ETAS

DRIVING EMBEDDED EXCELLENCE

RTA Solutions

Tailor-made Embedded Software Solutions

Basic Integration Package (BIP)

Integration Guide

Status: Released

Document Version: 02
Document Date: 2022/06/10

Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2022 ETAS GmbH, Stuttgart

The names and designations used in this document are trademarks or brands belonging to the respective owners.

Document : ETAS_BIP_Integration_Guide_RTA-CAR9-V2.docx

Contents

READ

ETAS

1 Introduction

The **B**asic **I**ntegration **P**ackage (**BIP**) provides a baseline integration of RTA-CAR (ETAS Classic AUTOSAR Solution) using the target microcontroller/microprocessor mentioned in section ***Target Hardware*** of <ETAS BIP Release Notes.pdf>.

It aiming to boost **user iteratively C**ontinuous **D**evelopment **C**ontinuous **I**ntegration by a configuration integration baseline and a set of Upper Tester components as test cases.

1.1 About this Document

This Integration Guide describes

- How to perform system description with provided reference templates.
- How to deploy system description to **Ecu C**onfiguration (**EcuC**).
- How to perform EcuC static configuration (arxml integration) with a set of baseline configuration.
- How to perform software integration between modules among BSW, MCAL and integration code.
- How to build and run the project on the target board.

It also describes what shall be taken care in terms of **arxml integration** and **software integration** when porting to a new target or compiler.

Information contained in below section are of special important that user shall take into consideration during integration.

- **Hint**
- **Integration Hints**

> **NOTE**
> Information and picture in this guideline may slightly differs from that of BIP in terms of font color, picture contents, software revisions, etc. For any differences user discovered, you are suggested to [contact ETAS] counterparts for an update.

1.2 Who should read this document

ETAS RTA-CAR users who use the **BIP** as baseline to iterate their own projects with project specific requirements should read this document before use.

> **NOTE**
> **Pre-conditions:** Fresh users of RTA-CAR are recommended to join RTA-CAR training and received BIP first to be able to understand better the configuration and description in the following sections.

1.3 How will you receive more information?

This document is not intended to describe configuration of each module. For that, please also contact the delivery window person of sharing partners to get more information.

1.4 Definitions and Abbreviations

Acronym	Definition

BIP	**B**asic **I**ntegration **P**ackage baseline of RTA-CAR covers system deployment, EcuC static configuration and software integration with MCAL
BSWMD	AUTOSAR **BSW M**odule **D**escription **T**emplate
DEXT	AUTOSAR **D**iagnostic **EXT**ract
OIP	**O**EM **I**ntegration **P**ackage of RTA-CAR covers system deployment, EcuC static configuration and software integration with MCAL and Upper Tester for OEM requirements
SWCD	AUTOSAR **S**oft**W**are **C**omponent **D**escription

Table 1: Definitions and Abbreviations

ETAS

2 Getting Started

Software development with AUTOSAR consist of four phases:

- **Phase 1: System Description and Deployment**: I-Signal communication, Network management, Diagnostic Description etc.
  - ❖ Require deep understanding of AUTOSAR toolchain and be able to rapidly iterate frequently changed description files (e.g., DBC, DEXT)
- **Phase 2: EcuC configuration** (arxml integration):
  - ❖ Require deep understanding of AUTOSAR toolchain and splitable BSW modular configurations.
- **Phase 3: Application Integration** (arxml configuration): AUTOSAR interface, standardized AUTOSAR interface
  - ❖ Require deep understanding of AUTOSAR toolchain and vendor specific parameters.
- **Phase 4: Software Integration**: Linker file, partition, startup, etc. integration

To support **C**ontinuous **D**evelopment and **C**ontinuous **I**ntegration efficiently, ETAS BIP provided **Cobra** assistant to help user development from step (1) network description to steep (7) software integration.

From section 3 to section 7, user will find out how to use Cobra assistant with BIP configuration and integration.

