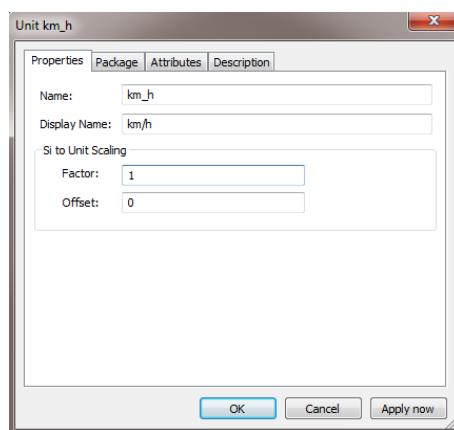
Units

- Units are globally defined
- Both Application Data Types and Compu Methods reference units



Only defined with no real influence on the code

- can be assigned to an Application Data type
- Defines how the Application types shall be interpreted
- Computation methods as well as units define how a signal conversion can be done from logic interpretation to physical unit



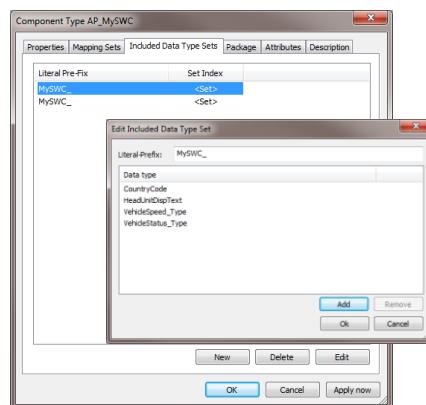
Included Data Type Sets

- ▶ Specifies the data types used within the implementation of an SWC (i.e., not used by a port)
- ▶ N lists of data types can be defined, each with a prefix



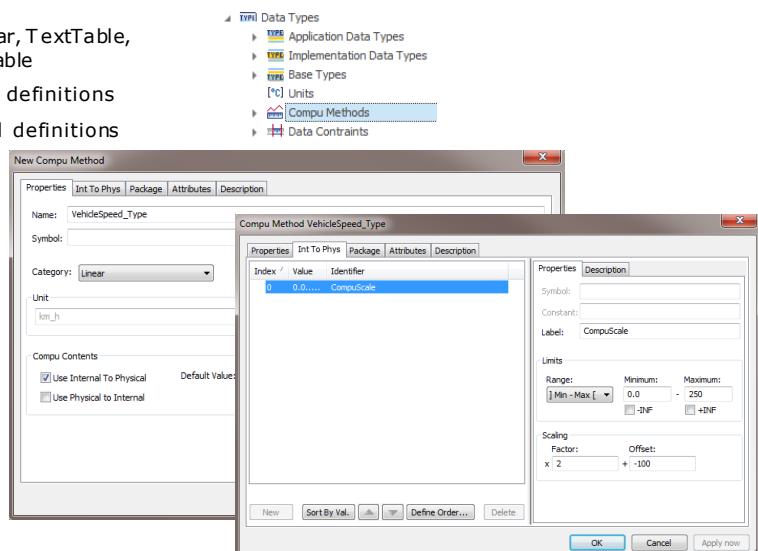
Grouping together which datatypes are used within the SWC

- Datatypes can be equipped with a prefix so they will be specific to the SWC
- Types will be generated/ defined by the RTE in the component header file



Compu Methods

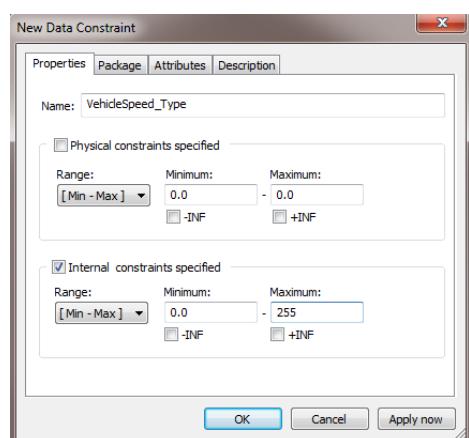
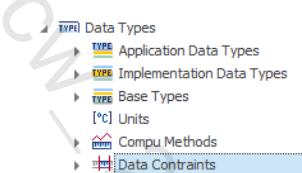
- Compu Methods are globally defined
- Categories: Identical, Linear, Scale-Linear, TextTable, Scale-LinearAndTextTable, BitfieldTextTable
- IDTs typically use **Internal-To-Physical** definitions
- ADTs typically use **Physical-To-Internal** definitions

**i**

Defines how the conversion from a physical to internal signal interpretation can be done, and vice versa

Data Constraints

- ▶ Data Constraints are globally defined
- ▶ Data Constraints apply to an Internal or Physical value
- ▶ ADTs and IDTs reference the data constraints

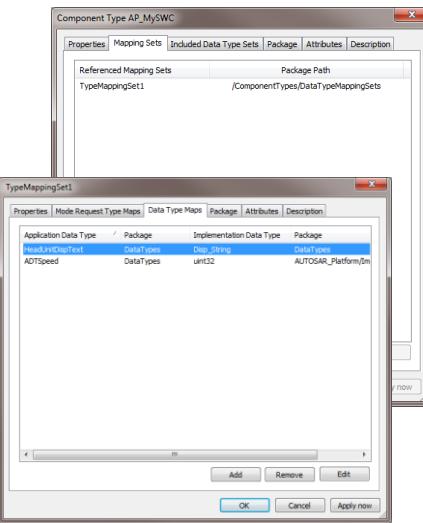
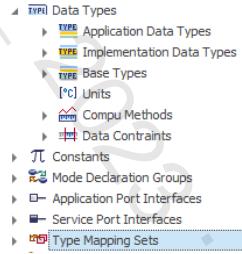


i

Defines Valid ranges for my Data Types
-> Replaces the invalid value from A SR3

Type Mapping Set

- ▶ Global definition of type mapping sets
 - ▶ Application Data Type → Implementation Data Type
 - ▶ Mode Declaration Group → Implementation Data Type
- ▶ Usage of type mapping set is in the context of an SWC
- ▶ Automatic Mapping function available (see next slide)



Type Mapping Set

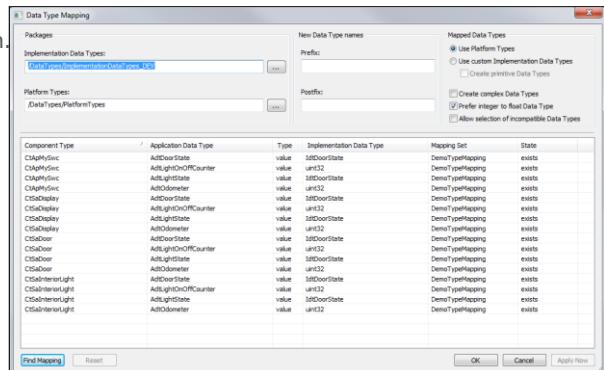


The data type mappings can be derived automatically. Corresponding implementation data types will be selected or created.

ADT to IDT mapping is required for RTE generation.



Click inside the "Home" tab on "Data Type Mapping"



The Data Type Mapping sets can automatically be created. This includes the resolution of which application data types map to which implementation data types. The implementation data types can also be created automatically.

Example

ApplicationDataType

- ▶ Data constraint
 - ▶ [-50°C; +150°C]
 - ▶ In 0,5 °C steps
- ▶ Number of values required
 - ▶ 2 steps for 1 °C
 - ▶ $2 * (150 - (-50)) + 1 = 401$ values required

Problem 1:

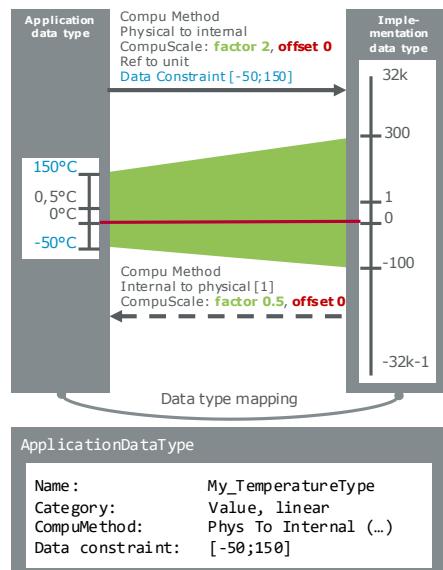
Make a choice for an **ImplementationDataType**

- ▶ uint8: 2^8 values = 256 values
→ too small
- ▶ uint16: 2^{16} values = 65536 values → choice

Problem 2:

How could a 16 bit type represent the temperature range?

- ▶ Signed value range, 401 values
 - ▶ Take sint16
- ▶ But: Scaling rule required
 - ▶ Provided over the CompuMethod → Physical To Internal → CompuScale
 - ▶ Mapping to ImplementationDataType



Example

i

[1] The physical to internal conversion is also used for internal to physical conversion in case the defined function is invertible. It is the case in this example, so it is enough to specify only the physical to internal conversion.

The linear conversion factor and offset are used in the following way: With X_i representing the internal value and X_p the physical one.

- Physical to internal: factor F_p and offset O_p :

$$X_i = F_p \cdot (X_p + O_p)$$

- Internal to physical: factor F_i and offset O_i :

$$X_p = F_i \cdot X_i + O_i$$

What becomes generated from which element?

Implementation Data Types

- ▶ Are the basis for the generation of "typedefs" on C-level (Rte_Type.h)
 - ▶ using existing platform types (Platform_Types.h)
 - ▶ defining own base types (Rte_Type.h)

Application Data Types

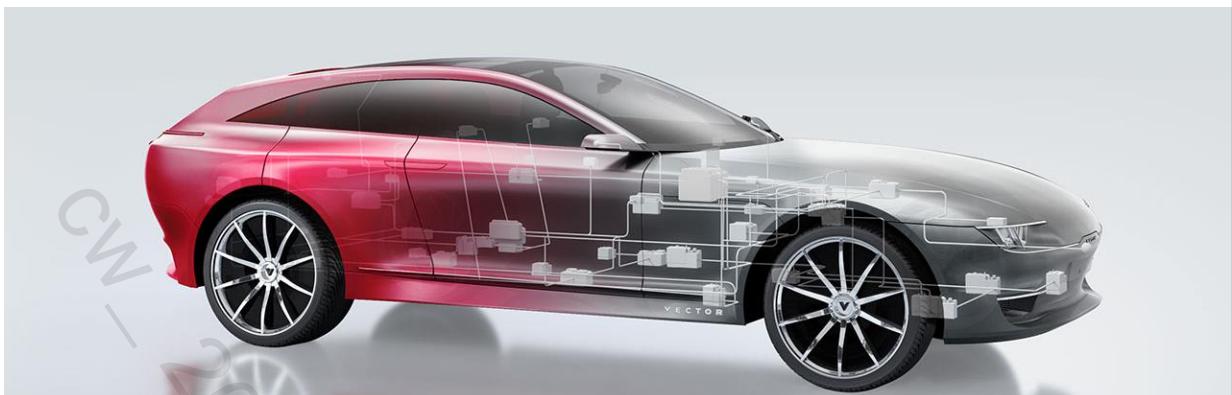
Are the basis for the generation of

- ▶ Init values
 - ▶ Calculated from
 - > Init value in the computation method settings
- ▶ Lower and upper range limits
 - ▶ Preprocessor defines for access in the SWCs
 - ▶ Calculated from
 - > constraint range, computation method
- ▶ Invalid value handling
- ▶ Enumerations
 - ▶ If defined computation method is a "text table"

Software Components

Exercise: SWCs

CW_2023_VH_Autosar_CP_Training_EN



AUTOSAR in Practice

Exercise 1: Software Components

V28 | 2023-03-28

Agenda

► **Introduction to Exercises**

Training Example

Exercise 1 - Software Components

Coming up next



1st Exercise – Introduction to Exercises

An overview of the exercise is given first to explain what has to be done.

The detailed, step-by-step description is on the following slides.

X**1st Exercise – Introduction to Exercises**

- 1** Do this...
- 2** Do that...
- 3** And then...
- 4** Test it

The steps for Exercise X are listed to give an overview of what has to be done.

[Detailed exercise description >](#)

Part 2/4

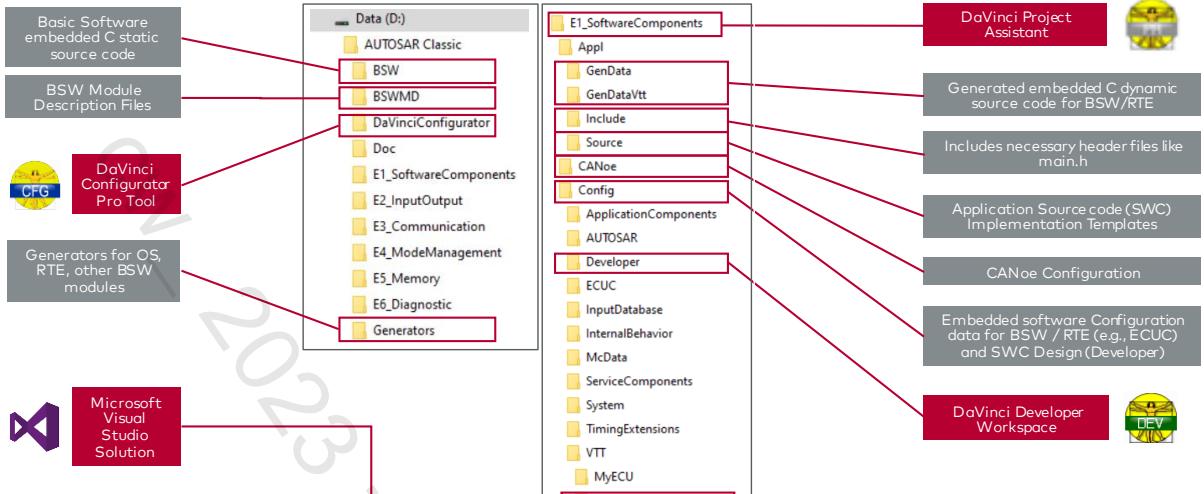
This is, for example, the second part
of four



1st Exercise – Introduction to Exercises

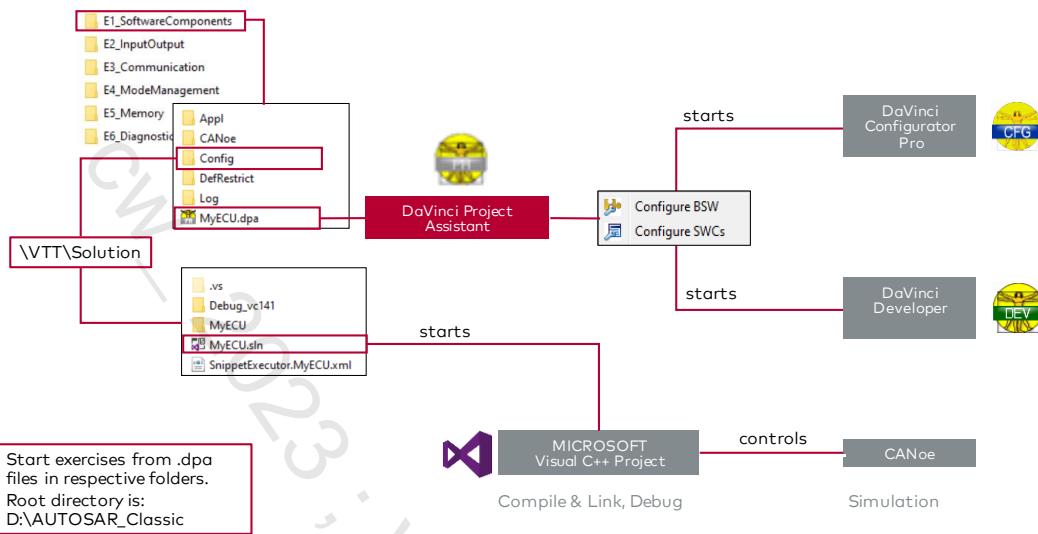
- ▶ Detailed description..
- ▶ ...of what has to be done...
- ▶ ...in the current Part of the ...
- ▶ ... exercise.

SIP and application folder structure



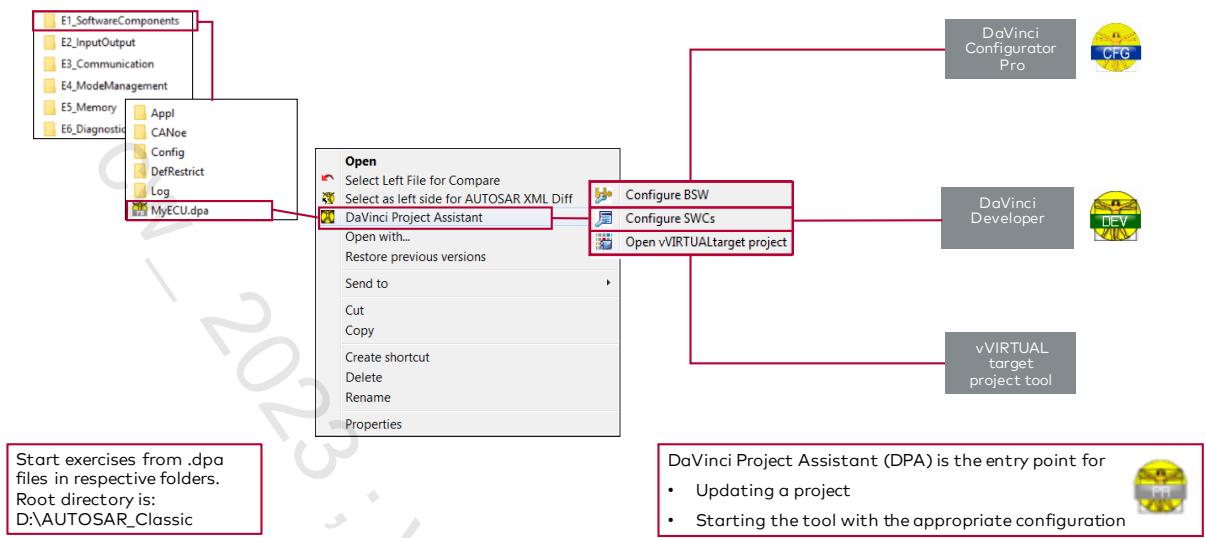
Configure the Basic Software / Design SWCs

Explorer (Win+E): D:\AUTOSAR_Classic

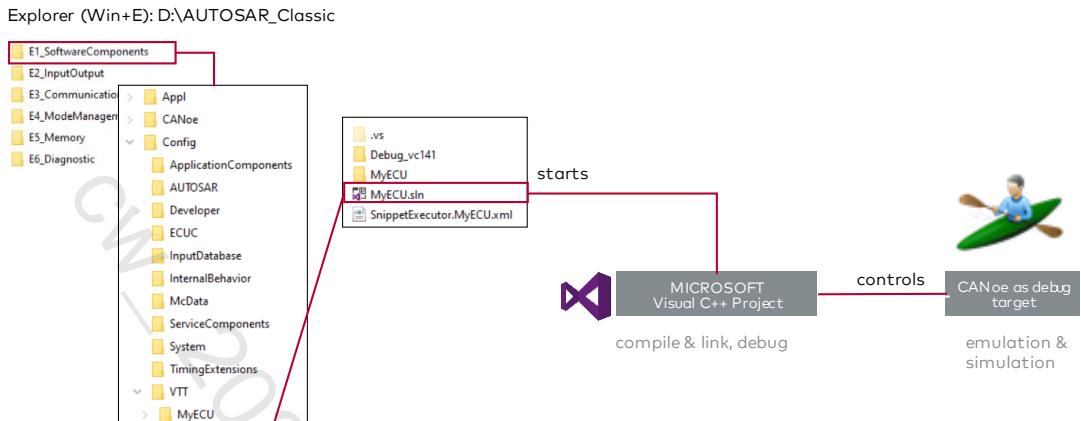


Configure the Basic Software / Design SWCs

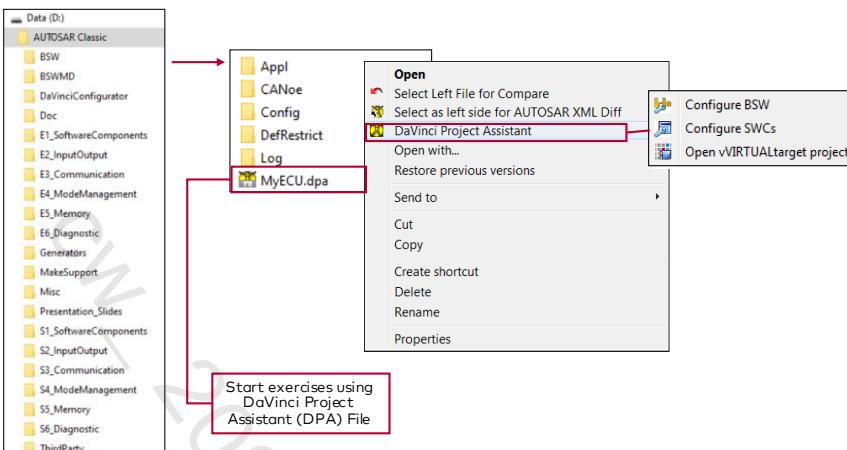
Explorer (Win+E): D:\AUTOSAR_Classic



Build and debug the embedded application



Look and Feel of all exercises



The DaVinci Project Assistant (DPA) helps you to easily set up your project and also to update the current project with new databases.

It contains links to all tools, files, and workspaces.

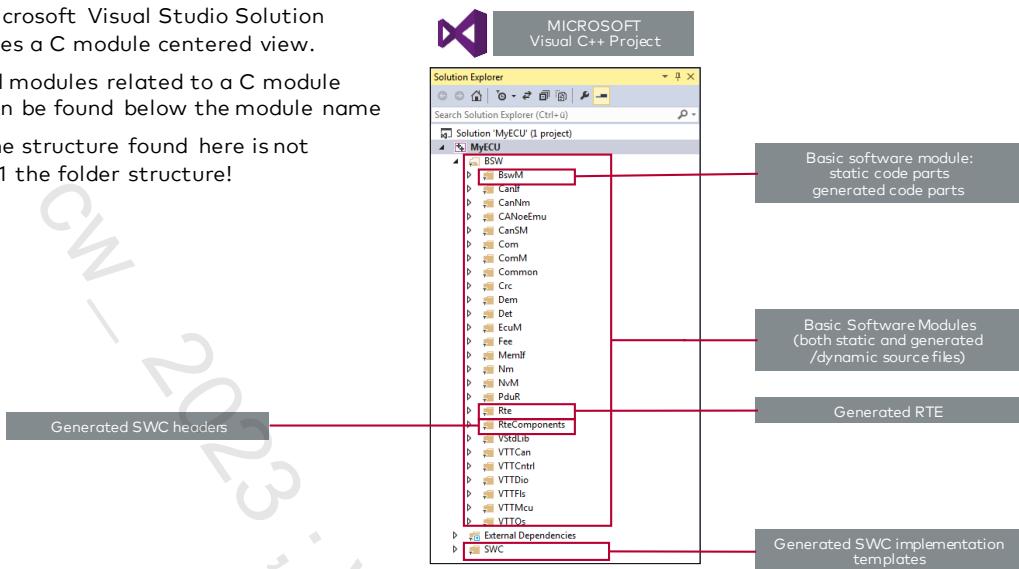
DaVinci Project Assistant (DPA) is the entry point for

- Setting up a project initially
- Updating a project
- Starting the tool with the appropriate configuration



Microsoft Visual Studio Solution

- ▶ Microsoft Visual Studio Solution uses a C module centered view.
- ▶ All modules related to a C module can be found below the module name
- ▶ The structure found here is not 1:1 the folder structure!



Naming conventions

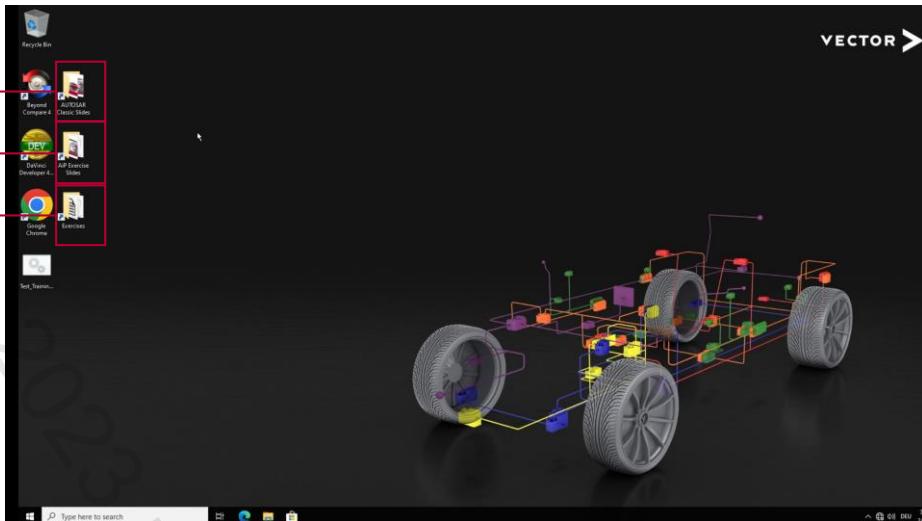
- For all elements created in DaVinci Developer in our exercises, the following naming conventions shall apply to easily identify the type of element.

Element	Convention	Example
Component Type Composition	<i>CtCo<Name></i>	CtCoApplication
Component Type Application	<i>CtAp<Name></i>	CtApMySwc
Component Type SensorActuator	<i>CtSa<Name></i>	CtSaDoor
Component Prototype	<i>Cp<Ap/Sa><Name></i>	CpApMySwc
Port Interface	<i>Pi<Name></i>	PiDoorState
Port Prototype	<i>Pp<Name></i>	PpDoorStateLeft
Data Element	<i>De<Name></i>	DeDoorState
Constant	<i>C<Name></i>	CDoorClosed
Runnable	<i>R<Ct<Name>><Runnable></i>	RCtApMySwcCode
Per Instance Memory	<i>Pim<Name></i>	PimOdometerValue
Application Data Type	<i>Adt<Name></i>	AdtLightState
Implementation Data Type	<i>Idt<Name></i>	IdtLightState
CompuMethod	<i>CM<Name></i>	CMDOORSTATE_OPEN
Mode Declaration Group	<i>Mdg<Name></i>	MdgRequestShutdown



It is advisable to avoid underscores "_" in the naming, because the RTE uses underscores in its naming rules e.g., "Rte_Write_<PortPrototype>_<DataElementPrototype>"

Desktop on Deskmate



Agenda

Introduction to Exercises

► **Training Example**

Exercise 1 - Software Components

Coming up next



The **Basic Idea** is

- ▶ One overall training example for all exercises
- ▶ The example is extended with new content in every exercise



▶ **Checkpoints**

- ▶ Completing the previous exercise is **not required** to be able to continue with the next one
- ▶ Each exercise starts at the same point for all students

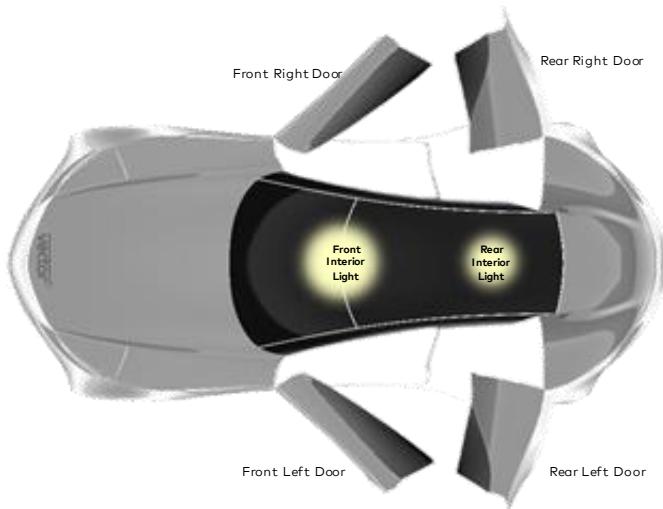
Vector Vehicle Introduction

Training Example

- ▶ Front Light and Rear Light are controlled by Front and Rear Doors
- ▶ When at least one door is open, both lights should turn on

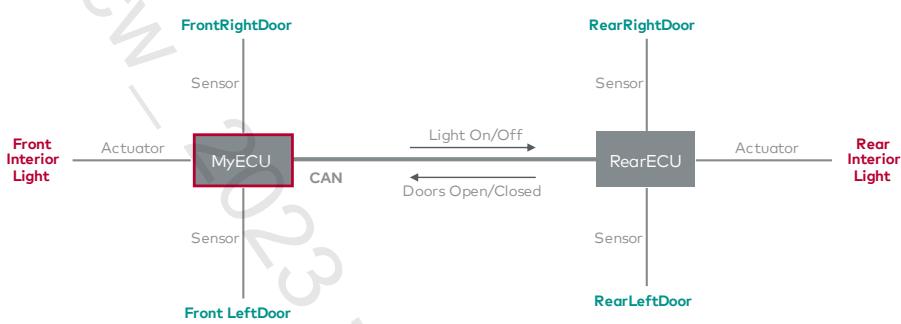
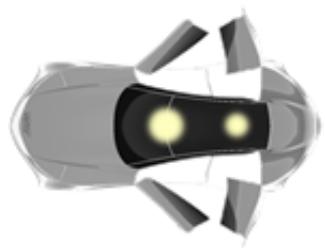
Vector Vehicle Sports Edition

- ▶ 4 Doors
- ▶ 2 Interior Lights



ECUs

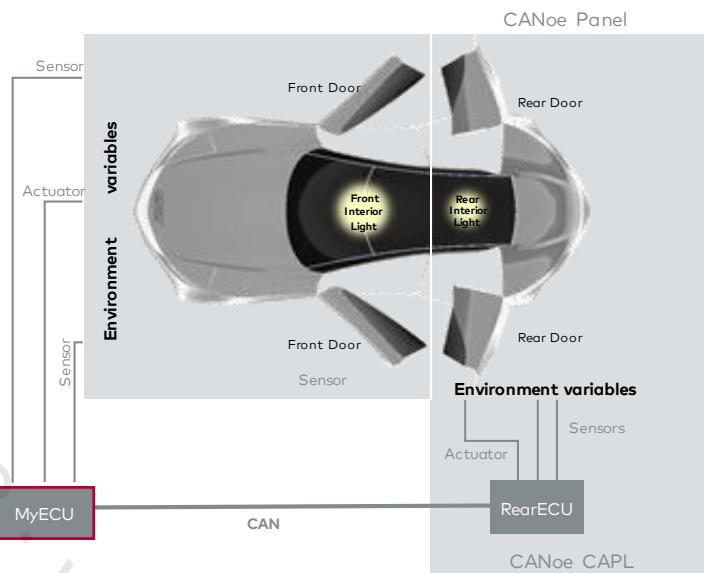
- ▶ Front doors and light are connected via I/O to MyECU
- ▶ MyECU is informed about rear doors via CAN message and also controls the rear light via CAN message



What has to be implemented and what is already there

- ▶ Only **MyECU** has to be implemented
- ▶ RearECU is CAPL code

This is your job >



What has to be implemented and what is already there

i

MyECU is the only ECU which has to have code written for it. The ECU is compiled and linked into a DLL and simulated by CANoe and the VTT MicrosarOS library.

The doors and lights are also simulated via CANoe and accessed using environment variables. The second ECU (RearECU) is a CAPL node.

How it should work:

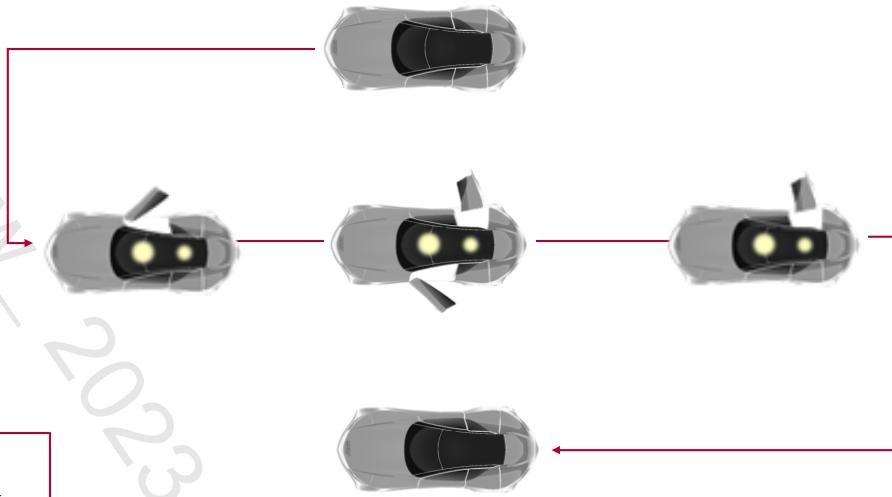
One of the front doors (or both) are opened:

Via environment variables this information is given to the I/O ports of YourECU and your software decides to switch on front light and rear light. The front light is also controlled via environment variables connected with the I/O port. The rear light is controlled via a CAN message.

One of the rear doors (or both) are opened:

Via environment variables this information is given to the CAPL node (RearECU) and the node sends a CAN message to MyECU. MyECU switches on the front light using I/O Port and environment variables and the rear light is switched on via a CAN message.

This is the desired behavior



Result of
Diagnostics
exercise is not
illustrated here!

Agenda

Introduction to Exercises

Training Example

► **Exercise 1 - Software Components**

Coming up next



1

Software Components and RTE

Get familiar with DaVinci Developer, design Software Components, and connect them via ports. Create the Runnables that will contain your code. Have a first look at DaVinci Configurator Pro to configure the OS.

Generate the BSW configuration and fill the runnable skeletons with your code.

The target of this first exercise is to generate the RTE and test the result via a breakpoint in Visual Studio.

1**Software Components and RTE**

- 1** Design Software Components, Ports, Data Elements, Connections
- 2** Create Runnables and Tasks
- 3** Configure AUTOSAR OS
- 4** Generate, code the Runnables and test

[Detailed exercise description >](#)

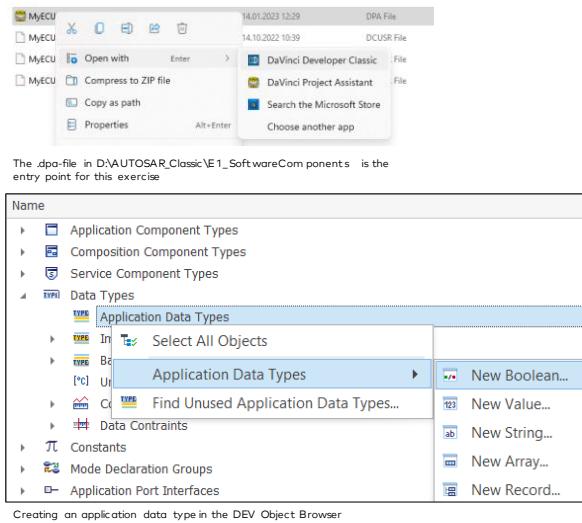
Part 1/4: Software Components

- ▶ "Open with > DaVinci Developer Classic" using **MyECU.dpa** file in folder **\E1_SoftwareComponents**

- ▶ **Create Application Component Types**
 - ▶ CtCoApplication (Type: Composition)
 - ▶ CtApMySwc (Type: Application; Support Multiple Instantiation: off; Implementation Code Type: Source Code)
 - ▶ CtSaDoor (Type: SensorActuator; Support Multiple Instantiation: on ; Implementation Code Type: Source Code)
 - ▶ CtSalteriorLight (Type: SensorActuator; Support Multiple Instantiation: on ; Implementation Code Type: Source Code)

- ▶ **Create an Application Data Type** (in the Developer Object Editor; see right)
 - ▶ AdtDoorState as New Boolean Type
 - > Assign Compu Method CMDoorState
 - ▶ AdtLightState as New Boolean Type
 - > Assign Compu Method CMLightState

- ▶ **Create Application S/R Port Interfaces**
 - ▶ PiDoorState with data element DeDoorState (Application Data Type: AdtDoorState)
 - ▶ PiLightState with data element DeLightState (Application Data Type: AdtLightState)



Part 1/4: Software Components

i

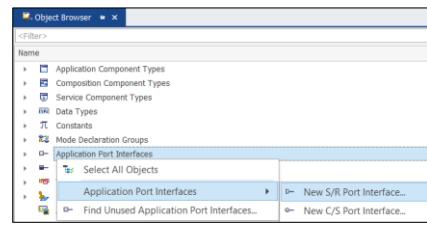
Component Types:

Component Types which may be instantiated more than once in an ECU should have **Support Multiple Instantiation** set to **On**.

The Component Type CtApMySwc will only be instantiated once in the ECU because it contains the application algorithm. Other component types may exist more than once (i.e., four doors, two interior lights).

S/R Port Interfaces

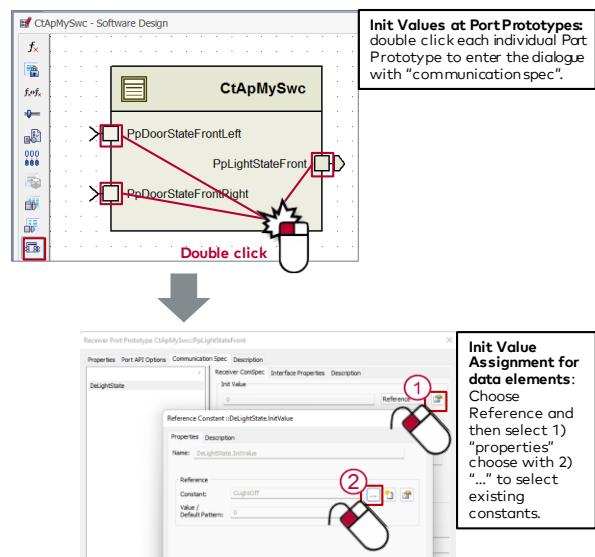
The port prototypes have to transport information about the door states and light states. Define one data element of the appropriate type for each port interface before port prototypes are instantiated.



DEV Object Browser: Creation of S/R Application Port Interfaces

Part 1/4: Software Components

- ▶ Create Port Prototypes by assigning Port Interfaces to Component Types
 - ▶ CtApMySwc
 - > PiDoorState -> PpDoorStateFrontLeft (Receiver, Init value: CDoorClosed)
 - > PiDoorState -> PpDoorStateFrontRight (Receiver, Init value: CDoorClosed)
 - > PiLightState -> PpLightStateFront (Sender, Init value: CLightOff)
- ▶ Create Port Prototypes by assigning Port Interfaces to Component Types
 - ▶ CtSaDoor
 - > PiDoorState -> PpDoorState (Sender, Init value: CDoorClosed)
 - ▶ CtSaInteriorLight
 - > PiLightState -> PpLightState (Receiver, Init value: CLightOff)



Part 1/4: Software Components



The assignment of Port Interfaces to SWC Types can only be done by instantiating the Port Interfaces as Port Prototypes.

You can do this by right-clicking on the SWC Type Interface Graphic and selecting New Port Prototype, or by dragging and dropping the Port Interface from the Developer Object Browser onto the SWC Type Interface Graphic.

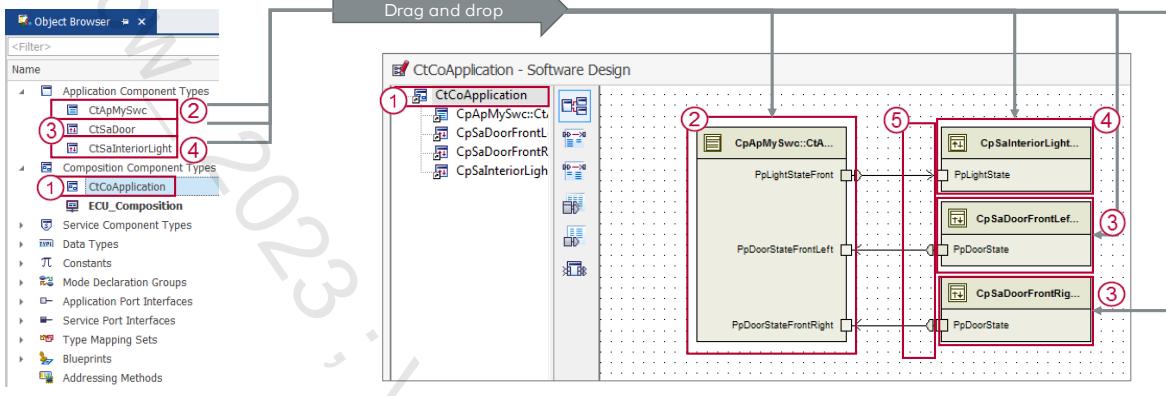
A right click on the Port Prototype → Properties allows you to select details like Prototype Name, ComSpec (Init Values).

If you dislike the graphical view, you can also switch to the Port Prototype List inside the SWC Component Type view. Use the second-lowest icon in the SWC Component Type view to access the Port Prototype List and select Application Ports → New → From Port Interface .

Autosar CP Training_EN

Part 1/4: Software Components

1. Open the composition CtCoApplication in the Object Browser
2. - 4. Instantiate all Component Prototypes inside the composition CtCoApplication
 - ▶ Drag and drop them from within the Application Components section of the Object Browser
 - ▶ Change the naming convention from **Ct*** → **Cp***
 - ▶ You need two instances of **CtSaDoor** (**CpSaDoorFrontLeft**, **CpSaDoorFrontRight**)
5. Draw the Connectors which link them among each other like shown below: **(Draw Connector Mode)**



Part 1/4: Software Components

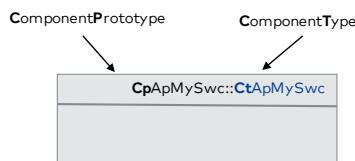
i

Please note the complete software design is contained in the composition **CtCoApplication**. This composition must contain the atomic software components you want to use in your software design.

As a final step, you must instantiate the **CtCoApplication** in the Software Design in DaVinci Developer.

The Component Type **CtSaDoor** is used twice. Two different instances (Component Prototypes) of the same Component Type exist, leading to less coding as the door code only has to be written once.

By connecting the Port Prototypes (**PpDoorState**) to different Port Prototypes of other Component Prototypes (**PpDoorStateFrontLeft**, **PpDoorStateFrontRight**), each instance has the same behavior, but with different input and output.



Also specify the default values for each defined Port Prototype.

Selected suitable constants which have already been created in the project.

Part 1/4: Software Components

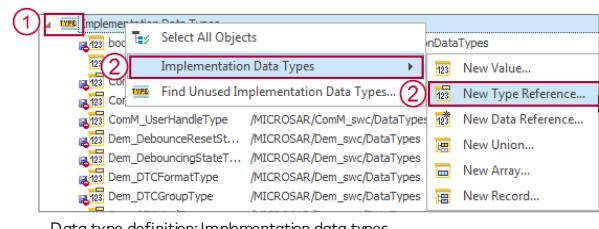
1. Go to Data Types → Implementation Data Types in the Object Browser
2. Right click, select Implementation Data Types → New Type Reference
3. Create **IdtDoorState**
 - ▶ Select Data Type 'boolean' from the Platform Types
 - ▶ Assign Compu Method CmDoorState
4. Execute Steps 1.-2. and create **IdtLightState**
 - ▶ Select Data Type 'boolean' from the Platform Types
 - ▶ Assign Compu Method CmLightState



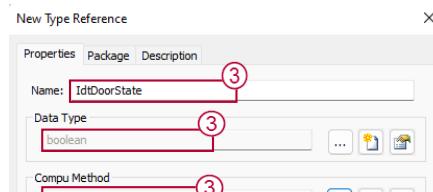
The Compu Method can be used to generate #defines into the SW component header file Rte_<SwCTypeName>.Type.h.

In our case these defines are:

CMLIGHTSTATE_LIGHTON/CMLIGHTSTATE_LIGHTOFF and
CMDOORSTATE_DOOROPEN/CLOSED



Data type definition: Implementation data types



Data Type: Implementation data type reference "boolean"
Compu Method: References enumerators for usage in the C-code

Part 1/4: Software Components

i

The constants used for initialization will not be generated by the RTE. If they are needed in the source code, you have to define your own global constants in the file Rte_usertype.h. Then these constants will be globally visible for all SWCs.

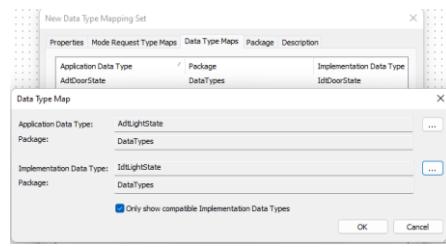
A more elegant solution could be to use the Compu Method enumerations assigned to your implementation data types. However, they are defined on a per SWC basis, depending on how the application data types are mapped to implementation data types in your SWC. The Compu Method enumerations will be generated in the Rte_<SwcTypeName>_Type.h file.

Part 1/4: Software Components

- ▶ Create a Type Mapping Set with the name "DemoTypeMapping"



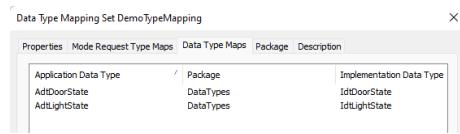
Define the type mapping set in the Object Browser, in the lower left corner of Developer GUI.



Definition of the mapping set between Application Data Type and Implementation Data Types. Our set is called "DemoTypeMapping" in our project.

- ▶ Define a Data Type Map for "DemoTypeMapping"

- ▶ Assign an Implementation Data Type to each Application Data Type
 - > AdtDoorState → IdtDoorState
 - > AdtLightState → IdtLightState

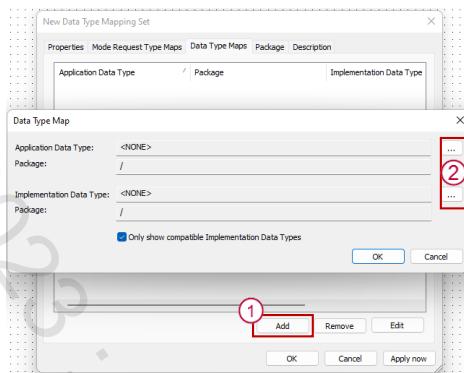


Result: the application data types are mapped to implementation data types

Part 1/4: Software Components

i

The Type Mapping set allows the implementer of a Software Component to use generic Application Data Types which do not have to be precisely defined. However, when it comes to implementation in the C language you have to be more specific about the data types. To establish a mapping of the generic Application Data Types and the concrete Implementation Data Types, each SWC has to have a reference to a Type mapping set which defines how to translate the data types.



Creation of a Data Type Map:

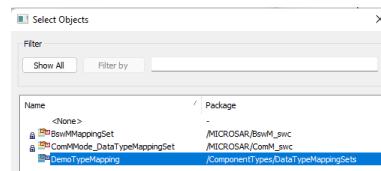
First (1) click "Add" and then specify (2) a suitable Application Data Type to Implementation Data Type mapping.

Part 1/4: Software Components

- ▶ Now it is time to assign the Type Mapping Set to all atomic SWCs in your design. Hint: You have 3 atomic SWC Types.
 - ▶ For the procedure, please have a look at the next slide
- ▶ For Each Atomic SWC Type
 - ▶ Open the Component Type Properties (see upper right figure)
 - ▶ Assign the Type Mapping Set to the SWC (see lower right figure)
- ▶ After this, you can perform a check on the SWC type. This is in the main Developer toolbar or in the context menu for an SWC Type. If **no** type mapping set is selected, DaVinci Developer reports in its message window:
 - 40317** | Missing data type mapping detected.(1 Message)
- ▶ You should **not** see this specific message (error 40317) if you have successfully assigned the data type maps to your SWC Types!
- ▶ Other warnings are OK at this point.



Adding a reference to the Type Mapping Set to the SWC Type



The type mapping set can be selected in this dialog

Part 1/4: Software Components

i

Component Type CTApMySwc

Properties **Mapping Sets** Included Data Type Sets Package Description

Referenced Mapping Set

① Properties

② Add

③ DemoTypeMapping

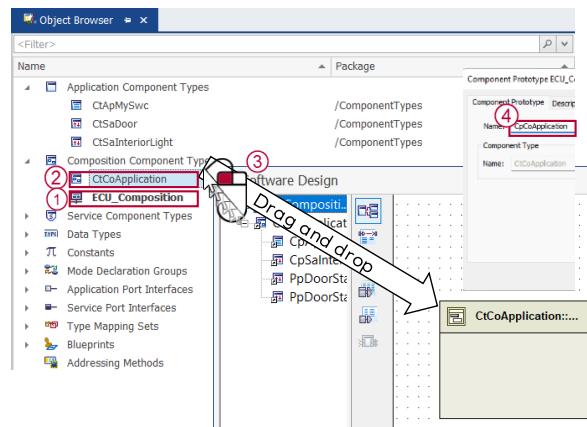
Assist the Type Mapping set to each Atomic SWC: Enter Properties → Mapping Sets (1) of the SWC Type; add (2) and select (3) the desired type mapping set.

④ Check Selection

Checking an SWC Type for completeness.

Part 1/4: Software Components

- ▶ Instantiate the Composition CtCoApplication in the Software Design view.
- ▶ Open the **ECU Composition (1)** view from the Object Browser section "Composition Component Types"
- ▶ Select **CtCoApplication (2)** from the same section and drag and drop it onto the Software Design Sheet (3) like illustrated in the right figure.
- ▶ Right click the resulting Component Prototype (4), select "Properties" and rename the resulting Component Prototype to "**CpCoApplication**."



Part 2/4: Runnables and Tasks

- ▶ The next step defines the internal behavior for the atomic SWC Types
- ▶ Each atomic SWC Type contains at least one "Runnable"
 - ▶ Each Runnable usually has a "trigger"
 - ▶ Each Runnable has a "access point" to the Port Prototypes of the SWC Type in order to process data, be called, or call an operation
 - ▶ The procedure for defining Runnables, their triggers, access points and names is illustrated in the notes section on the next slide
- ▶ For the actual Runnables for our exercise and their properties (Name, Trigger, Access Point), see the next page →

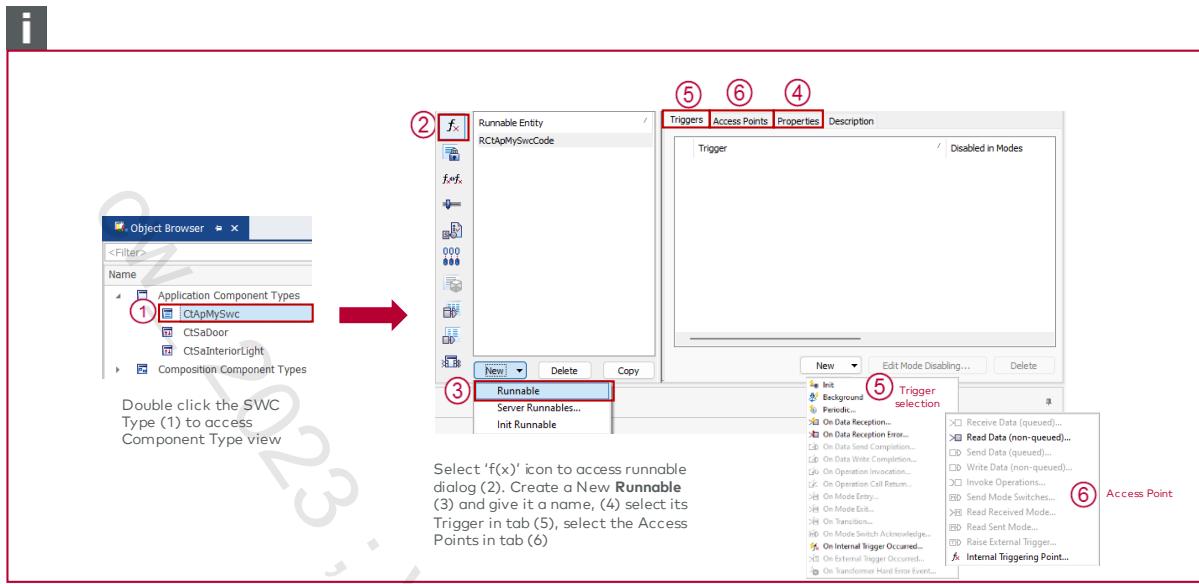


The RTE controls the execution of Runnables. The trigger condition decides when a Runnable will run. Will it run cyclically or on a certain event like data reception? This depends on your settings.

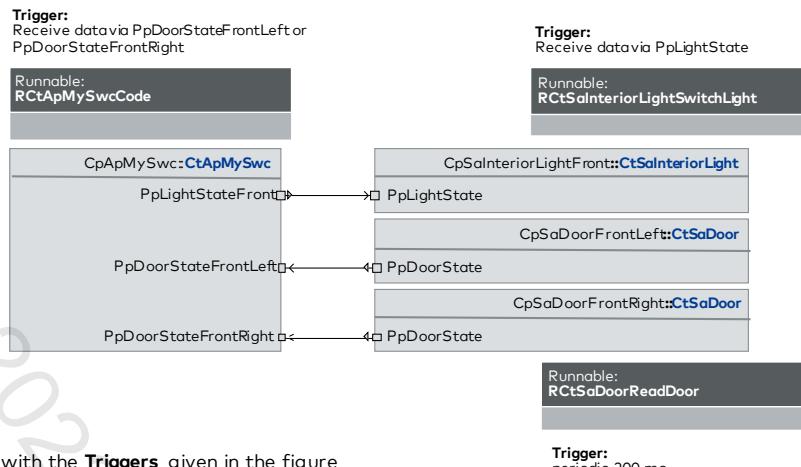
Does your code within the Runnable need access to the data elements of a receiver port, for example? Then allow the runnable to access this information via the Access Points tab.

[Illustration on next slide >](#)

Part 2/4: Runnables and Tasks



Part 2/4: Runnables and Tasks



- ▶ Define Runnables with the **Triggers** given in the figure
- ▶ Define the correct **Access Points** for each Runnable
- ▶ Check and save the workspace, then close Developer.

Part 2/4: Runnables and Tasks



Access Points for Component Types

CtSaDoor

- ▶ RCtSaDoorReadDoor
 - Add Access Point PpDoorState.DeDoorState (direct / explicit write)

CtSaInteriorLight

- ▶ RCtSaInteriorLightSwitchLight
 - Add Access Point PpLightState.DeLightState (direct / explicit read by argument)

CtApMySwc

- ▶ RCtApMySwcCode
 - Add Access Point PpDoorStateFrontLeft.DeDoorState (buffered / implicit read, just to see the difference)
 - Add Access Point PpDoorStateFrontRight.DeDoorState (direct / explicit by argument read, just to see the difference)
 - Add Access Point PpLightStateFront.DeLightState (direct / explicit write)

Part 2/4: Runnables and Tasks

- ▶ What do we do about the OS Configuration for our software architecture?
 - ▶ OS Tasks are required to execute the code of your SWCs
 - ▶ We have a design which does some I/O and has an application algorithm independent from our sensors and actuators. Therefore, we will create two tasks in the operating system:
- ▶ OS Task "My_Task"
 - ▶ All Runnables from Component Types with type **Application** (CtApMySwc) are triggered in this task.
 - ▶ Runnable RCtApMySwcCode will be mapped to this task later on.
- ▶ OS Task "IO_Task"
 - ▶ All Runnables from Component Types with type **SensorActuator** (CtSaDoor, CtSaInteriorLight) are triggered in this task.
 - ▶ The Runnables RCtSaDoorReadDoor and RCtSaInteriorLightSwitchLight will be mapped to this task later on.



Task Mapping of Runnable RCtSaDoorReadDoor

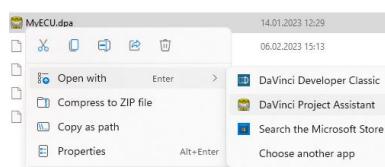
This Runnable has to be mapped twice, one time for each Component Prototype.



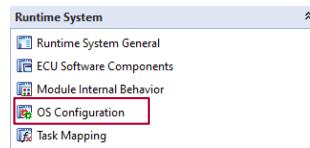
This slide summarizes only what tasks we need and for what we use them. See next slide for instructions on what to do.

Part 2/4: Runnables and Tasks

- ▶ "Configure BSW" using **MyECU.dpa** file in folder **\E1_SoftwareComponents**
- ▶ After opening Configurator, the Live Validation will run. Wait until the Live Validation has finished.
- ▶ Click on the solving action **Synchronize now**
- ▶ This brings up the RTE error **RTE01056 Unmapped runnable entity (4 messages)**
- ▶ Open **Runtime System** to address this
- ▶ Select **OS Configuration → Tasks**



The .dpa-file in
C:\AUTOSAR_Classic\E1_SoftwareComponents
entry point for this exercise



Part 2/4: Runnables and Tasks

- ▶ There are already some OS tasks defined
- ▶ **DO NOT CHANGE THESE TASKS!**
 - ▶ `IdleTask_OsCore` is an "AUTOSTART" task and is running during idle state
 - ▶ `Init_Task` is an "AUTOSTART" task and is responsible for initializing the Basic Software, SchM in particular.
 - ▶ `SchM_Task` is responsible for the periodic execution of the BSW and SchM main functions.
- ▶ Define the tasks for your application
 - ▶ `My_Task`
 - > Schedule Type: Non-preemptive;
 - Priority: 3; Activation: 2;
 - Task Type: Basic
 - ▶ `IO_Task`
 - > Schedule Type: Non-preemptive;
 - Priority: 2; Activation: 1;
 - Task Type: Extended

The screenshot shows the 'OS Configuration' dialog with the path 'All > Tasks > IdleTask_OsCore0'. A table lists four tasks:

Task	Task Activation	Task Priority	Task Schedule	Task Type
IdleTask_OsCore0	1	4294967295	FULL	AUTOMATIC
Init_Task	1	4	FULL	BASIC
SchM_Task	1	100	FULL	EXTENDED

The OS Configuration dialog in Configurator Pro. Add tasks here by clicking the '+' icon.

The screenshot shows the 'OS Configuration' dialog with the path 'All > Tasks > IdleTask_OsCore0, SchM_Task, Init_Task, My_Task, IO_Task'. A table lists five tasks:

Task	Task Activation	Task Priority	Task Schedule	Task Type
IdleTask_OsCore0	1	4294967295	FULL	AUTOMATIC
SchM_Task	1	100	FULL	BASIC
Init_Task	1	4	NON	BASIC
My_Task	2	3	NON	BASIC
IO_Task	1	2	NON	EXTENDED

Resulting task configuration shown here

Part 2/4: Runnables and Tasks

- ▶ Select **ECU Software Components** (see right)
- ▶ Your SW Design should now appear in Configurator (you will see all atomic SWC Prototypes in your design).

Triggered Function	Trigger Category	Trigger Condition	Owner	Task	Position
RCTApMySwcCode	Data Reception	PpDoorStateFrontLeft.DeDoorState	Component CpApMySwc	My_Task	
RCTApMySwcCode	Data Reception	PpDoorStateFrontRight.DeDoorState	Component CpApMySwc	My_Task	

- ▶ Define the Task mapping for each Software Component, like CpApMySwc
- ▶ Map all Runnable Function Triggers of CpApMySwc → **My_Task** (see above)
- ▶ Similarly, map all Runnable Function Triggers of
 - ▶ CpSaDoorFrontLeft → **IO_Task**
 - ▶ CpSaDoorFrontRight → **IO_Task**
 - ▶ CpSaFrontInteriorLight → **IO_Task**



Task Mapping of Runnable RCtSaDoorReadDoor

This Runnable has to be mapped twice, one time for each Component Prototype.

Part 2/4: Runnables and Tasks

- ▶ Define the position of the Function Triggers inside the tasks
- ▶ Expand **Runtime System** and open the **OS Configuration**
- ▶ For **My_Task** and **IO_Task**, select “**Mapped Functions**”
- ▶ **My_Task:**
- ▶ **RCtApMySwcCode**
PpDoorFrontLeft.DeDoorState: 0
PpDoorFrontRight.DeDoorState:0
- ▶ **IO_Task:**
- ▶ **RCtSaDoorReadDoor** 0.2 s cyclic
 - ▶ Left door:0
 - ▶ Right door:1
- ▶ **RCtSalInteriorLightSwitchLight**
PpLightStateFront.DeLightState:2



Later, we will use the **Task Mapping Assistant**. It selects the position in task automatically.

Triggered Function	Trigger Category	Trigger Condition	Position
RCtApMySwcCode	Data Reception	PpDoorStateFront.eht.DeDoorState	0
RCtApMySwcCode	Data Reception	PpDoorStateFrontRight.DeDoorState	0
SchM_Task			1

select position

Triggered Function	Trigger Category	Trigger Condition	Position
RCtSaDoorReadDoor	Periodical	200 ms	0
RCtSaDoorReadDoor	Periodical	200 ms	1
RCtSalInteriorLightSwitchLight	Data Reception	PpLightStateFront.DeLightState	2

select position

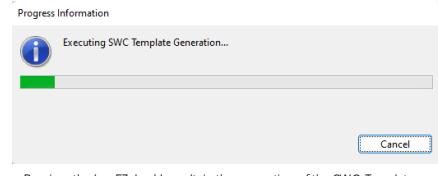
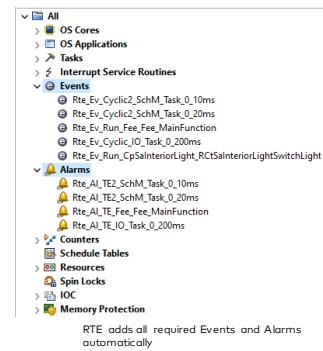


Questions:

How can you distinguish the Right Door Function Trigger from the Left Door Function Trigger? Why can you sometimes define the same task position?

Part 3/4: Configure AUTOSAR OS

- ▶ Validate the project
- ▶ Open **OS Configuration**
 - ▶ The RTE has already added all required elements resulting from the SWC Design
 - ▶ Alarms (Rte_AI_*), Events (Rte_Ev_*) ...
- ▶ Save the project
- ▶ Generate
- ▶ Click "OK" to close the 'Build VTT project' Dialog
- ▶ Close the Generation Dialog
- ▶ Press F7 to generate the SWC Templates (see (1) on next slide for details)



Pressing the key F7 should result in the generation of the SWC Templates

Part 3/4: Configure AUTOSAR OS

i

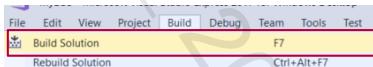
Please remember the SWC Templates also have to be updated. In the current version of the Configurator tool, this must be done using a separate icon next to the generate icon or the F7 key.



The SWC Templates have to be generated in a separate workflow step.

Part 4/4: Generate code, program Runnables and test it

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder **\E1_SoftwareComponents\Config\VTT\Solution**
- ▶ Fill in runnable skeletons in the **generated component templates** with your code (**CtApMySWC.c**, **CtSaDoor.c**, **CtSaFrontInteriorLight.c**)
 - ▶ Simulate the state of the front doors by code in **CtSaDoor.c**.
 - ▶ In **CtApMySWC.c**
 - > React on incoming information of the front doors
 - > Turn the front interior light on or off
 - ▶ Test triggering and received data in **CtSaInteriorLight.c**.
 - ▶ Build the ECU code in Visual Studio by pressing F7



What happens during code generation?

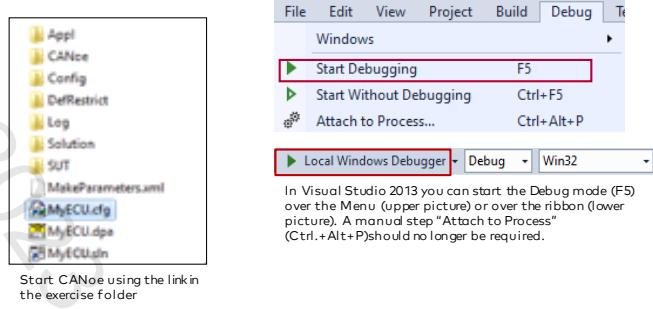
Configurator Pro generates the RTE and the component template files. Configurator Pro is also triggered to generate necessary OS files.

You will have component template files for every software component you have created. **These files can be found within your Microsoft Visual Studio solution in the folder SWC.**

Inside the C code template files are your runnable entities as empty functions – so-called "skeletons."

Part 4/4: Generate code, program Runnables and test it

- ▶ Start CANoe via the [MyECU.cfg.lnk](#) link to the CANoe configuration
- ▶ As soon as CANoe is opened, switch back to the Microsoft Visual Studio solution. Do not start the CANoe measurement yet.
- ▶ If you press F5 The Debug mode starts and attaches to RuntimeKernel.exe automatically
- ▶ Switch back to CANoe and start the simulation by pressing F9



Part 4/4: Generate code, program Runnables and test it

- ▶ Test your code. Visual C is the Debugger
 - > Use breakpoints for debugging and to test the result (see below)

**Test result of this first exercise:**

The runnable for the doors (**RCtSaDoorReadDoor**) is triggered periodically. Set a door value in code to be **CDoorClosed** or **CDoorOpen**.

As multiple instantiation is used for the component, there is only one function. A parameter 'self' is added to the API which must always be used to access the correct elements. For debugging different values, use a local variable that hands over the door state and change it in the watch window.

Set a breakpoint in the runnable **RCtApMySwcCode** and test whether the reception of the information "door is open" triggers the runnable and whether the values of the door states are correct.

Set a breakpoint in the runnable **RCtSaInteriorLightSwitchLight** and test whether the reception of the information "light on" triggers the runnable and whether the value of the light is correct.

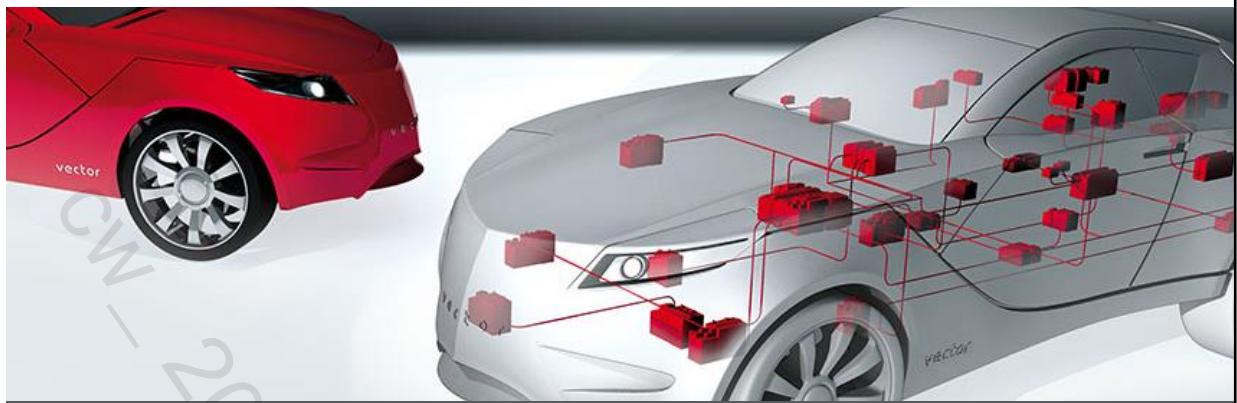
Use the interface description of the **Runnables** (comments in generated code template for every runnable) to figure out the available API. Use given Constants for setting values. These are at the beginning of the C code templates.

Self assessment

1. Where and how do you create SWC Types?
2. Where and how can you make use of SWC Types?
3. What is the relationship between SWC Types and SWC Prototypes?
With respect to
 1. their implementation
 2. the SWC prototype specific "private data"?
4. What is required to connect two SWCs to each other?
5. Is it possible to connect
 1. two SWC Types to each other?
 2. two SWC Prototypes to each other?
6. What does a Port Interface contain? (Sender/Receiver Interface type)
7. Which element of a software component executes code?
8. How can you define the point in time or condition at which this code becomes executed?
9. What do you have to do in order to read or write data from/to the Ports of an SWC?
 1. In the Developer tool
 2. On code level
10. Once the access is enabled, functions to read/write data from/to the Ports are globally visible inside the SWC. Please name the boundary condition which must be fulfilled for each Runnable additionally! Give a reason for this.

Software Components

Input Output



AUTOSAR in Practice

I/O

V1.0.0 | 2020-01-20

Agenda

► **Architecture of I/O**

I/O Drivers – General Characteristics

PORT Driver

DIO Driver

PWM Driver (for reference)

ICU Driver (for reference)

OCU Driver (for reference)

ADC Driver (for reference)

Client Server Concept

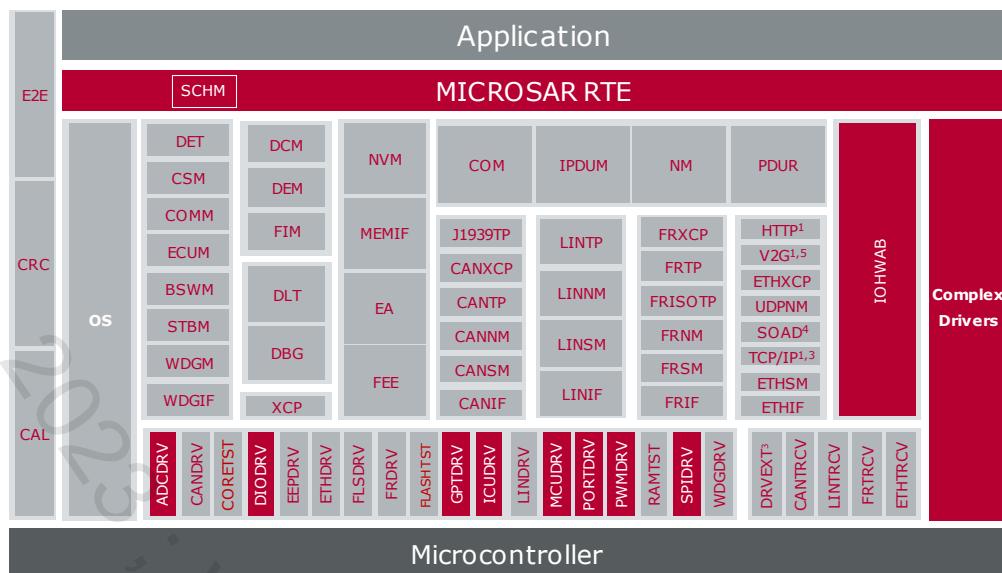
I/O Hardware Abstraction

Coming up next

Elements for Input / Output in AUTOSAR

Affected BSW Modules

- ▶ I/O Hardware Abstraction
- ▶ ICU Driver
- ▶ OCU Driver
- ▶ PWM Driver
- ▶ ADC Driver
- ▶ DIO Driver
- ▶ PORT Driver
- ▶ GPT Driver
- ▶ SPI Driver



Elements for Input / Output in AUTOSAR

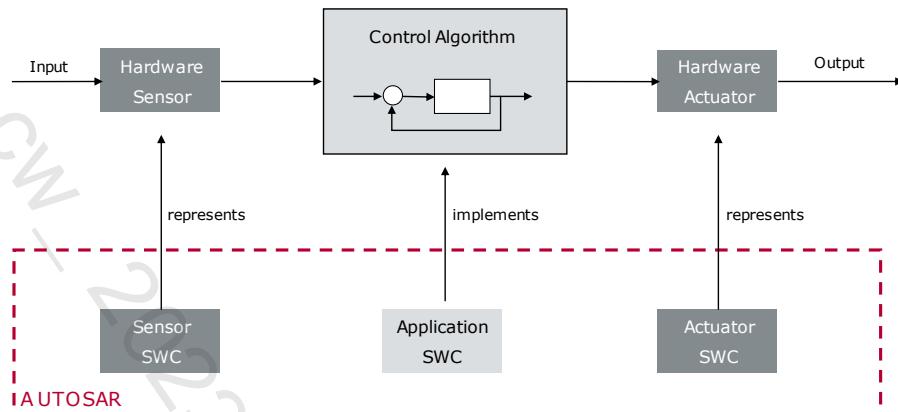
i

We will focus on the Microcontroller Abstraction Layer (MCAL), which is provided by the hardware vendor.

- The MCAL provides a standardized access to the HW of your microcontroller to the upper layers
- No signal state or pin state conversions are done by the MCAL

CW_2023_VH_Autosar CP Training_EN

Generic model of an application

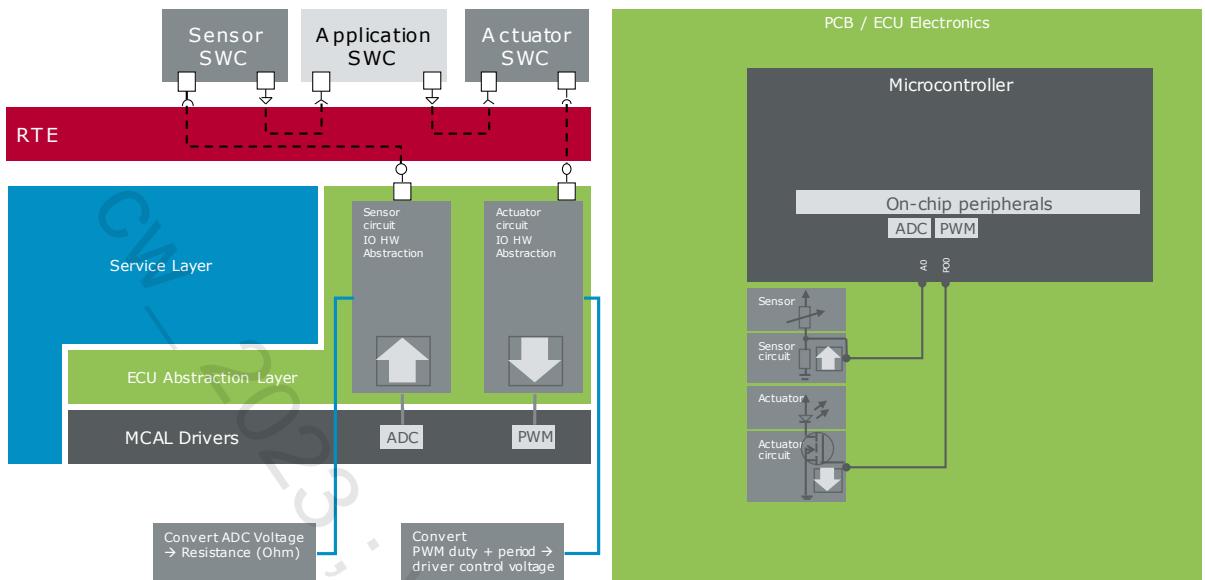


Generic model of an application

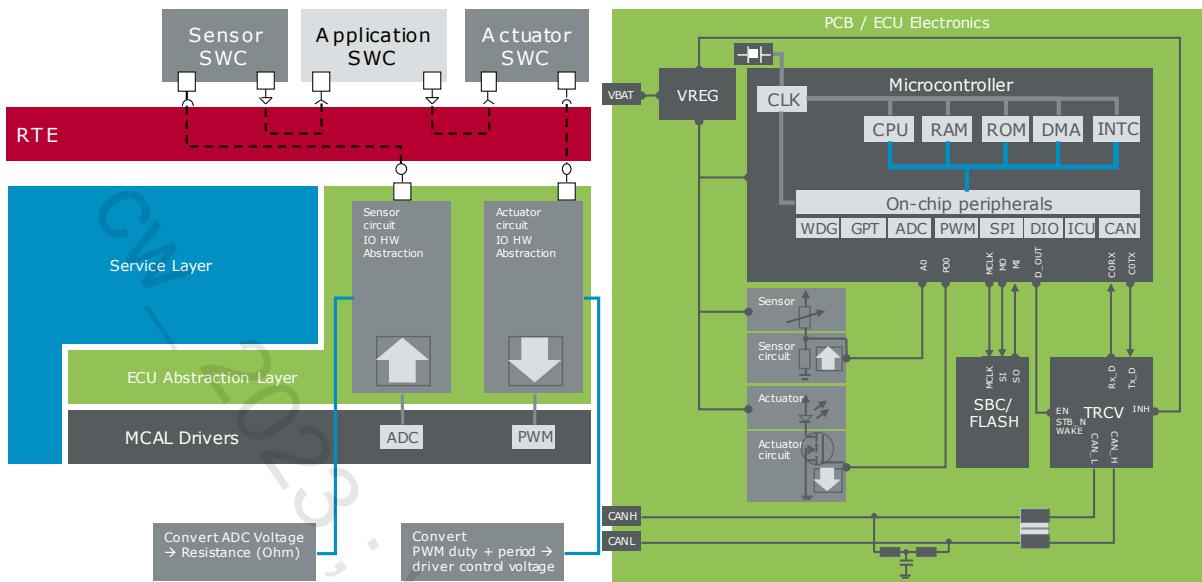
i

A main element of automotive applications are control algorithms which perform a function like fuel injection or temperature control. Control algorithms calculate output values based on certain input values. If the inputs and outputs are located on the same ECU as the control algorithm, then input/output peripherals of the Microcontroller or ECU are used. Usually sensor values have to be read and actuators have to be controlled. The following slides show how input/output control can be designed in AUTOSAR.

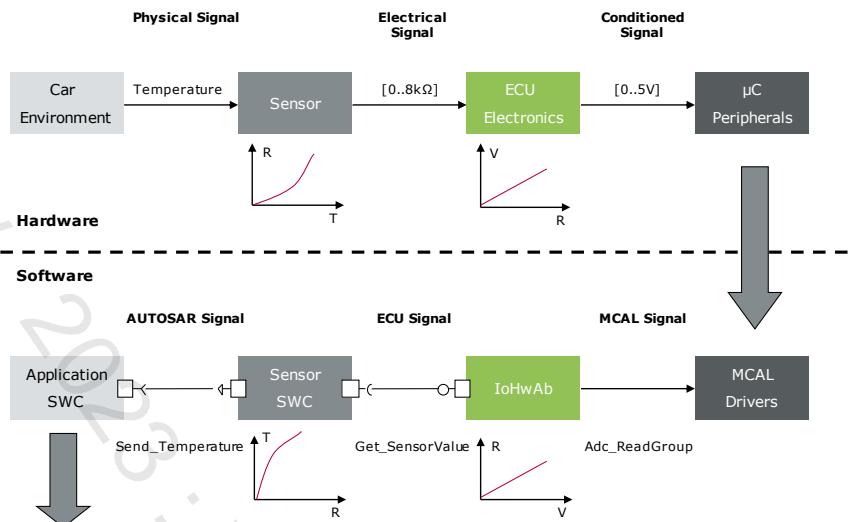
The ECU Abstraction Layer



The ECU Abstraction Layer



Sensor Signal Path



Sensor Signal Path

i

Sensor: Converts a physical or environmental signal into an electrical signal

ECU Electronics: Electronic circuit which converts/conditions the sensor's electrical signal into an input signal suitable for the microcontroller (e.g. 5V, 3.3 V)

μC Peripherals: Analog to Digital Converter, Capture Compare Unit, SPI

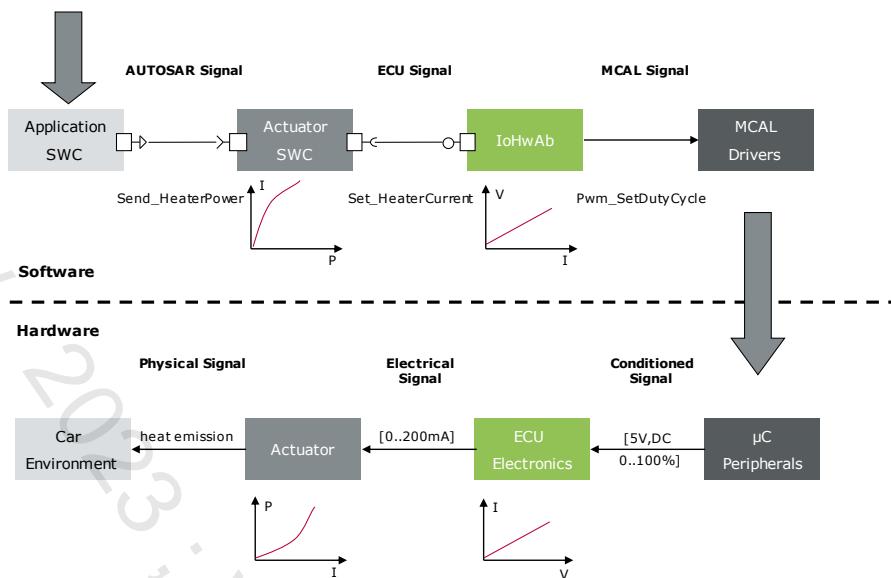
MCAL Drivers: AUTOSAR standardized peripheral software drivers

I/O Hardware Abstraction: Does signal conditioning/debouncing/filtering. Converts from hardware-specific raw values into physical values

Sensor SWC: SW representation of hardware sensor. Converts electrical input values into application signals. Has knowledge of sensor hardware characteristics

Application SWC: Implements algorithm for control of sensors and actuators

Actuator Signal Path



Actuator Signal Path



Application SWC: Implements algorithm for control of sensors and actuators

Actuator SWC: SW representation of hardware actuator. Converts the application signals/commands into hardware values. Has knowledge of actuator hardware characteristics

I/O Hardware Abstraction: Does signal conditioning/debouncing/filtering. Converts from hardware specific raw values into physical values

MCAL Drivers: AUTOSAR standardized peripheral software drivers

μC Peripherals: PWM Module, Digital IO, SPI

ECU Electronics: An electronic circuit which converts/conditions the sensor's electrical signal into an input signal for the microcontroller (e.g. 5V, 3.3 V)

Actuator: Converts an electrical signal into a physical or environmental signal

Agenda

Architecture of I/O

► **I/O Drivers – General Characteristics**

PORT Driver

DIO Driver

PWM Driver (for reference)

ICU Driver (for reference)

OCU Driver (for reference)

ADC Driver (for reference)

Client Server Concept

I/O Hardware Abstraction

Coming up next

- ▶ Basic driver access
- ▶ No data conversion -> plain reading of register values
 - ▶ PORT: No level inversion
 - ▶ ADC: No value conversion
 - ▶ PWM/ICU: No time/frequency conversion
- ▶ Time unit of register values is ticks
 - ▶ Actual time values depend on the MCU settings
 - ▶ Conversion from ticks to time is not done
 - > Has to take place in the I/O Hardware Abstraction

i

No time conversion for PWM or GPT. Basic tick value is based on the peripheral input clock domain.

