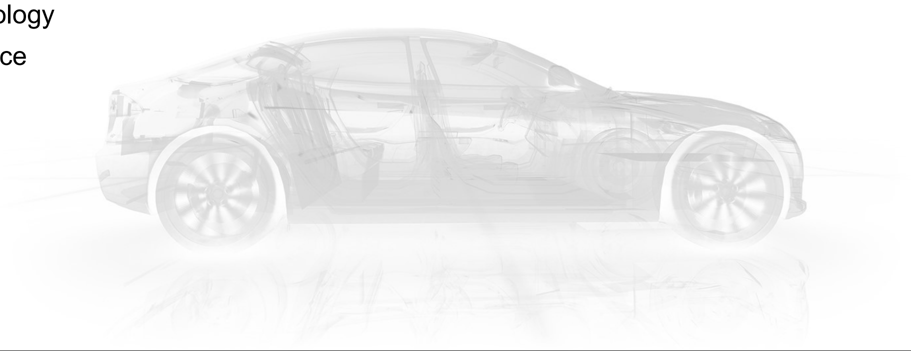
NIT

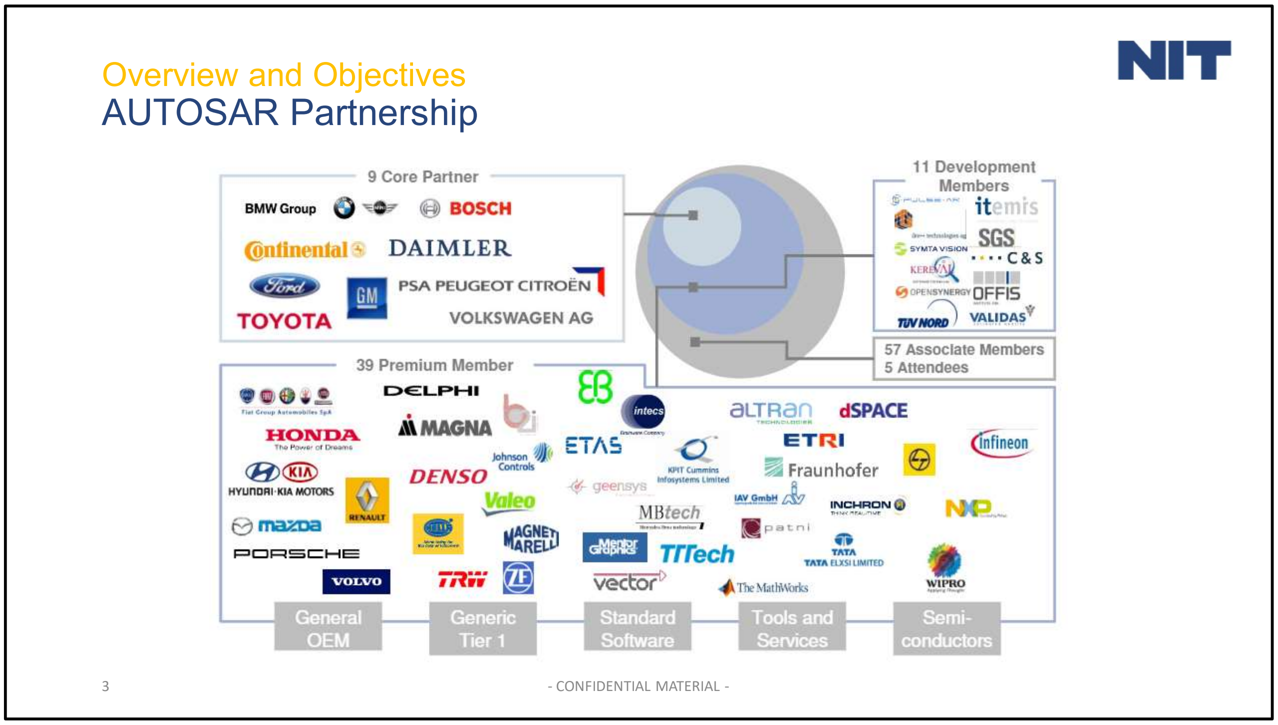
AUTOMOTIVE SOFTWARE WITH AUTOSAR

AUTOSAR

Agenda

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




NIT

Overview and Objectives
AUTOSAR Partnership

3 - CONFIDENTIAL MATERIAL -

Automotive Open System Architecture (AUTOSAR) is global development partnership of different parties in automotive (OEM companies, companies that develop tools, standard software, semiconductors, etc.). It was founded in 2003 with the goal of establishing an open standard for electrical/electronic architecture in the automotive industry.

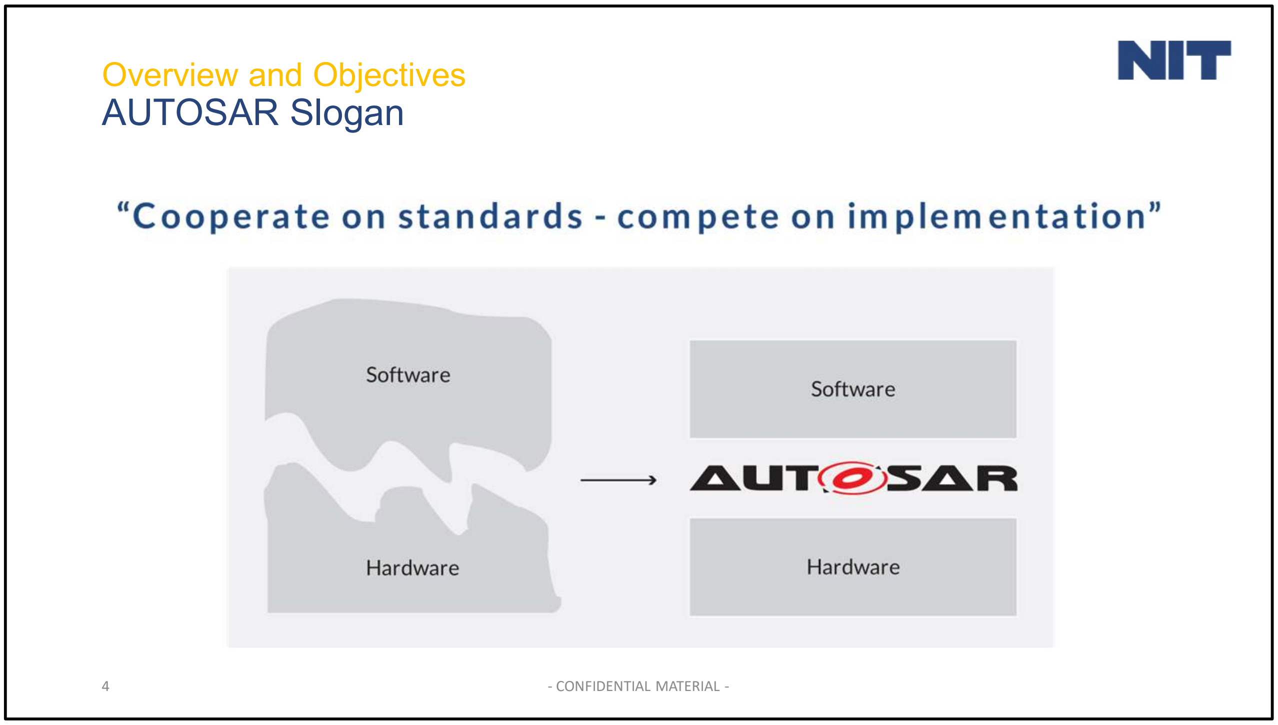
Standard Software/Basic Software consists of modules that provide services like communication, memory management and diagnostics to upper software layers.

Electronic Control Unit (ECU) implements one specific functionality (i.e., powertrain control, body control, ADAS).

Vehicle (System) contains large number of ECUs connected with vehicle bus (i.e., FlexRay, Ethernet, Can).

Original Equipment Manufacturer (OEM) is company that manufactories and sells vehicles to end customers (i.e., BMW, Daimler, Toyota, VW, etc.).

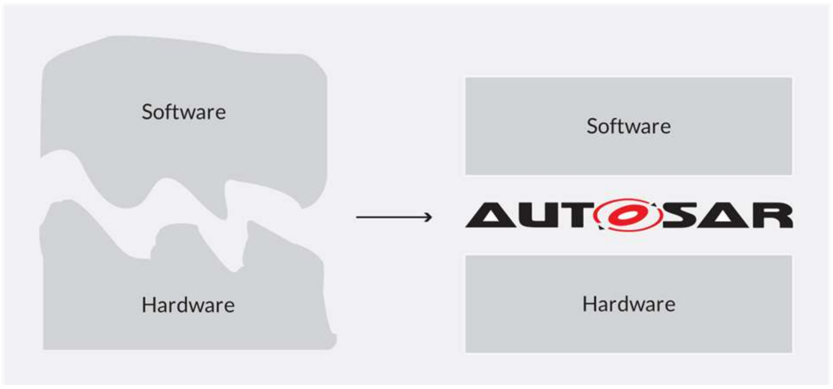
Tier 1 suppliers are companies that supply ECU or vehicle subsystem to OEMs.



Overview and Objectives

AUTOSAR Slogan

“Cooperate on standards - compete on implementation”



AUTOSAR puts emphasize on slogan "Cooperate on standards, compete on implementation". Since every CAN controller needs CAN driver, it makes little sense for each OEM to have its own implementation of it, since it does not give any competitive advantages. In contrast, competition is desired for applications on upper layers since those applications can give competitive advantages and are visible to the end users.

In the past, ECU software was highly dependent on hardware. ECUs usually had to be redeveloped from scratch whenever there were additional features, new controllers, different OEMs or modified architectures. Main goal of AUTOSAR is to create modular and hardware independent applications. ECU suppliers can, for example, develop applications that can be reused for different OEMs. On the other hand, OEMs can more easily port functions over different vehicle platforms.

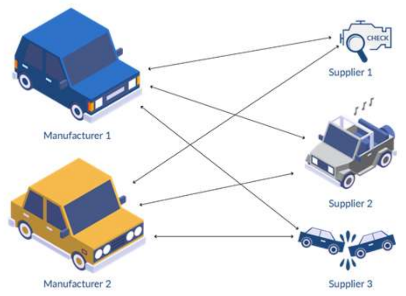
Overview and Objectives
The situation today

› Growing complexity of software in modern vehicles (driver assistance, infotainment, over the air programming, etc.)

› The life cycle of car is usually longer than that of the ECUs (need for sustainable spare parts, especially software maintenance)

› Many different hardware platforms are used, need for greater hardware abstraction, code modularity and reusability

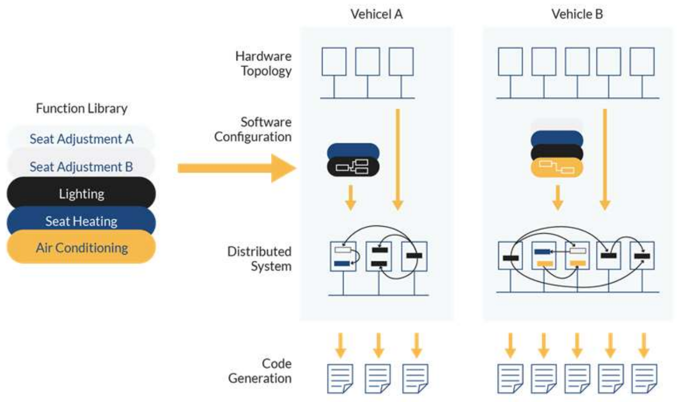
› Suppliers shall support different OEMs and vehicle platforms with their software



Overview and Objectives

AUTOSAR Objectives

NIT



On picture above you can see **reusability** of functions over different vehicles.

Runtime Environment (RTE) - configurable “middleware” that abstracts applications from basic software.
RTE implementation is generated for each ECU.

If functions are using standardized AUTOSAR interfaces, they can be easily reused on different ECUs or vehicle systems (platforms).

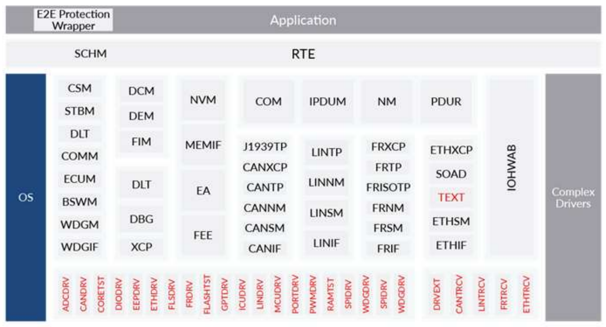
Overview and Objectives
Objectives

AUTOSAR

AUTOmotive **O**pen **S**ystem **AR**chitecture

Standardization

- of interfaces
- of exchange formats
- of methodology



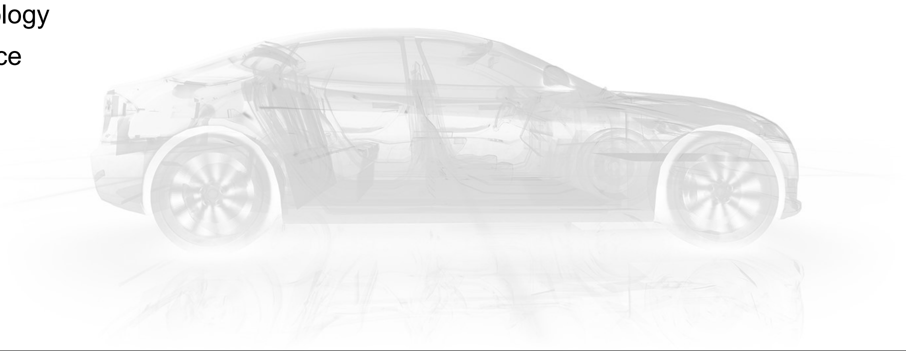
Standardization of interfaces between the application and basic software enables abstraction of the application from the hardware and smooth integration of functional modules.

Standardization of exchange formats allows easier information exchange (between different parties, OEM and Tier 1) and configuration/generation tools development (most of source code in modern cars is generated with different tools).

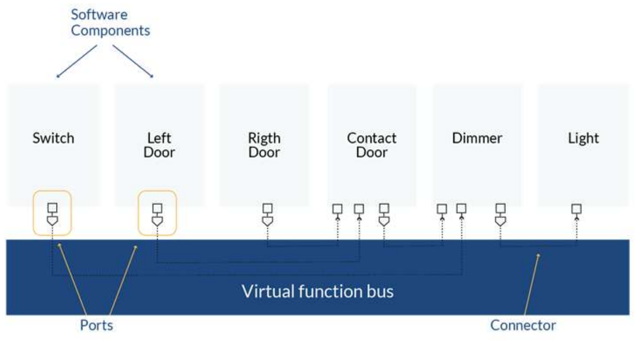
Methodology describes steps of process from system-level configuration to the generation of an ECU executable. It describes inputs, outputs and activates for different tasks of system development.

Agenda

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



AUTOSAR Application
Communication



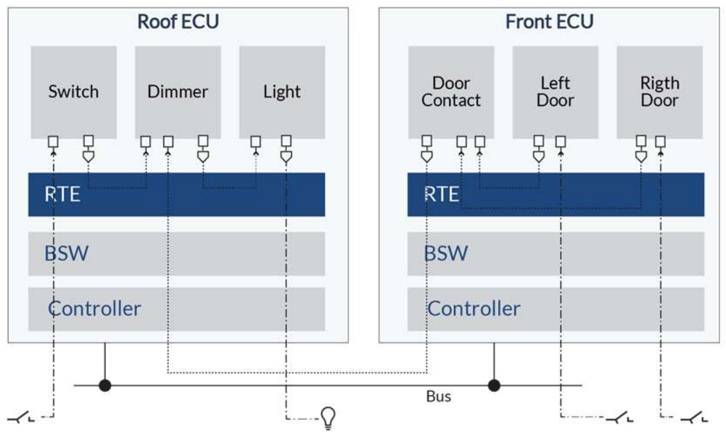
At the beginning, a system is only described by its functionality. Sub-functionalities are distributed to Software Components (SWCs).
These SWCs exchange information via ports.

The "Virtual Function Bus" symbolizes the communication between the SWCs.
At this point in time, allocation of SWCs to ECUs is still undefined.
The VFB represents communication within an ECU as well as between ECUs.
SWC has no knowledge of the underlying technologies. This makes it possible for SWC to be developed mostly independent of the hardware.

NIT

AUTOSAR Application

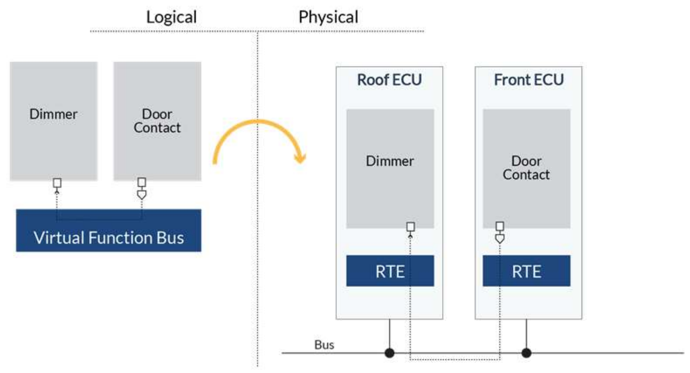
Distribution of components



After SWCs are defined, they are distributed to the relevant ECUs (mapping of SWCs to ECUs).

AUTOSAR Application

Distribution of components



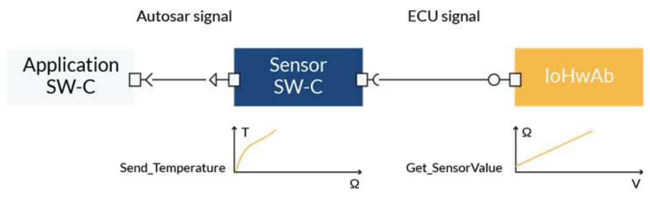
The VFB (Virtual Function Bus) is implemented with the help of an ECU-specific RTE.
The RTE implements communication between the components.
The RTE is generated for each ECU.