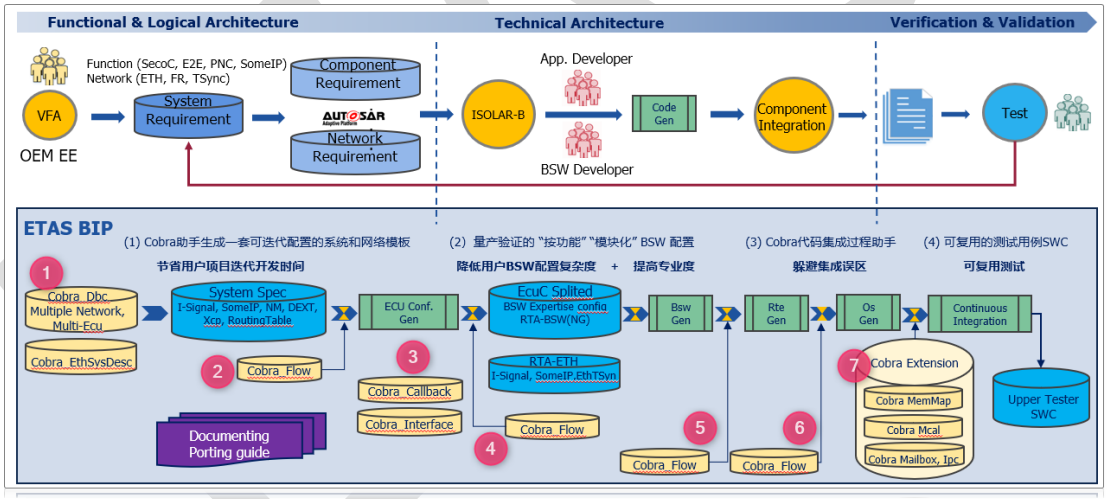


Figure 1 AUTOSAR Software development workflow

2.1 How to Start with BIP

The sections below describe the workflow of how BIP integrated from

- Phase 1 [3_System Description and Deployment]
- Phase 2 [4_ECU Configuration]
- Phase 3 [5_Application Integration]
- Phase 4 [6_Software Integration]

To start with debugging and running of BIP, user could start from section [2.2 How to Build]

To start with **iterating** configuration and integration from BIP to user projects, user could start from section [2.5 How to load BIP into ISOLAR-AB] section by section.

NOTE

User is strongly recommended to get familiar with the BIP from [2 Getting Started] to [6 Software Integration] before immediately modification to fit to project needs.

2.2 How to Build

BIP use python and scons to build executable. Users need to install these required tools to be able to build as detailed in section ***Build Tools*** of <ETAS BIP Release Notes.pdf>

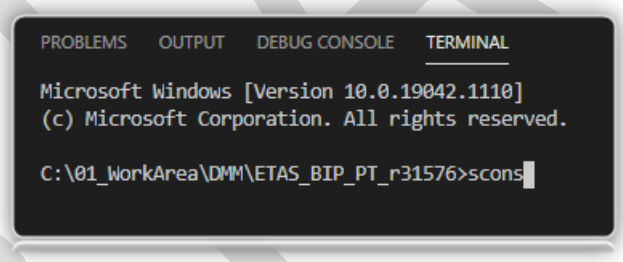
Hint:

how to install scons, user could find useful information in scons installation *README.txt*

To be able to find the required compiler by scons, user shall set his/her compiler path in variable `COMPILER_ROOT` of `[ProjectRoot]\settings.py`

```
19  # General configuration:---------------------------------------------------------------
20  OUTPUT_DIR          = 'output'
21  LOG_DIR             = 'log'
22  COMPILER_TOOL       = 'Arm'#Must match the file name located in the site_tools directory
23  COMPILER_ROOT       = 'C:/toolbase/arm/6.6/bin'
```

To build the BIP, execute the scons script by open console at the project root directory and execute 'scons'. This will invoke scons script tool to build the software.



After successfully build, the executable is generated into `[ProjectRoot]\output\dbg\` directory.

Hint: More details for build own projects with BIP can be found in section [6 Software Integration].

2.3 How to debug with BIP

The BIP has been provided with Trace32 debugging script in '**`TOOLS/T32/`**' directory.

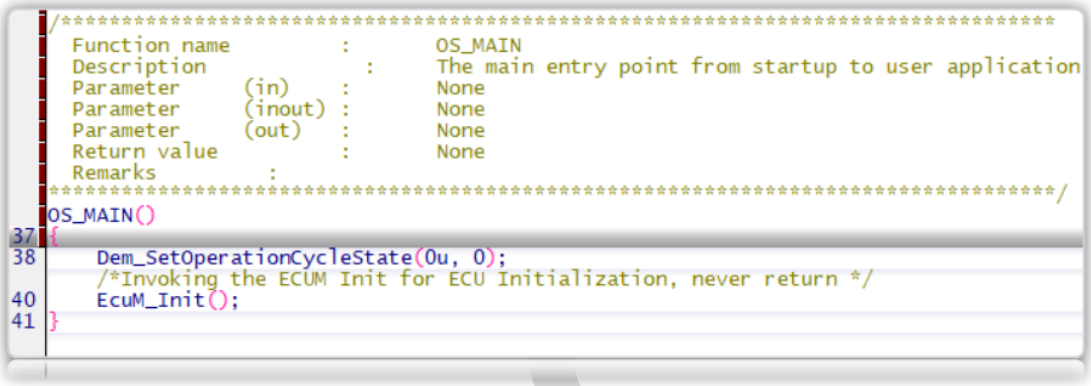
To start debugging, user shall set your Trace32 path with variable `T32` in '**`TOOLS/T32/run.bat`**'

```
2  set T32=C:\compiler\T32_202005
3
4  if not @%T32% == @ goto got_t32
5  @echo You need to set the env variable T32
6  goto exit
```