- ▶ Drivers make use of dedicated hardware functions
 - ▶ E.g., PWM, Capture Compare Unit
- ▶ Common concept
 - ▶ Channel -> I/O pin, ADC/PWM channel
 - ▶ Channel Group -> group of pins, group of ADC channels
 - ▶ Port -> a port register (e.g. 8-bit port)

i

MCALs work with the dedicated HW functions and peripherals!!!

Agenda

Architecture of I/O
I/O Drivers – General Characteristics

► **PORT Driver**

DIO Driver
PWM Driver (for reference)
ICU Driver (for reference)
OCU Driver (for reference)
ADC Driver (for reference)
Client Server Concept
I/O Hardware Abstraction
Coming up next

Characteristics

- ▶ Configuration of ports
 - ▶ Mode (GPIO/alternative mode)
 - ▶ Direction (input/output)
 - ▶ Direction changeable during runtime (yes/no)
 - ▶ Initial pin level (low/high)
- ▶ Optional
 - ▶ Slew rate control
 - ▶ Activation of internal pull ups
 - ▶ Input thresholds
 - ▶ Pin driver mode (push-pull/open drain)
 - ▶ Type of read-back support (pin level, output register value)

**PORT Driver API:**

`Port_Init`: Initialize port data register of a specific port pin

`Port_SetPinDirection`: Enables setting of the port pin direction during runtime

`Port_RefreshPortDirection`: Refresh direction of non-dynamically configurable port pins

`Port_SetPinMode`: Change the pin mode (GPIO or alternative function) during runtime

Port Driver needed to configure port functions like e.g. CAN_RX

→ This is not done by the CAN driver

→ Port Driver is HW dependent and may always look a bit different between platforms

Agenda

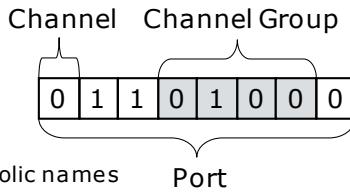
- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- ▶ **DIO Driver**
- PWM Driver (for reference)
- ICU Driver (for reference)
- OCU Driver (for reference)
- ADC Driver (for reference)
- Client Server Concept
- I/O Hardware Abstraction
- Coming up next

Characteristics

- ▶ Abstracts the access to digital I/O pins (channels)
- ▶ Allows grouping of port pins (channel groups)
- ▶ Provides read/write access of the General Purpose digital I/O for

- ▶ Channels
- ▶ Channel groups
- ▶ Ports

- ▶ Access to elements by user-defined symbolic names



Characteristics

i

Used by the BSW, e.g. by the Transceiver Drivers:

- Inside the Transceiver Driver a reference to a DIO channel is provided

Definition of Channel, Channel Group, Port

Channel -> a single port pin

Channelgroup -> a combination of **adjacent** DIO channels

Port -> a port register (e.g. 8-bit port)

DIO Driver API:

`Dio_ReadPort`: Reading of a complete port register

`Dio_WritePort`: Writing of a complete port register

`Dio_ReadChannelGroup`: Reading of a channel group

`Dio_WriteChannelGroup`: Writing of a channel group

`Dio_ReadChannel`: Reading of a single channel (Port-Pin)

`Dio_WriteChannel`: Writing of a single channel (Port-Pin)

`Dio_FlipChannel` : Reading of an output channel, inversion and writing of this value

Agenda

- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- DIO Driver
- ▶ **PWM Driver (for reference)**
- ICU Driver (for reference)
- OCU Driver (for reference)
- ADC Driver (for reference)
- Client Server Concept
- I/O Hardware Abstraction
- Coming up next

Characteristics

- ▶ Generation of Pulse -Width-Modulated (PWM) Signals using specific PWM hardware functions
- ▶ PWM configuration parameters for each channel
 - ▶ Assigned HW channel
 - ▶ Period
 - ▶ Duty Cycle
 - ▶ Polarity (high/low)
 - ▶ Idle State (high/low)
 - ▶ Channel Class
 - > Fixed period
 - > Fixed period, shifted (if supported by hardware)
 - > Variable period



No support for software PWM. Software PWM can be implemented in the I/O Hardware Abstraction using DIO Channels.

PWM module needs dedicated hardware peripheral.

Characteristics

- ▶ Optional configuration parameters
 - ▶ Phase shift
 - ▶ Reference channel for phase shift
 - ▶ Microcontroller specific channel properties
- ▶ Duty cycle -> 16 bit value
 - ▶ 0x0000 -> 0%
 - ▶ 0x8000 -> 100%
- ▶ Pwm_Main_PowerTransitionManager
 - ▶ Used in case of asynchronous power management, otherwise no periodic main function.

Characteristics



PWM Driver API

`Pwm_Init`: Initializes all internal variables and resources and starts all PWM channels with the configured default values; All notifications are disabled

`Pwm_DeInit`: De-initializes the PWM module; disables all PWM interrupts and notifications; Sets the PWM output signals to the configured idle states

`Pwm_SetDutyCycle`: Sets the duty cycle (Min = 0x0000, Max = 0x8000)

`Pwm_SetPeriodAndDuty`: Sets the period and duty cycle

`Pwm_SetOutputToIdle`: Sets the PWM output signals to the idle state

`Pwm_GetOutputState`: Reads the internal state of the PWM output signal

`Pwm_EnableNotification`: Enables the PWM signal edge notification (PWM_RISING_EDGE, PWM_FALLING_EDGE, PWM_BOTH_EDGES)

`Pwm_DisableNotification`: Disables the signal edge notification

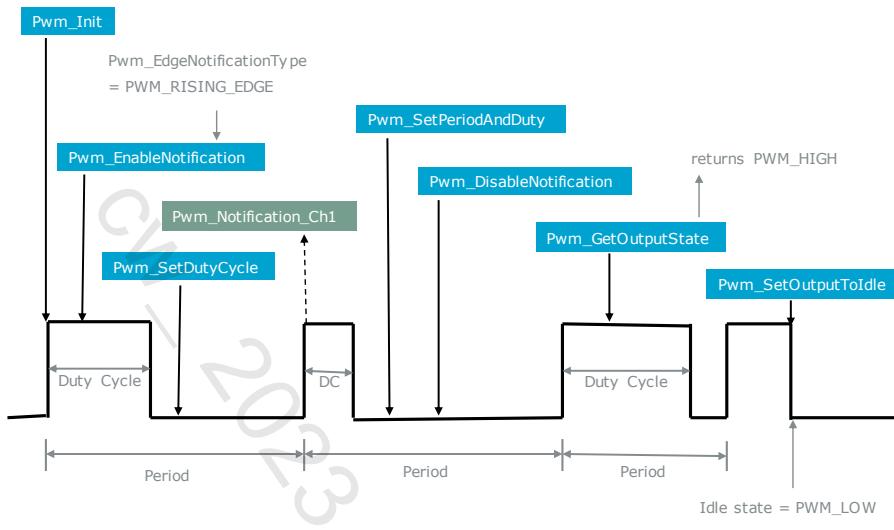
`Pwm_SetPowerState`: Set the power mode of the PWM HW Unit. Configures the Pwm module so that it enters the already prepared power state, chosen between a predefined set of configured ones.

`Pwm_GetCurrentPowerState`: Get the current power mode of the PWM HW Unit

`Pwm_GetTargetPowerState`: Target power mode of the PWM HW Unit. Returns the requested power state of the HW unit. This shall coincide with the current power state if no transition is ongoing. The API is considered to always succeed except in case of HW failures.

`Pwm_PreparePowerState`: This API starts the needed process to allow the PWM HW module to enter the requested power state. This API initiates all actions needed to enable a HW module to enter the target power state. The possibility to operate the peripheral depends on the power state and the HW features. These properties should be known to the integrator and the decision whether to use the peripheral or not is in his responsibility.

Example



Example



- 1) Pwm_Init is called. PWM starts with default duty cycle and period. Polarity = PWM_HIGH. Idle state = PWM_LOW
 - 2) Notification (rising edge) is enabled. Notification Callback is called on next rising edge. Callback API: Pwm_Notification_<Channel No>. In Example "Ch1" is used (user-defined name)
 - 3) Duty Cycle is set to a smaller value. Change takes effect at the end of the period
 - 4) Period and Duty Cycle are changed. Change takes effect at the end of the period
 - 5) Notification is disabled
 - 6) Output state is checked using Pwm_GetOutputState. PWM_HIGH is returned
 - 7) PWM is set to idle state => LOW (Interrupts still active)
- Not in picture:
- 8) To restart the PWM signal generation: Pwm_SetDutyCycle or Pwm_SetPeriodAndDuty has to be called
 - 9) Pwm_DeInit: all interrupts disabled, all Notifications disabled, PWM output set to configured idle state

Agenda

- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- DIO Driver
- PWM Driver (for reference)
- ▶ **ICU Driver (for reference)**
- OCU Driver (for reference)
- ADC Driver (for reference)
- Client Server Concept
- I/O Hardware Abstraction
- Coming up next

Characteristics

- ▶ Input Capture Unit (ICU) is used for
 - ▶ Edge Count
 - ▶ Signal Edge Detection / Notification
 - ▶ Time stamping
 - ▶ Signal Measurement



Used as well to generate an interrupt based on the detection of some signal edge:

Used for wakeup detection (either from CAN Transceiver or dedicated wakeup input)

Can be used for bringing MCU back from sleep mode

General API Functions

`Icu_Init`

Initializes the relevant registers of the configured hardware

Disables all notifications

Disables the wakeup capability of all channels

Sets status to ICU_IDLE

Sets module mode to ICU_MODE_NORMAL

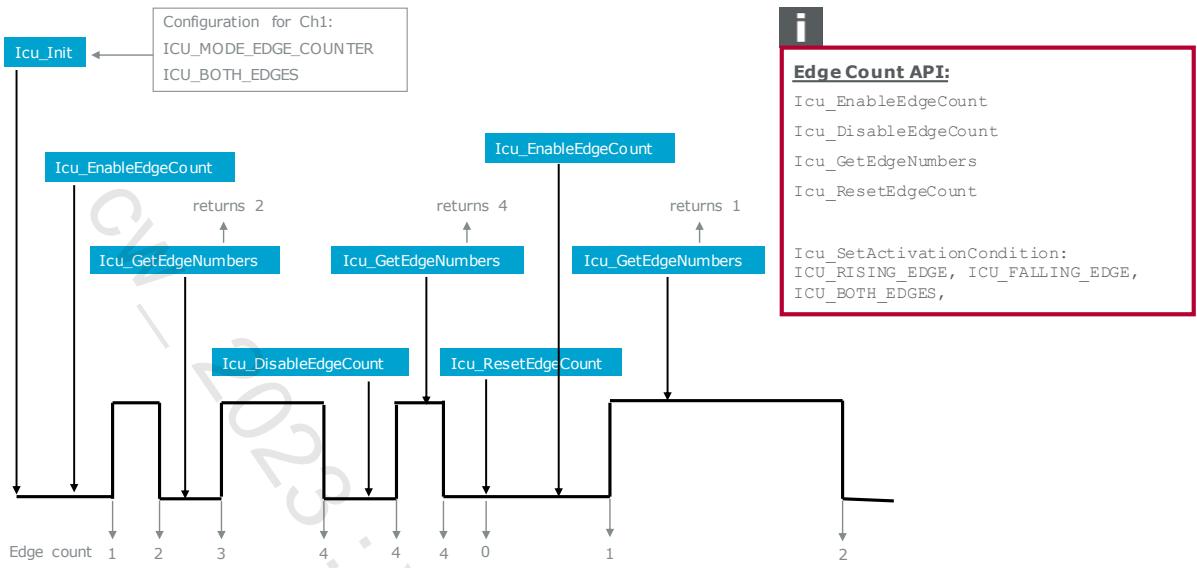
`Icu_DeInit`

Sets the state of the peripheral to same state as after power-on reset

Wakeup Functionality

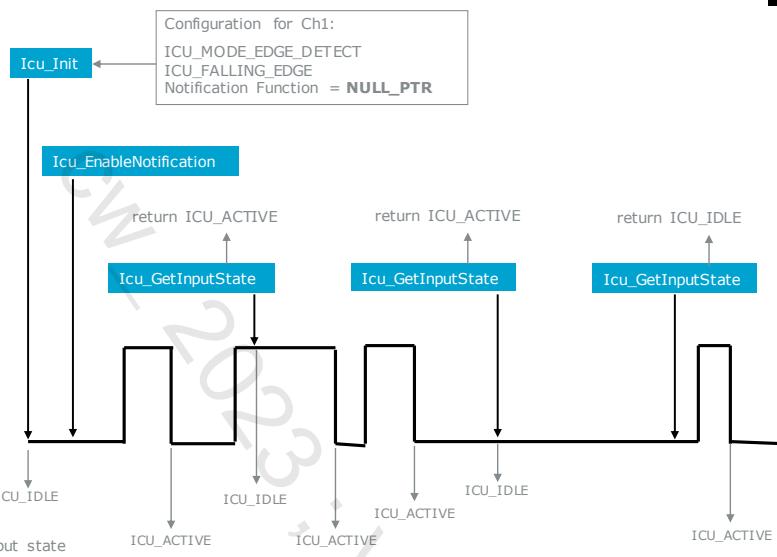
- ▶ ICU Driver can be used as a Wakeup Source
- ▶ **Wakeup API**
 - ▶ Icu_SetMode
 - ▶ Icu_EnableWakeup
 - ▶ Icu_DisableWakeup
- ▶ **From ISR or cyclically by EcuM:**
 - ▶ EcuM_CheckWakeup → Icu_CheckWakeup → EcuM_SetWakeupEvent

Edge Count - Example



Signal Edge Detection – Example 1

Polling



23

Signal Edge Detection – Example 1

i

ICU_ACTIVE is returned if a falling edge was detected before the latest **Icu_GetInputState** call

Note: No notification raised because Notification function is defined as NULL_PTR

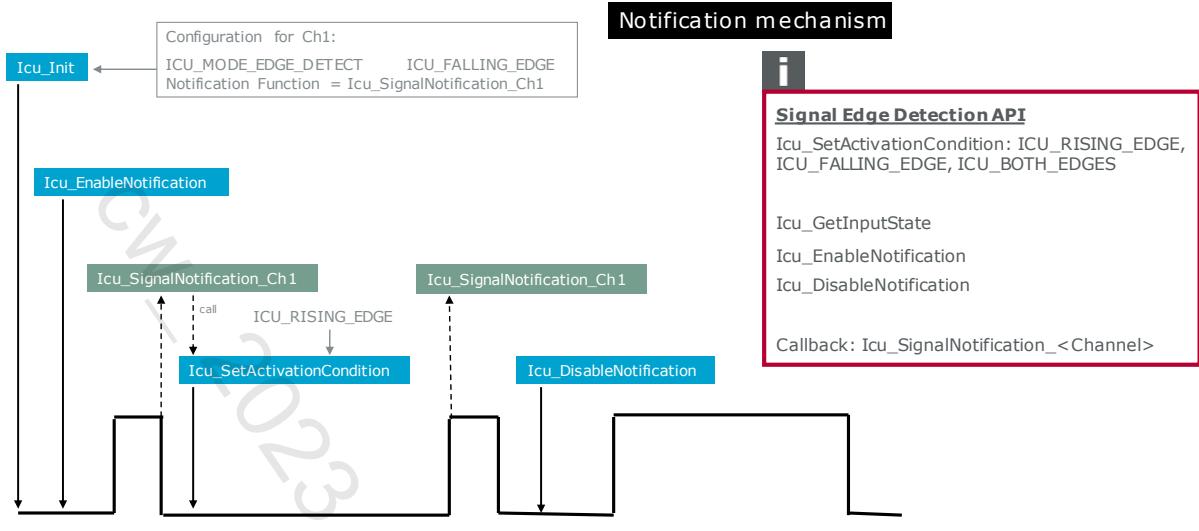
Signal Edge Detection API

Icu_SetActivationCondition: ICU_RISING_EDGE, ICU_FALLING_EDGE, ICU_BOTH_EDGES

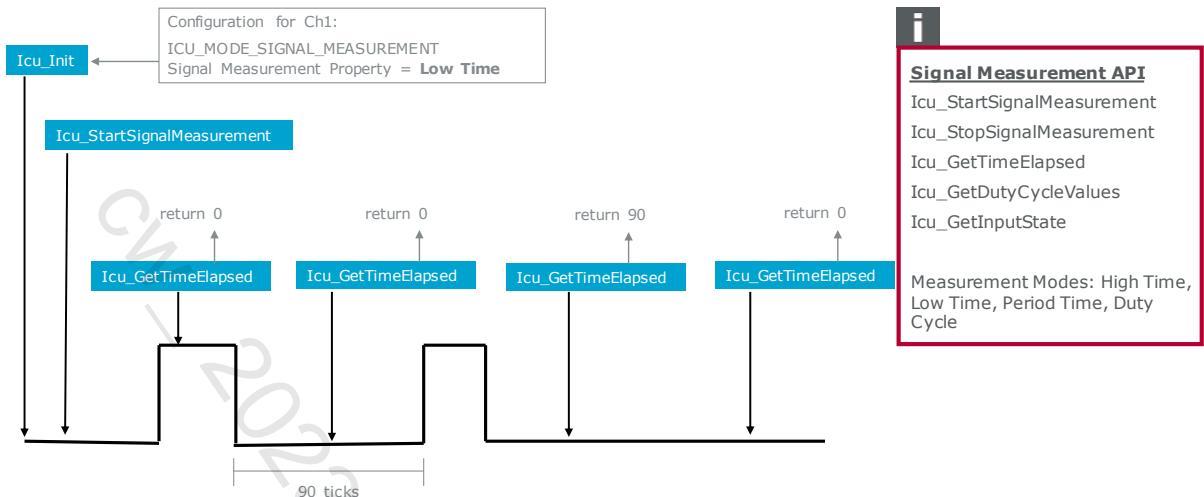
Icu_GetInputState
Icu_EnableNotification
Icu_DisableNotification
Icu_EnableEdgeDetection
Icu_DisableEdgeDetection

Callback: **Icu_SignalNotification_<Channel>**

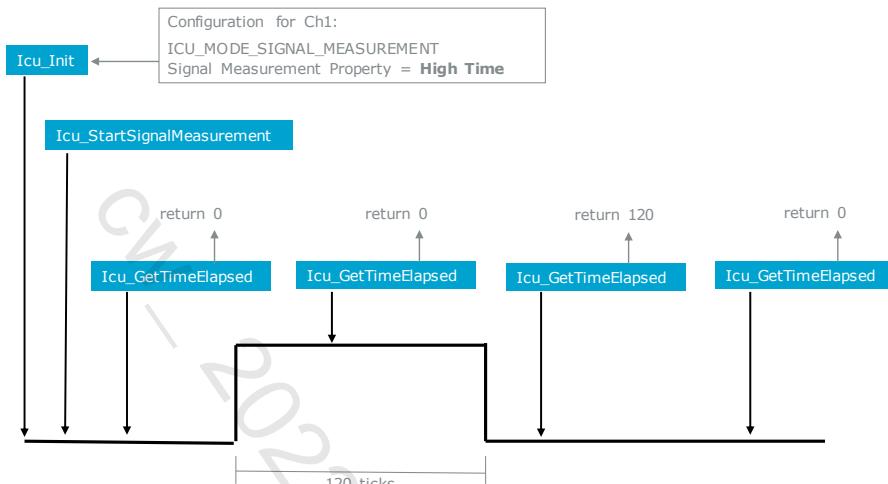
Signal Edge Detection – Example 2



Signal Measurement – Example 1



Signal Measurement – Example 2

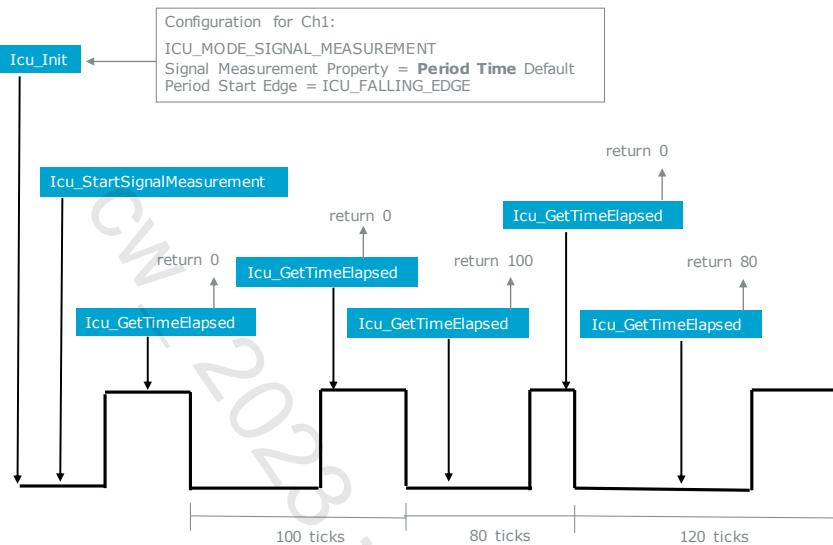


Signal Measurement – Example 2

i

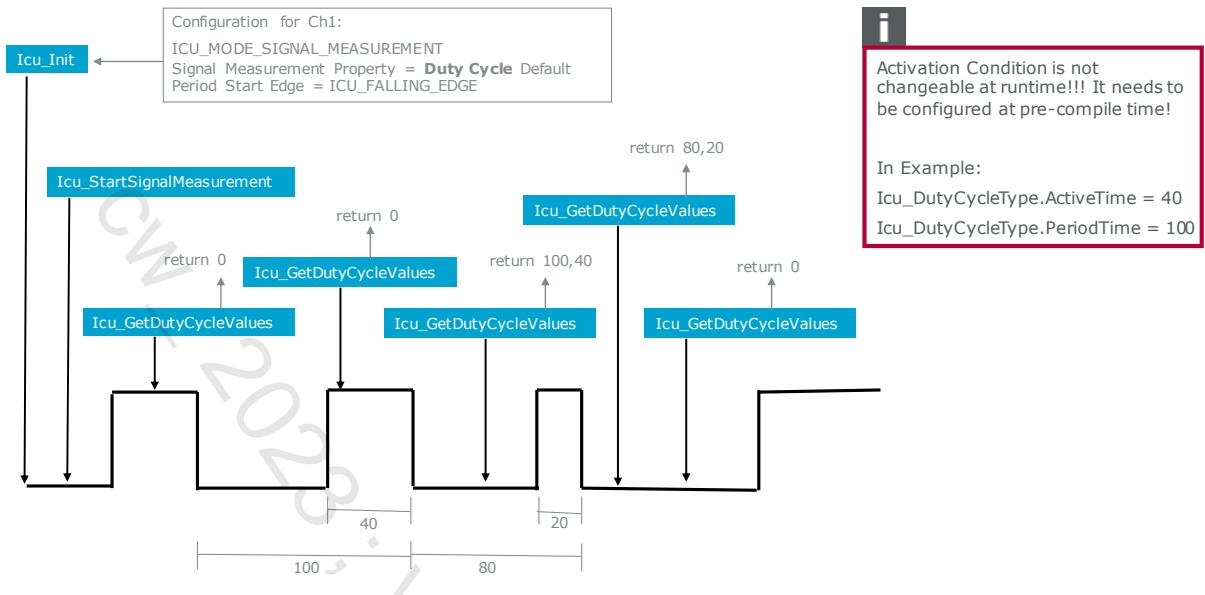
1. Icu_Init is called with the configured ConfigPtr
2. Icu_StartSignalMeasurement starts the measurement
3. First call of Icu_GetTimeElapsed returns 0 because no rising edge was detected yet
4. Internal measurement is started with rising edge
5. Second call of Icu_GetTimeElapsed still returns 0 because no falling edge was detected
6. Falling edge ends the measurement of high time
7. Third call of Icu_GetTimeElapsed returns measured high time (120 ticks); Internal measurement value is reset to 0
8. Fourth call of Icu_GetTimeElapsed: same as for 3.

Signal Measurement – Example 3

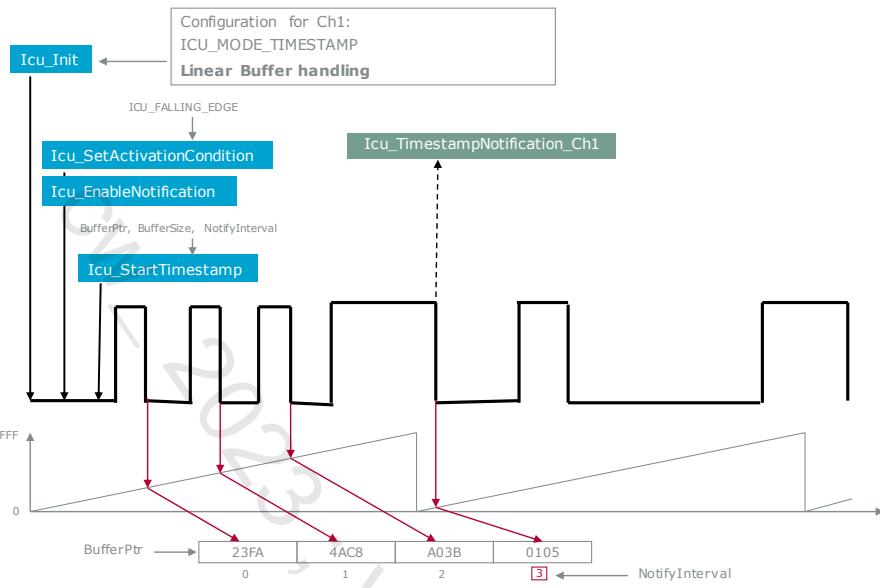


Activation Condition is not changeable at runtime!!! It needs to be configured at pre-compile time!

Signal Measurement – Example 4



Timestamping – Example 1



Timestamping – Example 1

i

Notification when the buffer is Full

Timestamping API

Icu_StartTimestamp

Icu_StopTimestamp

Icu_GetTimestampIndex

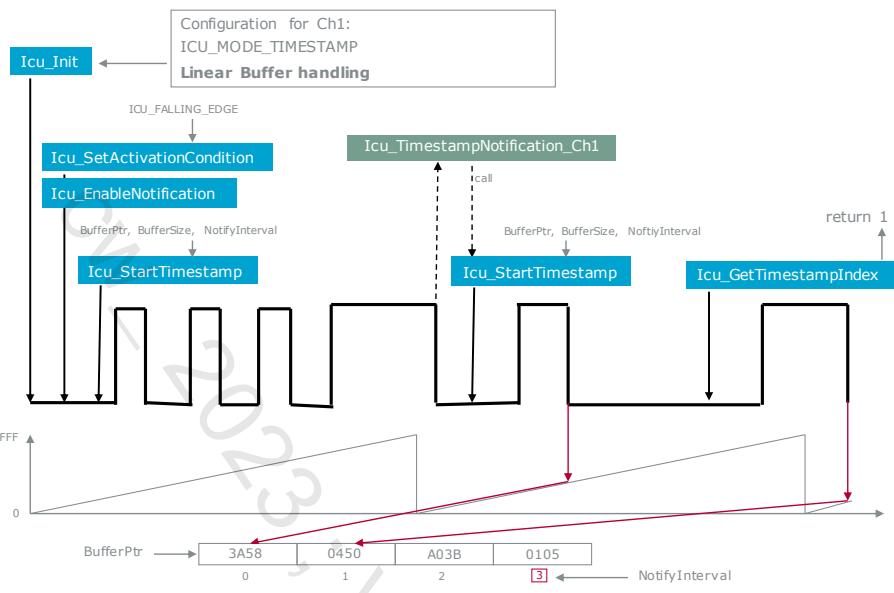
Icu_SetActivationCondition: ICU_RISING_EDGE, ICU_FALLING_EDGE, ICU_BOTH_EDGES

Icu_EnableNotification

Icu_DisableNotification

Callback: Icu_TimestampNotification_<Channel>

Timestamping – Example 1



Timestamping – Example 1

i

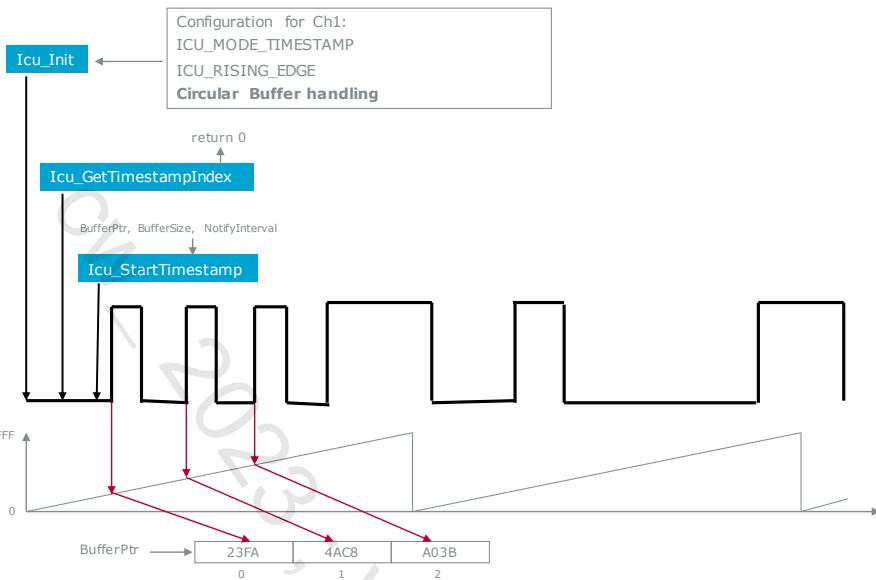
Same as the last page but after the Buffer Full notification, the Index value is retrieved by API

Use a notification function to get informed if buffer is full (or as an alternative poll the TimestampIndex)

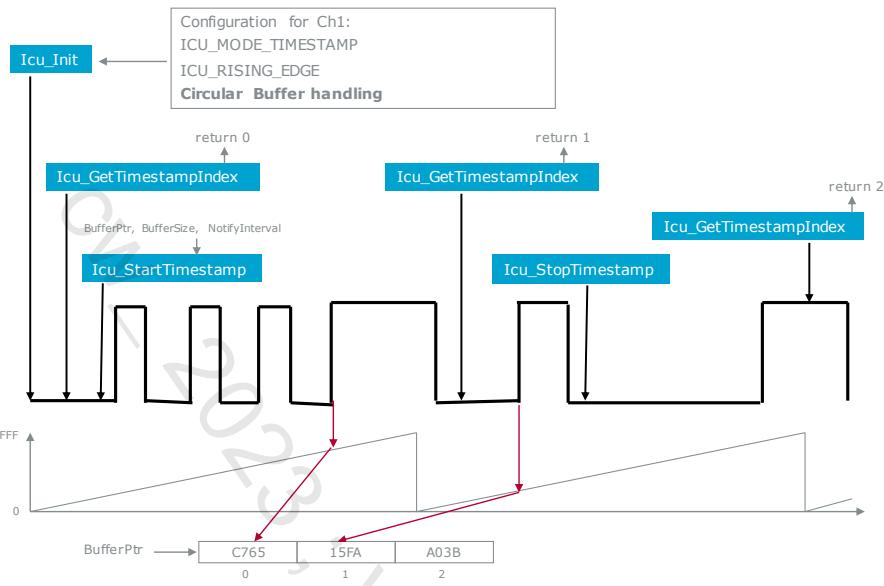
Notification function is only called if

- 1) Notification callback function is configured (it's not NULL_PTR)
- 2) Notification interval > 0
- 3) Notification enabled with Icu_EnableNotification
- 4) Number of events specified by the notification interval is captured

Timestamping – Example 2

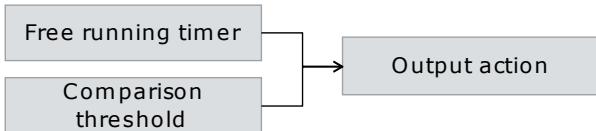


Timestamping – Example 2



Agenda

- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- DIO Driver
- PWM Driver (for reference)
- ICU Driver (for reference)
- ▶ **OCU Driver (for reference)**
- ADC Driver (for reference)
- Client Server Concept
- I/O Hardware Abstraction
- Coming up next



- ▶ Output Compare Unit
 - ▶ Start and stop a comparison process
 - ▶ Set a comparison threshold
 - ▶ Enable and disable notification mechanisms
 - ▶ Get counter values
 - ▶ Change output pin states
 - ▶ Trigger hardware resources (ADC, DMA) if available
- ▶ OCU Driver encapsulates
 - ▶ Either HW OCU Unit or just a GPT with SW comparison



Here you can define a trigger for some action by other hardware

For example, very special / sophisticated for the Infineon Tricore

Agenda

- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- DIO Driver
- PWM Driver (for reference)
- ICU Driver (for reference)
- OCU Driver (for reference)
- ▶ **ADC Driver (for reference)**
- Client Server Concept
- I/O Hardware Abstraction
- Coming up next

Characteristics

- ▶ Converts analog signals into digital signals
- ▶ Abstracts the access to converted signals
- ▶ Different conversion modes configurable
- ▶ Notification after conversion configurable

- ▶ Adc_Main_PowerTransitionManager
 - ▶ This API is cyclically called and supervises the power state transitions, checking for the readiness of the module and issuing the callbacks
IoHwAb_Adc_NotifyReadyForPowerState<Mode>



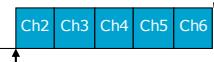
Checking if the ADC is powered by the microcontroller:

The actual Power State can be handled in the I/O HW Abstraction

Channel groups

- ▶ Grouping of ADC channels into ADC channel groups
- ▶ Channel Group contains at least one channel
- ▶ All channels in a group need to belong to the same ADC HW Unit
- ▶ Prioritization of channel groups

Conversion Modes

Channel Type	Conversion Type	Process
Multi-channel Group	Continuous	
Multi-channel Group	One-Shot	
Single-channel group	Continuous	
Single-channel group	One-Shot	

↑ Start of conversion (hardware or software triggered)

↑ Notification (if enabled)

Trigger Modes

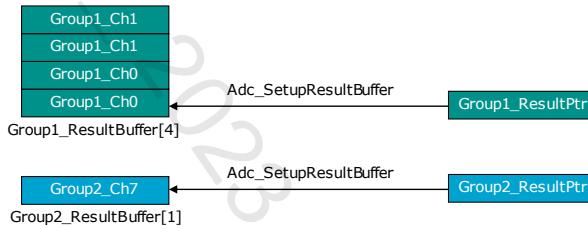
- ▶ Hardware Trigger
 - ▶ E.g. expired timer, edge detection on an I/O channel
- ▶ Software Trigger
 - ▶ Start/Stop of conversion through software API call



```
Adc_EnableHardwareTrigger  
Adc_DisableHardwareTrigger  
Adc_StartGroupConversion  
Adc_StopGroupConversion
```

Buffer initialization

- ▶ Result Pointer needs to be initialized during runtime
 - ▶ API function: `Adc_SetupResultBuffer`
- ▶ Result Buffer Size = $m \times n$
 - > m = Number of channels in group
 - > n = Number samples per channel (`ADC_STREAMING_NUM_SAMPLES`)
- ▶ ADC driver stores converted results into the Result Buffer



No direct access to ADC hardware registers!

Result pointer (pointer to the result buffer) is a pointer to a memory area specified by the application.

Number of samples for streaming is defined at pre-compile time (Parameter is 1 for Single Access Mode).

Configuration for Group 1:

Group consist of Ch0 and Ch1

Number of samples = 2

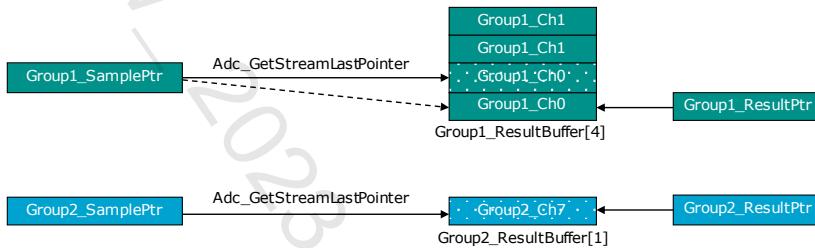
Configuration for Group 2:

Group consist of Ch7

Number of samples = 1 (implicit due to Single Access Mode)

Result Access Modes – Streaming Access

- ▶ `Adc_GetStreamLastPointer` returns the pointer to the latest converted value of the first group channel and the number of valid samples
- ▶ Stream Buffer Mode
 - ▶ Circular: Conversion continues at the beginning of the buffer if the stream buffer is full
 - ▶ Linear: Conversion is stopped if buffer is full



Result Access Modes – Streaming Access



Possible return values of `Adc_GetStreamLastPointer`:

- 0 – No valid value in result buffer
- !0 – Number of valid samples

For accessing further group channels the layout of the result buffer must be taken into account

Configuration for Group 1:

Group consist of Ch0 and Ch1

Number of samples = 2

Configuration for Group 2:

Group consist of Ch7

Number of samples = 1 (implicit due to Single Access Mode)

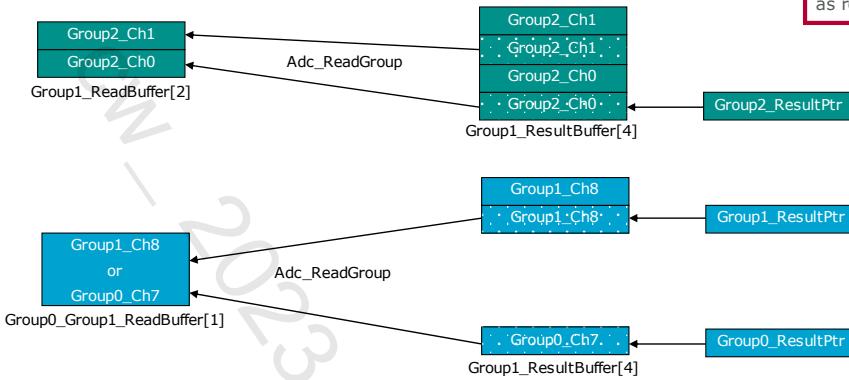
Note: the `Adc_GetStreamLastPointer` API is used for no matter what the access mode is (either Single Access or Streaming Access). In the case of Single Access, we treat it like a Streaming with buffer size 1.

Result Access Modes – Single Access

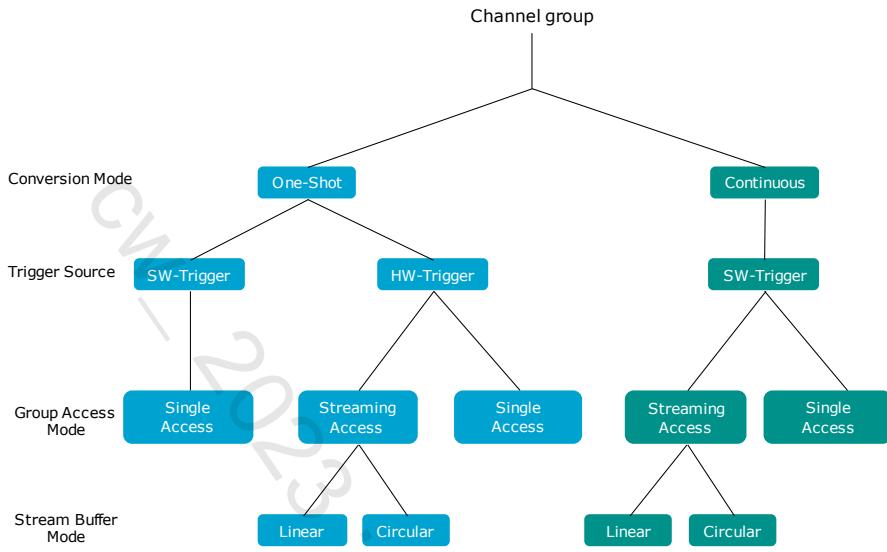
- `Adc_ReadGroup` returns the latest converted values of all channels in a group



`Group0_Group1_ReadBuffer[1]` is used as read buffer for Group 0 and Group 1



Channel group configuration options



Channel group configuration options



Pre-Compile Configuration:

```
Adc_GroupConvModeType: ADC_CONV_MODE_ONESHOT, ADC_CONV_MODE_CONTINUOUS  
Adc_TriggerSourceType: ADC_TRIGGER_SRC_SW, ADC_TRIGGER_SRC_HW  
Adc_GroupAccessModeType: ADC_ACCESS_MODE_SINGLE, ADC_ACCESS_MODE_STREAMING  
Adc_StreamBufferModeType: ADC_STREAM_BUFFER_LINEAR, ADC_STREAM_CIRCULAR
```

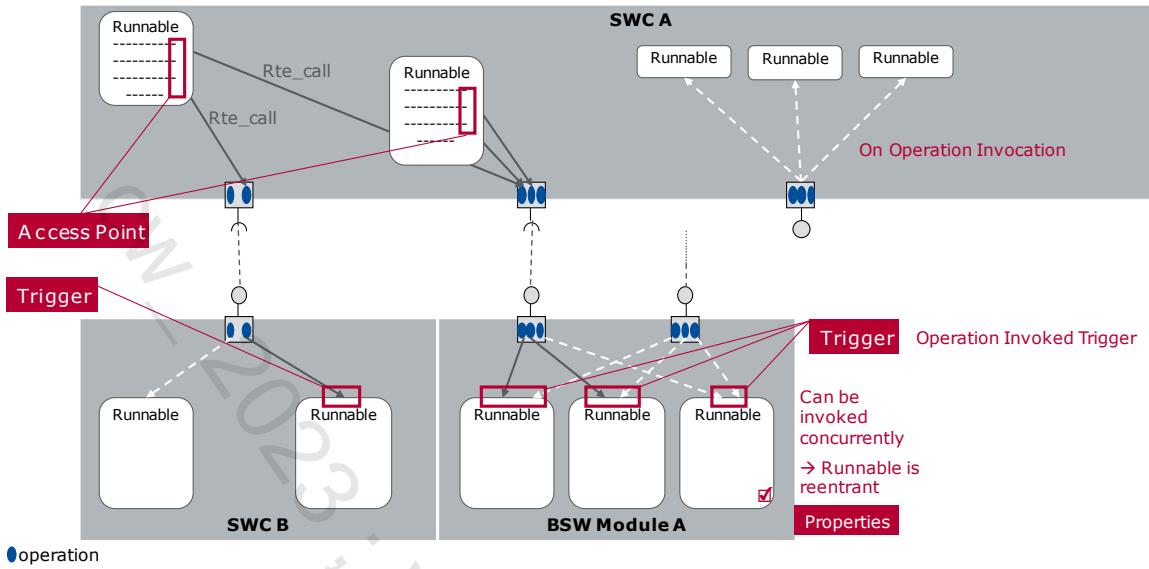
If Hardware Trigger is used:

```
Adc_HwTriggerSignalType: ADC_HW_TRIGGER_RISING_EDGE, ADC_HW_TRIGGER_FALLING_EDGE; ADC_HW_TRIGGER_BOTH_EDGE
```

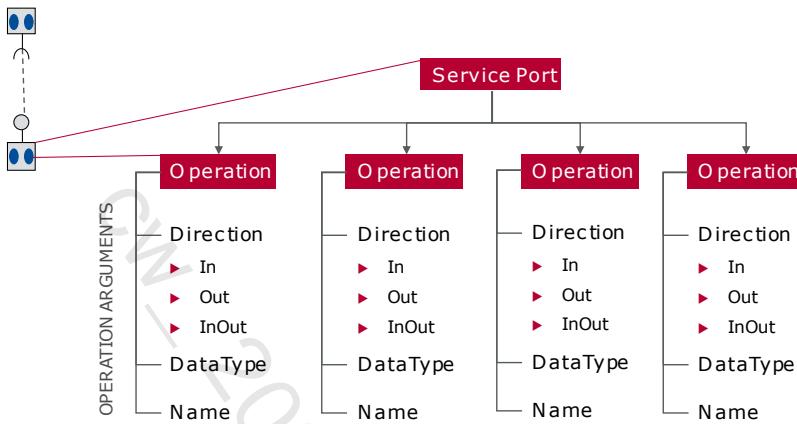
Agenda

- Architecture of I/O
- I/O Drivers – General Characteristics
- PORT Driver
- DIO Driver
- PWM Driver (for reference)
- ICU Driver (for reference)
- OCU Driver (for reference)
- ADC Driver (for reference)
- ▶ **Client Server Concept**
- I/O Hardware Abstraction
- Coming up next

Call of Server Ports



Operations



A Server has to provide one Runnable per Operation

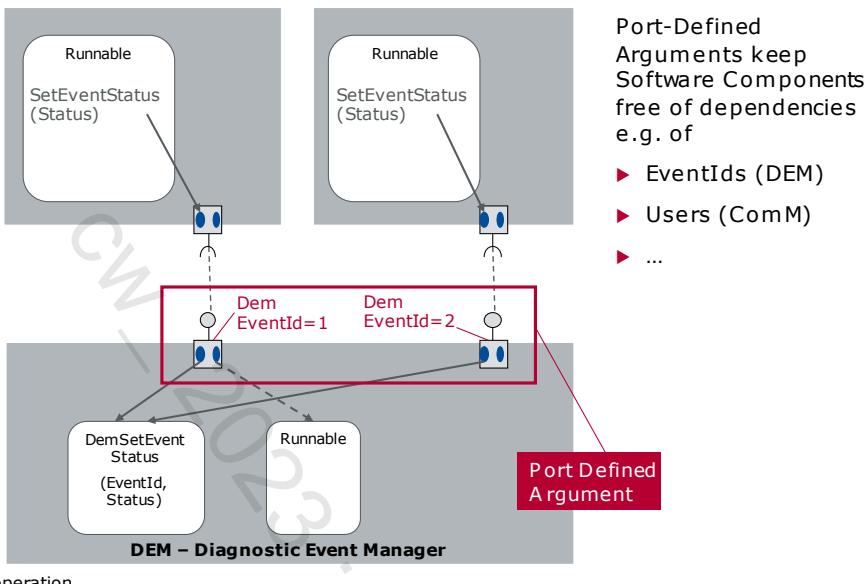
Operations

Arguments IN, IN/OUT, OUT

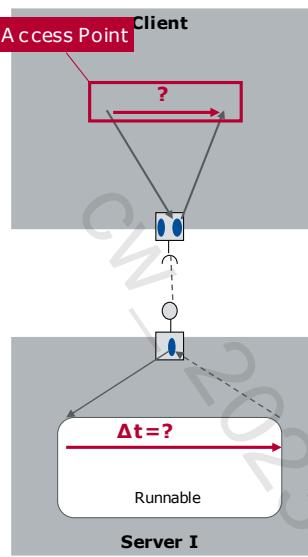
- ▶ IN
 - ▶ By value or by reference according to IDT
- ▶ OUT
 - ▶ By reference
- ▶ IN/OUT
 - ▶ Synchronous: By reference
 - ▶ Asynchronous: IN/OUT degrades to IN → by value
 - ▶ All pointers must be valid until API call returns

© 2023, VH Autosar CP Training_EN

Port-Defined Argument



Overview: Runtime of Servers



How to wait for the answer of a service (Runnable)?

- ▶ Synchronous → Timeout [unit]
- ▶ Asynchronous → Waiting → Timeout [unit]
- ▶ Asynchronous → Polling
- ▶ Asynchronous → None (trigger Runnable)

Get result using:

RequestResult on:

- ▶ Task Level
- ▶ In a triggered Runnable (if possible)

Synchronous with / without Timeout

Synchronous → Timeout [unit]

The Rte_Call

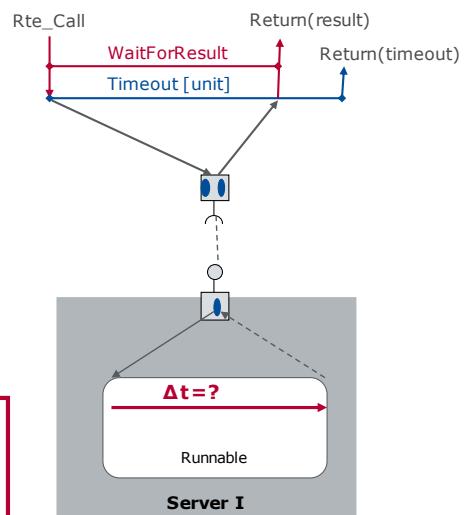
- ▶ Triggers the timeout monitoring
- ▶ Waits for the result

The return value contains:

- ▶ The result of the server when return occurs before the timeout expires
- ▶ An timeout error code as the timeout expires before the result of the server is ready



"With Timeout" requires a Task mapping (in the Operating System configuration)
 → Before activating the other task an OS alarm is started
 → When the alarm expired = TIMEOUT
 Task Mapping can be configured:
 -Consumes more runtime



Asynchronous with Timeout

Asynchronous → Waiting → Timeout [unit]

The Rte_Call

- ▶ Triggers the server (Runnable) and returns immediately

The Rte_Result

- ▶ Triggers the timeout monitoring

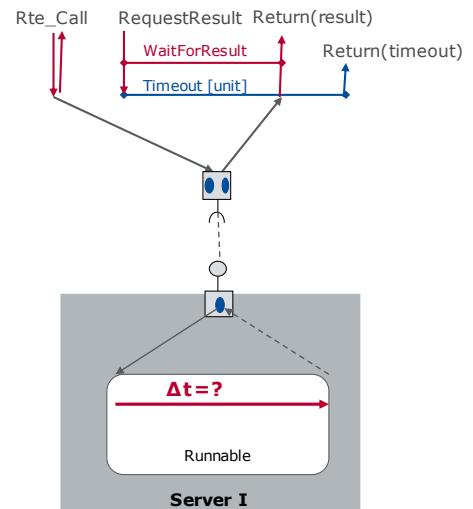
- ▶ Waits for the result

The return value contains:

- ▶ The result of the server when return occurs before the timeout expires
- ▶ A timeout error code if the timeout expires before the result of the server is ready



1. Rte_Call triggers an RTE / OS event
2. Rte_Result (waiting) will activate an alarm for timeout monitoring:
AND call WaitEvent (waiting point until the Server runnable has finished)
3. Server runnable has finished and sets the event for the RTE_RESULT



Asynchronous and Polling

Asynchronous → Polling

The Rte_Call

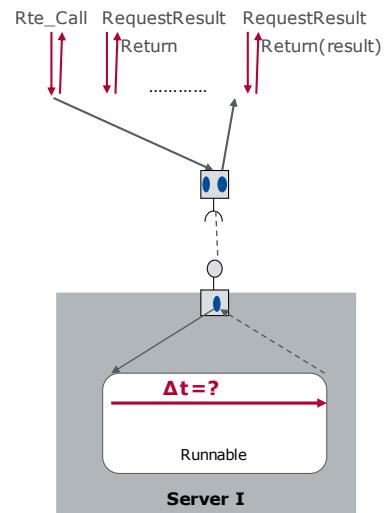
- ▶ Triggers the server (Runnable) and returns immediately

The Rte_Result

- ▶ Sees if the result is already there

The return value contains:

- ▶ The result of the server if present
- ▶ Information that result is not ready yet



Asynchronous and Trigger on Runnable

Asynchronous → None

The Rte_Call

- ▶ Triggers the server (Runnable) and returns immediately

The Result is present

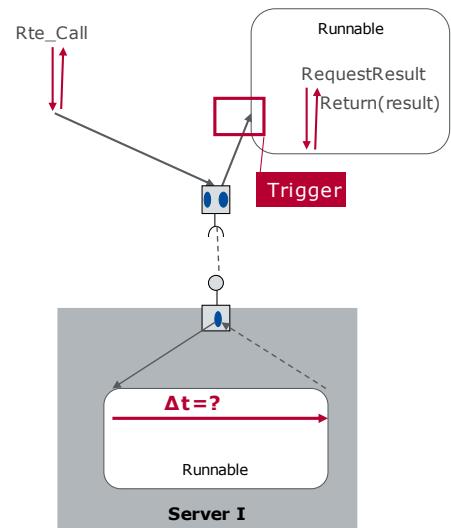
- ▶ The assigned Runnable is triggered

Get result

- ▶ As the result triggers the Runnable, the result is now present
- ▶ To get the result a an Rte_Result is necessary in the triggered Runnable



1. Rte_Call sets an Event for the Server Runnable
2. When the Server Runnable has finished, another Event will be set to notify the configured client Runnable
3. Everything should be configured already through the RTE Runnable Triggers



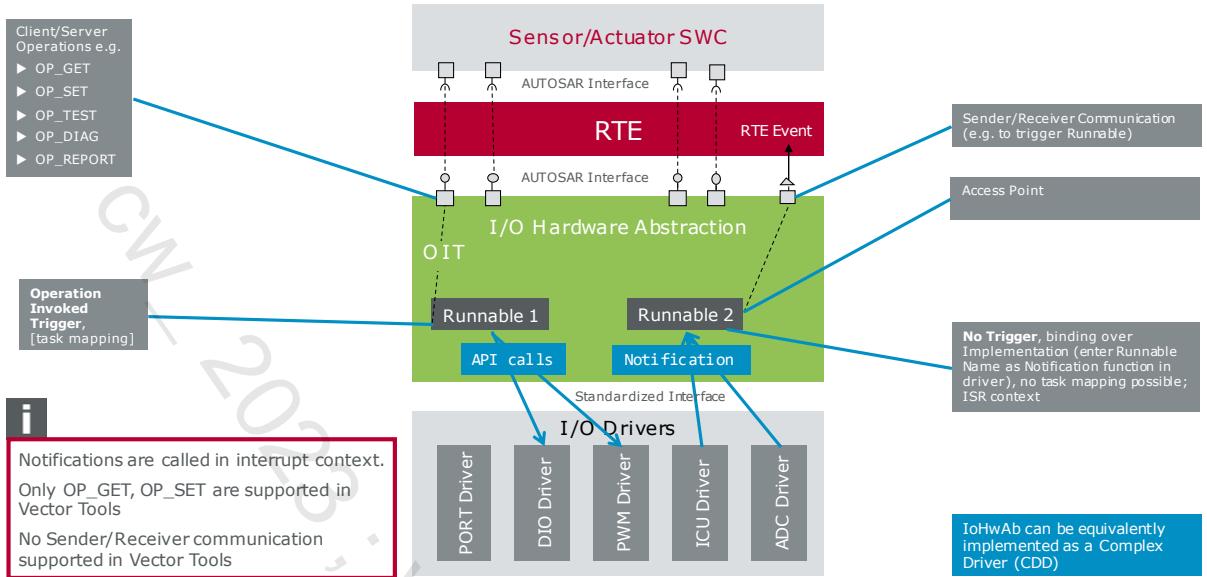
Agenda

Architecture of I/O
I/O Drivers – General Characteristics
PORT Driver
DIO Driver
PWM Driver (for reference)
ICU Driver (for reference)
OCU Driver (for reference)
ADC Driver (for reference)
Client Server Concept

► **I/O Hardware Abstraction**

Coming up next

Implementation of the IO HwAb

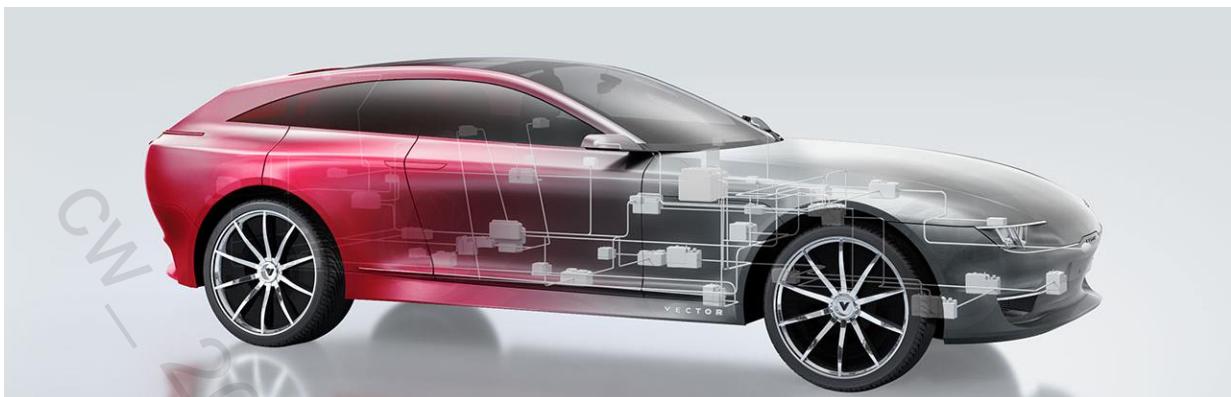


ECU Degradation concept

- ▶ Power State Transitions
 - ▶ Map ECU Power Modes to peripheral Power Modes over respective Power State API
- ▶ I/O HW Abstraction (IoHwAb) shall be able to
 1. Prepare the switch into certain Power State
 2. Execute transitions into the desired Target ECU Power State
 - ▶ This separation shall render it useless to poll for allowed transitions.
- ▶ Examples for modules supporting this concept:
 - ▶ ADC
 - ▶ PWM

I/O

Exercise: I/O



AUTOSAR in Practice

Exercise 2: I/O

V28 | 2023-03-28

Agenda

- ▶ **Exercise 2 - I/O**

coming up next

CW_2023_VH_Autosar CP Training_EN

2

I/O

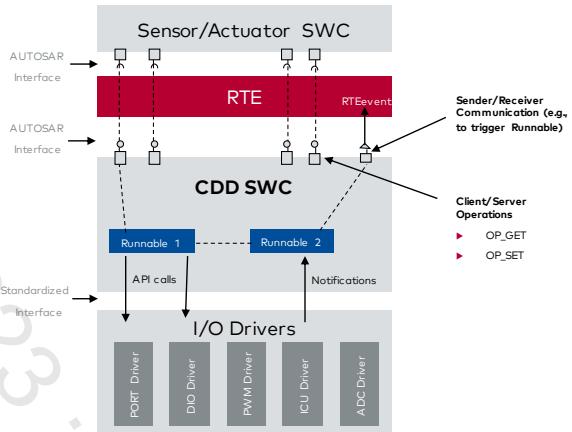
Now we will extend the first example. Configure the I/O Dio module to have access to the CANoe panel sensors `DoorFrontLeft` and `DoorFrontRight` and to the actuator `InteriorLightFront` (access via CANoe Environment Variables).

Create a Complex Device Driver (Cdd) SWC to provide access to the Dio module to the application.

Program the logic that connects opening front doors with switching on the front interior light in the corresponding Runnables.

i

For granting access to the DIO module for the application a Cdd Software Component will be used. The advantage of this approach is that the Cdd SWC can be designed completely as SWC and is more flexible.

Overview:

2**Exercise – I/O**

- **1** Configure I/O Driver (Dio)
- **2** Create a Cdd SWC and add it to the Software Design
- **3** Generate, program Runnables, and test it
- **4** Add-on

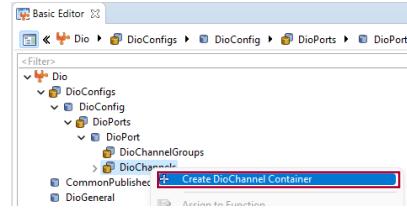
Part 1/4

2

Configure Dio and Hardware Abstraction

Please stick exactly to the given names
as the CANoe simulation needs
precisely these names to work
properly.

- ▶ "Open with > DaVinci Project Assistant" using **MyECU.dpa** file in folder **\E2_InputOutput**
- ▶ Open the **Basic Editor** (see notes section on next slide)
- ▶ select **Dio→DioConfig→DioPorts→DioPort→DioChannels** and create 3 new **DioChannels**.
- ▶ Use the following ShortNames for the new channels
 - ▶ **env_InteriorLightFront** ChannelId: 0
 - ▶ **env_DoorFrontLeft** ChannelId: 1
 - ▶ **env_DoorFrontRight** ChannelId: 2
- ▶ save the project , close DaVinci Configurator



Select DIO and create 3 DioChannels

	DioChannels	Channel Id
	env_InteriorLightFront	0
	env_DoorFrontLeft	1
	env_DoorFrontRight	2

Expected result: Completed list of DIO channels.

Part 1/4

i

The screenshot shows the 'Basic Editor' interface. On the left, there's a sidebar with 'Configuration Editors' and a list of project modules: Base Services, Communication, Diagnostics, Memory, Mode Management, Network Management, Runtime System, and Timing. Below this is a 'Basic Editor' button with a red border. The main area has a 'Project' menu with options like 'On-demand generator validation' (F8), 'Generate' (F9), 'Build VTT Project', 'Report...', 'Project Settings', 'Input Files', and 'Basic Editor' (which is highlighted with a red border). At the bottom is a 'Switch Configuration Phase...' button. To the right is a tree view titled 'Basic Editor' under 'Dio'. It shows 'DioConfigs' containing 'DioPort' which has 'DioPorts' and 'DioChannelGroups'. A context menu is open over 'DioPorts' with an option 'Create DioChannel Container' highlighted with a red border. Other menu items include 'CommonPublished...', 'DioGeneral', and 'Assign to Function'.

Basic Editor selection

Dio: In the Dio configuration there is one channel predefined. Overwrite this channel. Add all other channels using the "+" icon or right-click "Create DioChannel container." Add unique ChannelIds.

Dio configuration view: Appending Dio channels

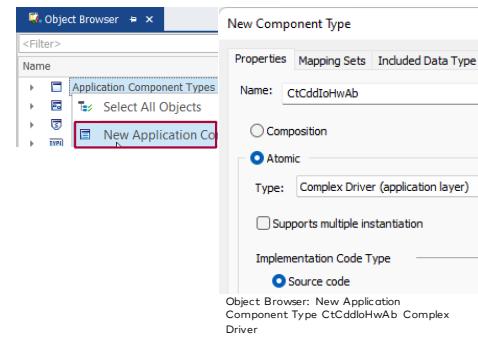
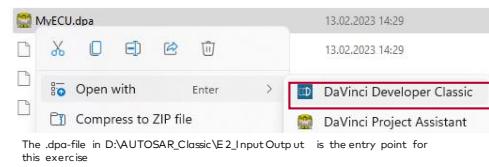
Part 2/4

2

Create Cdd SWC and extend Software Design

- ▶ "Open with > DaVinci Developer Classic" using MyECU.dpa file in folder \E2_InputOutput\ (see right)

- ▶ Go to the Developer Object Browser
- ▶ **Application Component Types**
- ▶ Create a new Application Component Type
 - ▶ Name: CtCddIoHwAb
 - ▶ **Complex Driver** (application layer)
- ▶ Create an **Implementation Data Type**
 - ▶ New Type Reference
 - ▶ Name: IdtDioValueType
 - ▶ Data Type: 'uint8' Platform Type



Part 2/4

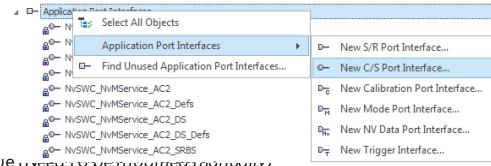
2

Create Cdd SWC and extend the Software Design

- ▶ Create **C/S Port Interfaces** as Application Port Interfaces (see right figure and next slide)

▶ **PiDoorIoHwAb**

- ▶ Application Errors: E_NOT_OK (Code = 1)
- ▶ Operation Prototypes: **ReadChannel**
- ▶ Argument Prototypes of Read Channel operation:
 - ▶ Direction: Out; Data Type: IdtDioValueType; Name: value (*Need to be modified manually*)

▶ **PiLightIoHwAb**

- ▶ Application Errors: E_NOT_OK (Code = 1)
- ▶ Operation Prototypes: **WriteChannel**
- ▶ Argument Prototypes of Write Channel operation:
 - ▶ Direction: In; Data Type: IdtDioValueType; Name: value (*Need to be modified manually*)

Please edit the already existing operation instead of creating a new one (see next slide)

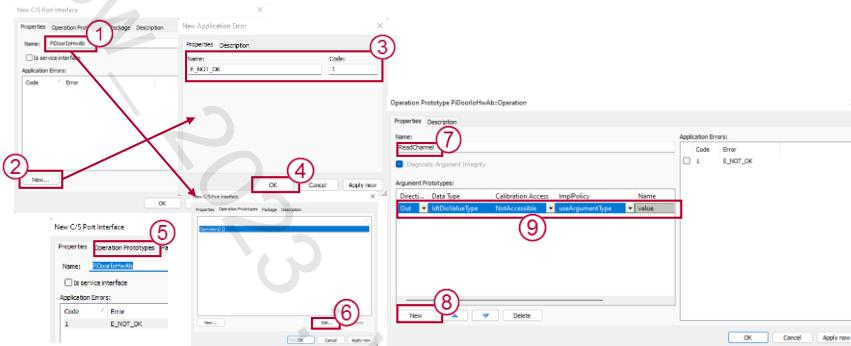


Part 2/4

**C/S Port Interfaces**

A Client / Server connection always consists of one or more Client port(s) and a Server Port. The Server side provides some functionality which can be triggered by the Client side.

Therefore, the port prototypes have to define operations which can be triggered by Client side and are provided by Server side. Within the CtCddIoHwAb, it will be the possibility to read or write a Dio Channel. An operation can always be defined with some input / output parameters and a Return Value in terms of an Error code.

Creation of a Client Server Connection:

- (1) Define C/S Interface **name**
- (2) Open **New Application Error** dialog
- (3) Define the **Application Error**
- (4) Confirm with **[OK]**
- (5) Open **Operation Prototypes** dialog
- (6) Edit the existing Operation via **[Edit...]**
- (7) Define the **Operation name**
- (8) Create New Argument Prototype
- (9) Define the **Argument Prototypes**

Part 2/4

2

Create Cdd SWC and extend Software Design

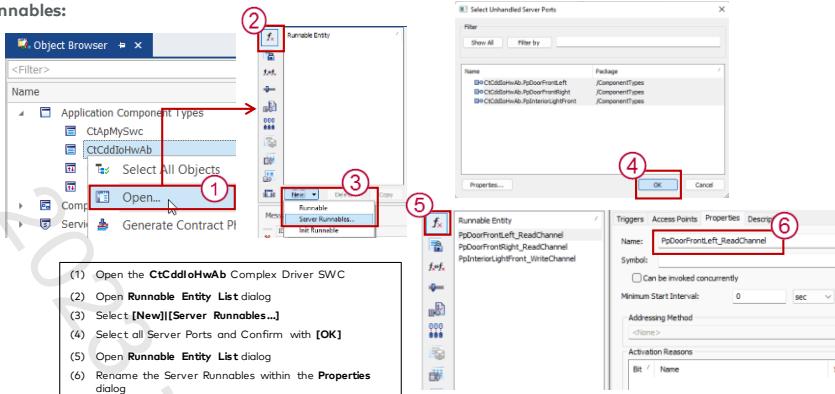
- ▶ Create the following **Port Prototypes** within the CtCddIoHwAb Complex Driver:
 - ▶ Create **Server** Port Prototype **PpDoorFrontLeft**
 - ▶ Type: **PiDoorIoHwAb**
 - ▶ Create **Server** Port Prototype **PpDoorFrontRight**
 - ▶ Type: **PiDoorIoHwAb**
 - ▶ Create **Server** Port Prototype **PpInteriorLightFront**
 - ▶ Type: **PiLightIoHwAb**
- ▶ Create the following **Server Runnables** within the CtCddIoHwAb Complex Driver (see section on next slide):
 - ▶ Create **Server** Runnable **RCtCddIoHwAb_PpDoorFrontLeft_ReadChannel**
 - ▶ Connected to Server Port **PpDoorFrontLeft**
 - ▶ Create **Server** Runnable **RCtCddIoHwAb_PpDoorFrontRight_ReadChannel**
 - ▶ Connected to Server Port **PpDoorFrontRight**
 - ▶ Create **Server** Runnable **RCtCddIoHwAb_PpInteriorLightFront_WriteChannel**
 - ▶ Connected to Server Port **PpInteriorLightFront**

Part 2/4

i

Server Ports and Server Runnables: An **operation** of a Server Port assigned to a SWC must **always** be connected to a **Server Runnable** within this SWC. This Server Runnable will be executed when the Client Port (of the C/S connection) accesses the Server Port. Therefore, the Server Runnable will trigger by an **On Operation Invocation Trigger Event** which provides the connection to the corresponding Server Port of the SWC.

Creation of Server Runnables:

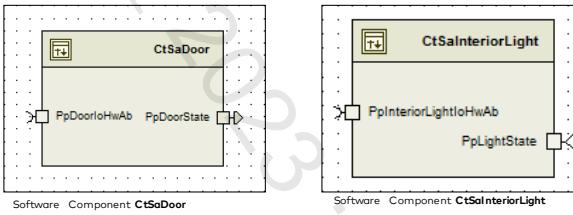


Part 2/4

2

Create Cdd SWC and extend the Software Design

- ▶ Create the following Port Prototypes in the given SWCs:
 - ▶ SWC Type CtSaDoor: Create **Client** Port Prototype **PpDoorIoHwAb**
 - ▶ Type: **PiDoorIoHwAb**
 - ▶ SWC Type CtSaInteriorLight: Create **Client** Port Prototype **PpInteriorLightIoHwAb**
 - ▶ Type: **PiLightIoHwAb**
- ▶ Add the required Access Point for each Runnable inside CtSaDoor and CtSaInteriorLight



Required Access Point: Think about the needed operations you have to call (Access Points Tab: **Invoke operation...**)

The **CtSaDoor** component has the duty to **get** the value of the DIO channel (ReadChannel) tied to the door sensor hardware. The **CtSaInteriorLight** component has the job to **set** the DIO channel with the appropriate value such that the Light will be switched on or off (WriteChannel). An access point is granted for a **Runnable** and allows the Runnable to access the Port Prototypes of the SWC to access information from external SWCs.

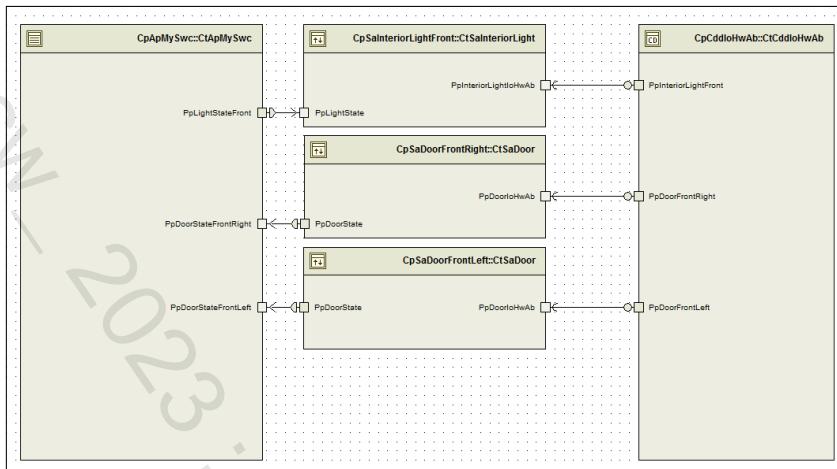
Part 2/4

2

Create Cdd SWC and extend Software Design

- ▶ Instantiate the **CtCddIoHwAb** Complex Driver SWC within the CtCoApplication Composition and rename it to **CpCddIoHwAb**
- ▶ Connect the Client Ports from **CpSaInteriorLightFront**, **CpSaDoorFrontRight**, **CpSaDoorFrontLeft** with the Server Ports from **CtCddIoHwAb** (see notes section on next slide)
- ▶ Save  and exit DaVinci Developer

Part 2/4

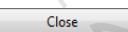
i**CpCoApplication Composition:**

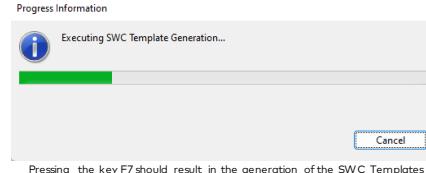
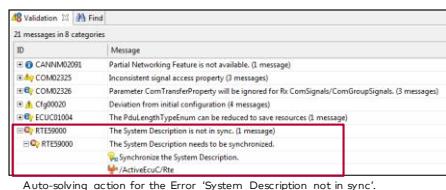
Part 3/4

2

Generation, program Runnables and test it

- ▶ "Open with > DaVinci Project Assistant" using **MyECU.dpa** file in folder **\E2\InputOutput**
- ▶ Solve the RTE error: "The System Description is not in sync." by clicking on the solving action  (see right)
- ▶ **Configurator:** Generate 
- ▶ Click "OK" to close the VTT Dialog
- ▶ Close the Generation dialog


- ▶ Press F7 to generate the SWC Templates



Part 3/4

2

Generation, program Runnables and test it

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder \E2_InputOutput\..\Solution
- ▶ In the SWC Type CtSaDoor, adapt the Runnable RCtSaDoorReadDoor to read in information from I/O (door state) and send it to CtApMySWC
- ▶ Adapt Runnable for CtSaInteriorLight to send information about interior light state to IoHwAb
- ▶ Adapt newly generated CtCddIoHwAb.c template file in folder \E2_InputOutput\App\Source
(see notes section on next slide)
 - ▶ Dio.h module has to be included (has already been done for this exercise)
 - ▶ The Server Runnables have to be implemented. You need to read/write values from/to the DIO module
- ▶ Build ECU code
- ▶ Start CANoe, switch back to Visual Studio, then press F5 to attach to the process RuntimeKernel.exe (Native)
- ▶ switch back to CANoe, then press F9
- ▶ Test your code

Part 3/4



Adapt CtCddIoHwAb.c.

In the Software Component Template generation process, a file called CtCddIoHwAb was generated.

Add the following line between the comments "<< Start of include and declaration area >>" and "<< End of include and declaration area >>"

```
#include "Dio.h"
```

At the moment, this file has no functionality inside of it, but only provides interfaces and templates for the Server Runnables configured within DaVinci developer. The bodies of these functions (i.e., the connections to the Dio module) have to be provided by the integrator. The following functions have to be provided:

- RCtCddIoHwAb_PpDoorFrontLeft_ReadChannel(value)
 - *value = Dio_ReadChannel(DioConf_DioChannel_env_DoorFrontLeft);
- RCtCddIoHwAb_PpDoorFrontRight_ReadChannel(value)
 - *value = Dio_ReadChannel(DioConf_DioChannel_env_DoorFrontRight);
- RCtCddIoHwAb_PpInteriorLightFront_WriteChannel(value)
 - Dio_WriteChannel(DioConf_DioChannel_env_InteriorLightFront, value);

Part 4/4 (Add-On only)

2

Add-On

- ▶ This exercise step is an add-on. Do this part only if you have finished the rest.
- ▶ This add-on simulates a DEFECT of the front interior light.
- ▶ **Configurator**
 - ▶ Dio: Add a **DioChannel** named **env_InteriorLightFrontDEFECT**
 - ▶ Save 
- ▶ **DaVinci Developer**
 - ▶ Provide your CtCddIoHwAb component with necessary IO port prototypes (Server port) and create a new Server Runnable
 - ▶ Provide your SA component with necessary IO port prototypes (Client port) and provideRunnables of CtSaInteriorLight with necessary access point.
 - ▶ Connect the new Client Port CtSaInteriorLight with the new Server Port from CtCddIoHwAb. Save .
 - ▶ Return to Configurator and generate , regenerate the SWC Templates.
- ▶ **Visual Studio** (see notes section on next slide)
- ▶ Poll the DEFECT state from the hardware. The front light shall stay off if DEFECT is set.

Part 4/4 (Add-On only)



Provide CtSalteriorLight components with necessary IO port prototypes (C/S ports):

Create a unique C/S Port Interface PiLightDefectIoHwAb with the following operation:

- ReadChannel(value)
 - Argument Prototype: value (Direction: Out, DataType: IdtDioValueType)

Add a new Client Port Prototype PplInteriorLightDefect (Port Interface PiLightDefectIoHwAb) to the SWC Type CtSalteriorLight.

Make sure that the Runnable inside the SWC Type has an Invoke Operation Access Point to the operation ReadChannel inside the Port Prototype.

Add a new Server Port Prototype PplInteriorLightFrontDEFECT (Port Interface PiLightDefectIoHwAb) to the Complex Driver SWC Type CtCddIoHwAb.

Add a new Server Runnable RCtCddIoHwAb_PplInteriorLightFrontDEFECT_ReadChannel connected to the new Server Port to the CtCddIoHwAb Complex Driver SWC.

Implement Code within Visual Studio

- Open Visual Studio Project via MyEcu solution file
- Implement Handling for DEFECT switch in CtSalteriorLight
- Add handling of Dio channel via Complex Driver SWC (CtCddIoHwAb.c)
 - RCtCddIoHwAb_PplInteriorLightFrontDEFECT_ReadChannel(value)
 - *value = Dio_ReadChannel(DioConf_DioChannel_env_InteriorLightFrontDEFECT);

Summary

2**Summary**

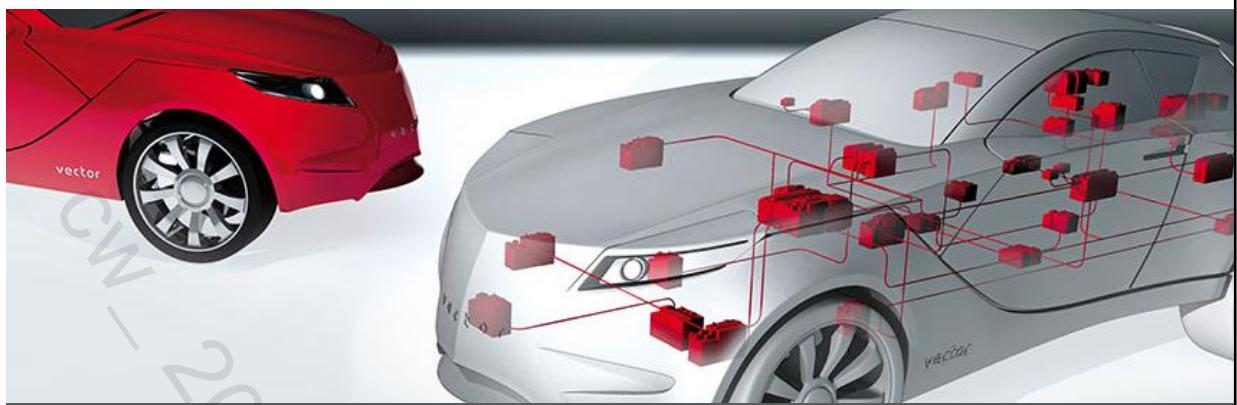
Now take a little time to think about what you have done in this exercise and why.

- ▶ Think about the difference between Sender/Receiver and Client/Server Ports.
- ▶ ... how do we have to write the information (set value of InteriorLightFront)?
 - ▶ Why is the port for our Runnable a Client port?
 - ▶ ...how do you access Client/Server Ports?

i**Client Port if you have to write information:**

The server provides the services ReadChannel and WriteChannel. To use the WriteChannel operation that is provided by a server, for example, you have to be a client.