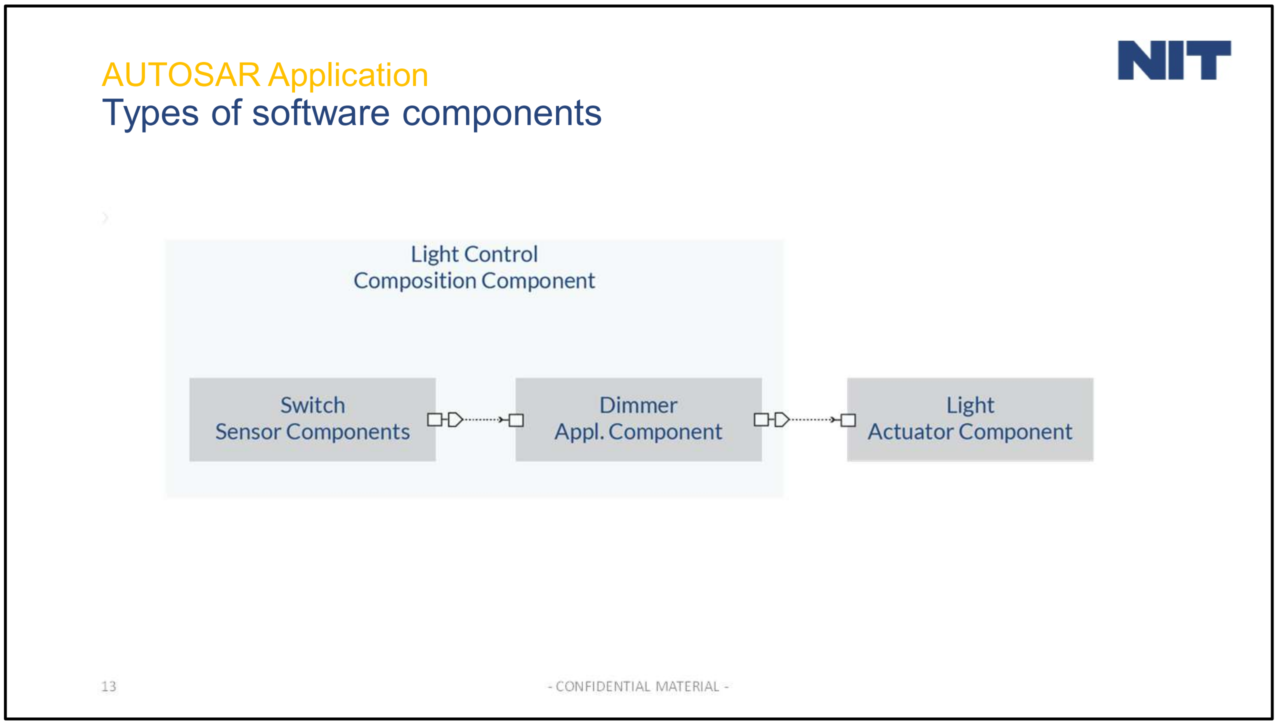
<span style="color:#FFC000">AUTOSAR Application</span>

Types of software components

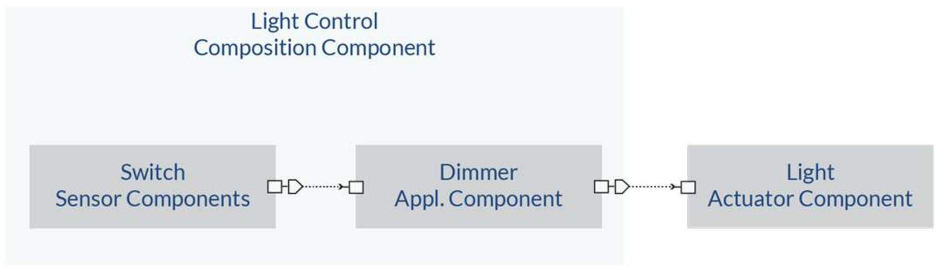
Atomic component (cannot be further subdivided)

- Application
  - Control application, that implements control algorithm
  - Independent of ECU
- Sensor/actuator
  - Prepares I/O data for control application
  - Dependent of ECU, due to dependencies to physical hardware (sensor or actuator)





AUTOSAR Application
Types of software components

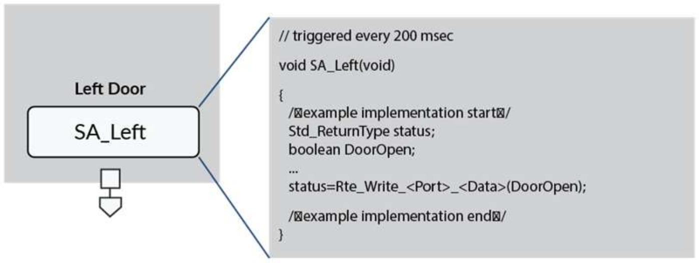


Composition is logical organization or encapsulation of SWCs (Atomic and/or Compositions).

Compositions just serve to provide structure and do not contribute any additional functionality. They do not exist in the source code.

AUTOSAR Application

Runnables



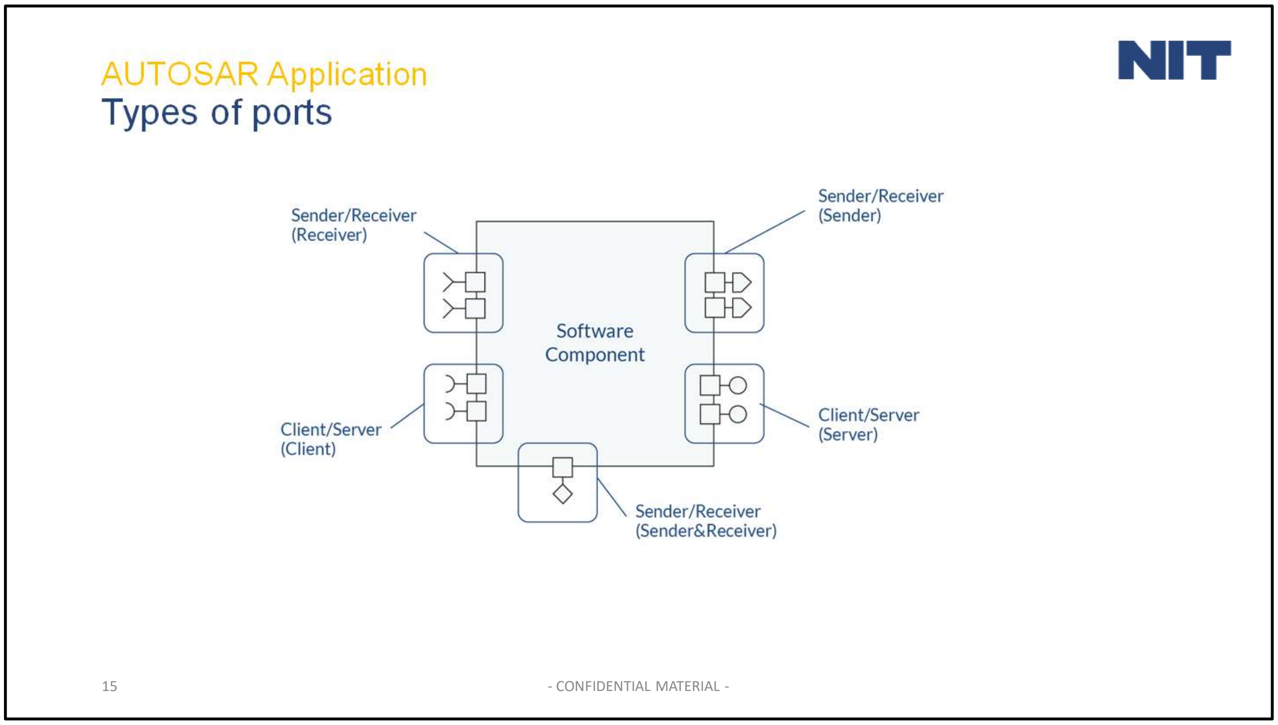
```
// triggered every 200 msec
void SA_Left(void)
{
  /⊠example implementation start⊠/
  Std_ReturnType status;
  boolean DoorOpen;
  ...
  status=Rte_Write_<Port>_<Data>(DoorOpen);
  /⊠example implementation end⊠/
}
```

Runnable entity is the executable unit of a SWC.
SWCs can contain one or more runnable entities, and SWC functionality is implemented in them.

Runnables are called when specific trigger occurs.

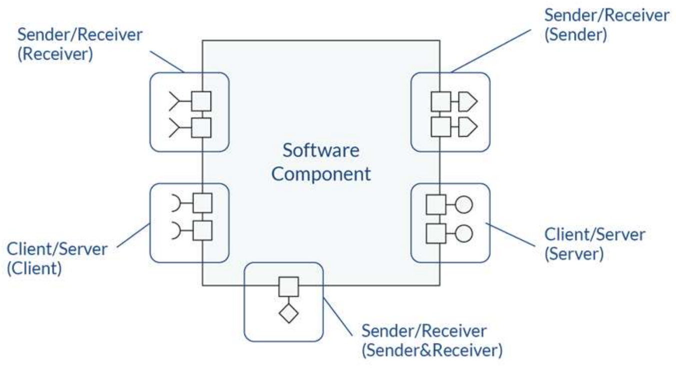
Trigger can be:

- Init – when ECU is started
- Cyclic - for example, every 200ms runnable will be called
- On operation invocation - client runnable triggers Server runnable execution
- On data received – specific data is updated by other SWC
- On mode change - ECU enters or exists specific mode

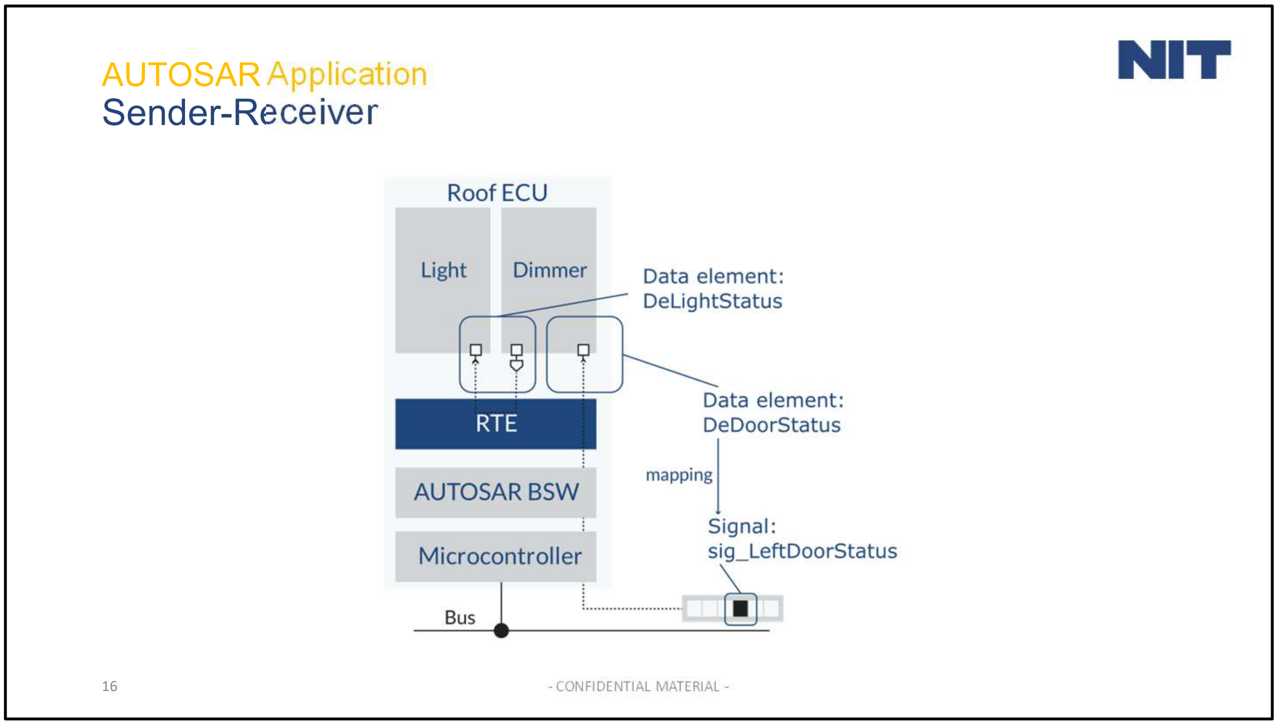


AUTOSAR Application

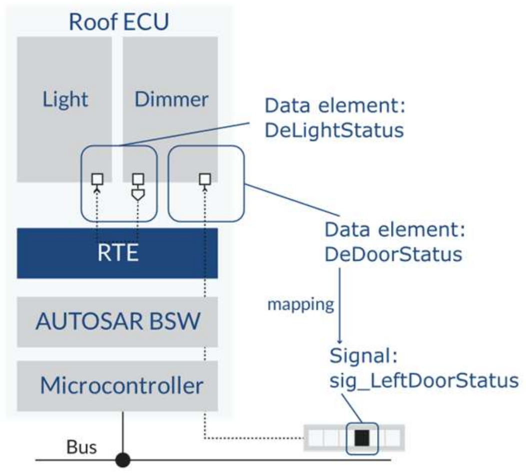
Types of ports



Ports
- SWCs are connected via ports, based on which corresponding interfaces for data exchange will be created
- Ports can be either Sender/Receiver or Client/Server
- Sender/Receiver ports contain data elements that are described via data types
- Client/Server ports contain operations with arguments
- Ports are connected via assembly connectors



AUTOSAR Application
Sender-Receiver



16 - CONFIDENTIAL MATERIAL -

Sender-Receiver communication is used for data transmission where sender writes data into buffer from which multiple receivers can read.
The sender is not aware about number of receivers.
Inside one sender-receiver port there can be multiple Data elements which are described by data types (structure, array, or simple data type).

A data element is "mapped" to a signal if the data is sent/received over the network bus.

Writing data to buffer is done by `Rte_Write` call on sender side and read of data from buffer is done by `Rte_Read` on receiver side.
For each data element specific API is created depending on port name, direction, SWC name, etc.

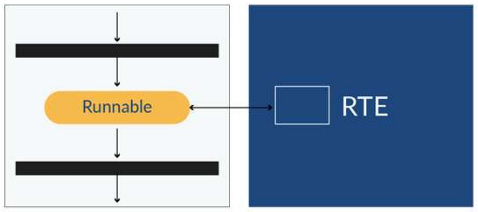
Example call from receiver side:
```
Rte_Read_<SWC name>_<Port name>_<Data element>
Rte_Read_Dimmer_PpDoorState_DeDoorStatus()
```

Example call from sender side:
```
Rte_Write_<SWC name>_<Port name>_<Data element>
Rte_Write_Dimmer_PpLightState_DeLightStatus()
```

AUTOSAR RTE

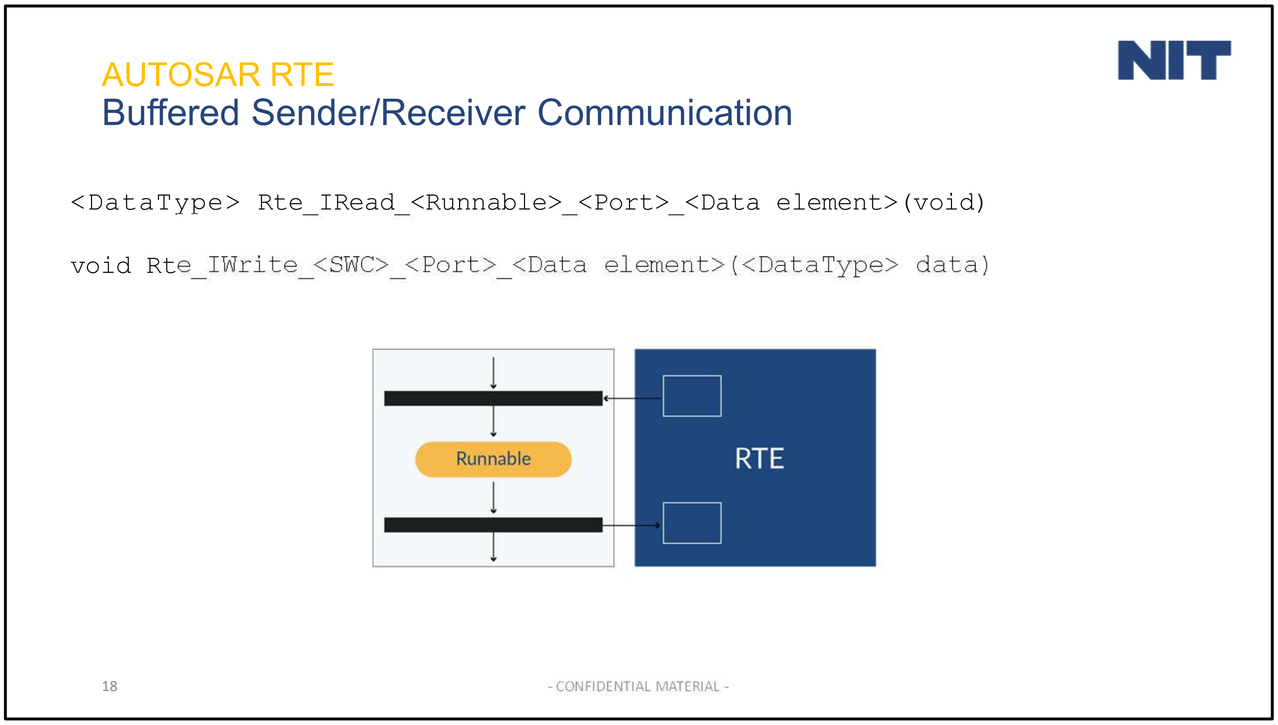
Direct Sender/Receiver Communication

```
Std_ReturnType Rte_Read_<SWC>_<Port>_<Data element>(<DataType> *data)

Std_ReturnType Rte_Write_<SWC>_<Port>_<Data element>(<DataType> data)
```



Direct sender/receiver communication:

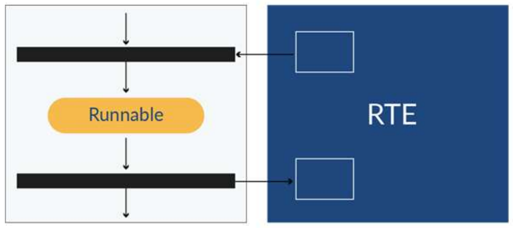
- Producer writes data into RTE buffer, receiver reads from it (direct access to buffer)
- Data written/read immediately, no additional buffer that is updated before start of runnable execution
- One producer multiple receivers, 1:n communication
- Last is best