Double click `run.bat` will automatically load TRACE32 debugging window for BIP debugging.

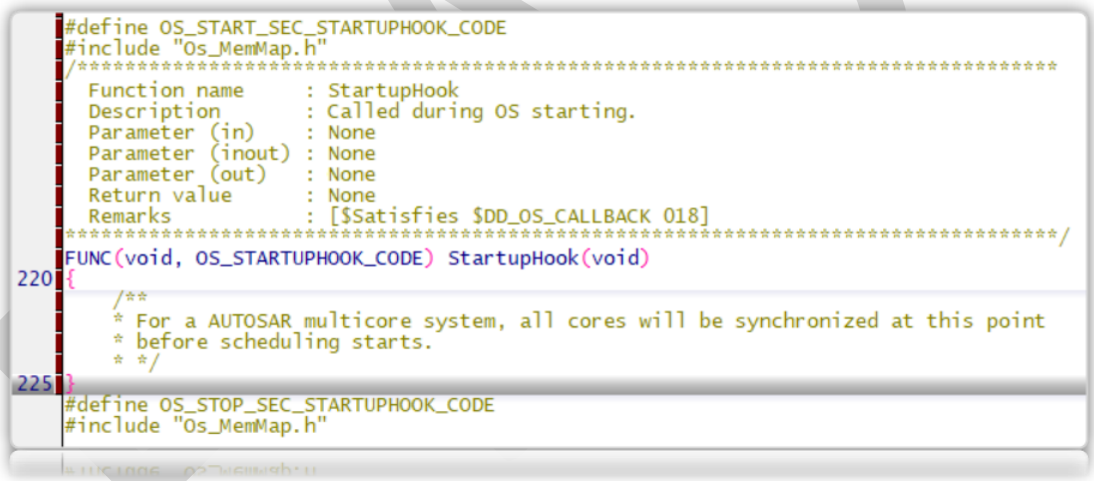
2.3.1 Run BIP Software

In debugger, you could set breakpoint at `OS_MAIN()` and check whether software could stop at `OS_MAIN()` which means hardware startup code successfully executed on master core as shown below.



```
/*******************************************************************************
  Function name          :       OS_MAIN
  Description              :     The main entry point from startup to user application
  Parameter      (in)    :       None
  Parameter      (inout) :       None
  Parameter      (out)   :       None
  Return value           :       None
  Remarks          :
*******************************************************************************/
OS_MAIN()
{
    Dem_SetOperationCycleState(0u, 0);
    /*Invoking the ECUM Init for ECU Initialization, never return */
    EcuM_Init();
}
```

Then you could set breakpoint at `StartupHook` and check software could stop at `StartupHook` which means software startup and EcuM initialization finished.