AUTOSAR in Practice

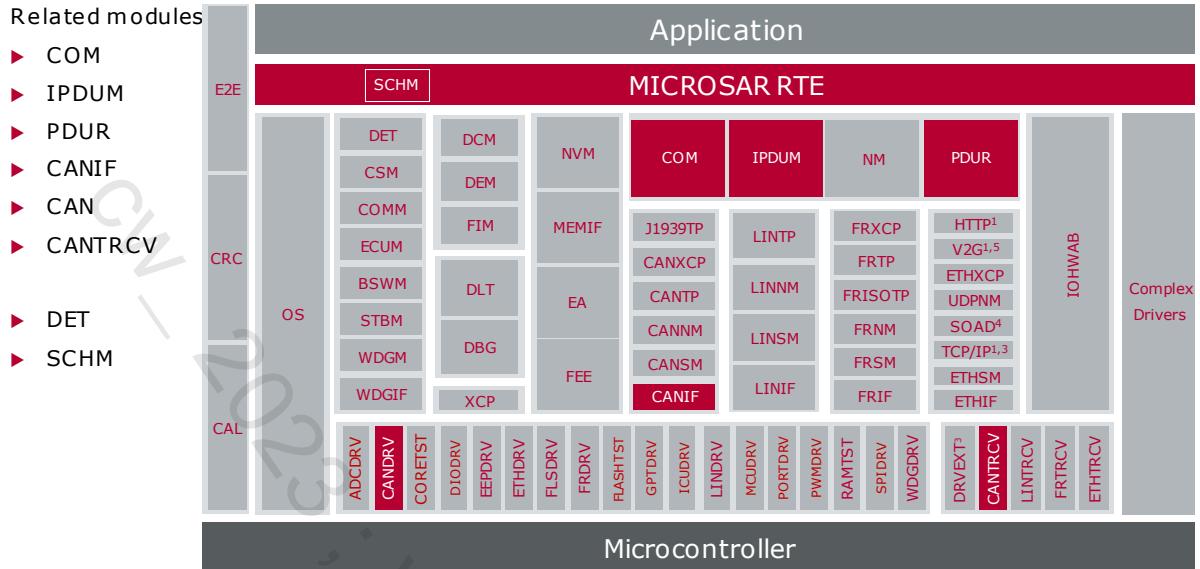
Communication

V1.0.0 | 2020-01-20

Agenda

► **Communication**

- Transmission
- Signal Groups
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- Invalidation
- Rx Filter
- Additional Return Values
- Coming up next



How do communication modules interact

COM

- ▶ Signal interface to RTE
- ▶ Signal gateway
- ▶ I-PDU transmission and reception
 - ▶ Periodic, on event, etc.
- ▶ Deadline monitoring
- ▶ Notifications to RTE
- ▶ Signal Invalidation

PDU Router

- ▶ Abstraction of the bus system
- ▶ Interface Layer I-PDU gateway
- ▶ TP Layer I-PDU gateway

CAN Interface

- ▶ HW independent
- ▶ Queuing
- ▶ Handling of CAN controller states

CAN Driver

- ▶ HW abstraction
- ▶ Controller initialization
- ▶ ISR implementation
- ▶ Writing/reading of L-PDU data from/to hardware
- ▶ Wake-up and bus-off handling



COM – Communication
PDUR – PDU Router
CANIF – CAN Interface
CAN – CAN Driver
DET – Development Error Tracer
SCHM – Schedule Manager

How do communication modules interact

**COM:**

Com operates on I-PDUs and signals, but has no awareness of the underlying bus technology. In AUTOSAR 4.x additional features have been added:

- large data types with max PDU size 4095 Mbytes. Data type UINT8_N.
- dynamic length signals and I-PDUs using the TP (varying size during runtime, max length defined at configuration time, data type UINT8_DYN)

NOT supported in MICROSAR: float data type, data sequence (ComIPduCounter, discarding and notification of violations over ComIPduCounterErrorNotification), communication protection (fan-in with voting)

PDU Router:

The PDU Router is responsible for routing (including multi-cast routing) of messages to the same type of bus system (CAN – CAN) or different bus systems (e.g. CAN – FlexRay) and between service layer modules (Com/Dcm) and transport protocol modules.

CAN interface:

Handles all CAN channels in the system.

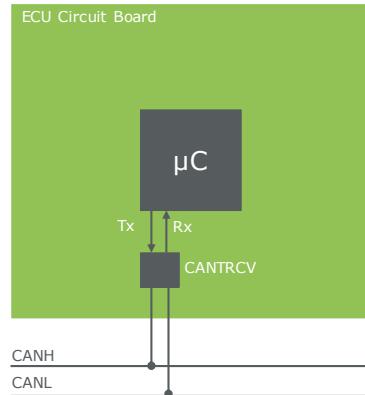
CAN Driver:

- Wakeup is reported to EcuM
- Bus-off state is reported to CanIf which forwards the state to CanSm

How do communication modules interact

CANTRCV

- ▶ Convert unidirectional signal µC pins into differential signals on the CAN bus
 - ▶ RxD, TxD \leftrightarrow CAN_H, CAN_L
- ▶ Bus biasing, bus driver control, line error detection, voltage control, wakeup detection
- ▶ Selective wakeup for partial networking (including asynchronous transceiver handling)



DET

- ▶ Collects all detected errors of the BSW modules
- ▶ For development only

SCHM

- ▶ Embeds BSW module implementations into AUTOSAR OS context
- ▶ Triggers main processing functions of the BSW modules
- ▶ Offers data consistency mechanisms for the BSW modules

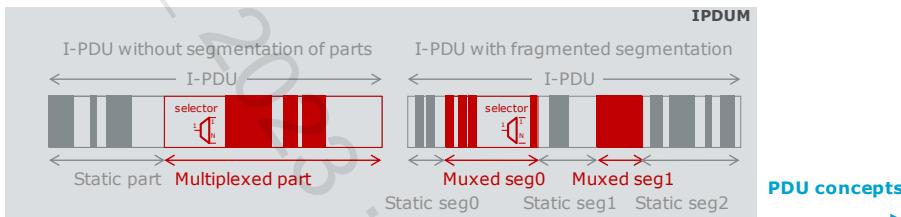


The RTE generator is responsible for the creation of the SCHM code. Similarly to how the RTE generates code to execute SWC Runnable Entities, it can do the same thing for the Basic Software Schedulable Entities (main functions). The RTE schedules them according to the internal behavior (IB) of the BSW Modules.

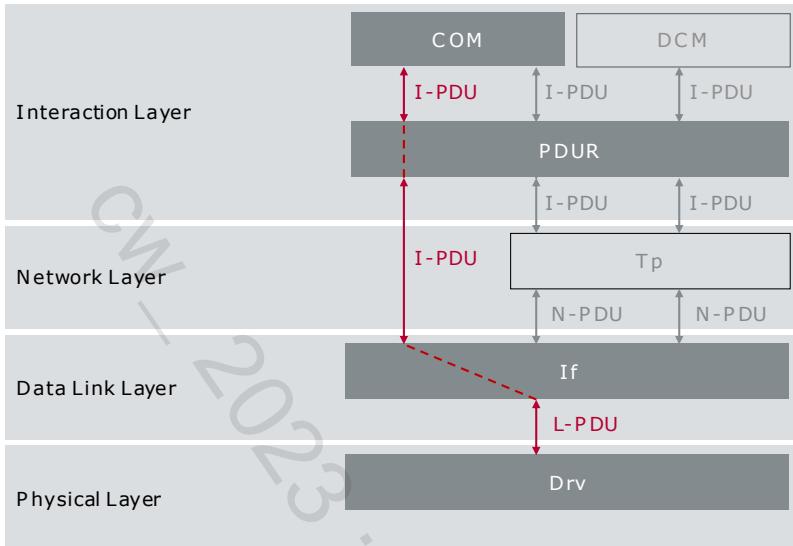
Optional module

IPDUM

- ▶ Multiplexed I-PDUs
 - ▶ Handle Tx/Rx I-PDU with multiple different signal layouts
 - ▶ Payload: zero or more signals or signal groups
 - ▶ Static and multiplexed part both contain payload and are flexibly segmented
 - ▶ Multiplexed part contains selector for multiplexed layout
- ▶ Usually used for CAN
 - ▶ LIN and FR also conceivable



PDU concept

**i**

L-PDU: Link Layer Protocol Data Unit

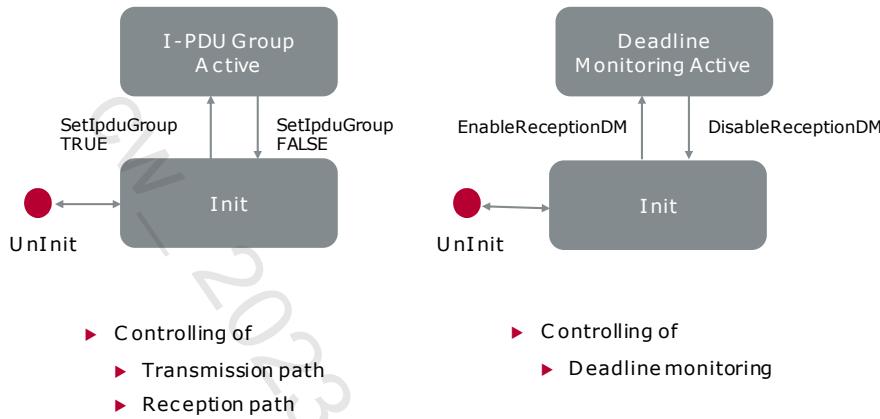
N-PDU: Network Layer Protocol Data Unit

I-PDU: Interaction Layer Protocol Data Unit

DCM: Diagnostic Communication Manager (introduced in chapter "Diagnostics")

COM I-PDU Groups

- ▶ States of I-PDU Groups



COM I-PDU Groups



-I-PDU Groups allow grouping of I-PDUs to channel-specific groups. The Bus State Managers (e.g. CanSm) switch the corresponding channel states and notify the BswM. The BswM uses the channel specific groups to control the bus communication based on these notifications.

- The EcuM initializes the Com and allows to perform the transition in the above state diagrams from "UnInit" to "Init" state.
- The activation and deactivation of I-PDU Groups and Deadline Monitoring are handled independently, as shown in the two state machines.

-In AUTOSAR 4, it is up to the **BswM** to set the I-PDU Groups and to control them. For this the APIs `Com_SetIpduGroup()` and `Com_IpduGroupControl()` are used.

CI - 2023 ; VH Autosar CP Training_EN

Agenda

Communication

► **Transmission**

Signal Groups

Update Bit

Reception

Data Mapping

Notification Mechanisms

Deadline Monitoring

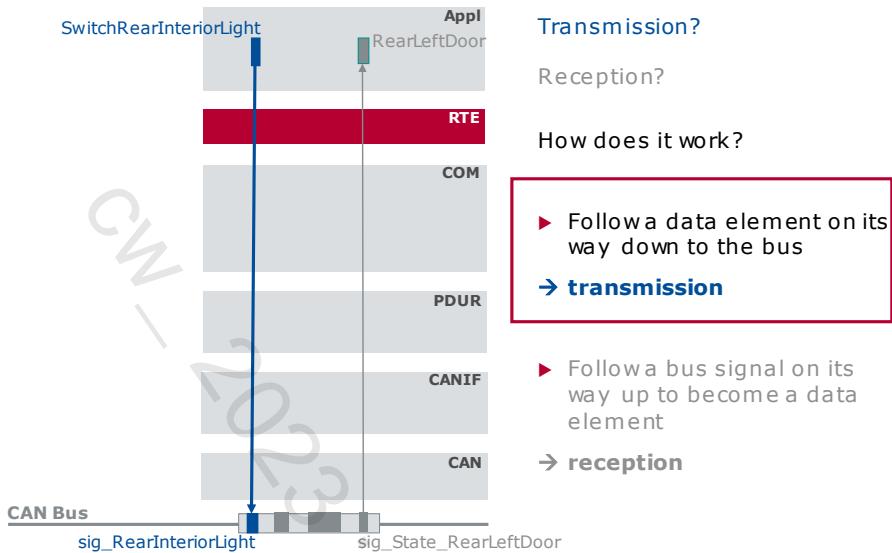
Invalidation

Rx Filter

Additional Return Values

Coming up next

Transmission and Reception



Signal transmission: Appl, RTE, COM

RTE Direct / Explicit Write

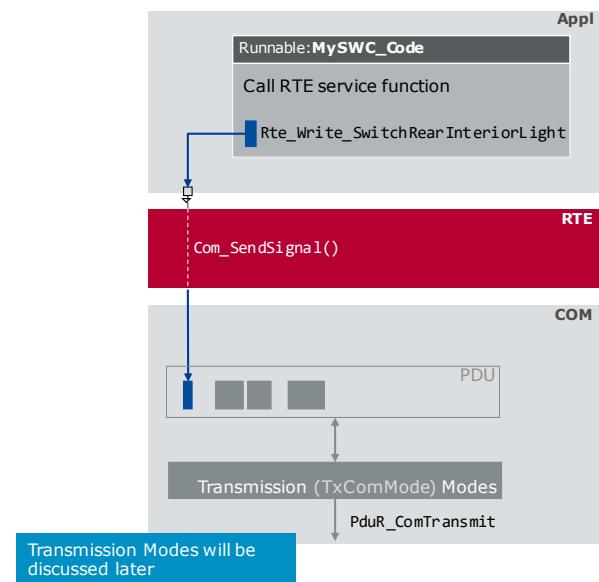
- The information is **immediately** copied to buffer in Com.

RTE Buffered / Implicit Write

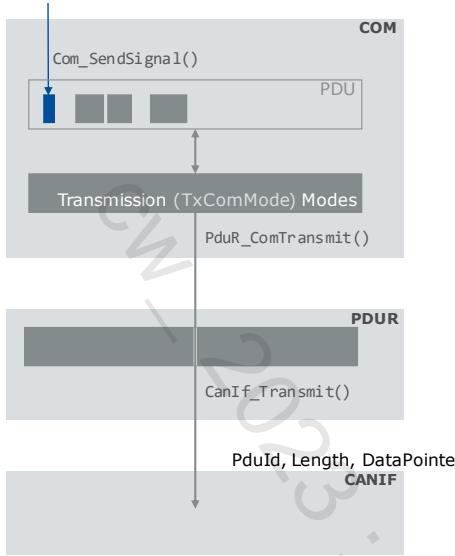
- The information is copied to buffer in Com **after** runnable is finished.

Com Transmission Modes

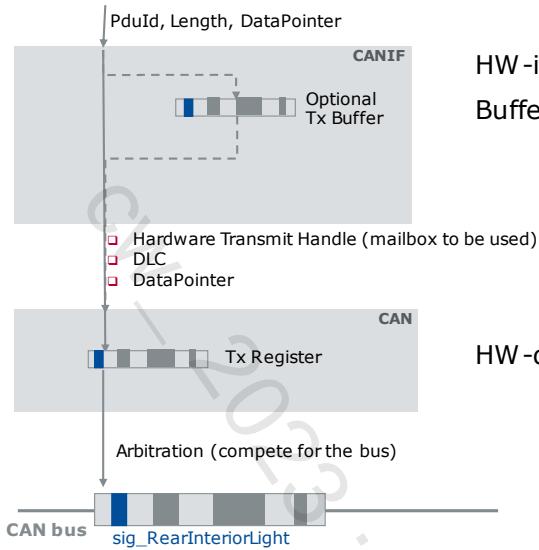
- Two Tx modes per I-PDU
- Which mode is used is based on signal filters



Signal transmission: COM, PDUR, CANIF



Signal transmission: CANIF, CAN, Bus



HW-independent
Buffer for transmit data

HW-dependent

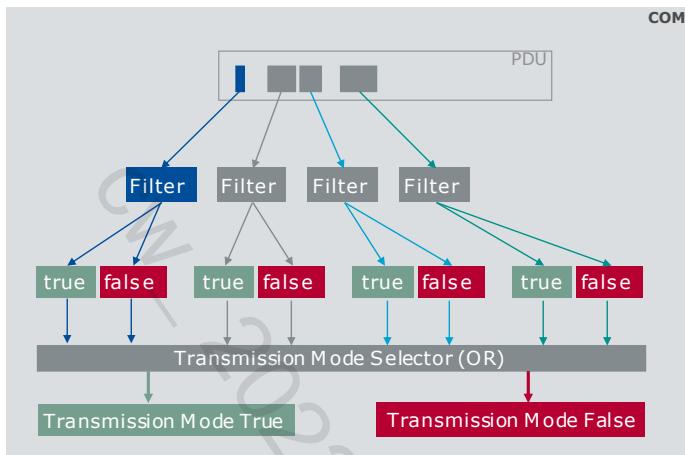


A Tx Buffer on CanIf level is not required if FullCAN is used or ComRetryFailedTransmitRequests == TRUE.

ComRetryFailedTransmitRequests is new in AUTOSAR 4.x. However in Vector Com it is already implemented since version 3.x.

In the case where ComRetryFailedTransmitRequests == TRUE, the Com evaluates the return value of PduR_ComTransmit and in case of a rejected transmission, the Com will call PduR_ComTransmit during the next Com_MainfunctionTx call. Only the repeated transmission attempt can be delayed; neither Tx Mode nor regular repetitions are affected by the retries.

Transmission Modes - Mechanism



What about
the
FILTERS?



In the above picture, only TX filters are shown. Nevertheless there are also RX filters (see next slides)

Transmission Modes – COM Signal Filters

Available signal-oriented Filters - Part 1

Filter

► **None:**

The signal has no filter

► **Always:**

This filter always evaluates to TRUE
→ No Tx mode "False" required

► **Never:**

This filter always evaluates to FALSE

► **MaskedNewDiffersMaskedOld:**

$((\text{new} \& \text{mask}) \neq (\text{old} \& \text{mask}))$

oldValue:0000 1010 newValue:1011 1001

mask = 0000 1111 → true

mask = 0000 1100 → false

Transmission Modes – COM Signal Filters

Available signal-oriented Filters Part 2

Filter

► **MaskedNewEqualsX:**

((new & mask) == x)
newValue:1011 1001 X=9

mask = 0000 1111 → true
mask = 1001 0000 → false

► **MaskedNewDiffersX:**

((new & mask) != x)
newValue:1011 1001 X=9

mask = 0000 1111 → false
mask = 1001 0000 → true

► **NewIsOutside:**

((new < min) || (max < new))
min = 10, max = 103

newValue = 10 → false
newValue = 254 → true

► **NewIsWithin:**

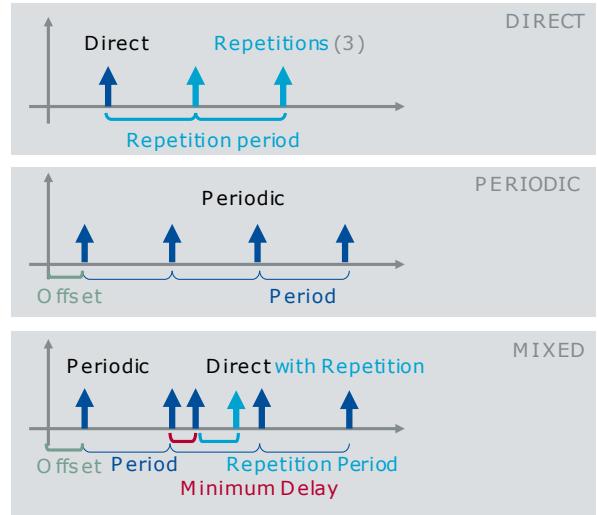
((min <= new) && (new <= max))
min = 10, max = 103

newValue = 10 → true
newValue = 254 → false

Transmission Modes - I-PDU Tx Modes

Available I-PDU Transmission Mode values

- ▶ **NONE**
- ▶ **DIRECT**
 - ▶ Repetition
 - ▶ Repetition Period
- ▶ **PERIODIC**
 - ▶ Period
 - ▶ Offset
- ▶ **MIXED**
 - ▶ Direct and Periodic
 - ▶ Period
 - ▶ Offset
 - ▶ Repetition
 - ▶ Repetition Period
 - ▶ Minimum Delay



Transmission Modes - I-PDU Tx Modes

i

AUTOSAR PARAMETER DEFINITION: ComTxModeTrue, ComTxModeFalse

The **Transmission Mode** determines the Tx properties of the I-PDU:

- ▶ **DIRECT**: the message is transmitted in the next `Com_MainFunctionTx()` call after writing to a signal.
- ▶ **PERIODIC**: the I-PDU is transmitted at the specified cycle time (`ComTxModeTimePeriodFactor`).
- ▶ **MIXED**: Combines the transmission modes PERIODIC and DIRECT.
- ▶ **NONE**: Com does not transmit these I-PDUs. Only lower layers can request the Com to transmit an I-PDU with transmission mode NONE by calling the function `Com_TriggerTransmit()`

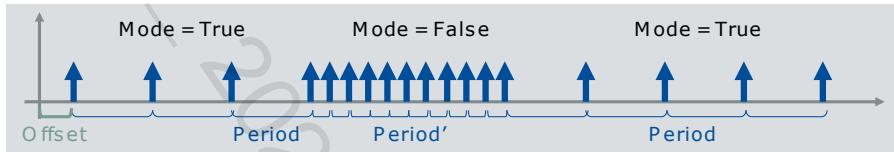
Generally, the Com triggers the transmission of any I-PDU by calling `PduR_ComTransmit()`.

All transmission modes, except NONE, maintain the Minimum Delay Time (`ComTxIPduMinimumDelayTimeFactor`) and delay a transmission to meet this time if required.

Transmission Modes - I-PDU Tx Modes

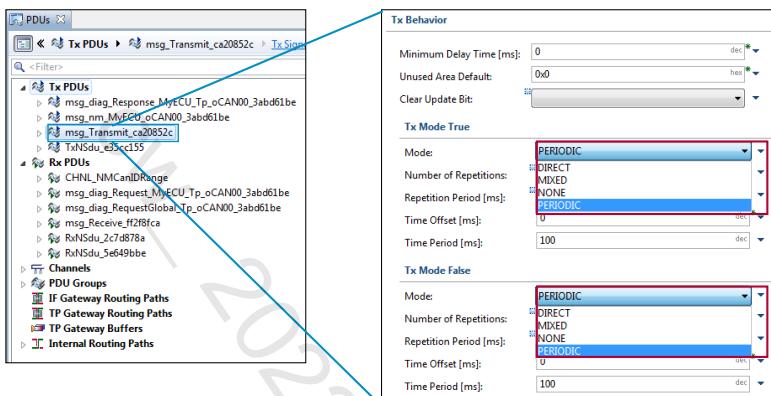
Switching from Transmission Mode "True" to Mode "False"

- ▶ All signal filters of an I-PDU are evaluated
 - ▶ Mode "True" is used if any filter evaluates to True
 - ▶ Mode "False" is used if all filters evaluate to False



Transmission Modes in Configurator

The COM Transmission Modes are
I-PDU Specific settings



Transfer Property in Configurator

The COM Transfer Property is Signal-Specific

► **PENDING**

writing data to this signal will not trigger a direct transmission

► **TRIGGERED** (see notes)

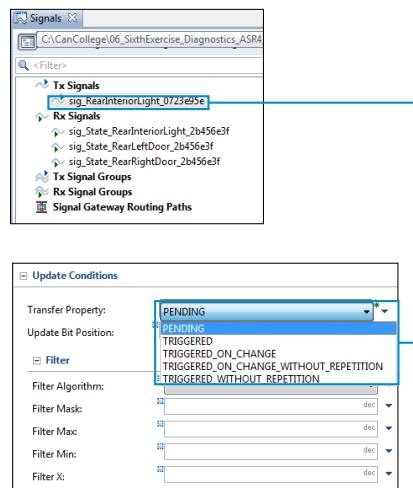
writing data to this signal is the trigger for a transmission

► For Direct or Mixed I-PDU Transmission Modes

► On Write or On Change (with Repetition)

► On Write or On Change without Repetition

For a Transmission Mode DIRECT or MIXED the Transfer Property must be set to TRIGGERED.



Transfer Property in Configurator



The Transfer Property is only used in connection with the Direct or the Mixed Transmission Mode. Exception
Set to **TRIGGERED** means : writing the signal is a trigger for direct transmission of the corresponding I-PDU
Set to **PENDING** means : even writing the signal will not trigger direct transmission of the corresponding I-PDU.

Example: all signals within a PDU are set to **Pending**. Transmission Mode is set to **Direct**. The writing any of the signals will not provoke a transmission of the PDU.

TRIGGERED - cause a transmission with repetitions according to the Tx Mode when signal is written

TRIGGERED_WITHOUT_REPEATITION - ignore repetitions of Tx mode and transmit only once

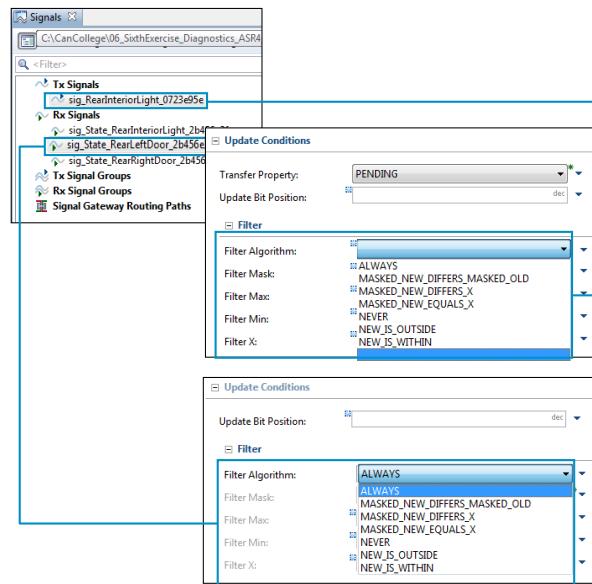
TRIGGERED_ON_CHANGE - trigger transmission including repetitions if signal differs from the locally stored value (last sent or init).

TRIGGERED_ON_CHANGE_WITHOUT_REPEATITION - trigger transmission without repetitions if signal differs to the locally stored (last sent or init) value.

The AUTOSAR Com module shall not support the transfer properties TRIGGERED_ON_CHANGE and TRIGGERED_ON_CHANGE_WITHOUT_REPEATITION for signals and group signals with ComBitSize configured to 0.

Signal Filters in Configurator

- ▶ The COM Filter settings are **Signal-Specific**
- ▶ Signal filters can be defined for Rx or Tx signals
- ▶ Signal filters have an impact on
 - ▶ The transmission mode selector (Tx)
 - ▶ On the indication of the respective signals (Rx)



Signal Filters in Configurator



For the transmission path, there are various filter settings. Filters are optional and in some situations the field can be left empty. If this is the case, the respective signal content is not considered for the Transmission Mode selection.

For the reception path, there is also the possibility to filter the reception seen by the application. The easiest (and default) behavior can be characterized as "ALWAYS" where there is no checking in the Com for signal values compared to the filters. However this can increase the processing time in the application. Therefore you can also select to be notified only on change by "MASKED_NEW_DIFFERS_MASKED_OLD" to decrease the triggering rate.

Since AUTOSAR 4.1.1 , there is one more filter called ONE_EVERY_N. This filter means Com counts transmission requests and on the given value, the transmission is carried out.

If the ComFilterAlgorithm ONE_EVERY_N is used, the value of the signal itself has no influence on the Transmission Mode selection. What counts is the number of send requests by the RTE.

Agenda

- Communication
- Transmission
- ▶ **Signal Groups**
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- Invalidation
- Rx Filter
- Additional Return Values
- Coming up next

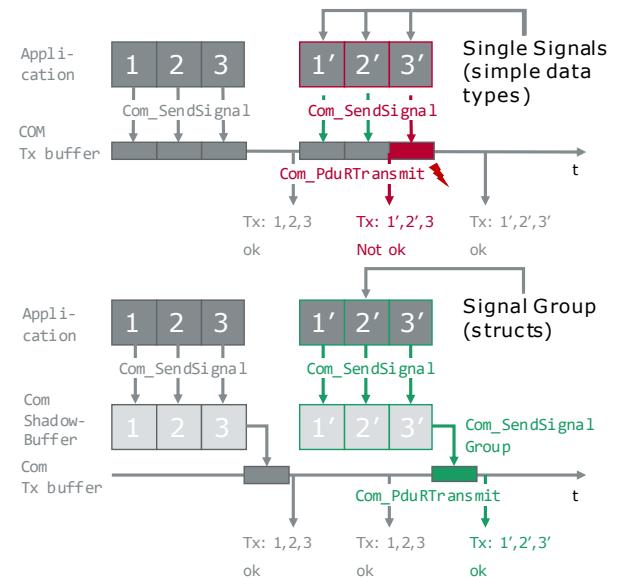
Data consistency using signal groups

Signal Groups

- ▶ Logical group of signals that belong together
- ▶ Signal groups are defined in the communication matrix
- ▶ Complex data types are mapped to signal groups by the RTE
- ▶ Signal groups comprise group signals

Signal Groups signify the following

- ▶ All group signals within a signal group are consistent
- ▶ Access is done by the RTE via a dedicated signal group and group signal API
- ▶ Signal group shadow buffer provided by Com
 - ▶ One global shadow buffer for each signal group



Data consistency using signal groups

i

Example: data that may be grouped in a signal group are the signals for velocity and vehicle direction

In AUTOSAR 4.x the API `Com_SendSignal()` writes a group signal into the shadow buffer, not directly to the I-PDU. After all group signals of a signal group have been updated, the consistent content of the shadow buffer is "transmitted" using the API `Com_SendSignalGroup()`. A similar API is available for signal group reception.

Note: In AUTOSAR versions before 4.x, the API `Com_UpdateShadowSignal()` is used instead of `Com_SendSignal()` to update Group signals. This API is now deprecated.

Agenda

- Communication
- Transmission
- Signal Groups
- ▶ **Update Bit**
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- Invalidation
- Rx Filter
- Additional Return Values
- Coming up next

Concept

Update Bit

- ▶ Is defined within the communication matrix
- ▶ Allows the receiver to determine if the transmitter has updated the value

What is the Update Bit?

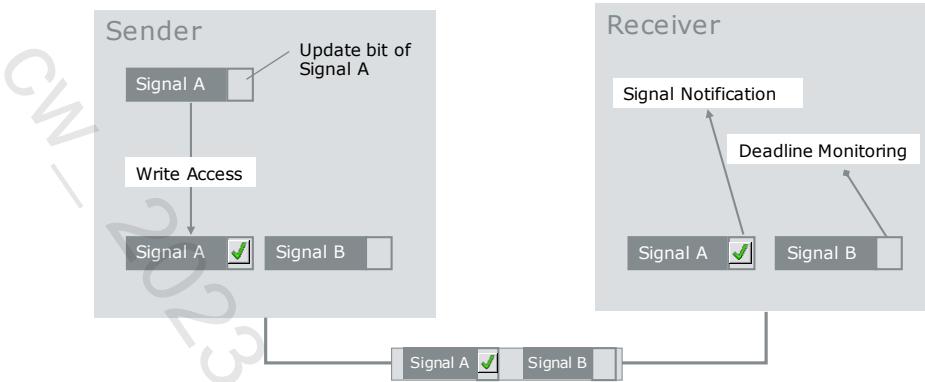
- ▶ Dedicated bit within the I-PDU of the related signal
- ▶ Not directly accessible for the application or RTE
- ▶ The update bit is **set by COM** when signal is written
- ▶ Update bits are **cleared by COM** according to
 - ▶ *ComTxIPduClearUpdateBit = {Confirmation|Transmit|TriggerTransmit}*
- ▶ Usage of update bit causes signal-based handling of
 - ▶ Notification functions
 - ▶ Deadline monitoring



The update bit is set independently of the signal filter result

Concept

- ▶ Update bits allow a signal-based timeout observation and a signal-based receive notification



Agenda

Communication

Transmission

Signal Groups

Update Bit

► Reception

Data Mapping

Notification Mechanisms

Deadline Monitoring

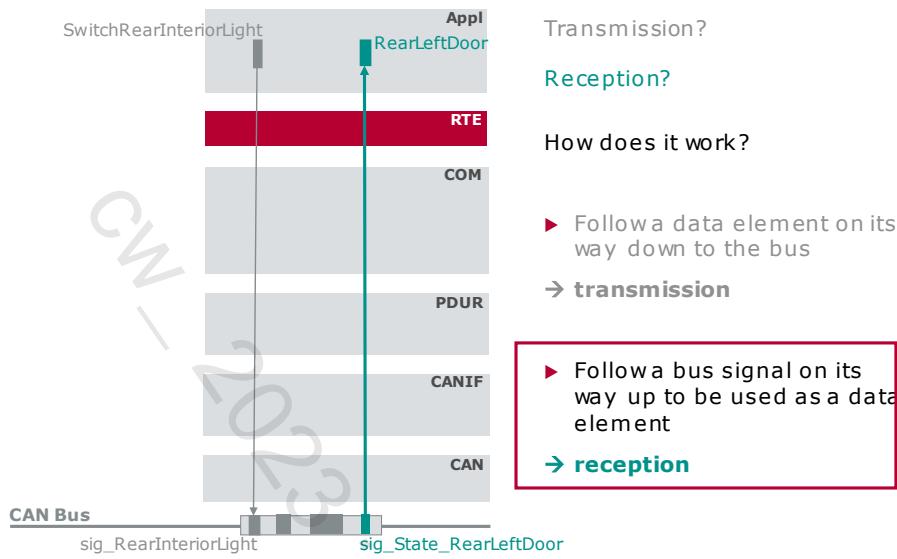
Invalidation

Rx Filter

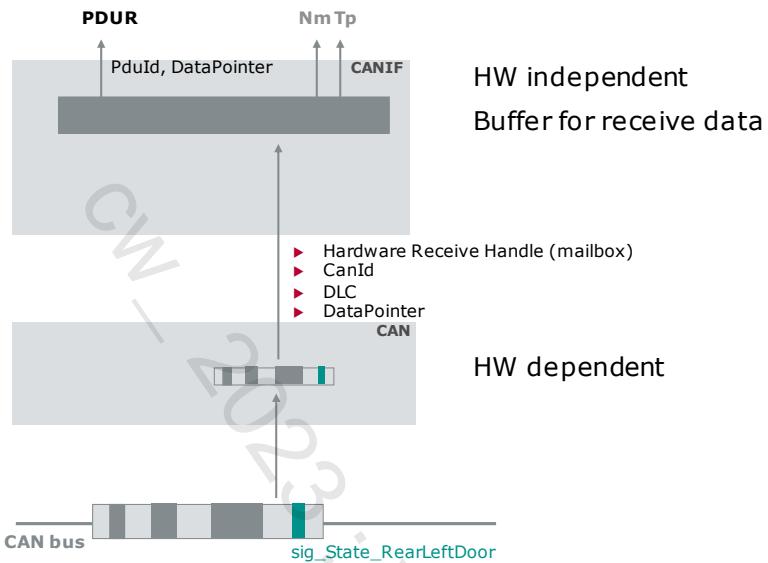
Additional Return Values

Coming up next

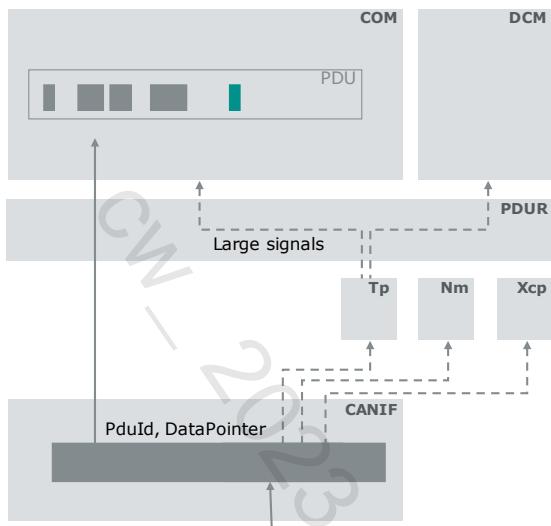
Signal Reception



Signal Reception: Bus, CAN, CANIF



Signal Reception: CANIF, PDUR, COM



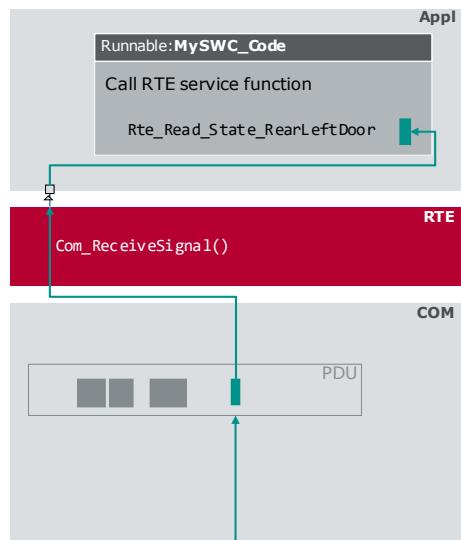
Signal Reception: COM, RTE, Appl

Direct / Explicit Read

- When reading the data element several times, the data may change as the latest signal value received by Com is returned.

Buffered / Implicit Read

- Before the start of the runnable, the signal value is copied to a buffer. The application accesses the data only from the buffer. Therefore the data element remains the same even when the underlying signal value changes.



Agenda

Communication

Transmission

Signal Groups

Update Bit

Reception

► **Data Mapping**

Notification Mechanisms

Deadline Monitoring

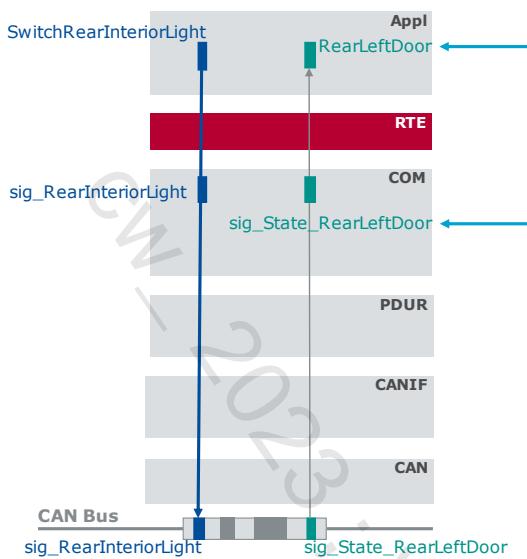
Invalidation

Rx Filter

Additional Return Values

Coming up next

Overview



In DaVinci Configurator

Data Elements have to be assigned to Network (Com) Signals.

This assignment is done when performing the **Data Mapping**.

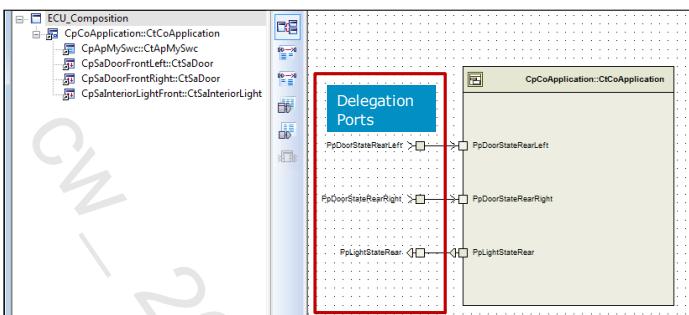
- ▶ Data Mapping is done for receive and transmit information
- ▶ Data Mapping means assigning **Data Elements** to **Network Signals**

Two ways to do Data Mapping

1. Based on the root element in the hierarchical SW Design
 - ▶ At the level of the **ECU SW Composition**
 - ▶ **Delegation Ports** are required
 - ▶ Can be done in both **Developer** and **Configurator**
2. Based on the leaves of the SW Design without respect to the hierarchy
 - ▶ At the level of **Atomic SWCs**
 - ▶ **No Delegation Ports** required
 - ▶ This option is only available in **Configurator**

AUTOSAR 3.x style

- **DaVinci Developer** needs Delegation Ports at the ECU Composition level to do Data Mapping.



The **Delegation Ports** promote the Port Prototypes to a higher level in the SWC hierarchy. In the above case this is the top level SWC "ECU_Composition" which is a composition type SWC .

Workspace	Data Element	Port	Direction	Network	Message	Network Signal
ECU Projects						
MyECU						
Software Design						
Data Mapping	10101	PpDoorStateRearLeft	Rx	CAN00	msg_Receive_pCAN00	sig_State_RearLeftDoor
		PpDoorStateRearRight	Rx	CAN00	msg_Receive_pCAN00	sig_State_RearRightDoor
		PpLightStateRear	Tx	CAN00	msg_Transmit_pCAN00	sig_RearInteriorLight

Data Mapping in **DaVinci Developer**: Port Prototypes of the ECU SW Composition

AUTOSAR 3.x or 4.x style

- In **Configurator**, the ECU SW Composition appears similarly

Delegation Ports with Data Elements

Port Prototype	Data Element Prototype	Direction	Channel	PDU	Signal
PpLightStateRear	DeLightState	Tx	CHNL	msg_Transmit_oCAN00	sig_RearInteriorLight
PpDoorStateRear_left	DeDoorState	Rx	CHNL	msg_Receive_oCAN00	sig_State_RearLeftDoor
PpDoorStateRear_right	DeDoorState	Rx	CHNL	msg_Receive_oCAN00	sig_State_RearRightDoor

Data Mapping in **Configurator**: Port Prototypes of the ECU SW Composition

- In addition, Configurator also allows Data Mapping at atomic SWCs further down in the hierarchy. This is usually empty.

Delegation Ports with Data Elements

Port Prototype	Data Element Prototype	Direction	Channel	PDU	Signal
PpDisplayState	DeOdometerValue	Tx	<unmapped>	<unmapped>	<unmapped>
PpDisplayState	DeOdometerWriteRequestPending	Tx	<unmapped>	<unmapped>	<unmapped>
PpDoorStateFrontLeft	DeDoorState	Rx	<unmapped>	<unmapped>	<unmapped>
PpDoorStateFrontRight	DeDoorState	Rx	<unmapped>	<unmapped>	<unmapped>
PpDoorStateRearLeft	DeDoorState	Rx	<unmapped>	<unmapped>	<unmapped>
PpDoorStateRearRight	DeDoorState	Rx	<unmapped>	<unmapped>	<unmapped>

Data Mapping in **Configurator**: Ordinary Port Prototypes of an arbitrary atomic SWC can directly be mapped without delegation ports.
In this picture the respective data elements are not mapped because this is done on ECU composition level (see figure on top)

Agenda

Communication

Transmission

Signal Groups

Update Bit

Reception

Data Mapping

► **Notification Mechanisms**

Deadline Monitoring

Invalidation

Rx Filter

Additional Return Values

Coming up next

I-PDU Signal Processing

This option determines the call context and the point in time when the signal's

- ▶ Rx notification (Rx indication) or
- ▶ Tx notification (Tx confirmation) is called.

▶ IMMEDIATE

The notification function is called within `Com_RxIndication()` or `Com_TxConfirmation()`. Depending on the lower layer interface, this might be in interrupt context.

▶ DEFERRED

The notification function is called on task level during the next call cycle of `Com_MainFunctionRx()` or `Com_MainFunctionTx()`.



Please note that in **DEFERRED** I-PDU Signal Processing, the signal unpacking of the data happens always during the `Com_MainFunctionRx`, regardless of the setting of `IpduSignalProcessing` parameter.

That means the user can only access data at earliest after the next `Com_MainfunctionRx` call.

For **IMMEDIATE** I-PDU Signal Processing the unpacking of the data already happens during the `CanIf_RxIndication`, i.e., maybe in interrupt context.

I-PDU Signal Processing

Notification is I-PDU-specific

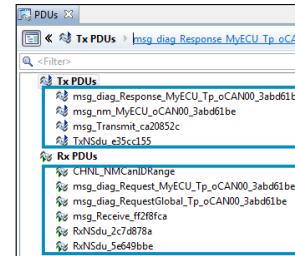
- ▶ Rx and Tx notification functions are defined specifically for a signal
- ▶ The kind of notification mechanism (IMMEDIATE or DEFERRED) is specific for an I-PDU

Pdu	Port	Channel	Pdu Groups	Callout	Handle Id	Signal Processing
CHNL_NMCaniDRange	CanIf	CHNL_f276589c	-			
msg_diag_Request_MyECU_Tp_oCAN00_3abd61be	CanIf	CHNL_f276589c	-			
msg_diag_RequestGlobal_Tp_oCAN00_3abd61be	CanIf	CHNL_f276589c	-			
msg_Receive_ff28fcfa	CanTp	<NONE>	-		0*	DEFERRED
RxNSdu_2c7d878a	CanTp	<NONE>	-			DEFERRED
RxNSdu_5e649bbe						IMMEDIATE

Pdu	Port	Channel	Pdu Groups	Callout	Signal Processing
msg_diag_Response_MyECU_Tp_oCAN00_3abd61be	CanIf	CHNL_f276589c	-		DEFERRED
msg_nm_MyECU_oCAN00_3abd61be	CanIf	CHNL_f276589c	-		DEFERRED
msg_Transmit_c20852c	CanIf	CHNL_f276589c	MyECU_oCAN00_Tx_fcef2243		IMMEDIATE
TxNSdu_e35cc155	CanTp	<NONE>	-		



In case update bits have been configured for an Rx signal, the Rx notification is called only if the associated update bit is set



Agenda

- Communication
- Transmission
- Signal Groups
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- ▶ **Deadline Monitoring**
- Invalidation
- Rx Filter
- Additional Return Values
- Coming up next

Mechanism and adjustment of time values

COM can detect timeouts on

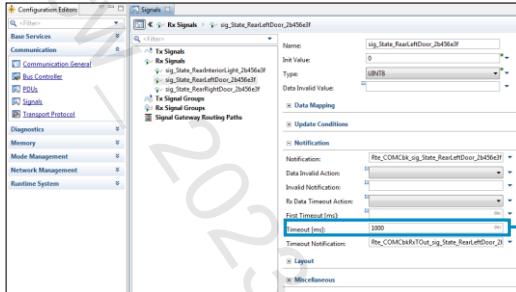
- ▶ I-PDU level
 - ▶ An Rx I-PDU is not received within a predefined timeout time
- ▶ Signal level
 - ▶ The update bit of a signal has not been set for a predefined timeout time

Timeout time

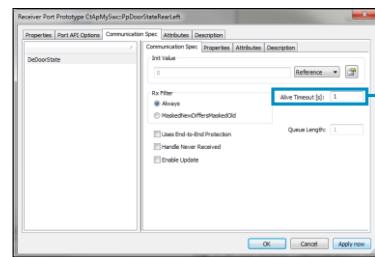
- ▶ Can be configured in DaVinci Developer (SWC) and Configurator (Com)
- ▶ Synchronization via Configurator
- ▶ The timeout time of an I-PDU is derived from the smallest timeout time of monitored signals within the I-PDU

Timeout settings in Configurator and Developer

- ▶ Only Inter-ECU timeout monitoring is supported
- ▶ since the Com module provides the Deadline Monitoring functionality



Configurator: Related Com signal after synchronization with the Developer project



Developer: Alive Timeout in the ComSpec of a data element prototype

Timeout on COM and RTE level

COM Timeout handling:

- ▶ A timeout notification is called by Com
 - Rte_COMCbkTOut_<ComSignalName>
- ▶ Optionally, affected signals are set to their initial values (if ComRxDataTimeoutAction is configured to TRUE)

RTE Timeout handling:

- ▶ A timeout can trigger a runnable in the RTE
- ▶ The return value of the function `Rte_Read <Pp> <De>()` indicates the error `RTE_E_MAX_AGE_EXCEEDED`
- ▶ In case of implicit S/R communication (`Rte_IRead` API) the status can be determined by the `Rte_IStatus` API

Timeout on COM and RTE level



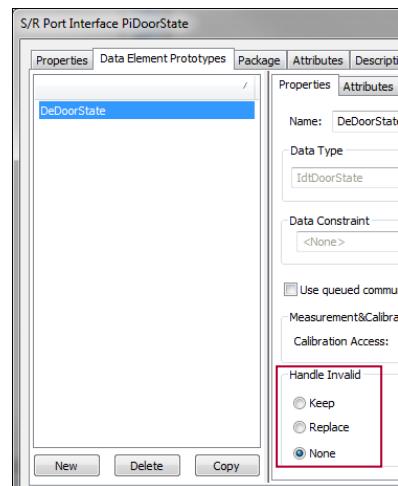
- For intra-ECU Sender / Receiver communication, the RTE does not perform timeout monitoring
- RTE can do timeout supervision for asynchronous, intra-ECU Client / Server API calls
- If Com detects a timeout, it will **not** set any Dem event automatically. If a Com timeout shall result in a Dem entry, the application has to set the Dem event by itself.
- Com calls the timeout notification cyclically as long as the timeout condition exists. The cycle time is equal to the configured timeout time.
- In case of a timeout the signal status is set to `RTE_E_MAX_AGE_EXCEEDED`. The signal status can be obtained in the return value of the `Rte_Read_<Pp>_<De>(...)` function call for the respective signal.
 - If you set the AliveTimeout value for a data element inside a Port Prototype to a non-zero value, this value will be used in the Com configuration for the signal. The `Rte_COMCbkTOut` callback will be turned on and the timeout factor will be set to the value from the SWC. This will happen as soon as the ECUEX becomes synchronized in Configurator and the RTE Validation takes place.
 - If you have a Com Signal data element relationship of 1:N, then the minimum timeout value of all related data elements will be used as the timeout value.

Agenda

- Communication
- Transmission
- Signal Groups
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- ▶ **Invalidation**
- Rx Filter
- Additional Return Values
- Coming up next

RTE level

- ▶ Only for explicit access, non-queued communication, or implicit API
- ▶ Inter-ECU and Intra-ECU supported
- ▶ Tx Signal (Handle Invalid != none)
 - ▶ Send invalid data via
 - > Rte_Invalidate (explicit API)
 - > Rte_IInvalidate (implicit API)
- ▶ Rx Signal
 - ▶ Handle Invalid receive data
 - > KEEP (RTE_E_INVALID available)
 - > REPLACE (no RTE_E_INVALID available)
 - > NONE (no RTE_E_INVALID available)



DEV (ASR4.x): Handle Invalid property is located in the DataElementPrototype inside the **Application Port Interface** definition (ASR3.x has it in the P/R-Port CommSpec)

Timeout on COM and RTE level



Implicit is buffered and uses Rte_IRead
Explicit is direct and uses Rte_Read

KEEP (Rte_COMCbkInv_<ComSignalName>)
Keeps the last received valid value in COM

Std_ReturnType RTE_E_INVALID

Optionally Data Receive Error Event Trigger via the Invalidation Callback
REPLACE (no Rte_COMCbkInv_<ComSignalName>)

Replaces value by "init" value in COM if Com_DataInvalidAction is REPLACE

No notification to the application that a replacement value is used: Data Receive Event, no RTE_E_INVALID available.

The Init value is not necessarily from the valid data range!

Data range invalidation (undefined reception range) does not exist in ASR3.x and is a feature of ASR 4.0.

NONE (no Rte_COMCbkInv_<ComSignalName>)

The invalid COM value propagates to application; no notification of RTE_E_INVALID

The application has to examine the signal value and handle the invalidation on its own

Invalidation	
Invalid Notification	Data Invalid Action
*	NONE
Re_COMCbkInv_Cm_sig_State_RearLeftDoor__msg_Receive__CAN00	NOTIFY
*	REPLACE

Corresponding GENy settings for the Inter ECU case. KEEP corresponds to NOTIFY, REPLACE is the same like on RTE level.

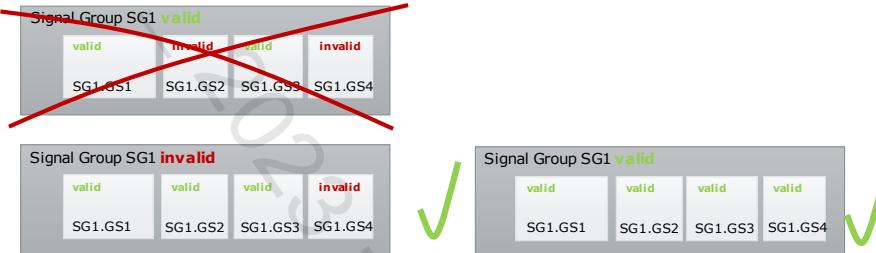
Remark for the Invalidation intra-ECU (ECU internally, without participation of the Com module):

If the sender data element is configured for validation support over "can Invalidate", a built-in check denies the setting "NONE" for the option "Handle Invalid" at the receiver data element.

3. VH Autosar CP Training_EN

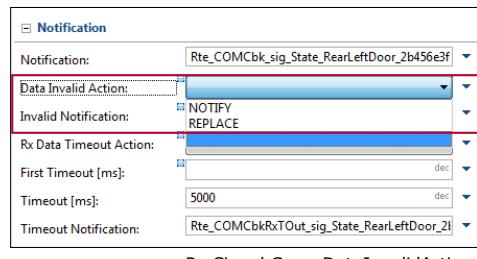
COM level

- ▶ Precondition for invalidation
 - ▶ Only signals which have an enabled `ComSignalDataInvalidValue` can be invalidated at all
- ▶ Invalidation of group signals
 1. A **valid** signal group can not contain any **invalid** group signals
 2. An **invalid** signal group contains at least one **invalid** group signal
 - ▶ If one of the group signals becomes invalid, this affects the entire signal group



COM level

- ▶ Com_DataInvalidAction
 - ▶ NONE
 - ▶ Nothing happens received invalid value is in the Com buffer
 - ▶ NOTIFY
 - ▶ The invalid notification Rte_COMCbkInv_<ComSignalName> will be called
 - ▶ Last valid received value remains in the Com buffer
 - ▶ REPLACE
 - ▶ Replace the received invalid value in the Com buffer by the Init value



Rx Signal Com_DataInvalidAction

Agenda

- Communication
- Transmission
- Signal Groups
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- Invalidation
- **Rx Filter**
- Additional Return Values
- Coming up next

Reception Filtering

- ▶ Inter-ECU
- ▶ Checkbox in Developer
 - ▶ R-Port data element Communication Spec
- ▶ Corresponding settings in Com for the Rx Signals
- ▶ Always
 - ▶ every reception causes a notification
- ▶ MaskedNewDiffersMaskedOld
 - ▶ RTE can produce a Data Received Event which only happens "on change"

Properties	Port API Options	Communication Spec	Attributes	Description
		DeDoorState		
		Init Value	0	
		Rx Filter <input checked="" type="radio"/> Always <input type="radio"/> MaskedNewDiffersMaskedOld		
		<input type="checkbox"/> Uses End-to-End Protection <input type="checkbox"/> Handle Never Received <input type="checkbox"/> Enable Update		

Developer R-Port Prototype: Rx Filter in the Communication Spec of a Data element

Signals	Rx Signals	sig_State_RearLeftDoor_2b456e3f	Update bit positions
	<Filter>		
	Filter	Filter Algorithm: ALWAYS	Filter Mask: MASKED_NEW_DIFFERS_MASKED_OLD MASKED_NEW_DIFFERS_X MASKED_NEW_EQUALS_X NEVER NEW_IS_OUTSIDE NEW_IS_WITHIN

CFG Signal Filter settings

Agenda

- Communication
- Transmission
- Signal Groups
- Update Bit
- Reception
- Data Mapping
- Notification Mechanisms
- Deadline Monitoring
- Invalidation
- Rx Filter

► Additional Return Values

Coming up next

Further Communication Specification settings

- ▶ "Handle Never Received" checkbox in Developer
- ▶ R-Port data element Communication Spec
- ▶ Rte_Read + return Value
Rte_IRead + Rte_IStatus
 - ▶ RTE_E_NEVER RECEIVED
 - ▶ Since Rte_Start, no data received over COM
- ▶ Rte_Read provides the Init value in the case of RTE_E_NEVER RECEIVED
- ▶ Further information
 - ▶ In case COM is switched off
 - > Rte_Write returns RTE_E_COM_STOPPED
 - ▶ "Enable Update" checkbox
 - > Boolean Rte_IsUpdated <p><d>([IN Rte Instance]) tells if new data is received.

Properties	Communication Spec	Attributes	Description
DeDoorState			
		Init Value	<None>
		Rx Filter	<input checked="" type="radio"/> Always <input type="radio"/> MaskedNewDiffersMaskedOld
		<input type="checkbox"/> Uses End-to-End Protection <input checked="" type="checkbox"/> Handle Never Received <input type="checkbox"/> Enable Update	

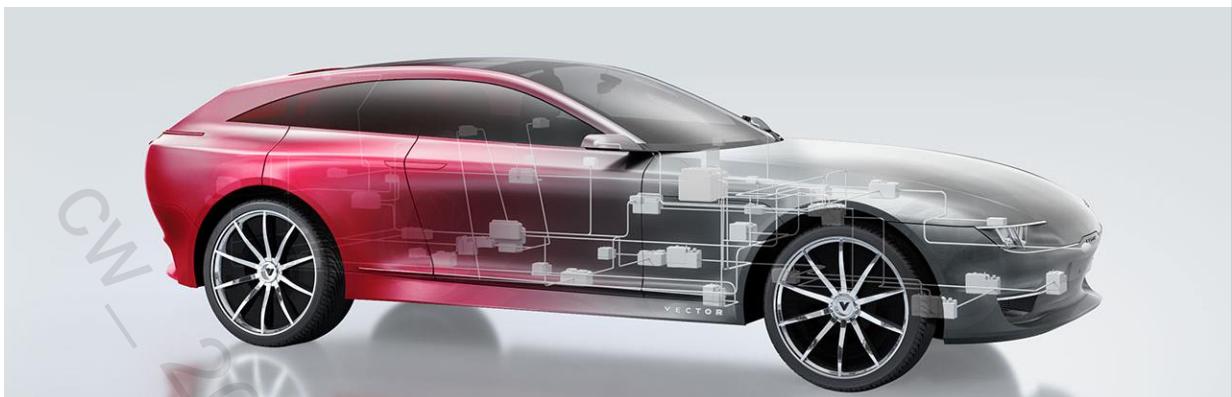
Developer R-Port Prototype: Handle Never Received in the Communication Spec of a Data element prototype



This feature is if you need to detect that a signal has been received for the first time by Com.

Communication

Exercise: Communication



AUTOSAR in Practice

Exercise 3: Communication

V28 | 2023-03-28

Agenda

► **Exercise 3 - Communication**

Coming up next

CW_2023_VH_Autosar CP Training_EN

Communication

3

Now the project is extended by adding CAN communication.

The information about the rear doors and the rear interior light is exchanged via CAN messages.

The RearECU informs MyECU about the state of the RearLeftDoor and the RearRightDoor. MyECU decides about the resulting state of the RearInteriorLight.

The RearInteriorLight is controlled via another CAN message.

3**Communication**

- **1** Create Ports for Communication
- **2** Data Mapping
- **3** BSW configuration for Communication
- **4** Generate program Runnables and Tests

Part 1/4: Create Ports for Communication

- ▶ "Open with > DaVinci Project Assistant" using `MyECU.dpa` file in folder `\E3_Communication\`
- ▶ Have a look at the COM Stack and try to figure out if there are any signals which might be required for your ECU design. Try to track them down to the CAN message they are in.
 - ▶ Check the Signal interface
 - ▶ How could the application be notified about the reception of data?
 - ▶ As we know from AUTOSAR, there is always a cyclic call into the `Com_MainFunctionRx` and `Tx` where reception and transmission takes place.
 - ▶ On the other hand, there are fields in Configurator for notification or indication functions.
 - ▶ Close **Configurator** again
- ▶ Check if the **DaVinci Developer** offers any CAN signals you can connect to (next slide)

Part 1/4: Create Ports for Communication

i

Check available CAN signals:

Open the Data Mapping window:



The signals have been defined in the (Ecu Extract of) System Configuration Description.

The available network signals can be seen in the "Data Mapping" view in DaVinci Developer. It has two different icons to change how data mapping is displayed.

Network Signal	Transformed	Message	Network	Direction	Port	Data Element / Operation
~sg_RearInvertLight	No	msg_Transmit_cA000	CAN00	Tx-Signal	-	-
10101	Yes	msg_Receive_cA000	CAN00	Rx-Signal	-	-
~sg_State_RearLeftDoor	No	msg_Receive_cA000	CAN00	Rx-Signal	-	-
~sg_State_RearRightDoor	No	msg_Receive_cA000	CAN00	Rx-Signal	-	-

The sine wave icon shows the network signals as primary column. On the right side you see data elements mapped on the signals:

Data Element / Operation	Port	Direction	Network	Message	Transf...	Network Signal
Bill_DoorState	PxDoorStateLeft	Rx	-	-	-	-
10101	PxDoorStateRight	Rx	-	-	-	-
Bill_DoorState	PoLightStateRear	Tx	-	-	-	-

The 10101 icon shows all data elements of the Delegation Ports of the ECU Composition, these elements are not yet existing in your project

Part 1/4: Create Ports for Communication

- ▶ If you haven't, open Developer using "**Open with > DaVinci Developer Classic**" from the **MyECU.dpa** file in folder **\E3_Communication**
- ▶ Use existing Port Interfaces for CAN communication (as shown in illustration on page 7)
 - ▶ Port Interfaces **PiDoorState** and **PiLightState** have been created in the first exercise
 - > There they were used for Intra-ECU communication (internal)
 - > Now we use the same port interfaces for Inter-ECU communication (over the CAN bus)
 - ▶ Create Receiver Port Prototypes on CtApMySwc for receiving the rear door state
 - > PiDoorState -> **PpDoorStateRearLeft**, initial value "0"
 - > PiDoorState -> **PpDoorStateRearRight**, initial value "0"
 - ▶ Create Sender Port Prototypes on CtApMySwc for sending the rear light state
 - > PiLightState -> **PpLightStateRear**, initial value "0"

Part 1/4: Create Ports for Communication



Ports and CAN Signals:

There is a CAN receive message that contains three signals: state of rear left door, state of rear right door, and the state of the rear interior light (for feedback; not used in the exercise). The two door signals should be mapped to corresponding data elements within a port.

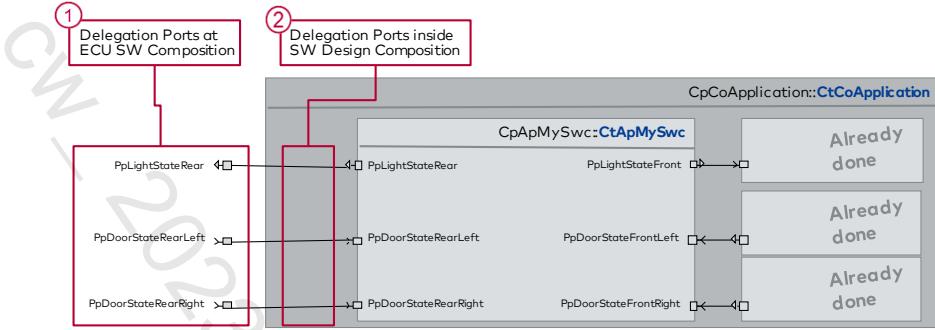
On the sender side there is a CAN transmit message containing the requested state of the rear interior light. That's the reason for the sender port containing the data element to control the state of the rear interior light. This data element should be mapped to the CAN transmit signal.

→ Data Mapping is coming later

N-2023-VH-Autosar-CP-Training_EN

Part 1/4: Create Ports for Communication

- ▶ Create 'Delegation Ports'
 - ▶ at the level inside **CpCoApplication** (1) and
 - ▶ a second time at the level **ECU_Composition** (2) → this is the part for data mapping!

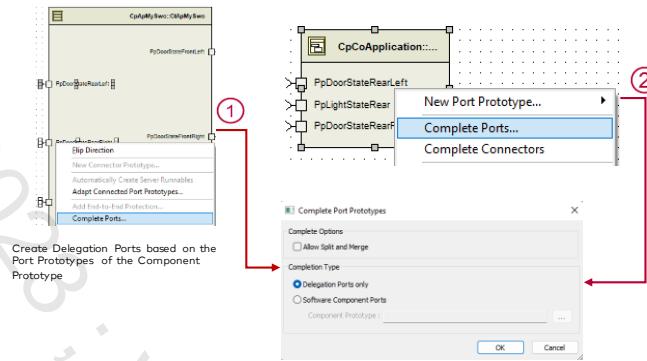


Part 1/4: Create Ports for Communication

i

How to create the Delegation Ports: Click on the Port Prototype(s) - e.g., with <Shift> pressed for multiple selection - of the Component **CpApMySwc** in the SW Design. Then right-click on the Component and select **"Complete Ports"**. In the following dialogue select **"Delegation ports only"**. This will create the Delegation Ports for the outer interface of the ECU Composition.

Please note that depending on the hierarchy level, you need to perform the steps described in the figure below a second time (you have an additional Composition within which your Atomic SWCs reside!).



Part 2/4: Data Mapping

- ▶ Perform Data Mapping – how? See notes section on next slide
- ▶ Go to Runnable **RCtApMySwcCode** in **CpApMySwc**
 - ▶ Add **Access Points** to **read** in the rear door states and **write** the rear light state
 - ▶ Read in PpDoorStateRearLeft.DeDoorState in a **buffered** (implicit) way
 - ▶ Read in PpDoorStateRearRight.DeDoorState in a **direct** (explicit by argument) way
 - ▶ Write PpLightStateRear.DeLightState in a **direct** (explicit) way
 - ▶ Add an **On Data Reception ... Trigger** for the rear door data elements
- ▶ Check Workspace 
Check Workspace
- ▶ Save Workspace 
Save Workspace
- ▶ Close DaVinci Developer

Part 2/4: Data Mapping



Perform Data Mapping

Perform the data mapping: e.g., right-click on each data element, then choose the respective network signal until all data elements have a data mapping.

	Data Element / Operation	Port	Direction	Network	Message	Transf...	Network Signal
10101	DeDoorState	DeDoorStateDataPort	-	-	-	-	-
		Create Mapping...	-	-	-	-	-
		Create complex Mapping...	-	-	-	-	-

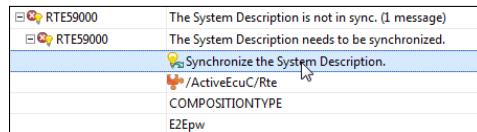
In the signal view, not all signals need to be mapped to a data element, e.g., we do not need an Rx signal. Leaving that unmapped is allowed. For Tx signals, we may not need them to fulfill the ECU functionality, but other ECUs might rely on them in order to implement their part of the distributed system. Tx signals need to be mapped, in order to provide information for other parts of the vehicle.

	Network Signal	Transf...	Message	Network	Direction	Port	Data Element / Operation
10101	~sig_RearInteriorLight	No	msg_Transmit_oCAN00	CAN00	Tx-Signal	PpLightStateRear	DelightState
	~sig_State_RearInteriorLight	No	msg_Receive_oCAN00	CAN00	Rx-Signal	-	-
	~sig_State_RearLeftDoor	No	msg_Receive_oCAN00	CAN00	Rx-Signal	PpDoorStateRearLeft	DeDoorState
	~sig_State_RearRightDoor	No	msg_Receive_oCAN00	CAN00	Rx-Signal	PpDoorStateRearRight	DeDoorState

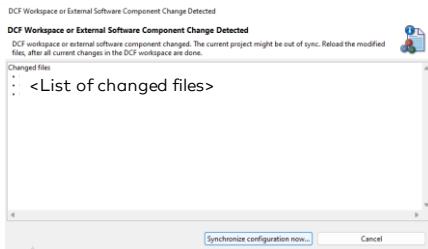
Data Mapping Dialog – signal view: To see, which signals are already mapped to data elements, you can choose a bus signal-based view.

Part 3/4: Watch BSW configuration for Communication

- ▶ “Open with > DaVinci Project Assistant” using MyECU.dpa file in folder \E3_Communication\
- ▶ Execute the auto-solving action for the following error:



- ▶ If Configurator was left open, you can choose “synchronize” from a pop-up dialog.



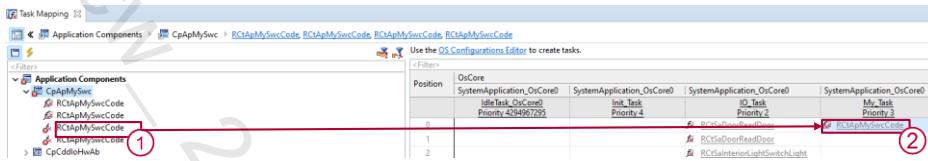
Part 3/4: Watch BSW configuration for Communication

- The new Data Receive Function triggers for the rear doors need to be mapped to a task.

RTE01056
RTE01059
Unmapped runnable entity (' message)
The runnable entity 'RC1ApMySwCode' is not mapped to a valid task.
RmMappedToTaskRef
/ActiveEcu/Rte/CpApMySw_EcuSwComposition/DIF_RC1ApMySwCode_PiDoorStateRearLeft_DeDoorState
/ActiveEcu/Rte/CpApMySw_EcuSwComposition/DIF_RC1ApMySwCode_PiDoorStateRearRight_DeDoorState

- Go to **Runtime System → Task Mapping → Application Components → CpApMySwC**

- Map one of the Triggers to the existing one in My_Task (drag & drop)



- Validate.

Part 3/4: Watch BSW configuration for Communication

- ▶ After the validation two errors occur which can be solved with solving actions.
- ▶ **COM02430:** Adds the header file Rte_Cbk.h for callback functions which shall be included by the COM module.
- ▶ **COM02432:** Set the number of ComSignal notifications to 1, which will be cached during unpacking of all received deferred ComIPdus with closed Rx ISR lock.

CANNM02091	Partial Networking Feature is not available. (1 message)
COM02326	Parameter ComTransferProperty will be ignored for Rx ComSignals/ComGroupSignals. (3 messages)
COM02430	Missing Rte callback header file. (1 message)
COM02430	A Rte callback function has been detected. Please add Rte_Cbk.h to ComUserCbkHeaderFile. Create the parameter ComUserCbkHeaderFile in /ActiveEcuC/Com/ComGeneral and set to value Rte_Cbk.h.
/ActiveEcuC/Com/ComGeneral	ComUserCbkHeaderFile
COM02432	Inconsistent deferred notification cache configuration. (1 message)
COM02432	The deferred notification cache size is set to 0, but deferred signal notifications are present. The deferred notification cache size must be at least 1. Set the parameter /ActiveEcuC/Com/ComConfig[ComRxDeferredNotificationCacheSize](value=0) to value 1. /ActiveEcuC/Com/ComConfig[ComRxDeferredNotificationCacheSize]

- ▶ Generate the code. Click "OK" to close the VTT Dialog
- ▶ Close the Generation dialog.
- ▶ Press F7 in order to update the SWC templates

Part 3/4: Watch BSW configuration for Communication

- ▶ For this exercise no further actions are necessary in Configurator.
- ▶ Become familiar with the default settings and other possible settings in Configurator
- ▶ E.g., look for settings of transmission modes of the I-PDUs ...
- ▶ Look at the **sig_State_Rear*** Rx Signals
- ▶ Notification function has been added by RTE Validator in **Configurator**, because a "Receive data..." trigger has been set.
- ▶ Search also for the generated Rte_COMCbk implementations in Rte.c.
- ▶ Close Configurator

The screenshot shows the Vector Configurator interface with the 'Rx Signals' list open. The path in the top navigation bar is 'Signals > Rx Signals > sig_State_RearLeftDoor_omsg_Receive_oCAN00_84ad4140_Rx'. The left sidebar shows categories like Tx Signals, Rx Signals, Tx Signal Groups, Rx Signal Groups, and Signal Gateway Routing Paths. The main list view has columns for 'Signal' and 'Notification'. The 'sig_State_RearLeftDoor_omsg_Receive_oCAN00_84ad4140_Rx' signal is selected, and its notification details are shown in the 'Notification' column. A red box highlights the 'Notification' column for this specific signal. A red arrow points from the text 'Notification callbacks' to this highlighted area.

As a consequence from configured Data Receive Events at SWC R-Ports in the RTE, the Configurator activates the corresponding COM callbacks to trigger these events. The RTE has to provide the implementation of the corresponding callbacks.

Part 4/4: Generation, program Runnables and test

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder **\E3_Communication\...\Solution**
- ▶ Add your code to the runnables
 - ▶ Read in the state of each rear door in SWC CtApMySwc.c
 - ▶ If any door is open
 - ▶ Turn on rear interior light and front interior light
 - ▶ If all doors are closed
 - ▶ Turn off front interior light and rear interior light
- ▶ Check if you implemented the right behavior
 - ▶ When opening or closing one rear door, both lights should turn on or turn off (all other doors remain closed)
 - ▶ The application algorithm already exists. You only have to extend the 'if' condition and add the rear doors and rear light handling.



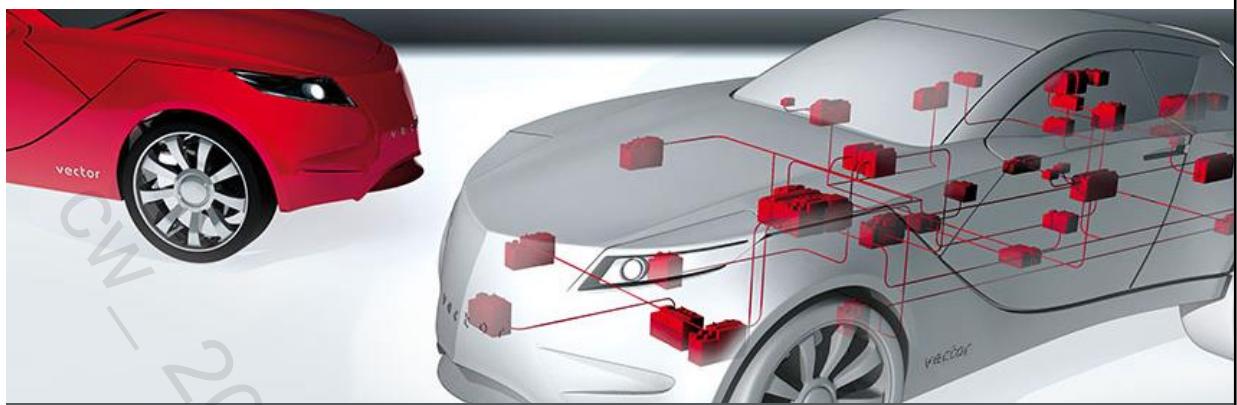
Use the given CompuMethod constants "CM*" as arguments for evaluation of the door state and the setting for the lights

Part 4/4: Generation, program Runnables and test

- ▶ Build ECU code
- ▶ Start CANoe, switch back to Visual Studio, then press F5
- ▶ Switch back to CANoe, then press F9
- ▶ Test your code

Communication

Mode Management



AUTOSAR in Practice

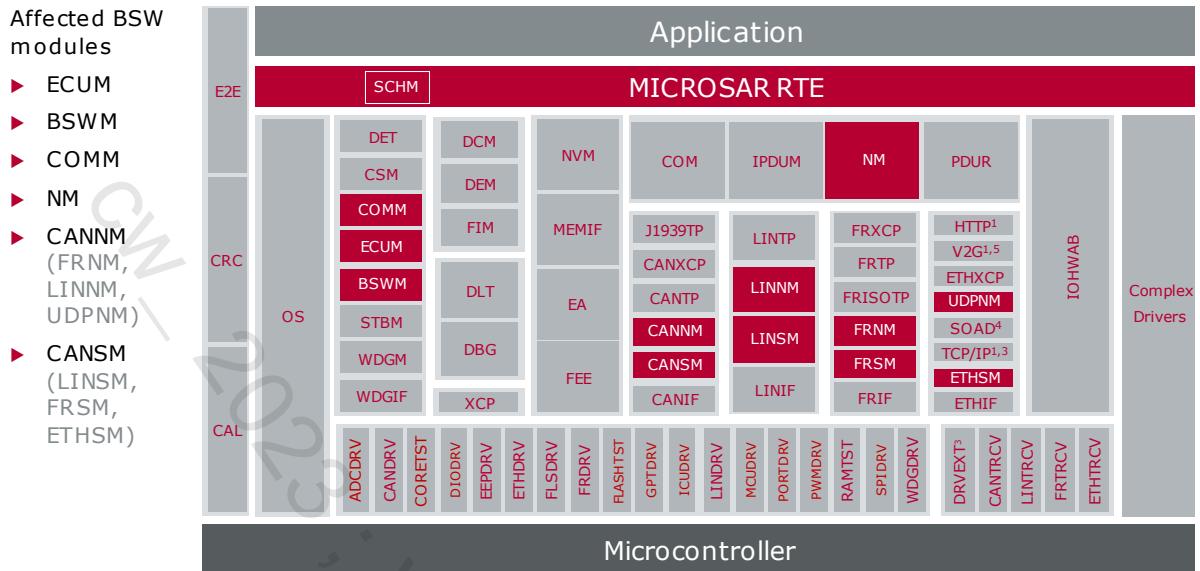
Mode Management

V1.0.0 | 2020-01-20

Agenda

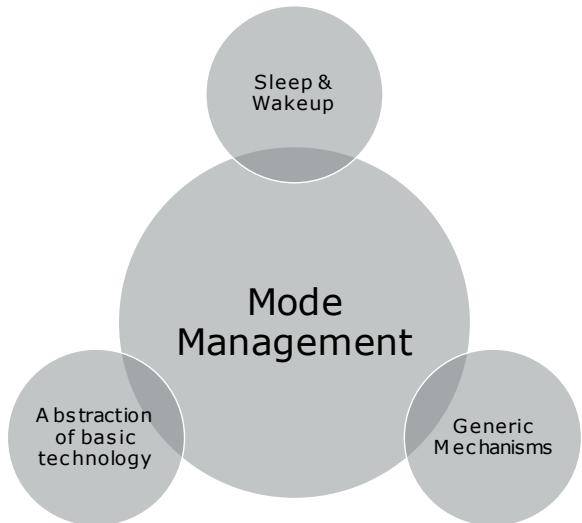
► **Mode Management**

- Wakeup Handling
- BSWM configuration
- ECUM configuration
- Mode Management
- Network Management Algorithm
- Mode Manager Concept
- Service Mapping
- Coming up next



Motivation

1. The BSW shall satisfy specific requirements for an ECU
 - ▶ "Wakeup and Sleep" functionality in the following cases, for example:
 - > ECU operation (μ C and local peripherals)
 - > Communication channel handling (vehicle networks)
2. Abstraction of technology from the application (same treatment of different basic technologies like CAN, LIN, FR, ETH)
3. Generic concept for (both user-defined and BSW-defined) mode management
 - ▶ Mechanisms for notification
 - ▶ Service requests for mode changes (e.g., by SWCs)
 - ▶ Mode arbitration (decision on how to handle transitions)
 - ▶ Mode control (execution of mode transitions)



ECU State Manager

ECUM

- ▶ AUTOSAR 4.x defines
 - ▶ ECUM Flexible (supported by Vector)
 - ▶ ECUM Fixed (similar to AUTOSAR 3.x, no **dedicated module*** from Vector)
- ▶ Manages fundamental ECU phases like
 - ▶ **OFF, UP** and **SLEEP**
 - ▶ The necessary transitions between them
- ▶ Initializes and de-initializes BSW, OS, and RTE
- ▶ Wakeup event handling
- ▶ Coordination of RUN requests from SWCs and BSWM
- ▶ Two startup and two shutdown phases



*) ECUM Fixed is **natively** supported by MICROSAR, but it is also possible to configure the **ECUM Flexible** automatically such that the behavior matches to the **ECUM Fixed** mode machine.

In order to do this, the BSWM has to be configured with the appropriate rules. Up until MICROSAR 4 Release 7, the configuration of the **ECUM Flexible** in the role of the **ECUM Fixed** is something the integration engineer has to configure manually. With MICROSAR 4 Release 8 and higher, the AUTO Configuration feature exists where the ESH (ECU State Handling) can be configured with a few clicks.

Basic Software Mode Manager BSWM

BSWM

- ▶ The BSWM is a generic, **rule-based** service module
 - ▶ Used for mode arbitration and mode control
- ▶ It can have a configured set of **rules** containing **default states**, **conditions**, and **actions**
 - ▶ Each **condition** evaluates a set of **logical expressions***
 - > Evaluation result for a rule is either TRUE or FALSE
 - ▶ **Action list(s)** must be given for at least one evaluation result per rule
 - > For example: call existing APIs, do a notification, perform a user-defined callout
- ▶ Operation principle for rule evaluation
 - ▶ Caller's context (**IMMEDIATE**)
 - ▶ BswM_MainFunction context (**DEFERRED**)
- ▶ The `BswM_MainFunction` serves to implement
 - ▶ **DEFERRED** mode request ports, BSWM Timer, RTE Mode Switch Actions



*Logical expressions contained in conditions are composed of atomic expressions. This corresponds to the common way to handle conditions in boolean expressions like the "if" statement in the C programming language.

Basic Software Mode Manager BSWM

1. BSW Modes (Standardized C-API)

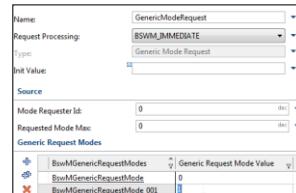
- ▶ `BswM_<BSW>_currentState/Mode`
Standardized C language API
- ▶ Functionality built into the Basic Software. Can be *Mode Switch Notifications* from BSW Modules acting as Mode Managers, but also *Mode Requests*, e.g., by the LINTP to switch the LIN schedule in the LNSM

2. SWC Modes

- ▶ `Rte_Switch / Rte_Mode` API, AUTOSAR Interface, i.e., "Ports"
- ▶ The SWC notifies its mode switches to the BSWM over a *Mode Declaration Group*, either directly or indirectly. The BSWM rules which react on the Mode Switch can trigger an action list.

3. Generic Modes (not using the RTE)

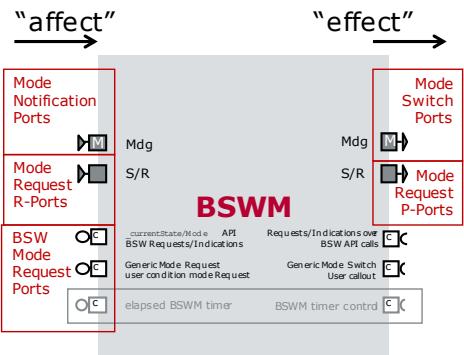
- ▶ User defined C API, e.g.
`BswM_RequestMode(MReqId0, BswMGenericRequestMode_01)`



Generic Mode: If there is no built-in `BswM_<BSWM>_currentMode/State` Notification like from the BSW and there is no Mode Switch Notification over the RTE, the Generic Mode can be used. Generic Modes are arbitrarily defined (not in terms of a Mode Declaration Group!).