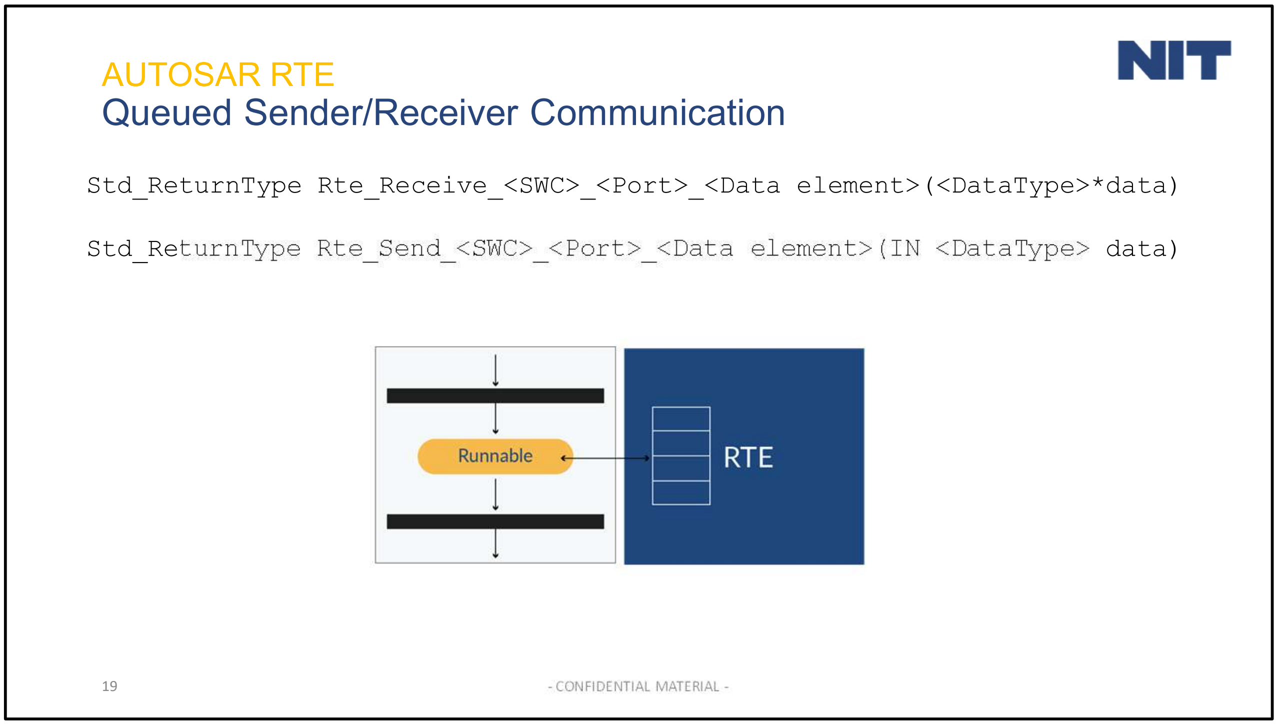
AUTOSAR RTE

Buffered Sender/Receiver Communication

```
<DataType> Rte_IRead_<Runnable>_<Port>_<Data element>(void)

void Rte_IWrite_<SWC>_<Port>_<Data element>(<DataType> data)
```



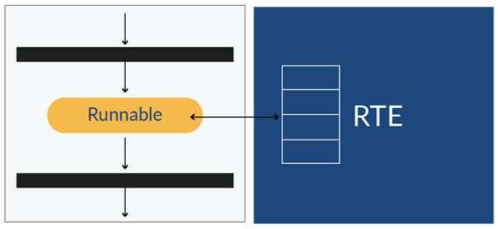
Buffered sender/receiver communication:
- Producer runnable writes data into local buffer, after producer runnable finishes, data is copied from local to RTE buffer
- Data is copied from RTE to local buffer before receiver runnable is started, receiver runnable reads from local buffer
- One producer multiple receivers, 1:n communication
- Last is best



AUTOSAR RTE

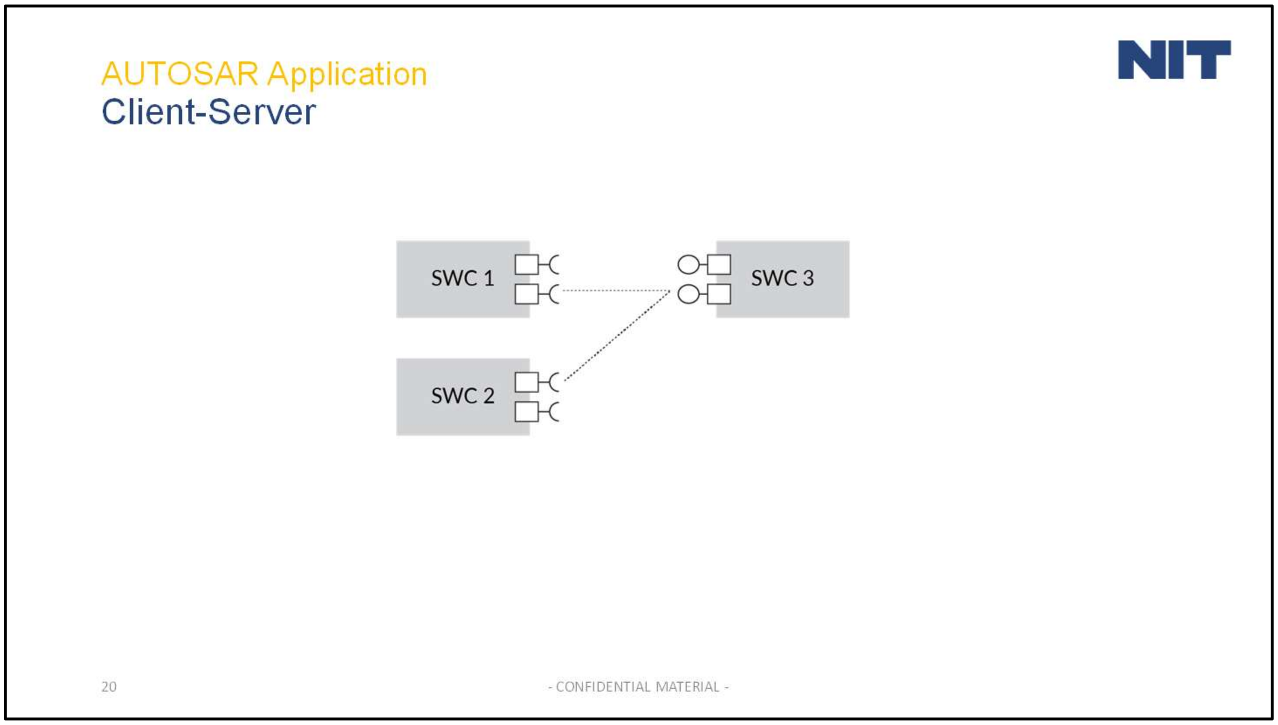
Queued Sender/Receiver Communication

```
Std_ReturnType Rte_Receive_<SWC>_<Port>_<Data element>(<DataType>*data)

Std_ReturnType Rte_Send_<SWC>_<Port>_<Data element>(IN <DataType> data)
```

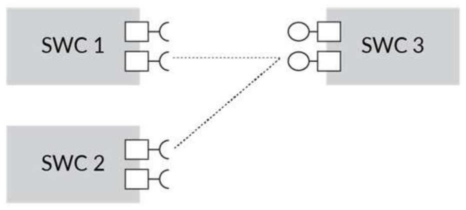


Queued sender/receiver communication:

- RTE reads from a specific receiver queue
- Length of queue could be defined (i.e., producer triggered every 20ms, receiver triggered every 40ms, queue length has to be 2, so we don't lose any data)
- Rte_Receive pops data from queue, Rte_Send pushes data to the queue
- First In First Out (FIFO)



AUTOSAR Application
Client-Server



Inside one port there can be multiple operations with input and/or output arguments described by data types. For each operation inside CS port Rte_Call API is created.
Call of Rte_Call API on Client side will trigger execution of corresponding Server runnable.

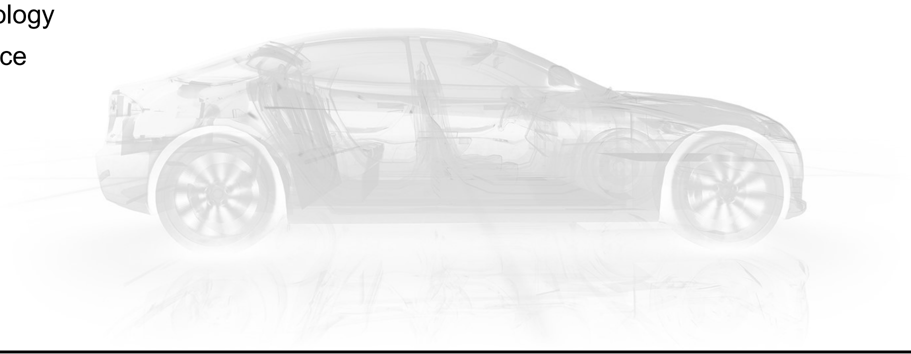
Server runnables can be triggered synchronously (server runnable is executed in context of caller-client runnable) or asynchronously (server runnable is executed in separate context).
One server runnable can be triggered by multiple client port operations.

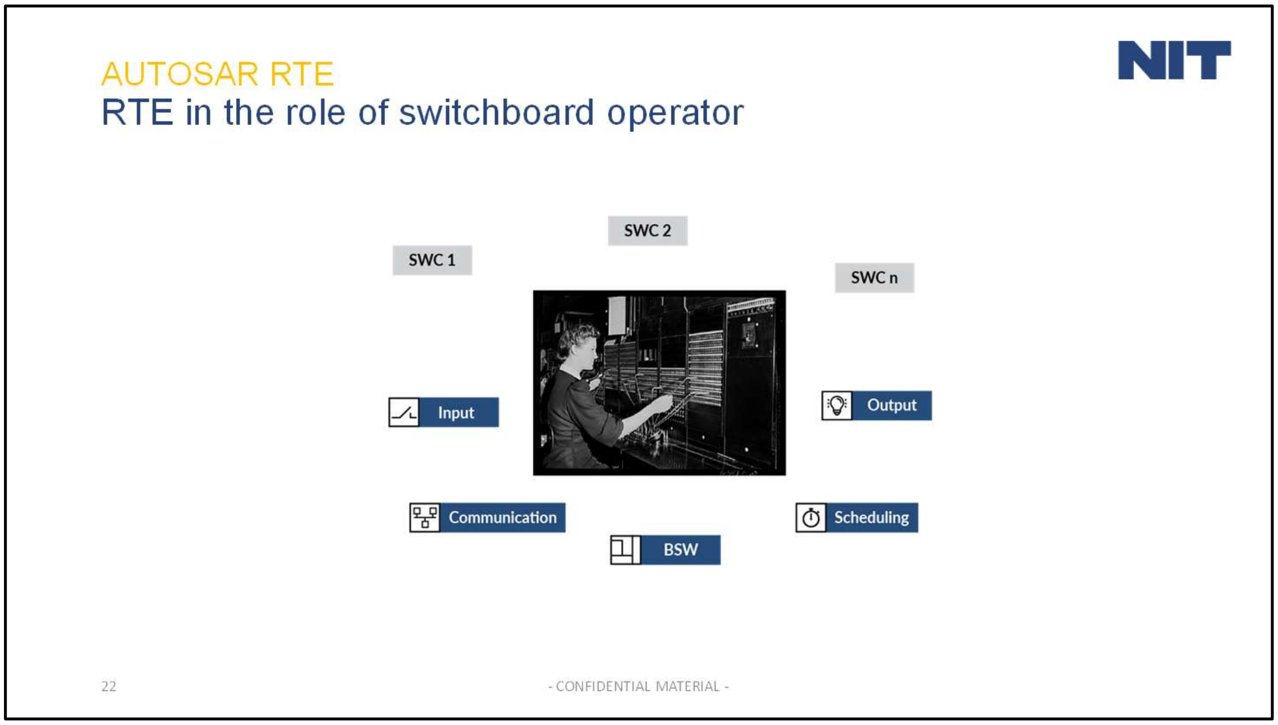
Example call from Client:

```
Rte_Call_<SWC name>_<Port>_<Operation>
Rte_Call_Door_PpDoorState_readDoorState()
```

Agenda

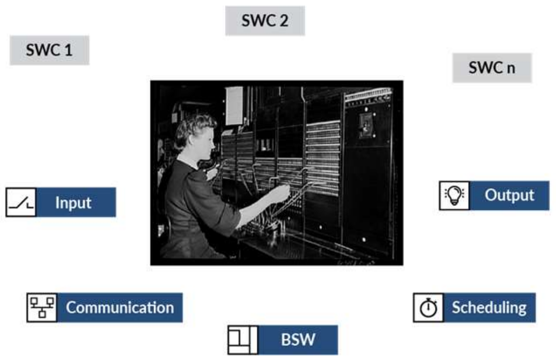
- Overview and Objectives
- AUTOSAR Application
- **AUTOSAR RTE**
- AUTOSAR BSW
- AUTOSAR Methodology
- AUTOSAR in Practice





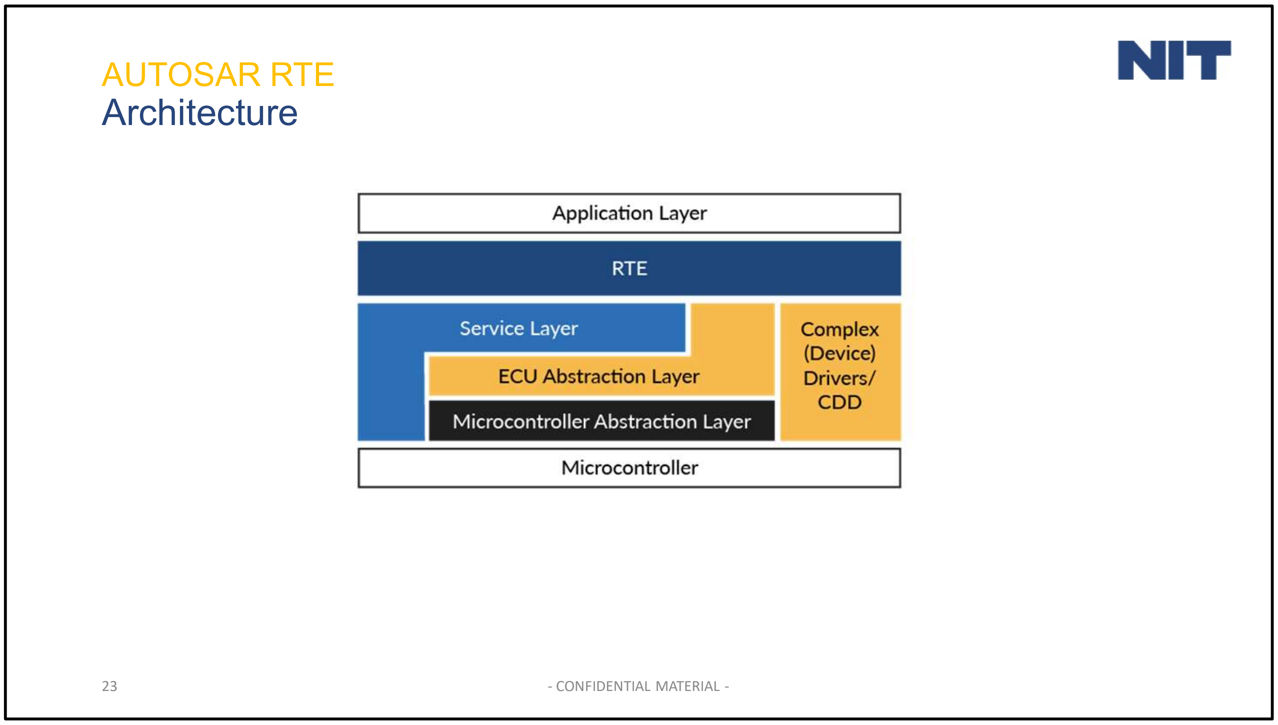
AUTOSAR RTE

RTE in the role of switchboard operator

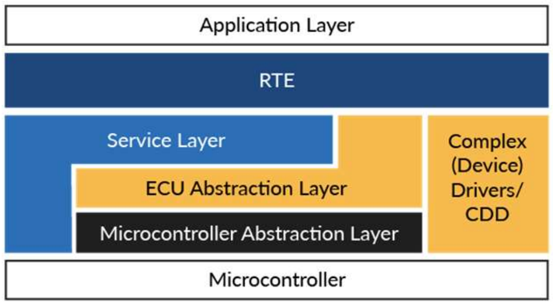


- CONFIDENTIAL MATERIAL -

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.



AUTOSAR RTE
Architecture



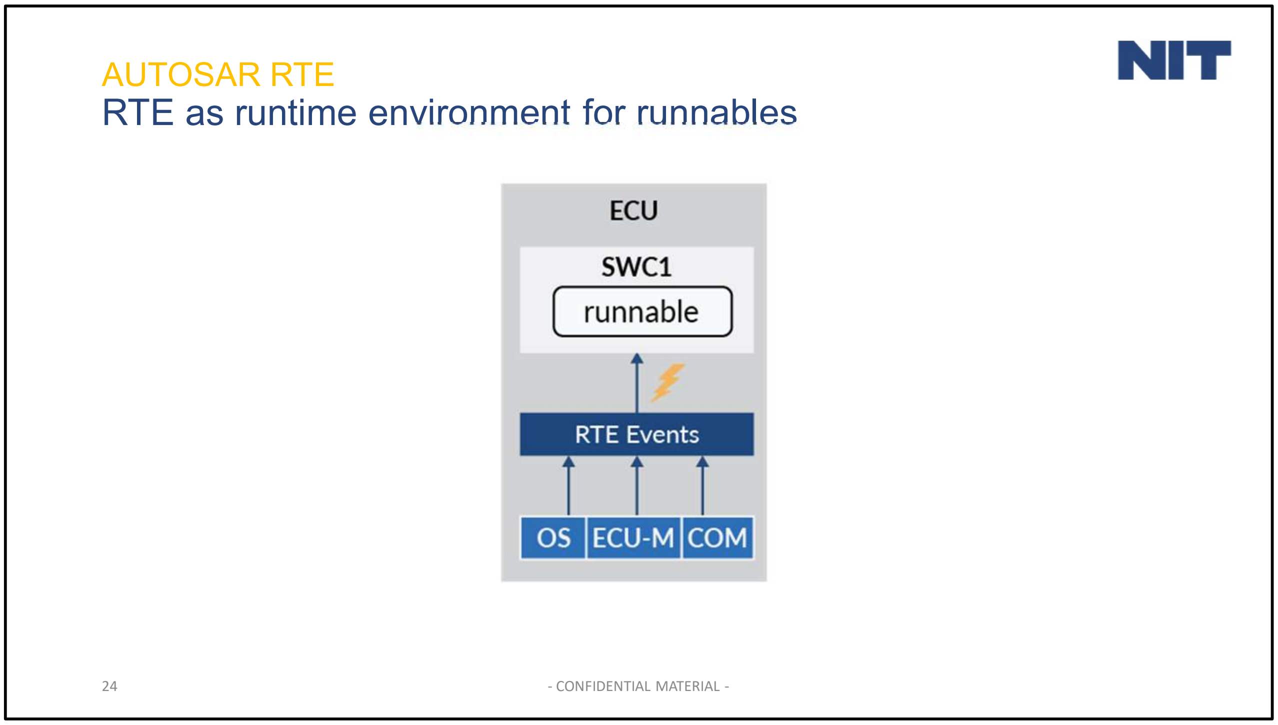
The Runtime Environment (RTE) as middleware layer that integrates applications with the basic software.
RTE enables applications to be independent of ECU.
It organizes the communication (inter/intra ECU) between SWCs, between SWC and basic software and it handles execution of runnables.

Since all interfaces in basic software are standardized, SWCs no longer needs to know how basic software operates.

The RTE offers abstractions of:

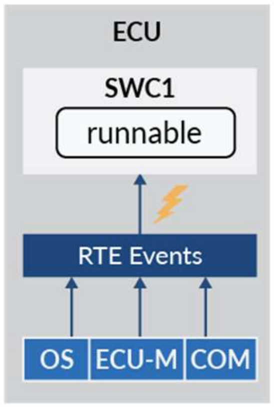
- Operating system
- Communication services
- Hardware interfaces

The RTE shifts a considerable share of development work from programming to configuration and integration.