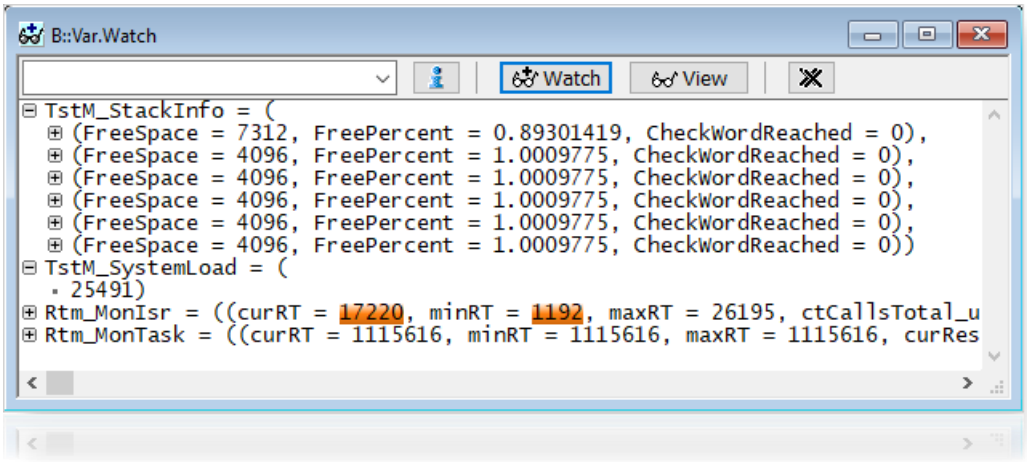


```
#define OS_START_SEC_STARTUPHOOK_CODE
#include "Os_MemMap.h"
/*******************************************************************************
  Function name      : StartupHook
  Description        : Called during OS starting.
  Parameter (in)     : None
  Parameter (inout)  : None
  Parameter (out)    : None
  Return value       : None
  Remarks            : [$Satisfies $DD_OS_CALLBACK 018]
*******************************************************************************/
FUNC(void, OS_STARTUPHOOK_CODE) StartupHook(void)
{
    /**
    * For a AUTOSAR multicore system, all cores will be synchronized at this point
    * before scheduling starts.
    * */
}
#define OS_STOP_SEC_STARTUPHOOK_CODE
#include "Os_MemMap.h"
```

After return from `StartupHook()`, OS kernel running and it will start scheduling Tasks.

2.3.2 Observe Software Performance

BIP is developed with Cobra Extension Rtm and StackM. User could use Rtm and StackM to monitor software performance by loading `BIP_PerformanceMonitor.cmm` from path `[ProjectRoot]\TOOLS\T32` into TRACE32 debugger.



Interpreting of Performance Monitor Information:

For details of software performance, please refer to <ETAS BIP Release Notes.pdf>.

Variable	**Information**
`TstM_StackInfo`	`FreePercent` Represent the stack utilization, e.g. 0.89 means 89% of the stack is free.
`TstM_SystemLoad`	Represent system load calculated during runtime.
`Rtm_MonIsr`	Contains information for each OS CAT2 ISR of current Runtime value, minimum Runtime value, maximum Runtime value, cumulated runtime value, average runtime value, total number of calling count, etc.
`Rtm_MonTask`	Contains information for each OS Task of current Runtime value, minimum Runtime value, maximum Runtime value, cumulated runtime value, average runtime value, total number of calling count, etc.

2.4 BIP Startup Sequence

BIP startup with below sequences from step 1 to step 5:

1. Hardware specific startup code for master core
   a) Slave core startup
2. User level startup `OS_MAIN()`
3. EcuM Startup `EcuM_Init()`
   1) EcuM_AL_DriverInitZero
   2) EcuM_AL_DriverInitOne
   3) EcuM_Prv_SetDefinedMcuWakeupSource
   4) EcuM_Prv_StartSlaveCores
4. RTA-OS Startup `StartOS()`

a) StartupHook

5. RTA-OS Scheduling
   a) `BswM_Init()`
   b) Application running

[$BIP_INTG 133] For different targets (e.g., single core, multicore core), BIP startup sequence could be differs as described in sections below.

Such as in S32G, Can_43_LLCE_Init should be invoked after OS run normal, because it need to get os time. And Mcu_InitializeClock should be invoked twice before Port_Init and after Port_Init.

2.5 How to load BIP into ISOLAR-AB

To apply modification or navigation of BIP AUTOSAR configurations, user could load the project into ISOLAR-AB.

1. Open ISOLAR-AB
2. Import the project by **ISOLAR-AB** | **File** | **Import** | **Existing Projects into Workspace,** then select the BSW folder to import.