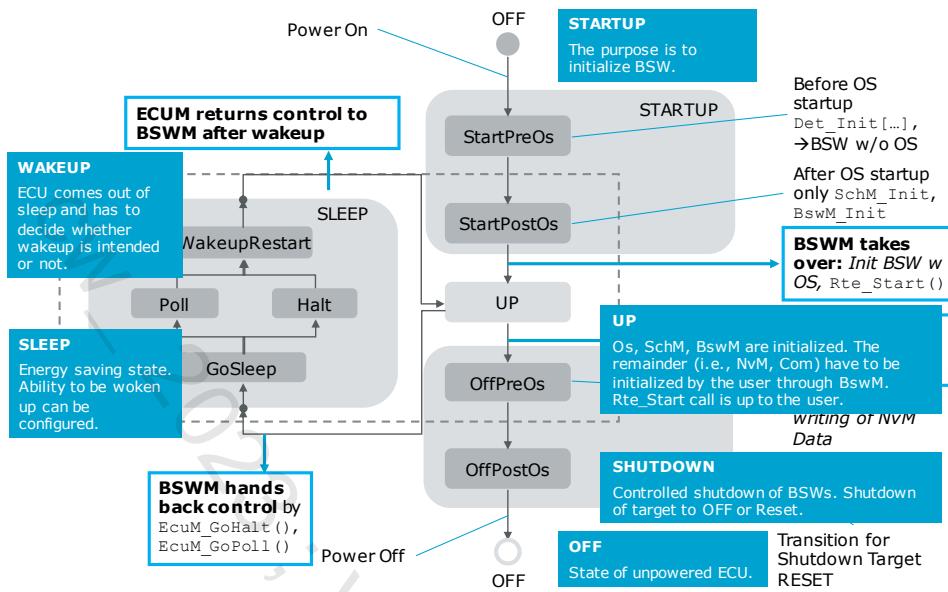
Basic Software Mode Manager BSWM

- ▶ **Mode Notification Port (R-Port)**
 - ▶ SWC indicates Mode to BSWM over a Mode Declaration Group
- ▶ **Mode Switch Port (P-Port)**
 - ▶ BSWM notifies Mode switch to a SWC using a Mode Declaration Group
- ▶ **Mode Request R-Port**
 - ▶ SWC requests Mode from BSWM over S/R
- ▶ **Mode Request P-Port**
 - ▶ BSWM requests Mode from a SWC Mode Manager over S/R

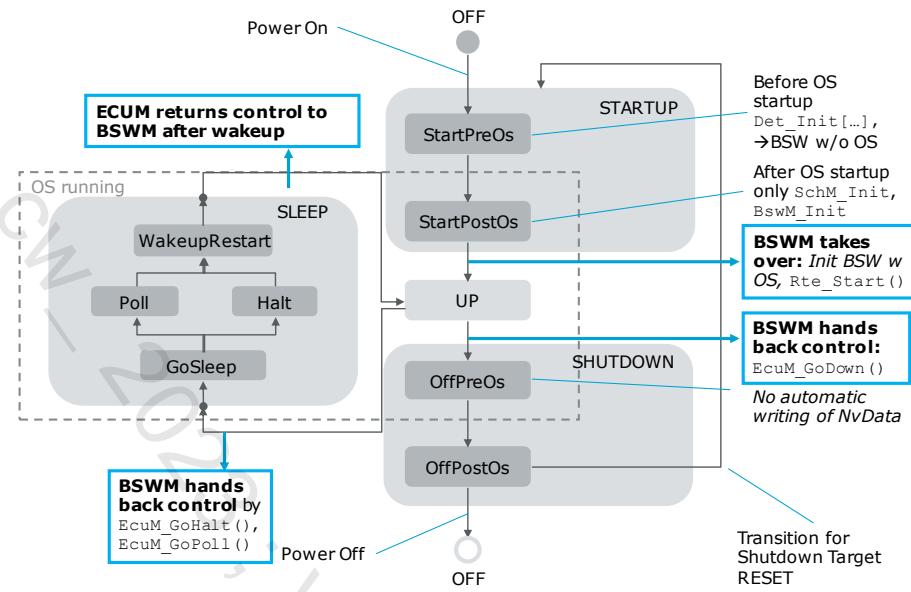


Module representation of the BSWM. The Ports labelled with "C" are only Standardized Interfaces or C APIs. Mode Ports (Mdg) or S/R Ports are AUTOSAR interfaces. Timers are only used internally in the BSWM; therefore they are printed in grey.

ECU State Manager (Flexible)



ECU State Manager (Flexible)



ECU State Manager (Flexible)



Here the ECU state machine is implemented as general modes and - during the UP Phase - under the control of the BSW Mode Manager module.

STARTUP PHASE: The purpose of this phase is to initialize BSW. It is divided into two parts, one before OS is started and the other after OS is started. In this phase, `SchM_Init()` and `BswM_Init()` must be called in the given sequence such that the StartPostOs already works based on the SchM Task(s) and an operating BSWM.

UP PHASE: This phase is entered after Os, SchM and BSWM have been initialized by the ECU State Manager. Please note that the restoring of NVRAM blocks or the communication stack initialization is not automatically done. Further, the RTE is not yet running as this is also something the user has to take care of. During UP phase, the ECU goes from State to State and from Mode to Mode, as modelled by the integrator-defined state machine and the help of BSWM. The BSWM implements the desired state machine which is required to restore and save NVRAM data, initialize and deinitialize the Communication stack and so on. Optionally, SLEEP modes can be entered from within the UP Phase of the ECUM. This phase must be requested by the application (explicitly or implicitly) whenever it is needed to keep the ECU awake.

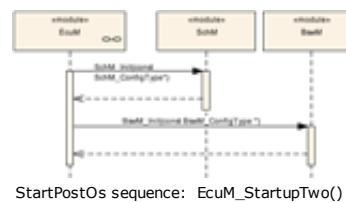
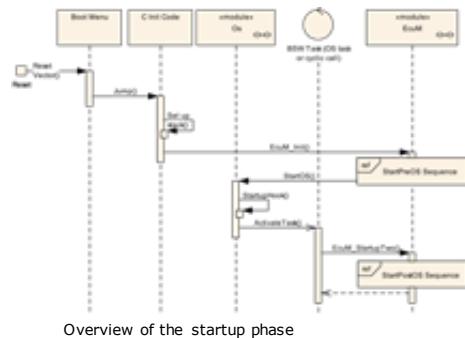
SLEEP PHASE: This is an energy saving state. Typically no code is executed but power is still supplied and, if configured, the ECU is able to be woken up in this state. The sleep can either halt execution of the processor or poll for wakeup events. **Wakeup Restart:** Wakeup is reached when the ECU comes out of **SLEEP** due to intended or unintended wakeup. (The **Wakeup validation protocol** allows to distinguish intended wakeup events from unintended wakeup events.)

SHUTDOWN PHASE: This phase will be entered by the call `EcuM_GoDown()`. The OffPreOs state will then run (NVRAM blocks should be written before). At the end of this state, `ShutdownOS()` will be called. Then the sequence `ShutdownHook() → EcuM_Shutdown()` will start the OffPostOs state. The shutdown phase handles controlled shutdown of BSW and results in shutdown target for the ECU: Off or Reset

OFF PHASE: This is the state of an unpowered ECU. The ability to wake up from this state may be required but only for wakeup sources with integrated power control. In any case the ECU must be able to be started (like through a reset).

STARTUP PHASE

- ▶ **StartPreOs** Sequence begins after `EcuM_Init()` and ends with `StartOs()`
 - ▶ **StartPreOs** corresponds to STARTUP I in AUTOSAR 3.x
- ▶ **StartPostOs** Sequence runs in a task with the property `AUTOSTART=true`
 - ▶ it begins with the call of `EcuM_StartupTwo()`¹
 - ▶ `BswM_Init()` executes the action list `BswM_ActionList_INIT_AL_Initialize`
 - ▶ **StartPostOs** corresponds² to STARTUP II in AUTOSAR 3.x



source: SWS
ECUM R4.1.1

STARTUP PHASE



- 1) The call of `EcuM_StartupTwo()` must be performed in an AUTOSTART Initialization Task which has the highest priority of all other (AUTOSTART) tasks in the system and is ideally non-preemptive. e.g.:

```
TASK(Init_Task)
{
    EcuM_StartupTwo();
    TerminateTask();
}
```

- 2) In contrast to **STARTUP II** in AUTOSAR 3.x, neither ECUM nor **StartPostOs** Sequence restore the NV Blocks or start the RTE automatically.

The restoring of NV Data is up to the user (literally the BSWM), as well as the call `Rte_Start()`.

Moreover, the permission for bus communication has to be granted by the user (or BSWM respectively):

The Call of `ComM_AllowCommunication()` is required in order to allow the `ComM_RequestComMode(ComM_User, COMM_FULL_COMMUNICATION)` to be successful.

The user has to configure the **BSWM** accordingly to manage these issues during the **UP PHASE**.

At the point where `BswM_Init` is called, the BSWM can already immediately evaluate a rule which reacts on the transition into the `BswM_INIT` state and hence take over the ECU Management from this point in time.

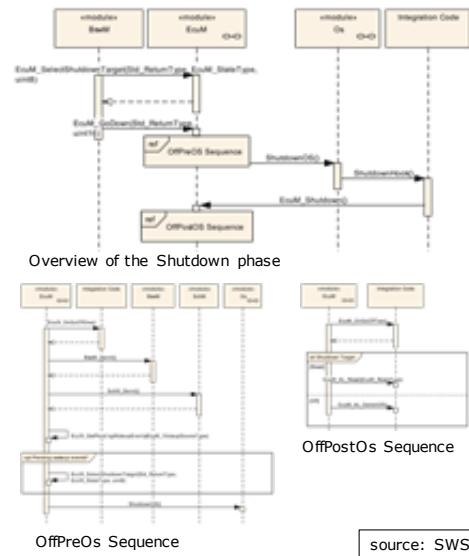
Such a rule contains actions which can be the following:

- Loading the NV Data over `NvM_ReadAll()`
- Granting the possibility to go into `COMM_FULL_COMMUNICATION`: `ComM_AllowCommunication()`
- Starting the RTE over `Rte_Start()` API

(In the simplest case this could be what is called `EcuM_InitListTwo()` in AUTOSAR 3.x, i.e. the `InitListTwo` would finally call `ComM_AllowCommunication` and start the RTE.)

SHUTDOWN PHASE

- ▶ **OffPreOs** Sequence begins by the BSWM calling `EcuM_GoDown(ModuleId caller)` and ends by the ECUM calling `ShutdownOs()`
 - ▶ **OffPreOs** corresponds to GO OFF I in AUTOSAR 3.x
- ▶ **OffPostOs** Sequence begins with the call of `EcuM_Shutdown()`
 - ▶ In the `ShutdownHook()` the integrator has to insert the call to `EcuM_Shutdown()`
 - ▶ It ends in `EcuM_A1_SwitchOff()`
 - ▶ **OffPostOs** corresponds* to GO OFF II in AUTOSAR 3.x



source: SWS
ECUM R4.1.1

SHUTDOWN PHASE



*) In contrast to **GO OFF II** in AUTOSAR 3.x, neither the ECUM itself nor the **OffPostOs** Sequence stop the RTE or write NV Blocks. The stopping of the RTE (`Rte_Stop()`) and the writing of the `NvM_WriteAll` Blocks is the job of the user.

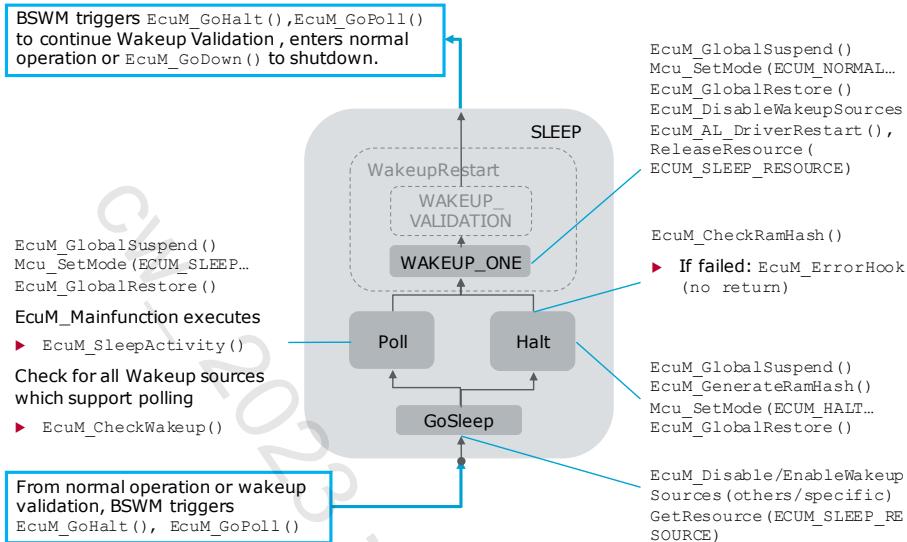
The user has to configure the BSWM accordingly to manage these during the **UP PHASE** already.

If a handling for the RTE and NVM is needed later on, the last possibility is during `EcuM_OnGoOffOne()`, because directly afterwards the BSWM and SCHM become deinitialized.

For the Shutdown, the **BSWM** could have a rule which reacts on an application state. If the application requests the shutdown e.g. over a Mode Request, the BSWM could

- Stop the RTE over `Rte_Stop()`
- Deinitialize the DEM (`Dem_Shutdown()`) and trigger the NVRAM to be written (`NvM_WriteAll()`)
- Call `EcuM_GoDown()` to shutdown the system

SLEEP PHASE: Poll or Halt



SLEEP PHASE: Poll or Halt



GoSleep:

The ECUM prepares the hardware for going to sleep and setting the WakeUp sources, e.g., `EcuM_EnableWakeupSources`, as well as occupying the scheduler resource.

Poll:

The ECUM checks for pending `EcuMWakeupSourcePolling` wakeup sources cyclically by calling `EcuM_CheckWakeup()`. Auxiliary `EcuM_SleepActivity()` must be called for, e.g., updating the alarm clock. This activity will be performed through the `EcuM_Mainfunction`, which is still called while the scheduler resource of the OS is occupied.

Halt:

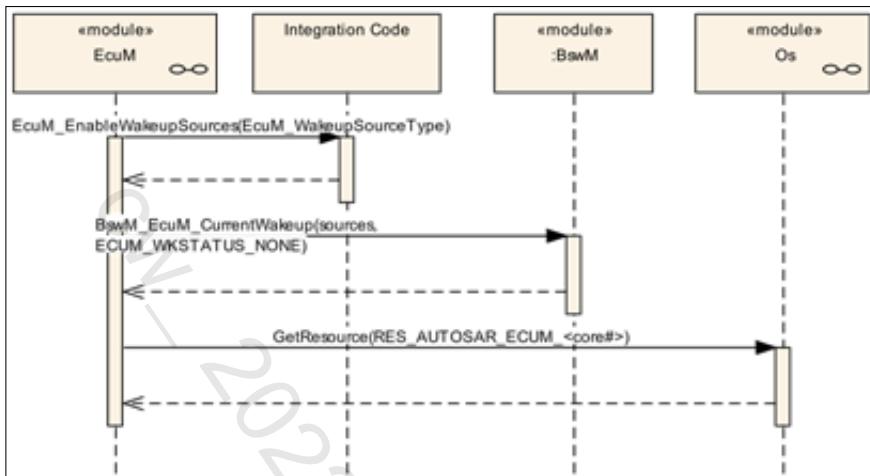
No more code is executed. Before halting the MCU, ECUM must invoke `EcuM_GenerateRamHash` and call `EcuM_CheckRamHash` before leaving Halt. On Multicore: only check the master core.

Wakeup Restart:

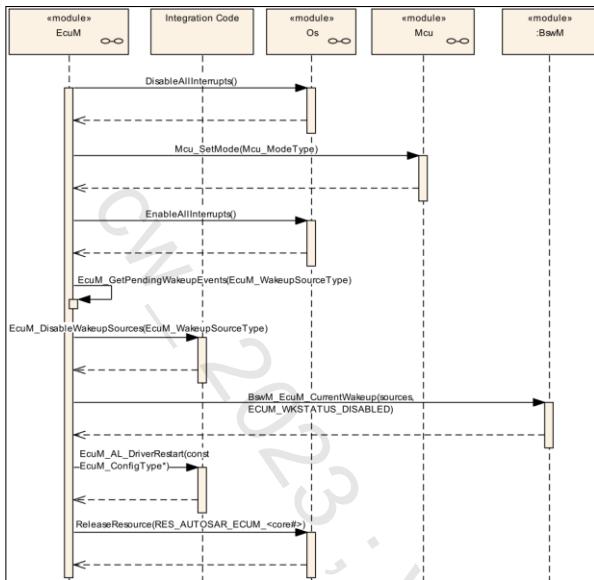
WAKEUP_ONE:

- 1) Restore MCU normal mode
- 2) Get Pending Wakeup Sources
- 3) `EcuM_DisableWakeupSources`
- 4) `EcuM_AL_DriverRestart`
- 5) Release scheduler resource

If after WAKEUP_ONE `ValidatedWakeups` is equal to False, the BSWM can trigger the transition back into GoSleep over `EcuM_GoHalt/EcuM_GoPoll` again, or even call `EcuM_GoDown`. Every pending event is validated independently (if configured) and the ECUM publishes the result as a mode request to the BSWM, which in turn can trigger state changes in the ECUM

SLEEP PHASE: GoSleep

SLEEP PHASE: WakeupRestart



15

Agenda

Mode Management

► **Wakeup Handling**

BSWM configuration

ECUM configuration

Mode Management

Network Management Algorithm

Mode Manager Concept

Service Mapping

Coming up next

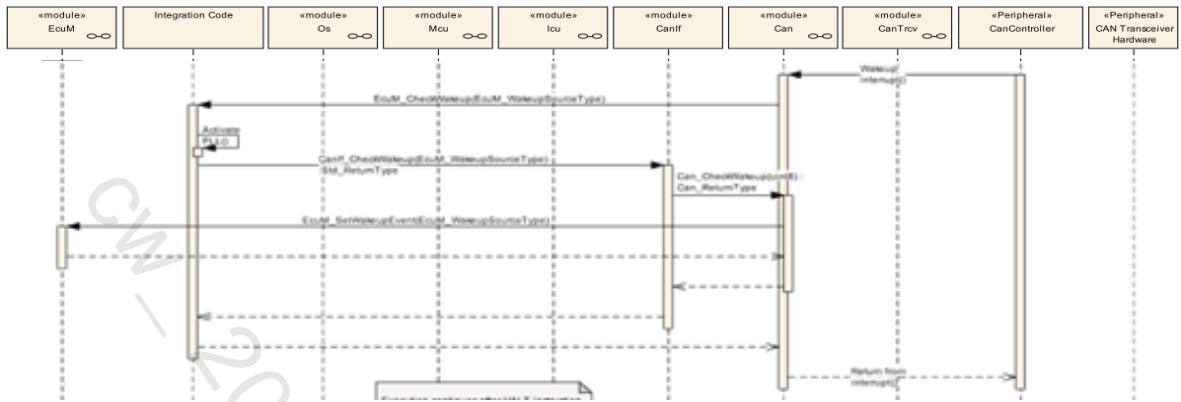
ECUM Wakeup Sources

- ▶ The AUTOSAR Wakeup procedure consists of two steps
 1. Check Wakeup Source & report Wakeup Event
 2. Wakeup Validation Protocol
- ▶ Wakeup procedure independent from ECUM SLEEP PHASE
 - ▶ But used to wakeup from the SLEEP PHASE
- ▶ Each ECU owns a set of Wakeup Sources (maximum 32 sources)
 - ▶ Wakeup is coordinated by the ECUM
 - ▶ There are predefined Wakeup Sources
 - ▶ these Wakeup Sources do not need any validation

ECUM_WKSOURCE_POWER
ECUM_WKSOURCE_RESET
ECUM_WKSOURCE_INTERNAL_RESET
ECUM_WKSOURCE_INTERNAL_WDG
ECUM_WKSOURCE_EXTERNAL_WDG

- ▶ Up to 27 user defined Wakeup Sources
- ▶ Each Wakeup Source either
 - ▶ Needs validation or
 - ▶ Is validated per default

Cooperation of CAN, CANIF and ECUM



- ▶ One possible wakeup scenario
 - ▶ CAN Controller Wakeup by Interrupt

Cooperation of CAN, CANIF and ECUM



Generic Elements: ECU firmware (file `EcuM_Callout_Stubs.c`)

- Functions that have to be implemented by the ECU supplier
- Function templates will be provided
- Intention of this firmware is to have the possibility to add project-specific code for mode changes
- In the exercise of this chapter the behavior shown above in function `EcuM_CheckWakeups()` has to be implemented

The ECUM handles the Wakeup over the APIs:

`EcuM_CheckWakeups(wakeupSource)` is called by the hardware from interrupt context. Depending on the origin of the interrupt, the call must be deferred to `<Bus>If_CheckWakeups(wakeupSource)` function.

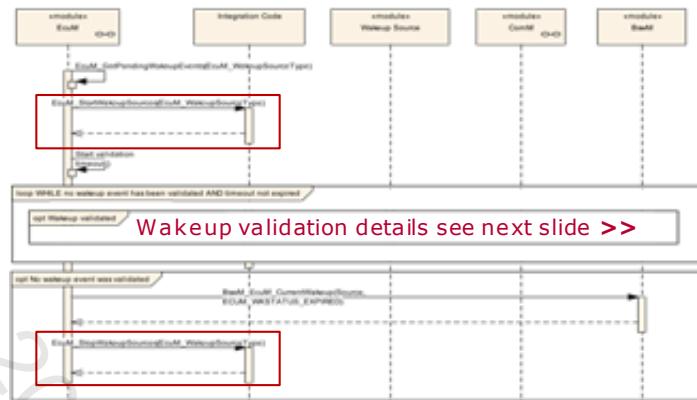
The BusIf will then check the corresponding driver which is registered for the reported wakeup Source, e.g., `<Bus>_CheckWakeups`. The driver's CheckWakeups function will then report `EcuM_SetWakeupsEvent()` in case it has detected a confirmed wakeup. This event will then be recorded in the pending wakeup events list.

How to exclude Electromagnetic Interference (EMI):

The sequence from above may look a little awkward, but it is really required in some cases (however it could be skipped in our case). If you have an SBC (System Basis Chip) or transceiver chip, and an Interrupt arrives which was caused by an ICU edge detection on the Rx pin at the µC Rx line, we need to double-check by reading out the SBC/transceiver state to confirm it was a wakeup event caused by the CAN bus and not an EMI spike between transceiver and controller on the Rx pin. Therefore the calling sequence is:
`EcuM_CheckWakeups → CanIf_CheckWakeups → Can_CheckWakeups OR CanTrcv_CheckWakeups`, respectively. This step is a validation on physical layer.

→ This step is NOT YET the wakeup validation, which is a plausibility check on data link layer!

Wakeup Validation



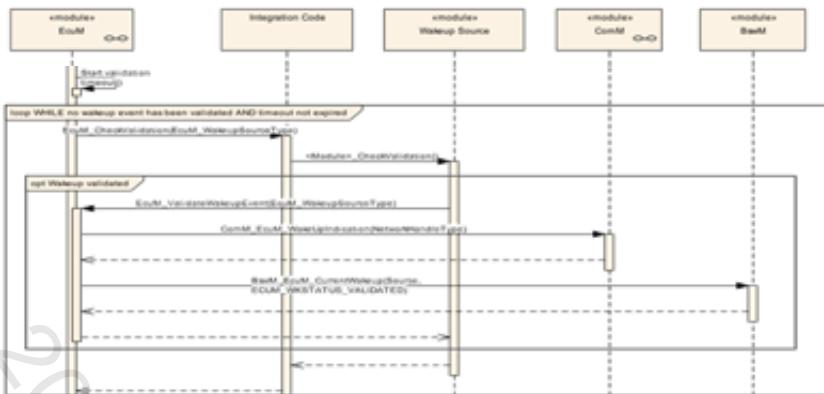
- ▶ The wakeup sources have to be **started** before the wakeup validation can proceed (`EcuM_StartWakeupSources`)
- ▶ after the failed validation the wakeup sources have to be **stopped** (`EcuM_StopWakeupSources`)

Wakeup Validation



- A wakeup source can have one of the following states:
 - ECUM_WKSTATUS_NONE: no wakeup was detected
 - ECUM_WKSTATUS_PENDING: wakeup was detected and reported but not validated
 - ECUM_WKSTATUS_VALIDATED: wakeup event was successfully validated
 - ECUM_WKSTATUS_EXPIRED: validation failed
 - ECUM_WKSTATUS_ENABLED: wakeup source is active and can report `EcuM_SetWakeupEvent()`
 - ECUM_WKSTATUS_DISABLED: wakeup source is disabled and cannot produce wakeup events.

Wakeup Validation



- ▶ CAN Wakeup Validation
- ▶ can be enabled for each Wakeup Source separately
(configure `EcuMValidationTimeout != 0`)

Wakeup Validation



As soon as the wakeup validation has succeeded from `EcuM_ValidateWakeupEvent()`, the event is removed from the pending wakeup events and added to the validated wakeup events.

If this cannot be accomplished, the Validation Timeout expires and the wakeup source would be considered expired (`ECUM_WKSTATUS_EXPIRED`).

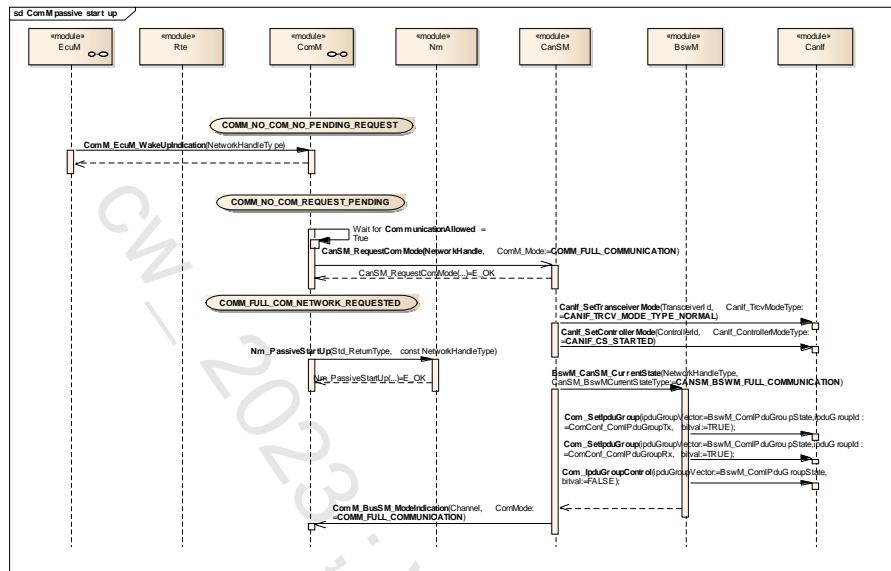
Depending on the outcome of the wakeup validation, the BSWM will be informed about the result.

The Wakeup source is `ECUM_WKSTATUS_VALIDATED` in our example. If this is not successful, it will become `ECUM_WKSTATUS_EXPIRED`. The validated Wakeup events become notified to the BSWM over `BswM_EcuM_CurrentWakeupEvent(wakeupSource, ECUM_WKSTATUS_VALIDATED)`.

The call `BswM_EcuM_CurrentWakeupEvent(wakeupSource, ECUM_WKSTATUS_VALIDATED)` makes it possible to setup BSWM rules which perform actions on a wakeup source. This is especially the case when the wakeup source does not have a COMM Channel Reference and must be treated differently (because no `ComM_EcuM_WakeUpIndication` can be used in this case)

If the wakeup source has a COMM Channel Reference, the ECUM further calls `ComM_EcuM_WakeUpIndication(NetworkHandleType Channel)` to notify the COMM to turn the communication channel on.

Cooperation of ECUM, COMM, BSWM, NM, CANSM at wakeup



21

Agenda

- Mode Management
- Wakeup Handling
- ▶ **BSWM configuration**
- ECUM configuration
- Mode Management
- Network Management Algorithm
- Mode Manager Concept
- Service Mapping
- Coming up next

Auto Configuration

- ▶ Fundamentally different between AUTOSAR 3 and AUTOSAR 4
 - ▶ ECUM AUTOSAR 3 (ECUM Fixed AUTOSAR 3)
 - ▶ ECUM Flexible (AUTOSAR 4)

However, for compatibility reasons, it would be nice to configure the AUTOSAR 4 ECUM + BSWM such that you obtain an ECUM Fixed according to AUTOSAR 3!

- ▶ This is what Vector's *Auto Configuration* feature of the BSWM can do!
- ▶ Auto configuration exists for
 1. ECU state handling
 2. Module initialization
 3. Communication handling

Auto Configuration: ECU State Handling (ESH)

- ▶ The BSWM supports “pre-configured state machines”
- ▶ The Rules, Actions, Conditions, etc. they create are only a suggestion and may be edited by the integrator afterwards

Configure Ecu State Handling

Use this hyperlink to configure the Ecu State Handling settings.
The currently configured settings are displayed below.

Ecu State Machine: Enables or disables Ecu State Handling.

<input checked="" type="checkbox"/> Ecu State Machine	Self Run Request Timeout	0.1
<input type="checkbox"/> Den Handling [Not available]	Number of Run Request User	1
<input type="checkbox"/> Support CommM	Number of PostRun Request User	1
<input type="checkbox"/> NvM Handling [Not available]		
<input type="checkbox"/> Rte Mode Synchronistaion		
<input type="checkbox"/> Enable Callouts		

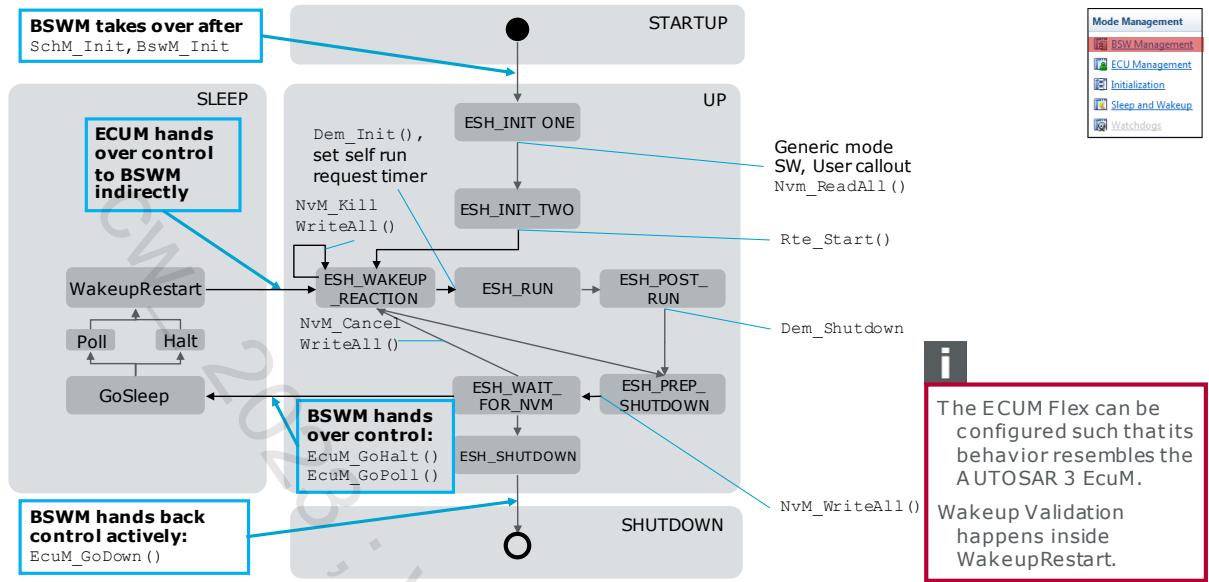
Support CommM
allows communication

Most recent configuration shown
(not editable in this view!)

Self Run Request = 0
restores the AUTOSAR 4 behavior

Mode Management
BSW Management
ECU Management
Initialization
Sleep and Wakeup
Watchdogs

Auto Configuration: ECU State Handling (ESH)



Auto Configuration: Module Initialization

- ▶ Initialization of Basic Software Modules after OS Start
- ▶ BSWM lists all modules in the ECU configuration
 - ▶ Can be configured by setting the checkbox of each module separately
 - ▶ Modules listed alphabetically in the GUI, not chronologically. Actual initialization sequence can be seen in the **action list**
- ▶ BSW Modules which the BSWM does not know are listed in an extra category
 - ▶ <BSW>_Init Function and header must be entered by the integrator
 - ▶ Unknown modules will be initialized at the end of the **action list**
- ▶ The initialization order within the **action list** can be rearranged afterwards



Auto Configuration: Module Initialization

- ▶ After Auto Configuration, a reordering of BSW Module initialization steps is possible
- ▶ If the Auto Configuration for the Module Initialization is executed again, the default sequence will be restored

Mode Management
BSW Management
ECU Management
Initialization
Sleep and Wakeup
Watchdogs

2. Move selected module up / down in the initialization order

1. Select the module initialization entry



Auto Configuration: Module Initialization → Action Lists → INIT_AL_Initialize

Auto Configuration: Communication Assistant

- BSWM now provides an Auto Configuration for Communication Handling.

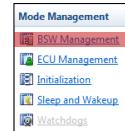
[Configure Communication Control](#)
Use this hyperlink to configure the Communication Control settings.
The currently configured settings are displayed below.

CAN00_f26020e5: Handle this Channel	
<input type="checkbox"/>	RelInitialize TX
<input type="checkbox"/>	RelInitialize RX
<input type="checkbox"/>	DCM_ENABLE_RX_DISABLE_TX_NORM
<input type="checkbox"/>	DCM_DISABLE_RX_ENABLE_TX_NORM
<input type="checkbox"/>	DCM_DISABLE_RX_TX_NORMAL
<input checked="" type="checkbox"/>	DCM_ENABLE_RX_DISABLE_TX_NM
<input type="checkbox"/>	DCM_DISABLE_RX_ENABLE_TX_NM
<input type="checkbox"/>	DCM_DISABLE_RX_TX_NM
<input checked="" type="checkbox"/>	DCM_ENABLE_RX_DISABLE_TX_NORM_NM
<input type="checkbox"/>	DCM_DISABLE_RX_ENABLE_TX_NORM_NM
<input checked="" type="checkbox"/>	DCM_DISABLE_RX_TX_NORM_NM

Mode Management
BSW Management
ECU Management
Initialization
Sleep and Wakeup
Watchdogs

Auto Configuration: Communication Assistant

- ▶ Analyzing all PDU Groups and assigning them to a channel
 - ▶ PDU Groups which have PDUs of different channels can not be handled and will be listed as 'Not available'
- ▶ Switching of PDU Groups for CAN, LIN, FR, and J1939
 - ▶ User can choose to reinitialize I-PDU Groups or not (re-init TX / re-init Rx parameter)
 - ▶ CAN re-initialization will only performed if the Bus State changes from NO_COM to FULL_COM. In case of BUS_OFF or SILENT no re-initialization will be performed
 - ▶ CAN, FR: Enabling/Disabling the NM
 - ▶ DCM Modes can be considered for
 - ▶ Switching PDU Groups (CAN, FR)
 - ▶ Enabling/Disabling NM
 - ▶ DCM Modes are configurable per channel



Auto Configuration: Communication Assistant

- ▶ Partial Network Clusters (PNCs) are supported for CAN / FR
 - ▶ If a PDU Group can be assigned to a PNC, PDU Group is listed as a sub-feature of corresponding PNC and will be switched on/off depending on the PNC Status
 - ▶ A PNC can only be determined if there are PNC mapping entries in the System-Description which are transferred to the FlatEcuc.arxml file
- ▶ LIN
 - ▶ Only I-PDU Groups will be switched; no LIN Schedule handling is available
 - ▶ PDU Groups which contain only PDUs of the same Schedule will be switched depending on the schedule status

Mode Management
BSW Management
ECU Management
Initialization
Sleep and Wakeup
Watchdogs



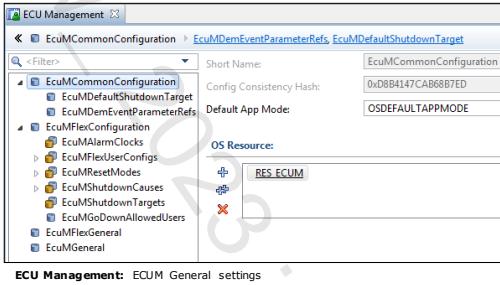
- ▶ Nm Mode will be forwarded to RM and DCM if the corresponding Node is configured there too
- ▶ PDU Groups which contains only PDUs of the same Node will be switched depending on Node Status
- ▶ PDU Groups which are determined as Broadcast Groups will be switched depending on Dcm Broadcast status

Agenda

- Mode Management
- Wakeup Handling
- BSWM configuration
- ▶ **ECUM configuration**
 - Mode Management
 - Network Management Algorithm
 - Mode Manager Concept
 - Service Mapping
 - Coming up next

ECUM General settings

- ▶ Since most of the work is done by the BSWM, the ECUM has become a very slim module
- ▶ However, a number of things can still be configured in the ECUM configuration
 - > Os Application Mode
 - > Scheduler Resource
 - > DefaultShutdownTarget
 - > Multicore operation
 - > ...



Mandatory modules

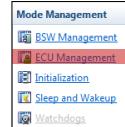
- ▶ AUTOSAR OS
- ▶ MCU driver
- ▶ DEM ("include DEM" switch has no meaning. Actual DEM usage will be determined by configured production errors)
- ▶ COMM
- ▶ SCHM

Shutdown related topics

- ▶ Reset modes specialize the Shutdown Target ECUM_STATE_SLEEP or ECUM_STATE_RESET
 - ▶ Identify the reason for an upcoming ECU reset
 - ▶ Selected by using `EcuM_SelectShutdownTarget`
 - ▶ Callout `EcuM_AI_Reset(EcuM_ResetType)`
 - ▶ User-specified implementation
- ▶ Select shutdown cause prior to shutting down
 - ▶ `EcuM_SelectShutdownCause`
 - ▶ ECUM_CAUSE_ECU_STATE - ECU state machine entered a state for shutdown
 - ▶ ECUM_CAUSE_WDGM - WdgM detected failure
 - ▶ ECUM_CAUSE_DCM - Dcm requests shutdown
 - ▶ User-configurable causes

E DCM, ECUM_CAUSE_UNKNOWN, ECUM_CAUSE_WDGM, ECUM CAU	
EcuMShutdownCauses	Shutdown Cause Id
ECUM_CAUSE_DCM	3
ECUM_CAUSE_ECU_STATE	1
ECUM_CAUSE_UNKNOWN	0
ECUM_CAUSE_WDGM	2

E FCUM RESET IO, ECUM RESET MCU, ECUM RESET WAKEUP	
EcuMResetModes	Reset Mode Id
ECUM_RESET_IO	2
ECUM_RESET MCU	0
ECUM_RESET_WAKEUP	3
ECUM_RESET_WDG	1



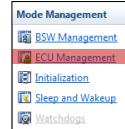
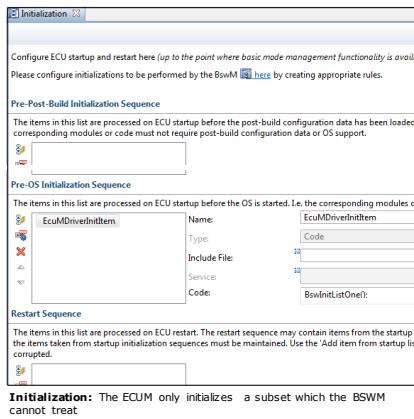
Shutdown related topics



- ▶ The reset modes can be used to identify the reason for an upcoming ECU reset. A set of reset modes is defined by the AUTOSAR standard. Additional modes can be added by the configuration. The reset mode is passed over to the callout EcuM_AL_Reset(EcuM_ResetType) and the user can implement different ways to reset the ECU, depending on the reason for this reset.
- ▶ The Vector extension ECUM_RESET_WAKEUP is used as the reset mode in the case of a late wake-up event in the shutdown phase. If a wake-up occurs during the shutdown procedure, the shutdown target is changed by the EcuM to ECUM_STATE_RESET and the described mode is used.

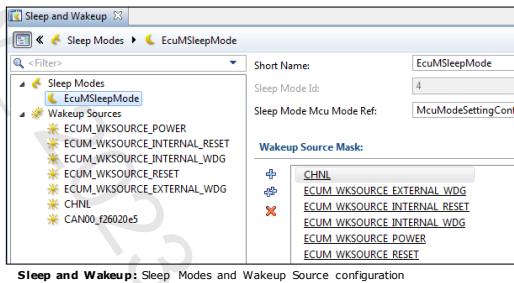
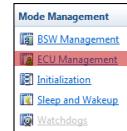
ECUM fundamental Initialization steps

- ▶ Startup Pre-OS
 - ▶ Pre-Post-Build and Pre-OS Initialization Sequences
- ▶ ECUM Restart Sequence
- ▶ Startup Post-OS
 - ▶ Handled by the BSWM



ECUM Sleep Modes and Wakeup Sources

- ▶ Wakeup Sources
 - ▶ These wakeup paths exist in the ECU
- ▶ Sleep Mode configuration
 - ▶ Each Sleep Mode has a MCU Mode and an associated set of possible wakeup sources to return into RUN



Sleep and Wakeup: Sleep Modes and Wakeup Source configuration

Agenda

Mode Management

Wakeup Handling

BSWM configuration

ECUM configuration

► **Mode Management**

Network Management Algorithm

Mode Manager Concept

Service Mapping

Coming up next



Communication Manager

COMM

- ▶ Controls the BSW regarding communication
- ▶ Coordinates bus communication requests from SWCs
- ▶ COMM cannot keep the ECU actively awake
 - ▶ ECUM Fixed or BSWM (ECUM Flexible) indicate to COMM when communication is allowed
 - > ComM_CommunicationAllowed API
 - ▶ For the transition of the ECU into shutdown, ECUM (or BSWM respectively) must call
 - > EcuM_GoDown API
- ▶ Uses NM of the networks to keep the networks awake
- ▶ Uses SM of the networks to control bus activity

Communication Manager



Network Management

- Keeps the bus awake as long as an ECU requires communication
- Coordinates the transition to bus sleep

State Manager

- Enables and disables the bus communication (incl. transceiver handling)
- Controls COM I-PDU communication on each network indirectly over BSWM (BSWM operates the Com IPDUGroup API based on the CANSM currentState)
- CAN Bus-off handling
- LIN Schedule Table handling

- If **ECUM Fixed** is used:

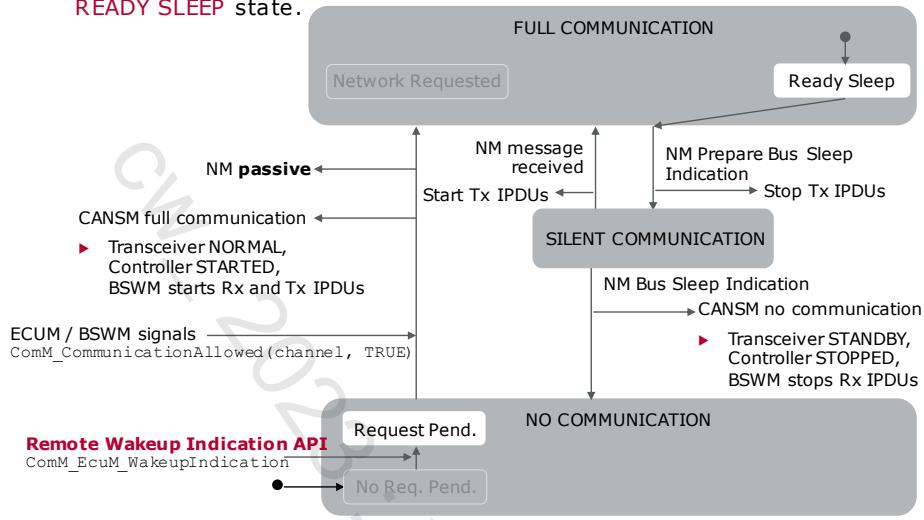
ECUM - Fixed needs to call and check `ComM_GetState()` before an ECU shutdown

- If **ECUM Flexible** is used:

BswM should use `BswM_ComM_RequestMode()` indications to know when ECU can be shutdown. You can configure a rule which has the condition `ComMIndication == COMM_NO_COMMUNICATION` and has a corresponding action `EcuM_GoDown()`.

Communication Manager – External Wakeup

Always ready to sleep, cannot keep the network awake → start leads to **READY SLEEP** state.



Communication Manager – External Wakeup



No communication (COMM_NO_COMMUNICATION)

ECU is not participating in bus communication. This mode is local for one channel, i.e., the ECU could communicate via otherchannels.
Run mode must not be requested.

Silent communication (COMM_SILENT_COMMUNICATION)

Sending of PDUs for the corresponding channel is forbidden. Receiving PDUs is permitted.

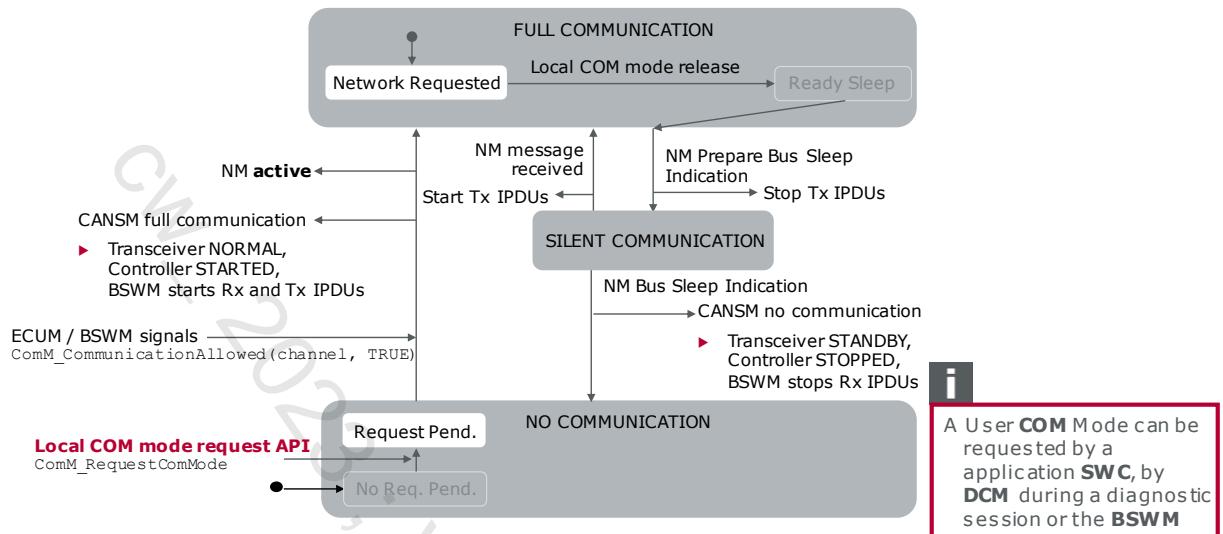
Full communication (COMM_FULL_COMMUNICATION)

Transmission and Reception capability of corresponding channels is switched to on.

- **NM** passive indicates that the **NM** does not actively keep the bus awake. No **NM** messages are therefore transmitted (except when being in Repeat Message state for a short time).
- The **BSWM** controls the transmission and reception path over the state of the bus specific state manager module. To accomplish this, the **BSWM** uses the **COM I-PDU-Group API**.
- A transition to "internal communication request" – see following slide – is possible at any time if an internal communication request is made by a **SWC** or **DCM**

Communication Manager – Internal Wakeup

can keep the network awake → start leads to **NETWORK REQUESTED** state



A User **COM** Mode can be requested by a application **SWC**, by **DCM** during a diagnostic session or the **BSWM**

State Manager

CANSM

- ▶ Translation of network communication mode requests
- ▶ Control of peripherals (indirectly over CANIF)
 - ▶ CAN Transceivers
 - ▶ CAN Controllers
- ▶ Control of IPDU groups of COM for
 - ▶ AUTOSAR version \leq 3.1: CANSM directly controls IPDU Groups
 - ▶ AUTOSAR version \geq 3.2: BSWM uses CANSM state to control IPDU Groups
- ▶ Handle the network mode via a separate state machine per network
- ▶ CAN Bus error management: Bus-off recovery via a separate state machine per network



Network Management

- Keeps the bus awake as long as an ECU requires communication
- Coordinates the transition to bus sleep

State Manager

- Enables and disables the bus communication (incl. transceiver handling) in conjunction with the CANIF.
- Controls COM I-PDU communication on each network for AUTOSAR 3.1 and lower, but not for AUTOSAR 3.2 and higher.
- CAN Bus-off handling
- LIN Schedule Table handling

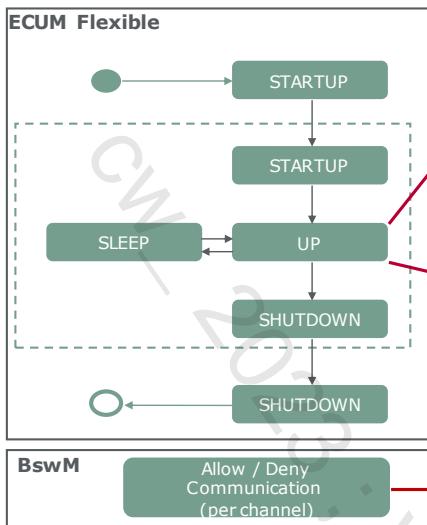
Network Management

NM

- ▶ Controlled transition of all ECUs into bus-sleep mode and vice versa
- ▶ Coordination functionality between different networks
- ▶ Node detection handling
- ▶ Abstraction of technology-specific NMs (CANNM, FRNM, LINNM, UDPNM) by a generic state machine

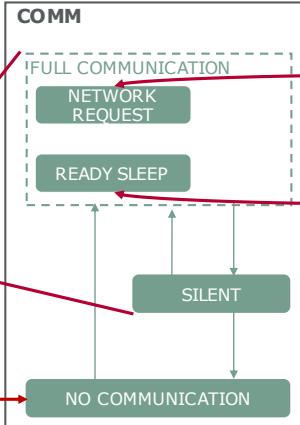
Interaction of ECUM, COMM, CANNM and BSWM

MCU State



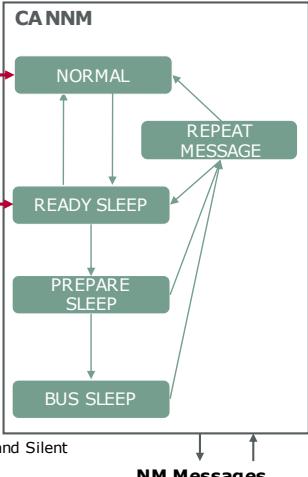
Network State

"legislative"



Network State

"executive"



The necessary ECUM state for Full Communication and Silent state of the COMM is the UP mode with allowed communication (e.g. by BSWM)

NM Messages

Network Management for CAN

CAN NM

The AUTOSAR NM is a **direct** and **distributed** Network Management

- ▶ **Direct NM** - Each node has its own NM message
 - ▶ The NM message is reserved for the Network Management
 - ▶ The NM message signalizes that bus communication is needed
 - ▶ The NM message data is **not relevant** for the NM mechanism
- ▶ **Distributed NM** - All nodes are equal (there is no NM Master or NM Slave)
 - ▶ Optional bus load reduction
 - ▶ Only two active nodes transmit NM messages
 - ▶ Avoids the bus load being dependent on the number of participating nodes
 - ▶ Partial Networking
 - ▶ Impose logical network clusters (PNCs)

Communication Manager – External Wakeup



The OSEK NM is also a direct and distributed Network Management.

The NM message data of OSEK NM is relevant for the NM mechanism as there are multiple NM message types

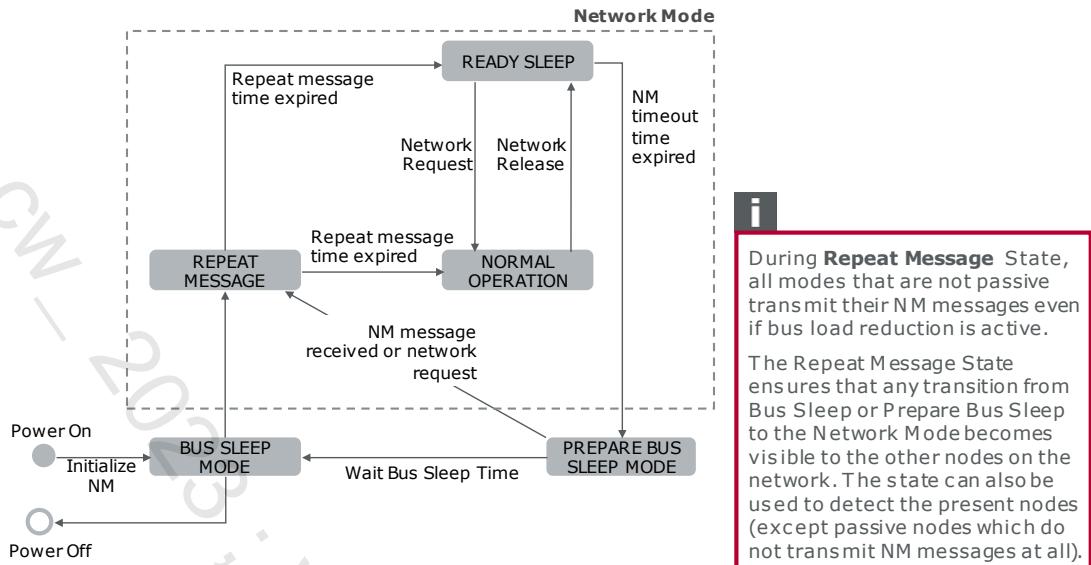
which can be differentiated by the message payload (Sleep Indication and Sleep Acknowledge fields)

- Alive message
- Ring message
- Limphome message

With AUTOSAR NM, there is only one kind of NM message without payload, there is no Sleep Indication/Acknowledge field at all.

However, there is an NM API to allow access to a user or OEM-defined payload content which is not part of the NM algorithm.

Network Management for CAN



Agenda

- Mode Management
- Wakeup Handling
- BSWM configuration
- ECUM configuration
- Mode Management
- ▶ **Network Management Algorithm**
- Mode Manager Concept
- Service Mapping
- Coming up next

Basic Mechanism – Single ECU (CAN)

**Repeat Message**

Each ECU transmits its own NM message cyclically for NM_REPEAT_MSG_TIME.

Ready Sleep

ECU is ready to sleep, no NM message transmission, restart of timeout timer upon NM message Rx.

Normal

NM message transmission and restart of timeout timer upon NM message Rx and Tx.

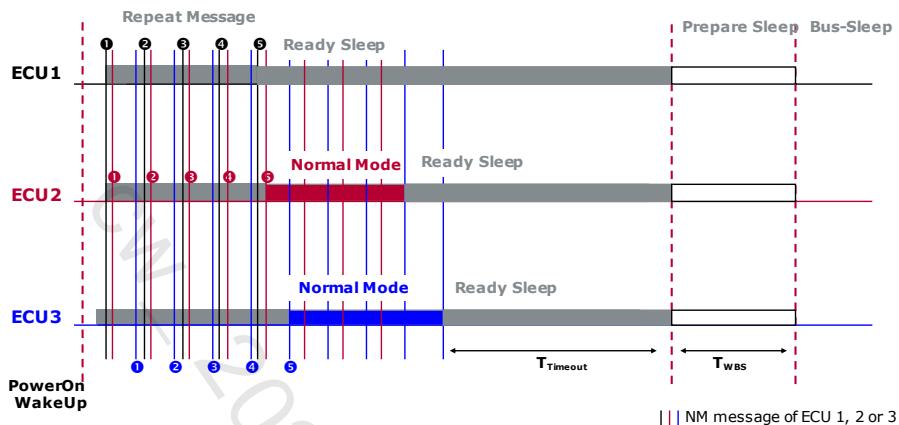
Prepare Sleep

If NM_TIMEOUT_TIME expired and no NM message has been transmitted or received meantime.

Bus-Sleep

After NM_WAIT_BUS_SLEEP_TIME transition to bus-sleep mode.

Basic Mechanism – Network (CAN)



Repeat Message: Each ECU transmits its own NM message cyclically for NM_REPEAT_MSG_TIME.

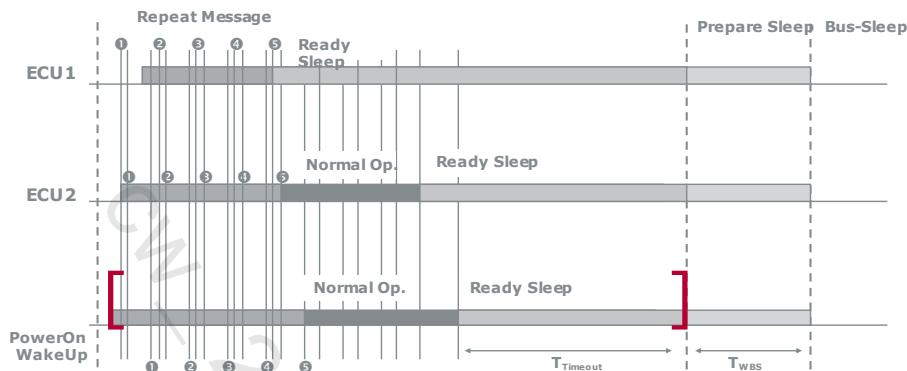
Ready Sleep: ECU is ready to sleep, no NM message transmission, restart of timeout timer upon NM message Rx.

Normal: NM message transmission and restart of timeout timer upon NM message Rx and Tx.

Prepare Sleep: If NM_TIMEOUT_TIME expired and no NM message has been transmitted or received in the meantime.

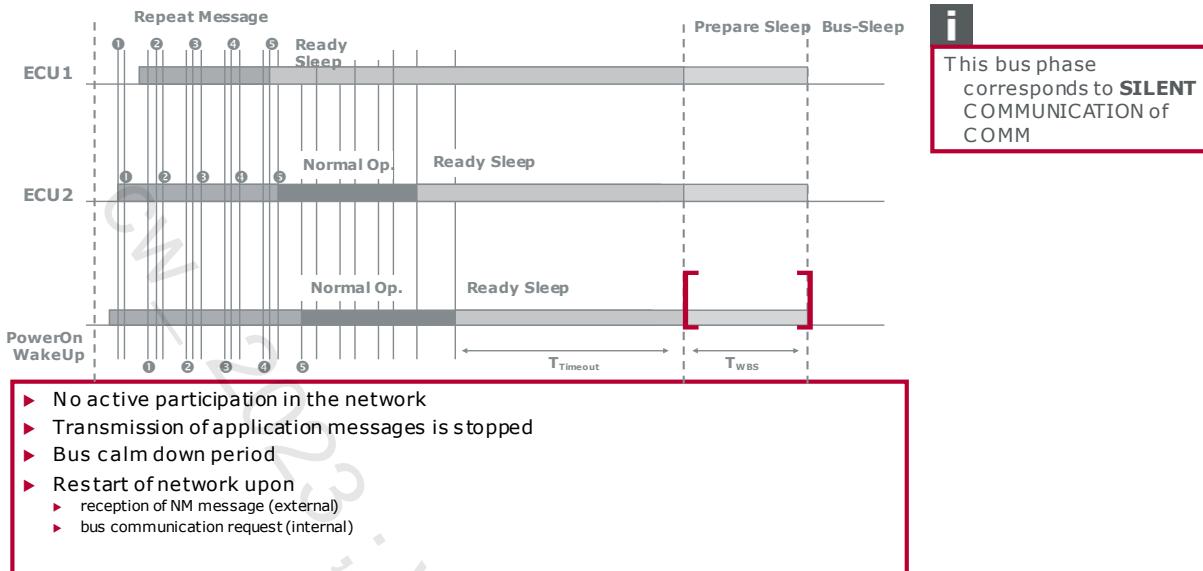
Bus-Sleep: After NM_WAIT_BUS_SLEEP_TIME transition to bus-sleep mode.

Network Mode (CAN)

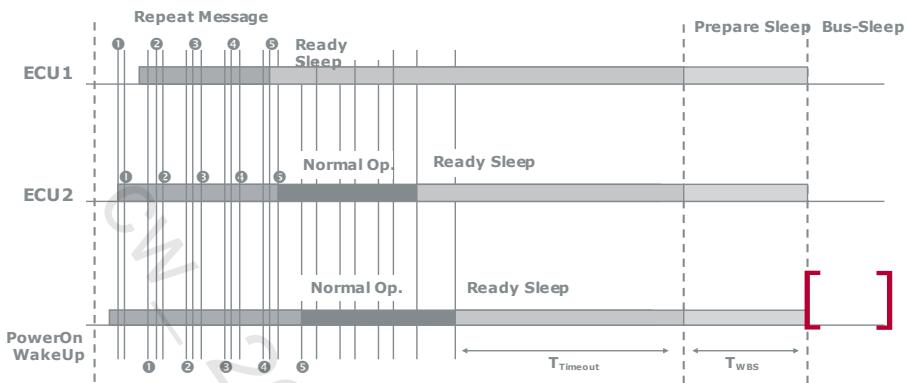


- ▶ Reception and transmission of application messages enabled
- ▶ Repeat Message State
 - ▶ cyclic transmission of NM messages for a certain time
 - ▶ all NM active nodes participate in NM message communication
- ▶ Normal Operation State
 - ▶ transmission of NM messages (if activated reduced transmission)
- ▶ Ready Sleep State
 - ▶ transmission of NM messages is stopped

Prepare Bus-Sleep Mode (CAN)



Bus-Sleep Mode (CAN)



- ▶ Communication bus is shut down
- ▶ Restart (wakeup) of network management
 - ▶ wakeup on communication bus (external)
 - ▶ reception of a NM message (external)
 - ▶ application requires bus communication (internal)

Agenda

- Mode Management
- Wakeup Handling
- BSWM configuration
- ECUM configuration
- Mode Management
- Network Management Algorithm

► **Mode Manager Concept**

- Service Mapping
- Coming up next

What is a Mode Manager?

- ▶ We can distinguish the following roles
 - ▶ Mode Manager, Mode User, Mode Requester
- ▶ A **Mode Manager (MMgr)** is an arbitrary SWC or BSW module which has
 - ▶ A set of one or more modes which the Mode Manager **provides** as published information to other SWCs (Mode Declaration Group(s))
 - ▶ The ability to signalize current mode (state) and mode changes (transitions) over its Port Interface
 - ▶ An implementation of the finite state machine corresponding to the published modes
- ▶ **Mode User (MUsr)**
 - ▶ Arbitrary SWC or the BSWM
 - ▶ May **require** / subscribe the transition or state information
- ▶ **Mode Requester (MRqr)**
 - ▶ Arbitrary SWC or the BSWM with a Mode Request Port

What is a Mode Manager?



The **Mode Requester (MRqr)** is very often identical to the **Mode User (MUsr)**, but not always! A Mode Request Port is usually a Sender-Receiver Interface with no special typing. However, such an interface can appear as a mode declaration group when used by the BSWM.

A **Mode Requester** can use the `BSWM_RequestMode` API to request a mode from the BSWM in case it uses a **Generic Mode**. If it has its own **Mode Request Interface**, the Mode Request Interface will be used instead.

The **Mode Manager** can either use the `Rte_Switch` API to indicate an internal mode change to the RTE or (in case it uses Generic Modes) it can use a **Generic Mode Trigger Function**.

The **Mode User** can be an SWC which consumes Mode Switch Events or Mode Disabling Dependencies, or it queries the Mode over `Rte_Mode` API. But it can also be a Basic Software module which uses mode information using Mode Trigger Functions.

Important terms for mode management to look up in the AUTOSAR specifications

- Mode Manager
- Mode Declaration Group
- Mode Switch Event
- Mode Disabling Dependency
- Mode Request Port

Elements and mechanisms

- ▶ **Mode Declaration Group** (set of supported modes by the MMgr)
 - ▶ Defined in the Port Interface of the MMgr
 - ▶ Port Interface contains enumerations for the modes
 - ▶ Mode Declaration Group has an initial mode value prior to any mode switching
 - ▶ Modes are not hierarchically organized (flat view of all modes)
- ▶ **Mode Switch Event**
 - ▶ Occurs upon transitions between modes (On Mode Exit, On Transition, On Mode Entry)
 - ▶ Can be a trigger for a runnable in an arbitrary SWC
 - ▶ Can not have a respective waitpoint inside a runnable
- ▶ **Mode Disabling Dependency**
 - ▶ Any RTE Event can be equipped with a "Mode Disabling Dependency"
 - ▶ Is valid during the transition into and the mode itself
 - ▶ Prerequisite: consuming Mode User must be connected to MMgr



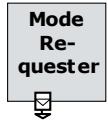
The RTE can signal

Notification of transitions between modes or the current mode which take place inside a Mode Manager

Particularly:

- RTE issues transitions between modes to trigger Runnables (Mode Switch Event(s))
- Prevent a Runnable from being executed during a special mode state (Mode Disabling Dependencies)

Role: Mode Requester



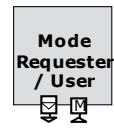
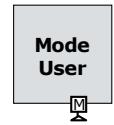
► The Mode Requester

- A SWC can request a mode switch from the MMgr over a **Mode Request P-Port Prototype** (a Sender/Receiver communication)
 - No special annotation in Developer
 - AUTOSAR restriction: isService = TRUE
- The Mode Requester doesn't need to be an SWC:
 - In case the Mode Requester does not have a Mode Request port, it can use the `BswM_RequestMode()` API (Generic Mode)

Role: Mode User

► The Mode User

- The Mode User has a **Mode Switch R-Port Prototype** to be informed about mode changes
- Can invoke `Rte_Mode` API to get knowledge about a Mode Change
- A Runnable can be triggered by the Mode Switch
 - On Exit, On Transition, On Entry
- Frequently, a Mode User is also a **Mode Requester**
 - e.g., with a **Mode Request P-Port Prototype** (a Sender/Receiver communication)



Note: In ASR3.x, the `Rte_Feedback` API could be used for modes by Mode users. In ASR 4.x this is no longer the case. In ASR4.x the Feedback API is dedicated to `DataSendCompletion` only.

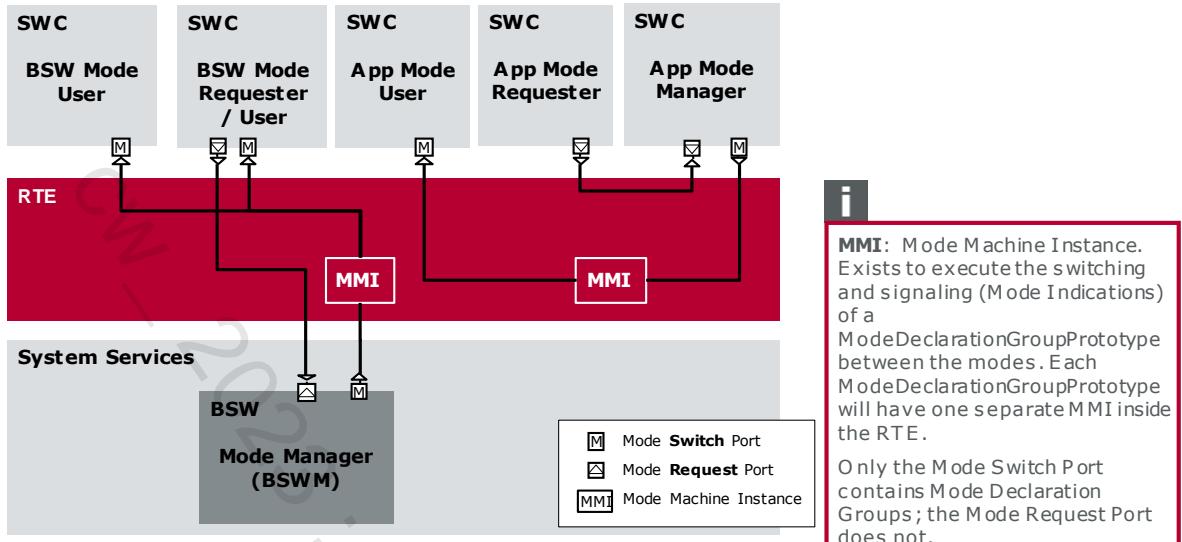
Role: Mode Manager

- ▶ The **Mode Manager** uses
 - ▶ Rte_Switch API to indicate the switch into a requested mode to the RTE
 - ▶ The RTE keeps a Mode Machine Instance (MMI) to reflect the mode system-wide.
 - ▶ Rte_SwitchAck API to get a Mode Switch Acknowledge in a non-blocking or blocking way.
 - ▶ Additionally, Rte_Mode can be called to query the current MMI state in a non-blocking way.
- ▶ The RTE indicates the Mode Switch over a **Mode Switch Port** (containing a Mode Declaration Group) to Mode Users as soon as the RTE initiates the mode transition.



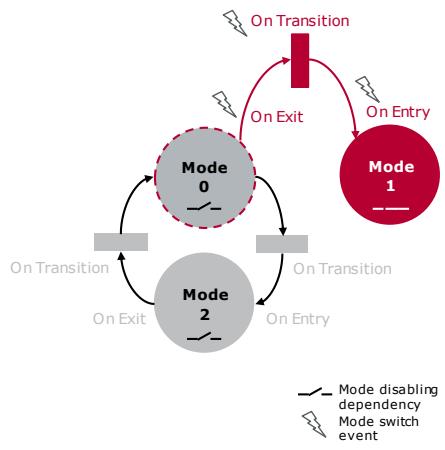
Of course, the Mode manager can also use the Rte_Mode API to get knowledge about the current MMI state.

Mode Managers vs. Mode Users/Requesters



Mode Switching

- ▶ Mode Switch
 - ▶ On Exit
 - ▶ On Transition
 - ▶ On Entry
- ▶ Duration of Mode Disabling
 - ▶ From On Exit of last mode until Mode itself
- ▶ Mode Switching procedure
 - ▶ Synchronous or Asynchronous
 - ▶ Defined in
 - ▶ *ModeSwitchReceiverComSpec*



Why does Exit come before Entry? Cleaning up from last mode is necessary prior to switching to a new one!

Mode Switching



The mode switching procedure can be either

- synchronous
- asynchronous

depending on the ability of the Mode Manager's users. If all Mode users are able to use asynchronous operation, the RTE will implement it as such. If at least one of the mode users only supports synchronous operation, the RTE will implement it in the synchronous way. The ability of supporting the mode switch procedure is defined in the ComSepc of the ModeSwitch Receiver (ModeSwitchReceiverComSpec) mode Parameter: supportsAsynchronousModeSwitch.

If the synchronous operation is chosen, there is a waiting condition which will ensure the respective Executable Entities have terminated before entering the next mode.

If the asynchronous operation is chosen, there is no longer a waiting condition for the termination of executable entities. In this case mode disabling-dependent executable entities may not yet have terminated before On Entry or On Exit related-executable entities become active. This can have effects of preemption and delay.

[SWS_Rte_02664] The RTE generator shall reject configurations of a task with

- OnExit ExecutableEntities mapped after OnEntry ExecutableEntities or
- OnTransition ExecutableEntities mapped after OnEntry ExecutableEntities or
- OnExit ExecutableEntities mapped after OnTransition ExecutableEntities

of the same mode machine instance supporting a synchronous mode switching procedure. (SRS_Rte_00143, SRS_Rte_00213, SRS_Rte_00018)

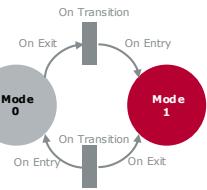
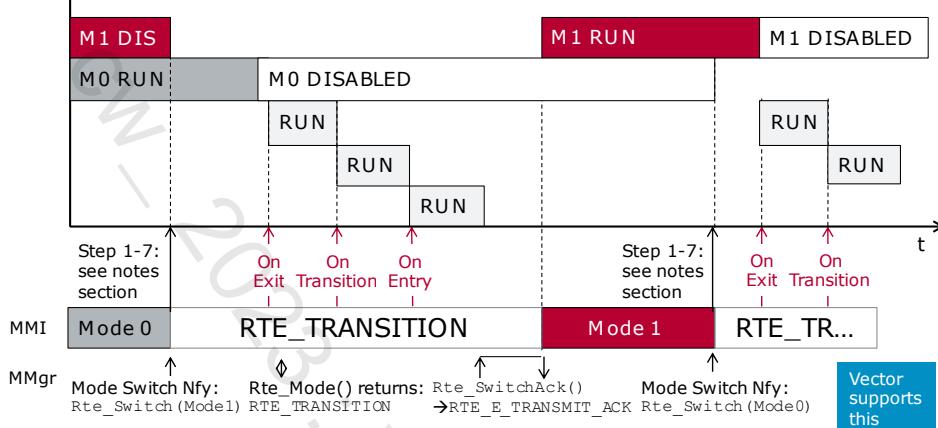
Rationale: This restriction simplifies the implementation of the semantics of asynchronous mode switch.

ModeSwitchReceiverComSpec

- Synchronous mode switching procedure

► *supportsAsynchronousModeSwitch = FALSE*

priority ↑



ModeSwitchReceiverComSpec



The Mode manager utilizes the Rte_Switch API to indicate a Mode Change to the RTE.

The RTE then updates the associated MMI accordingly and puts it into the state RTE_TRANSITION.

This state can be queried by the Rte_Mode API in the MMgr. The MMgr can also wait for the mode switch acknowledge by calling Rte_SwitchAck.

If the Mode Switch has been performed successfully, the RTE signalizes RTE_E_TRANSMIT_ACK.

Upon Reception of a mode switch Notification from a MMgr to the RTE, the RTE performs the following steps:

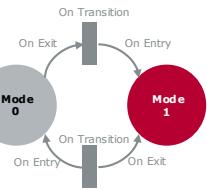
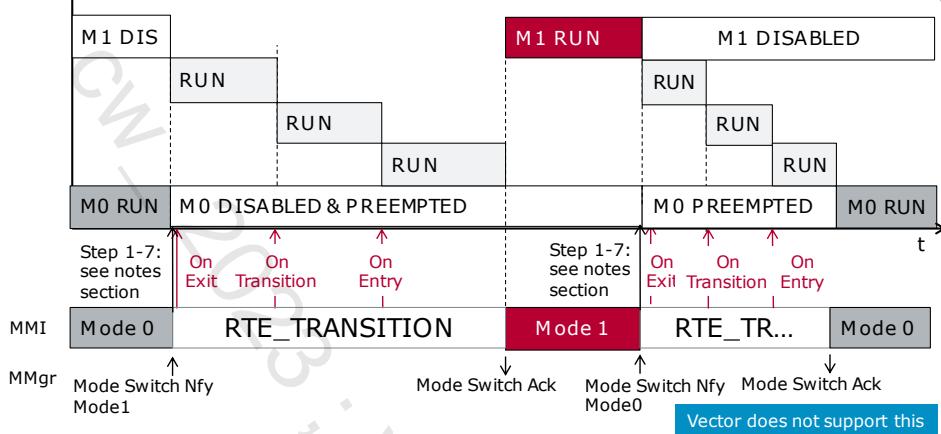
1. Activate mode disablings for the next mode
2. Wait for the newly disabled ExecutableEntities to terminate in case of synchronous mode switching procedure
3. Execute 'clean up ExecutableEntities'
4. Wait for the 'clean up ExecutableEntities' to terminate in case of synchronous mode switching procedure
5. Execute 'initialization ExecutableEntities'
6. Wait for the 'initialization ExecutableEntities' to terminate in case of synchronous mode switching procedure
7. Deactivate mode disablings for the previous modes and enable ExecutableEntities that have been disabled in the previous mode.

ModeSwitchReceiverComSpec

- Asynchronous mode switching procedure

► *supportsAsynchronousModeSwitch = TRUE*

priority ↑



ModeSwitchReceiverComSpec

i

Upon Reception of a mode switch Notification from a MMgr to the RTE, the RTE performs the following steps:

1. Activate mode disabling for the next mode
2. Wait for the newly disabled ExecutableEntities to terminate in case of synchronous mode switching procedure
3. Execute 'clean up ExecutableEntities'
4. Wait for the 'clean up ExecutableEntities' to terminate in case of synchronous mode switching procedure
5. Execute 'Initialization ExecutableEntities'
6. Wait for the 'Initialization ExecutableEntities' to terminate in case of synchronous mode switching procedure
7. Deactivate mode disabling for the previous modes and enable ExecutableEntities that have been disabled in the previous mode.

Agenda

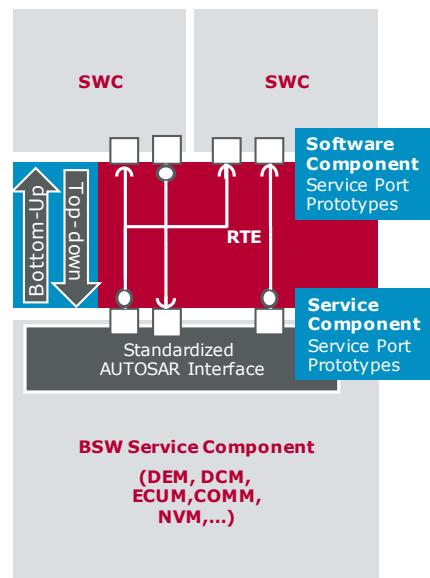
- Mode Management
- Wakeup Handling
- BSWM configuration
- ECUM configuration
- Mode Management
- Network Management Algorithm
- Mode Manager Concept

► **Service Mapping**

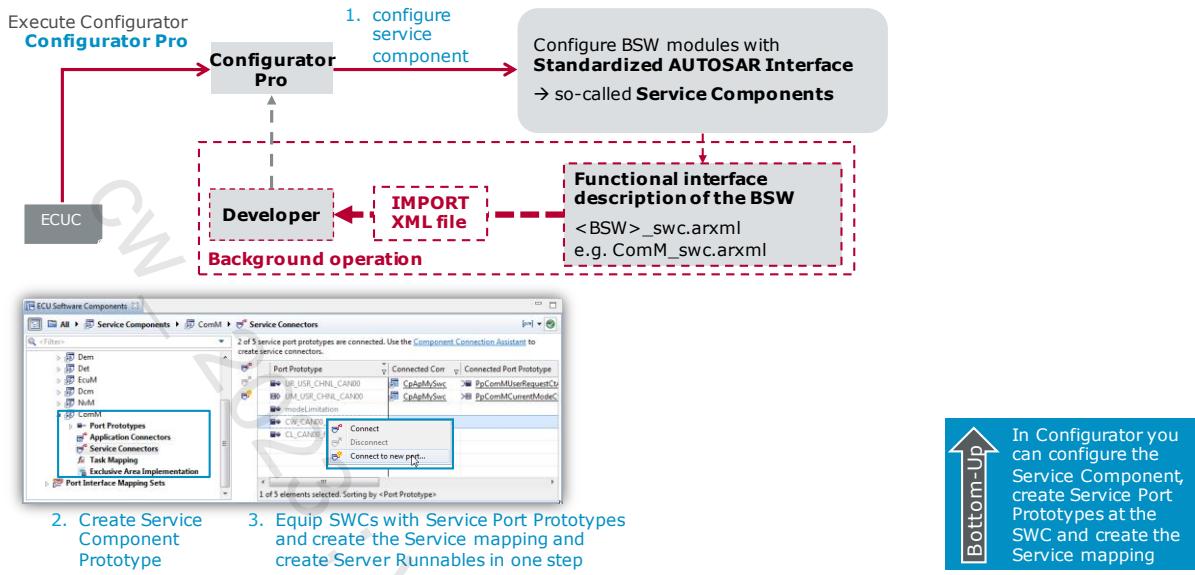
Coming up next

Service Mapping

- ▶ Assigning Service Ports of Software Components to Service Ports of Service Components or vice versa
- ▶ Bottom-up service configuration
 1. Configure a service module in the basic software
 2. Assign the respective service port prototypes to the SWCs where appropriate
- ▶ Top-down service configuration
 1. Use Service Needs
 - > Document at the SWC level what is required from the BSW as a service
 2. Derive a service module configuration based on these requirements
 3. Configure the basic software service module in detail

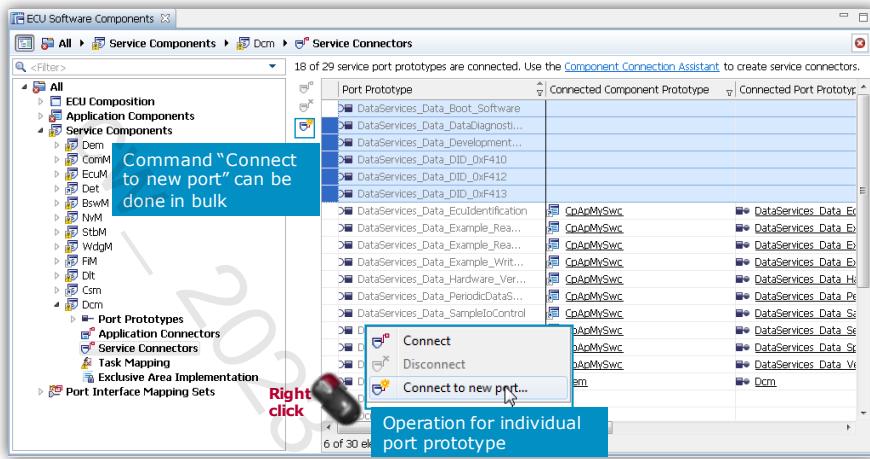


Service Component Configuration (one-stop solution)



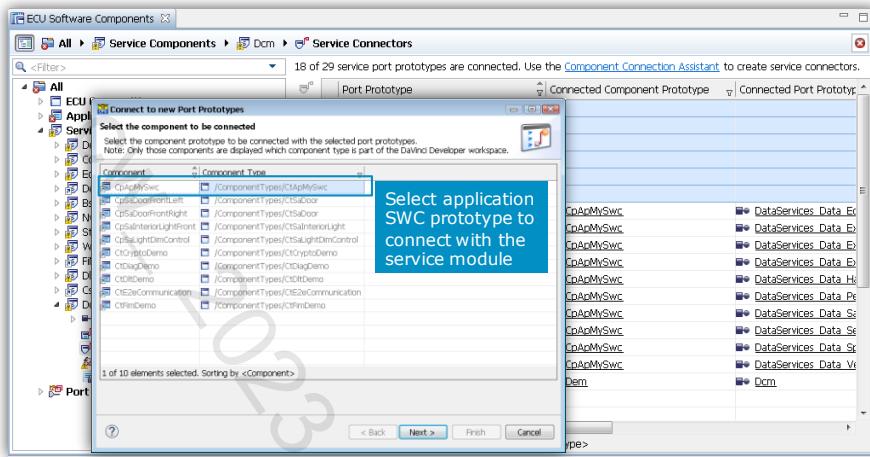
Bottom-up Service Ports (1)

- ▶ Select service port prototype to support



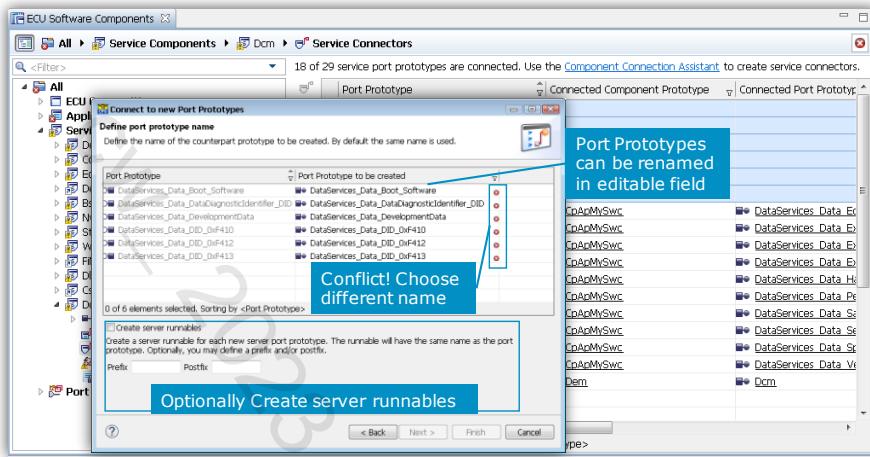
Bottom-up Service Ports (2)

- ▶ Choose SWC which shall implement or call the service

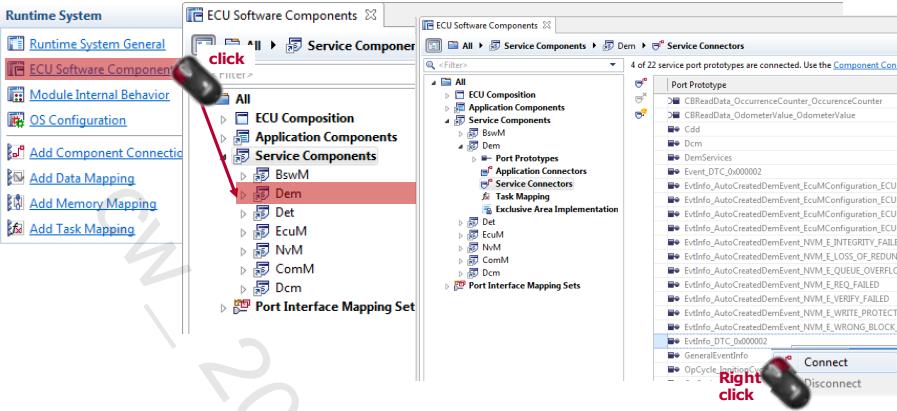


Bottom-up Service Ports (3)

- ▶ Create Port Prototypes and in case of Server Ports, create server runnables as well.