AUTOSAR RTE

RTE as runtime environment for runnables



RTE does not have any runtime component.
RTE is generated for each ECU.
If there is change in SWC to ECU mapping or in any SWC interface, RTE must be regenerated.

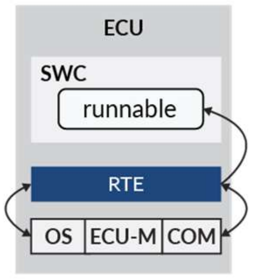
RTE is triggering runnables via RTE events

- Init
- TimingEvent
- DataReceivedEvent (S/R ports)
- DataReceiveErrorEvent (S/R ports)
- DataSendCompletedEvent (S/R ports)
- OperationInvokedEvent (C/S ports)
- AsynchronousServerCallReturnsEvent (C/S ports)
- ModeSwitchEvent
- ModeSwitchAckEvent
- ExtemalTriggerOccurredEvent
- InternalTriggerOccurredEvent
- BackgroundEvent

AUTOSAR RTE

Interaction of RTE with other layers

OS:
SetRelAlarm()
ActivateTask()
Schedule()
SetEvent()
GetEvent()
ClearEvent()
WaitEvent()
...



SWC:
```
Runnable1(){
  // Code for Runnable
}
```

COM:
Rte_COMCbk_<sigName>

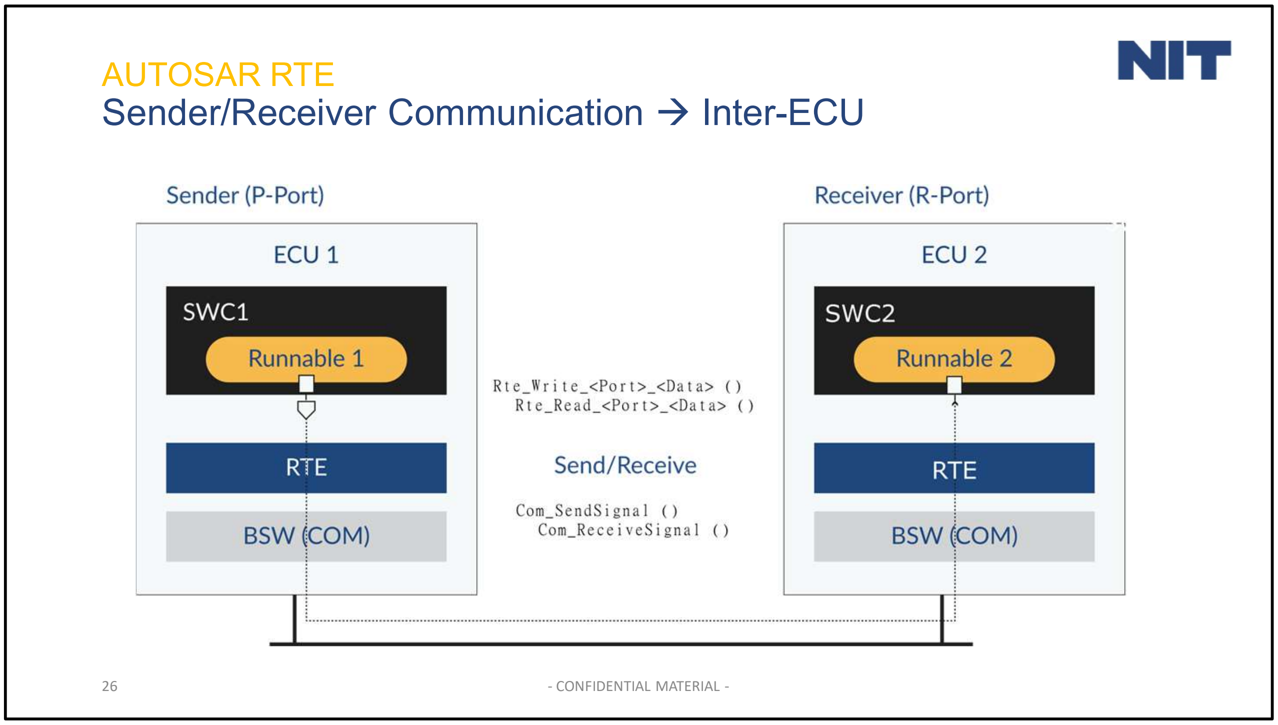
ECU-M:
Rte_Start(), Rte_Stop()

ECU Manager (ECUM) starts and stops RTE by calling Rte_Start() and Rte_Stop().

RTE can activate task, transition task to WAITING or RUNNING by calling specific OS APIs.
Activation of task or transition of task from WAITING to RUNNING triggers runnable execution.

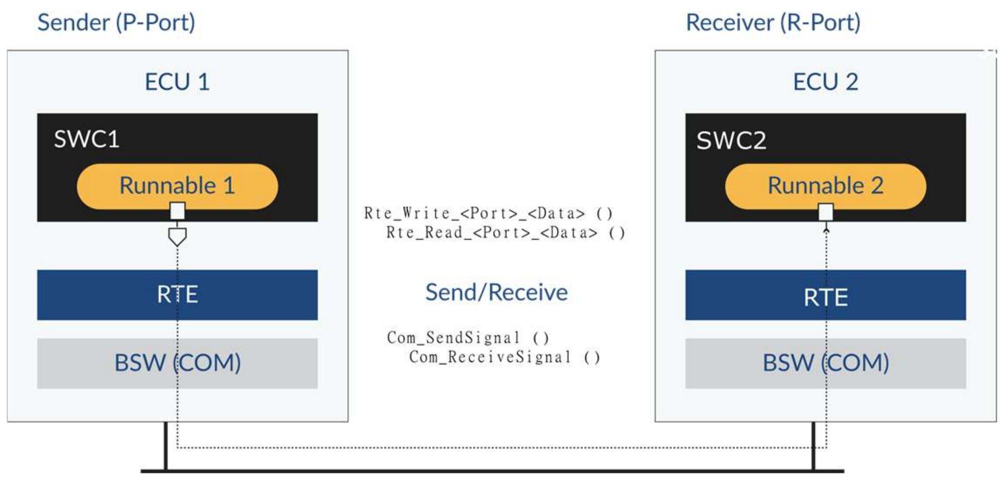
Com calls RTE callback functions on signal reception or successful transition.

Other RTE features:
- Mechanisms for data consistency (inter SWC, intra SWC)
- Support of platform and complex data (structures)
- Multiple instantiation of SWC types



AUTOSAR RTE

Sender/Receiver Communication → Inter-ECU



For Inter-ECU communication RTE must relay on BSW.

Sender side will call `Rte_Write_<SWC name>_<Port name>_<Data element>` from which `Com_SendSignal()` is called.
`Com_SendSignal()` will trigger transmission of signal that is mapped to Data_element.

On receiver side `Rte_Read_<SWC name>_<Port name>_<Data element>` will call `Com_ReceiveSignal()`.
`Com_ReceiveSignal()` reads signal value and forwards it to caller.

AUTOSAR RTE

Inter-ECU Sender/Receiver Communication

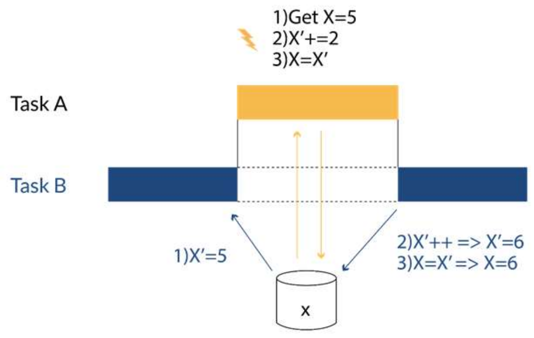
```
// triggerd every 100 msec
void RCtApMySwcCode(void)
{
  IdtDoorState    localDoorState;

  (void)Rte_Read_PpDoorState_DeDoorState(&localDoorState);

  /* Check if any door is open. */
  if (CMDOORSTATE_DOOROPEN == localDoorState)
  {
    (void)Rte_Write_PpLightState_DeLightState(CMLIGHTSTATE_LIGHTON);
  }
  else
  {
    (void)Rte_Write_PpLightState_DeLightState(CMLIGHTSTATE_LIGHTOFF);
  }
}
```

AUTOSAR RTE
Inter-ECU Sender/Receiver Communication

```
Std_ReturnType Rte_Read_PpDoorState_DeDoorState(data)
{
    Std_ReturnType ret = RTE_E_OK;
    ret |= Com_ReceiveSignal(ComConf_ComSignal_sig_StateDoor_In, (data));
    return ret;
}


Std_ReturnType Rte_Write_PpLightState_DeLightState(IdtLightState data)
{
  Std_ReturnType ret = RTE_E_OK;
  ret |= Com_SendSignal(ComConf_ComSignal_sig_InteriorLight_Out, (&data));
  return ret;
}
```

AUTOSAR RTE
Intra-SWC Communication

NIT



Problem: Communication between runnables within same SWC, which run on different tasks.

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

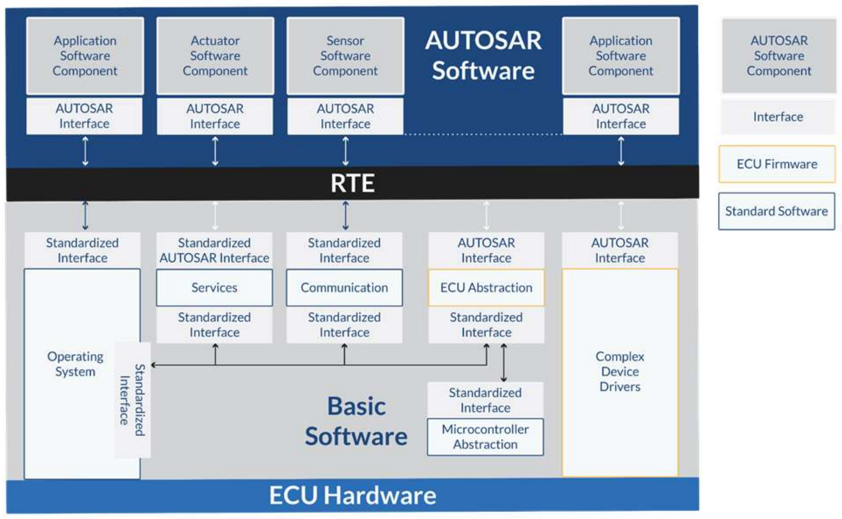
Solution:

- Exclusive Areas (EAs), Rte_Enter to enter protected area and Rte_Exit to exit protected area
- Inter Runnable Variables (IRVs) , variables can only be accessed via Rte_IrvWrite and Rte_IrvRead

Note: There are no problems in communication of different SWCs, since this kind of communication is handled by the RTE.

AUTOSAR RTE

Interfaces



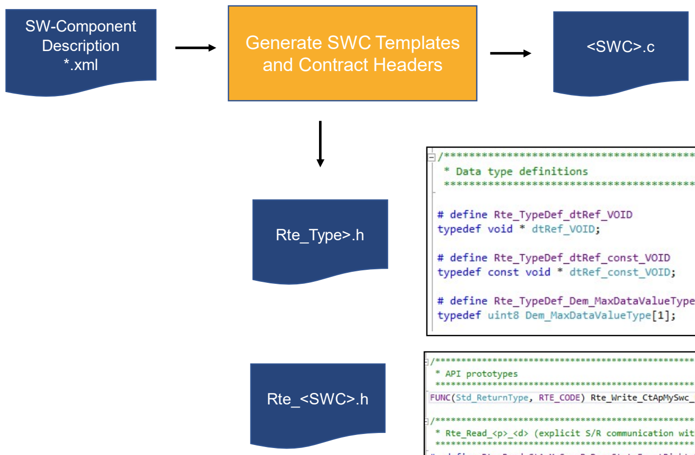
AUTOSAR interfaces are described by generic AUTOSAR, but their semantics isn’t prescribed by the AUTOSAR standard.
These interfaces are used by the application SWCs and described with application ports.
i.e., `Rte_Write_<SWC name>_<Port name>_<Data element>`

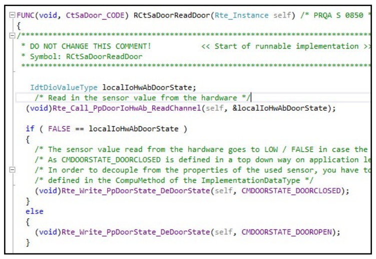
Standardized interfaces are defined by AUTOSAR standard and are implemented as direct C-APIs for optimization reasons.
i.e., `Rte_Start()`, `Rte_Stop`, `SendSignal`, `WaitEvent`, etc.

Standardized AUTOSAR interfaces are described by generic AUTOSAR, and their semantics are prescribed by the AUTOSAR standard.
They are used exclusively for communication between Application and Service SWCs (services of basic software).
They are described with service ports.
i.e., `Rte_Call_<PortPrototype>_SetEventStatus`

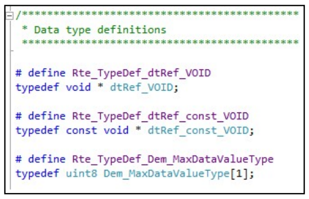
AUTOSAR RTE

SWC Templates and Contract Phase

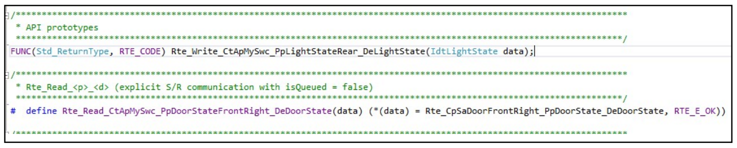

SW-Component Description *.xml
Generate SWC Templates and Contract Headers
<SWC>.c
Rte_Type>.h
Rte_<SWC>.h



```
FUNC(void, CtSaDoor_CODE) RCtSaDoorReadDoor(Rte_Instance self) /* PRQA S 0850 *
{
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << Start of runnable implementation >>
 * Symbol: RCtSaDoorReadDoor
 **********************************************************************************

   IdtDioValueType localIoHwAbDoorState;
    /* Read in the sensor value from the hardware */
  (void)Rte_Call_PpDoorIoHwAb_ReadChannel(self, &localIoHwAbDoorState);

  if ( FALSE == localIoHwAbDoorState )
  {
    /* The sensor value read from the hardware goes to LOW / FALSE in case the
    /* As CMDOORSTATE_DOORCLOSED is defined in a top down way on application le
    /* In order to decouple from the properties of the used sensor, you have to
    /* defined in the CompuMethod of the ImplementationDataType */
    (void)Rte_Write_PpDoorState_DeDoorState(self, CMDOORSTATE_DOORCLOSED);
  }
  else
  {
    (void)Rte_Write_PpDoorState_DeDoorState(self, CMDOORSTATE_DOOROPEN);
  }
```



```
/**********************************************
 * Data type definitions
 **********************************************

# define Rte_TypeDef_dtRef_VOID
typedef void * dtRef_VOID;

# define Rte_TypeDef_dtRef_const_VOID
typedef const void * dtRef_const_VOID;

# define Rte_TypeDef_Dem_MaxDataValueType
typedef uint8 Dem_MaxDataValueType[1];
```



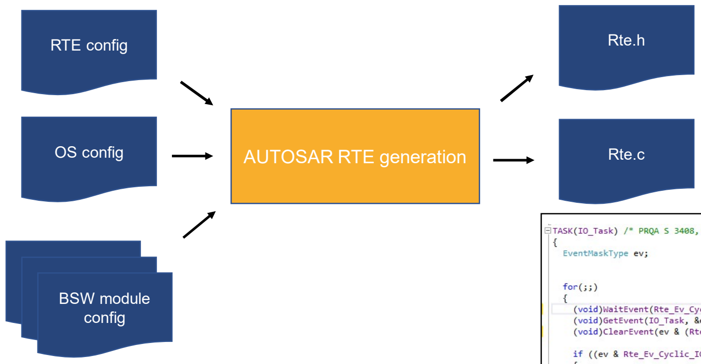
```
/**********************************************************************************************************************
 * API prototypes
 *********************************************************************************************************************/
FUNC(Std_ReturnType, RTE_CODE) Rte_Write_CtApMySwc_PpLightStateRear_DeLightState(IdtLightState data);

/**********************************************************************************************************************
 * Rte_Read_<p>_<d> (explicit S/R communication with isQueued = false)
 *********************************************************************************************************************/
#  define Rte_Read_CtApMySwc_PpDoorStateFrontRight_DeDoorState(data) (*(data) = Rte_CpSaDoorFrontRight_PpDoorState_DeDoorState, RTE_E_OK))

/**********************************************************************************************************************
```

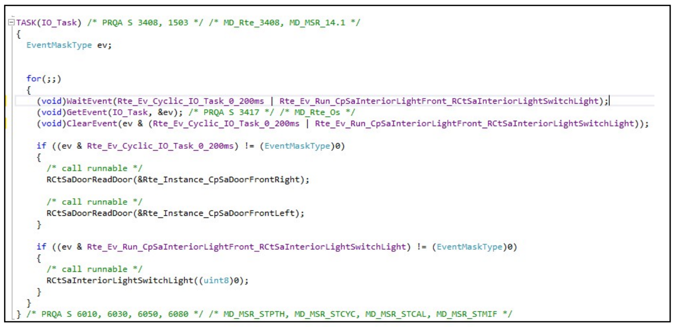
Implementation Templates are generated files that are starting point for SWC code development.

Content:
- Definitions of Runnables, only placeholder where developer will add its code
- APIs available to SWC, in comment section
- Data types and enumeration values available to SWC, in comment section
- Information about date of generation

Contract headers contain function declarations, macro definitions of all APIs available to SWC and definitions of all data types available to it.