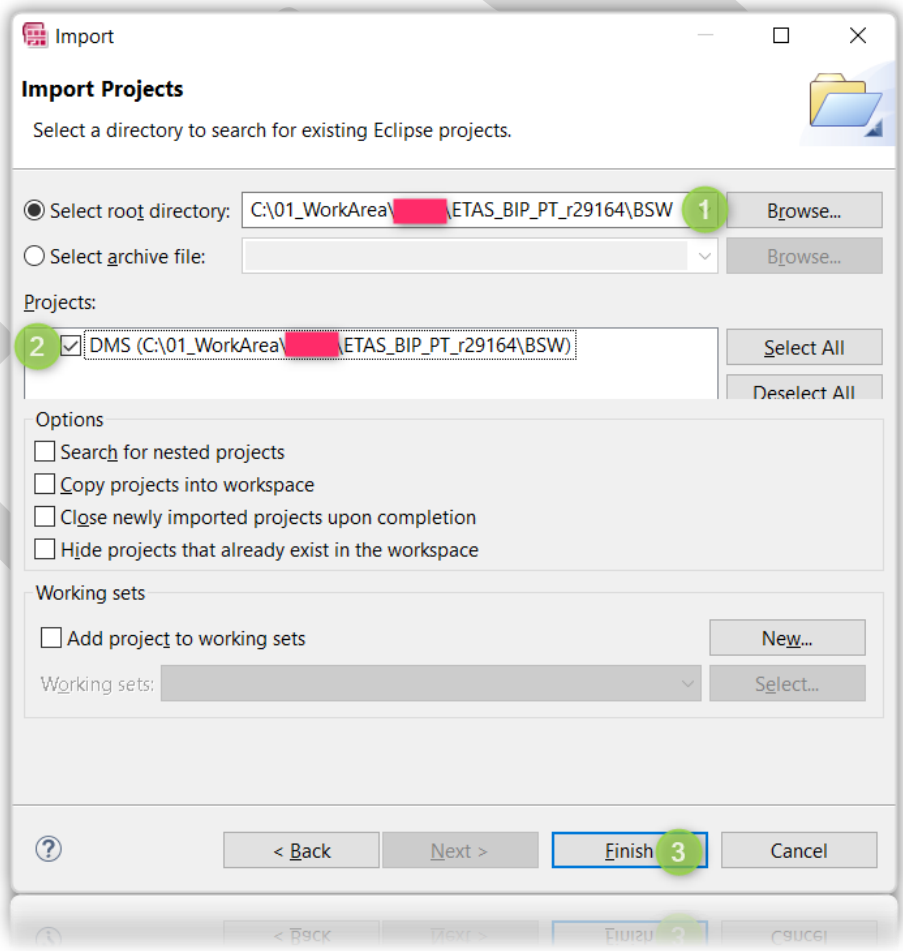


Figure 2 Loading BIP to ISOLAR-AB

The RTA-CAR configuration of BIP is now be imported into ISOLAR-AB.

NOTE

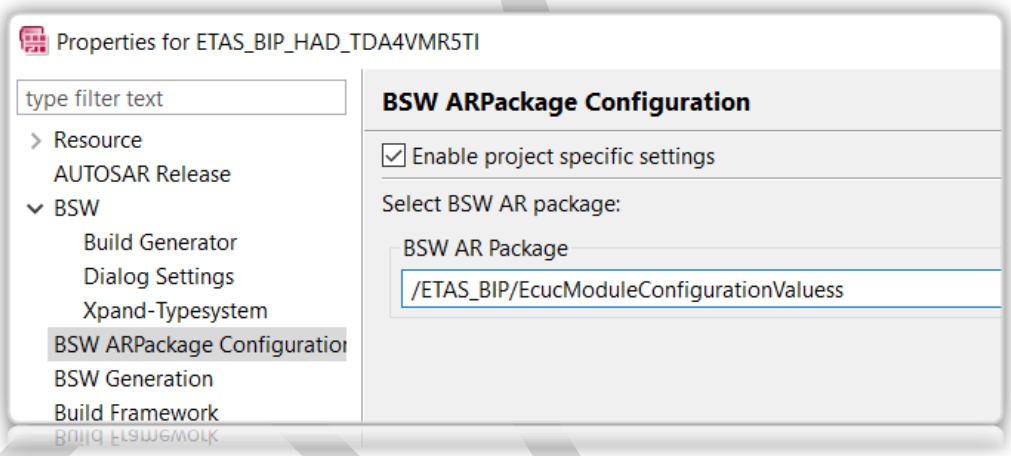
User could find out details of folder structure by refer to section ***Folder Structure*** of <ETAS BIP Release Notes.pdf> comes with BIP.

2.6 How to specify AR-Package

Building AUTOSAR based software is a huge and systematic work that usually requires teamwork.

AUTOSAR Package (**AR-Package**) is the main configuration component of AUTOSAR arxml file which contains the AUTOSAR elements that are referencing to each other.

[$BIP_INTG 001] To make the integration of AUTOSAR elements configured by different developers easily without resolving reference errors, BIP specifies AR-package path for BSWs with `/ETAS_BIP/EcucModuleConfigurationValuess/[NameOfBswModule]`.



NOTE

When add new modules, users are recommended for each module to start with AR-Package path `/ETAS_BIP/[NameofSWC]` or an AR-Package path been wildly agreed within team.