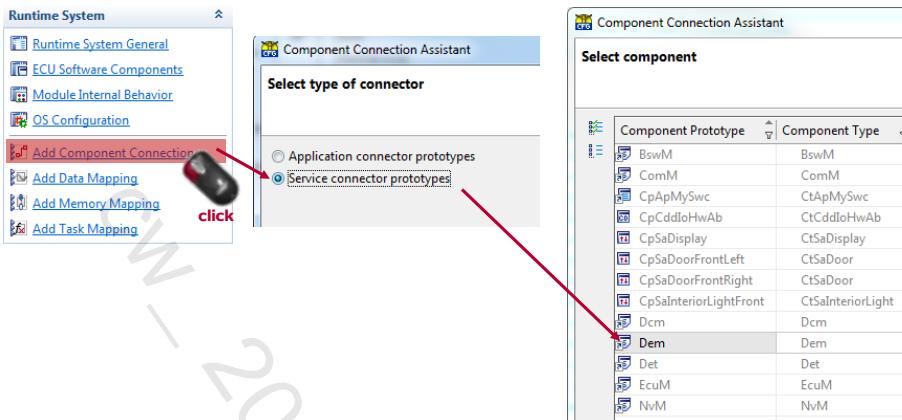


Service Mapping in the ECU Software Components View



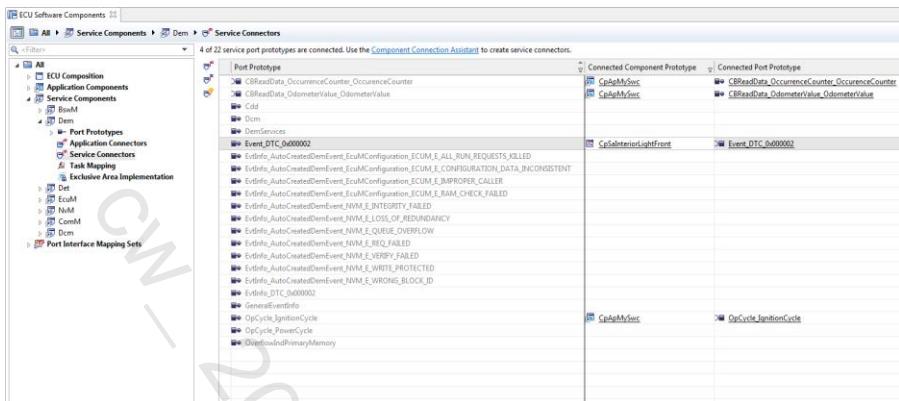
- ▶ Service mapping is visible in the ECU Software Components View. Here you can also perform service mappings.

Service Mapping in the Component Connection Assistant



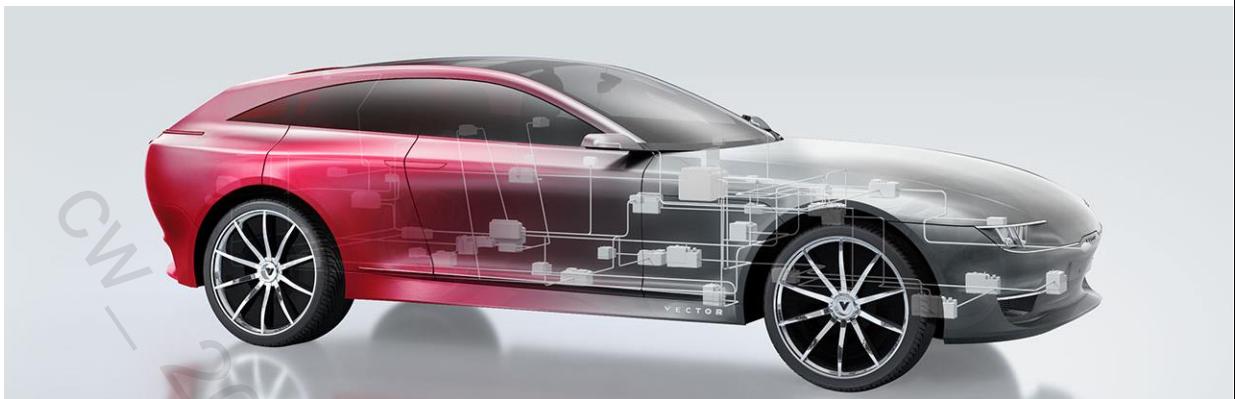
- ▶ You can also do the service mapping in a global view by using the Component Connection Assistant.

Result of Service Mapping



Mode Management

Exercise: Mode Management



AUTOSAR in Practice

Mode Management

V28 | 2023-03-28

Agenda

► **Exercise 4 - Mode Management**

Coming up next

CW_2023_VH_Autosar CP Training_EN

Overview

Mode Management



The ECU has to be initialized properly upon startup. This is implemented in **StartPostOs**. You will use the **BSWM** to do this.

Furthermore, communication is only necessary when information needs to be exchanged between the two ECUs: **MyECU** and **RearECU**.

This is necessary when either front or rear doors are opened. The network will wake up in this case and messages will be sent. When all doors are closed the network falls asleep again.

Implement wakeup and sleep of network for **MyECU**.

Overview

- ▶ An additional SWC has already been added to our design
 - ▶ Used for displaying status information in the CANoe configuration
 - ▶ **CpSaDisplay::CtSaDisplay**
 - ▶ Please do not change this SWC!
 - ▶ Use the Port Interface **PiDisplayState** to use the display in CANoe
 - ▶ Overview in DaVinci Developer:



4

Mode Management

- **1** ECU State Handling, Module Initialization
- **2** Wakeup Source, ComM user, BSWM Communication Control
- **3** Perform Service Mapping, generate the code
- **4** Program integration code, implement Runnables and test functionality
- **5** Add on I: Change a status LED using Mode Switch Events
- **6** Add on II: ECU Shutdown based on Clamp 15 (ignition) with 10 second's delay

Part 1/6: ECU State Handling, Module Initialization

- ▶ "Open with > DaVinci Project Assistant" using **MyECU.dpa** file in folder **\E4_ModeManagement**
 - ▶ Press F9 to generate the code, Click "OK" to close the VTT Dialog
 - ▶ Close the Generation dialog, Press F7 in order to update the SWC templates
- ▶ **Open Visual Studio Project** by double-clicking on **MyECU.sln** file in folder **\E4_ModeManagement\...\Solution**
- ▶ Open CANoe project, switch back to Visual Studio
- ▶ Build the project (F7) and debug it (F5). Switch back to CANoe and press F9.
- ▶ Observe the startup process using breakpoints
 - ▶ `EcuM_Init()` → calls `ECUM_DRIVERINITLIST_ONE()` (`EcuM_AL_DriverInitOne()`) and `EcuM_StartOS()`
 - ▶ `TASK(Init_Task)` → calls `EcuM_StartupTwo()` → calls `SchM_Init()`
- ▶ You eventually end up in the DET, because the ECUM still lacks the BSW initialization.
- ▶ Exit Visual Studio
- ▶ This is what we will configure now using the BSWM "Auto Configuration" Feature
 1. ECU State Handling
 2. Module Initialization

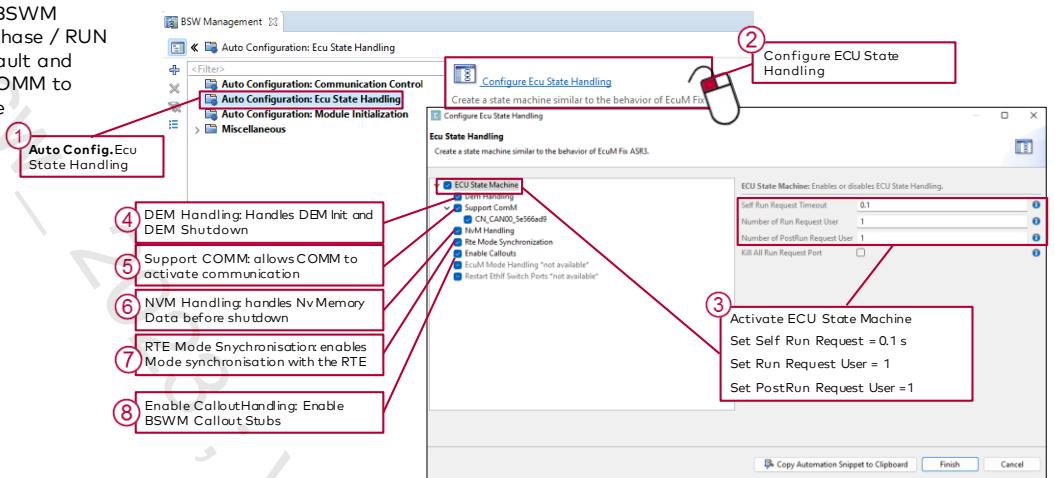
Part 1/6: ECU State Handling

► "Open With > DaVinci Project Assistant" using [MyECU.dpo](#) file in folder `\E4_ModeManagement\`

► Go to **BSW Management** → **Auto Configuration: Ecu State Handling**

► Follow steps 1) – 8) in the figure below and confirm your selection with "Finish"

► Now the BSWM stays in UP Phase / RUN Mode by default and allows the COMM to communicate



Part 1/6: ECU State Handling

- ▶ Synchronize now your configuration [Synchronize now](#), the configurator will complain about the missing Multiblock Job Status Information and offer solving actions in its Validation window.

ID	Message
► BASE01134	Tresos generator message (4 messages)
► BSWM01039	The BswMNvMJobModeIndication mode request sources are not in synch with NvMBswMMultiBlockJobStatusInformation. (1 message) There should be no BswMNvMJobModeIndications.
► BSWM01039	Remove all BswMNvMJobModeIndication mode request sources
► Set NvMBswMMultiBlockJobStatusInformation(value=false) to true	Set NvMBswMMultiBlockJobStatusInformation(value=false) to true
► CAN02019	An unexpected baud rate value is configured (1 message)

- ▶ Choose "Set NvMBswMMultiBlockJobStatusInformation(value=false) to true"

Part 1/6: ECU State Handling



► What did we create by activating Ecu State Handling?

- **BSW Management → Auto Configuration: Ecu State Handling (ESH):** If we inspect the BSWM configuration, several rules and corresponding actions have been created. Selection: shows the actions which enable and disable the Communication (call ComM_CommunicationAllowed API)

The screenshot shows the BSW Management interface with the path: BSW Management > Auto Configuration: Ecu State Handling > Actions > ESH_Action_EmcCheckPendingRequests, ESH_Action_ComMClockPendingRequests, ESH_Action_CmfdDialog. A context menu is open over the 'Actions' node, with the option 'Modify properties of selected item' highlighted.

Name	Description
ESH_Action_CancelNvMWriteAll	Call NvM_CancelNvMWriteAll()
ESH_Action_CancelWriteAllTimer_Start	ESH_Action_CancelWriteAllTimer_Start
ESH_Action_CancelWriteAllTimer_Stop	ESH_Action_CancelWriteAllTimer_Stop
ESH_Action_ComMClockPendingRequest	Call ComM_ClockPendingRequest()
ESH_Action_CmfdAllow_CN_CAN00_5e566ad9	Enable ComMChannel CN_CAN00_5e566ad9
ESH_Action_CmfdCheckPendingRequests	Call ESH_Cmfd_CheckPendingRequests()
ESH_Action_CmfdDialogAllow_CN_CAN00_5e566ad9	Disable ComMChannel CN_CAN00_5e566ad9
ESH_Action_DemInit	Call Dem_Init(Dem_Config_Pt)
ESH_Action_DemInitialised	ESH_Action_DemInitialised

- **ESH Actions:** The Action allows or denies the operation for ComM channels

The screenshot shows the BSW Management interface with the path: BSW Management > Auto Configuration: Ecu State Handling > Actions > ESH_Action_CmfdAllow_CN_CAN00_5e566ad9. The action properties are displayed:

- Name:** ESH_Action_CmfdAllow_CN_CAN00_5e566ad9
- Com Allowed:**
- ComM Allow Channel Ref:** [/ActiveEcuC/ComM/ComMConfigSet/CN_CAN00_5e566ad9]
- Triggering Rules:**
- Rules triggering this action:** ESH_WakeupToRun, ESH_PortRunNeeded

Part 1/6: ECU State Handling



► What did we create by activating Ecu State Handling?

- **Basic Editor → BSWM:** In the Generic Configuration Editor, the activities of allowing the communication and the start of the RTE, as well as the self run request timer, appear in the BSWM.

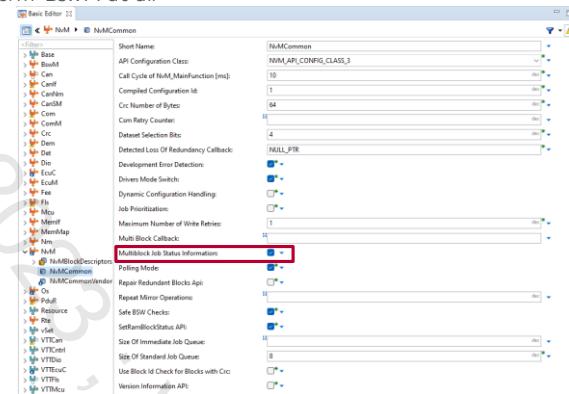
Action List Item Index	Action List Item Ref
0	ESH_Action_ComMDisallow_CN_CAN00_5e566ad9
*	ESH_Action_EcuMClearValidatedWakeupEvents
4	ESH_Action_ESH_PostRun
2	ESH_Action_OnEnterPostRun
3	ESH_Action_SwitchPostRun

Part 1/6: ECU State Handling

i

► Where to find this parameter?

- **NvM → NvMCommon:** This parameter defines whether BswM is informed about the current status of the multi block job.
 - > True: call BswM_NvM_CurrentJobMode if ReadAll and WriteAll are started, finished, canceled
 - > False: do not inform BswM at all



Part 1/6: Module Initialization

- Go to **BSW Management** → **Auto Configuration: Module Initialization**

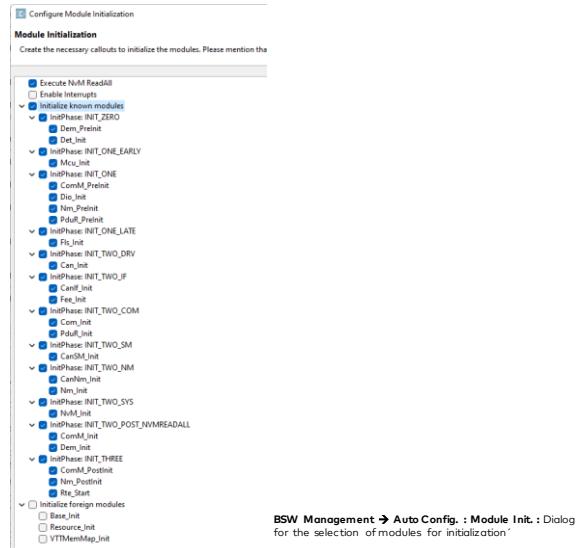
The screenshot shows the BSW Management interface with the following details:

- BSW Management** tab is selected.
- Auto Configuration: Module Initialization** is the active sub-tab.
- Configure Module Initialization...** button is highlighted with a red box and a cursor icon.
- Help** link: Provide informations for foreign modules click [here](#).
- Create the necessary callouts to initialize the modules.** Please mention that MCAL modules must be initialized.
- Configuration Options:**
 - Execute NvM ReadAll
 - Enable Interrupts
 - Initialize known modules
 - InitPhase: INIT_ZERO
 - Dem_Prelinit
 - Det_Init
 - InitPhase: INIT_ONE_EARLY
 - Mcu_Init
 - InitPhase: INIT_ONE
 - ComM_Prelinit
 - Dio_Init
 - Nm_Prelinit
 - PduR_Prelinit
 - InitPhase: INIT_ONE_LATE

BSW Management → Auto Configuration: Module Initialization: Entry Point "Configure Module Initialization"

Part 1/6: Module Initialization

- ▶ Select all existing modules for initialization, except for foreign modules (see checkboxes in figure right)
- ▶ Click "Finish", Validate, and Generate.
- ▶ File \source\BSWM_Callout_Stub.c becomes updated. Purpose: Read NVRAM data
- ▶ Other BSW modules and RTE are initialized in the BSWM file
\gendata\BswM_Lcfg.c



Part 1/6: Module Initialization

i

- ▶ What will be generated by setting Module Initialization?

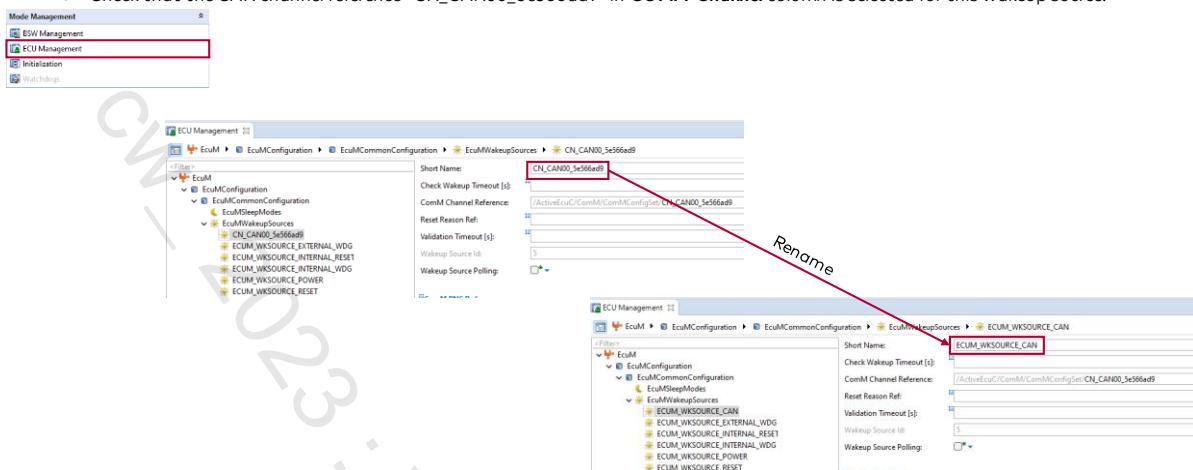
- ▶ **BswM_Lcfg.c:** Basic software module initialization
(invocation of <BSW>_Init() APIs or BSWM callouts)

- ▶ **BswM_Callout_Stubs.c**: Implementation of the restoring of NVRAM data as BSWM callouts

Part 2/6: Wakeup Source

Go to **Mode Management**, select **ECU Management** and go to **EcuMWakeupsources**

- > If it's not already there, insert a new Wakeup Source or rename existing source "CN_CAN00_5e566ad9" to "ECUM_WKSOURCE_CAN."
- > Check that the CAN channel reference "CN_CAN00_5e566ad9" in **Comm Channel** column is selected for this wakeup source.



Part 2/6 ComM user

► Open Network Management → Communication Users

- View the default COMM User, it has already been assigned an appropriate channel (don't enable the 'Mode Notification' yet)

- Rename the COMM User "CN_CAN00_5e566ad9" to "USR_INTERIOR_LIGHT".

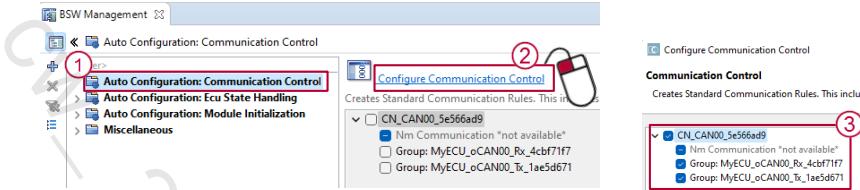
► The Reference to the CAN Channel is already here, so we do not have anything else to do.

- Execute "Synchronize now" [Synchronize now](#)

- **Mode Notification:** This enables the Mode Declaration Group of the COMM for this User. We will use this in the first Add-On.

Part 3/6: BSWM Communication Control

- Now the **BSWM** has to be configured to switch the communication (IPDU Groups) on and off, depending on the **BusSM** State.
- Up until now the BSWM does not have any rules to switch the IPDU Groups.
- Open **BSW Management** → **Auto Configuration: Communication Control**. Click on Configure Communication Control.

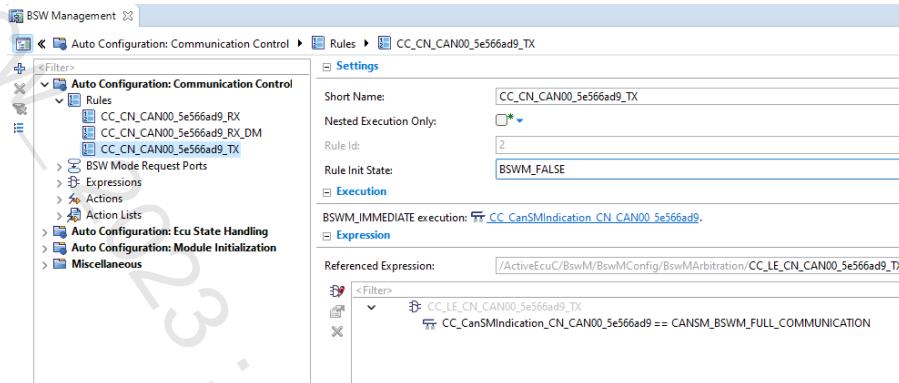


- Activate both checkboxes for the IPDU Group Tx and Rx (see figure above)
- Click "Finish." Now the BSWM has rules to switch PDU Groups according to the CANSM Mode (see next slide)

Part 3/6: BSWM Communication Control

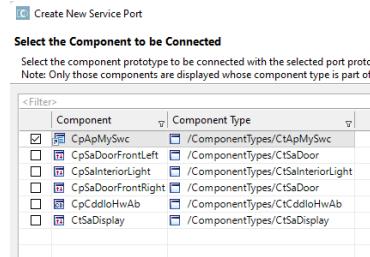
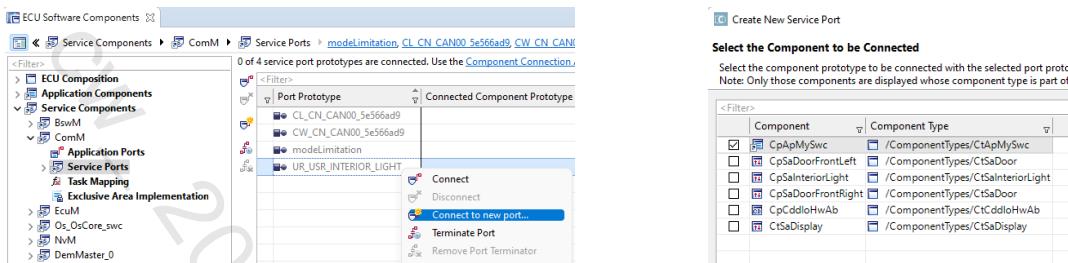


- **Auto Configuration: Communication Control:** After having executed the steps of the Assistant, a set of rules for switching the deadline Monitoring and the IPDU Groups has been created. In AUTOSAR 4, the BSWM has to set the state of Deadline Monitoring and IPDU Groups depending on the mode of the CANSM. The main idea behind this is that the BSWM will do this for partial networking scenarios for a PNC state. Even if we do not use Partial Networking, it is necessary that the BSWM has a configuration to handle the communication.



Part 4/6: Service Mapping (bottom-up)

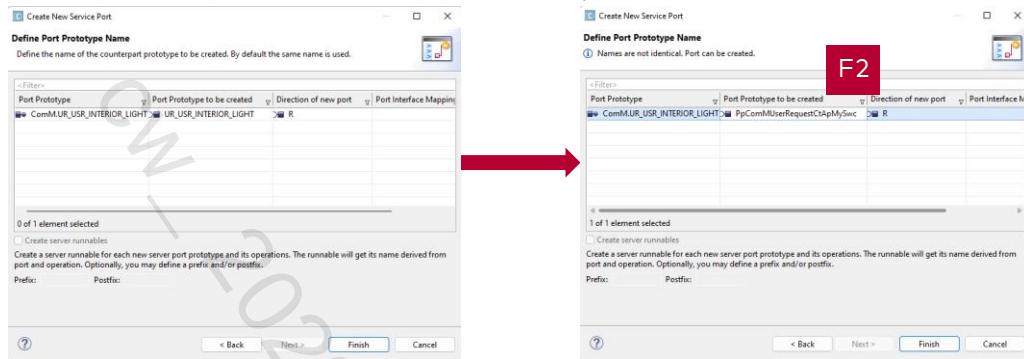
- The COMM is now configured. We will propagate a resulting Service Port Prototype to the SWC Type **CtApMySwc** in order to be able to use the service inside the SWC.
- Stay in Configurator and go to **Runtime System → Ecu Software Components**
 - Open the subtree **Service Components → ComM → Service Ports**
 - Click on the Port Prototype **UR_USR_INTERIOR_LIGHT**, right click and choose "Connect to new port..."



- This will bring up a dialog where you can select **CpApMySwc**, the SWC where we want to control the COMM channel.

Part 4/6: Service Mapping (bottom-up)

- ▶ Configurator proposes to create a new Port Prototype called **UR_USR_INTERIOR_LIGHT**.
- ▶ Change the name of the Port Prototype to **PpComMUserRequestCtApMySwc** and click on **Finish**
- ▶ Click in the field of **UR_USR_INTERIOR_LIGHT** and press F2 to rename



- ▶ As a result, a service mapping becomes established between **CtApMySwc** and **COMM service component** (check in Service Connectors view)

Part 4/6: Service Mapping (bottom-up)

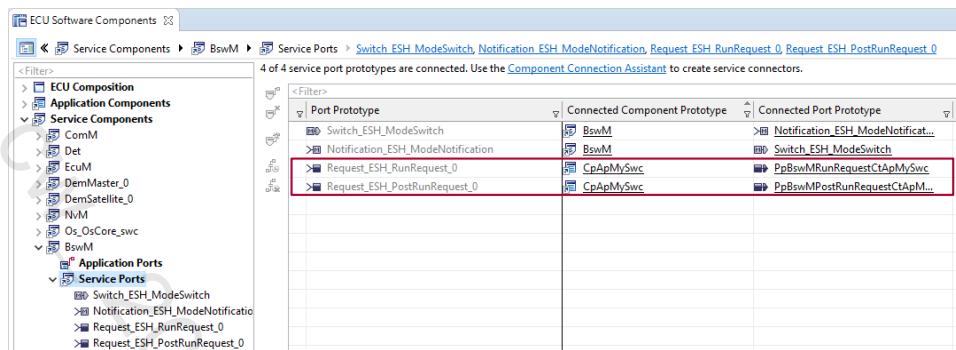
- ▶ The BSWM provides now a Run Request and a Post Run Request. We will propagate a resulting Service Port Prototype to the SWC Type **CtApMySwc** for each Request to be able to request the services inside the SWC.

- ▶ Stay in **Runtime System** → **Ecu Software Components**

- ▶ Open the subtree **Service Components** → **BswM** → **Service Ports**
- ▶ Click on the Port Prototype **Request_ESH_RunRequest_0**, right click and choose "Connect to new port..."
- ▶ Change this name of the Port Prototype to **PpBswMRunRequestCtApMySwc** and click on **Finish**
- ▶ Create also a new port for the Port Prototype **Request_ESH_PostRunRequest_0**, right click and choose "Connect to new port..." **PostRunRequest**
- ▶ Change the name of the Port Prototype to **PpBswMPostRunRequestCtApMySwc** and click on **Finish**

Part 4/6: Service Mapping (bottom-up)

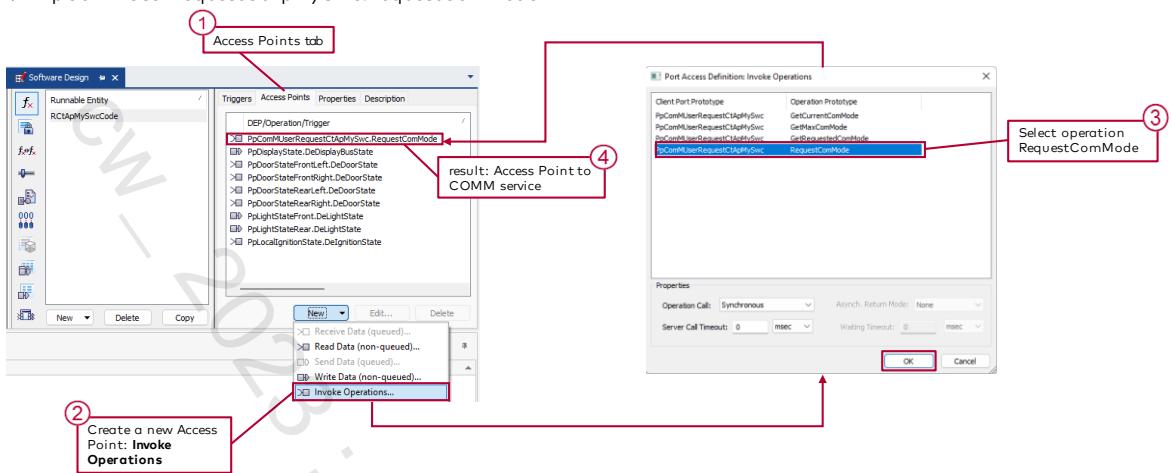
- As a result, a service mapping becomes established between **CtApMySwc** and **BSWM service components** (check in Service Ports view)



- Save your Project (you can leave Configurator open)

Part 4/6: Service Mapping (bottom-up)

- ▶ "Open with > DaVinci Developer Classic" using **MyECU.dpa** file in folder **\E4_ModeManagement**
- ▶ Add an "Invoke Operations..." **Access Points** for the Runnable **RCtApMySwcCode**
- ▶ **PpComUserRequestCtApMySwc.RequestComMode**



Part 4/6: Service Mapping (bottom-up)

i

Service Component Description Files _swc.arxml

On Saving the Project, Configurator generates the necessary Service Component files (*_swc.arxml) that have to be imported into the Developer.

Object Browser:

Have a look at the object browser inside of Developer:

- ▶ ComM is now available in the Service Component Types
- ▶ ComMMode is now available in the Mode Declaration Groups
- ▶ Different ComM Port Interfaces are available in the Service Port Interfaces

Service Ports

- ▶ ComM_UserRequest -> PpComMUserRequestCtApMySwc
- ▶ C/S Port Interface for requesting a communication mode at the COMM.
- ▶ CtApMySwc is a Client

Part 4/6: Service Mapping (bottom-up)

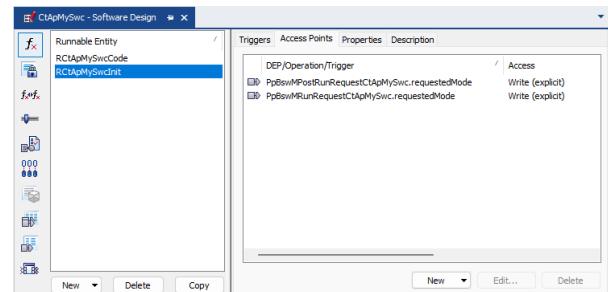
- Add an Init Runnable to **CtApMySwcCode**, name it **RCtApMySwcInit**



- Add two "Write Data (non-queued)..." **Access Point** for the Runnable **RCtApMySwcInit**

- PpBswMRunReQuestCtApMySwc.requestedMode**
- PpBswMPostRunReQuestCtApMySwc.requestedMode**

- Save and exit **Developer**



Part 4/6: Service Mapping, Code Generation

- ▶ If Configurator was closed, reopen it using “**Open with > DaVinci Project Assistant**” from the **\E4_ModeManagement\MyECU.dpa** context menu
- ▶ ⚠ Synchronize the System Description using the proffered solving action. (if Configurator was left open, you can choose “synchronize” from the pop-up dialog.)
 - ▶ This will bring in the updated Component Type **CtApMySwc** including the access point for the **COMM** and **BSWM**
- ▶ Perform the **Task Mapping**. Simply double click the error message, it jumps directly to the Task mapping, or go over the menu **ECU Software Components → Application Components → CpApMySwc**
- ▶ Map the new **Runnable** in **CpApMySwc** to **My_Task**
 - ▶ **RCtApMySwcInit**: Position = 0
 - ▶ **RCtApMySwcCode**: Position = 1
- ▶ Generate Code  (F9), click on finish in the VTT dialog, then close the Generation dialog.
- ▶ Generate SWC Templates  (F7)

<input type="checkbox"/> RTE01056	Unmapped runnable entity (1 message)
<input checked="" type="checkbox"/> RTE01056	The runnable entity <RCtApMySwcInit> is not mapped to a valid task.
	 /ActiveEcuC/Rte/CpApMySwc_EcuSwComposition/CtApMySwc_InitEvent
	RteMappedToTaskRef

Part 4/6: Integration Code, Implementation, Test

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder **\E4_ModeManagement\...\Solution**
- ▶ Program integration code in the ECUM firmware file **EcuM_Callout_Stubs.c**
 - ▶ Modify the function **EcuM_CheckWakeup**
 - > Check whether the parameter **wakeupSource** is the one for CAN
 - > If so, pass the parameter **wakeupSource** to CANIF via the **CanIf_CheckWakeup** API
 - > Example(you can copy and paste it to the **EcuM_CheckWakeup** function):

```
/* Add implementation of EcuM_CheckWakeup() */
```

```
if ((wakeupSource & ECUM_WKSOURCE_CAN) != 0)
```

```
{
```

```
    CanIf_CheckWakeup(ECUM_WKSOURCE_CAN);
```

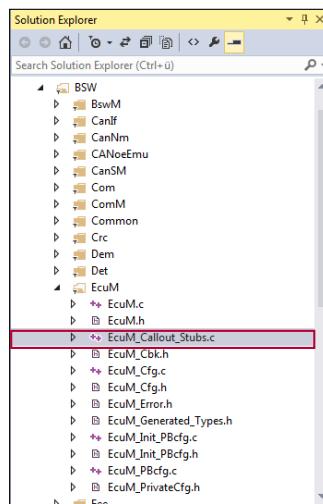
```
}
```

```
return;
```

- ▶ Implement the following behavior in Runnable **RCtApMySwcCode**
 - ▶ Use the ComM RequestComMode operation accessible over RTE access function **Rte_Call_PpComMUserRequestCtApMySwc_RequestComMode**
 - > Request full communication as soon as at least one door is open
 - > Request no communication as soon as all doors are closed
 - > Hint: You find the requests defined in the Software Component Template

Part 4/6: Integration Code, Implementation, Test**i**

- ▶ Where can I find EcuM_Callout_Stub.c in the Visual Studio project?



Part 4/6: Integration Code, Implementation, Test

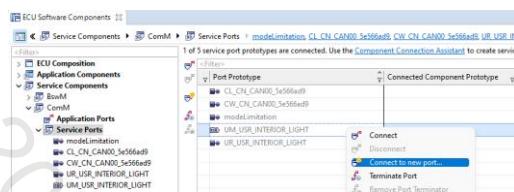
- ▶ Implement the following behavior in Runnable **RCtApMySwcInit**
 - ▶ Use the BswM requestedMode operations accessible over RTE access functions
Rte_Write_PpBswMRunRequestCtApMySwc_requestedMode and
Rte_Write_PpBswMPostRunRequestCtApMySwc_requestedMode
 - > Request a Run Request and a Post Run Request
 - > Hint: You find the requests defined in the Software Component Template

▶ Compile and test

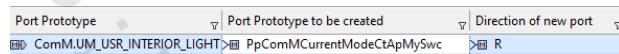
- ▶ Wait until bus is asleep, then you can
 - ▶ Open a front door. This will cause the ECU to do a local wakeup on the CAN bus.
 - ▶ Open a rear door. This will cause a remote Wakeup in the ECU which will be detected by ECUM prior to informing the COMM to switch on communication.
- ▶ After closing all doors, CAN bus should fall asleep again.

Part 5/6: Add-On I: Update Status LED using Mode Switch Events

- ▶ Now we want to control the LED in CANoe indicating the current communication mode (i.e., FULL_COMM, NO_COMM). This shall happen over a Mode Switch Port.
- ▶ If Configurator was closed, reopen it using “**Open with > DaVinci Project Assistant**” from the **\E4_ModeManagement\MyECU.dpa** context menu
- ▶ Go to **Network Management** → **Communication Users** → **USR_INTERIOR_LIGHT**
- ▶ Enable the ‘**Mode Notification**’ checkbox in **Communication Users** (see next slide) and [Synchronize now](#).
- ▶ Go to **Runtime System** → **ECU Software Components** → **Service Components** → **ComM** → **Service Ports**



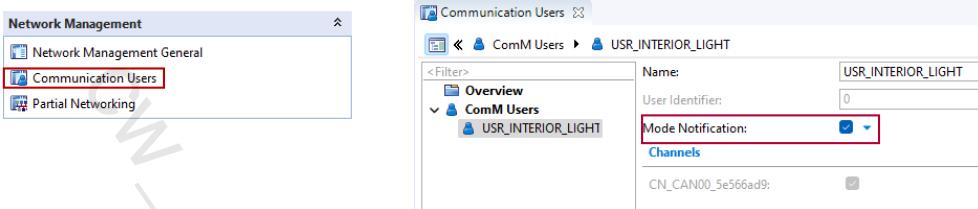
- ▶ Create Service R-Port based on the **CurrentMode** Port and do service mapping for **CtApMySwC** using bottom-up service configuration
- ▶ Rename the **UM_USR_INTERIOR_LIGHT** to **PpComMCurrentModeCtApMySwc**



Part 5/6: Add-On I: Update Status LED using Mode Switch Events

► Activate ComM Mode Notification

The COMM uses the Mode Notification to report its Mode over a Mode Switch Port. This Port can be used by the RTE to synchronize the application to the COMM Mode Transitions.



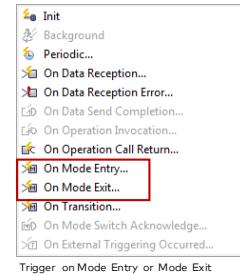
ComM User Configuration: Now Enable Mode Notification for Channel USR_INTERIOR_LIGHT

► PpDisplayState

The Port Prototype PpDisplayState is used to display if the current state of the bus is FULL_COMMUNICATION or NO_COMMUNICATION.

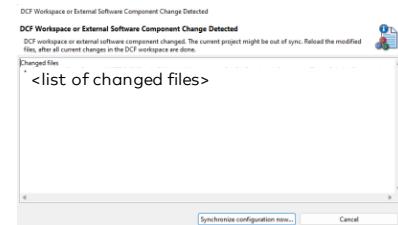
Part 5/6: Add-On I: Update Status LED using Mode Switch Events

- ▶ Open "Open with > DaVinci Developer Classic" using **MyECU.dpa** file in folder **\E4_ModeManagement**
- ▶ Open SWC Type **CtApMySwc**
- ▶ Define Runnables and necessary Triggers and Access Points:
 - ▶ **RCtApMySwcComM_ModeChange_FULL_COMM_Entry**
 - > Trigger "On Mode Entry" entering COMM_FULL_COMMUNICATION
 - > Add Access Point (write) to PpDisplayState.DeDisplayBusState
 - ▶ **RCtApMySwcComM_ModeChange_FULL_COMM_Exit**
 - > Trigger "On Mode Exit" exiting COMM_FULL_COMMUNICATION
 - > Add Access Point (write) to PpDisplayState.DeDisplayBusState
- ▶ Check 
- ▶ Save in Developer 



Part 5/6: Add-On I: Mode Switch Events

- ▶ If not already open, open "**Configure BSW**" using **MyECU.dpa** file in folder **\E4_ModeManagement**
- ▶ Execute the Auto-Solve action "Synchronize the System Description" or choose "Synchronize configuration now"



- ▶ Perform the **Task Mapping**. Simply double click the error message, it jumps directly to the Task mapping, or go over the menu **ECU Software Components** → **Application Components** → **CpApMySwc**

Validation		Element Usage	Unresolved References
57 messages in 17 categories			
ID		Message	
RTE01056		Unmapped runnable entity (2 messages)	
RTE01056		The runnable entity </RCApMySwc/ComM_ModeChange_FULL_COMM_Entry> is not mapped to a valid task. RunMapperToTaskRef	
RTE01056		/ActivEcoC/Rte/CpApMySwc_EcuSwComposition/MST_RCApMySwc/ComM_ModeChange_FULL_COMM_Entry_0	
		The runnable entity </RCApMySwc/ComM_ModeChange_FULL_COMM_Entry> is not mapped to a valid task. RunMapperToTaskRef	
		/ActivEcoC/Rte/CpApMySwc_EcuSwComposition/MST_RCApMySwc/ComM_ModeChange_FULL_COMM_Entry_0	
Tresis79007		Combined non module specific tresis warnings (1 message)	

- ▶ Map the new **Runnables** in **CpApMySwc** to **My_Task** (see next slide)
- ▶ Hint: You have to leave a mode before you can enter a new one.
- ▶ Generate Code , Generate SWC Templates

Part 5/6: Add-On I: Mode Switch Events



► **Task Mapping:** According to AUTOSAR requirements, OnExit runnables have to be mapped before OnEntry runnables.

► Attribute "Position in Task" does not necessarily require unique settings. If the Triggered function is the same for all trigger events, the same position can be selected for all Trigger events. Our OnEntry/OnExit Runnables require separate positions since they execute different Triggered functions (Runnables).

The screenshot shows the 'Task Mapping' section of the ECU Software Components interface. The left sidebar displays a tree structure of ECU Composition, Application Components (selected), CpApMySwc, Application Ports, Service Ports, Task Mapping (selected), Exclusive Area Implementation, and Memory Mapping. The main area shows a table titled 'Triggered Function' with four rows. The table has columns for Triggered Function, Task, and Position. The data is as follows:

Triggered Function	Task	Position
RctApMySwcInit	My_Task	0
RctApMySwcCode	My_Task	1
RctApMySwcComM_ModeChange_FULL_COMM_Exit	My_Task	2
RctApMySwcComM_ModeChange_FULL_COMM_Entry	My_Task	3

Part 5/6: Add-On I: Mode Switch Events

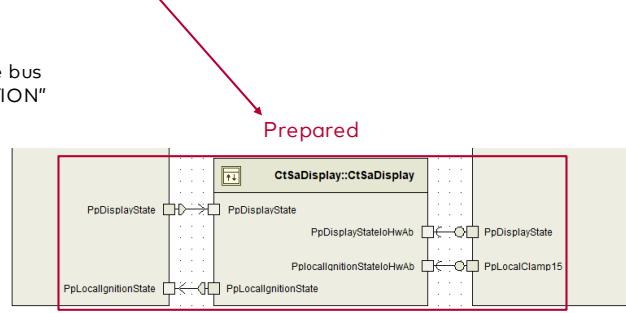
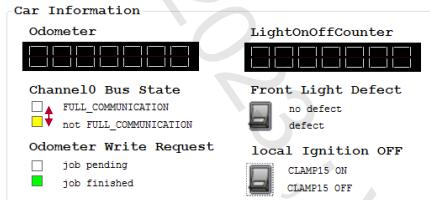


- **Service mapping:** The current Mode Port has to be connected. Otherwise, you won't get the OnEntry or OnExit events.

Port Prototype	Connected Component Prototype	Connected Port Prototype
PpBswMPostRunRequestCtApMySwc	BswM	> Request_ESH_PostRunRequest_0
PpBswMRunRequestCtApMySwc	BswM	> Request_ESH_RunRequest_0
PpComMCurrentModeCtApMySwc	ComM	UM_USR_INTERIOR_LIGHT
PpComMUserRequestCtApMySwc	ComM	UR_USR_INTERIOR_LIGHT

Part 5/6: Add-On I: Mode Switch Events

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder **\E4_ModeManagement\...\Solution**
- ▶ Program new Runnables to have the following behavior
 - ▶ **RCtApMySwcComM_ModeChange_FULL_COMM_Entry**
 - ▶ set DeDisplayBusState to CDisplayOn.
 - ▶ **RCtApMySwcComM_ModeChange_FULL_COMM_Exit**
 - ▶ set DeDisplayBusState to CDisplayOff.
- ▶ Hint: The implementation of the Runnable **RCtApDisplayBusStateHandling** in the SWC **CtSaDisplay.c** and the Server Runnable in the SWC **CtCddIoHwAb.c** is prepared from our side. Everything we have prepared here for you should be known.
- ▶ Compile and test
- ▶ The field "Car Information" now displays the state of the bus "FULL_COMMUNICATION" or "not FULL_COMMUNICATION"



Part 5/6: Add-On I: Mode Switch Events



- Invocation of the State Change Notification to the RTE (`Rte_Switch`) is decoupled over a constant function pointer table `ComM_StateChangeNotificationFct[]`.

```
if( userIndex < ComM_GetSizeOfUserModeNotifFunc() ) /* COV_COV1_GENDATA_CHECK */ /* PRQA S 3395, 3358 */ /* NO_ComM_3395_3358 */
{
    /* #40 If a new mode is available for the current user notify it using the Mode Switch Interface (sender-receiver) 'ComM_CurrentMode' */
    switch(ComM_CalcMode)
    {
        case COMM_FULL_COMMUNICATION:
            retValue = ComM.GetUserModeNotifFunc(userIndex)(RTE_MODE_CommMode_COMM_FULL_COMMUNICATION); /* SBSW_COMM_UserModeNotifFunc_PointerCall */
        case COMM_SILENT_COMMUNICATION:
            retValue = ComM.GetUserModeNotifFunc(userIndex)(RTE_MODE_CommMode_COMM_SILENT_COMMUNICATION); /* SBSW_COMM_UserModeNotifFunc_PointerCall */
        case COMM_NO_COMMUNICATION:
            retValue = ComM.GetUserModeNotifFunc(userIndex)(RTE_MODE_CommMode_COMM_NO_COMMUNICATION); /* SBSW_COMM_UserModeNotifFunc_PointerCall */
        break;
        /* PRQA S 2018 3 */ /* NO_ComM_2018 */
        default: /* COV_COMM_MISRA */
        break;
    }
}
```

ComM.c: ComM Mode calculation and notification

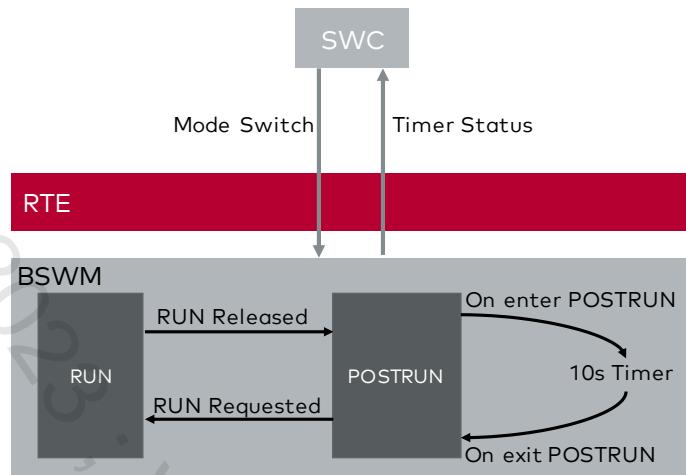
- Debug hint:** Set a Breakpoint into `ComM_StateChangeNotification` to see how the state will propagate through the RTE and become transferred into an `RTE_Switch` which results in a Mode Switch Event caught by the `_Entry` and `_Exit` runnables.

Call Stack	
Name	
MyECU.dll!Rte_Switch_ComM_UM_USR_INTERIOR_LIGHT_currentMode(unsigned char nextMode)	Line 499
MyECU.dll!ComM_StateChangeNotification(unsigned char channel)	Line 3318
MyECU.dll!ComM_MainFunction(unsigned char Channel)	Line 5293
MyECU.dll!ComM_MainFunction_0()	Line 214
MyECU.dll!Os_Task_SchM_Task()	Line 727

Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

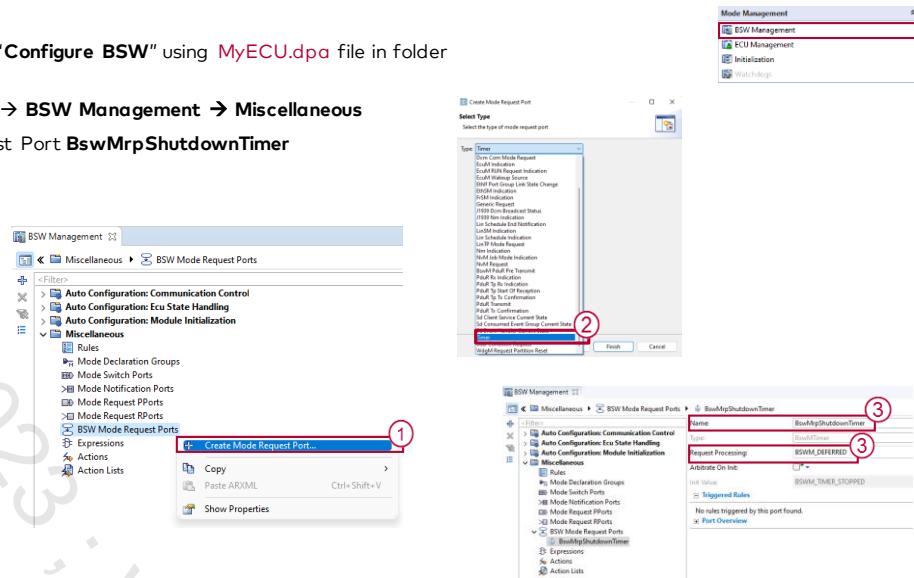
- Now we want to add functionality to power down the ECU in case the car is turned off (clamp 15 signal). In addition, the interior light shall stay awake for 10 seconds before power down of the ECU. We can accomplish this task by employing the BSWM.



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ If not already open, open “**Configure BSW**” using **MyECU.dpa** file in folder **\E4_ModeManagement**
- ▶ Go to **Mode Management → BSW Management → Miscellaneous**
- ▶ Create BSW Mode Request Port **BswMrpShutdownTimer**
 - ▶ Type:
 - > Timer
 - ▶ Request Processing:
 - > **BSWM_DEFERRED**

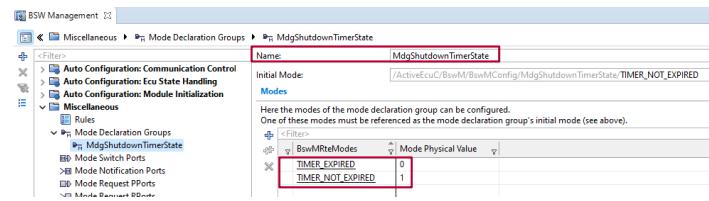


Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

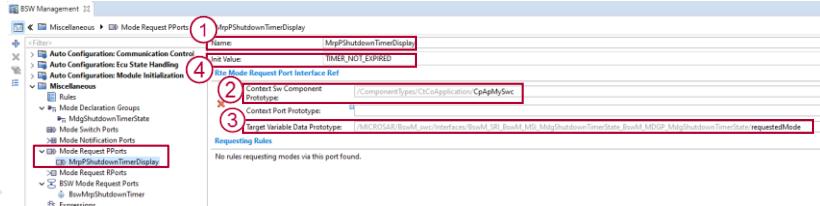
- ▶ Create a Mode Declaration Group **MdgShutdownTimerState** with two modes
 1. **TIMER_EXPIRED** (Value:0)
 2. **TIMER_NOT_EXPIRED** (Value:1)
 - > Make this one the **Initial Mode** of the ModeDeclGroup

▶ Synchronize now [Synchronize now](#)



- ▶ Create Mode Request PPort **MrpPShutdownTimerDisplay**
 - > Context Sw Component Prototype: **CpApMySwc**
 - > Target Variable Data Prototype (Click once on the name to see the full path): **BswM_SRI_BswM_MSI_MdgShutdownTimerState_BswM_MDGP_MdgShutdownTimerState/requestedMode**
 - > Init Value: **TIMER_NOT_EXPIRED**

▶ Synchronize now [Synchronize now](#)



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ Create the following three Actions (for detailed description see next slide)

1. **ActShutdownTimerStart**

- > Type: **Timer Control**
- > Timer Action: **BSWM_TIMER_START**
- > Timer expiration [s]: **10**
- > Timer Ref: **BswMrpShutdownTimer**

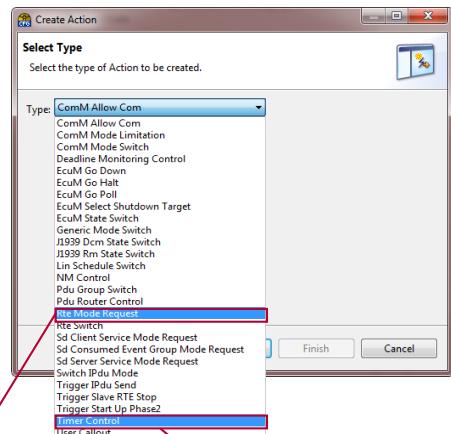
2. **ActShutdownTimerStop**

- > Type: **Timer Control**
- > Timer Action: **BSWM_TIMER_STOP**
- > Timer Ref: **BswMrpShutdownTimer**

3. **RTE_REQUEST_MrpPShutdownTimerDisplay_TIMER_EXPIRE**

- D
- > Type: **Rte Mode Request**
- > Requested Mode:
- > **MrpPShutdownTimerDisplay** → **TIMER_EXPIRED**

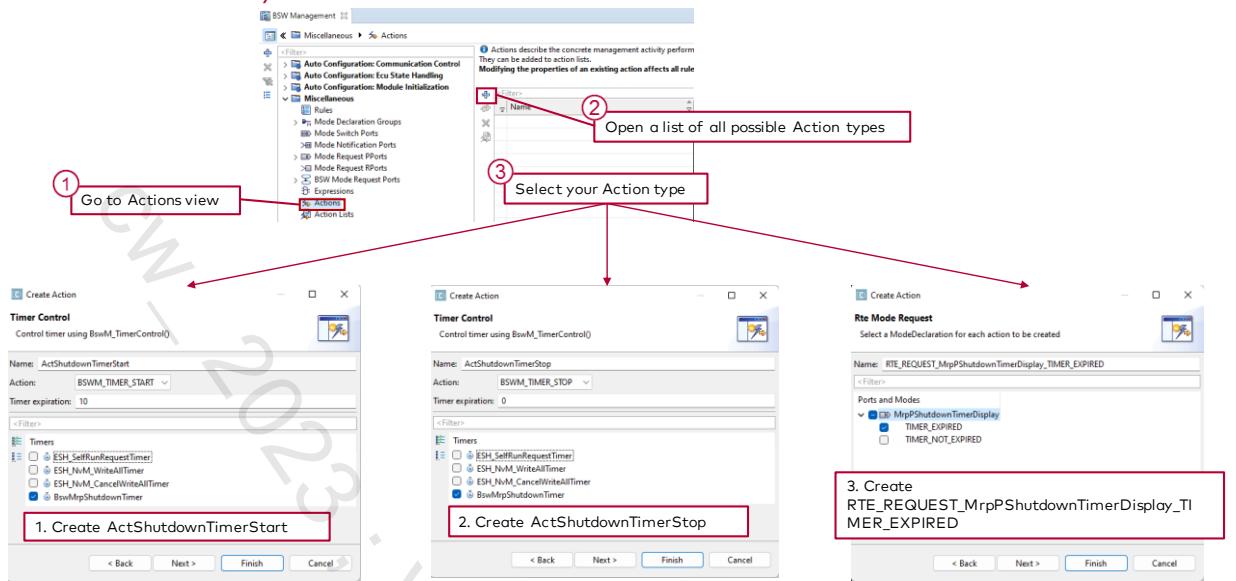
3. Create new Action with type "Rte Mode Request"



1. & 2. Create new Action with type "Timer Control"

Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay



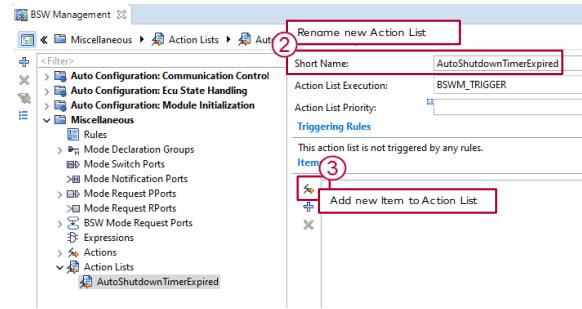
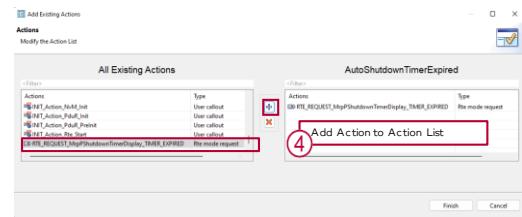
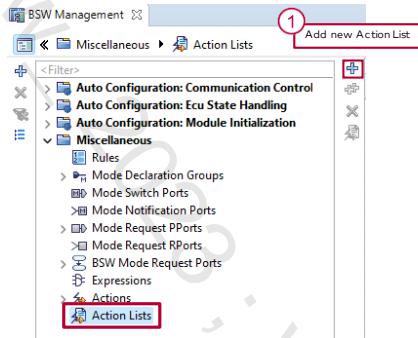
Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- Still in **Mode Management** → **BSWM Management** → **Miscellaneous**

- Create three **Action Lists** in the BSWM called:

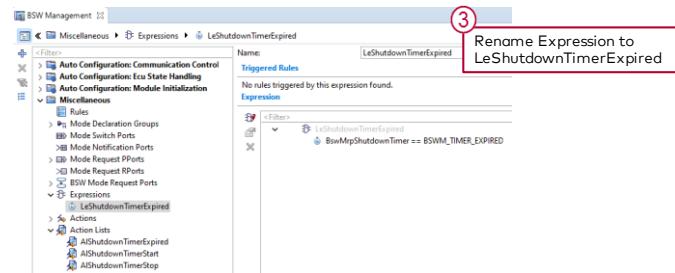
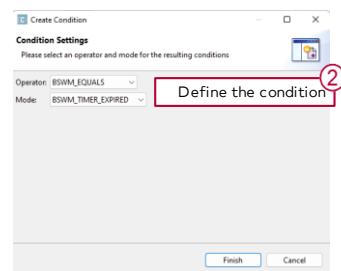
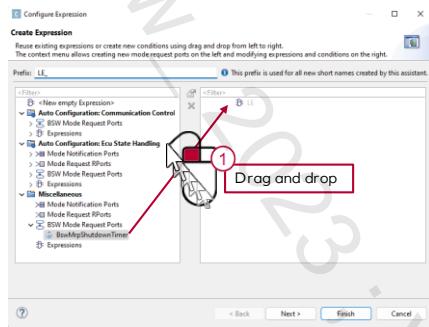
- 1. AIShutdownTimerExpired**
Item: RTE_REQUEST_MrpShutdownTimerDisplay_TIMER_EXPIRED
- 2. AIShutdownTimerStart**
Item: ActShutdownTimerStart
- 3. AIShutdownTimerStop**
Item: ActShutdownTimerStop



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

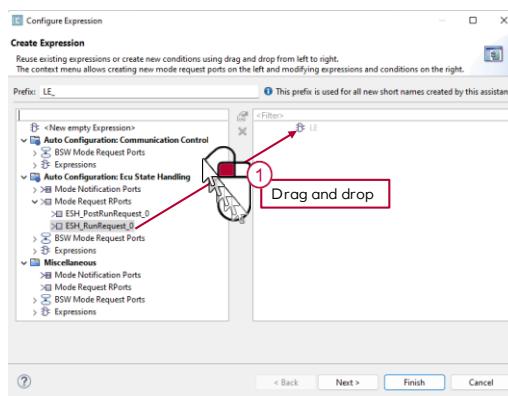
- ▶ Go now to Expressions
- ▶ Click on to create a new Expression
- ▶ Create the Expression **LeShutdownTimerExpired**
 - ▶ (1) Drag and drop the **BswMrpshutdownTimer** to LE
 - ▶ (2) Define the condition with
 - > Operator: **BSWM_EQUALS**
 - > Mode: **BSWM_TIMER_EXPIRED**
 - ▶ (3) Rename the Expression



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

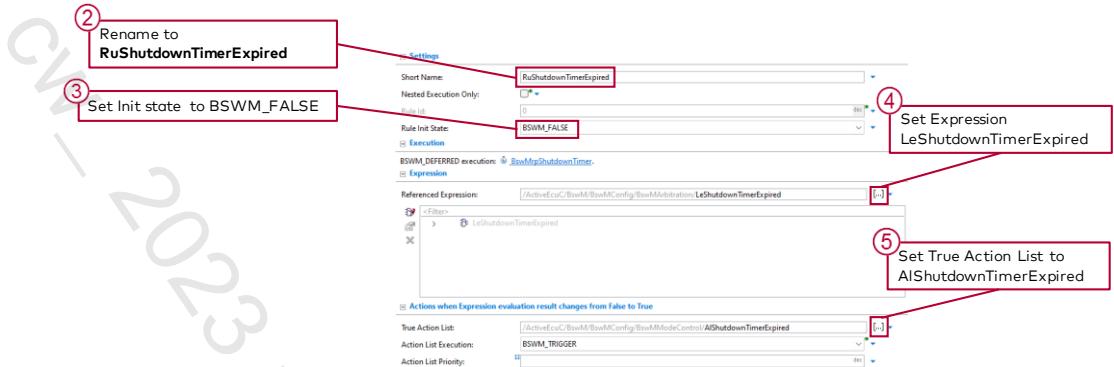
- ▶ Still in Expressions
- ▶ Create the Expression **LeShutdownTimerStart**
- ▶ Drag and drop the **ESH_RunRequest_0** to LE
- ▶ Define the condition with
 - > Operator: **BSWM_EQUALS**
 - > Mode: **RELEASED**
- ▶ Rename the Expression



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ (1) Add a new BSWM Rule
- ▶ (2) Short Name: **RuShutdownTimerExpired**
- ▶ (3) Rule Init State: **BSWM_FALSE**
- ▶ (4) Referenced Expression: **LeShutdownTimerExpired**
- ▶ (5) True Action List ("No Actions when Expression evaluates to True"): **AIShutdownTimerExpired**



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ Create second BSWM Rule
- ▶ Short Name: **RuShutdownTimerStart**
- ▶ Rule Init State: **BSWM_FALSE**
- ▶ Referenced Expression: **LeShutdownTimerStart**
- ▶ True Action List ("No Actions when Expression evaluates to True"): **AIShutdownTimerStart**
- ▶ False Action List ("No Actions when Expression evaluates to False"): **AIShutdownTimerStop**

The screenshot shows the BSW Management software interface. The left sidebar shows a tree structure with 'Miscellaneous' selected. Under 'Miscellaneous', 'Rules' is expanded, showing two entries: 'RuShutdownTimerExpired' and 'RuShutdownTimerStart'. The right panel displays a table with the following data:

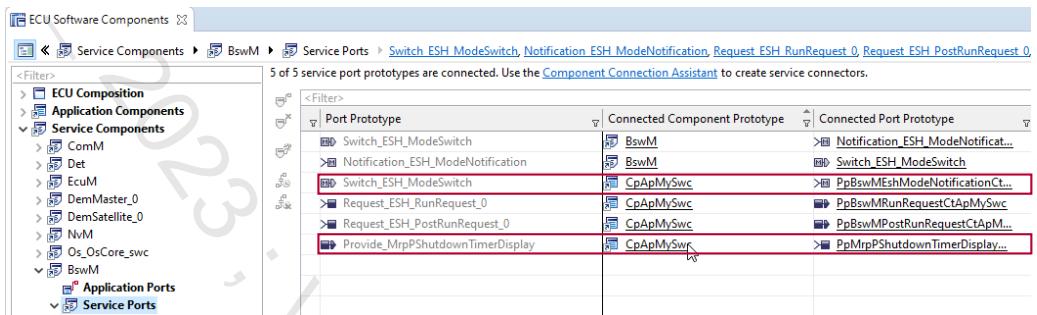
Name	Nested Execution Only	Init State	True Action List	False Action List	Referenced Expression
RuShutdownTimerExpired	<input type="checkbox"/>	* BSWM_FALSE	AIShutdownTimerExpired		LeShutdownTimerExpired
RuShutdownTimerStart	<input type="checkbox"/>	* BSWM_FALSE	AIShutdownTimerStart	AIShutdownTimerStop	LeShutdownTimerStart

A tooltip above the table provides information about rules and their execution logic.

Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ (Synchronize [Synchronize now](#) and save)
- ▶ Go to **Runtime System** → **ECU Software Components** → **Service Components** → **BswM** → **Service Ports**
- ▶ Perform a Bottom-Up service configuration of Service Port **Provide_MrpPShutdownTimerDisplay** to **CtApMySwc**
 - ▶ Create a P-Port Port Prototype for **CtApMySwc** as a counterpart to the Mode Request PPort of the BSWM
 - ▶ Rename it to **PpMrpPShutdownTimerDisplayCtApMySwc**
- ▶ Perform another Bottom-Up service configuration of Service Port **Switch_ESH_ModeSwitch** to **CtApMySwc**
 - ▶ Create Port Prototype at **CtApMySwc** as counterpart to the Mode Switch Port of the BSWM
 - ▶ Rename it to **PpBswMEshModeNotificationCtApMySwc**
- ▶ Save and exit



Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

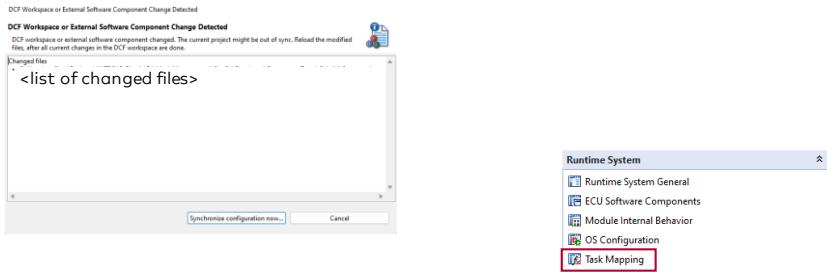
with 10 seconds delay

- ▶ "Configure SWCs" using **MyECU.dpa** file in folder \E4_ModeManagement\
- ▶ Add new Runnable **RCtApMySwcPostRunCode**
 - ▶ Add Access Points:
 - > **PpLightStateFront.DeLightState** (direct / explicit write)
 - > **PpLightStateRear.DeLightState** (direct / explicit write)
 - > **PpBswMPostRunRequestCtApMySwc.requestedMode** (direct / explicit write)
 - > **PpComMUserRequestCtApMySwc.RequestedComMode** (invoke operation)
 - > **PpMrpPShutdownTimerDisplayCtApMySwc.requestedMode** (direct / explicit read)
 - ▶ Add Triggers:
 - > **PpMrpPShutdownTimerDisplayCtApMySwc.requestedMode** (On Data Reception)
 - > **PpBswMModeNotificationCtApMySwc.POSTRUN** (On Mode Entry)
- ▶ Add Access Point to **RCtApMySwcCode**
 - ▶ **PpBswMRunRequestCtApMySwc.requestedMode** (direct / explicit write)
- ▶ Save project and exit Developer

Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)

with 10 seconds delay

- ▶ Change to Configurator Pro, execute auto-solve action "Synchronize System Description" or click "Synchronize configuration now"



- ▶ In the Validation window the error **RTE01056** will occur
- ▶ Go to **Runtime System** → **Task Mapping** → **Application Components** → **CpApMySwc**
- ▶ Mark the Runnables **RCtApMySwcPostRunCode** and move them to **Position 4** of **My_Task**

Part 6/6: Add-On II: ECU Shutdown based on Clamp 15 (ignition)**with 10 seconds delay**

- ▶ Generate
- ▶ Press F7 in order to generate SWC Templates
- ▶ Program and test the desired behavior within **RCtApMySwcCode** and **RCtApMySwcPostRunCode**
 - ▶ **RCtApMySwcCode**
 - > Simulate that the door states are just checked if the ignition is on
 - > Request Run Request if the ignition is on
 - > Release Run Request if the ignition is turned off
 - ▶ **RCtApMySwcPostRunCode**
 - > React on the current state of the ShutdownTimer
 - > If the ShutdownTimer is still running turn on the interior lights and request a full communication
 - > If the ShutdownTimer is expired turn off the interior lights and request the ComM mode no communication. Also release the current PostRun mode

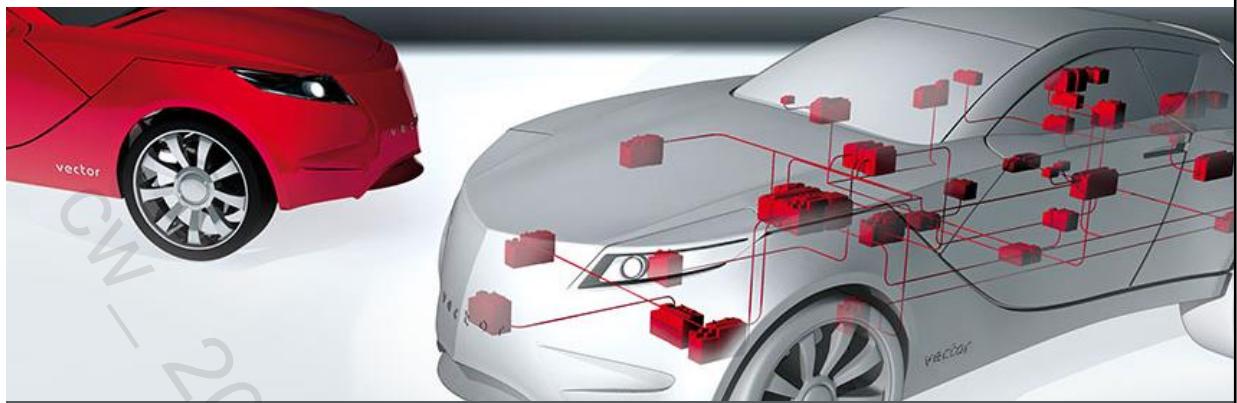
Summary

Think about what you have done in this exercise and why.

- ▶ There is a new service component: **COMM**
- ▶ How did you use it?
- ▶ Think about the connection between **COMM** and **ECUM** for wakeup source detection.
- ▶ Think about requesting and releasing communication.
- ▶ Mode Management was used to visualize state changes in the COMM.
 - ▶ Keyword: Mode Switch Events (Mode Disabling Dependencies)
- ▶ A shutdown algorithm was implemented using the BSWM.
 - ▶ You should have an idea about what the BSWM can do
 - ▶ Keyword: Rule-based system

Mode Management





AUTOSAR in Practice

Busses

V1.0.0 | 2020-01-20

Agenda

► **Introduction**

Busses - LIN

Busses - FlexRay

Ethernet

Coming up next

Bus access methods

- ▶ Master-Slave method
 - ▶ Master ECU controls data transmission on the bus
 - ▶ Slave ECUs may only respond to requests by the Master ECU
 - ▶ Example: **LIN** bus
- ▶ Token passing method
 - ▶ Send authorization (Token) is passed from one ECU to the next
 - ▶ Sending is only possible with send authorization
 - ▶ Examples: Token ring, Profibus
- ▶ Event-driven method
 - ▶ Any ECU can access the available bus at any time
 - ▶ Priority assignments makes it possible to control message traffic
 - ▶ Examples: **CAN** bus, **Ethernet**
- ▶ Time-synchronous method
 - ▶ Sending process occurs in periodic time windows
 - ▶ Time windows are allocated to the individual ECUs
 - ▶ Examples: **FlexRay**, **TTCAN**

Bus access methods



Bus access methods control access to the medium used. They are used to define which participants (nodes, sending stations) may occupy the bus for a send operation. Bus access methods may be subclassified into four types:

- Master-Slave methods

In the Master-Slave methods, data transmission is controlled by a higher-level node (Master). Subordinate nodes (Slaves) may only access the bus when there is a request from the Master that should be answered.

- Token passing methods

In the Token Passing method, the right to access the bus is passed from one node to the next with the help of a send authorization (Token). This method ensures that only one node is authorized to send at any given time.

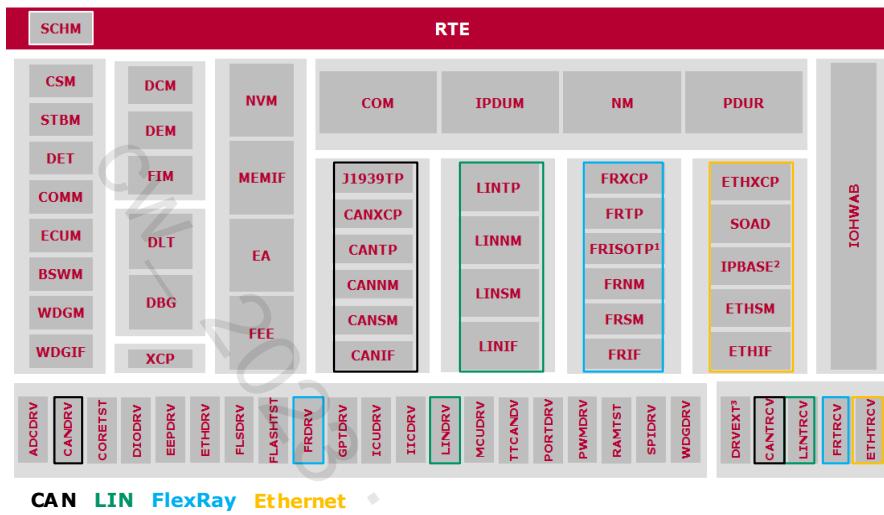
- Event-driven methods

In event-driven methods, any ECU may access the available bus at any desired point in time. As soon as a node obtains bus access, all other nodes must wait until the bus is released again. As an extension, it is also possible to assign message priorities to exert control over the message traffic. The most popular event-driven methods are CSMA/CD (Carrier Sense Multiple Access / Collision Detection → Ethernet) and CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance → CAN bus).

- Time-synchronous methods

Time-synchronous methods provide participants in a network with periodic time windows in which each individual node may send out its desired messages. In contrast to Token Passing methods, no send authorization is passed. Instead, the time windows are permanently allocated to the individual nodes. In FlexRay, the TDMA method is used.

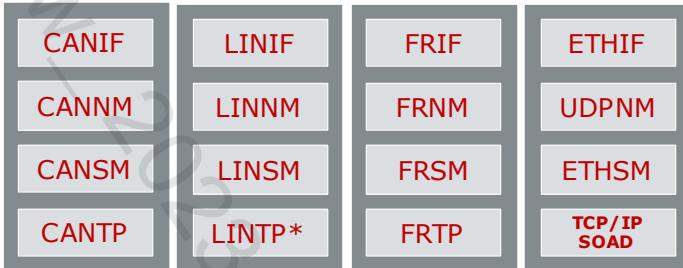
CAN, LIN, FlexRay and Ethernet in AUTOSAR



CAN – LIN – FlexRay

Appearance of AUTOSAR for CAN, LIN and FlexRay is almost identical

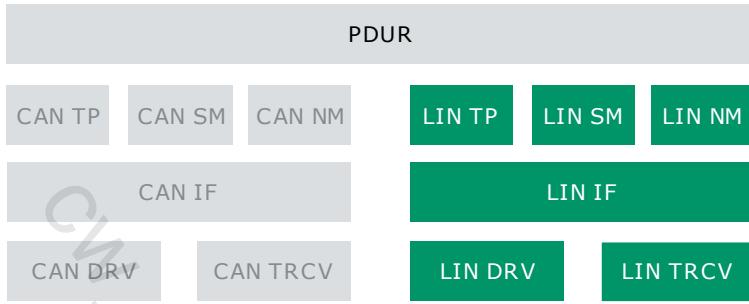
- ▶ Handling of Data Elements in the higher layers independent of underlying bus system
- ▶ Depending on bus protocol concepts there are differences...



*option included in LinIf

Agenda

- Introduction
- ▶ **Busses - LIN**
- Busses - FlexRay
- Ethernet
- Coming up next

CAN – LIN

The LinNm functionality only coordinated between the network mode and the sleep mode of the LIN cluster.

In AUTOSAR <v4.0 is not a module of its own. Ther, the lean functionality is performed by the LinSM.

In AUTOSAR >v4.0 the LINNM is located below the NM.

LIN BSW Modules

LIN SM

- ▶ Schedule requests, set transceiver mode, control wakeup and sleep

LIN DRV

- ▶ Low level driver for LIN communication via SCI/UART of the µC.
- ▶ Initialization of SCI Hardware
- ▶ API for generating LIN Synch break
- ▶ **LINNM**
- ▶ Manage Network Mode or Sleep Mode, Transmit NM message
- ▶ Provide Interface to NM
- ▶ coordinator functionality

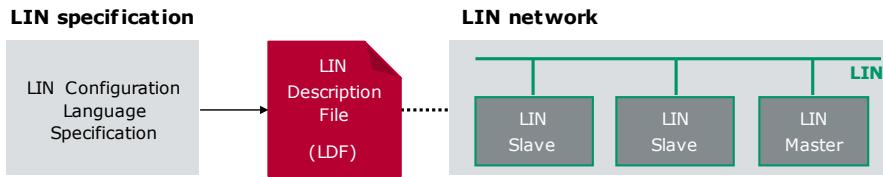
LIN TRCV

- ▶ Driver for external LIN Transceiver
- ▶ Control of wakeup/sleep
- ▶ Network diagnostics (short circuit, open line, ...)