AUTOSAR RTE
Generation Phase



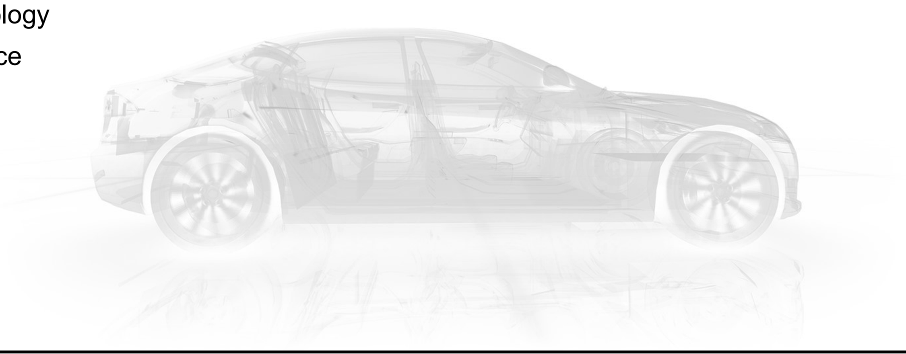


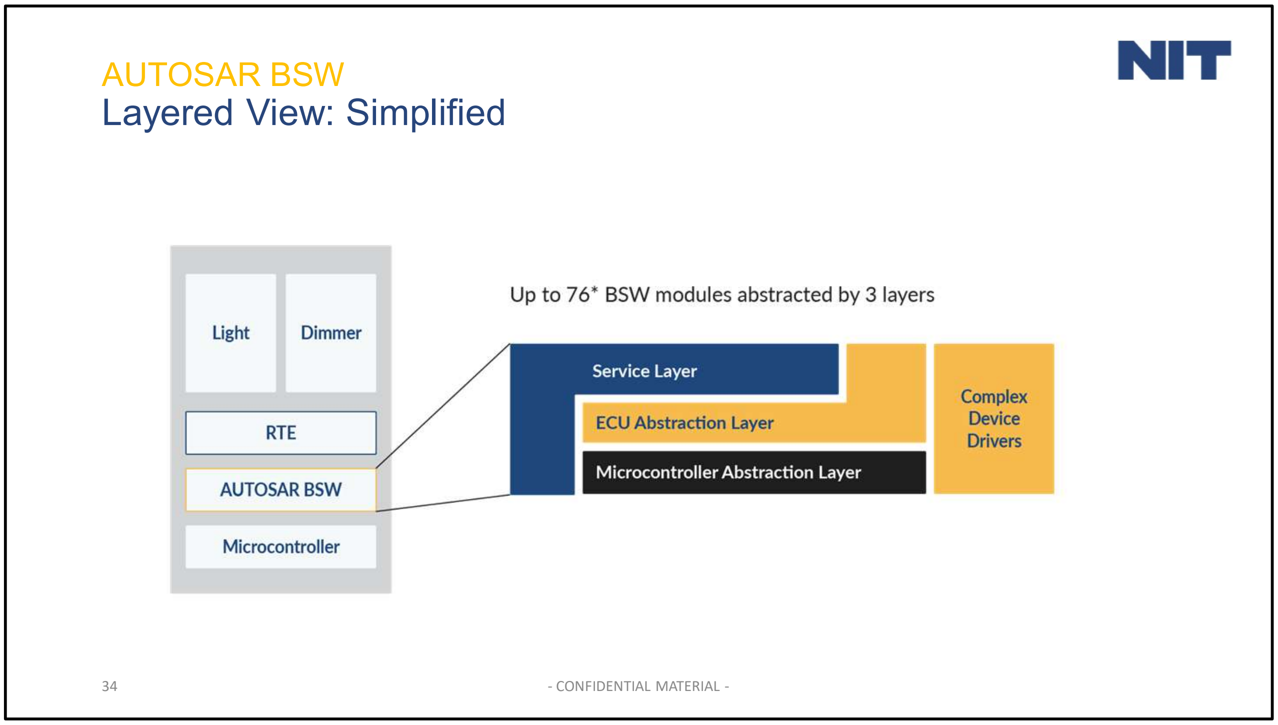
```
TASK(IO_Task) /* PRQA S 3408, 1503 */ /* MD_Rte_3408, MD_MSR_14.1 */
{
  EventMaskType ev;


  for(;;)
  {
    (void)WaitEvent(Rte_Ev_Cyclic_IO_Task_0_200ms | Rte_Ev_Run_CpSaInteriorLightFront_RCtSaInteriorLightSwitchLight);
    (void)GetEvent(IO_Task, &ev); /* PRQA S 3417 */ /* MD_Rte_Os */
    (void)ClearEvent(ev & (Rte_Ev_Cyclic_IO_Task_0_200ms | Rte_Ev_Run_CpSaInteriorLightFront_RCtSaInteriorLightSwitchLight));

    if ((ev & Rte_Ev_Cyclic_IO_Task_0_200ms) != (EventMaskType)0)
    {
      /* call runnable */
      RCtSaDoorReadDoor(&Rte_Instance_CpSaDoorFrontRight);

      /* call runnable */
      RCtSaDoorReadDoor(&Rte_Instance_CpSaDoorFrontLeft);
    }

    if ((ev & Rte_Ev_Run_CpSaInteriorLightFront_RCtSaInteriorLightSwitchLight) != (EventMaskType)0)
    {
      /* call runnable */
      RCtSaInteriorLightSwitchLight((uint8)0);
    }
  }
} /* PRQA S 6010, 6030, 6050, 6080 */ /* MD_MSR_STPTH, MD_MSR_STCYC, MD_MSR_STCAL, MD_MSR_STMIF */
```

AUTOSAR RTE generator creates Rte.c and Rte.h files.
Rte.h file contains configuration parameters of Rte.
Rte.c file contains implementations of all OS task and definitions of Rte functions.

OS Task waits for specific events, when some of them occur corresponding rupnnable is called.

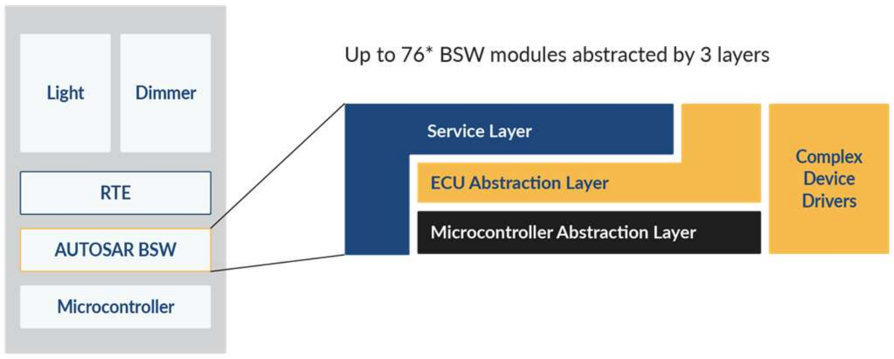
Agenda





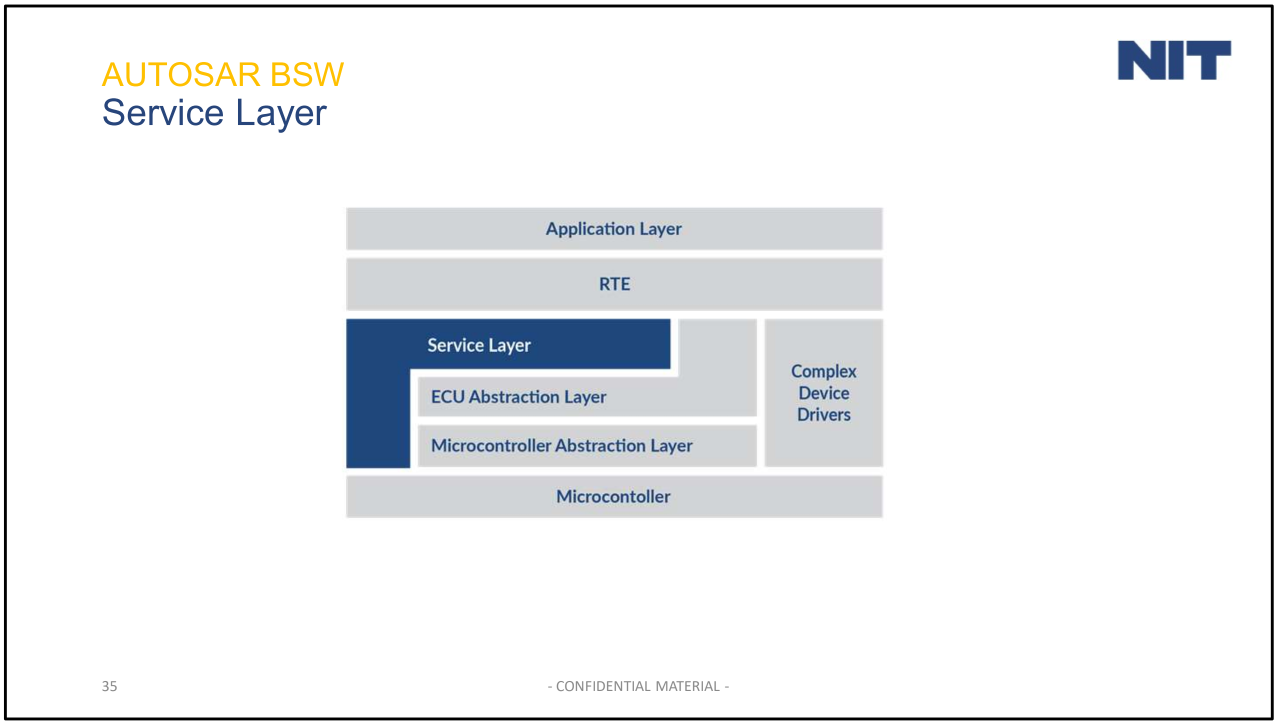
AUTOSAR BSW

Layered View: Simplified



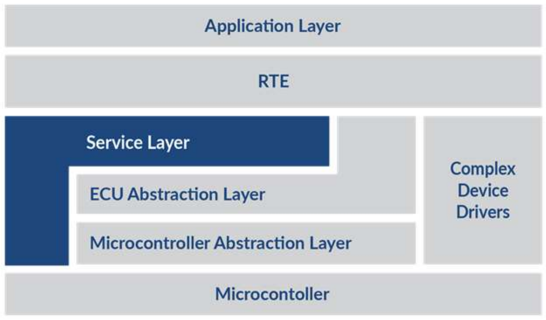
AUTOSAR BSW is subdivided into four layers:

- Service Layer
- ECU Abstraction Layer
- Complex Device Drivers
- Microcontroller Abstraction Layer (MCAL)



AUTOSAR BSW

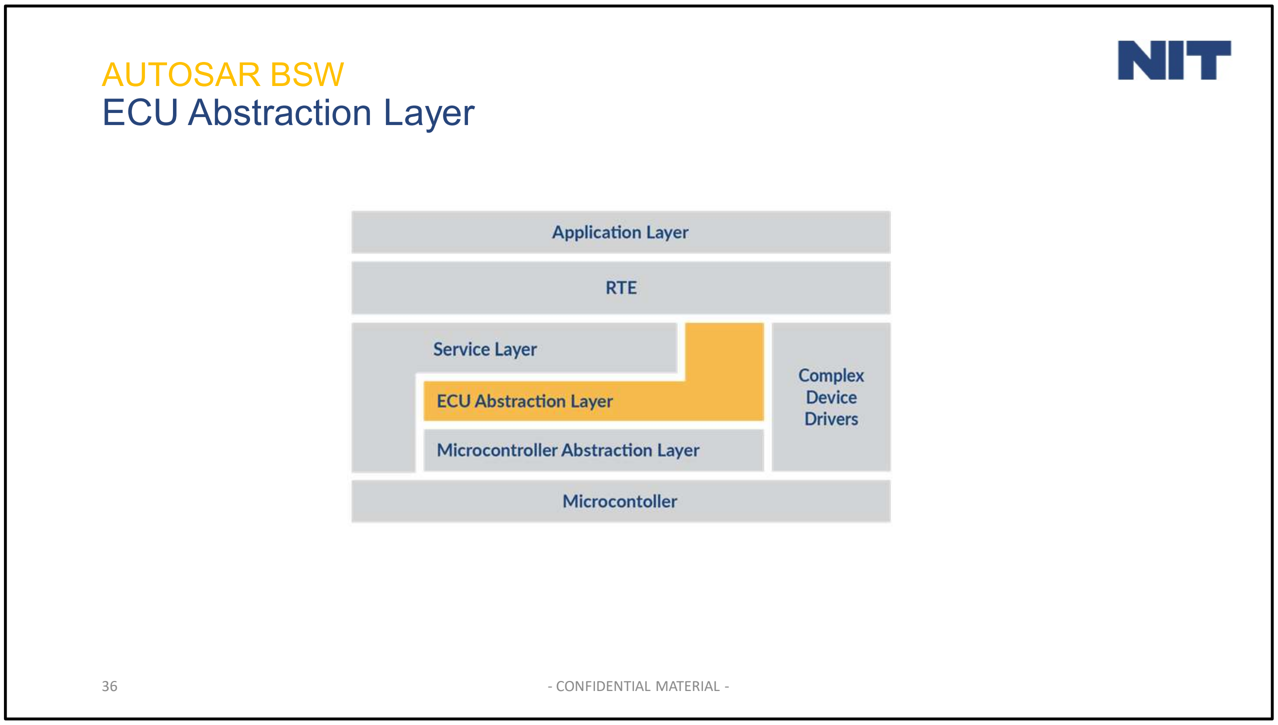
Service Layer



The Service Layer is the uppermost layer of the basic software, and it offers following services:

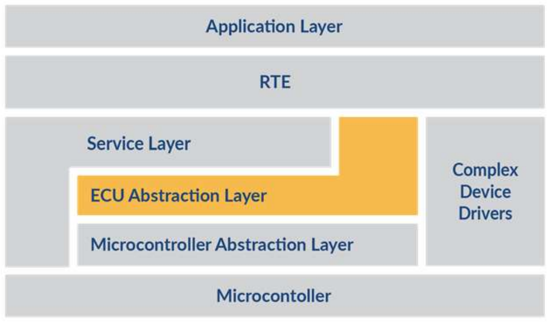
- Operating system services
- Vehicle network communication and management services
- Memory services (NVRAM management)
- Diagnostic Services (including UDS communication and error memory)
- ECU state management

Main task of the Service Layer is to provide basic services to the Application Layer (SWCs).



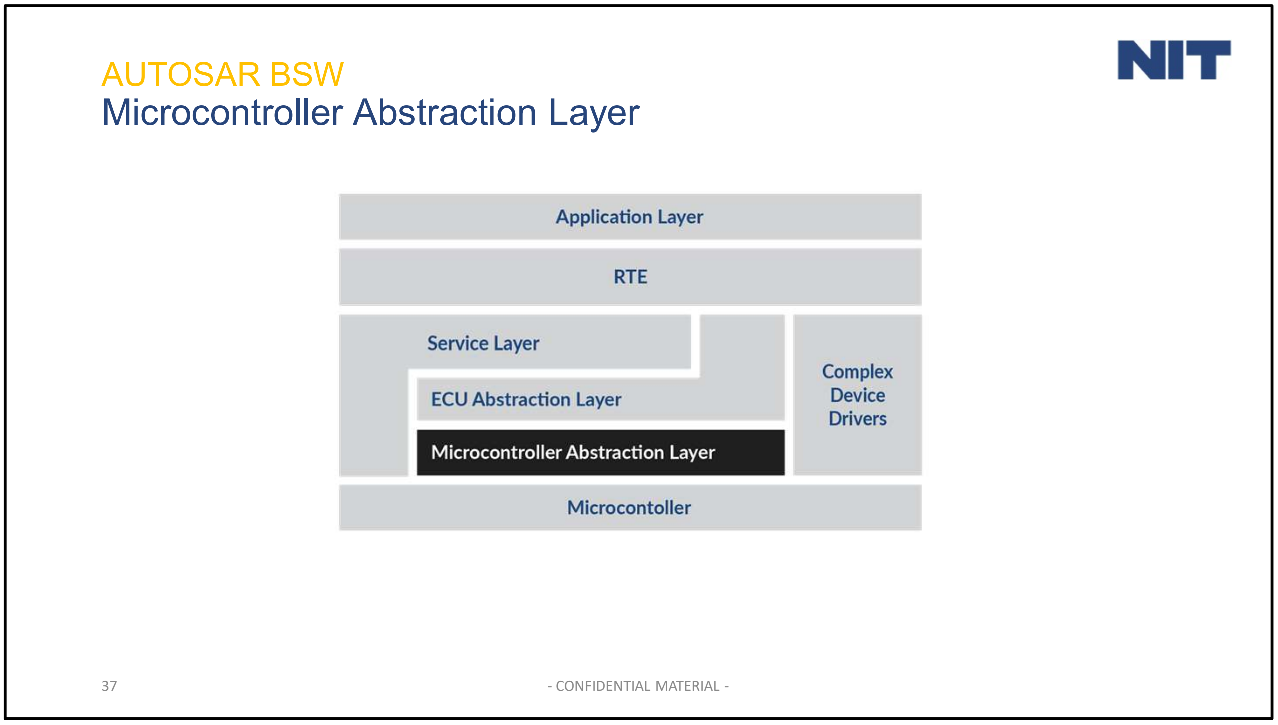
AUTOSAR BSW

ECU Abstraction Layer

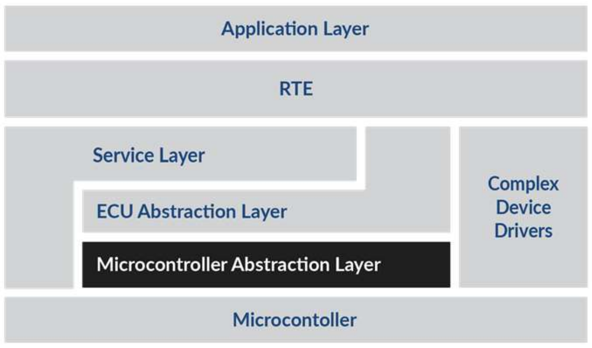


The ECU Abstraction Layer implements drivers for external devices and provides interfaces for internal and external periphery (IO, Memory, Watchdog(s) and communication).

The task of the ECU Abstraction Layer is to make higher layers independent of the ECU hardware layout.