3 System Description and Deployment

This section describes:

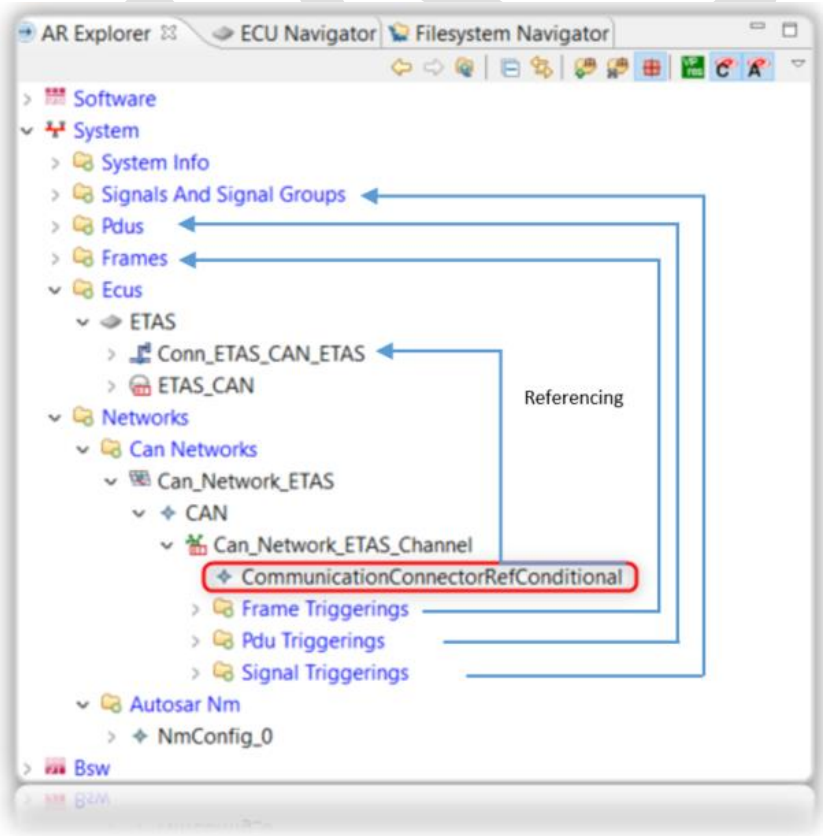
- [Section 3.1]: What system description (I-Signal communication, Network management, Diagnostic Description etc.) has been provided with BIP.
- [Section 3.2]: How user could update with their own network communication.
- [Section 3.3]: How to deploy system description (e.g. diagnostic, network) to Ecu level module configuration.
- [Section 3.5]: What shall be noticed when migrating to new version of RTA-CAR.
- [Section 3.6]: What shall be noticed when migrating to new target.
- [Section 3.7]: What shall be noticed when migrating to new compiler.

3.1 BIP System Description

BIP provides network communication and AUTOSAR diagnostic descriptions that can be automatically deploy into Ecu configurations.

3.1.1 Network Description

Different from **OIP**, **BIP** is shipped with one ECU node, with one or more controllers configured and connected to one or more network (depending on type of BIP).



In this BIP, one ECU node "**`ETAS`**" is described with one CAN network "**`Can_Network_ETAS`**".

[$BIP_INTG 002] Network description is imported by [3.2 How to update CAN Network Description] using DBC templates in folder `[ProjectRoot]\BasicSoftware\DBC` with four types of Pdus