LIN IF/TP

- ▶ Schedule table implementation
- ▶ Transmission/Reception of LIN frames
- ▶ Sleep/Wakeup/Error/Timeout Handling
- ▶ Transport Protocol for Diagnostics

Network Description



- ▶ The LIN Description File (LDF) describes a LIN network using the language defined in the LIN Configuration Language Specification
- ▶ The LDF contains information relevant to network development and analysis
 - ▶ Global information such as baud rate and versions
 - ▶ Information on nodes, signals and frames, sender and receiver
 - ▶ Schedule

Network Description

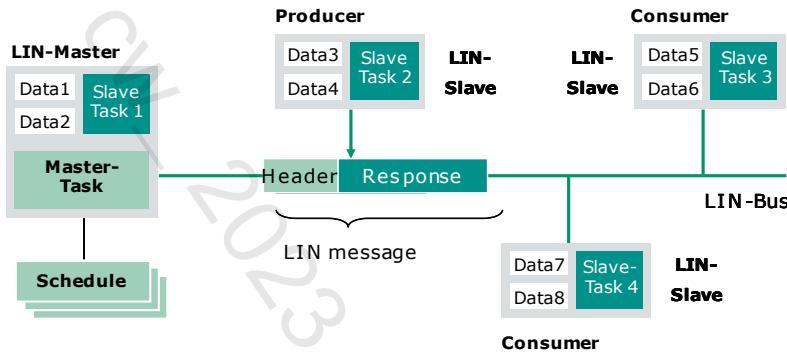


The LIN Configuration Language Specification is utilized to create the so-called "LIN Description File" or "LDF". The LIN Description File encompasses, among other things, the LIN protocol version, LIN language version, baud rate, definitions of nodes, signals and frames, as well as information on the schedule. Consequently, the LDF is the foundation for developing a LIN network and associated LIN nodes and for analyzing and observing a LIN network.

The syntax of the LDF is so simple that it is rather easy to create such LDFs manually. Nonetheless, computer-supported development, i.e. automatic design and generation, play a dominant role here. That is because introduction of the so-called "Node Capability Language" makes it possible to create LDFs automatically.

Centrally controlled message distribution system

- ▶ LIN nodes do not have equal rights due to master-slave architecture
- ▶ LIN master delegates communication (Delegated Token Principle)
- ▶ Message distribution based on message addressing
- ▶ 64 message addresses (identifiers)



Centrally controlled message distribution system

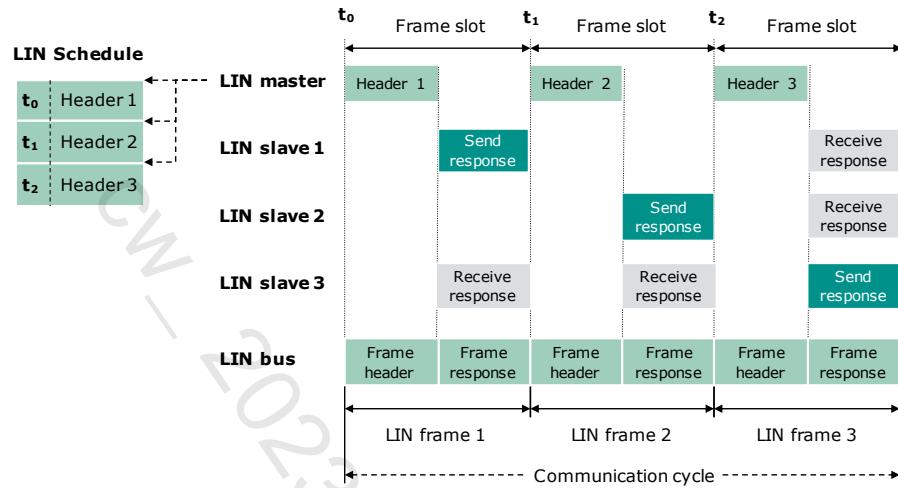
i

The LIN network is based on a master-slave architecture. Reason: The master/slave method underlying a master-slave architecture is easy and economical to implement. One node is chosen to control all communication. This is the LIN master. The LIN master performs the functionality of a bus arbiter with the help of the so-called "master task" and the so-called "LIN schedule".

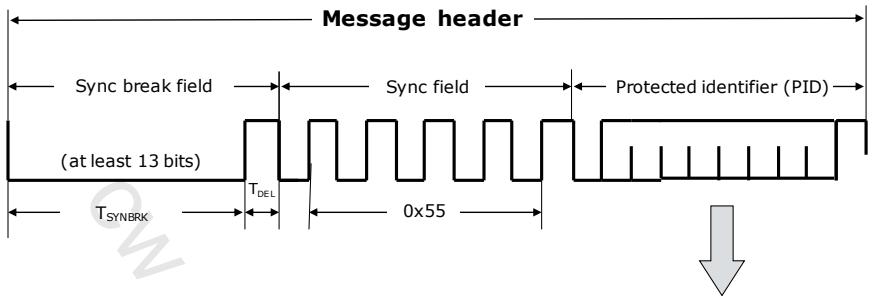
The LIN schedule establishes the send times of the LIN messages to be transmitted. Conforming to the LIN schedule, the LIN master places special messages (headers) on the bus at defined times. A header can be understood as a request, and it contains a message address. This is evaluated by the LIN slaves. A LIN slave has three alternatives for reacting to the header: Send response, receive response, neither send nor receive.

The combination of header and response is referred to as a LIN message. A total of up to 64 LIN messages may be defined. Each LIN message is available for reception by all LIN nodes due to message addressing, even the LIN master, if it has a slave task. The LIN network is essentially a message distribution system with central control.

Scheduling



Messages - Identifier



- ▶ P0 = even parity
over ID0, ID1, ID2, ID4
 - ▶ P1 = odd parity
over ID1, ID3, ID4, ID5



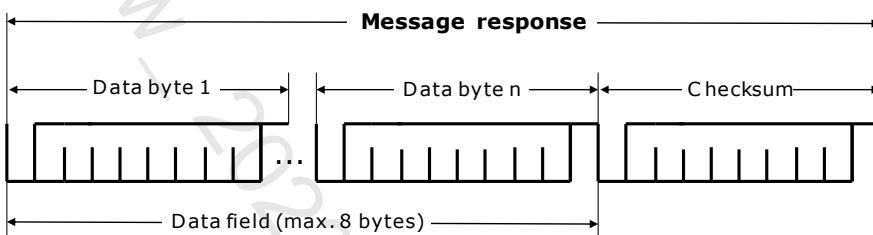
- ▶ Address range: 0-63
 - ▶ 60, 61: Diagnostics
 - ▶ 62, 63: Reserved

i

The identifier is transmitted in the context of sending the protected identifier. It is composed of six bits. This means that 64 different LIN messages can be specified. The identifier is protected by two parity bits (even and odd parity). The two identifiers 60 and 61 are utilized for diagnostics. The two identifiers 62 and 63 are reserved.

Message - Response

- ▶ The message response is sent by a slave task
- ▶ The message response contains the data and the checksum
 - ▶ Classic checksum over data field
 - ▶ Enhanced checksum over data field and ID field



Message - Response



The message response is sent by a LIN slave delegated by the message address. A maximum of eight useful bytes may be transported with one message response. It must be ensured that byte transmission begins with the LSB. In transporting a word, transmission begins with the low byte (little endian transmission, Intel mode). In principle, a message response can be received and accepted by all LIN slaves.

The useful data are protected by a checksum. Checksum formation is based on modulo-2 arithmetic and carry bit transmission. The individual useful bytes are added by modulo-2 arithmetic. Overflow bits are transmitted. Finally, the result is inverted.

There are two different types of checksum: Classic checksum and enhanced checksum. The classic checksum only protects the useful data. The enhanced checksum protects the useful data and the identifier. In LIN 1.3 conformant LIN nodes, the message responses to be transmitted are always equipped with the classic checksum, because LIN 1.3 does not recognize the enhanced checksum. It should be ensured that LIN messages with identifiers 60 and 61 (diagnostics) are always protected by the classic checksum.

Message Types

Unconditional frame (ID 0-59)

- ▶ A message response is assigned to the message header
- ▶ Message header is always sent in the reserved frame slot

Event triggered frame (ID 0-59)

- ▶ Several message responses are assigned to the message header
- ▶ Message header is always sent in the reserved frame slot
- ▶ Message response is only sent if necessary
- ▶ Collisions are resolved by the LIN master

Sporadic frame (ID 0-59)

- ▶ Multiple unconditional frames are assigned to a frame slot
- ▶ Transmission of unconditional frames depends on needs

Diagnostic frame (ID 60-61)

- ▶ ID=60: Master request frame (= Diagnostic request)
- ▶ ID=61: Slave request frame (= Diagnostic response)

Message Types



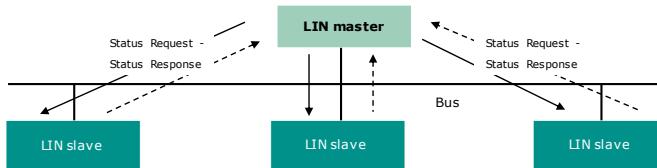
Different frame types are available for data transmission in a LIN network: Unconditional frame, event triggered frame, sporadic frame and diagnostic frame.

The unconditional frame is characterized in that there is exactly one sender for the message response. In contrast to this, in event triggered frames multiple responses are assigned to the header, which if necessary can be sent by the relevant LIN slaves. A collision leads to a switchover to the so-called "collision resolving schedule table".

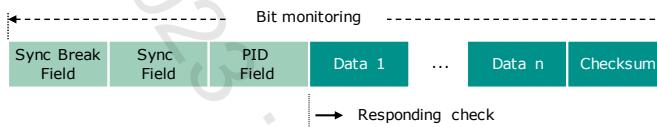
A sporadic frame represents a group of unconditional frames that share the same frame slot and are transmitted as needed. In the case where multiple nodes wish to send, the order of sending is determined by the priority of the unconditional frames. If none of the slaves wishes to send, the frame slot remains empty.

The two IDs 0x3C and 0x3D are used for diagnostics. The LIN frame with ID 0x3C is referred to as the master request frame, and it corresponds to the diagnostic request. The LIN frame with ID 0x3D is identified as the slave request frame, and it corresponds to the diagnostic response. The layout of these two frames results from the node configuration and identification specification.

Status Management



- ▶ Error signaling with the help of "Status bits" (Response_Error)
 - ▶ Use of Unconditional Frames
 - > Message header: Status request
 - > Message response: Status response (with Response_Error)
- ▶ LIN master collects the status bits
- ▶ Error handling is not part of the LIN specification

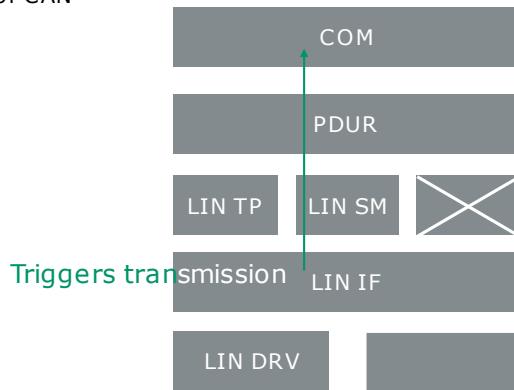


A LIN slave manages two status bits (`error_in_response` and `successful_transfer`), which are available for further processing. The LIN slave must transmit the "`error_in_response`" bit once per communication cycle with an unconditional frame. In principle, the message corresponds to the status request, and the message response corresponds to the "`error_in_response`" bit of the status response. This bit is also referred to as the status bit or "`response_error`". Error handling is based on the status bits collected by the LIN master; however, this is not part of the LIN specification.

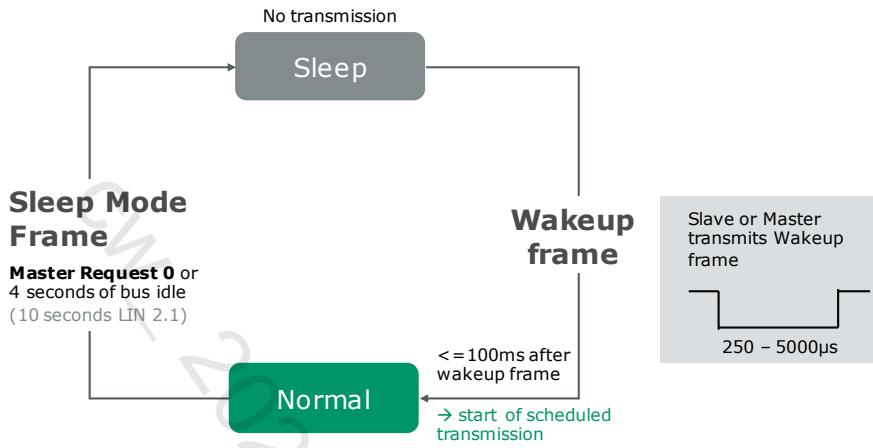
Transmission Modes

LIN IF has knowledge about Schedule tables

- ▶ LIN IF controls transmission of LIN frame according to current Schedule table
- ▶ No transmission modes like for CAN

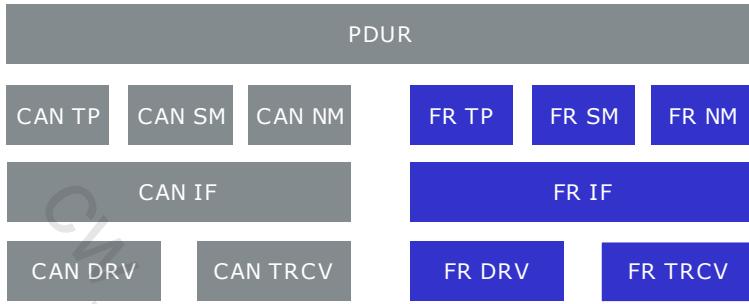


Sleep and Wakeup



Agenda

- Introduction
- Busses - LIN
- **Busses - FlexRay**
- Ethernet
- Coming up next

CAN – FlexRay

FlexRay BSW Modules

FR SM

- ▶ Mastering states for FlexRay bus

FR IF

- ▶ Provides equal mechanism to access FlexRay bus channel

FR TP

- ▶ Uses dynamic and static part of communication
- ▶ Segmentation of data in transmit direction
- ▶ Collection of data in receive direction

FR NM

- ▶ Synchronized transition to bus sleep
- ▶ Network configuration at startup and during operation
- ▶ Error recovery

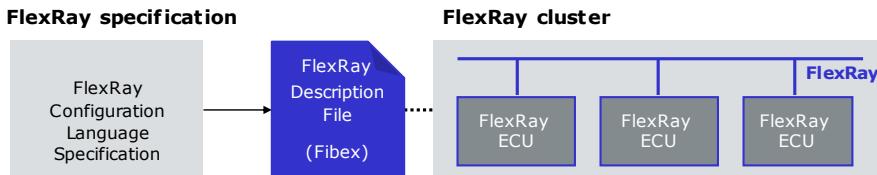
FR DRV

- ▶ Abstracts hardware-related differences of FlexRay controllers
- ▶ Provides uniform API

FR TRCV

- ▶ Controls wakeup/sleep
- ▶ Network diagnostic (short circuit, open line, ...)

Network Description



- ▶ The Field Bus Exchange Format (Fibex) describes a FlexRay network using the language defined in the Field Bus Exchange Format.
- ▶ The Fibex contains information relevant to network development and analysis
 - ▶ Global information such as e.g. baud rate
 - ▶ Information on nodes, signals and frames, sender and receiver
 - ▶ Schedule

FlexRay Basics

- ▶ Time Triggered protocol with a data rate up to 10 MBit/s
- ▶ 2 channels possible
- ▶ Payload up to 254 bytes
- ▶ Global time synchronization within whole FlexRay cluster
- ▶ Time synchronization must be achieved before data transmission starts
- ▶ Static segment
 - ▶ TDMA access (slots) with no arbitration
 - ▶ Fixed bandwidth allocation per ECU → null frames in case no data transmitted
- ▶ Optional dynamic segment
 - ▶ FT DMA (minislots) used for arbitration
 - ▶ Flexible bandwidth allocation → transmission only if necessary
- ▶ Wake up possible only through special FR wake up pattern

**TDMA**

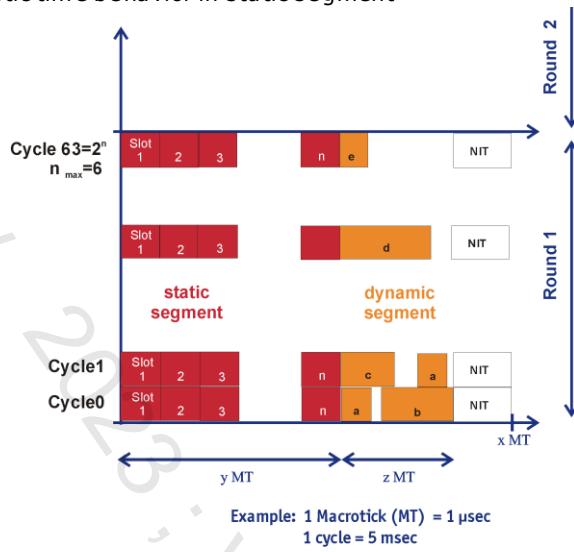
Time Division Multiple Access

FTDMA

Flexible Time Division Multiple Access

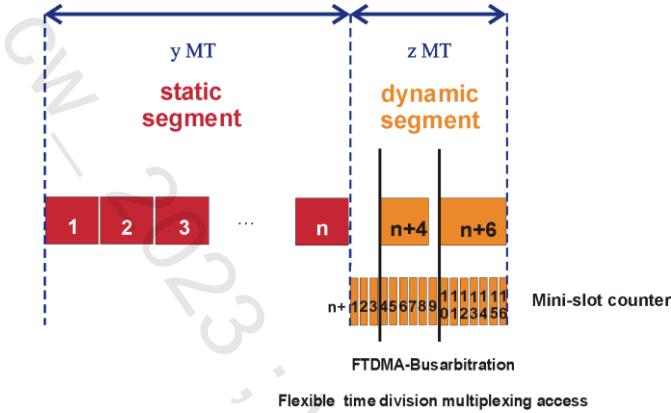
FlexRay Basics

- Deterministic time behavior in static segment



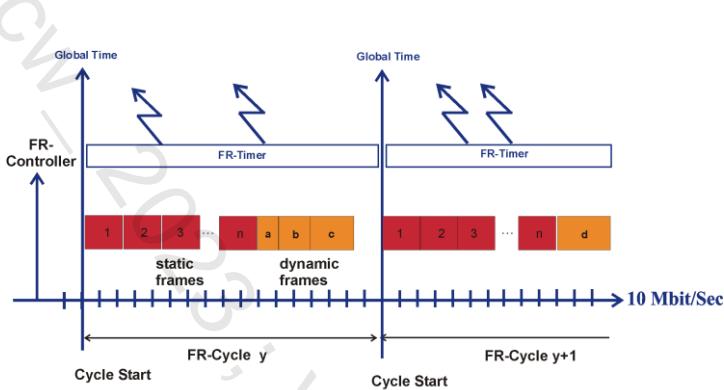
FlexRay Basics

- ▶ Priority controlled bus access in dynamic segment (like CAN)
- ▶ In case too many frames with higher priority: frame sent in a later cycle
- ▶ Transmission in dynamic segment only guaranteed for certain identifiers



FlexRay Basics

- ▶ Reception and transmission are Time Triggered
- ▶ Synchronization of application components might be necessary
- ▶ Synchronization of BSW modules thanks to:
 - ▶ FR-Global Time (TP, NM, ...)
 - ▶ FR-Timer (FrIf)



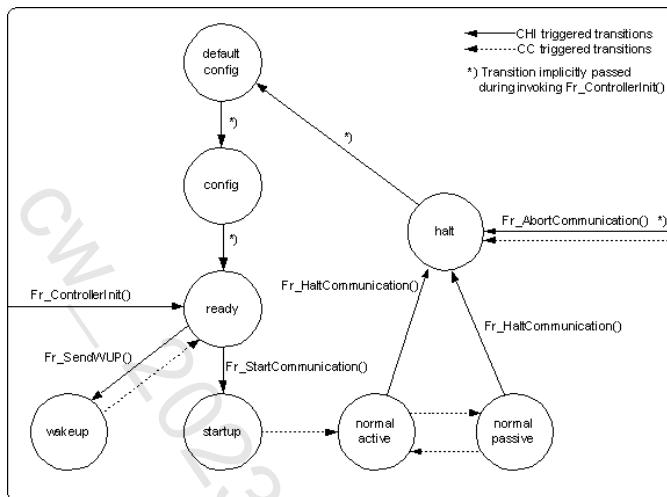
FlexRay Driver - General Description

Hardware abstraction providing:

- ▶ Initialization (based on ECU-configuration using FIBEX) and Control of Communication Controller (CC)
- ▶ Start of the FlexRay communication
- ▶ Message buffer access (sending and receiving frames)
- ▶ FR-Timer services and FlexRay Global Time access
- ▶ Interrupt handling (interrupt flags, interrupt handler)
- ▶ Error handling

No AUTOSAR main function (periodic function)

FlexRay Driver - Hardware Control



CC Status model

FlexRay Interface - General Concepts

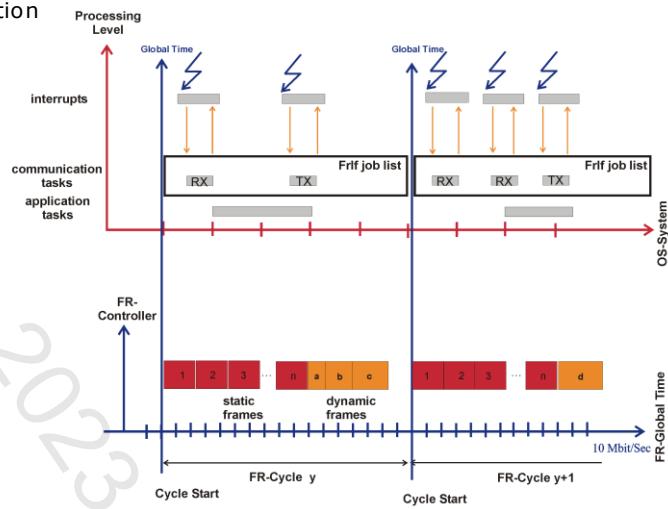
- ▶ **Functionality**
 - > Abstracts the usage of multiple FlexRay communication controllers in an ECU
 - > Provides a PDU based API for upper software layer modules
 - > Uses a job-list for the execution of communication operations:
 - > Assembles N-PDUs into frames (L-PDU) and gives them to Driver
 - > Polls Driver for received frames (L-PDU) and disassembles them into N-PDUs
 - > The job-list is directly scheduled by the FR-Timer (interrupt)

FlexRay Interface - General Concepts

- ▶ Particularities
 - ▶ Reception: No polling possible - Active indication to upper layer via (XXX_RxIndication) callback
 - ▶ Transmission: Active retrieval of payload from upper layer possible via (XXX_TriggerTransmit) callback
- ▶ FIBEX (ASAM – Field Bus Exchange Format):
 - ▶ Contains FlexRay configuration information
 - ▶ Up to version 2.0.1 uses Signal Groups for PDU-definition

FlexRay Interface - General Concepts

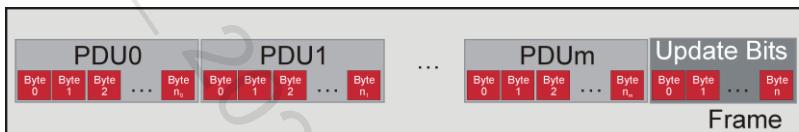
► Job List Execution



FlexRay Interface - Update Bits

All PDUs for which no transmission has been requested shall be marked within the Update Bits as invalid.

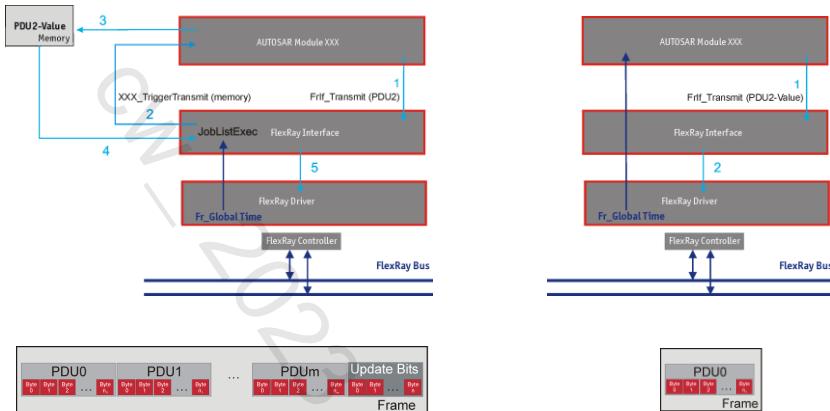
- ▶ Position of the Update Bits is not defined within FIBEX
- ▶ OEM-specific rules define position of the Update Bits in frame
- ▶ **FrIf Update Bits are different from COM Update Bits!**
- ▶ Only included in frame if decoupled transmission configured



FlexRay Interface - Transmission modes

► Decoupled transmission

- ❑ Transmission with **immediate** buffer access (only possible if single PDU / Frame)



FlexRay Transport Protocol - General Concepts

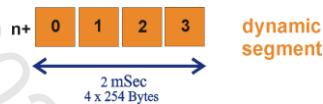
The AUTOSAR FrTp can be configured compatible to ISO 15765-2.

It basically provides the following additional services:

- ▶ Max. payload length of 4 GByte
- ▶ Segmentation of data in max. 254 Byte-PDUs (configurable)
- ▶ Variable transport frame length depending on PDU-Type:
 Rx-Frame / Tx-Frame / FlowControl-Frame
- ▶ Different acknowledgement types possible:
 None / ACK / ACK-RT
- ▶ Retry in case of error and transmit cancellation
- ▶ Immediate or decoupled transmission

FlexRay Transport Protocol - General Concepts

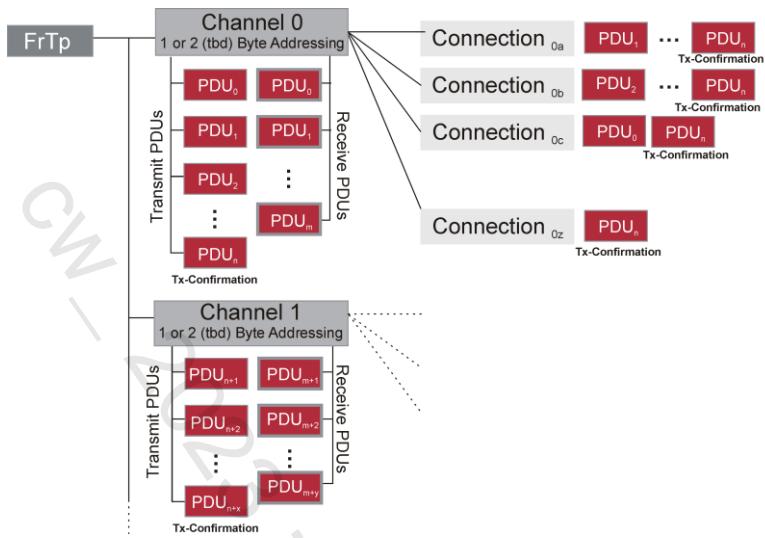
- ▶ AUTOSAR does not allow for dynamic payload length change during Run-Time.
- ▶ Transport frames can be transmitted in static or dynamic segment.
- ▶ Advantage of dynamic segment:
 - ▶ Transport frame length is independent of the application frame length.
 - ▶ Guaranteed dynamic transmission, if total frame length is in accordance to max. dynamic segment length. For instance if 40 % of one 5 msec-cycle shall be reserved for the dynamic segment:
 - > Example 1 (Flash-PDU): 4 guaranteed dynamic transport frames à 254 Bytes



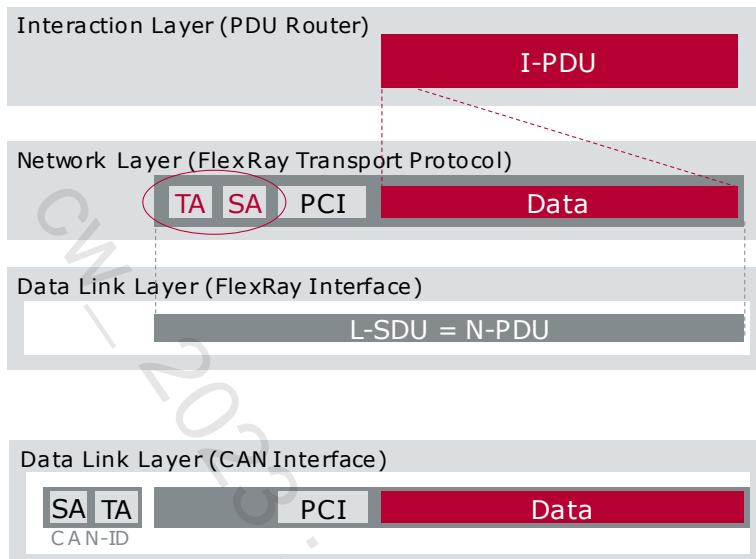
- Example 2 (Diagnostic-PDU): 31 guaranteed dynamic transport frames à 32 Bytes



FlexRay Transport Protocol - Channel & Connection



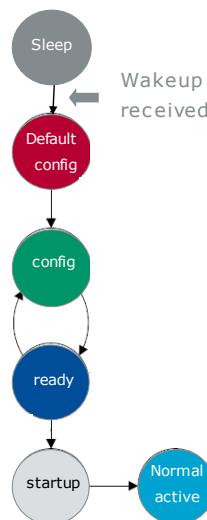
FlexRay Transport Protocol - Addressing Modes



Wakeup and Sleep

Wakeup

- When the bus driver receives the wakeup pattern from the FlexRay bus, this activates the entire FlexRay ECU.
- An ECU runs through the states up to **Ready**, beginning from **Sleep Mode**. Afterwards, it is ready for communication, but is not yet synchronized.



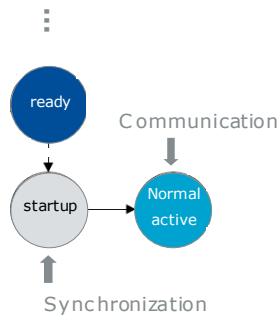
Source: Protocol specification 2.1 A

Wakeup and Sleep - Synchronization

State change of the ECU

(Ready → Normal Active)

- ▶ In Startup Mode certain ECUs begin with **synchronization**.
- ▶ These ECUs are referred to as Coldstart nodes, and they send out FlexRay frames with activated indicator bits:
 - > Startup bit → Frame used as Startup Frame
 - > Sync bit → Frame used as Sync Frame
- ▶ A **leading coldstart node** and a **following coldstart node** together form the communication cycle to which other ECUs without coldstart capability synchronize.



Wakeup and Sleep - Synchronization

i

Once all ECUs have reached the Ready mode after the wakeup process, it is possible to begin with synchronization (startup). Startup is executed by a macro; when it ends, this signifies the beginning of normal communication (normal active). There must be at least two ECUs in the FlexRay network that are coldstart capable. A coldstart refers to triggering of the startup macro, by which synchronization is produced. ECUs with coldstart capability are called coldstart nodes.

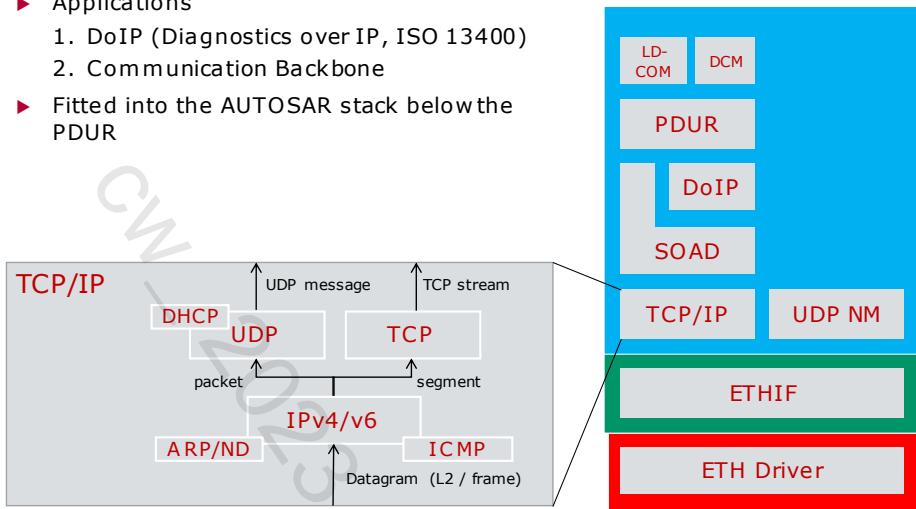
In the startup phase, the communication cycle is initially executed with the frames of the leading coldstart node and is then supplemented by the frames of the following coldstart node. The frames that are used for this are characterized by special indicator bits that are specific to the startup and sync frames. These frames are sent synchronously on both FlexRay channels. After the end of startup, the startup bit is cleared and the sync bit remains for continuous synchronization.

Wakeup and Sleep

Sleep

- ▶ FrNm only supports shut down
- ▶ All nodes switch off simultaneously
- ▶ VOTE: marks a node that needs the bus
 - ▶ Decrement of a counter for all cycles

- ▶ Applications
 1. DoIP (Diagnostics over IP, ISO 13400)
 2. Communication Backbone
- ▶ Fitted into the AUTOSAR stack below the PDUR





DHCP: Dynamic Host Configuration Protocol acc. To RFC 2131. DHCP allows to dynamically give an address to clients without manual configuration.

ARP: Address Resolution Protocol, used in IPv4 to retrieve a MAC Addresses suitable to a given IP address on the local network. It uses a broadcast for this.

ND: Neighbor Discovery Protocol (NDP), used in IPv6, replaces ARP. It works in a local network. If a node is not reachable on the local network, it determines the router to take.

ICMP: Internet Control Message Protocol, Status and diagnostics regarding the LAN Status, e.g. TTL.

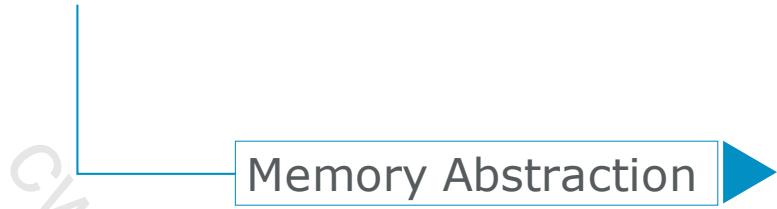
UDP: User Datagram Protocol (Layer 4)

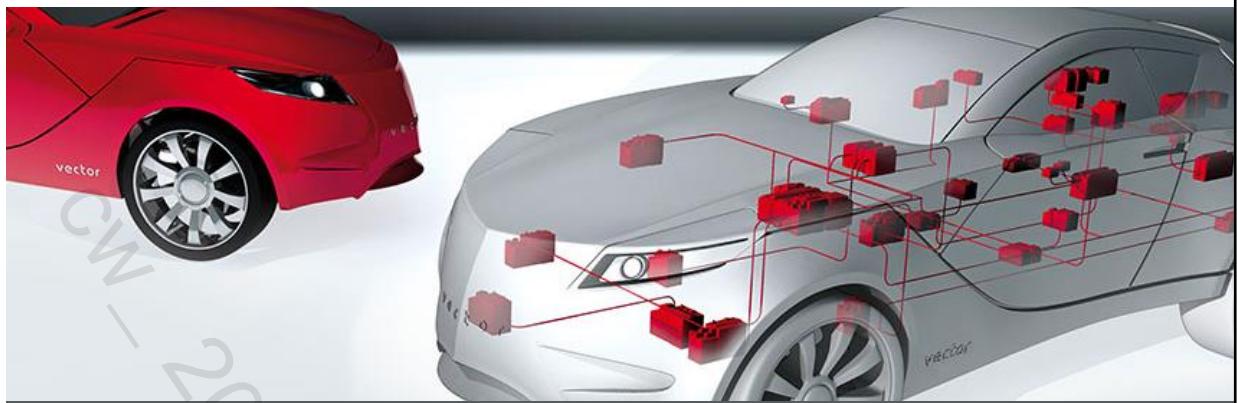
TCP: Transmission Control Protocol (Layer 4)

IP: Internet Protocol (Layer 3)

W2023_VH_Autosar CP Training_EN

Software Components





AUTOSAR in Practice

Memory Abstraction

V1.0.0 | 2020-01-20

Agenda

► **Principles of Memory Technologies**

The MICROSAR.MEM solution

Memory Handling

Configuration

NvBlockSwComponents

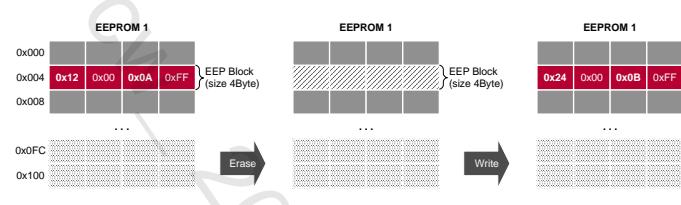
Service Mapping

Coming up next

Differences between EEPROM and Flash

EEPROM

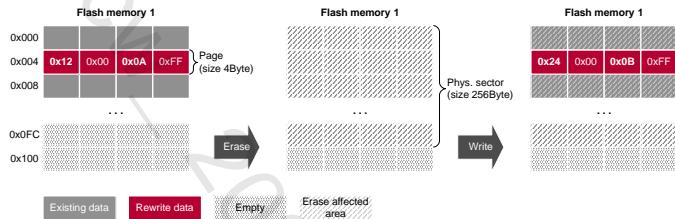
- ▶ Erasable and rewriteable in small blocks (1-4 bytes)
- ▶ Data can be rewritten to same memory cell easily



Differences between EEPROM and Flash

Flash EEPROM

- ▶ Has larger memory elements
- ▶ Smallest addressable memory unit (page) is not erasable on its own
- ▶ Multiple pages are combined into a physical sector which is erasable



Having different physical sector sizes also implies different time durations needed for sector erase. To erase 16KB may take hundreds of milliseconds, therefore an erase operation can last many seconds.

Agenda

Principles of Memory Technologies

► **The MICROSAR.MEM solution**

Memory Handling

Configuration

NvBlockSwComponents

Service Mapping

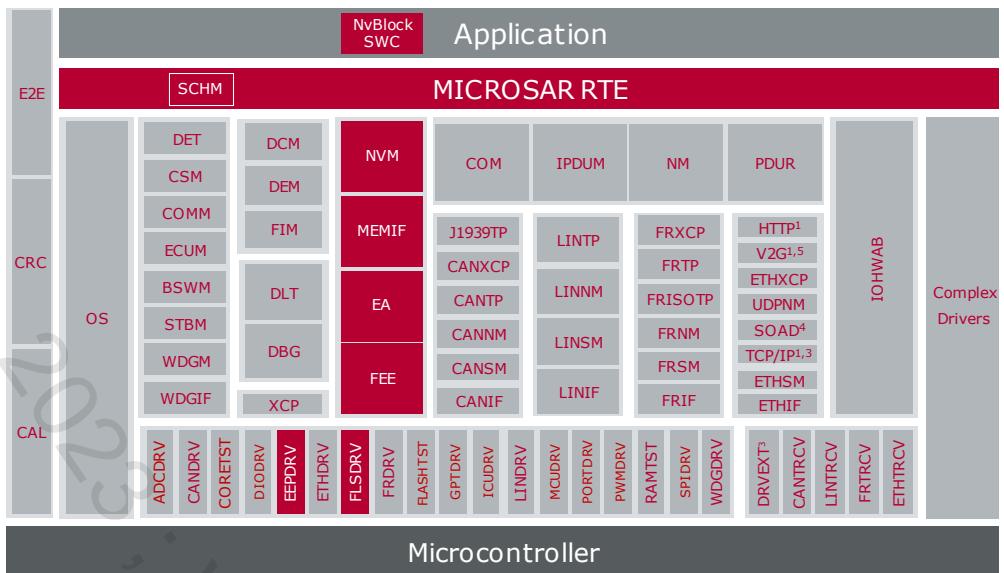
Coming up next

AUTOSAR BSW Modules for Memory Abstraction

- ▶ NVM
- ▶ MEMIF
- ▶ EA
- ▶ FEE
- ▶ EEPDRV
- ▶ FLSDRV

Optionally at
RTE level:

- ▶ NvBlock-Sw-
Component-
Type



AUTOSAR BSW Modules for Memory Abstraction

NVM

- ▶ Provides abstract data storage
- ▶ Startup and shutdown handling for NV data
- ▶ Associates NV data with ROM defaults and RAM working copy
 - ▶ permanently or temporarily
- ▶ Data may be spread across multiple "devices"
- ▶ Explicit Synchronization support
- ▶ Improved Job handling
- ▶ BSWM support*
 - ▶ Block Status Mode Notification
 - > Multi-Block, Single Block



*) BSWM Multi Block Job Status or Single Block Status Information: The NVM calls

- BswM_NvM_CurrentJobMode() (Multi Block, no call of the Multi Block Callback)
- BswM_NvM_CurrentBlockMode() (Single Block)

to notify the BSWM upon any job status changes of the Multi Block or Single Block respectively.

MEMIF

- ▶ Dispatches between memory technologies (FLASH or EEPROM)

FEE/EA

- ▶ Abstraction from different NV memory technologies (e.g. segmentation and number of write/erase cycles)

FLS/EEP

- ▶ NV memory hardware drivers
- ▶ Can access external devices e.g. via SPI

Nonvolatile Memory Manager (NVM)

- ▶ Queued job processing
 - ▶ Selectable between prioritized Queue or simple FIFO
 - ▶ Configurable queue depth
 - ▶ Separate queue for writing immediate data (e.g., crash data)
 - > Guarantees job acceptance and fast start of job
 - ▶ Job cancellation API to abort pending requests
- ▶ Startup and shutdown handling
 - ▶ Read data blocks from NV at startup
 - ▶ Write data blocks at shutdown
 - ▶ Blocks are user-selectable
- ▶ Support of multiple RAM data blocks
 - ▶ Via configurable **permanently-assigned** and **temporary** RAM blocks
 - ▶ Temporary RAM Blocks allow SWCs to implement double-buffering

Nonvolatile Memory Manager (NVM)

- ▶ Data CRC
 - ▶ Configurable per-block: OFF, CRC16, CRC32
 - ▶ To protect data in NV memory – detection of errors
 - ▶ To protect data in RAM – consistency check after reset
 - > E.g., due to watchdog reset or brown-out
 - ▶ Block-wise CRC recalculation in background (queued) to reduce CPU load
- ▶ Automatic / user-requested load of default data in case of read errors
 - ▶ From ROM or via callback (configurable per block)
- ▶ Configuration check at start-up - “dynamic configuration handling”
 - ▶ Ensures that current configuration is compatible with data layout of NV memory
 - ▶ Using NV data or block defaults in case of incompatibility
 - ▶ Potential incompatibilities are ignored if the “resistant against changed software” feature (“Res”) is enabled

Nonvolatile Memory Manager (NVM)

- ▶ Write protection of data
 - ▶ Temporary (set/unset): configurable at runtime
 - ▶ Write Once: persistent
- ▶ Handling of RAM block states
 - ▶ Valid/Invalid data → avoid writing of invalid data
 - ▶ Modified/Unchanged data → avoid writing of unchanged data
 - > NvM_SetRamBlockStatus() API exists to mark RAM data as
 - > "Valid and Changed" OR
 - > "Invalid and Unchanged"
- ▶ Redundancy
 - ▶ Increases data availability
 - > In case of write error or abort (e.g. through reset), a second instance is available
 - ▶ Redundancy shall be used for data which is critical for application
 - ▶ Redundant blocks shall always be secured with a CRC

Nonvolatile Memory Manager (NVM)

- ▶ An NVRAM block consists of:
 - ▶ A RAM block (permanent/temporary) to transfer data between application and NVM.
 - > Must be provided by application
 - ▶ RAM block management information (state, last request result)
 - ▶ One or more NV memory blocks of equal size
 - > Payload size equal to RAM block size
 - > Located in the same NV memory "device"
 - ▶ An optional default block which is either
 - > A user-provided block of data in ROM (no CRC)
 - > A user-provided callback

NVM Explicit Synchronization feature

- ▶ In the time between

`Nvm_WriteBlock/NvM_ReadBlock` and
`NvM_JobEndNotification`, the RAM working
copy is not accessible or valid for the application

- ▶ Solution alternatives

1. Double buffering using temporary RAM blocks
(RAM mirror and double buffer in RTE/SWC)

2. Explicit synchronization (double buffer as
extra RAM mirror implemented by the NVM)

- ▶ Can be enabled for single NvBlocks

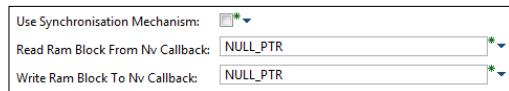
- ▶ Parameter:

`NvMBlockUseSyncMechanism=TRUE|FALSE`

- ▶ When the NVM reads or writes the RAM mirror, it
calls a callback function configured in Configurator
to collect data from or deliver data to the
application RAM mirror

> `NvMReadRamBlockFromNvM`

> `NvMWriteRamBlockToNvM`



12

NVM supports an optional Explicit Synchronization mechanism between application and NVM. It is implemented by an additional RAM mirror in the NVM module itself. The data is transferred by callbacks to the application which are called by the NVM module when needed. The synchronization mechanism can be configured for every NVRAM block separately. The NVM will create its RAM mirror based on the largest NVM blocks which uses explicit synchronization.

You can configure the names of the callback functions in Configurator:

```
Std_ReturnType <NvM_WriteRamBlockToNvm> ( void* NvMBuffer )
```

```
Std_ReturnType <NvM_ReadRamBlockFromNvm> ( const void* NvMBuffer )
```

Please note that for these callbacks are not Operations inside a Service Port Prototype. As a consequence, you have to provide *.c and *.h files where you will implement these functions. You have to include the *.h file as an "additional include" for the NVM module. Please also note that you have to remove the permanent RAM block from the respective block configuration, because the callbacks will copy the data without using the permanent block.

NVM Explicit Synchronization feature

i

NVM supports an optional Explicit Synchronization mechanism between application and NVM. It is implemented by an additional RAM mirror in the NVM module itself. The data is transferred by callbacks to the application which are called by the NVM module when needed. The synchronization mechanism can be configured for every NVRAM block separately. The NVM will create its RAM mirror based on the largest NVM blocks which uses explicit synchronization.

You can configure the names of the callback functions in Configurator:

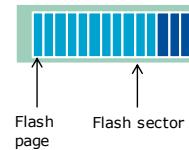
```
Std_ReturnType < NvM_WriteRamBlockToNvm > ( void* NvMBuffer )  
Std_ReturnType < NvM_ReadRamBlockFromNvm > ( const void* NvMBuffer )
```

Please note that for these callbacks are not Operations inside a Service Port Prototype. As a consequence, you have to provide *.c and *.h files where you will implement these functions. You have to include the *.h file as an "additional include" for the NVM module. Please also note that you have to remove the permanent RAM block from the respective block configuration, because the callbacks will copy the data without using the permanent block.

Flash EEPROM Emulation (FEE)

FEE is implemented according to the properties of Flash Memory

- ▶ Access to the memory using Flash pages only
 - ▶ These consist of several bytes
 - ▶ A Flash page
 - ▶ Can be written once, typically*
 - ▶ Cannot be erased individually
 - ▶ Sectors comprise several pages
 - ▶ Only sectors can be erased individually
 - > All constituent pages are erased together
 - ▶ Limited number of write/erase cycles per cell
 - > High utilization of a sector is advisable!
- Therefore a **dynamic data layout** is used to access the Flash memory and exploit one sector as effectively as possible



*) The flash technology only allows to modify bits from 1 → 0, but not from 0 → 1. Therefore a flash page already written has to be erased prior to further usage.

Flash EEPROM Emulation (FEE)

- ▶ Uses Flash drivers (FLS) to access flash hardware
- ▶ Dedicated Data Flash or Read-While-Write feature required in hardware
- ▶ Manages data through “linked lists” within a FEE sector
- ▶ Memory is always allocated for the configured “chunk” size
- ▶ Automatic switching between two FEE sectors
 - ▶ Copies most recent data block instances
 - ▶ Erases old FEE sector
 - ▶ At least two Flash sectors are necessary



1. Arrow indicates a link between two chunks.
2. In this example, each chunk consists of four chunk instances which are connected by a linked list.
3. A new list is created in a free Flash memory area after all four chunk instances have been filled with data.



Flash EEPROM Emulation (FEE) – key features

- ▶ Detection of incomplete writes
(e.g., due to reset or due to cancellation)
- ▶ Combines multiple physical Flash sectors to logical FEE sectors
→ FEE always handles two logical sectors
- ▶ Logical sectors may differ in size
- ▶ Multiple partitions (in pairs of 2 FEE sectors) are supported for independent operation
- ▶ Redundant internal management information for increased robustness
- ▶ Configuration update supported
- ▶ Shared configuration usage between Flash Bootloader and application possible



1. At least two sectors are required. This allows to erase a completely filled sector while writing to the second one.
2. The latest data is copied from the previously used sector to the newly erased sector while switching between the Flash sectors.

Flash Driver (FLS)

- ▶ Provides access to Flash hardware
- ▶ Depending on the platform, Flash area(s) to be used can be configured
- ▶ Sometimes limited to address either Data Flash or Program Flash.
 - ▶ Some types of Program Flash cannot be written at the same time you execute the code from them
 - ▶ Normally the application software code resides in ROM (Program Flash)
 - ▶ Data Flash can typically be accessed at any time

EEPROM Abstraction (EA)

EEPROM Memory

- ▶ Strategy to increase write cycles
 - ▶ Write one block to several memory locations in an alternating way



EEPROM Abstraction (EA) – Key features

- ▶ Detection of incomplete write operations by using header and footer
 - ▶ Caused by reset or cancellation
- ▶ ECC used for internally-added management information
 - ▶ Increased robustness and reliability
- ▶ “Walking concept” to increase number of write cycles
 - ▶ Each NV memory block consists of several instances
 - ▶ Written in a round-robin manner
- ▶ Automatic alignment of data supported in tool
 - ▶ Can speed up write operations
 - ▶ Avoids influences on other (adjacent) data blocks/instances

i

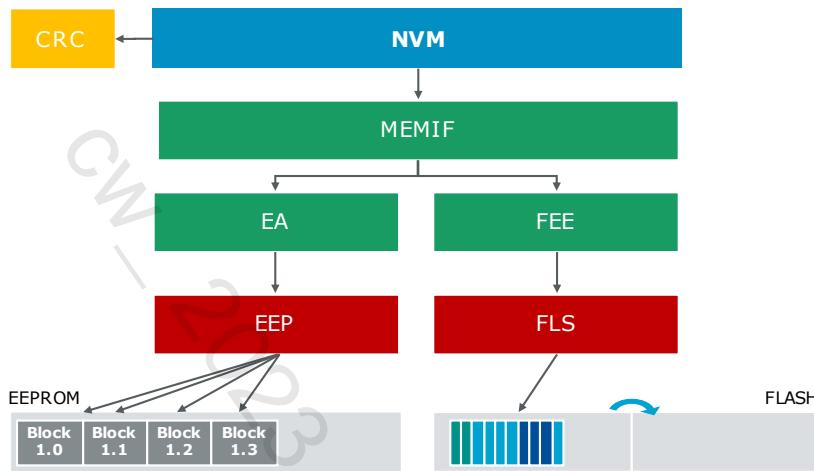
ECC: Error Correction Code – usually not provided by EEPROM hardware
ECC algorithm: combination of parity check and inverse management storage

EEPROM Driver (EEP)

- ▶ Direct access to the EEPROM HW
- ▶ Provides byte-wise writing even if not supported by HW
- ▶ Write Cycle reduction (configurable)
 - ▶ Write only data if it is different from the current content
- ▶ Provides Erase service even if not supported by hardware
 - ▶ E.g., SPI EEPROMs don't have an ERASE command
- ▶ Publishes device information that is necessary for the EA
 - ▶ Size of device, erased value (0x00/0xFF), writeable/erasable unit size

Summary

BSW Modules Overview

**i****CRC**

CRC Routines

NVM

NVRAM Manager

MEMIF

Memory Abstraction Interface

EA

EEPROM Abstraction

EEP

Internal/External EEPROM Driver

FEE

Flash EEPROM Emulation

FLS

Internal/External Flash Driver

Agenda

Principles of Memory Technologies

The MICROSAR.MEM solution

► **Memory Handling**

Configuration

NvBlockSwComponents

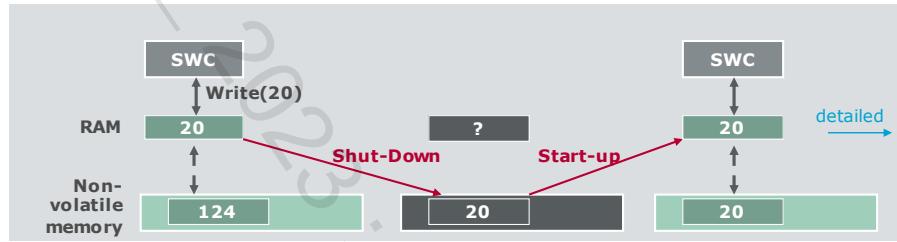
Service Mapping

Coming up next

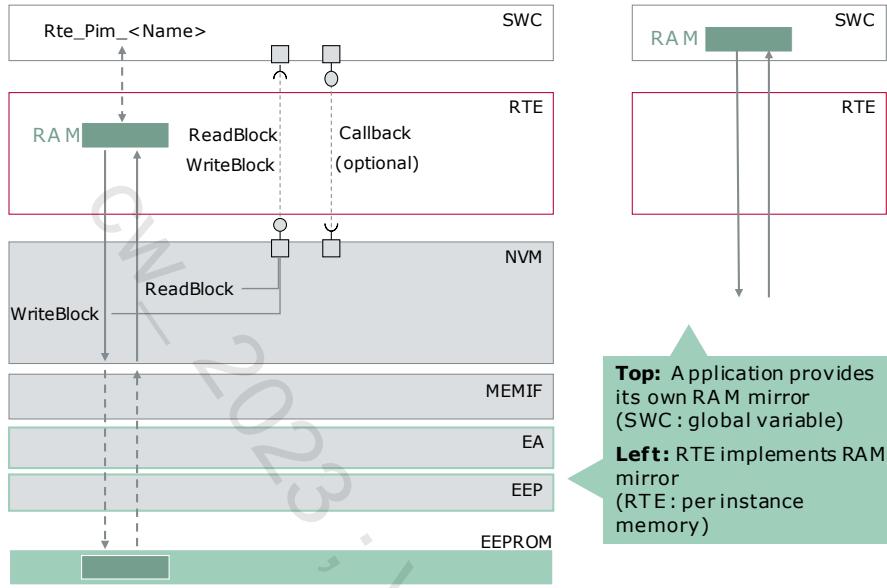
AUTOSAR Memory Abstraction

Reason for putting data into Non-Volatile (NV) Memory

- ▶ Power-off data persistence
- ▶ Application works on RAM mirror of the NV data
- ▶ At startup data can be copied from NV memory to RAM
- ▶ At shutdown data is copied from RAM to NV memory
- ▶ Data can be read from or written to NV memory on request



Interaction with the NVM Module



23

The NV Block

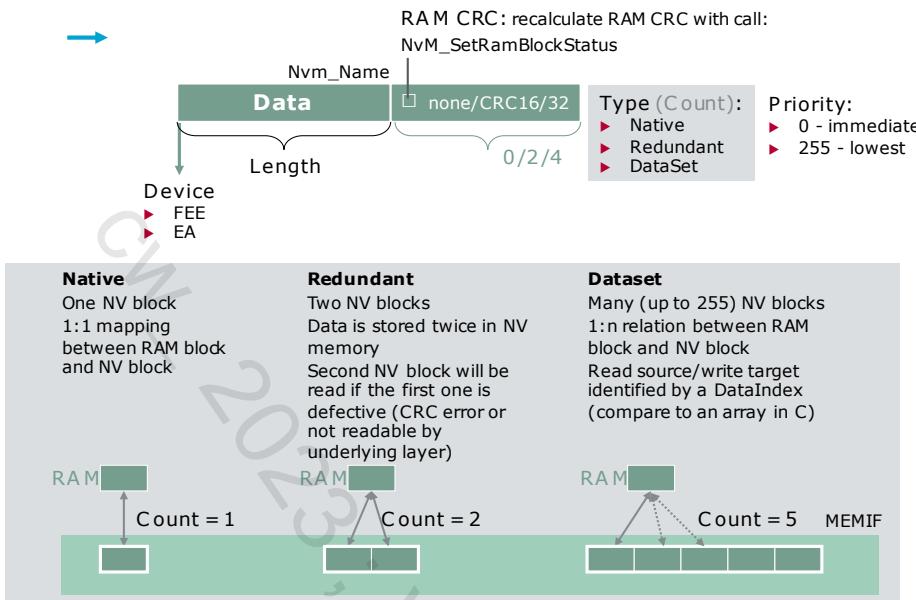
Base element is an NV Block

- ▶ It represents a memory area consisting of
 - ▶ NV user data
 - ▶ CRC value (optionally)
- ▶ Additional information necessary to define a Block
 - ▶ Name, Block ID, Size ...



The way how the blocks are defined and administered in the NV memory can be configured. >>

NVM Block Parameters



25

1. `NvM_SetRamBlockStatus()` triggers CRC calculation if configured and marks the RAM block as being changed and valid in order to trigger an NV block update during ECU shut-down
2. The currently used NV block for NVM **datasets** is configured by `NvM_SetDataIndex()`

The (re-)calculation of a block's CRC is done asynchronously by the `NvM_MainFunction()`. A CRC protected block's CRC value is calculated every time the block shall be written to NV memory. If a block is read from NV memory the CRC value is recalculated and compared to the one that was just read from NV memory.

If `NvM_SetRamBlockStatus(TRUE)` is called, the calculation of the CRC value over the RAM block's data is also initiated, unless the configuration option 'Calculate RAM CRC' was disabled for this block. The purpose of requesting recalculation of the RAM CRC with every call to `NvM_SetRamBlockStatus` is to provide the possibility to reuse the RAM data even if a soft reset (short power loss, watchdog reset) occurs.

Since CRC calculation is quite time consuming, especially if performed frequently and/or over large data blocks, it might be more applicable to disable it using the option mentioned above. Then data can only be restored after a reset if it did not change since last CRC recalculation (caused by a write request or after the last successful read request). In case the CRC in uninitialized RAM still is correct, the NVM will not reload the corresponding blocks but instead use the RAM copy.

NVM Block Parameters



1. NvM_SetRamBlockStatus() triggers CRC calculation if configured and marks the RAM block as being changed and valid in order to trigger an NV block update during ECU shut-down
2. The currently used NV block for NVM **datasets** is configured by NvM_SetDataIndex()

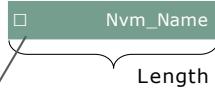
The (re-)calculation of a block's CRC is done asynchronously by the NvM_MainFunction(). A CRC protected block's CRC value is calculated every time the block shall be written to NV memory. If a block is read from NV memory the CRC value is recalculated and compared to the one that was just read from NV memory.

If NvM_SetRamBlockStatus(TRUE) is called, the calculation of the CRC value over the RAM block's data is also initiated, unless the configuration option 'Calculate RAM CRC' was disabled for this block. The purpose of requesting recalculation of the RAM CRC with every call to NvM_SetRamBlockStatus is to provide the possibility to reuse the RAM data even if a soft reset (short power loss, watchdog reset) occurs.

Since CRC calculation is quite time consuming, especially if performed frequently and/or over large data blocks, it might be more applicable to disable it using the option mentioned above. Then data can only be restored after a reset if it did not change since last CRC recalculation (caused by a write request or after the last successful read request). In case the CRC in uninitialized RAM still is correct, the NVM will not reload the corresponding blocks but instead use the RAM copy.

FEE Block Parameters

Fee_Name



Chunk Block Count:

- Write Cycles:
- No effect on code
- Relevant for **FEE Optimization Assistant**
- The chunk size should be increased if a higher number of write cycles is expected.

1, 3, 7, 15, 31, 63

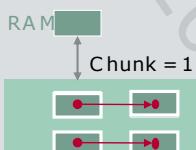
High Priority: this block must be written as fast as possible (e.g. crash data)

Organization of data blocks in "chunks" of configurable size (number of data block instances allocated at one time)

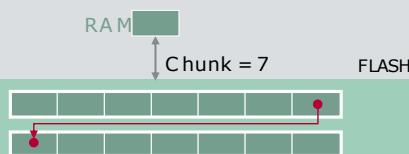
Chunks of one block form a linked list throughout the Flash memory sector

Binary search algorithm used within a chunk to increase performance

NVM Block: Redundant

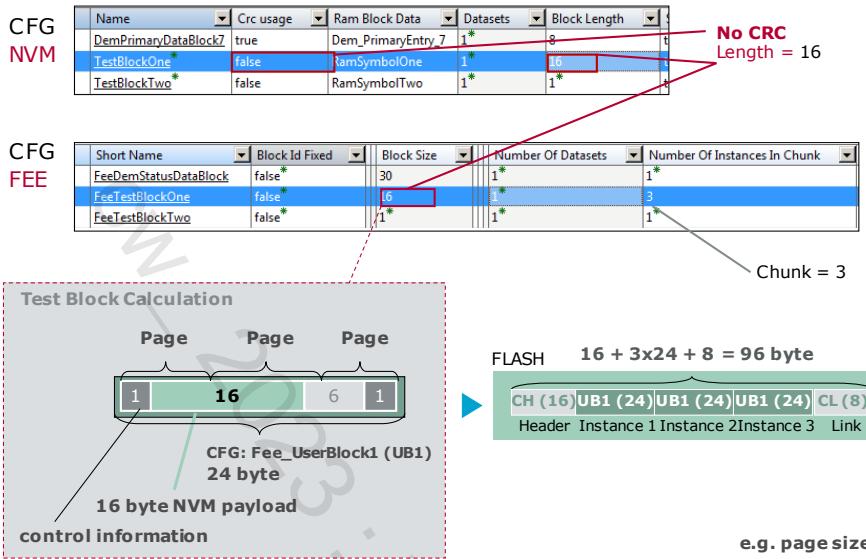


NVM Block: Native



In MICROSAR FEE, the chunk size must always be a number equal to $2^n - 1$ because the FEE performs a binary search inside one chunk to retrieve the most recent data instance.

Relation between NV Block and FEE Block



Relation between NV Block and FEE Block

i

Since version 3.02.00 of MICROSAR NVM, the BSW module provides its own storage for the CRC value of RAM blocks. AUTOSAR originally required that the application has to include memory for the CRC with its own variable. This caused frequent problems as the additional RAM had not been provided by the application.

The chunk header (CH), chunk instance (CI), chunk link (CL) have to be aligned to whole pages individually. This does not cause any problems as long as the page size is smaller than and is a factor of the CH, CI, and CL sizes. If the page size is significantly greater than the CH, CI, and CL sizes, then there will be a lot of padding bytes, e.g., Infineon Tricore has a page size of 128 bytes. For the chunk link there would be 120 bytes of padding.

Rationale: The chunk header (CH) or chunk link (CL), as well as the chunk instance (CI) has to be written individually and atomically. This requires the usage of multiples of the pagesize for these individual writing operations for these parts.

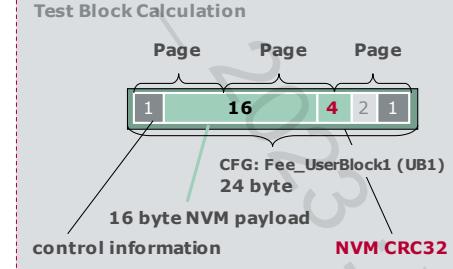
Relation between NV Block and FEE Block

CFG NVM	Name	Crc usage	Ram Block Data	Datasets	Block Length	
	DemPrimaryDataBlock7	true	Dem_PrimaryEntry_7	1*	8	
	TestBlockOne	true	RamSymbolOne	1*	16	
	TestBlockTwo	false	RamSymbolTwo	1*	1*	

CRC32
Length = 16 + 4
= 20

CFG FEE	Short Name	Block Id Fixed	Block Size	Number Of Datasets	Number Of Instances In Chunk
	FeeDemStatusDataBlock	false*	30	1*	1*
	FeeTestBlockOne	false*	20	1*	3
	FeeTestBlockTwo	false*	1*	1*	1

Chunk = 3



FLASH $16 + 3 \times 24 + 8 = 96$ byte

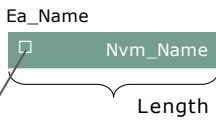
Header Instance 1 Instance 2 Instance 3 Link

e.g. page size = 8



Since version 3.02.00 of MICROSAR NVM, the BSW module provides its own storage for the CRC value of RAM blocks. AUTOSAR originally required that the application has to include memory for the CRC with its own variable. This caused frequent problems as the additional RAM had not been provided by the application.

EA Block Parameters



Write Cycles:

- ▶ How often will this block be written during life time of the system

Verify Writing:

- ▶ Writing to EEPROM is verified or not
- ▶ Recommended for sensitive data

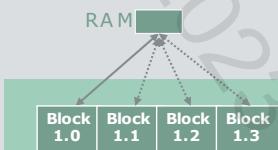
High Priority: this block must be written as fast as possible (e.g. crash data)

Information about how often memory can be rewritten

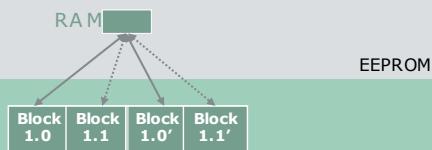
Physical write cycle value is basis for calculation of necessary instances of the EA block in the EEPROM

At runtime these blocks are written alternately

NVM Block: Native

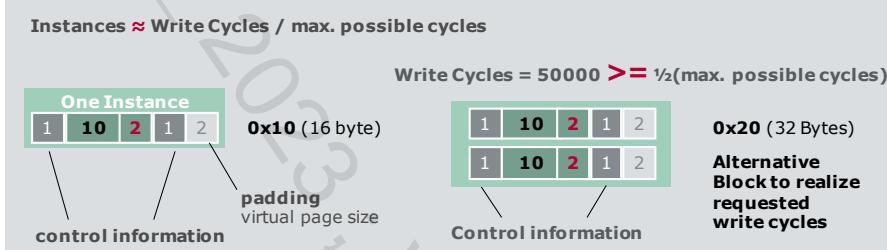


NVM Block: Redundant



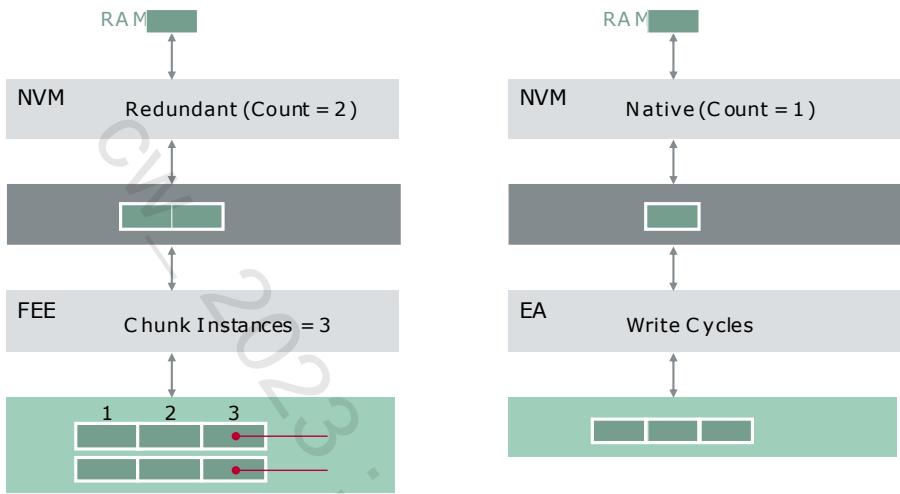
Relation between NV Block and EA Block

CFG	NVM	EA																																																																								
	<table border="1"> <thead> <tr> <th>#</th><th>Name</th><th>Device</th><th>Mngmt Type</th><th>CRC Type</th><th>RAM CRC</th><th>Prio</th><th>Length</th><th>Count</th></tr> </thead> <tbody> <tr> <td>2</td><td>TestBlockTwo</td><td>Ea</td><td>NATIVE</td><td>CRC16</td><td><input checked="" type="checkbox"/></td><td>127</td><td>10</td><td>1</td></tr> </tbody> </table>	#	Name	Device	Mngmt Type	CRC Type	RAM CRC	Prio	Length	Count	2	TestBlockTwo	Ea	NATIVE	CRC16	<input checked="" type="checkbox"/>	127	10	1	<p>Length = 10</p> <p>CRC16 Length = 10 + 2 = 12</p>																																																						
#	Name	Device	Mngmt Type	CRC Type	RAM CRC	Prio	Length	Count																																																																		
2	TestBlockTwo	Ea	NATIVE	CRC16	<input checked="" type="checkbox"/>	127	10	1																																																																		
NVM	<table border="1"> <thead> <tr> <th colspan="9">Calculate</th> </tr> <tr> <th>NvM Block...</th><th>Ea BlockName</th><th>Block Num</th><th>Block Addr</th><th>Datasets</th><th>Size</th><th>Write Cycles</th><th>High Prio</th><th>Verify Writing</th></tr> </thead> <tbody> <tr> <td>TestBlock1Two</td><td>Ea_UserBlock2</td><td>0x02</td><td>0x10</td><td>1</td><td>12</td><td>10000</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr> <td></td><td>Ea_TestBlock</td><td>0x04</td><td>0x10</td><td>4</td><td>10000</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> </tbody> </table>	Calculate									NvM Block...	Ea BlockName	Block Num	Block Addr	Datasets	Size	Write Cycles	High Prio	Verify Writing	TestBlock1Two	Ea_UserBlock2	0x02	0x10	1	12	10000	<input type="checkbox"/>	<input type="checkbox"/>		Ea_TestBlock	0x04	0x10	4	10000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<table border="1"> <thead> <tr> <th colspan="9">Calculate</th> </tr> <tr> <th>NvM Block...</th><th>Ea BlockName</th><th>Block Num</th><th>Block Addr</th><th>Datasets</th><th>Size</th><th>Write Cycles</th><th>High Prio</th><th>Verify Writing</th></tr> </thead> <tbody> <tr> <td>TestBlock1Two</td><td>Ea_UserBlock2</td><td>0x02</td><td>0x00</td><td>1</td><td>12</td><td>50000</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> <tr> <td></td><td>Ea_TestBlock</td><td>0x04</td><td>0x20</td><td>1</td><td>4</td><td>10000</td><td><input type="checkbox"/></td><td><input type="checkbox"/></td></tr> </tbody> </table>	Calculate									NvM Block...	Ea BlockName	Block Num	Block Addr	Datasets	Size	Write Cycles	High Prio	Verify Writing	TestBlock1Two	Ea_UserBlock2	0x02	0x00	1	12	50000	<input type="checkbox"/>	<input type="checkbox"/>		Ea_TestBlock	0x04	0x20	1	4	10000	<input type="checkbox"/>	<input type="checkbox"/>
Calculate																																																																										
NvM Block...	Ea BlockName	Block Num	Block Addr	Datasets	Size	Write Cycles	High Prio	Verify Writing																																																																		
TestBlock1Two	Ea_UserBlock2	0x02	0x10	1	12	10000	<input type="checkbox"/>	<input type="checkbox"/>																																																																		
	Ea_TestBlock	0x04	0x10	4	10000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																																																																		
Calculate																																																																										
NvM Block...	Ea BlockName	Block Num	Block Addr	Datasets	Size	Write Cycles	High Prio	Verify Writing																																																																		
TestBlock1Two	Ea_UserBlock2	0x02	0x00	1	12	50000	<input type="checkbox"/>	<input type="checkbox"/>																																																																		
	Ea_TestBlock	0x04	0x20	1	4	10000	<input type="checkbox"/>	<input type="checkbox"/>																																																																		



Remark: Screenshots not taken from most recent tool (legacy toolchain GENy + Configurator 4)

Summary



Agenda

Principles of Memory Technologies

The MICROSAR.MEM solution

Memory Handling

► Configuration

NvBlockSwComponents

Service Mapping

Coming up next

NVM: Fundamental Settings

Memory Blocks

The screenshot shows the configuration dialog for a memory block named 'TestBlockOne'. Key settings include:

- Name:** TestBlockOne
- Management Type:** NVM_BLOCK_NATIVE
- Ram Block Data:** RamSymbolOne
- Block Length:** 16
- Number Of Write Cycles:** 10000
- Service Ports:** A section for generating service port descriptions.
- RAM Block:** Fields for name, NULL_PTR, and optimization parameters.
- ReadAll:** A section indicating the block is loaded by NvM_ReadAll at startup.

Service Ports

- Generate a service port description

RAM Block

- Enter Name for RAM Block in Application
- Enter NULL_PTR if no permanent RAM block is required
- E.g., name of the Rte Per-Instance Memory

ReadAll

- Block loaded by NvM_ReadAll at startup

33

Diagnostic data does not need to be marked as ReadAll. It can be read on request to reduce ECU startup time.

The feature “Resistant against changed configuration (Res)” works based on the configuration ID value.

If this ID is increased each time the configuration (application) is changed, the NVM can detect configuration changes.

Then, blocks containing the “Res” bit can be retained in the application, provided that the NV Block structure has not changed.

This is intended to conserve NV data like odometer counters, diagnostic data, or seat memory settings throughout the life cycle in the field, without changing due to software updates performed by service.



NVM: Fundamental Settings



Diagnostic data does not need to be marked as ReadAll. It can be read on request to reduce ECU startup time.

The feature "Resistant against changed configuration (Res)" works based on the configuration ID value.

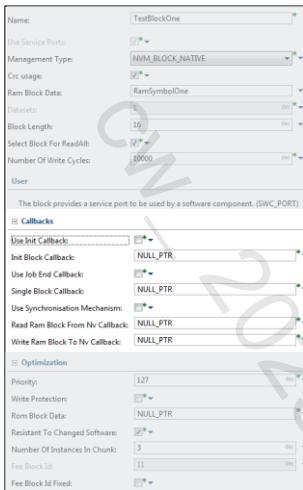
If this ID is increased each time the configuration (application) is changed, the NVM can detect configuration changes.

Then, blocks containing the "Res" bit can be retained in the application, provided that the NV Block structure has not changed.

This is intended to conserve NV data like odometer counters, diagnostic data, or seat memory settings throughout the life cycle in the field, without changing due to software updates performed by service.

NVM: Callback configuration

Further Settings



Init Block Callback

- ▶ Use configured init callback to provide default data
- ▶ Callback is called if no "ROM Block" is configured to initialize this block (NULL_PTR)
- ▶ NULL_PTR: no callback function to be called

Use Job End Callback

- ▶ Use the configured callback for single block job end notification, either directly or via Service Port (RTE) depending on the 'Use Service Port' setting

Explicit Synchronization

- ▶ RAM mirror in the NVM module. The data is transferred to/from the application in both directions via callback routines called by the NVM module.

NVM: Optimizations

Further Settings

The screenshot shows the 'NVM' configuration dialog with the following settings:

- Name:** TestBlockOne
- Management Type:** NVM_BLOCK_NATIVE
- Crc usage:** Enabled
- Ram Block Data:** RamSymbolOne
- Block Length:** 16
- Select Block For ReadAll:** Enabled
- Number Of Write Cycles:** 10000
- User:** None
- Callbacks:**
 - Init Block Callback: NULL_PTR
 - Use Job End Callback: Enabled
 - Single Block Callback: NULL_PTR
- Use Synchronisation Mechanism:**
 - Read Ram Block From Nv Callback: NULL_PTR
 - Write Ram Block To Nv Callback: NULL_PTR
- Optimization:**
 - Priority: 127
 - Write Protection: Enabled
 - Rom Block Data: NULL_PTR
 - Resistant To Changed Software: Enabled
 - Number Of Instances In Chunk: 3
 - Free Block Id: 11
 - Free Block Id Fixed: Enabled

Priority

- ▶ Priority in the Job Queue

Write Protection

- ▶ Software Write Protection
- ▶ Write protection can be removed by API

ROM Block Data

- ▶ Start Address (symbol name) of ROM Block data
- ▶ Name of the RTE calibration data

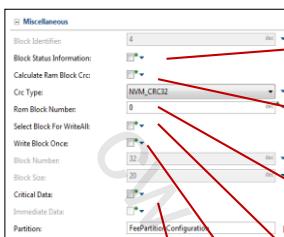
Res (Only for ReadAll)

- ▶ Resistant against changed configuration
- ▶ Preserve old user settings



NVM: Miscellaneous

Further Settings



Block Status Info

- ▶ Defines whether BswM is informed about the current status of this block

Ram Block CRC

- ▶ Calling Nvm_SetRamBlockStatus retriggers RAM Block calculation

Rom Block Number

- ▶ Defines the number of multiple ROM blocks in a contiguous area according to the given block management type.

Write All

- ▶ Write on Nvm_WriteAll call
- ▶ Only write this block once

Critical Data / Immediate Data

- ▶ Handle data loss in exceptional situations (especially Sector Overflow)
TRUE: always keep most recent data (frequently written data is not critical)
- ▶ Immediate Data marks high priority data for a different job queue



Agenda

Principles of Memory Technologies

The MICROSAR.MEM solution

Memory Handling

Configuration

► **NvBlockSwComponents**

Service Mapping

Coming up next

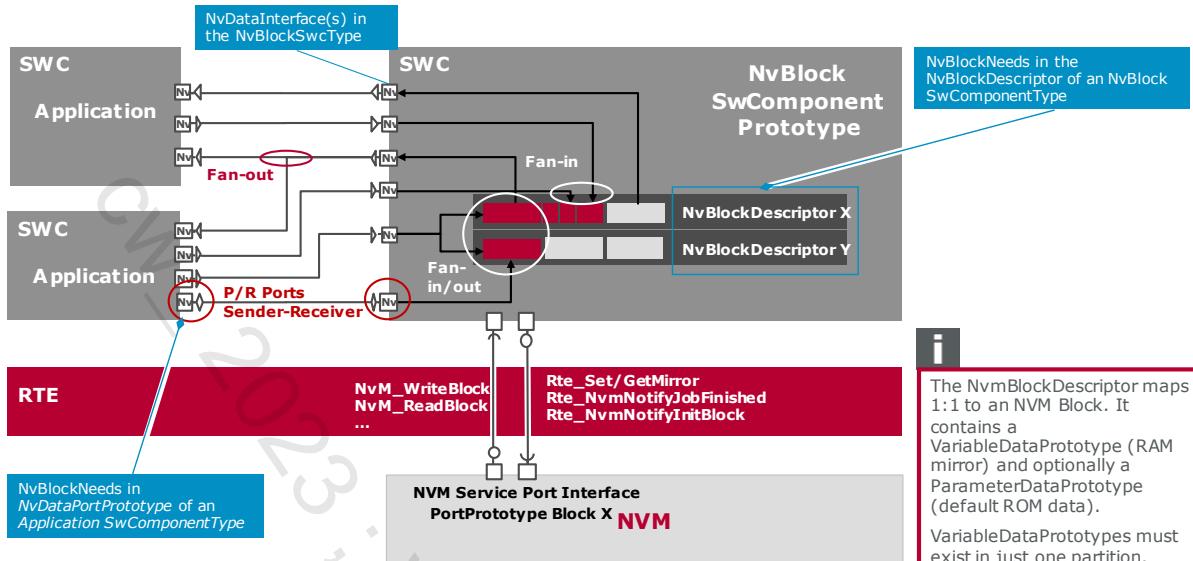
Introduction to NvBlockSwComponents

- ▶ AUTOSAR 3.x: NVRAM handling is block oriented
 - ▶ Management at the NV block level is not sufficient especially if small data has to be stored individually
 - ▶ Inefficient usage of paged flash memories
 - ▶ Distribution of non-volatile data between SWCs is difficult to handle
 - ▶ PIMs or module-local static variables are private members of an SWC
 - ▶ Access over PIMs is not protected against concurrent access
- ▶ AUTOSAR 4.x addresses this situation by introducing optional **NvBlockSwComponents**

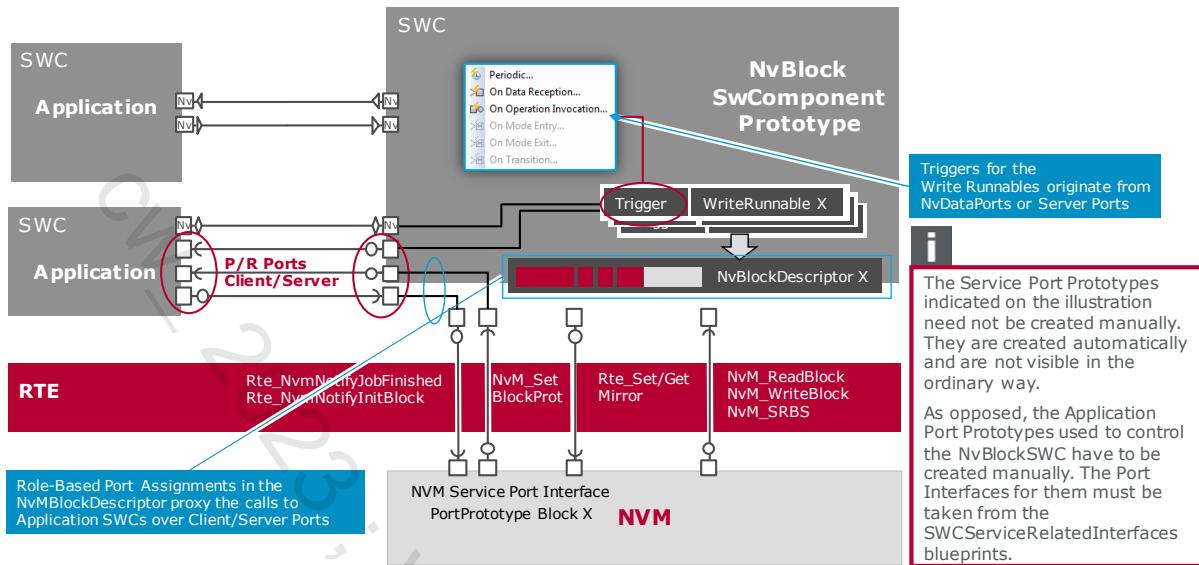
Introduction to NvBlockSwComponents

- ▶ S/R oriented NVRAM handling
- ▶ Aggregation of several RAM variables into one NVRAM block (analogous to Signals and Signal Groups in Com)
- ▶ "Fan-in" and "fan-out" of data elements to and from NVRAM blocks
 - ▶ Additional RAM is required because an extra buffer stage is required for handling the "fan-in"
 - ▶ "Fan-out": overall RAM consumption may be lower because the same data elements can be shared in the ECU with this feature (like calibration data)
 - ▶ NvData elements can fan into/out of the same (shared) NVRAM Block Element
- ▶ Flexible NvData distribution across multiple SWCs
- ▶ Protection against concurrent access by RTE

Introduction to NvBlockSwComponents



Control Flow



NV Data Ports and their relation to NV Block Descriptors

- ▶ In AUTOSAR, the NVData elements can already carry contributions to the Service Needs
- ▶ If there are several NVData elements which are combined in to one NvBlockDescriptor, there have to be arbitration rules on how the NVBlock in the NVM must be configured.
- ▶ Example: NVM Parameters
 1. Writing frequency
 2. RAM Block CRC


Legend:

'=': recommended to be identical

'=MAX': recommended to be the maximum of all existing values

'OR': recommended to be set TRUE if at least one of existing values of NV data Ports is set TRUE.