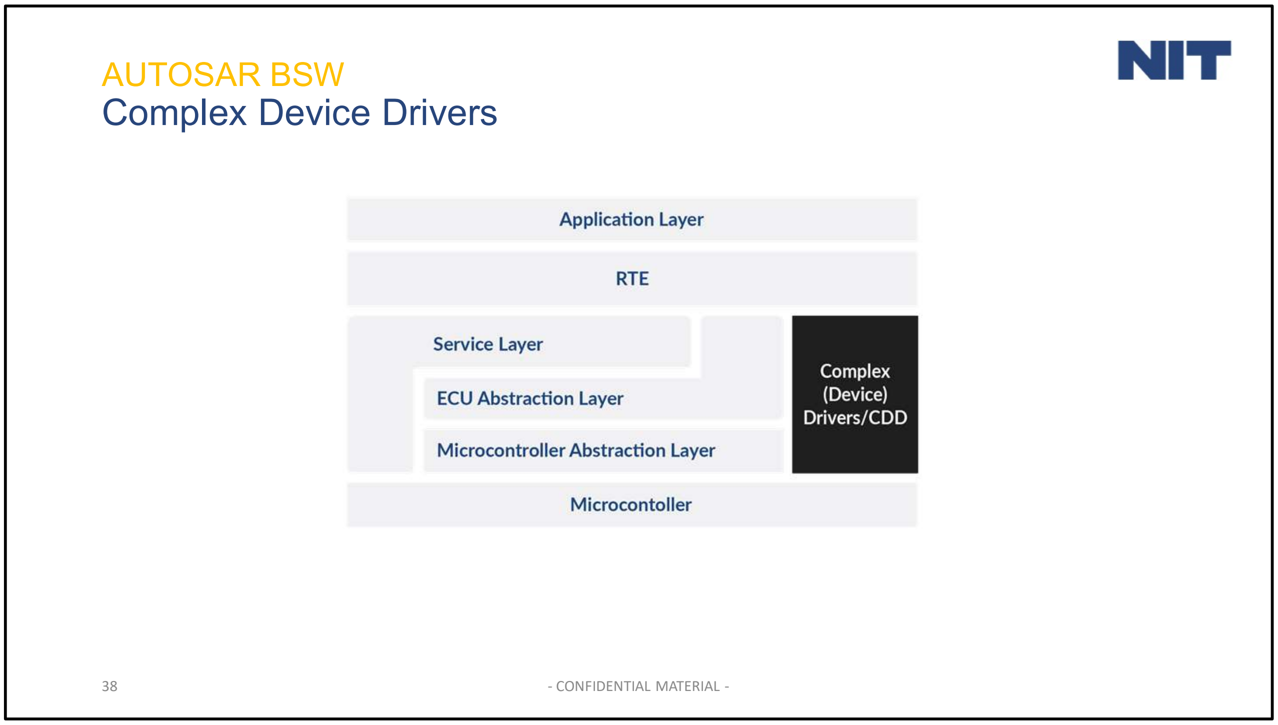


AUTOSAR BSW
Microcontroller Abstraction Layer



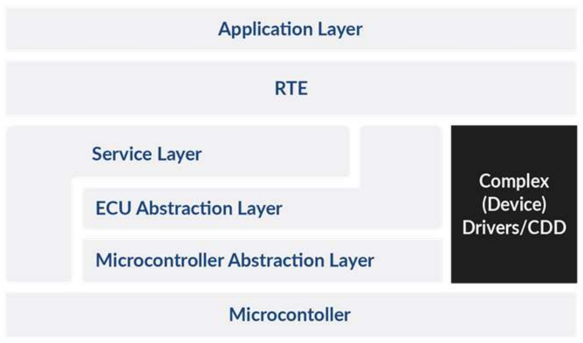
The Microcontroller Abstraction Layer (MCAL) is the lowermost software layer of the basic software. This layer contains drivers for direct access to the microcontrollers, internal peripherals and memory mapped microcontrollers of external devices.

The task of the Microcontroller Abstraction Layer is to make higher layers independent of the microcontroller.



AUTOSAR BSW

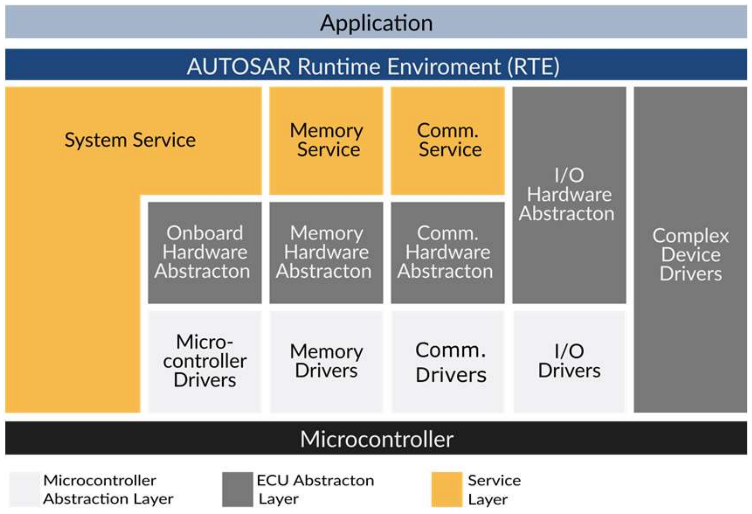
Complex Device Drivers



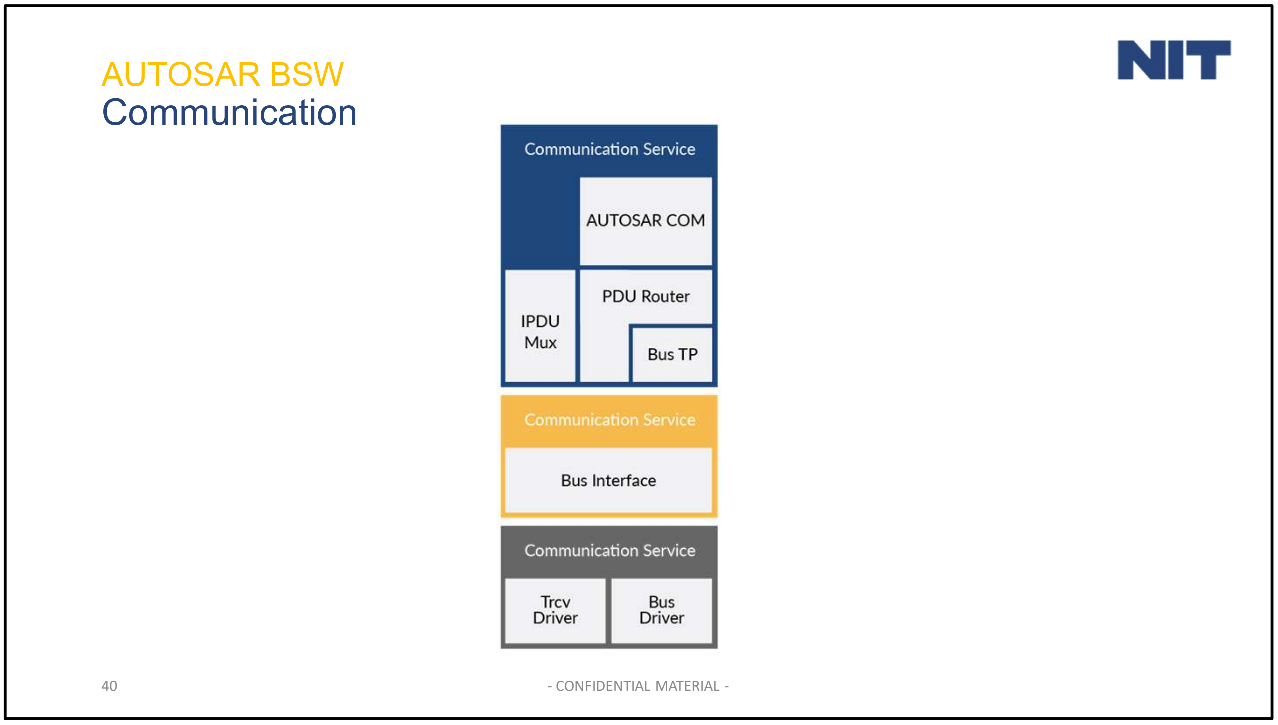
Complex Device Drivers represent a special case in the Layered Software Architecture of AUTOSAR. They control special sensors and actuators by direct microcontroller access, e.g., modules with critical timing constraints (i.e., injection control, battery management, etc.) or non-AUTOSAR (legacy) modules. One key concept to follow strict timing requirements is the possible usage of interrupts (not allowed above the RTE layer). Complex Drivers may have interfaces to modules in the BSW and application layer.

For example, CDD may implement a new communication system like Bluetooth or USB connections, PduR configuration can take this into account by calling `CDDIf_Transmit()`.

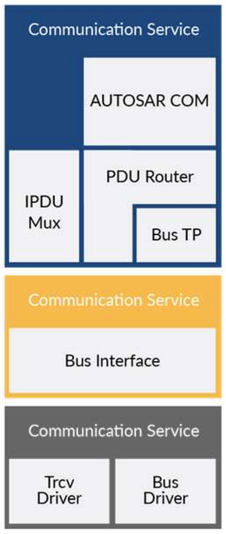
AUTOSAR BSW
Layered View: Detailed



The AUTOSAR basic software architecture is organized horizontally in different clusters (areas of function of the basic software) and vertically in different layers of abstraction.



AUTOSAR BSW
Communication



40 - CONFIDENTIAL MATERIAL -

Com module operates on signals but has no awareness of the underlying bus technology. Com module implements `Com_ReceiveSignal`, function that reads signal value and provides it to SWC, and `Com_SendSignal`, function that triggers sending of message over bus (signals are contained in messages).

PDU Router is responsible for routing of messages (for same or different type of buses, i.e., CAN -> CAN, CAN -> FlexRay) and between service layer modules (COM/DCM) and transport protocol modules.

PDU router abstracts bus system technology for higher layers.

Transport protocol is used for segmentation and desegmentation of messages. It is only used by the Diagnostic Communication Manager (DCM) to allow bigger messages then allowed by underlying bus technology.

Maximum size of message is limited by the bus technology:

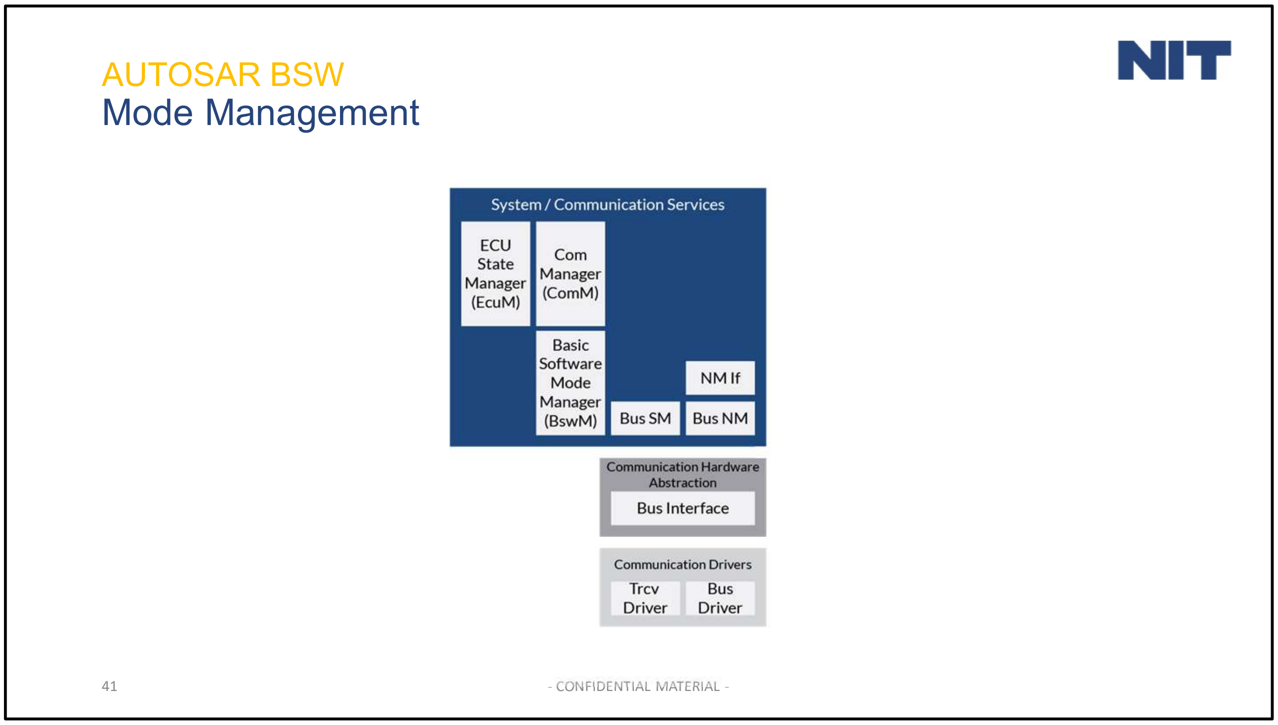
- CAN - 8B
- LIN – 8B
- CANFD – 64B
- FlexRay – 254B

Driver consists of a hardware-specific driver (i.e., CAN-DRV) and the hardware-independent interface (i.e., CAN-IF).

Hardware-independent driver is implementing additional queuing, microcontroller state control handling and generic APIs to hardware-specific driver..

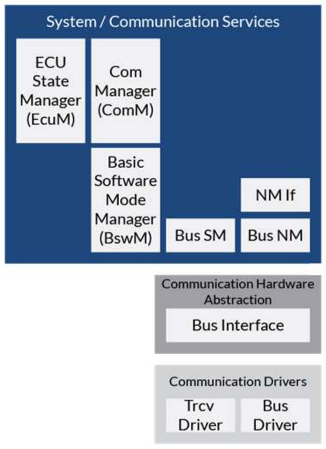
Hardware-specific driver is implementing microcontroller initialization, reading/writing of data from/to buffer, ISR routines.

Transceiver driver adapts the signal levels that are used on the CAN bus to the digital signals recognized by the microcontroller.
In addition, the transceivers can detect electrical malfunctions like wiring issues.



AUTOSAR BSW

Mode Management



ECU State Manager (EcuM)

- Manages fundamental ECU phases (OFF, UP, SLEEP, etc.)
- Wakeup validation and handling
- Initialization and de-initialization of BSW, OS and RTE

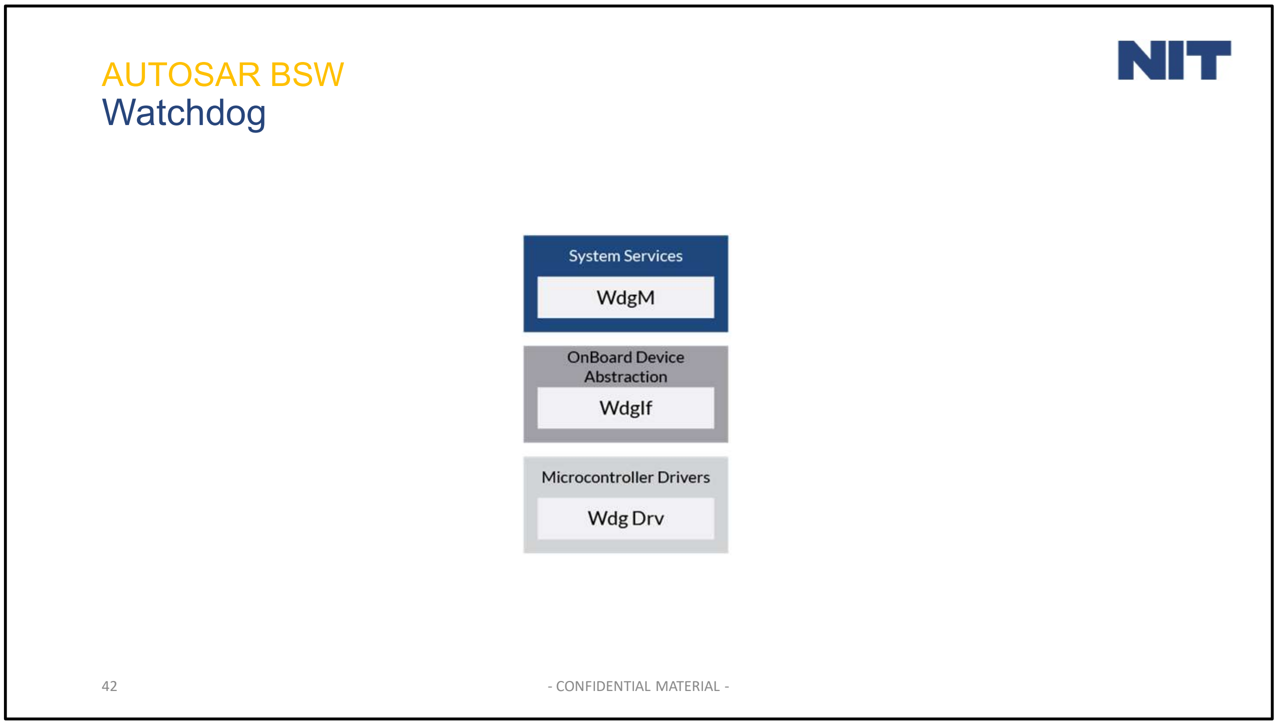
Communication Manager (ComM)

- Abstracted bus state machine
- Startup/shutdown of communication

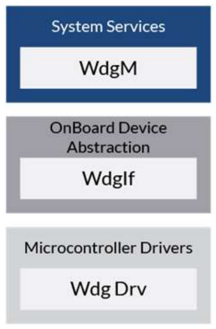
Network Management (Nm If, Bus NM) keeps the bus awake and coordinates bus shutdown.

Bus State Manager (Bus SM) handles the startup and shutdown of the network communication.
Bus SM maps the Bus SM states to the states of the ComM and causes the necessary actions to change the Bus SM state to those requested by the ComM.

Basic Software Mode Manager (BswM) is generic rule-based module that controls BSW behavior (e.g., switch ON/OFF specific bus channel of ECU).
BswM arbitrates mode requests from application layer SWCs or other BSW modules and performs actions based on the arbitration result.


AUTOSAR BSW

Watchdog



Watchdog supervises execution of application software.

WdgM

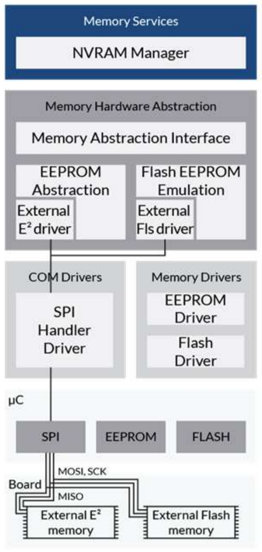
- Manages alive indications from all supervised entities (runnables of application)
- Derives Global Supervision Status
- Triggers WdgIf if runnables behave like expected

WdgIf triggers Hardware Watchdog(s)

WdgDrv implements internal or external Watchdog driver

AUTOSAR BSW

Memory Services

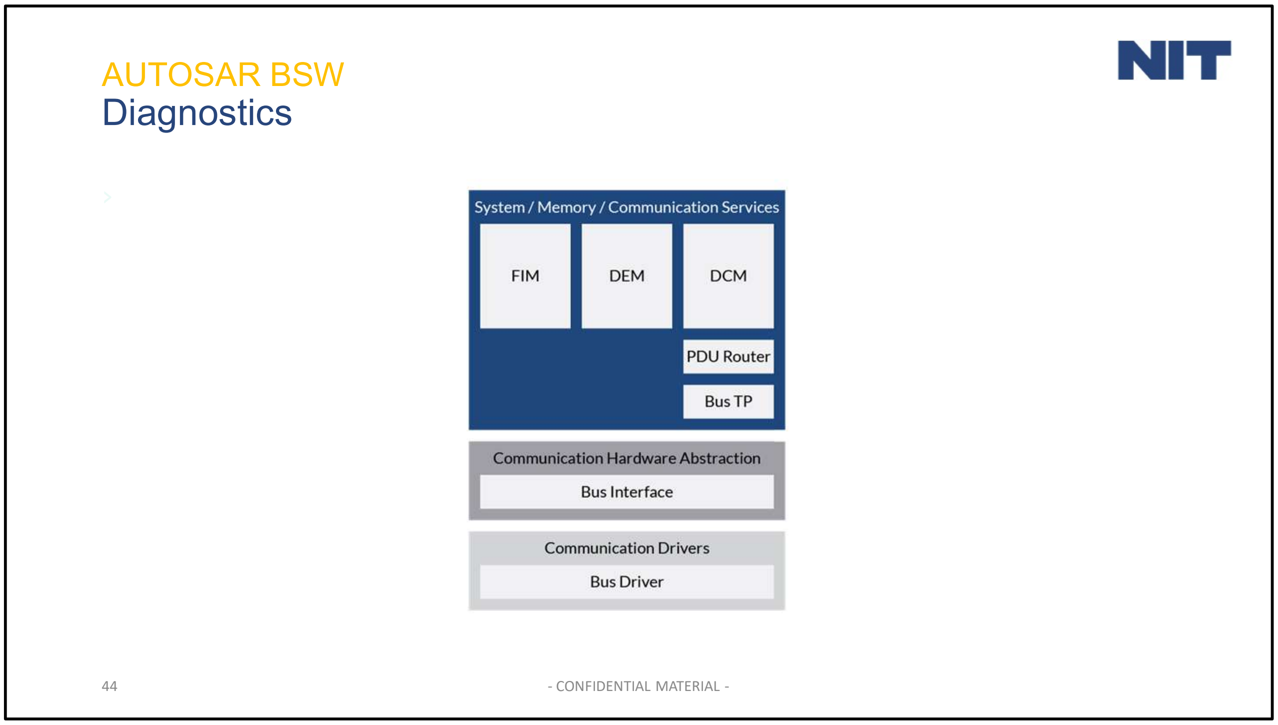


The NVRAM Manager converts data blocks with ID to memory addresses. NVRAM Manager in unaware of underlying technology (FLASH or EEPROM).