a. ISignal IPdu: interaction layer PDUs for signal base communication over CAN

b. Dcm Ipdu: Diagnostic PDU for diagnostic communciation over CAN

c. General Purpose Ipdu: XCP PDUs for measurement and calibraiton over CAN

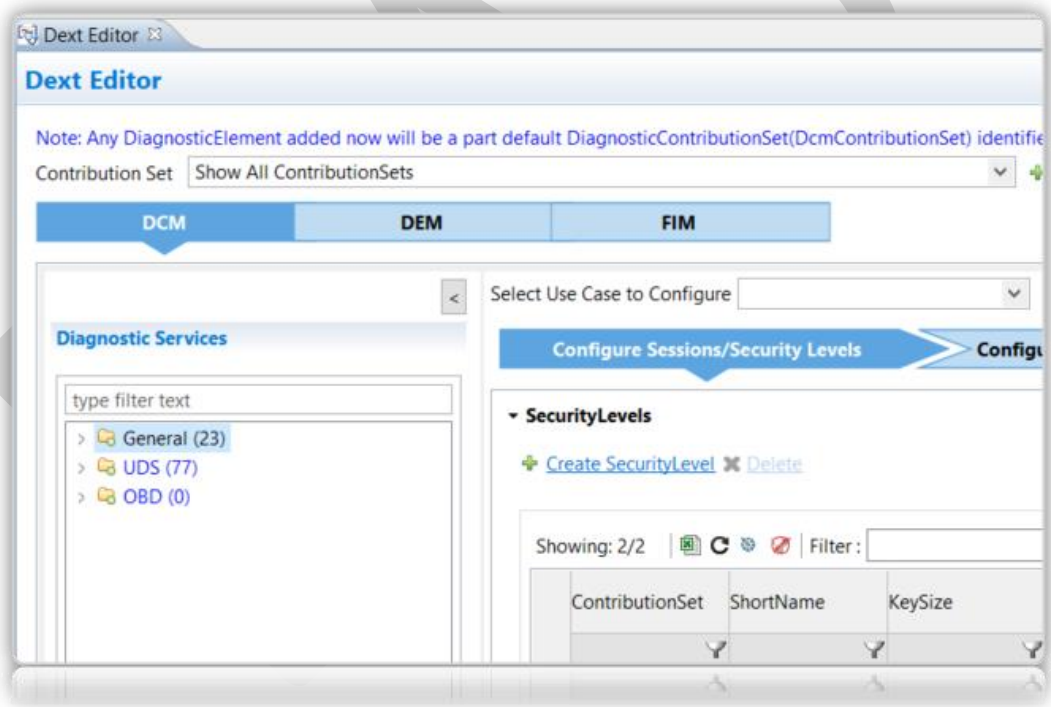
d. NM Pdu: AUTOSAR Network management PDUs for NM over CAN

Network Cluster	Type of Network	Network	ECU Node	Type of Frame	Type of Pdu
AUTOSAR NM	CAN	Can_Network_ETAS	ETAS	CAN Frame	I-Signal IPdu
					Dcm IPdu
					General Purpose IPdu
					NM Pdu
	Ethernet	N.A.	N.A.	N.A.	N.A.
	Flexray	N.A.	N.A.	N.A.	N.A.
	LIN	N.A.	N.A.	N.A.	N.A.

3.1.2 Diagnostic Description

AUTOSAR 4.2 has standardized diagnostic database by system level DEXT which could be exchanged between OEMs and Tiers.

ETAS BIP has shipped with one standard DEXT template which can be automaticly deployed into Ecu level configuration referencing [3.5 How to deploy System to EcuC] to accerate user configuration and integration speed.



3.2 How to update CAN Network Description

BIP ships with a tool Cobra_DBCImport which is bounded to the RTA-CAR version and BIP.

User could iteratively create and update their ISOLAR system description by using Cobra for

- add more networks by more DBCs
- add/remove/change frames
- add/remove/change signal descriptions

Integration Hints: [$BIP_INTG 003] When user changed any network descriptions, he/she shall

- **Step1**: Update new signals/PDUs/Network to ECU by following section [3.5 How to deploy System to EcuC]
- **Step2**: Generate BSW code for updates signals/PDUs by following section [4.4 How to generate BSW Code]. If user has changed name of ECUs, name of PDUs, please also perform EcuC arxml integration in section [4.2 Integrate Per-module]
- **Step3**: update signal interfaces following section [5.2 ComCluster – How to update Signal Interfaces]

3.2.1 Setup ***Cobra_DBCImport***

Input:

`Parameter.ini`, DBC files

Output:

System Description file

Arguments:

`--ProjectPath`: The path of the project obtained by selecting the project in navigator.

`--Ini`: The name of ini file.

Workflow:

1. Firstly, edit the `Parameter.ini` file at path `[ProjectRoot]\BasicSoftware` to configure the importation as shown in Figure 3. The highlighted parameters are those who normally could be modified by user.

Property	Description
ECUs	The ECU names can be listed in section `DBC_Parameters` and split with ','. **Hint:** the ECU who needs to be configured shall be placed at first place as target ECU.
DBC files	In section `DBC_FileList_Parameters`, DBC files for multiple channels can be configured. **Hint**: The DBC files must be stored in **DBC folder** under the project path. **Hint**: The format is **`<NodeName>@<NetworkName>=<DBC_file1.dbc>, <DBC_file2.dbc>`** (if multiple DBCs for one network available). - Here the `NodeName` refers to `CanCommunicationController` of Ecu in the system description.
Output filename	User can define own output file name by editing `Config_GenOutFileName`.