'!=': may be set differently among different NV data Ports

The highest writing frequency among all NV data ports determines the configuration value for the NvBlock in the NVM

NvBlockNeeds attrib.	@NV data Ports	@NvBlockDescr.
Readonly	=	=
Reliability	!=	=MAX
Resistant to change	=	=
Restore at startup	=	=
Store at shutdown	=	=
writeOnlyOnce	=	=
writingFrequency	!=	=MAX
writingPriority	!=	=MAX
writeVerification	!=	OR
calcRamBlockCrc	!=	OR
checkStaticBlockId	!=	OR
ramBlockStatusControl	!=	OR

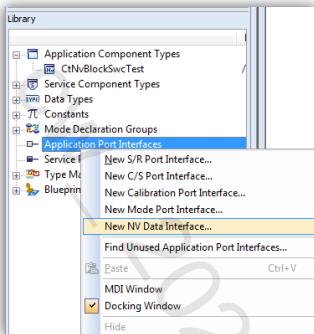
If any NV data Port requires a CRC protection, the whole NvBlockDescriptor has a service need / NvBlock in the NVM has to support CRC.

Implementation of NvBlockSwComponents

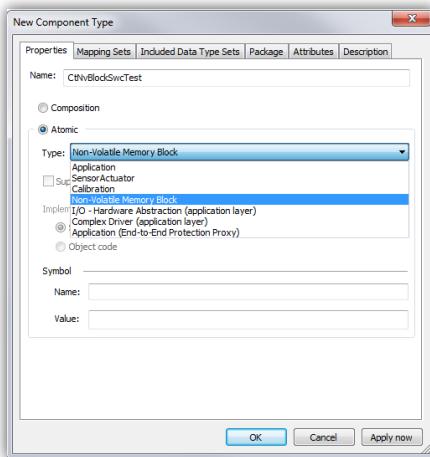
- ▶ SWC type with interface to NVM
 - ▶ Support for NvMAdmin, NvMService, NvMJobFinished, NvMNotifyInitBlock
 - ▶ The NVM API can be operated through the NvMBlockSWC directly
 - ▶ Client/Server and ModeSwitch interfaces
 - ▶ Multiple NvBlock to port mappings for dual-sided RAM block access
 - ▶ Explicit modelling of different writing strategies
 - ▶ Periodic, On Data Reception, On Operation Invocation
 - ▶ On Mode Entry, On Mode Exit, On Transition

Implementation of NvBlockSwComponents

- SWC type Equipped with P/R Nv Data Interfaces



DaVinci Developer: NV Data can be described using NV Data Interfaces. This is similar to the S/R communication paradigm.



DaVinci Developer: A new Nv Block SWC can be created by choosing Non-Volatile Memory Block as the Atomic SWC Type. This SWC Type is created automatically and only treats the NV Data and has no separate functionality.



NV Block Data Mapping

The NvBlockDataMapping specifies the mapping of DataElements of a P/R PortPrototype to DataElements inside the RAM block.

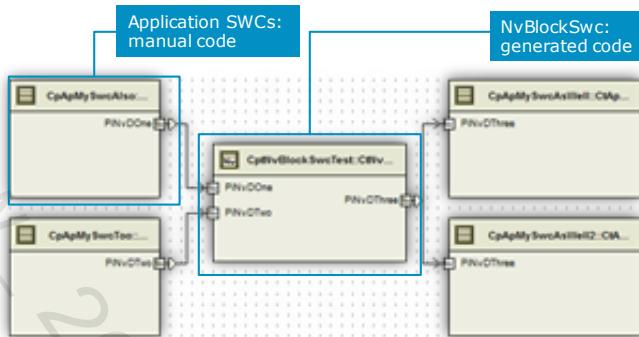
Multiple mappings to the same RAM block element are possible.

When data is written to a RAM Block, the corresponding data comes from an R-Port that is connected to an application component.

If data is read from the RAM block this data is sent through a P-Port to the connected components.

Implementation of NvBlockSwComponents

Example Software Design



DaVinci Developer: The Software Design shows NV Data Port Prototypes using the "Nv" label in the port pictogram. Arbitrary SWCs can use the NvBlockSwComponent, which manages the NVRAM Blocks.

NV Block Data Mapping

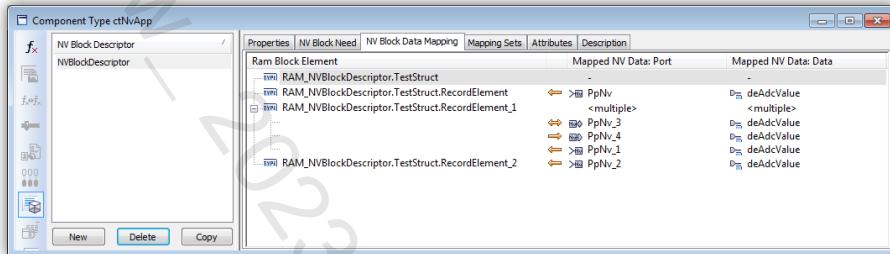
- ▶ Internally, the SWC must have at least one NvBlock Descriptor
- ▶ The NvBlock Descriptor is used to describe the RAM mirror
 - ▶ Fan-in is only possible if the NvBlock descriptors is a complex data type
 - ▶ Fan-out is always possible since the distribution of a data element will be done by the RTE Provider (Sender) Port

Ram Block Element	Read NV Data: Port	Read NV Data: Data Element	Written NV Data: Port	Written NV Data: Data Element
RAM_NvBlockDescriptor.TestStruct	-	-	-	-
RAM_NvBlockDescriptor.TestStruct.RecordElement_0	-	-	PIVOTOne	DByte
RAM_NvBlockDescriptor.TestStruct.RecordElement_1	-	-	PIVOTTwo	DByte
RAM_NvBlockDescriptor.TestStruct.RecordElement_2	PIVOTThree	DByte	-	-

DaVinci Developer: NV Block Data Mapping: NVBlockDescriptors are the link between NvData Ports and the NVRAM Block in NVM. If a fan-in is required, the NVBlockDescriptor must have a complex data type (MICROSAR4 Release 9 shown).

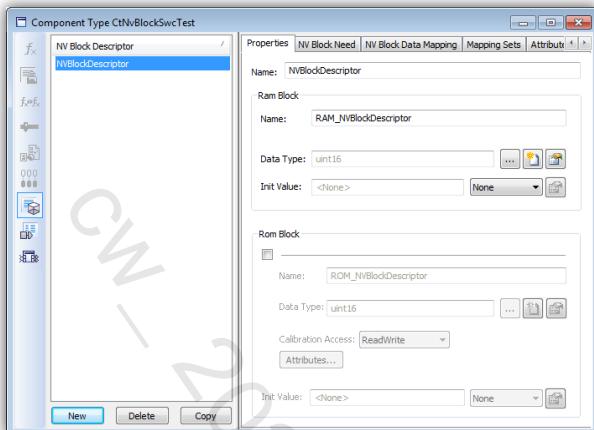
NV Block Data Mapping

- ▶ Specifies the mapping of DataElements of a P/R PortPrototype to DataElements inside the RAM block
- ▶ Multiple mappings to the same RAM block element are possible
- ▶ When data is written to a RAM Block, the corresponding data comes from an R-Port that is connected to an application component
- ▶ If data is read from the RAM block this data is sent through a P-Port to the connected components



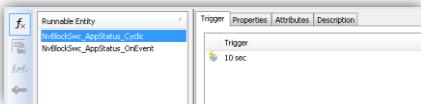
DaVinci Developer: NV Block Data Mapping: One RAM block element can be assigned to several Nv Data Ports / Elements. You can recognize that from the "<multiple>" string in the table. Please also note that S/R and SR Ports are supported (MICROSAR4 Release 13 shown).

Properties of the NV Block Descriptor



DaVinci Developer: The NV Block Descriptor defines the data type(s) which may be used for a NVBlock Data Mapping. Here only an atomic data type 'uint16' is used. But we may also use a record (C struct) here to allow for efficient NVRAM Block configurations. The NV Block Descriptor has a 1:1 mapping to NVRAM Block.

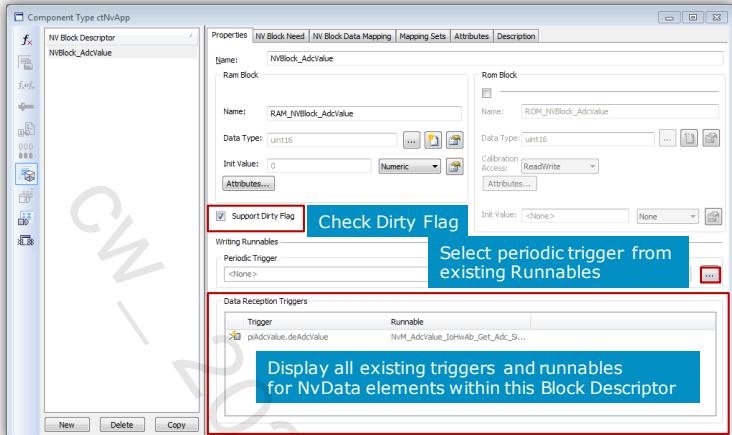
Write Runnable definitions



Definition of **Runnables**

- Periodic trigger
- On Data Reception trigger

Properties of the NV Block Descriptor

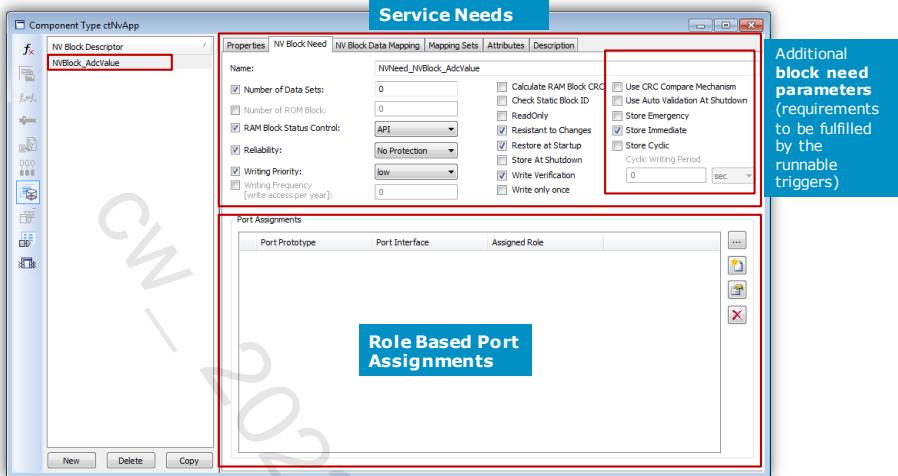


DaVinci Developer: The linkage between the Write Runnables and the NvBlockDescriptors is established by the data receive triggers of NvData Elements. These Elements are mapped onto NvMBlock Descriptors. The Triggers for the Write Runnables are displayed. Additionally, it is possible to define a cyclic writing activity using a periodic trigger (of a Writing Runnable)



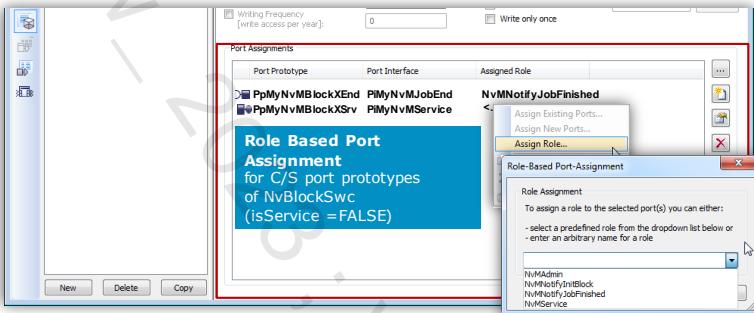
Support Dirty Flag
Specifies whether the RTE shall control the calling of NvM functions for writing and/or status control for potentially modified RAM Blocks

Service Needs (NvBlockNeeds)



Role-Based Port Assignments (part of Service Needs)

- ▶ Per NvBlockDescriptor, the Port Assignments define which Client/Server port(s) of the NvBlockSwComponentType are associated with what kind of service or notification
- ▶ For notifications, the RTE provides one common callback function for each individual kind of notification defined by the "role"
- ▶ Role-Based PortAssignments
 - ▶ Client Ports: NvMNotifyJobFinished, NvMNotifyInitBlock
 - ▶ Server Ports: NvMAdmin, NvMService



Role-Based Port Assignments (part of Service Needs)

- ▶ Support of C/S Interfaces for Nv Block SWCs
 - ▶ In addition to NV-Ports, there are
 - > Server Ports
 - > Client Ports
 - > Mode R-Ports
 - ▶ Only **application port interfaces** can be selected
- ▶ What is the exact mapping between a C/S Port and a Port Assignment?
 - ▶ For example, how does the RTE know which function of the **NvMService Port** maps to the **custom C/S Port**?
 - ▶ Use compatible port descriptions (blueprints)
 - > Available at www.autosar.org
 - > AUTOSAR_MOD_SWCSERVICERELATEDINTERFACES_Blueprint.arxml
AUTOSAR_MOD_GeneralBlueprints.zip)
 - ▶ Also supported in CFG5
 - ▶ Display the internal behavior of NvBlock SWCs

Agenda

Principles of Memory Technologies

The MICROSAR.MEM solution

Memory Handling

Configuration

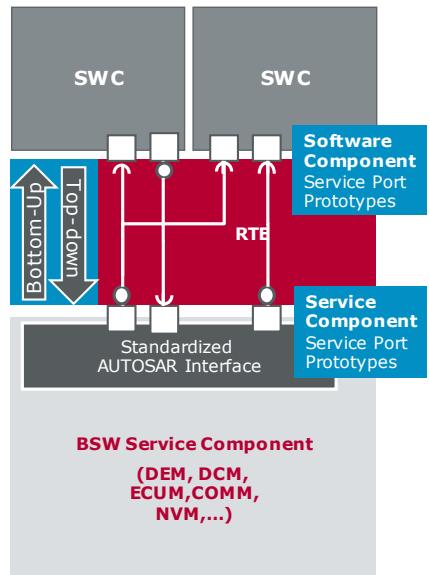
NvBlockSwComponents

► **Service Mapping**

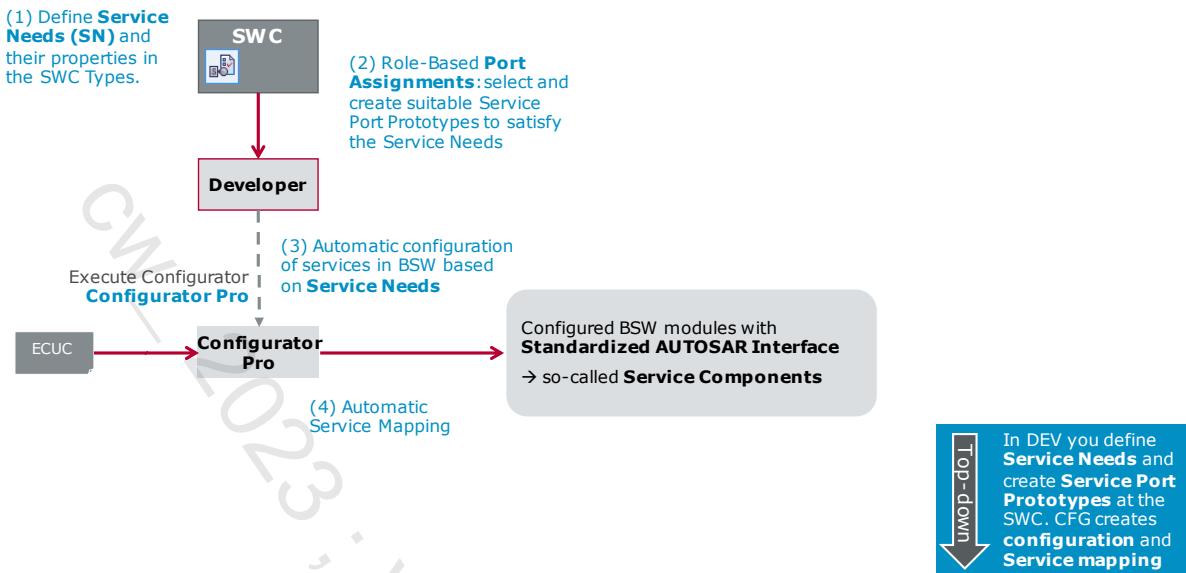
Coming up next

Service Mapping

- ▶ Assigning Service Ports of Software Components to Service Ports of Service Components or vice versa
- ▶ Bottom-up service configuration
 1. Configure a service module in the basic software
 2. Assign the respective service port prototypes to the SWCs where appropriate
- ▶ Top-down service configuration
 1. Use Service Needs
 - > Document at the SWC level what is required from the BSW as a service
 2. Derive a service module configuration based on these requirements
 3. Configure the basic software service module in detail

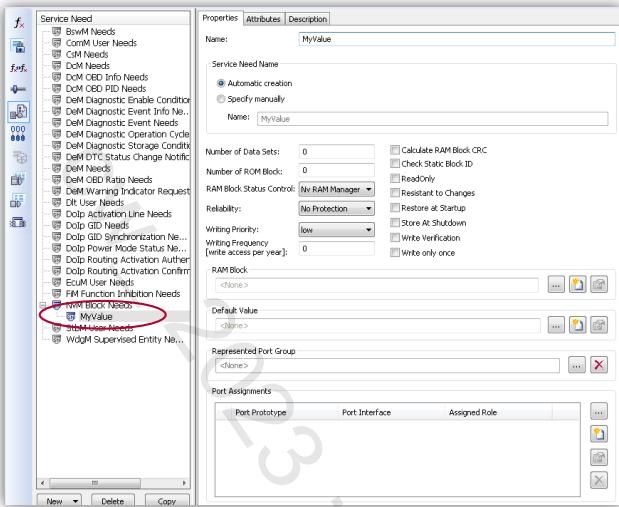


Service Component Configuration (one-stop solution)



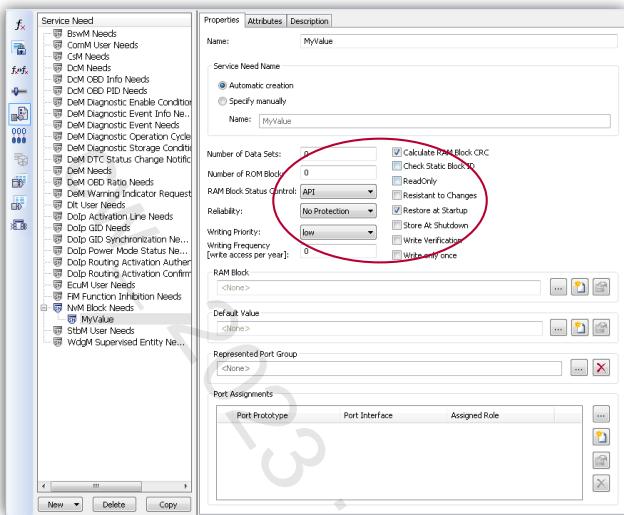
Top-down service configuration (NvM)

- DEV: Create a new NvM Block Service Need



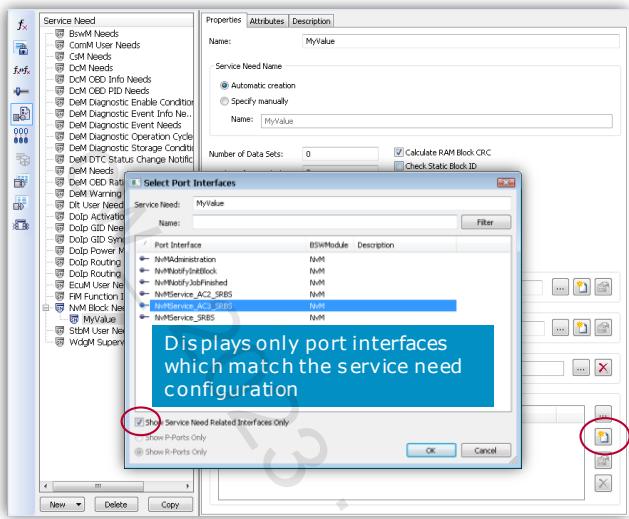
Top-down service configuration (NvM)

- DEV: Configure the service need properties



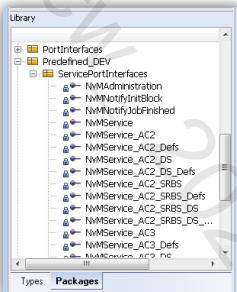
Top-down service configuration (NvM)

- DEV: Create service ports matching to the service need



Top-down service configuration (NvM)

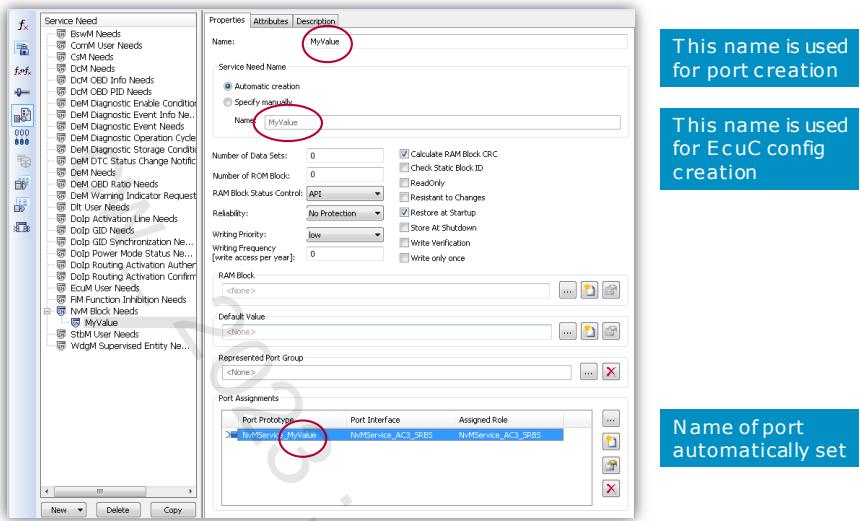
- ▶ Finding suitable service ports can be done top-down without NVM being configured
- ▶ DEV: **Predefined Service Port Interfaces**
 - ▶ File *ServicePortInterfaces_4.1.2.arxml* in tool installation directory
 - ▶ New workspace: automatically imported
 - ▶ Existing workspace: manual import required



Developer: Predefined
Service Port Interfaces after
Import

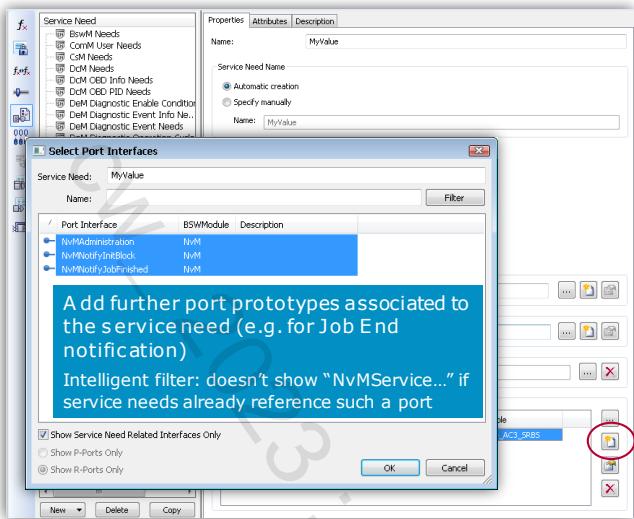
Top-down service configuration (NvM)

- DEV: Create service ports matching the service need



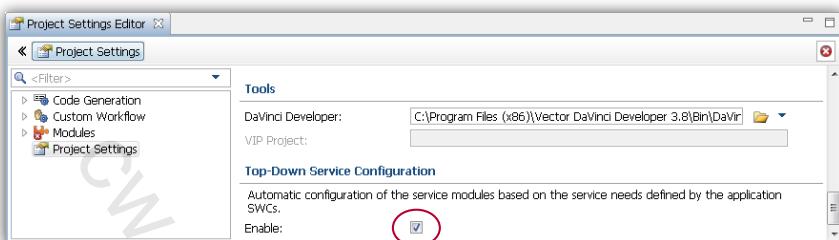
Top-down service configuration (NvM)

- DEV: Create service ports matching the service need

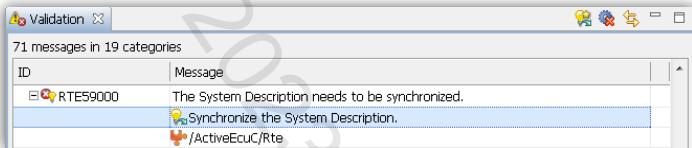


Top-down service configuration (NvM)

- ▶ CFG5: Enable top-down service configuration

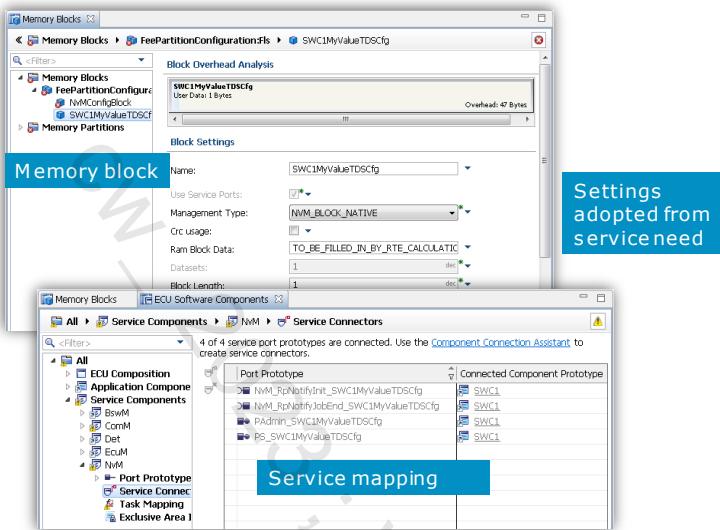


- ▶ CFG5: Run System Description synchronization



Top-down service configuration (NvM)

► CFG5: Result

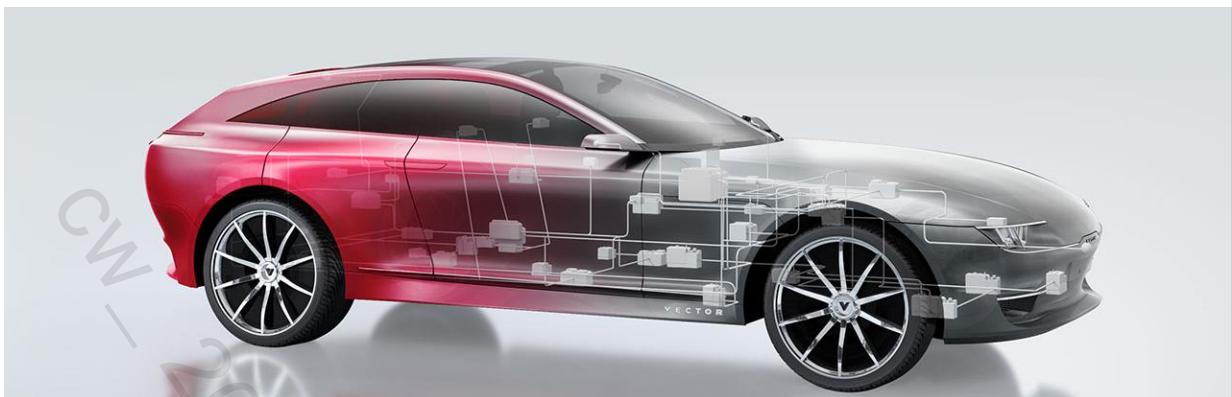


Top-down service configuration (NvM)

- ▶ DEV: Consistency check of ECU global NvM settings
 - ▶ Ensure same setting for "RAM Block Status Control"
 - ▶ Ensure that all "NvMServices" ports have the same NvM API config class

Memory Abstraction

Diagnostics



AUTOSAR in Practice

Memory Abstraction

V28 | 2023-03-28

Agenda

- ▶ **Exercise 5 - Memory Abstraction**

Coming up next

CW_2023_VH_Autosar CP Training_EN

Overview

5

Memory Abstraction

- 1** Create NvBlock Needs, Service Ports, and Port Assignments
- 2** Configure BSW and create NV Blocks with Configurator Pro
- 3** Program Runnables and test it

Overview

Memory Abstraction

5

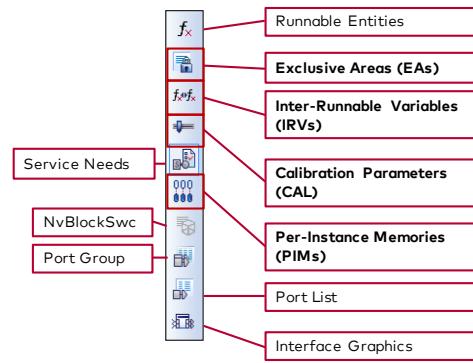
Data that has to be restored after a reset or power off has to be handled by the memory stack.

An odometer value and a door open counter will be written to non-volatile memory. During startup after a reset or power off, the last stored values are maintained.

In this exercise we will perform the writing of non-volatile data.

Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

- ▶ "Configure SWCs" using `MyECU.dpa` file in folder `\E5_Memory\`
- ▶ Open `CtApMySwc`
 - ▶ Add Exclusive Areas (EA)
 - > Name: `ExArLightOnOffCounter`
 - > Access Mode "Can Enter" for `RCtApMySwcCode`, `RCTApMySwcInit`
 - > Name: `ExArOdometer`
 - > Access Mode "Can Enter" for `RCtApMySwcCode`
 - ▶ Add an Inter-Runnable Variable (IRV)
 - > Name: `IrVDoorStateOld`
 - > Data Type: `AdtDoorState`
 - > Init Value Reference: `CDoorClosed`
 - > Access Mode "Read / Write" for `RCtApMySwcCode`
 - ▶ Add a Calibration Parameter (CAL)
 - > Name: `CalOdometer`
 - > Data Type: `AdtOdometer`
 - > Init Value Reference: `COdometerDefault`
 - ▶ Add Per-Instance Memories (PIM)
 - > Name: `PimOdometer`
 - > Data Type: `AdtOdometer`
 - > Name: `PimLightOnOffCounter`
 - > Data Type: `AdtLightOnOffCounter`



Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

i

The **Inter-Runnable Variable (IRV)** is private data of the SWC Prototype. It is enforced by the RTE that the access to this variable is atomic (on task level). The RTE knows which Runnables need what access and can protect against data corruption of the **IRV** in case of a race condition. The Inter-Runnable Variable cannot be used for Memory Mapping (i.e., NvM access).

Inter-Runnable Variable Access:	
Runnable	Access Mode
f _x RCTApMySwcCode	Read/Write
f _x RCTApMySwcComM_ModeChange_FULL_COMM_Entry	No Access
f _x RCTApMySwcComM_ModeChange_FULL_COMM_Exit	No Access
f _x RCTApMySwcInit	No Access
f _x RCTApMySwcPostRunCode	No Access

An **Exclusive Area (EA)** is the AUTOSAR version of a critical section. The sequence of instructions inside an **EA** is executed atomically (on task level). The RTE enforces atomic access inside the **EA**. The RTE knows which Runnables can access or run completely inside the **EA** and can therefore protect against data corruption case by case. The RTE generator analyzes the configuration for this. If a runnable is inside an **EA**, it must not call RTE APIs which access the Port Prototypes of the SWC Prototype, especially if C/S with blocking access or S/R with queued access is used. This may lead to deadlocks.

Exclusive Area Access:	
Runnable	Access Mode
f _x RCTApMySwcCode	/
f _x RCTApMySwcInit	Can Enter
f _x RCTApMySwcComM_ModeChange_FULL_COMM_Entry	Can Enter
f _x RCTApMySwcComM_ModeChange_FULL_COMM_Exit	No Access
f _x RCTApMySwcPostRunCode	No Access

Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments**i**

The **Per-Instance Memory (PIM)** is also private data of the SWC Prototype. The RTE does not enforce atomic access to this variable. Therefore, it is advisable to access the Per-Instance Memory inside an Exclusive Area. The **PIM** can be used for Memory Mapping.

Calibration Parameters (CAL) are ROM constants which can be private read-only memories of the SWC Prototype. However, it is also possible to share the ROM constant among all instances of the SWC Type. **CAL** can be used as default values for non-volatile data.

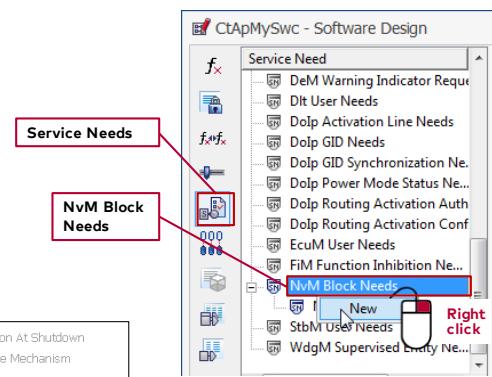
Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

► Add Service Needs → NvM Block Needs

- **SnNvMPimLightOnOffCounter**
 - > RAM Block: PimLightOnOffCounter
- **SnNvMPimOdometer**
 - > RAM Block: PimOdometer
 - > Default value: CalOdometer

► Configure the following properties for each NvM Block Need (let the black boxes unchecked)

<input type="checkbox"/> Number of Data Sets:	0	<input type="checkbox"/> Calculate RAM Block CRC	<input type="checkbox"/> Use Auto Validation At Shutdown
<input type="checkbox"/> Number of ROM Block:	0	<input checked="" type="checkbox"/> Check Static Block ID	<input type="checkbox"/> Use CRC Compare Mechanism
<input type="checkbox"/> RAM Block Status Control:	API	<input type="checkbox"/> Readonly	<input type="checkbox"/> Store Emergency
<input type="checkbox"/> Reliability:	No Protection	<input type="checkbox"/> Resistant to Changes	<input type="checkbox"/> Store Immediate
<input type="checkbox"/> Writing Priority:	low	<input checked="" type="checkbox"/> Restore at Startup	<input type="checkbox"/> Store Cyclic
<input type="checkbox"/> Writing Frequency [write access per year]:	0	<input type="checkbox"/> Store At Shutdown	Cyclic Writing Period 0 sec
		<input type="checkbox"/> Write Verification	<input type="checkbox"/> Write only once



Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

i

The **Service Needs** concept is a very powerful Top-Down approach to specifying requirements contained in the SWCs. These requirements apply in areas like Diagnostics, Memory, IO, etc. and are typically satisfied by connecting the SWC to the BSW (services of the Basic Software will be used).

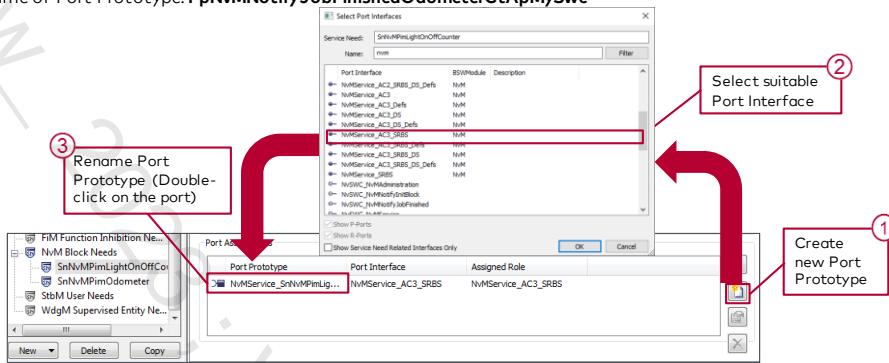
From the workflow it is a typical AUTOSAR approach (Top-Down as opposed to Bottom-Up).

At SWC creation time, the component developer can already foresee a later need for diagnostic data support and persistent memory. Later on the ECU integrator can use this information since the requirements for the Basic Software configuration are already in the SWC description. A powerful tool can also generate the BSW configuration automatically from this information. This approach shall be followed here.

Parameter settings provided inside the NvM Block Needs description will be used for the NvM Block configuration inside DaVinci Configurator, e.g., "Restore at Startup" (Select Block for ReadAll) or "Store At Shutdown" (Select Block for WriteAll).

Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

- ▶ Add **Port Assignments** for the Nv Block Service Needs (See next slide)
- ▶ **SnNvMPimLightOnOffCounter**
 - > Create new Port Prototype, Select PortInterface: NvMService_AC3_SRBS
 - > Rename Port Prototype: **PpNvMLightOnOffCtApMySwc**
- ▶ **SnNvMPimOdometer**
 - > Create new Port Prototype, Select PortInterface: NvMService_AC3_SRBS_Defs
 - > Rename Port Prototype: **PpNvMOdometerCtApMySwc**
 - > Create new Port Prototype, Select PortInterface: NvMNotifyJobFinished
 - > Rename of Port Prototype: **PpNvMNotifyJobFinishedOdometerCtApMySwc**



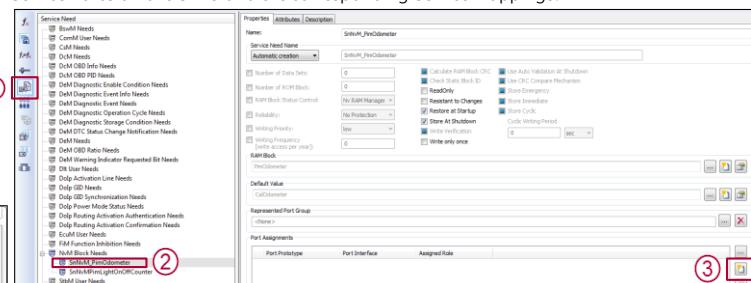
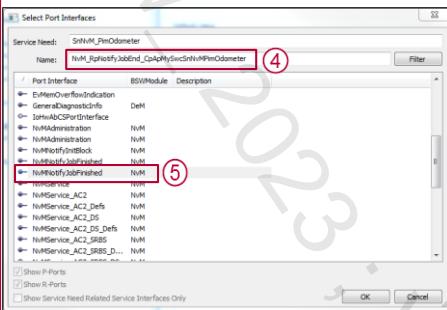
Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments



► **Role-Based Port Assignments** can be used to define on Service Need level how the Service Component shall be connected to your Software Component. Therefore, at SWC creation time, the component developer can already define how the BSW shall be connected via the RTE to the SWC. This will lead to the automatic generation of Service Ports on the SWC and the corresponding Service Mappings.

► Add Port Assignments:

- (1) Open **Service Needs** tab
- (2) Select **NvM Block Needs**
- (3) Open **Select Port Interfaces** dialog
- (4) Set the Port Prototype name
- (5) Choose Port Interface type



What do the interface names NvM_AC3_SRBS, NvM_AC3_SRBS_Defs

NvM: Module Name (Short Name Value Prefix)

AC3: NvM API Class 3 (the API is configurable)

SRBS: SetRamBlockStatus API used

Defs: means there is a ROM Default and an additional operation 'RestoreBlockDefaults' exists.

Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments

- ▶ Configure Runnables of **CtApMySwc**

- ▶ RCtApMySwcCode:

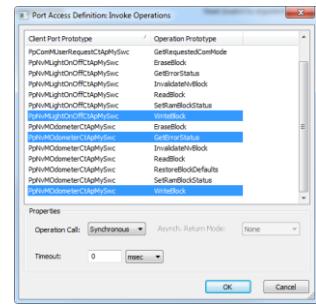
- ▶ Add "Invoke Operations..." Access Points
 - > **PpNvMOdometerCtApMySwc.WriteBlock**
 - > **PpNvMOdometerCtApMySwc.GetErrorStatus**
 - > **PpNvMLightOnOffCtApMySwc.WriteBlock**
- ▶ Add "Write Data" Access Points
 - > **PpDisplayState.DeLightOnOffCounter**
 - > **PpDisplayState.DeOdometerValue**

- ▶ RCtApMySwcInit:

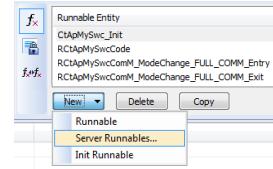
- ▶ Add "Write Data" Access Points
 - > **PpDisplayState.DeLightOnOffCounter**

- ▶ **Create a New Server Runnable** (see right)

- ▶ **PpNvMNotifyJobFinishedOdometerCtApMySwc_JobFinished**
- ▶ Allow this runnable to have a Access Point (write) to **PpDisplayState.DeOdometerWriteRequestPending**
- ▶ Check, save, and close **Developer** workspace



Access Point: Invoke operation

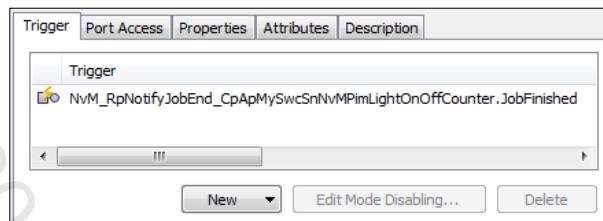


SWC Type view: Here you can create Server Runnables with a Trigger from a Service Port Prototype.

Part 1/3: Create Nv Block Needs, Service Ports and Port Assignments



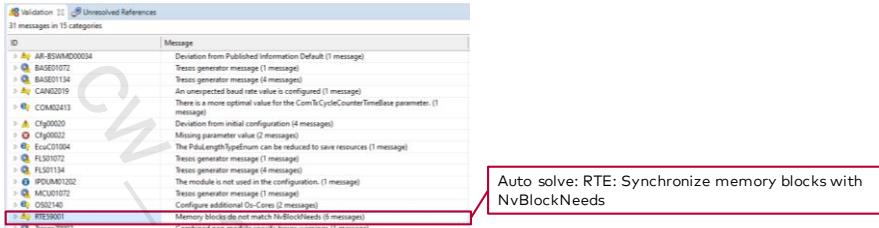
PpNvMNotifyJobFinishedOdometerCtApMySwc_JobFinished is a server Runnable. This Runnable has an "On Operation Invocation" Trigger. That means that it is an implementation of an operation inside a Server Port Prototype. Whenever a client calls the operation using the Rte_Call function in the Server Port Prototype, the Runnable is invoked.



SWC Type: On Operation Invocation Trigger for PpNvMNotifyJobFinishedOdometerCtApMySwc_JobFinished will automatically be created

Part 1/3: Configure BSW and create NV Blocks with Configurator Pro

- ▶ "Configure BSW" using [MyECU.dpa](#) file in folder \E5_Memory\
- ▶ First click on [Synchronize now](#), to fix System Description Reference Inconsistencies.
- ▶ Then auto solve the RTE59001(if it's there)



- ▶ Afterwards synchronize again [Synchronize now](#)

Part 1/3: Configure BSW and create NV Blocks with Configurator Pro

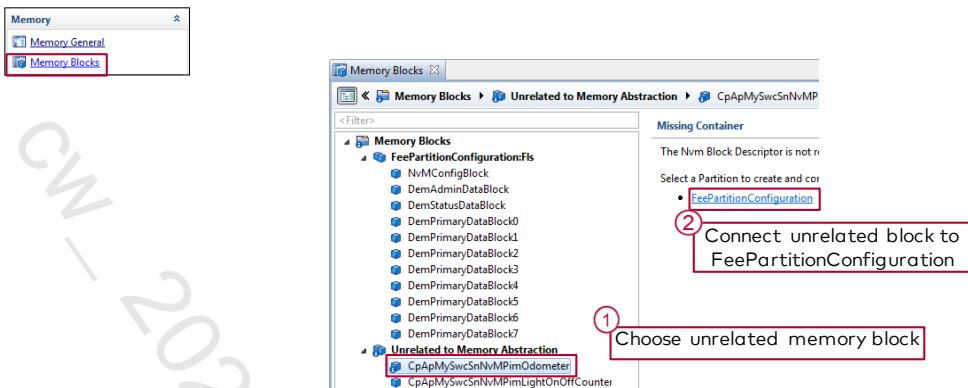
- ▶ Afterwards click on [Synchronize now](#)
- ▶ Two errors "A choice-container must have exactly one child container" pop up (see below)
 - ▶ AR-ECU02067 → **2x Choose solving action**
`/MICROSAR/NvM/NvMBlockDescriptor/NvMTargtBlockReference/NvMFeeRef`
 - ▶ Cause: the creation of memory blocks for PimOdometer and PimLightOnOffCounter need a memory

Validation		Element Usage	Find
25 messages in 6 categories			
ID	Message		
AR-ECUC02067	A choice-container must have exactly one choice child container (2 messages)		
AR-ECUC02067	The choice-container NvMTargtBlockReference must have exactly one choice as child, but has 0 child containers.		
	↳ Choose /MICROSAR/NvM/NvMBlockDescriptor/NvMTargtBlockReference/NvMEARef		
	↳ Choose /MICROSAR/NvM/NvMBlockDescriptor/NvMTargtBlockReference/NvMFeeRef		
AR-ECUC02067	The choice-container NvMTargtBlockReference must have exactly one choice as child, but has 0 child containers.		
	↳ Choose /MICROSAR/NvM/NvMBlockDescriptor/NvMTargtBlockReference/NvMEARef		
	↳ Choose /MICROSAR/NvM/NvMBlockDescriptor/NvMTargtBlockReference/NvMFeeRef		
	↳ /ActiveEcu/CpApMySwcSnNvMPimOdometer/NvMTargtBlockReference		
CANNM02091	Partial Networking Feature is not available. (1 message)		
Cfg00020	Deviation from initial configuration (17 messages)		
ECUC01004	The PduLengthTypeEnum can be reduced to save resources (1 message)		
IPDUM01202	The module is not used in the configuration. (1 message)		
NV/M01037	For blocks with enabled NvMUseServicePorts the initBlockCallback and SingleBlockCallback won't be invoked during		

- ▶ Or see next slide how to solve it manually

Part 2/3: Configure BSW and create NV Blocks with Configurator Pro

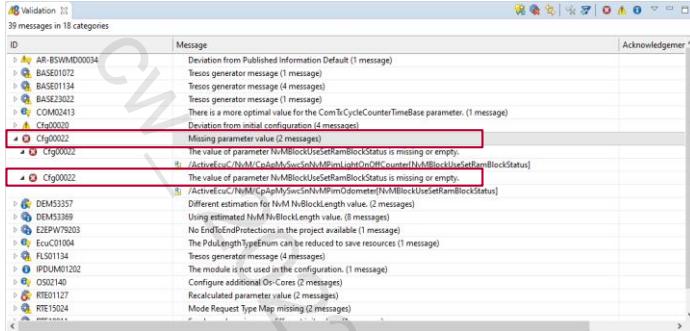
- Add the new Memory Blocks (**CpApMySwcSnNvMPimOdometer** and **CpApMySwcSnNvMPimLightOnOffCounter**) to FeePartitionConfiguration



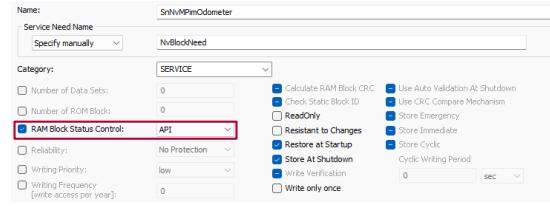
Part 1/3: Configure BSW and create NV Blocks with Configurator Pro

- Error Cfg00022. You can solve it here or in the DaVinci Developer

DaVinci Config. see next slide

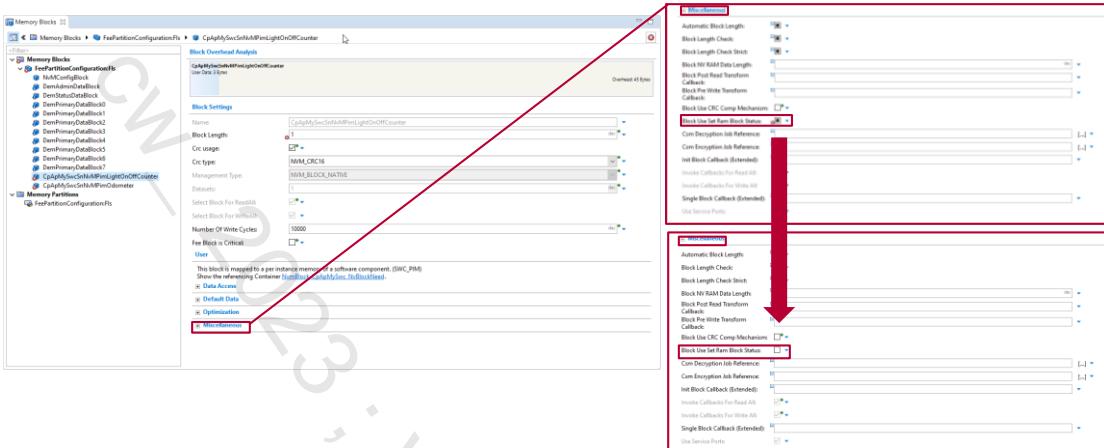


DaVinci Dev.



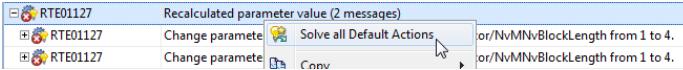
Part 2/3: Configure BSW and create NV Blocks with Configurator Pro

- To handle the error the undefined `NvMBlockUseSetRamBlockStatus` must be set
 - Go to **Memory** → **Memory Blocks** → **FeePartitionConfiguration:Fls**
 - For both `CpApMySwcSnNvMPimLightOnOffCounter` and `CpApMySwcSnNvMPimOdometer` Set the value of `NvMBlockUseSetRamBlockStatus` to **false** (empty checkbox)

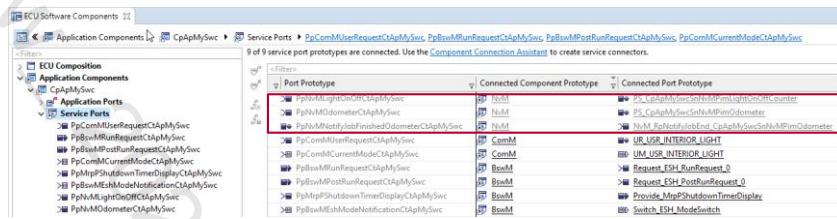


Part 2/3: Configure BSW and create NV Blocks with Configurator Pro

- ▶ Push the validation button 
 - ▶ Result: the RTE recalculated the NvM block length and asks via solving action to change them to the value 4
 - ▶ Right click on RTE01127 and choose "Solve all Default Actions"



- ▶ Go to **Runtime System** → **ECU Software Components** → **Application Components**
 - ▶ Select **CpApMySwc** → **Service Ports** and check that the service mapping between CpApMySwc and the NVM exists



- ▶ Generate the project
 - ▶ Press F7 in order to update the SWC Templates
 - ▶ Close **DaVinci Configurator**

Part 2/3: Configure BSW and create NV Blocks with Configurator Pro

Due to the usage of Port Assignments during the definition of the NvM Block Service Need within the DaVinci Developer, an explicit Service Mapping for SnNvMPimOdometer and SnNvMPimLightOnOffCounter NvM Blocks to the NvM Service Component is not needed. This is automatically done by DaVinci configurator based on the Port Assignments definition.

The NvM Config Block is not part of the NvM Block Service Needs definition. Therefore, an explicit Service Port definition on the CpApMySwc and an explicit Service Mapping is needed.

Part 3/3: Program Runnables and test it

- ▶ Open Visual Studio Project by double-clicking on **MyECU.sln** file in folder **\E5_Memory\...\Solution**
- ▶ Program Runnables in **CtApMySwc** to have the following behavior
 - ▶ In **RctApMySwcInit**
 - > Send the current value of the PIM Light On Off Counter to the Display SWC. Why? See next slide
 - > When accessing the PIM, use Exclusive Area access to make the sequence of instructions atomic
 - ▶ In **RctApMySwcCode**
 - > Each time the light state changes from light off to light on, do the following (use the existing implementation for door open / close handling):
 - > Increment the PIM for Light On/Off Counter (protect the access with an Exclusive Area)
 - > Send the new value to the Display SWC
 - > Request an NvM Write to store the new value
 - ▶ For more detailed information see next slide

Part 3/3: Program Runnables and test it**i**

RCtApMySwcInit: The data of the Light On/Off Counter has been restored by the non-volatile memory stack. This value is forwarded to the display to have a valid value after startup.

Using Exclusive Areas: Within an Exclusive Area no access to the Port Prototypes of the SWCs is allowed! Rte_Receive / Rte_Send API (Queued S/R) or Rte_Calls to Runnables mapped to TASKS especially can cause deadlocks.

Please note that the option "runs in" is also dangerous in this respect. Runnables which are running completely inside an EA may not have any access point to the Port Prototypes of the SWC and have to be used completely "internally" in the SWC. But this condition can be detected by the Configurator tool.

Note: If you have chosen "Can Enter" for the Runnable with respect to the Exclusive Area, the Tool cannot detect if you stick to the rule not to call RTE APIs for the Port Prototypes of an SWC. You have to take care of this yourself!

Part 3/3: Program Runnables and test it

- ▶ Program Runnables in **CtApMySwc** to have the following behavior
 - ▶ In **RCtApMySwcCode**
 - > Increment the odometer value in each call of the Runnable.
 - > Do the PIM access inside an Exclusive Area (**Rte_Enter**, **Rte_Exit** API)
 - > Send the new value to the Display SWC
 - > Request storing the odometer value to NvM if the value increased at least by 32 since it has been stored the last time AND the NvM has finished the last Write Block request for that Block (see next slide)
 - > If request to store to the odometer value has been accepted, notify the display that the write request is Pending.
 - > Use constants "CWriteReqOnPending" and "CWriteReqNotPending"
 - ▶ In **PpNvMNotifyJobFinishedOdometerCtApMySwc_JobFinished**
 - > If the job finished successfully, notify the display that the write request is not Pending anymore.
 - ▶ Compile and test
 - ▶ If you open the doors too quickly you may see a DET error caused by NvM! Why?

```
#include "CANoeAPI.h"
```

Hint: Use the solution S5 to understand what's going on. Most part is C programming and not AUTOSAR!

Part 3/3: Program Runnables and test it

i

NvM Block service requests are processed asynchronously in the background by the ASR Memory Stack. The application has the possibility to poll the NvM for the end of the service request by invoking the **GetErrorStatus** operation of the NvM Block Service Port.

The ErrorStatus **NVM_REQ_PENDING** will be returned as long as the service request is queued or still in process. Once the service request is finished, the actual job result will be returned.

Therefore, the **NvM_WriteBlock** operation shall only be called for the odometer if the value increased at least by 32 and the ErrorStatus of the NvM Block is unequal to **NVM_REQ_PENDING**.

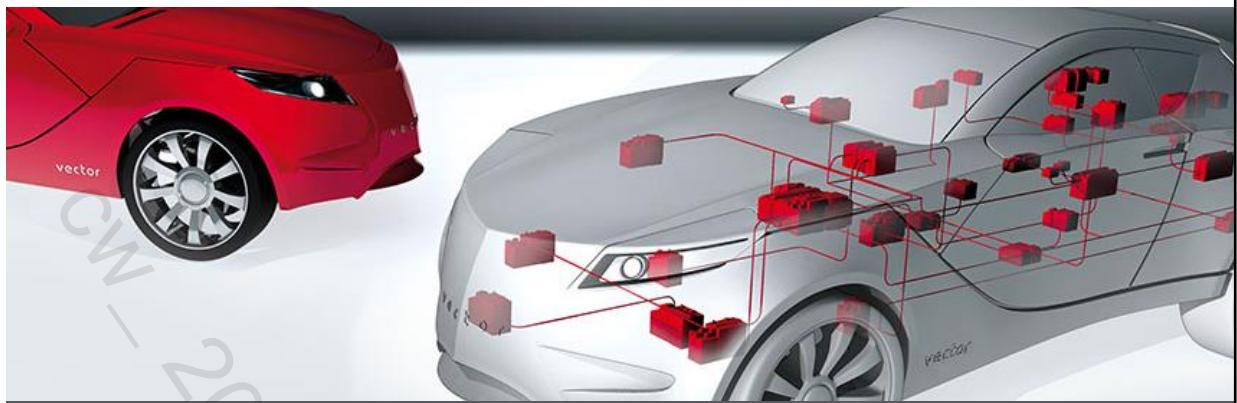
Summary

Think about what you have done in this exercise and why.

- ▶ We have added **NVM** to our project.
- ▶ What data did we store to Non-Volatile Memory?
- ▶ Think about the connection between **Service Needs** for top-down **Service Mapping**.
- ▶ Think about defining **Per-Instance Memories**, **Exclusive Areas**, and **Inter-Runnable Variables** and how they are used with **NVM**.
- ▶ How do you request an **NVM** job? How can you monitor its result?

Memory Abstraction

Diagnostics



AUTOSAR in Practice

Diagnostics

V1.0.0 | 2020-01-20

Agenda

► **Diagnostics**

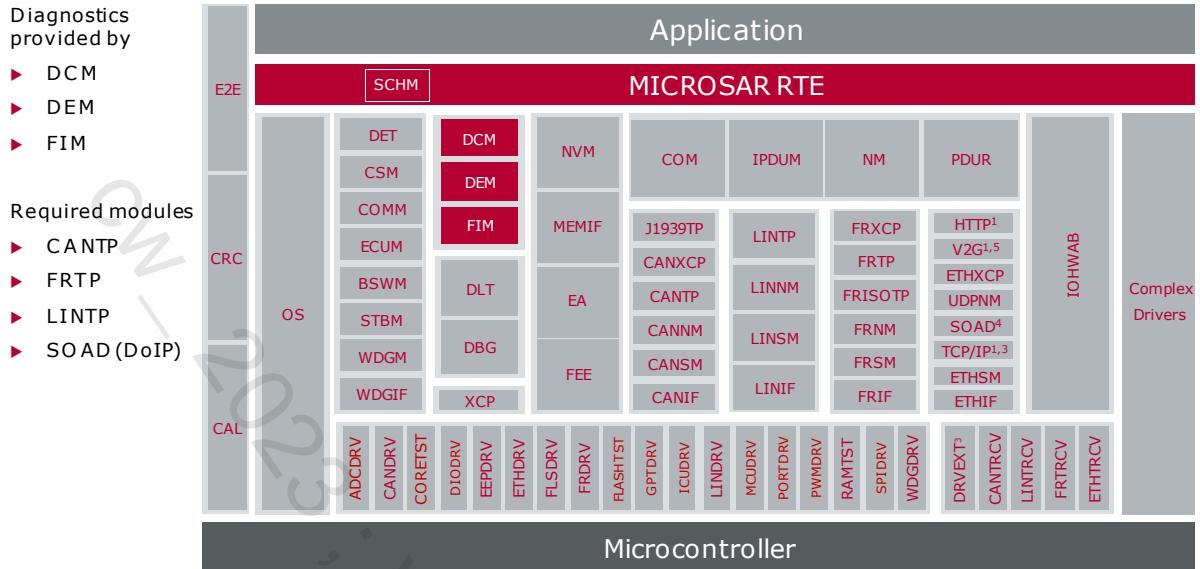
CANTP – CAN Transport Layer

DCM – Diagnostic Communication Manager

DEM – Diagnostics Event Manager

FIM – Function Inhibition Manager

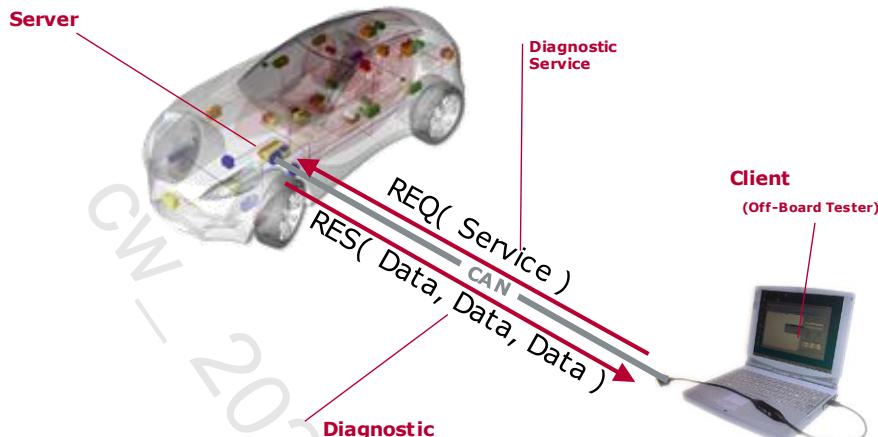
Coming up next



ISO 14229-1

- ▶ Also known as: UDS = Unified Diagnostic Services
- ▶ Definition of diagnostic protocol with services
- ▶ Network-independent
- ▶ Definitions
 - ▶ A "diagnostic protocol" is necessary for communication between a diagnostic tester and an ECU
 - ▶ A diagnostic protocol incorporates a set of diagnostic services that are transmitted serially between the diagnostic tester and ECU(s)
 - ▶ A diagnostic protocol contains data and information on the meanings of the messages to be sent and the ECU's behavior in response to a message

Network View of Diagnostics



Network View of Diagnostics



A **Diagnostic Request** is either **functional** or **physical**. These terms refer to the addressing strategy. Functional addressing is a "broadcast" for all ECUs; physical addressing is peer-to-peer communication (point-to-point).

- A **physical request** will be answered **physically** as this accounts for the peer-to-peer property of the addressing.
- A **functional request** will also be answered **physically**, since the tester has to differentiate the responses of all ECUs from each other.

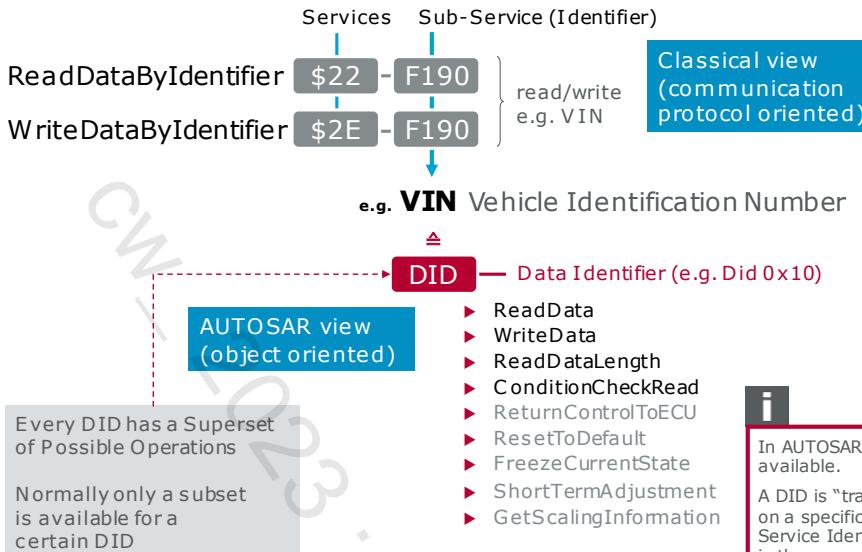
Diagnostic Requests (REQ) and **Diagnostic Responses (RES)** may already contain data, but this is not necessarily the case and depends on the definition of the diagnostic service.

A diagnostic request will typically be followed by a diagnostic response. If there is no response at all the tester will observe a timeout. Some services, like the functional service Tester Present, do not need any response from the ECU at all.

The **Diagnostic Response (RES)** can be one of the following:

- **Positive Response:** The service request has been processed correctly and a response, possibly with data, is on its way to the tester.
- **Negative Response with Negative Response Code (NRC):** The service processing encountered an error specified by the NRC code, and this is indicated to the tester.
- **Negative Response with NRC RequestCorrectlyReceiveResponsePending (RCRRP):** The diagnostic server needs more time to process the request and can not yet deliver a positive or negative response. According to UDS a diagnostic server can respond with RCRRP indefinitely. However, all services are expected to be processed within a finite time. The protocol is not finished unless the ECU answers with a final positive or negative response.

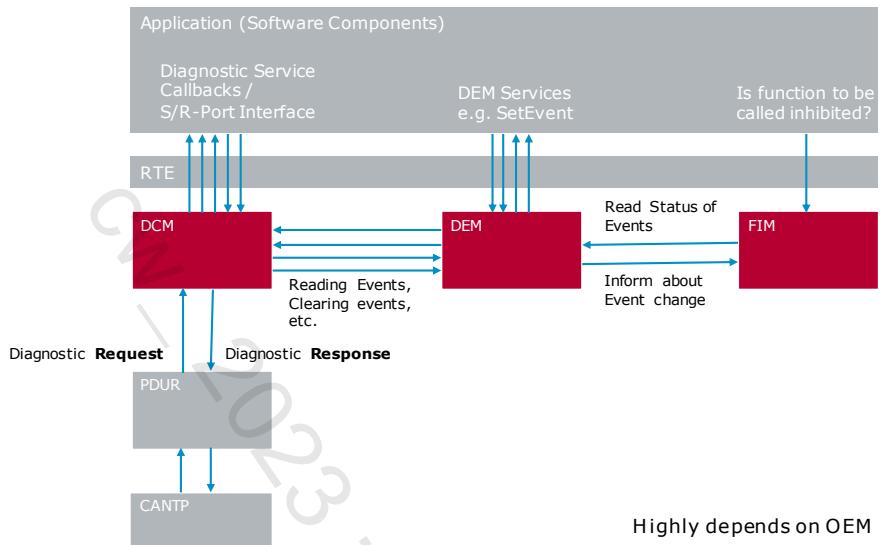
DID – Data Identifier



In AUTOSAR configuration only DIDs are available.

A DID is “translated” into UDS. Every Operation on a specific DID is translated into a Service and Service Identifier, i.e., the appearance on the bus is the same as non-AUTOSAR UDS.

DCM – DEM – FIM



Agenda

Diagnostics

► **CANTP – CAN Transport Layer**

DCM – Diagnostic Communication Manager

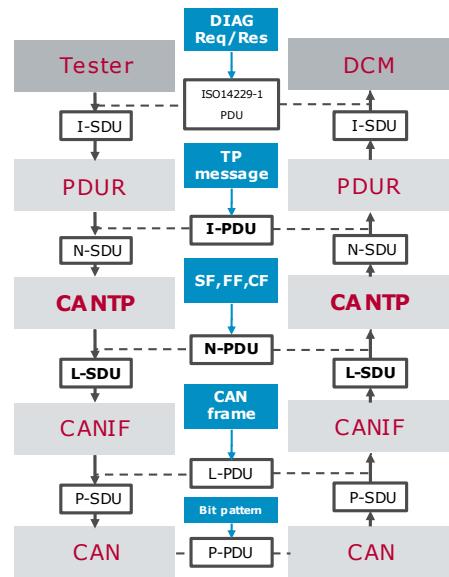
DEM – Diagnostics Event Manager

FIM – Function Inhibition Manager

Coming up next

Module short information

- ▶ CANTP is the module between the PDU Router and the CAN Interface module
- ▶ The main purpose of the CAN TP module is to segment and reassemble diagnostic I-PDUs longer than 8 bytes
 - ▶ A TP message is called an N-SDU (Network-layer Service Data Unit)
 - ▶ Handling of Single (SF), First (FF) and Consecutive Frames (CF) uses N-PDUs (L-SDUs)
- ▶ Support of different addressing formats
 - ▶ Functional and physical
 - ▶ Normal, extended, and mixed



Module short information



N_SA	Network Source Address
N_TA	Network Target Address
N_TAtype	Network Target Address Type
Mtype	Message Type diagnostics, remote diagnostics
N_PCI	Network Layer Protocol Information
N_Data	payload

Normal addressing format

A unique CAN identifier is assigned to each combination of N_SA, N_TA, N_TAtype and Mtype. N_PCI and N_Data are filed in the CAN frame data field.

Extended addressing format

A unique CAN identifier is assigned to each combination of N_SA and Mtype. A unique address is filed to each combination of N_TA and N_TAtype in the first data byte of the CAN frame data field. N_PCI and N_Data are filed in the remaining bytes of the CAN frame data field.

Mixed addressing format

A unique CAN identifier is assigned to each combination of N_SA, N_TA, N_TAtype. N_AE is placed in the first data byte of the CAN frame data field. N_PCI and N_Data are placed in the remaining bytes of the CAN frame data field.

Agenda

Diagnostics

CANTP – CAN Transport Layer

- ▶ **DCM – Diagnostic Communication Manager**

DEM – Diagnostics Event Manager

FIM – Function Inhibition Manager

Coming up next

Overview

- ▶ DCM is network independent
- ▶ Provides a common API for diagnostic services
- ▶ The functionality of this AUTOSAR-Basic-SW is used by external diagnostic tools e.g. in the development, manufacturing, service or legislative OBD
- ▶ The DCM handles different (application layer) **diagnostic protocols**
 - ▶ OBD (ISO 15031-5)
 - ▶ Enhanced diagnostics (ISO 14229-1)
- ▶ For diagnostic **session handling** the network independent sections of the following specifications are also handled by the DCM
 - ▶ ISO 15765-3: Implementation of unified diagnostic services (UDS on CAN)
 - ▶ ISO 15765-4: Requirements for emissions-related systems

Overview



OBD (On Board Diagnostics) refers to legislative diagnostics requirements defined by the respective country in which the vehicle is on the road. OBD is related to the so called "emission related" diagnostics.

In OBD, the diagnostic tester is often referred to as the "generic scan tool."

Among other things, OBD requires:

- Parallel operation to the "ordinary" UDS diagnostic communication
- Special UDS services with their own semantics defined for OBD diagnostics only. Examples:
 - DCM: OBD communication services \$01 - \$0A (defined in SAE J1979 Rev May 2007)
 - DEM: a special OBD FreezeFrame "record 0"...
- Implementing the above requirements in DEM and DCM, AUTOSAR OBD functionality meets all **light duty OBD regulations** in the world
 - EOBD, Japan OBD, California OBDII, ...
- Open issues, not treated in AUTOSAR
 - MIL (Malfunction Indicator Lamp): blinking, light bulb check of the MIL, etc.
 - Misfire fault handling: Debouncing and filtering
 - No description on how to achieve OBD compliant diagnostic applications concerning state handling or diagnostic algorithms.
 - This is due to the difficulties that OBD regulations may change frequently as they are driven by politics and differences over the nations.

Overview

The DCM

- ▶ Ensures diagnostic data flow
- ▶ Manages the diagnostic states
 - ▶ Diagnostic sessions
 - ▶ Security states
- ▶ Checks that the diagnostic service request is supported
- ▶ Checks that the service may be executed in the current session and with the current security state

Overview

- ▶ In the AUTOSAR architecture, the Diagnostic Communication Manager (DCM) is located in the Communication Services (Service Layer)
- ▶ In the Communication process, the DCM receives a diagnostic message from the PDU Router
- ▶ The DCM will check the diagnostic message
- ▶ Depending on the Diagnostic Service ID (SID), the corresponding calls in the Application Layer will be done
- ▶ Supported services
 - ▶ UDS
 - ▶ OBD
 - ▶ Flash Bootloader Interaction
 - > Jump from App ↔ FBL, Flag Management

Services for the Bootloader

- ▶ AUTOSAR 4 DCM may be used in Bootloader software since it supports already the needed services
- ▶ The AUTOSAR 4 DCM has APIs related to the Bootloader
 - ▶ `Dcm_SetProgConditions`
 - > Store relevant information prior to
 - > Jump to bootloader
 - > Jump due to ECUReset request
 - ▶ `Dcm_GetProgConditions`
 - > Called during `Dcm_Init`
 - > allows to determine if a response (\$50 or \$51) has to be sent

Service Id	UDS Service	ASR 4.x
0x23	<i>ReadMemoryByAddress (RMBA)</i>	C
0x28	<i>CommunicationControl</i>	■
0x2C	<i>DynamicallyDefineDataIdentifier</i>	■
0x34	<i>Request Download</i>	C
0x35	<i>Request Upload</i>	C
0x36	<i>TransferData</i>	■
0x37	<i>RequestTransferExit</i>	■
0x3D	<i>WriteMemoryByAddress (WMBA)</i>	C
0x86	<i>ResponseOnEvent (RoE)</i>	■
0x87	<i>LinkControl</i>	Opt.

Block data transfer, also for bootloader

Services for the Bootloader



Legend to the above table

□ not supported (NRC Service Not Supported)

■ supported

C callout (handwritten code, no service ID)

Opt. optionally

Callouts for RMBA/WMBA

- Dcm_ReadMemory

- Dcm_WriteMemory

The AUTOSAR architecture doesn't provide the possibility to access the ECU memory using a physical address. This is implemented using a BlockId which identifies a memory block. Due to this fact, the DCM is not able to fully support the implementation of ISO 14229-1 services which use a physical memory access.

As a solution, the DCM defines a callout to implement this kind of memory access. This callout implementation could simply be realized by defining a mapping between the BlockId and the physical memory address.

Further callouts of the DCM:

- Dcm_Confirmation – successful transmission or error during execution of a diagnostic service.
- Dcm_SetProgConditions – store Bootloader related information before jump into Bootloader
- Dcm_GetProgConditions – read Bootloader information upon startup of ECU (Dcm_Init()) to determine if a RC 0x50 or 0x51 has to be sent
- Dcm_ProcessRequestTransferExit – DCM calls this if a download or upload shall be exited
- Dcm_ProcessRequestUpload – start upload
- Dcm_ProcessRequestDownload – start download

Example for OBD related services

- ▶ The Service Range of OBD Diagnostics has a special Service Identifier (SID) range
- ▶ is prescribed by legislative regulations
 - ▶ Legislative diagnostics
- ▶ This OBD range does not have any intersection with the commonly known UDS services
 - ▶ Manufacturer diagnostics

SID	OBD Service
0x01	<i>Request Current Powertrain diagnostic Data</i>
0x02	<i>Request Power Train FreezeFrame Data</i>
0x03	<i>Request emission-related diagnostic trouble codes</i>
0x04	<i>Clear/reset emission-related diagnostic information</i>
0x06	<i>Request On-Board Monitoring Test-results for Specific Monitored Systems</i>
0x07	<i>Request Emission-Related Diagnostic Trouble Codes Detected during Current or Last Completed Driving Cycle</i>
0x08	<i>Request Control of On-Board System, Test or Component</i>
0x09	<i>Request Vehicle Information</i>
0x0A	<i>Request Emission-Related Diagnostic Trouble Codes with Permanent Status</i>

DCM ComM Request

- ▶ DCM requests and releases communication directly through the ComM
 - ▶ During an ongoing communication service
 - ▶ DCM calls `ComM_DCM_ActiveDiagnostic` upon reception of the UDS request (`Dcm_TpRxIndication`)
 - ▶ DCM calls `ComM_DCM_InactiveDiagnostic` upon transmission of the UDS response (`Dcm_TpTxConfirmation`)
 - ▶ Upon transition into a non-default session
 - ▶ `ComM_DCM_ActiveDiagnostic`
 - ▶ Upon transition into the default session
 - ▶ `ComM_DCM_InactiveDiagnostic`
- ▶ Communication Control service is an exception; this is handled by BswM.
- ▶ Additionally DCM notifies the BSWM about its current Mode using the `BswM_DCM` API
- ▶ An ongoing diagnostics session keeps the communication channels requested
- ▶ In default session, you will only have a COMM Request between sending the UDS Request and the UDS Response from the ECU
- ▶ If you are in a higher ("non-default") session
 - ▶ it will be kept alive by the Tester Present Service
 - ▶ You should observe a permanent COMM Full Communication Request during the whole stay in this session

Mode Management

- ▶ The DCM sends mode requests to the BswM based on the UDS requests it receives
- ▶ DCM acts as a Mode manager for
 - ▶ DcmDiagnosticSessionControl (service 0x10)
 - ▶ DcmEcuReset (part of service 0x11)
 - ▶ DcmModeRapidPowerShutDown (part of service 0x11)
 - ▶ DcmCommunicationControl_<symbolic name of ComMChannelId>. (service 0x28)
 - ▶ DcmControlDTCSetting (service 0x85)
 - ▶ DcmResponseOnEvent_<RoeEventID> (service 0x86)



Example: DCM can request "Disable Normal Communication." During this mode BswM will turn off the corresponding I-PDU groups and NM PDUs.

Data use port

- ▶ Client / Server communication can be used for DIDs
 - ▶ Synchronous or asynchronous handling supported
 - ▶ DID Operations are implemented as Server Runnables in the application SWCs
- ▶ Sender / Receiver communication can also be used for DIDs
 - ▶ R-Port Interface for DcmDspDidRead
 - ▶ P-Port Interface for DcmDspDidWrite
- ▶ Scaling at Ports
 - ▶ The CompuMethod for the data type can optionally be derived from the DcmDspDiagnosisScaling



Sender/Receiver and Scaling supported since MSR4 R17

DCM buffer

- ▶ The DCM buffer must be at least as big as the biggest DID for service 0x22 RDBI / 0x2E WDBI
- ▶ There is also a paged buffer support but this is dedicated for service 0x19 (Read DTCs) and its size is also determined by the biggest extended data record

CW_2023_VH_Autosar CP Training_EN

Agenda

Diagnostics

CANTP – CAN Transport Layer

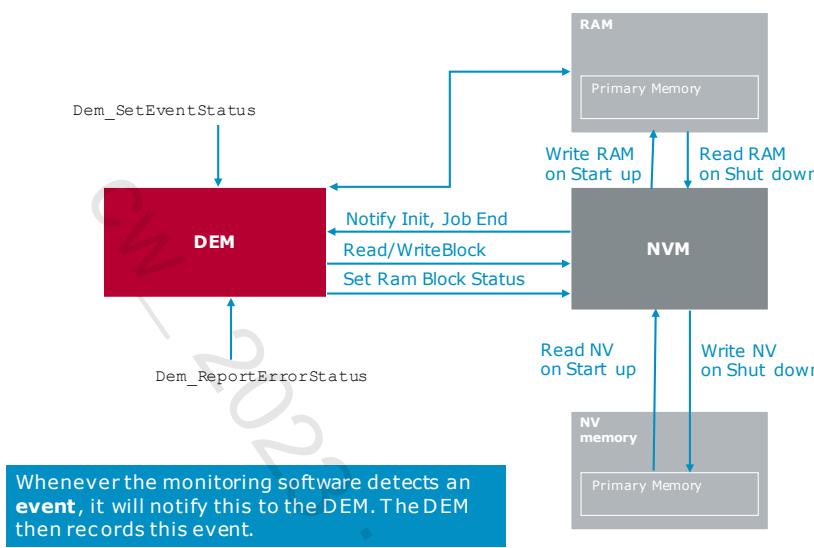
DCM – Diagnostic Communication Manager

► **DEM – Diagnostics Event Manager**

FIM – Function Inhibition Manager

Coming up next

Module operation

**i**

SWCs normally signal fault Events to the DEM by invocation of the Dem_SetEventStatus() API. However not only SWCs can report Events to the DEM. The Basic Software Modules can also set "internal" Events by using the Dem_ReportErrorStatus() API.

The DEM uses various NvM Blocks to store its data:

AdminDataBlock: timestamps and operation cycle counter are saved every driving cycle

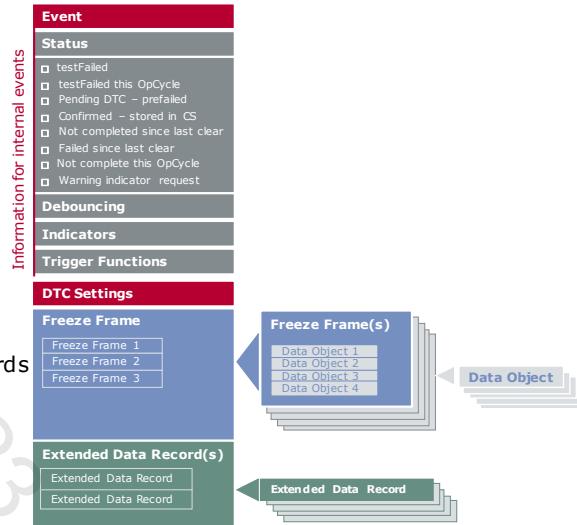
StatusDataBlock: Event data

DemPrimaryDataBlock0..n: The number of remembered DTCs (primary or secondary).