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.

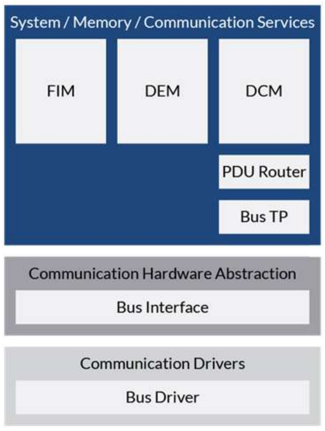
Since the EEPROM or Flash is slow compared to program execution, the memory is written piecewise and waits until the write process has ended.

Writing occurs in the MainFunction of the driver, which is called periodically. After the entire block has been written JobEndNotification is passed from the driver to the NVRAM Manager.



AUTOSAR BSW

Diagnostics



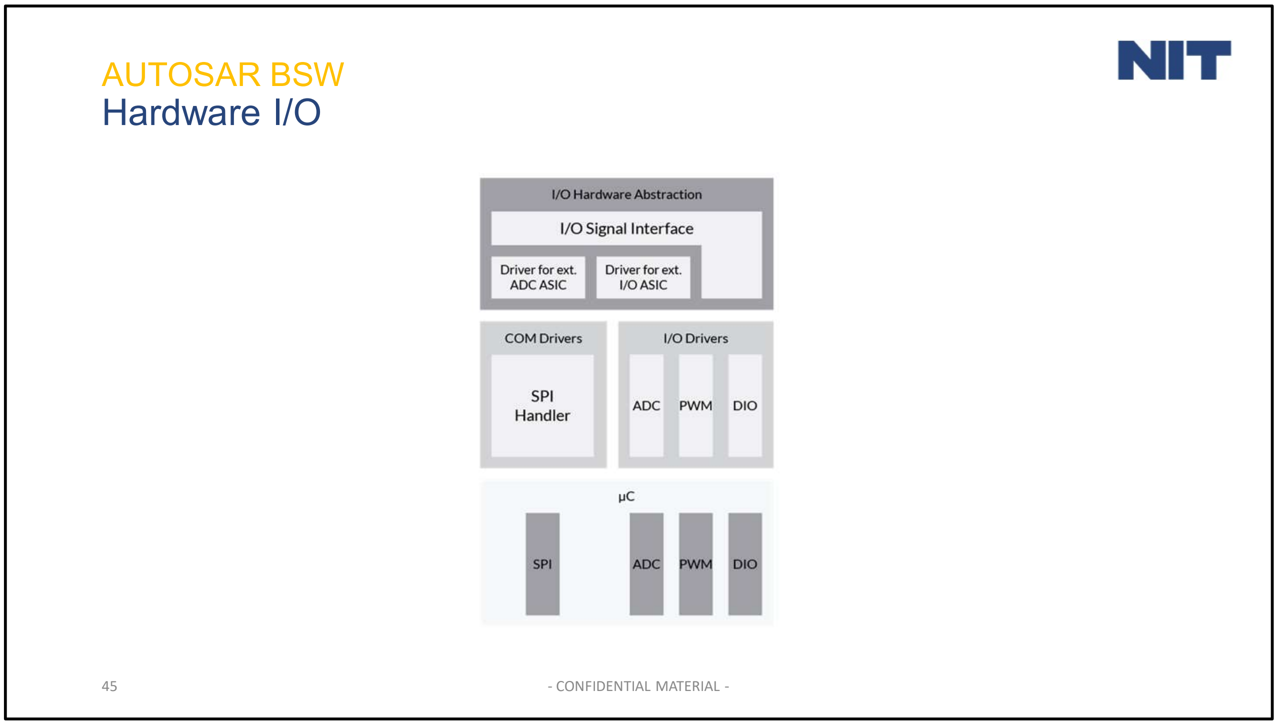
System / Memory / Communication Services
FIM
DEM
DCM
PDU Router
Bus TP
Communication Hardware Abstraction
Bus Interface
Communication Drivers
Bus Driver

Diagnostic Communication Manager (DCM) is bus independent module that ensures flow of diagnostic data (ECU <-> External Diagnostic tool) and manages diagnostic states (diagnostic sessions and security states).

Diagnostic Event Manager (DEM) saves diagnostic errors (events) reported from BSW and application.

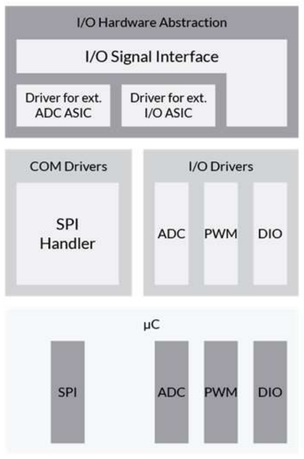
Diagnostics Event Manager (DEM) is responsible for storing and processing diagnostic events and all the data associated with it.
In addition to it, DEM provides the diagnostic trouble codes (DTC) to the Diagnostic Communication Manager (DCM) when required.

Function Inhibition Manager is responsible for providing a control mechanism for SWCs (in case of error, a SWC is inhibited, some functions are switched off due to errors).



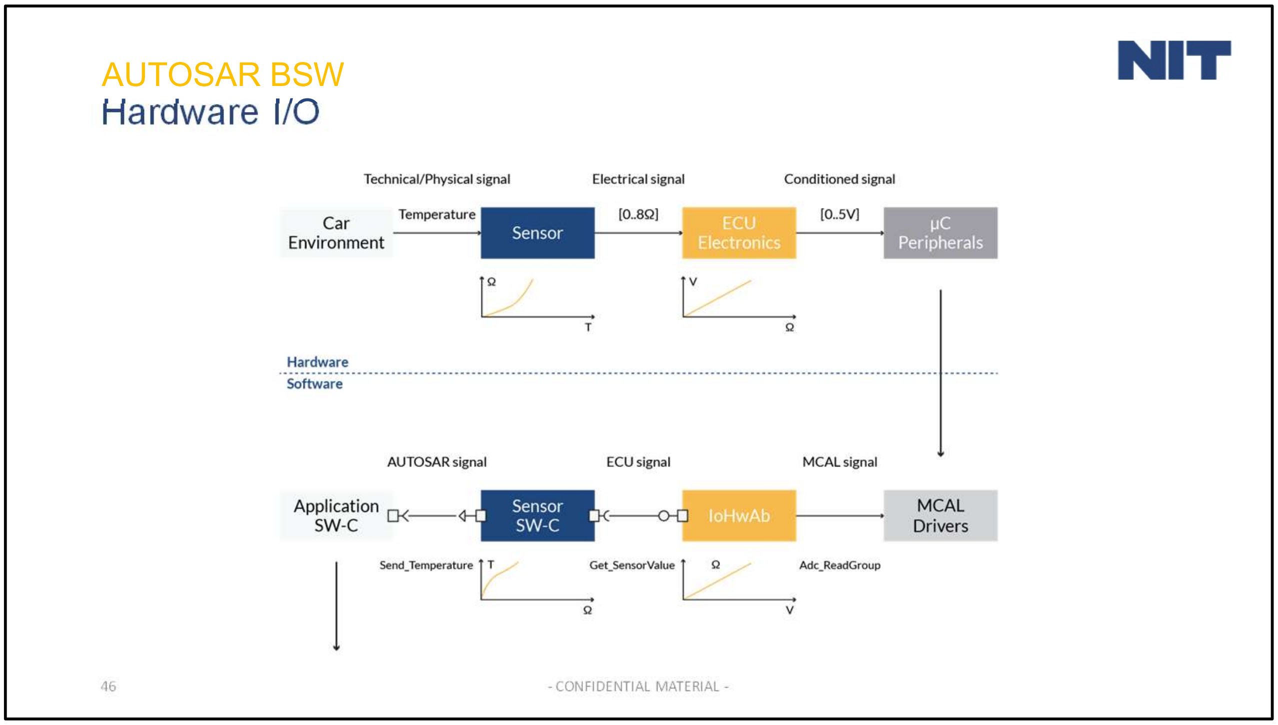
AUTOSAR BSW

Hardware I/O



Abstraction of location of I/O peripheral devices and layout.

Inside Hardware I/O MCAL is implemented.
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.



AUTOSAR BSW
Hardware I/O

NIT

46 - CONFIDENTIAL MATERIAL -

Sensor converts a physical or environmental signal into an electrical signal (e.g., temperature to resistance).

ECU Electronics converts sensor's electrical signals into an input signal suitable for the microcontroller (e.g., resistance to voltage).

uC Peripherals: Analog to Digital Converter (ADC), Digital Input Output (Dio), SPI…

MCAL Drivers implement peripheral drivers according to AUTOSAR (Dio, ADC, PWM, etc.).

I/O Hardware Abstraction converts from hardware-specific raw values into physical values (suitable for sensor SWC).

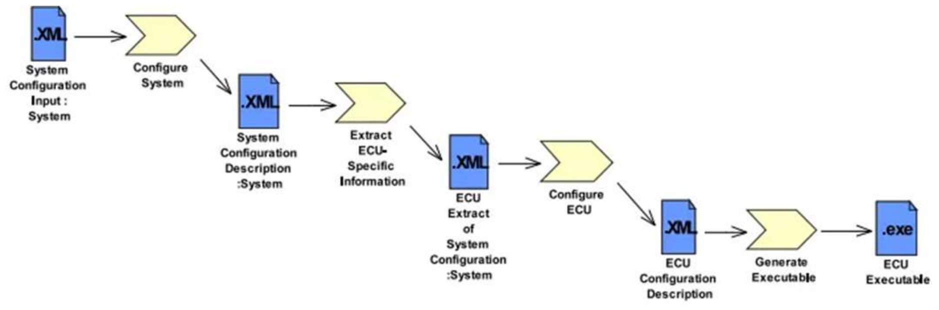
Sensor SWC is SW representation of hardware sensor. It converts electrical input values into application signals. Sensor SWC knows sensor hardware characteristics.

Application SWC implements control algorithm.

Agenda

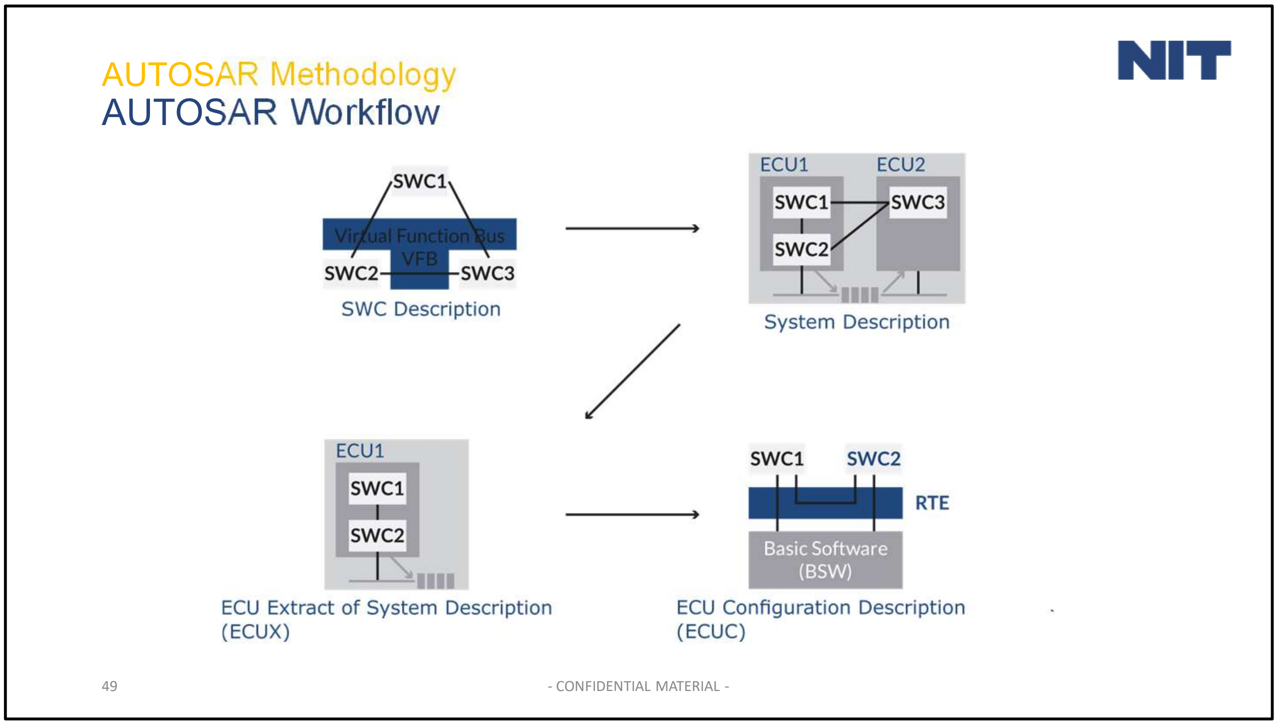
- Overview and Objectives
- AUTOSAR Application
- AUTOSAR RTE
- AUTOSAR BSW
- **AUTOSAR Methodology**
- AUTOSAR in Practice

AUTOSAR Methodology
Overview

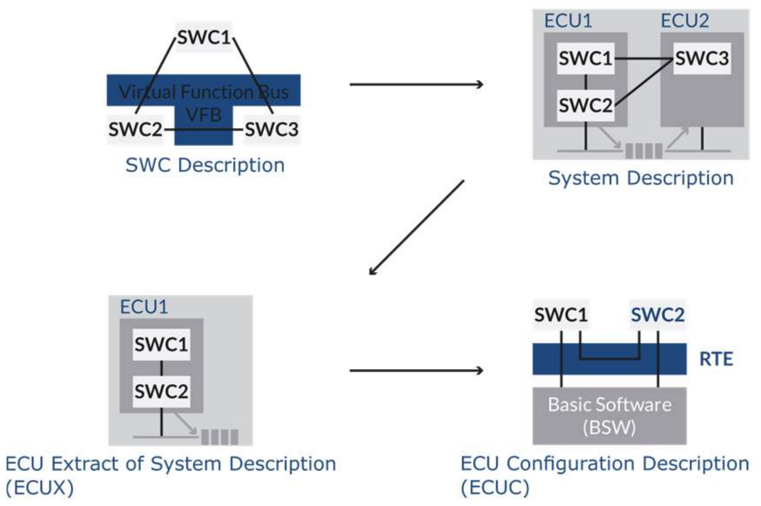

.XML
System Configuration Input : System
Configure System
.XML
System Configuration Description :System
Extract ECU-Specific Information
.XML
ECU Extract of System Configuration :System
Configure ECU
.XML
ECU Configuration Description
Generate Executable
.exe
ECU Executable

AUTOSAR Methodology describes all major steps of development of a system with AUTOSAR: from the system-level configuration to the generation of an ECU executable.
AUTOSAR Methodology is not a complete process description, but it defines dependencies of different activities on work-products.

AUTOSAR describes the methodology using the Software Process Engineering meta-model (SPEM). SPEM is designed to describe software development processes.



AUTOSAR Methodology
AUTOSAR Workflow



SWC Description contains the following information:

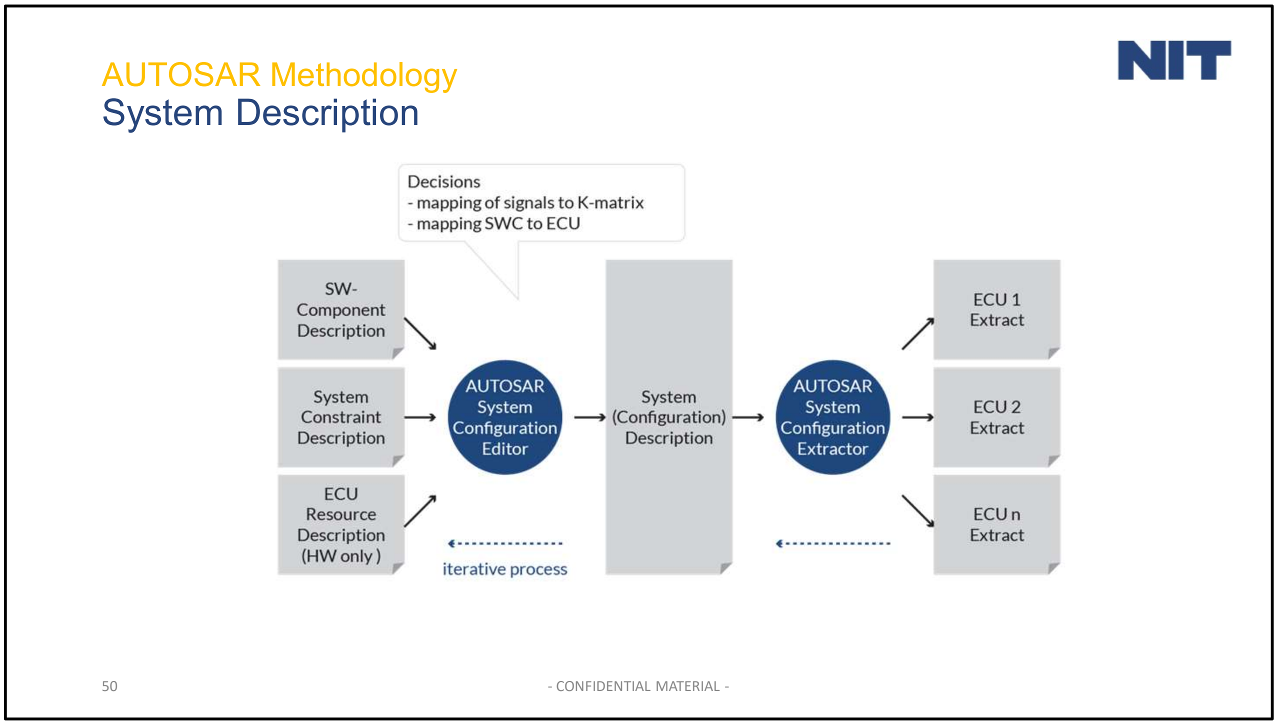
- Data and operations which are part of the component
- Resource needs of the component (memory, CPU time, etc.)
- Information relating to the specific implementation (repetition rate)
- Component interfaces

System Description contains the following information:

- Information on network
- Topologies
- Limitations ("Constraints")
- Protocols
- K-matrix
- Baud rate and timing parameters

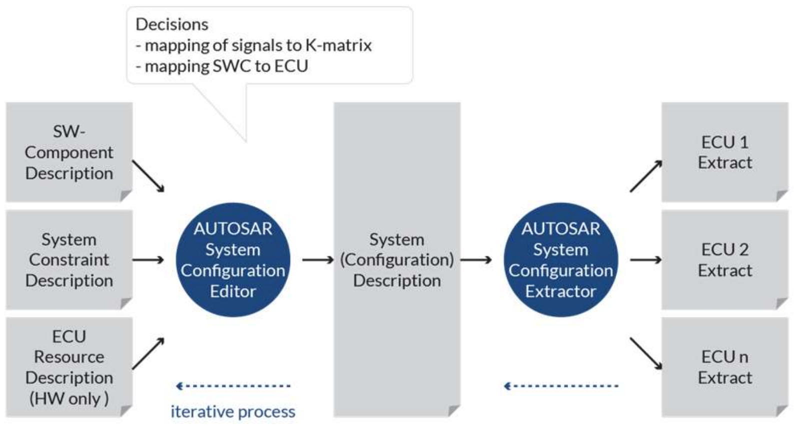
ECU Extract of System Description contains the following information:

- Description of the hardware being used
- Sensor, actuator
- Memory
- Processor
- Communications periphery
- Pin assignments



AUTOSAR Methodology

System Description


Decisions
- mapping of signals to K-matrix
- mapping SWC to ECU
SW-Component Description
System Constraint Description
ECU Resource Description (HW only)
AUTOSAR System Configuration Editor
System (Configuration) Description
AUTOSAR System Configuration Extractor
ECU 1 Extract
ECU 2 Extract
ECU n Extract
iterative process

The Software Component Description contains detailed information on the ports, interfaces and connections between software components.