The DEM triggers NVM Block operations during runtime, but some blocks like StatusData and AdminData are only marked as valid and changed with NvM_SetRamBlockStatus API and actually written upon shutdown in NvM_WriteAll().

Events

- ▶ Events have a status
- ▶ “Learning” and “Forgetting” faults: (pre-)debouncing
- ▶ Indicator
- ▶ DTC information
- ▶ Additional information to specify an event
 - ▶ Freeze Frames
 - > Freeze frames have data objects
 - ▶ Extended data records



Events



Freeze Frame:

OEM-specific set of the vehicle/system operation conditions at a specific time. The term "Freeze Frame" is defined as a record of data (DIDs / PIDs). ISO 14229-1 refers to FreezeFrames as "SnapShotRecords". There is the possibility to use pre-stored FreezeFrames. It means the data is already available and does not need to be acquired when the event status changes. This way time-critical Freeze Frame data can be saved together with the event.

Extended data records:

OEM-specific additional information assigned for a distinctive event which is not contained in the Freeze Frame and also described in ISO 14229-1. Extended Data Records are implemented by callbacks to the application. These callbacks are issued by the DEM at the point in time when the Diagnostic Monitor sets the error event in the DEM. The diagnostic configuration can provide 1..n extended data records for a DEM event. Depending on the point in time, different extended data records will be recorded together with the error event. In this way the first occurrence and the last occurrence can be considered.

Example for data not contained in Freeze Frames are: statistical information like **most recent** occurrence counter and **most recent** operation cycle / aging counter (they change over time). The same information contained in Freeze Frames would not change and is "**frozen**".

Event Storage

Each DTC (Diagnostic Trouble Code) is coupled with one byte providing status information. The bits represent the following meanings:

testFailed (bit0)

If the testFailed bit is set it means that the fault is active. The test for an error condition or wrong behavior has succeeded.

testFailedThisOperationCycle (bit1)

This bit is only set if the fault occurs in the current operation cycle. If the fault is active and this flag is not set it means that the fault occurred in a previous operation cycle.

pendingDTC – prefailed (bit2)

Indicates that the corresponding DTC will be set in the near future.

confirmedDTC – stored in Primary Memory (bit3)

As soon as the fault is stored in the Primary Memory, this bit is set. If the fault is cleared from the event storage the value of this bit will be changed too.

testNotCompletedSinceLastClear (bit4)

If there is no information about whether the DTC test has passed or failed since the last clear, this flag is active.

testFailedSinceLastClear (bit5)

This flag informs you that the DTC has set since the last clearing action.

notCompletedThisOperationCycle (bit6)

This bit corresponds with the NotCompletedSinceLastClear. It informs whether the complete information was set or reset in the current operation cycle.

warningIndicatorRequest (bit7)

Indicates whether the occurrence of this DTC should set a warning indicator.

Event Storage

To provide the tester with more information about the point in time a DTC occurred, additional information is stored. This information is OEM-specific and is split into standardized and optional data.

Examples for OEM-specific standardized data:

Occurrence Flag

Distinguishes an entry as occurrence or fault

Original Odometer

This is the odometer value when the fault occurred for the first time.

Most Recent Odometer

This is the odometer value when the fault occurred the most recent time.

Frequency

This is the total number of times the DTC status has transitioned from "stored – not active" to "active."

Operation Cycle Counter

This counter shows the amount of operation cycles (e.g., a new cycle starts after ignition off, ignition run, then wait 10 seconds). This information is used for a so-called self-healing process. If a DTC is stored but not active for a defined amount of operation cycles, it will be cleared from memory if it supports healing.

Pre-debouncing

- ▶ The DEM can debounce Events reported to it via API

```
Dem_SetEventStatus(Dem_EventIdType EventId, uint8 EventStatus)
```

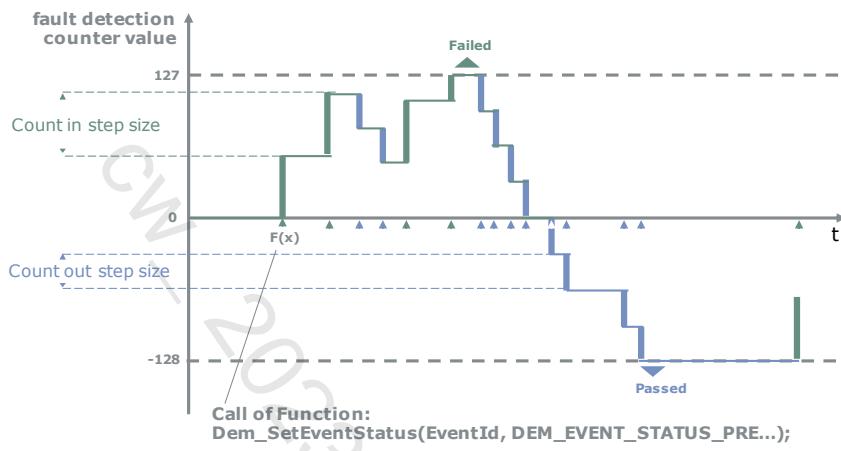
- ▶ Argument values of EventStatus for pre-debouncing:
 - ▶ DEM_EVENT_STATUS_PREPASSED
 - ▶ DEM_EVENT_STATUS_PREFAILED
- ▶ Algorithms for pre-debouncing
 1. Counter based
 2. Time based
 3. Debounce monitor internal^[1]



[1] Monitor internal debouncing is used for monitors that implement the debouncing themselves, and only report qualified results (PASSED/FAILED) to the DEM.

Pre-debouncing: Counter based

Events up – down counter



Pre-debouncing: Counter based



Function Dem_SetEvent() can be called with the following parameter values

- DEM_EVENT_STATUS_PASSED: No debouncing, event is set to passed.
- DEM_EVENT_STATUS_FAILED: No debouncing, event is set to failed.
- DEM_EVENT_STATUS_PREPASSED: Debouncing relevant, configured debouncing done.
- DEM_EVENT_STATUS_PREFAILED: Debouncing relevant, configured debouncing done.

Current fault detection counter value can be requested via service port (Vector extension).

In the previous picture:

PREFAILED indicates the application calls

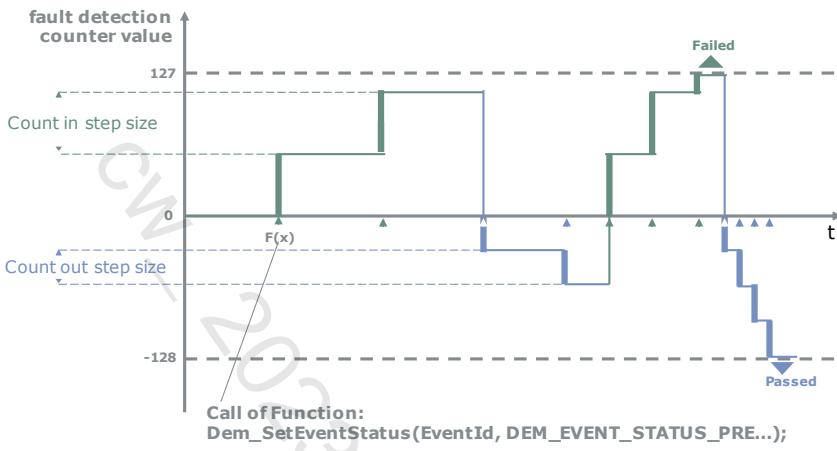
```
Dem_SetEventStatus(EventId, DEM_EVENT_STATUS_PREFAILED)
```

PREPASSED indicates the application calls

```
Dem_SetEventStatus(EventId, DEM_EVENT_STATUS_PREPASSED)
```

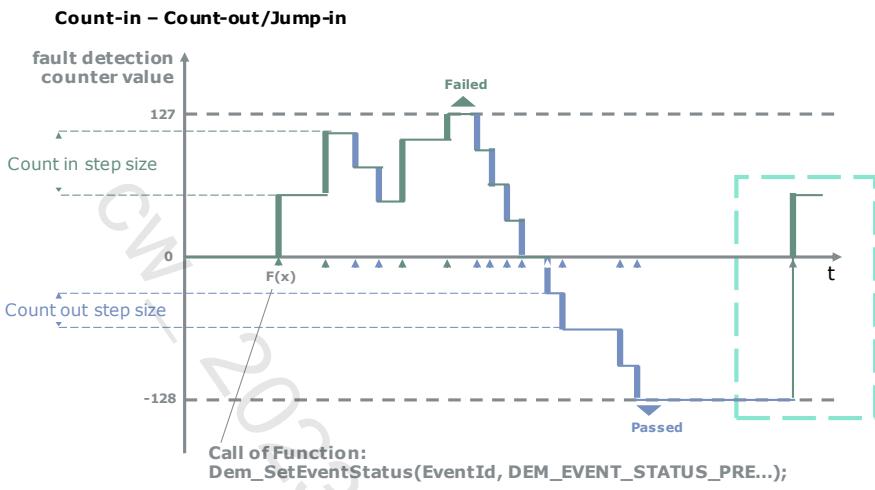
Pre-debouncing: Counter based

Events in a row counter

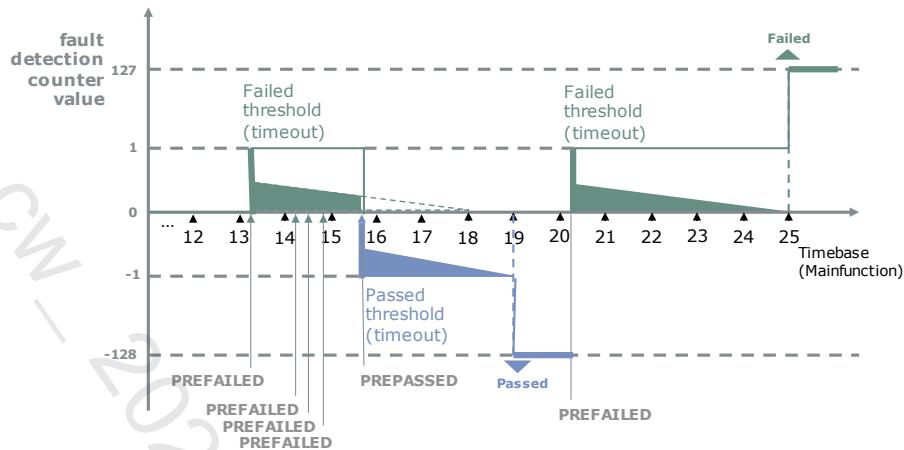


Jumps back to zero if new Dem_SetEvent argument is different from last call.

Pre-debouncing: Counter based



Pre-debouncing: Time based



Failed threshold and passed threshold can be defined individually per DTC and do not necessarily have the same value.

In the above picture:

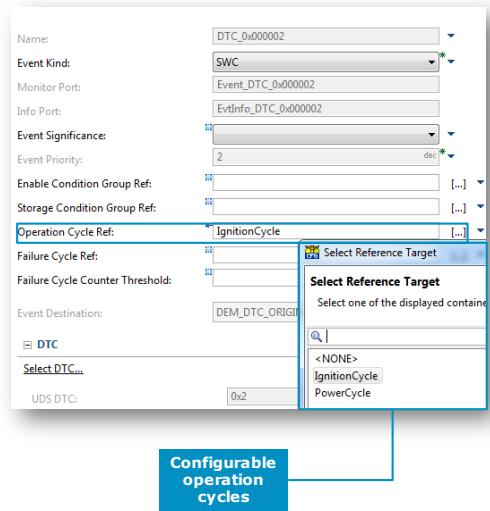
PREFAIRED indicates the application calls
PREPASSED indicates the application calls

`Dem_SetEventStatus(EventId, DEM_EVENT_STATUS_PREFAIRED)`
`Dem_SetEventStatus(EventId, DEM_EVENT_STATUS_PREPASSED)`

Operation Cycles – Events – Mechanisms

Examples of operation cycles are

- ▶ driving cycle
- ▶ engine warm up cycle
- ▶ ignition on/off cycle
- ▶ power up/power down cycle
- ▶ operation active/passive cycle
- ▶ accumulated operating time



Internal and external events

- ▶ EventKind BSW events
 - ▶ Typically not accessible over UDS
 - ▶ The BSW reports them as Production Errors
 - ▶ Defined by A UTOSAR
- ▶ EventKind SWC events
 - ▶ Defined by the application
 - ▶ Typically accessible over UDS
- ▶ If you assign a DTC to an event, it can be queried over UDS
 - ▶ you can also equip an existing event (especially EventKind BSW) with a DTC by redefining it in the Diagnostic description file (CDD, DiagEx)

Dem Events	Event Kind	Monitor Port	Operation Cycle Ref
DTC_0x000002	SWC*	Event_DTC_0x000002	IgnitionCycle
AutoCreatedDemEvent_EcuMConfiguration_ECUM_E_CONFIG	BSW		PowerCycle
AutoCreatedDemEvent_EcuMConfiguration_ECUM_E_RAM_CL	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_INTEGRITY_FAILED	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_LOSS_OF_REDUNDANCY	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_QUEUE_OVERFLOW	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_REQ_FAILED	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_VERIFY_FAILED	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_WRITE_PROTECTED	BSW		PowerCycle
AutoCreatedDemEvent_NVM_E_WRONG_BLOCK_ID	BSW		PowerCycle

Internal and external events



Operation Cycle Kind of operation cycle for the event storage (e.g. power cycle, driving cycle, ...)

Healing Support Switch to allow healing/unlearning or not.

Healing Cycles Number of Operation Cycles necessary to heal/erase event.

Monitor Initialization Function Function which resets the error monitoring in the application

Event Destination

Bit 0: DEM_DTC_ORIGIN_PRIMARY_MEMORY

Bit 1: DEM_DTC_ORIGIN_SECONDARY_MEMORY

Bit 2: DEM_DTC_ORIGIN_MIRROR_MEMORY

Set the according Bit to '1' means event is assigned to the origin.

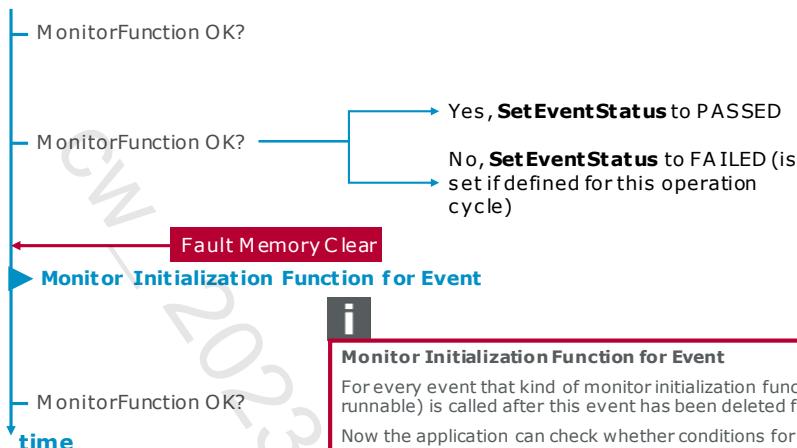
If an event is not assigned to an origin, then it is only handled internally and it is not visible externally.

Multiple origins are not supported.

Priority Priority of an event. This priority comes into play if the event memory is full. The priority decides if the new event is lost or if it overwrites a lower priority event in the memory.

DTC Number Diagnostic Trouble Code is a numeric code which consists of 3 bytes unique identifier for the error event. Together with this information, the status byte is also always saved. That means the minimum memory consumption of a DTC is 4 bytes.

Diagnostic Monitor Initialization



After event is cleared a monitor initialization function is called.

The application can check whether conditions for this event are still there or not.



Monitor Initialization Function for Event

For every event that kind of monitor initialization function can be defined. This function (a runnable) is called after this event has been deleted from DTC memory.

Now the application can check whether conditions for entering this event are still present (e.g. for usage in the station to check currently present events).

Normally the monitor functions have large call cycles and may only be called seldom. With this initialization function, reaction time to a still-present event is very short.

OBD functionality

- ▶ OBD functionality services \$01 - \$0A defined in SAE J1979 Rev May 2007
 - ▶ light duty OBD regulations OBDII, EOBD, Japan OBD, all others
- ▶ Only Legislative FreezeFrame (record 0) supported, records 1 and above are ignored
- ▶ The DEM calculates some PIDs internally (see next slide)
- ▶ The PID 0x0D Vehicle Speed is required for all OBD ECUs

OBD functionality



Some details on the interaction between DEM and specific SWC might remain open, since they are dependent on the DEM and SWC implementation.

The following functionality is not defined:

- Malfunction Indicator Lamp (MIL)-activation (interface DEM to MIL handler, MIL-bulb check, readiness blinking, blinking in case of catalyst damaging misfire...)
- Misfire fault handling (common debouncing, filtering single / multiple misfire faults).

However the DEM does not prescribe implementation details on how OBD compliance can be achieved within the DEM module, e.g. concerning state handling. Furthermore, the DEM does not prescribe implementation details on the diagnostic algorithms of the SWC necessary to achieve OBD compliance (how to detect a fault, when to trigger increment of IUMPR-numerator...).

IUMPR = In-Use-Monitor Performance Ratio

Internally calculated PIDs

- PID \$01 – Monitor status since DTCs cleared (4 byte)
- PID \$02 – FreezeFrame DTC (2 byte)
- PID \$21 – km with MIL On (2 byte)
- PID \$30 – number of Warm Up cycles (WUC) since last fault clear (1 byte)
- PID \$31 – km since last fault clear (2 byte)
- PID \$41 – Monitor status this driving cycle (4 byte)
- PID \$4D – time with MIL On (2 byte)
- PID \$4E – time since last fault clear (2 byte)
- PID \$1C – OBD requirements to which vehicle or engine is certified. (1 byte)

SRDataServices

- ▶ Sender Receiver Communication support
 - ▶ Port Interface SRDataServices_<SyncDataElement>
 - ▶ One Port Prototype per DID, PID, Extended Data Record Interface

Agenda

Diagnostics

CANTP – CAN Transport Layer

DCM – Diagnostic Communication Manager

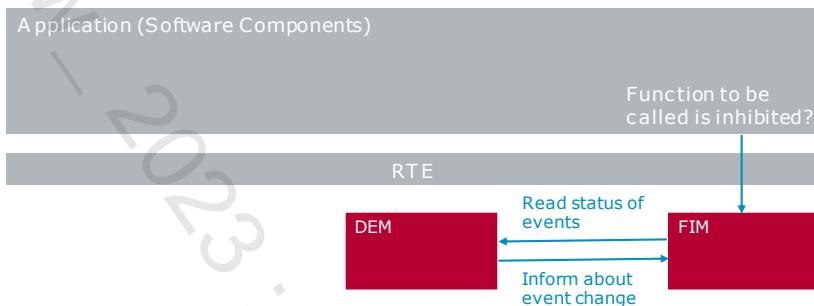
DEM – Diagnostics Event Manager

► **FIM – Function Inhibition Manager**

Coming up next

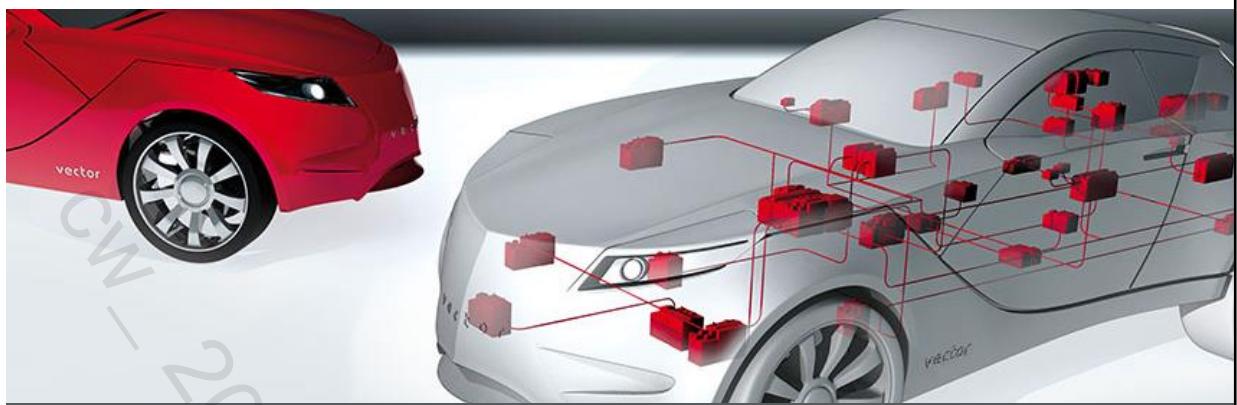
The FIM knows which functions are inhibited at the moment

- ▶ The application has to ask the FIM whether a certain function can be run or not.
 - ▶ FIM either stores the condition for each function identifier (FID) and is triggered by DEM on occurrence of an event
 - ▶ Or FIM requests the current events state from DEM and evaluates the condition for function identifier on request.



Diagnostics

Exercise: Diagnostics



AUTOSAR in Practice

Diagnostics

V28 | 2023-03-28

Agenda

► **Exercise 6 - Diagnostics**

Coming up next

CW_2023_VH_Autosar CP Training_EN

Diagnostics

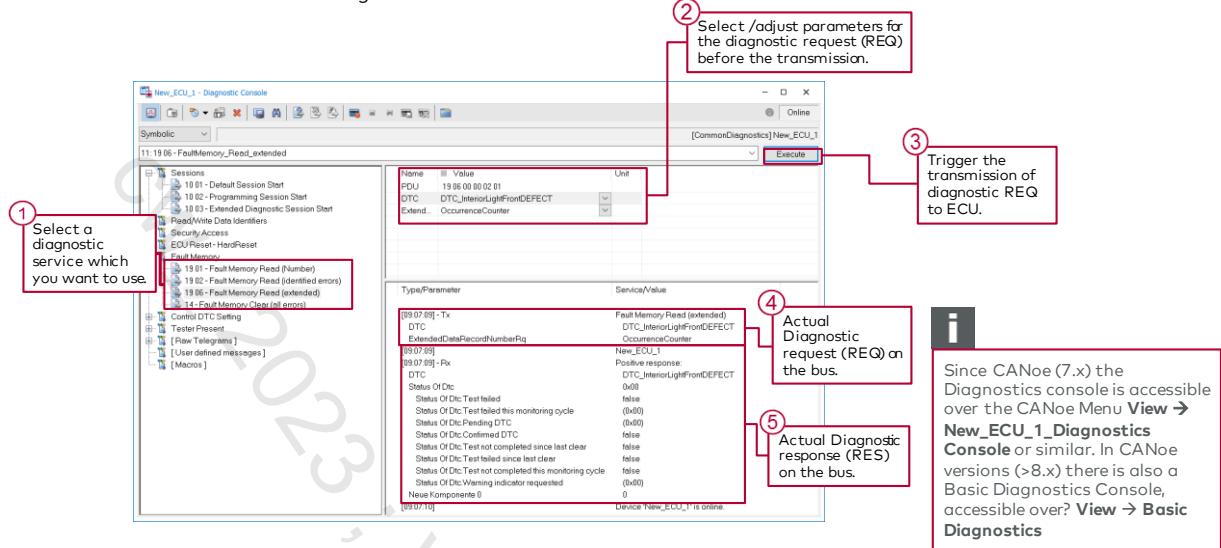


Extend the example with diagnostics (**DCM**) and fault memory (**DEM**) derived from a given CANdela Diagnostic Description (CDD) file.
Configure the modules in the Configurator and import them as Service Components into the Developer.

Provide necessary Service Ports and Runnables and do the coding.

Diagnostic Tester in CANoe

A TESTER is simulated via the Diagnostics Console of CANoe



What is possible in which session

Dependencies						
Service	Prefix	Default	Programming	Extended	Safety System	
Session/*/Start	10 LL	(mixed)	(mixed)	(mixed)	(mixed)	
Default	10 01	Default	Default	Default	Default	
Programming	10 02	Programming	Programming	Programming	Programming	
Extended	10 03	Extended	n.a.	Extended	Extended	
Safety System Diagnostic Session	10 04	SafetySystem	n.a.	SafetySystem	SafetySystem	
Ecu Reset/*/Reset	11 LL	✓	✓	✓	✓	
HardReset	11 01	✓	✓	✓	✓	
Security Access/*/Request	27 LL fd	n.a.	✓	✓	✓	
Security Access Mode 1 - Seed	27 01 fd	n.a.	✓	✓	✓	
Security Access /*/Send	27 LL yy	n.a.	✓	✓	✓	
Security Access Mode 1 - Key	27 02 yy	n.a.	✓	✓	✓	
Tester Present/Process	3E 00	✓	✓	✓	✓	
Tester Present	3E 00	✓	✓	✓	✓	
Control DTC Setting/*/Control	85 LL	n.a.	n.a.	✓	✓	
DTC Setting Mode On	85 01	n.a.	n.a.	✓	✓	
DTC Setting Mode Off	85 02	n.a.	n.a.	✓	✓	
Identification/*/Read	22 LL	✓	✓	✓	✓	
Fault Memory/*/Clear	14 pp	✓	n.a.	✓	✓	
Fault Memory/ReportNbrOfDTCStatusMask	19 01 zz	✓	✓	✓	✓	
Fault Memory/ReportNbrOfMirrorMemoryDTCByStatusMask	19 11 zz	✓	✓	✓	✓	
Fault Memory/ReportDTCByStatusMask	19 02 zz	✓	✓	✓	✓	
Fault Memory/ReportMirrorMemoryDTCByStatusMask	19 0F zz	✓	✓	✓	✓	
Fault Memory/ReportDTCExtendedDataByDTCNbr	19 06 vv ss	✓	✓	✓	✓	
Fault Memory/ReportMirrorMemoryDTCExtendedDataByDTCNbr	19 10 vv ss	✓	✓	✓	✓	
Fault Memory/ReportSupportedDTCs	19 0A	✓	✓	✓	✓	



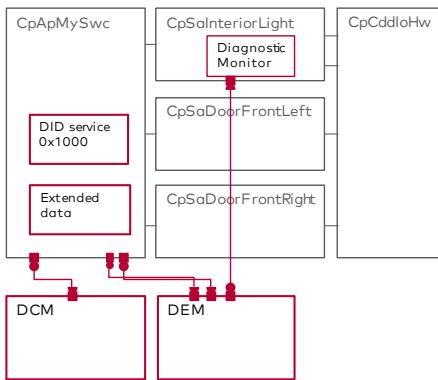
You can switch sessions using the Diagnostic Console of CANoe. To know what is possible in which session, use the table above.

Possible Sessions:

- Default
- Programming
- Extended
- SafetySystem

Overview

- ▶ Concept for our diagnostics application
- ▶ **CtSaInteriorLight**
 - ▶ detects if the light bulb of the interior light is defective and reports the test verdict for DTC_0x000002 to the **DEM**. This is a good place for our **Diagnostic Monitor** which performs the tests on the functionality of the light.
- ▶ **CtApMySwc**
 - ▶ can ideally provide additional information for the recording of the **DEM Error Event** (Odometer value and occurrence counter) in Extended Data Records collected in **CSDataservices**.
 - ▶ offers also the counter how many switching events of the interior light we (have) observed (until now) (LightOnOffCounter), so the DCM can gather if (there) is information here and even alter the value (with) **DataServices_Data_Diag_RWDI**.



6

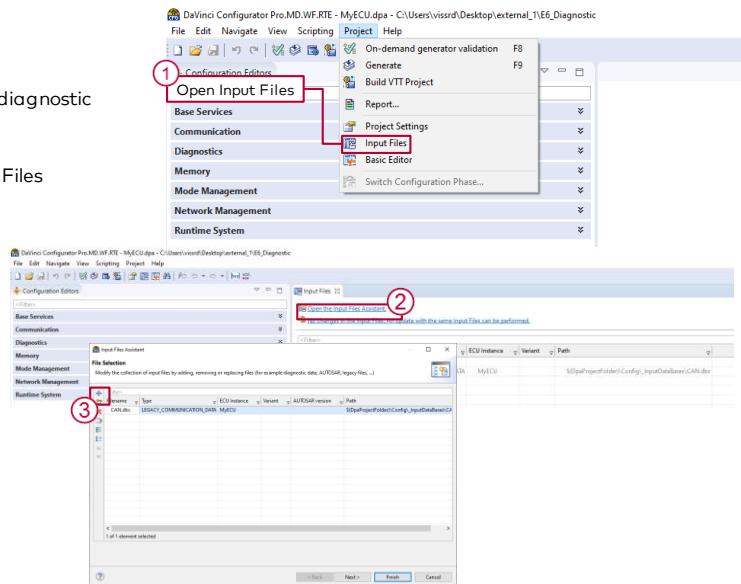
Diagnostics

- **1** Configure DCM and DEM (derived from CDD)
- **2** Service Port Prototypes and Access Points
- **3** Generate, program Runnables and Test it
- **4** Add-On I

Part 1/4: Configure DCM and DEM with Configurator

- ▶ "Configure BSW" using **MyECU.dpa** file in folder **\E6_Diagnostics**

- ▶ 1. Go to **Project → Input Files** to add the diagnostic description to your project
- ▶ 2. Click on the blue link to open the Input Files Assistant
- ▶ 3. Select Plus to add new files



8

Validate configuration

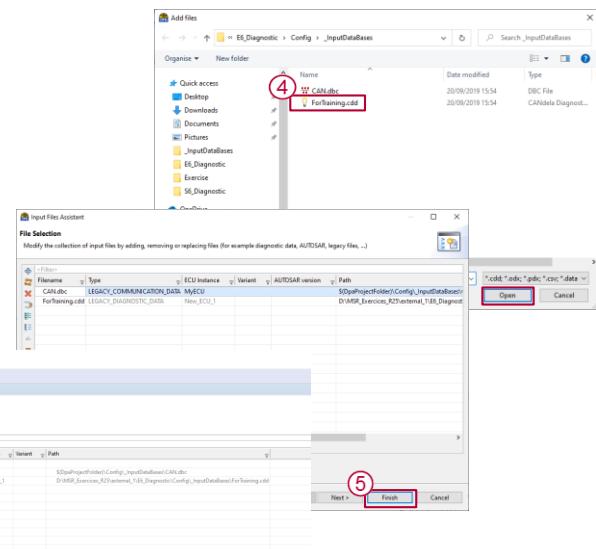
Start the validation and code generation process. Optionally tI configured.

Generation Target: Virtual Target (vVIRTUALtarget)

<Filter>	
<input type="checkbox"/>	Generation Step
<input type="checkbox"/>	Communication
<input checked="" type="checkbox"/>	Complex Driver
<input type="checkbox"/>	Diagnostics
<input type="checkbox"/>	CanTp: CanTp
<input type="checkbox"/>	Dcm: Dcm
<input checked="" type="checkbox"/>	Dem: Dem

Part 1/4: Configure DCM and DEM with Configurator

- ▶ 4. choose the correct diagnostic description "ForTraining.cdd" and click open → here cdd stands for CANdela Diagnostic Description
- ▶ 5. Select Finish
- ▶ 6. After you finished your selection you can update your configuration by clicking the "Update the configuration now to commit the project modifications" Link



9

Validate configuration

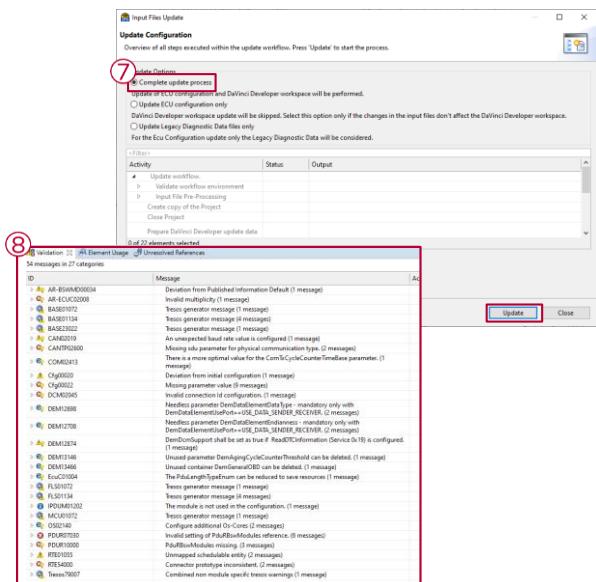
Start the validation and code generation process. Optionally the configuration can be validated.

Generation Target: Virtual Target (\VIRTUALtarget)

<Filter>	
<input type="checkbox"/>	Generation Step
<input type="checkbox"/>	Communication
<input checked="" type="checkbox"/>	Complex Driver
<input type="checkbox"/>	Diagnostics
<input type="checkbox"/>	CanTp: CanTp
<input type="checkbox"/>	Dcm: Dcm
<input checked="" type="checkbox"/>	Dem: Dem

Part 1/4: Configure DCM and DEM with Configurator

- ▶ 7. Select "Complete update process" to update your ECU configuration and the DaVinci Developer Workspace and click Update
- ▶ 8. As a result, your Validation Window should look like this. Most of the infos and warnings are already known from the exercises before.



10

Validate configuration

Start the validation and code generation process. Optionally tick the checkboxes to validate and generate the configuration.

Generation Target: Virtual Target (vVIRTUALtarget)

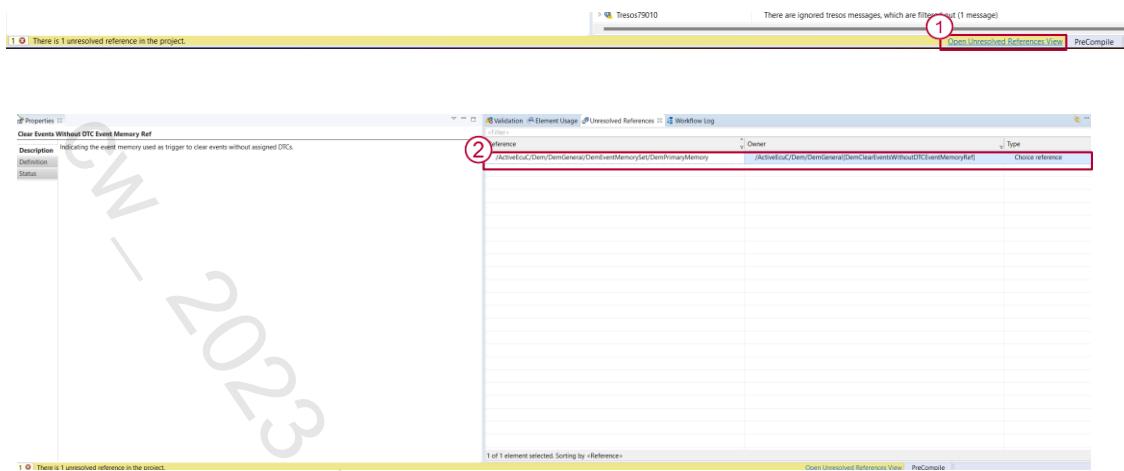
<Filter>

Generation Step

- ▶ Communication
- ▶ Complex Driver
- ▶ Diagnostics
 - CanTp: CanTp
 - Dcm: Dcm
 - ✓ Dem: Dem

Part 1/4: Set Primary Memory for DEM Events

- ▶ 1. Navigate to "Open Unresolved References View"
- ▶ 2. Open with a double click the "Reference"



11

Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (vVIRTUALtarget)

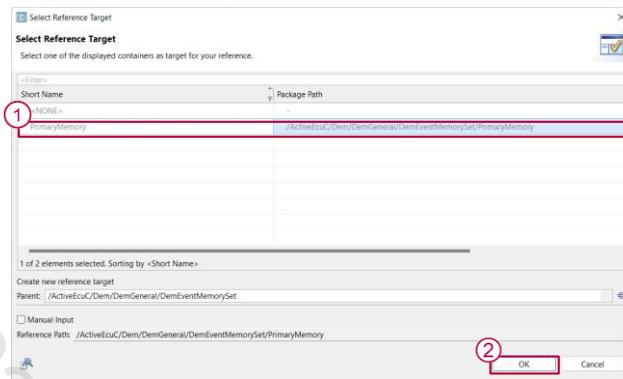
<Filter>

Generation Step

- Communication
- Complex Driver
- Diagnostics
- CanTp: CanTp
- Dcm: Dcm
- Dem: Dem

Part 1/4: Set Primary Memory for DEM Events

- ▶ 1. Select the PrimaryMemory
- ▶ 2. Confirm with "OK"



12

Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (vVIRTUALtarget)

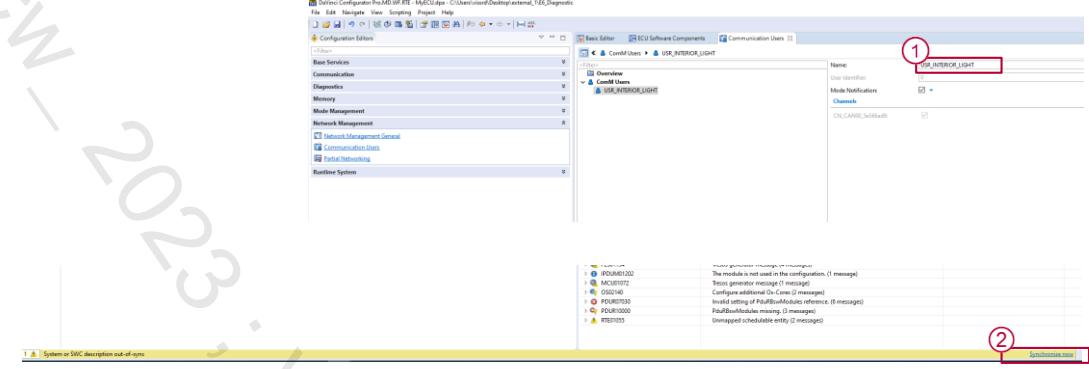
<Filter>

Generation Step

- ▶ Communication
- ▶ Complex Driver
- ▶ Diagnostics
- ▶ CanTp: CanTp
- ▶ Dcm: Dcm
- ▶ Dem: Dem

Part 1/4: Rename Communication User

- ▶ First let's solve the RTE54000 Error. This Error is a Result of updating the Workspace
- ▶ 1. Navigate to the Network Management and go to the Communication User and change the name to "USR_INTERIOR_LIGHT"
- ▶ 2. Synchronize your System

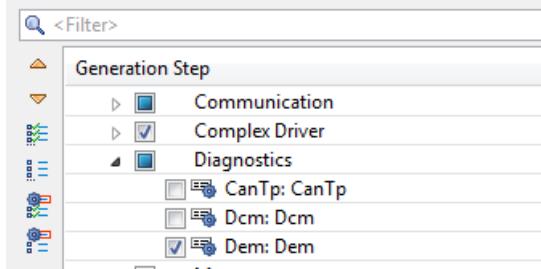


13

Validate configuration

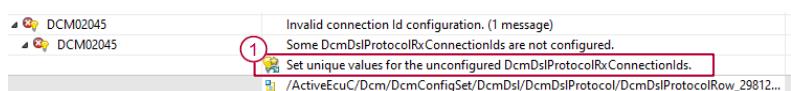
Start the validation and code generation process. Optionally the configuration can be generated.

Generation Target: Virtual Target (vVIRTUALtarget)



Part 1/4: Configure DCM and DEM with Configurator

- ▶ The next Error which we want to solve is the DCM2045. This Error appears because a unique identifier must be set for the DCM Main Connection.
 - > "Unique identifier of the tester which uses this connection for diagnostic communication."
- ▶ 1. To solve this Error simply use the Auto solving Action.



Validate configuration

Start the validation and code generation process. Optionally the configuration can be generated.

Generation Target: Virtual Target (\VIRTUALtarget)

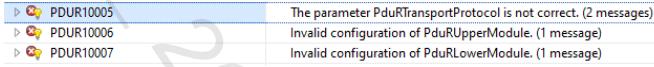
<Filter>	
Generation Step <ul style="list-style-type: none"> ▶ <input type="checkbox"/> Communication ▶ <input checked="" type="checkbox"/> Complex Driver ▶ <input type="checkbox"/> Diagnostics <ul style="list-style-type: none"> <input type="checkbox"/> CanTp: CanTp <input type="checkbox"/> Dcm: Dcm <input checked="" type="checkbox"/> Dem: Dem 	

Part 1/4: Configure DCM and DEM with Configurator

- ▶ PDUR1000 appears because with the update of our System we automatically added a CANTP and a DCM Module to our Workspace but the reference in the already existing PDUR Module is missing.

- > 1. Use the auto solving Action by do a right click on PDUR10000:
"Solve all default Actions"

- ▶ After using the auto solving action the following PDUR Errors appear
how to solve these errors you will see on the next slides


 ▶ PDUR10000 Solve all Default Actions Modules missing. (3 messages)
 ▶ PDUR10000 PduRModule CanTp is involved in a PduR routing path by the global Pdu reference PduSrcPduRef(values=msg_diag_RequestGlobal_oCAN00_3d4e807_Rx).
 ▶ PDUR10000 Create PduRModule referencing /ActiveEcu/CanTp
 ▶ PDUR10000 /ActiveEcu/C/PduR/PduRRoutingTables/PduRRoutingTable/msg_diag_RequestGlobal_oC...
 ▶ PDUR10000 The container PduRModules referencing the BswModule CanTp is missing. The BswModule CanTp is involved in a PduR routing path by the global Pdu reference PduSrcPduRef(values=msg_diag_Request_MvECU_oCAN00_a223fbba_Rx).
 ▶ PDUR10000 The container PduRModules referencing the BswModule Dcm is missing. The BswModule Dcm is involved in a PduR routing path by the global Pdu reference PduSrcPduRef(values=msg_dian_Response_MvFCU_oCAN00_f4dchhhd_Tx).

▶ PDUR1000 The parameter PduRTransportProtocol is not correct. (2 messages)
 ▶ PDUR1000 Invalid configuration of PduRUpperModule. (1 message)
 ▶ PDUR1000 Invalid configuration of PduRLowerModule. (1 message)

Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (vVIRTUALtarget)

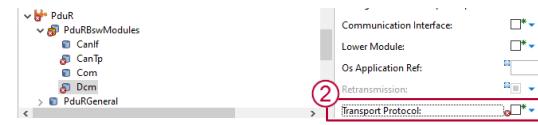
< Filter>

Generation Step

- Communication
- Complex Driver
- Diagnostics
- CanTp: CanTp
- Dcm: Dcm
- Dem: Dem

Part 1/4: Configure DCM and DEM with Configurator

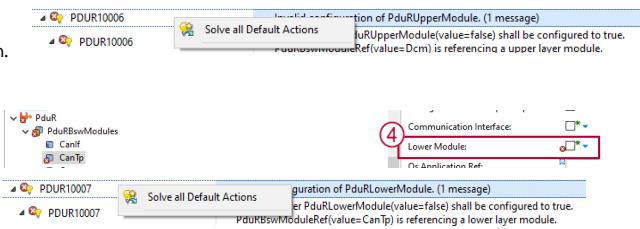
- 2. The PDUR 10005 error appears because the DCM is using the CANTP. Therefor we have to set the parameter "Transport Protocol" to True. Look to the right to find the parameter in the Basic Editor or use simply the auto solving action and set the parameter to true.



- 3. To solve the error 10006 use the autosolving action. That effects that the PduR can use the APIs of the upper module Dcm



- 4. To solve the error 10007 use the autosolving action. That effects that the PduR can use the APIs of the lower module CanTp



Validate configuration

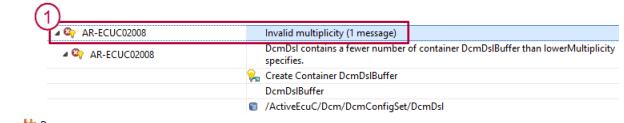
Start the validation and code generation process. Optionally the configuration can be generated.

Generation Target: Virtual Target (\VIRTUALtarget)

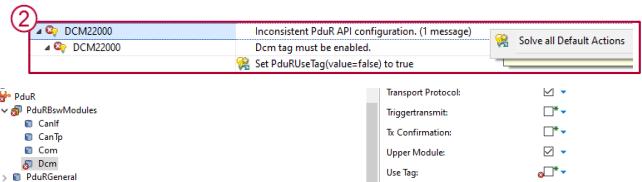
<Filter>	
<input type="checkbox"/>	Generation Step
<input type="checkbox"/>	Communication
<input checked="" type="checkbox"/>	Complex Driver
<input type="checkbox"/>	Diagnostics
<input type="checkbox"/>	CanTp: CanTp
<input type="checkbox"/>	Dcm: Dcm
<input checked="" type="checkbox"/>	Dem: Dem

Part 1/4: Configure DCM and DEM with Configurator

- ▶ 1. Now for the DCM we have to create a buffer. You can do this by using the autosolving action or navigate to Dcm/DcmConfigSet/DcmDslBuffer. With a right click you can add a new buffer



- ▶ 2. To solve the DCM Error 22000 use the autosolving action or navigate to the corresponding parameter



Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (\VIRTUALtarget)

<Filter>	
	Generation Step
	Communication
	Complex Driver
	Diagnostics
	CanTp: CanTp
	Dcm: Dcm
	Dem: Dem

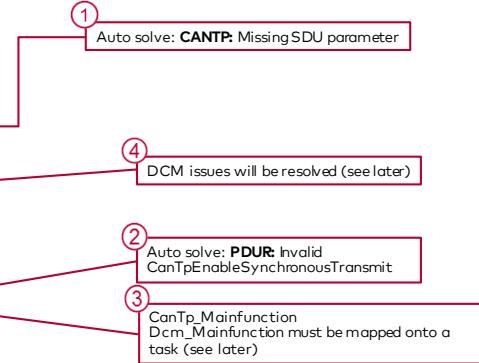
Part 1/4: Configure DCM and DEM with Configurator

- 1. Set the DemDcmSupport parameter to true with the help of the autosolving action. If you set this parameter to TRUE the standard Dem behaviour and APIs are available.

DEM12874	DemDcmSupport shall be set as true if ReadDTCInformation (Service 0x19) is configured. (1 message)
DEM12874	The parameter DemDcmSupport is set to true whenever ReadDTCInformation (Service 0x19) is configured. Set DemDcmSupport(value:=true) to true

- 2.

Validation		Element Usage	Unresolved References
34 messages in 19 categories			
ID	Message		
> AR-BSWMD00034	Deviation from Published Information Default (1 message)		
> BASE01072	Tresos generator message (1 message)		
> BASE01134	Tresos generator message (4 messages)		
> BASE23022	Tresos generator message (1 message)		
> CAN02019	An unexpected baud rate value is configured (1 message)		
> CANTP02600	Missing sdu parameter for physical communication type. (2 messages)		
> COM02413	There is a more optimal value for the ComTx/CycleCounterTimeBase parameter. (1 message)		
> Cfg00022	Deviation from initial configuration (2 messages)		
> Cfg00022	Missing parameter value (5 messages)		
> DEM13466	Unused container DemGeneralOBd can be deleted. (1 message)		
> E2EPW79203	No EndToEndProtections in the project available (1 message)		
> EcoG01004	The PduLengthTypeEnum can be reduced to save resources (1 message)		
> FLS01134	Tresos generator message (4 messages)		
> IPDUM01202	The module is not used in the configuration. (1 message)		
> OS02140	Configure additional Os-Cores (2 messages)		
> PDUR11005	Invalid CanTp.CanTpEnableSynchronousTransmit configuration. (1 message)		
> RTE01055	Unmapped schedulable entity (2 messages)		
> RTE13024	Mode request type Map missing (2 messages)		
> Tresos79007	Combined non module specific tresos warnings (1 message)		



Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (\VIRTUALtarget)

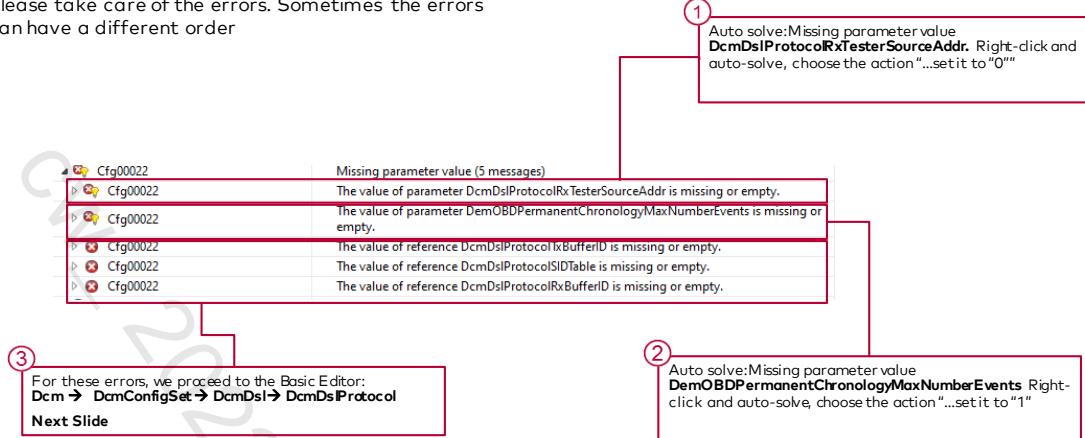
< Filter>

Generation Step

- Communication
- Complex Driver
- Diagnostics
 - CanTp: CanTp
 - Dcm: Dcm
 - Dem: Dem

Part 1/4: Configure DCM and DEM with Configurator

- Please take care of the errors. Sometimes the errors can have a different order



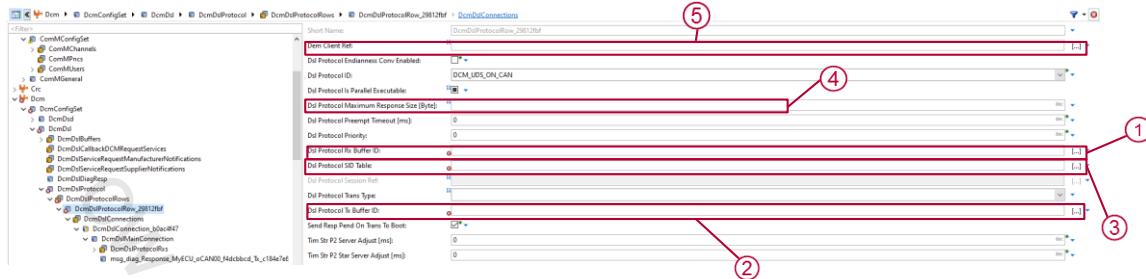
Validate configuration

Start the validation and code generation process. Optionally the target can be configured.

Generation Target: Virtual Target (vVIRTUALtarget)

<Filter>	
<input type="checkbox"/>	Generation Step
<input type="checkbox"/>	Communication
<input checked="" type="checkbox"/>	Complex Driver
<input type="checkbox"/>	Diagnostics
<input type="checkbox"/>	CanTp: CanTp
<input type="checkbox"/>	Dcm: Dcm
<input checked="" type="checkbox"/>	Dem: Dem

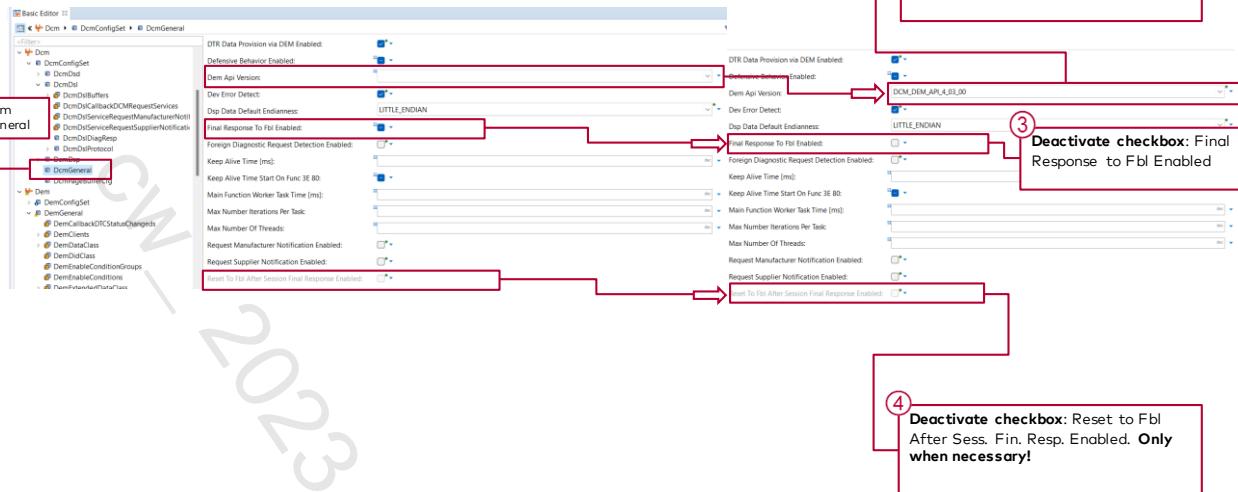
Part 1/4: Configure DCM and DEM with Configurator



- ▶ In Dcm → DcmConfigSet → DcmDsIProtocol → DcmDsIProtocolRows → DcmDsIProtocolRow_29812fbf
- ▶ Go to DcmDsIProtocolRow and
- ▶ (1) Add Rx and (2) Tx Buffer ID References to the existing DcmDsIBuffer, by selecting Reference [...] to Tx/Rx Buffer
- ▶ (3) Add a reference to the existing Service Id Table DcmDsdServiceTable
- ▶ (4) Create Parameter DcmDsIProtocolMaximumResponseSize of 4096
- ▶ (5) Select Reference [...] to Dem Client on DemClient
- ▶ Now Synchronize your Workspace or autosolve the DCM94990 Error.

Part 1/4: Configure DCM and DEM with Configurator

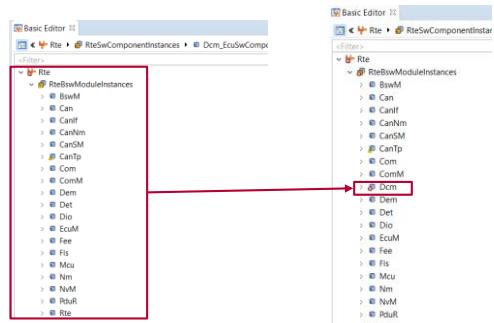
- Go to the Basic Editor **Dcm** → **DcmConfigSet** → **DcmGeneral**. Change the FBL response behavior:



Part 1/4: Configure DCM and DEM with Configurator

- 1. A instance of the Dcm in the RTE is needed. Use the auto solving action to create one.

Validation	
ID	Message
> FLS01134	Tresos generator message (4 messages)
> OS02140	Configure additional Os-Cores (2 messages)
▲ RTE01005	Missing BswModuleInstance configuration (1 message)
▲ RTE01005	Rte configuration misses BswModuleInstanceContainer for Dcm
	└ create BswModuleInstance for Dcm
	└ /ActiveEcuC/Rte

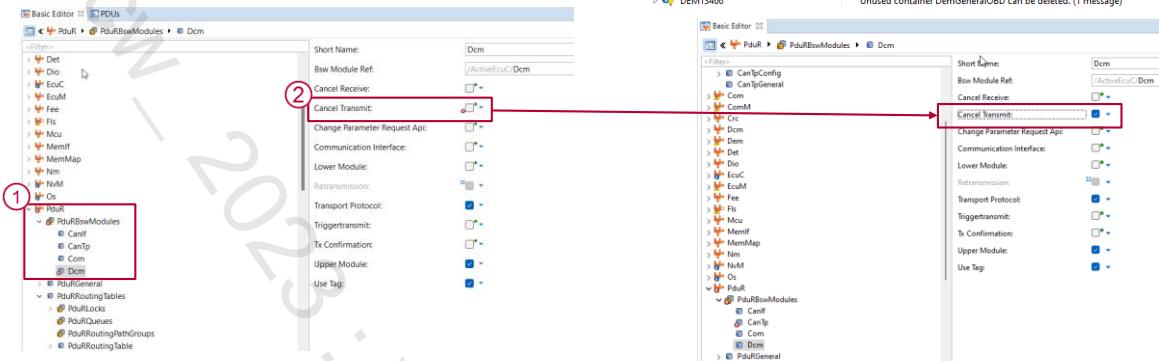


No instance

with instance

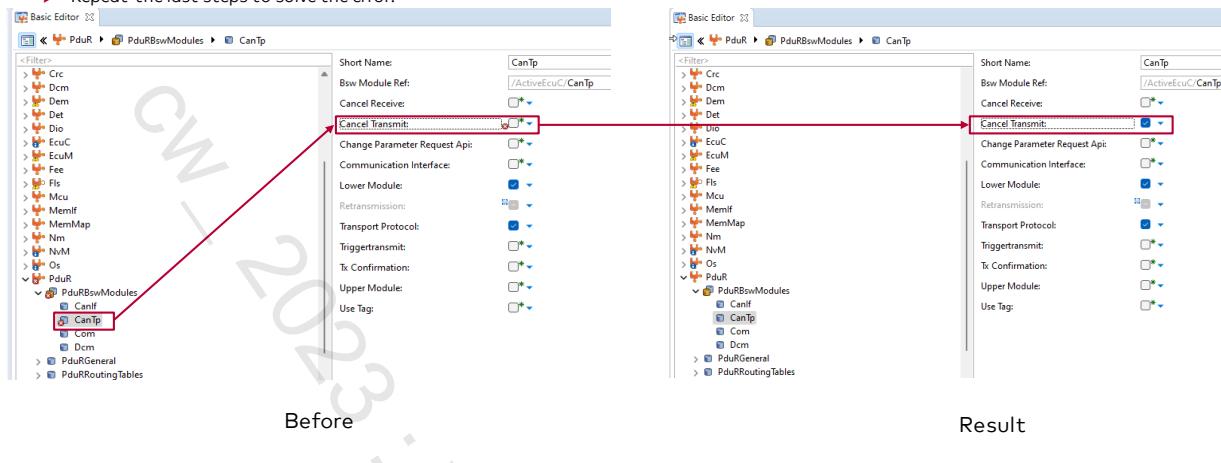
Part 1/4: Configure DCM and DEM with Configurator

- DCM requires the Cancel Transmit API. To set it up follow the steps or use the autosolving action.
- 1. Navigate in the Basic Editor to PduR\PduRBswModules\DCM
- 2. select the "Cancel Transmit" Parameter.



Part 1/4: Configure DCM and DEM with Configurator

- The result of the last step is an error in PduR\PduRBswModules\CanTp because also the lower Module needs to support the "Cancel Transmit"
- Repeat the last steps to solve the error.



Part 1/4: Configure DCM and DEM with Configurator

- As a last step, enable the "Transmit Cancellation" in the CanTp Module.
- Navigate to CanTp/CanTpGeneral
- Repeat the last steps to solve the error.

The screenshot shows two configurations in the Vector Configurator software:

Before:

- Left pane: Shows the navigation tree with nodes like Base, BswM, Can, CanIf, CanNm, CanSM, CanTp, CanTpConfig, and CanTpGeneral.
- Right pane: Shows configuration settings for CanTpGeneral. The 'Transmit Cancellation' checkbox is unselected (unchecked).

Result:

- Left pane: Same navigation tree as 'Before'.
- Right pane: Shows configuration settings for CanTpGeneral. The 'Transmit Cancellation' checkbox is selected (checked).

25

Part 1/4: Configure DCM and DEM with Configurator



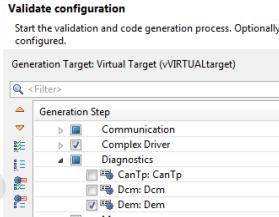
When starting the Generation Process in Configurator, the tool also generates Service Component description files for

- ComM, **DCM**, **DEM**,....

These service component files are XML files and must be imported in DaVinci Developer in order to obtain the Port Interfaces required for the service communication.

Import CDD (CANdela .cdd file)

CANdela Diagnostic Description contains all diagnostic information which can be used to configure the Diagnostic stack automatically. Importing this file triggers an Update of the whole project. Because of the Update, the ECU Extract is also imported again.



Part 1/4: Configure DCM and DEM with Configurator

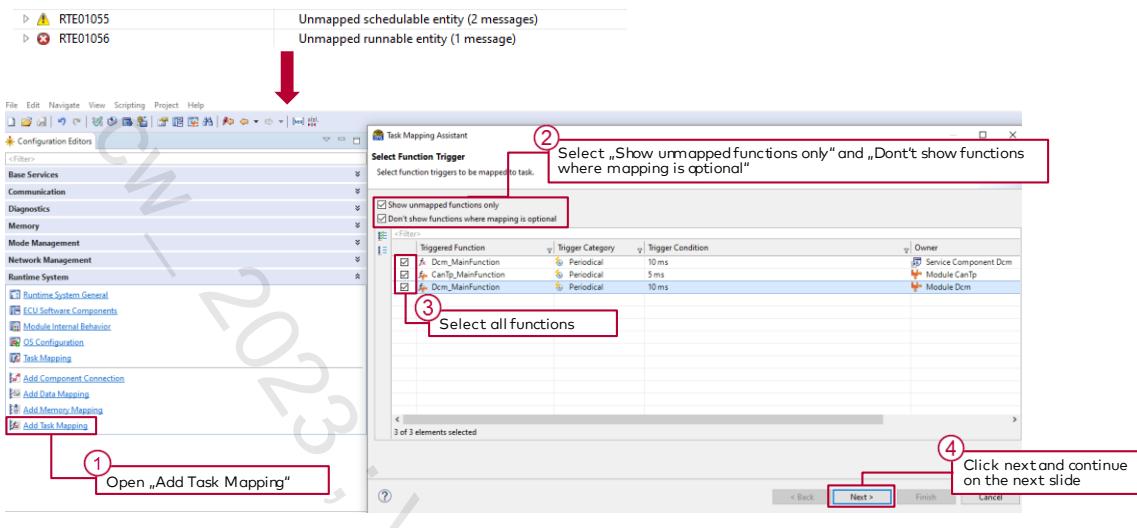


How to solve errors after updating databases:

- (1)AR-ECUC: *Invalid multiplicity. DcmDsl contains a fewer number of container DcmDslBuffer than lowerMultiplicity specifies.* Auto-solve action to **Create Container DcmDslBuffer**. This will create a DcmBuffer in Dcm->DcmConfigSet->DcmDsl->DcmDslBuffers with a preconfigured buffer size of 50.
- (2)BSWM: *Feature must be enabled.* Dcm Module was added to the configuration during update of the communication database. Therefore, the Dcm-related BswM API must be enabled in the BswM.
- (3)CANIF: *Inconsistent setting of parameter.* Auto-solve action to **Set CanIfBufferSize to 3**. The new value of 3 is needed because of an additional Tx CAN PDU added during the update of the communication database for the diagnostics response.
- (4)COMM: *Invalid handle ID configuration.* For the ComM module, execute the auto-solving action to **Delete the parameter**
- (5)DEM: *Invalid handle ID configuration.* Recalculate all handle ID values
- (6)DEM: *Checksum for NvRAM data parameters differs from value in DemCompilerPostbuildCrc.* Use auto- solve action to set the DemCompiledPostbuildCrc to the correct value.

Part 1/4: Configure DCM and DEM with Configurator

- ▶ **Task mapping** for **CANTP** and **DCM** main functions to solve the RTE01056 Error
- ▶ Go to **Runtime System** → **Task Mapping** and execute these steps (1-3)



Part 1/4: Configure DCM and DEM with Configurator

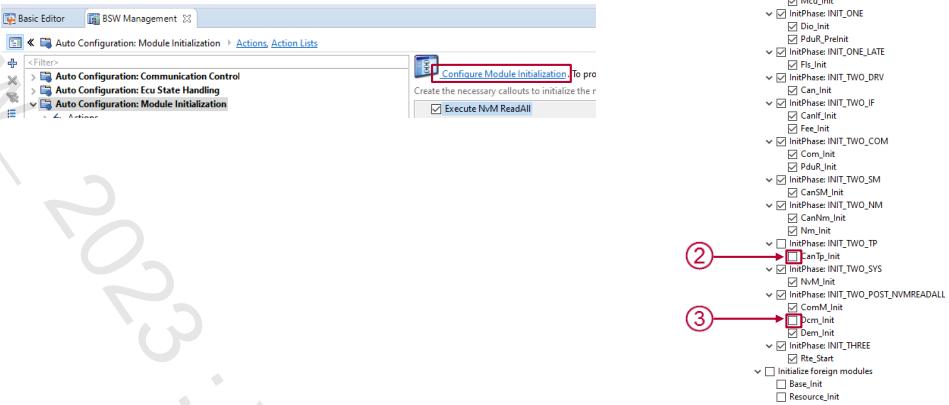
The screenshot shows two windows of the Task Mapping Assistant:

- Select Task:** Shows a tree view of tasks. A red box highlights the "SchM_Task" node under "Task". A callout with circle 1 points to this node with the text: "Select the „SchM_Task“ → all Mainfunctions are mapped to the SchM_Task".
- Order the Triggered Functions:** Shows a list of triggered functions. A red box highlights the "Finish" button at the bottom right. A callout with circle 3 points to this button with the text: "Click „Finish“ if it's necessary you can change the order of the Mainfunctions here".

Between the two windows, there is a red box with circle 2 pointing to the "Next >" button with the text: "Click „Next“".

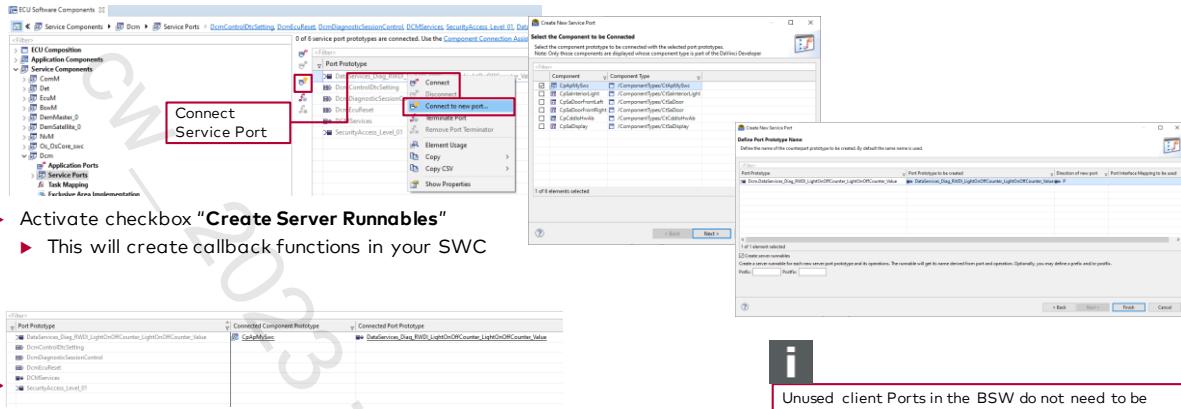
Part 1/4: Configure DCM and DEM with Configurator

- ▶ Validate
- ▶ Go to **Mode Management** → **BSW Management** → **Auto Configuration: Module Initialization**
- ▶ Click on **Configure Module Initialization** to access the BSW Module Initialization
- ▶ **Activate** the Dem_PreInit, CANTP and DCM checkboxes. Click on **Finish**.



Part 2/4: Service Port Prototypes and Access Points

- Go to **Runtime System** → **ECU Software Components** → **Service Components** and proceed to **Dcm** → **Service Ports**
- Create **Server** Port Prototype at the **CpApMySwc** for the DCM **Client** Service Port using the icon (or right-click on one of the selected Port Prototype) from the Port **DataServices_Diag_RWDI_LightOnOffCounter_LightOn OffCounter_Value**



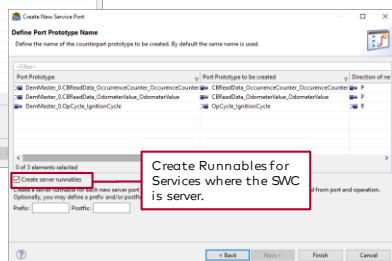
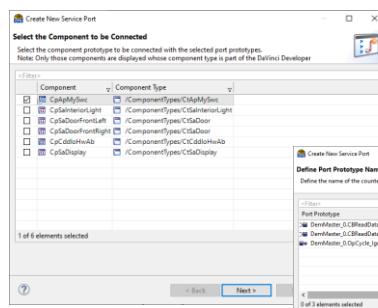
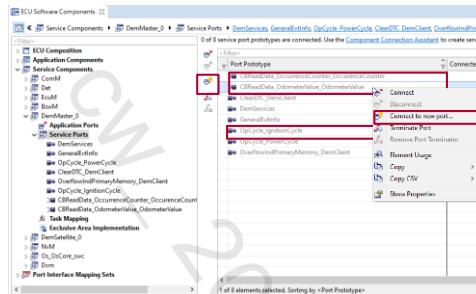
- Activate checkbox "Create Server Runnables"
- This will create callback functions in your SWC

i

Unused client Ports in the BSW do not need to be connected over service mapping. In this case the RTE will create UDS/Dcm NRCs for them.

Part 2/4: Service Port Prototypes and Access Points

- ▶ **Runtime System → ECU Software Components → Service Components** and proceed to **DemMaster_0 → Service Ports**
- ▶ Create **Server / Client** Port Prototypes at the **CpApMySwc** for the following DEM Services using the icon (or right-click on one of the selected Port Prototypes)



- ▶ Activate checkbox "Create Server Runnables" for **CtApMySwc** Server ports only
 - ▶ This will create callback functions in your SWC
- ▶ Click **Finish**

Part 2/4: Service Port Prototypes and Access Points



Creation of Server / Client Port Prototypes:

The procedure how to create the Server / Client Port Prototypes for the Dcm service component is the same as for the Dcm Service Component. This was described in detail on the previous page.

Creation of Server Runnables

*Please note: It depends on the **Standardized AUTOSAR Interface** whether the Service Module offers a Client(Server) and the SWC has to offer a Server(Client) Port Prototype.*

Server Runnables can automatically be created together with the Server Port Prototypes by DaVinci Configurator (as described before) or can be manually created within DaVinci Developer.

You can go to the Runnables dialog and create **New → Server Runnables....**. In case you have to implement **Server Port Prototypes** in your SWCs, you have to implement various Runnables which are triggered by "**On Operation Invocation**" by the Service Module(s). This can be accomplished after having added the **Service Port Prototype** in the correct orientation for the SWC Type.

Part 2/4: Service Port Prototypes and Access Points

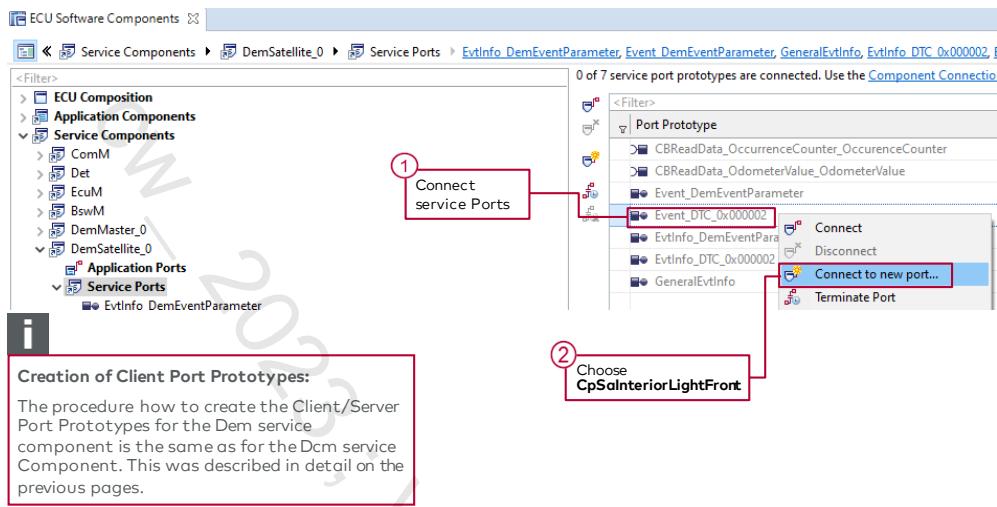
- Go to **Service Ports** of the **DemSatellite_0** Service Component
- Create **Client** Port Prototype for **CpSaInteriorLightFront** for the following DEM Services using the icon  from the Port Event_DTC_0x000002

ECU Software Components

Service Components

Port Prototype

Creation of Client Port Prototypes:
The procedure how to create the Client/Server Port Prototypes for the Dem service component is the same as for the Dcm service Component. This was described in detail on the previous pages.

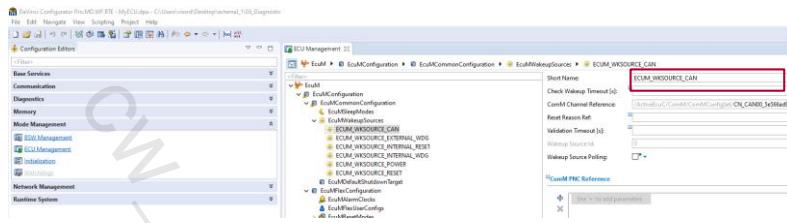


① Connect service Ports

② Choose CpSaInteriorLightFront

Part 2/4: Service Port Prototypes and Access Points

- Go to the **Mode Management** and open the **ECU Management**. Here Change the name of the Wakeup Source **CN_CAN00_5e566ad9** back to **ECUM_WKSOURCE_CAN** like in the picture below.



- **Generate BSW and SWC Templates**
- **Exit Configurator**



Creation of Client Port Prototypes:

The procedure how to create the Client/Server Port Prototypes for the Dem service component is the same as for the Dcm service Component. This was described in detail on the previous pages.

Part 2/4: Service Port Prototypes and Access Points

- ▶ "Configure SWCs" using `MyECU.dpa` file in folder `\E6_Diagnostics\`
- ▶ Define necessary Service **Access Points** for the following Runnables 

 - ▶ Go to SWC Type **CtApMySwc**
 - > Select Runnable `RCtApMySwcInit`
 - > Add an **Access Point** → **New** → **Invoke Operations...**
 - > `OpCycle_IgnitionCycle.SetOperationCycleState`
 - ▶ Go to SWC Type **CtSaInteriorLight**
 - > Select Runnable `RCtSaInteriorLightSwitchLight`
 - > Add an Access Point **Invoke Operations...** for `Event_DTC_0x000002.SetEventStatus` for DTC`0x00002`
 - ▶ Go back to SWC Type **CtApMySwc**
 - ▶ Allow access to the existing Exclusive Areas 
 - ▶ **ExArLightOnOffCounter**
 - > For the Runnables named '`DataServices_Diag_RWDI_LightOnOffCounter_*`', set access to 'Can Enter' (3 Runnables in total)
 - > We are accessing the PimLightOnOffCounter in these Runnables
 - ▶ **ExArOdometer**
 - > In Runnable `CBReadData_OdometerValue_ReadData`, set access to 'Can Enter'
 - > We are accessing the PimOdometer in this Runnable
 - ▶ Check and save **Developer** workspace and exit



Which Software Component has to do what?

CtSaInteriorLight is able to detect if the light bulb of the interior light is defective and has to report the test result for **DTC_0x000002** to the **DEM**. The additional information for the Error Event (odometer and occurrence) can be collected in the **CtApMySwc**. Furthermore, the counter for how many on/off cycles of the interior light we have observed until now can be retrieved from **CtApMySwc** as well.

Part 3/4: Generate, program Runnables and test

- ▶ Open "Configure BSW" using **MyECU.dpa** file in folder **\E6_Diagnostics**
- ▶ Because we have changed the **DEM** error memory content, we need to change the NvRamBlock size of the **DEM** blocks in **NVM**.
 - ▶ Fortunately, this is already prepared, so there is nothing to do for you here. (In a real project, you'll have to change the NvRamBlock size accordingly. Refer to the notes section and the next slide to see how it works.)
- ▶ Validate the project and generate (press F9). Press F7 to update the SWCs.
- ▶ Exit Configurator
- ▶ **Open Visual Studio Project** by double-clicking on **MyECU.sln** file in folder **\E6_Diagnostics\...\Solution**
- ▶ Build the project (press F7)
- ▶ Place a breakpoint into **Det_ReportError**
- ▶ Open CANoe
- ▶ In Visual Studio, start the debug mode by pressing F5
- ▶ Change back to CANoe, start the measurement (press F9)
 - ▶ In the case of an error you will hit your breakpoint. Inspect the such a case! You may not hit the breakpoint if



DEM NvRamBlocks

The **DEM** needs certain NvRamBlocks configured in **NVM**. If you change the configuration, i.e., add or remove DTCs and environmental data (extended data records), their size in **NVM** has to be changed manually.

call stack in

The actual demand for the **DEM** variables is checked during compilation time. In case that the size of your **NvRamBlock** does not match the size of your **NvMemoryBlock** defined in the **NvM Block** configuration a compilation error is thrown.

To solve that problem you'll have to adapt the **NvM Block Size** in dependence of the actual size of the **NvRamBlock** variable.

Part 3/4: Generate, program Runnables and test

- ▶ In Visual Studio, code up the implementation in **CtApMySwc**
 - ▶ **CtApMySWCInit**
 - > Start Operation Cycle
 - ▶ **DataServices_Diag_RWDI_LightOnOffCounter_LightOnOffCounter_Value_ReadData**
 - > Diagnostic function (server runnable) to read LightOnOffCounter (Did 0x1000)
 - > The tester wants to read out the current value (4 bytes) of the LightOnOffCounter. Fill the buffer of the function parameter using PimLightOnOffCounter. When accessing PIM protect it with Exclusive Area 'ExArLightOnOffCounter'. **See next slide**
 - ▶ **CBReadData_OccurrenceCounter_OccurrenceCounter_ReadData**
 - > Write current Occurrence Counter value to function parameter (use existing variable CtApMySWC_OccurrenceCounter) and increment Occurrence Counter.
 - ▶ **CBReadData_OdometerValue_OdometerValue_ReadData**
 - > Write current Odometer value to function parameter (use existing PimOdometer). When accessing PIM protect it with Exclusive Area 'ExArOdometer'.

Part 2/4: Service Port Prototypes and Access Points



How to write to the data buffer for reading the LightOnOffCounter?

Depending on the switch RTE_PTR2ARRAYBASETYPE_PASSING you can have different interfaces to diagnostics. The variable you write to is usually a pointer to an array type.

(„P2VAR Dcm_X_Byte_Type“, X>0) Dcm_X_Byte_Type is an array of X bytes length.

You can also have the reference to the data content directly over P2VAR Uint8, then this can be treated like an ordinary array. Use the syntax (*data)[i] to access the variable per byte. Remember to shift the right value of the expression according to the byte alignment on the left side of the expression! This depends on the bus data endianness defined in the CANdela file inside the data types which you use.

Expression to copy local variable into an array type of suitable size using a pointer cast

```
(localVarType *)Data = localVar;
```

Expression to copy array data (P2VAR) of the suitable size into a local variable of the type localVarType using a pointer cast and dereferencing.

```
localVar = *(localVarType *)Data;
```

Expression to copy a local variable into array (P2VAR) using memcpy routine.

```
memcpy(Data, &localVar, sizeof(localVar));
```

Occurrence Counter

Usually, the DEM offers a built-in occurrence counter functionality so that it is not required to implement this manually. Here we do it for exercise reasons!

Part 3/4: Generate, program Runnables and test

- ▶ Go to the **CtSaInteriorLight**
 - ▶ Implement **RCtSaInteriorLightSwitchLight**
 - > Depending on the defect state, set DTC_0x000002 event status either to DEM_EVENT_STATUS_PASSED or to DEM_EVENT_STATUS_FAILED.
- ▶ Build the project (press F7)
- ▶ Open CANoe
- ▶ In Visual Studio, start the debug mode by pressing F5
- ▶ Change back to CANoe, start the measurement (press F9)
- ▶ Test your code
 - ▶ Send Diagnostic Requests using the Diagnostics Console in CANoe!

For diagnostics the bus must be running. Please open a door, so the bus will not fall asleep.

The DCM normally also requests the bus using its own user API ComM_DCM_ActiveDiagnostics and releases the communication over ComM_DCM_InactiveDiagnostics. The COMM request only holds for

- **in default session:** The duration between diagnostic request and diagnostic response.
- **all other sessions:** The duration the DCM stays in this session. The Tester Present service will keep the communication requested.



Part 4/4: Add-On I

- ▶ Diagnostic function to write **LightOnOffCounter** (DID 0x1000)
 - ▶ The tester wants to write the current value (4 bytes) of the LightOnOffCounter. Use the buffer of the function parameter.
 - ▶ In CtApMySwc, go to the Runnables 
 - ▶ For Runnable DataService_Diag_RWDI_LightOnOffCounter_LightOnOffCounter_Value_WriteData
 - > Adjust the **Access Point** → **New** → **Invoke Operations**
 - > Select PpNvMLightOnOffCtApMySWC.WriteBlock
 - > Adjust the **Access Point** → **New** → **Write Data**
 - > Select PpDisplayState.DeLightOnOffCounter
 - ▶ Add your implementation in order to write the **LightOnOffCounter**
 - > Send it to the Display SWC and request a write to NVM for the block

Diagnostics

Finished