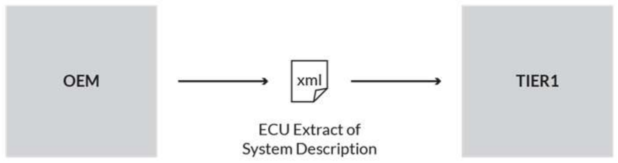
AUTOSAR Methodology

AUTOSAR Workflow with OEM and TIER1

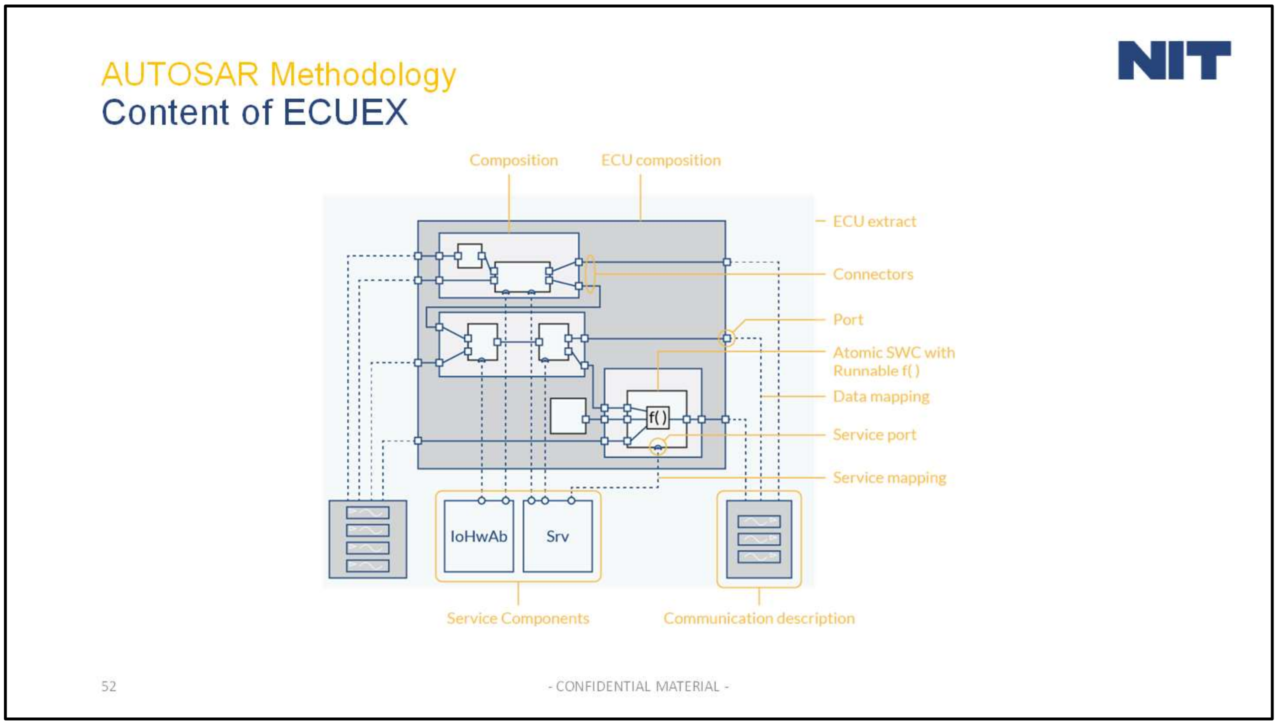
Interface between OEM* and TIER1*: ECU Extract of System Description (ECUEX)

OEM creates ECUEX based on vehicle system design

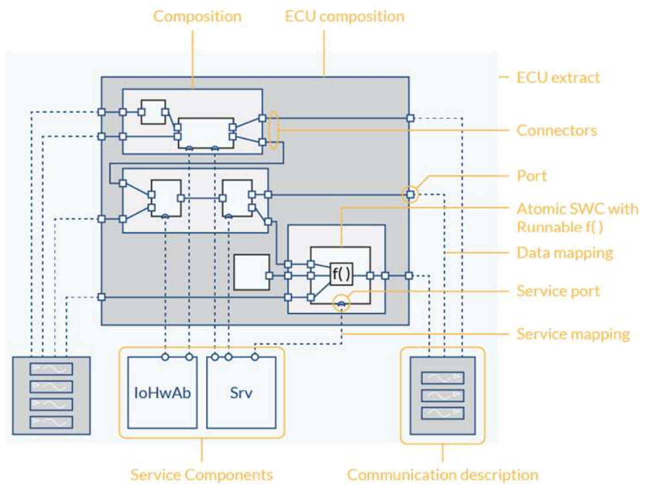
TIER1 configures AUTOSAR ECU based on ECUEX



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



AUTOSAR Methodology
Content of ECUEX

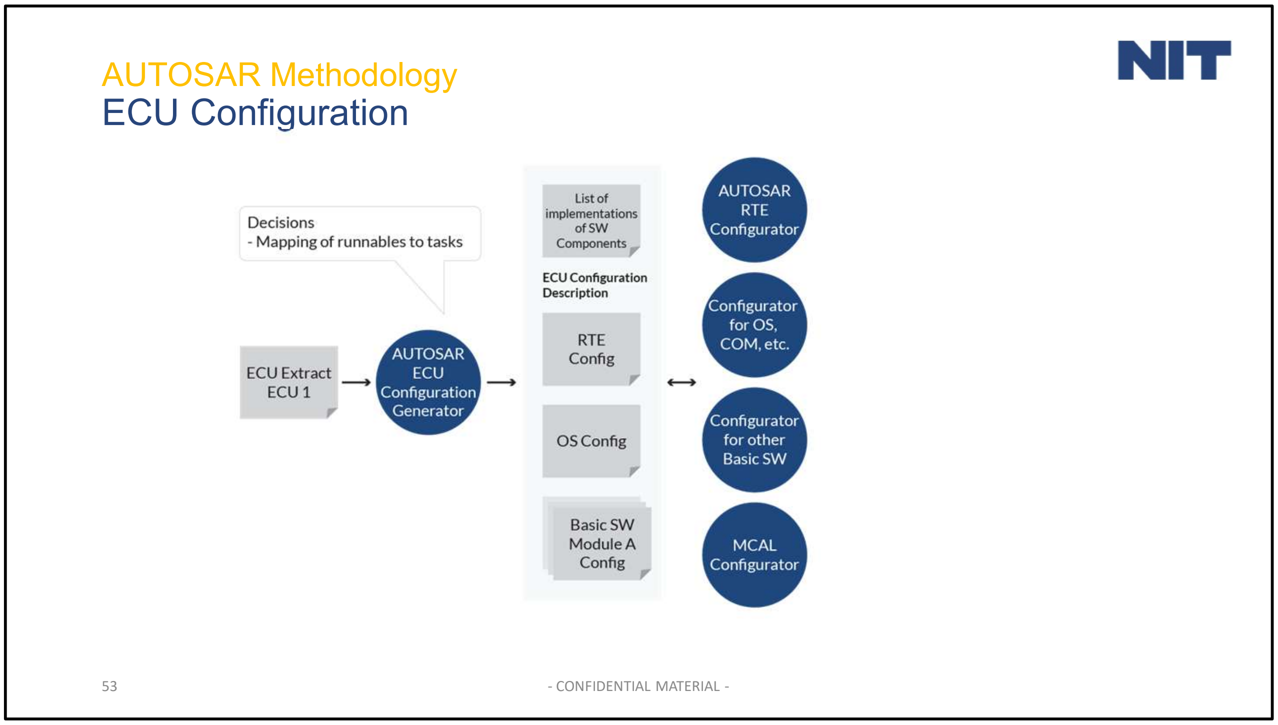


Communication description of the ECU:

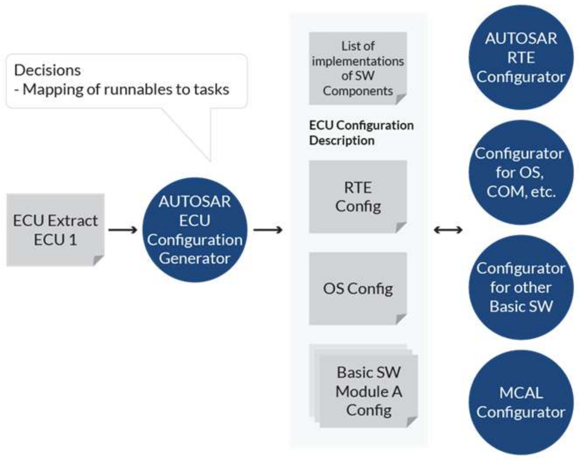
- Tx-/Rx-Frames, IDs, network signals
- Send types
- PDUs

SWCs, Ports (grouped within a central ECU composition)

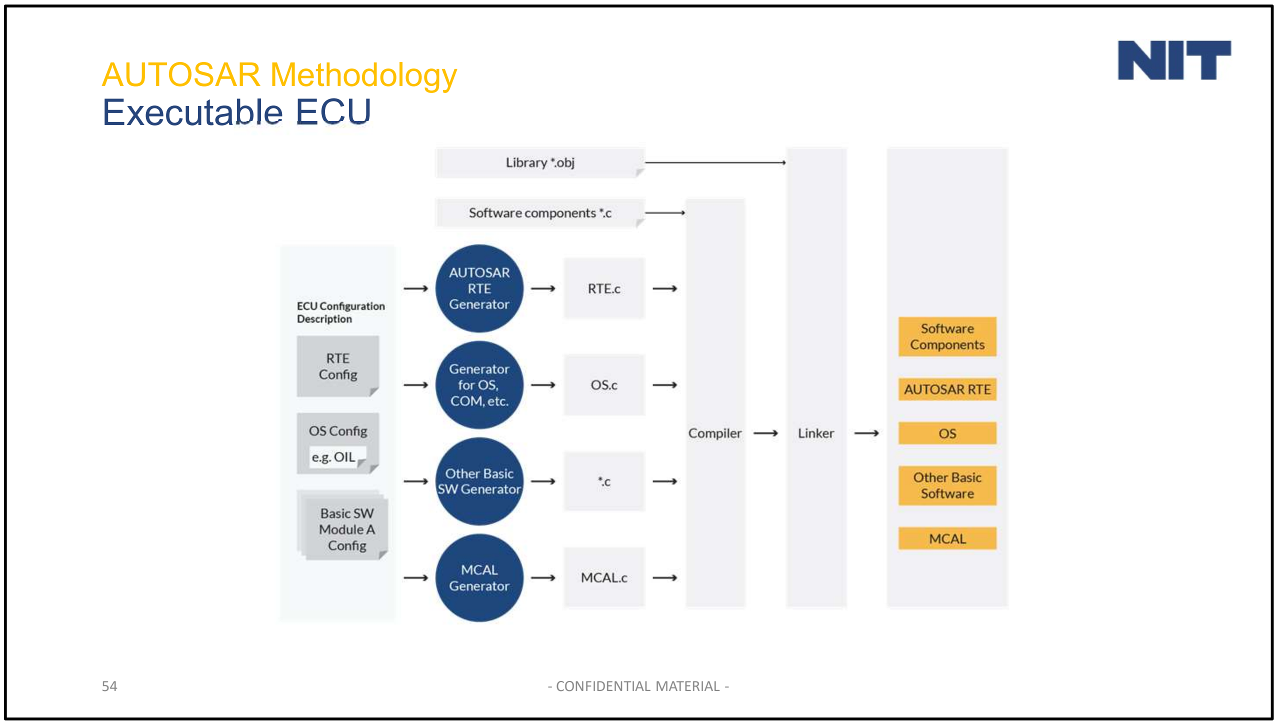
- Atomic SWCs, Runnables
- Compositions
- Port Interfaces, Data Types, Constants
- Connectors between the SWCs
- Data Mapping (connecting SWCs with network signals)
- Service Components (providing interfaces of the used BSW services for the SWCs - need to match with the ECU Configuration)
- Service Mapping (connecting SWCs with Service Components)



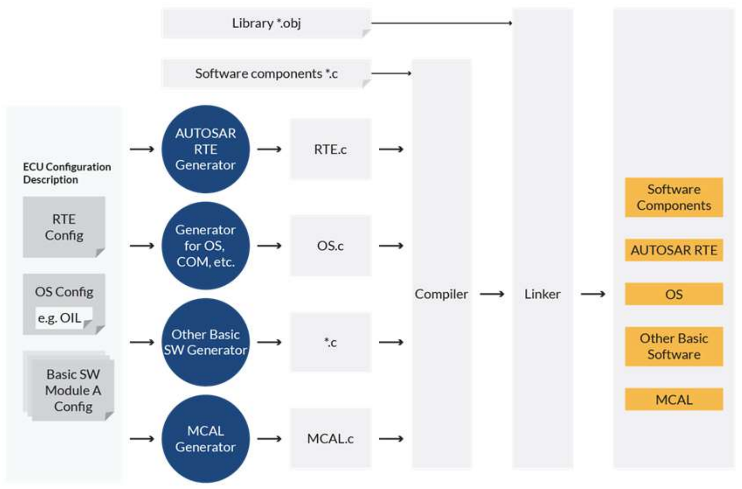
AUTOSAR Methodology
ECU Configuration


Decisions
- Mapping of runnables to tasks
ECU Extract ECU 1
AUTOSAR ECU Configuration Generator
List of implementations of SW Components
ECU Configuration Description
RTE Config
OS Config
Basic SW Module A Config
AUTOSAR RTE Configurator
Configurator for OS, COM, etc.
Configurator for other Basic SW
MCAL Configurator

The information of the ECU Description is used to configure the basic software of the ECU.



AUTOSAR Methodology
Executable ECU

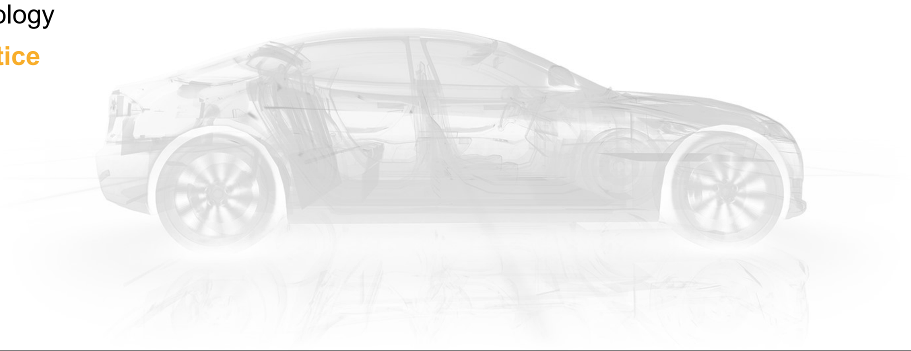


- CONFIDENTIAL MATERIAL -

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

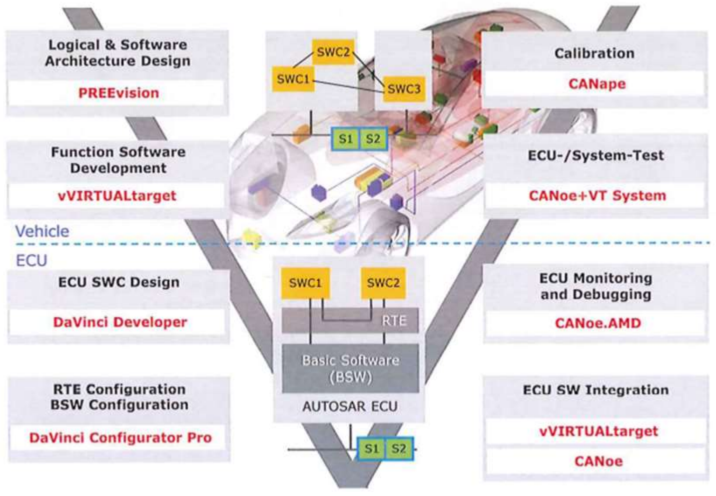
Agenda

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



AUTOSAR in practice

Vector Tools



PREEvision is a model-based tool that supports the entire vehicle system development – from architecture design all the way to the final wiring harness.

DaVinci Developer is tool for ECU SWC design and validation in graphical editor (ports, runnables, triggers, etc.).

DaVinci Configurator Pro is the central tool for configuration, validation and generation of the basic software (BSW) and the runtime environment (RTE) of an AUTOSAR ECU.

Integration of ECU SW in a virtual environment
- Single-source ECU configuration for real ECU and virtual target
- Execution and debugging of ECU code on the PC with vVIRTUALtarget

CANoe is tool for development, test and analysis of individual ECUs and entire ECU networks.

CANape is used for calibration of ECU parameters and recording of ECU signals.