Baudrate and Can Driver specific settings	If all networks and can drivers use the same parameter values, user only need to configure section `CanBaudrate_default` and `CanControllerConfiguration_default`.
	If specific values shall be configured copy and paste these two sections and modify the section name as following format.
	Hint: if network support CanFd, `CANFD_Baudrate` line shall be uncommented.
	`CanBaudrate _<NetworkName>`
	`CanControllerConfiguration_<NodeName>`

Table 2 Cobra_DBCImport Parameter

Parameter.ini

```
[Arxml_Packagepath]
Config_BswArpackagePath = /ETAS_Project/EcucModuleConfigurationValuess

;***
[Project_Path]
ParamDef_Path = /ecu_config/paramdefs

;***                    target ECU
[DBC_Paramters]
Ecu_Config_Node = BDU,AC,ACU,AFS,APS,APTC,AVAS,BMS,DCDC,DSM,DVR,EAC,EPS,ESP,HU

;*** (NodeName@NetworkName=DBC files)
[DBC_FileList_Paramters]
CANNODE_ADAS@Can_Network_0_ADAS =                                    .dbc
CANNODE_Body@Can_Network_1_Body =                                    .dbc
CANNODE_Diag@Can_Network_2_Diag =                                    .dbc
CANNODE_Info@Can_Network_3_Info =                                    .dbc
CANNODE_PT@Can_Network_4_PT =                                .dbc
CANNODE_Tbox@Can_Network_5_Tbox =                                    .dbc


[ImportDbcConfigParameters]
Config_GenOutFileName = DBC_ImportGenOut.arxml
; Recommend AppendFrameNameToSignalName to be True
Config_AppendFrameNameToSignalName = True
Config_EnableCompuMethodsConvert = True
Config_DIAG = True
Config_NM = True
Config_XCP = True

;***
[CanBaudrate_default]
# Baudrate in kbps
CAN_BaudRate = 500
;CANFD_BaudRate = 2000   If CANFD included, uncomment this line

;[CanControllerConfiguration_<Node Name>]
[CanControllerConfiguration_default]
CAN_TimeSyn_PropSeg = 0
CAN_TimeSyn_Sjw = 0
CAN_TimeSyn_Tseg1 = 10
CAN_TimeSyn_Tseg2 = 0
CanControllerBaudRateConfigID = 1

CANFD_PaddingValues = 255
CANFD_TimeSyn_PropSeg = 0
CANFD_TimeSyn_Sjw = 0
CANFD_TimeSyn_Tseg1 = 10
CANFD_TimeSyn_Tseg2 = 0
CANFD_TrcvDelaycompensationOffset =
CANFD_TxBitRateSwitch = false

CanBusoffProcessing = POLLING
CanRxProcessing = POLLING
CanTxProcessing = POLLING
CanWakeupProcessing = POLLING
BaseAddress = 0
CanWakeupSupport = false
```

Figure 3 Cobra Parameter.ini

2. Place the `Parameter.ini` file and all necessary DBC files under the project path as following structure (Figure 4).

   `Parameter.ini` can be renamed as user will, but please also remember to edit the argument `--Ini` as section [3.2.2 Install and Run Cobra].

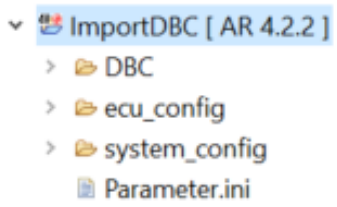
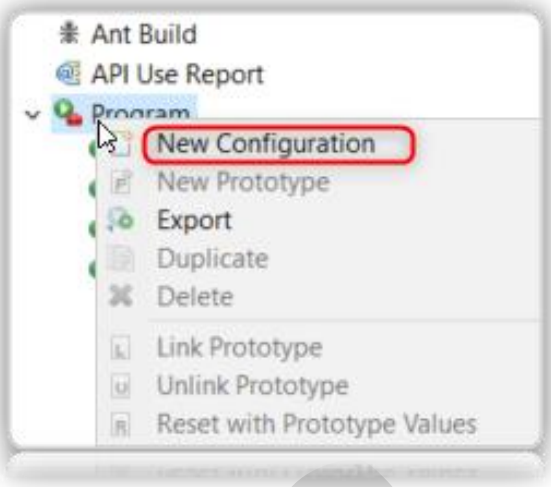


Figure 4 File structure for Cobra DBC import

3.2.2 Install and Run Cobra

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_DBCImport tool environment and Run it by click on "**Run**".

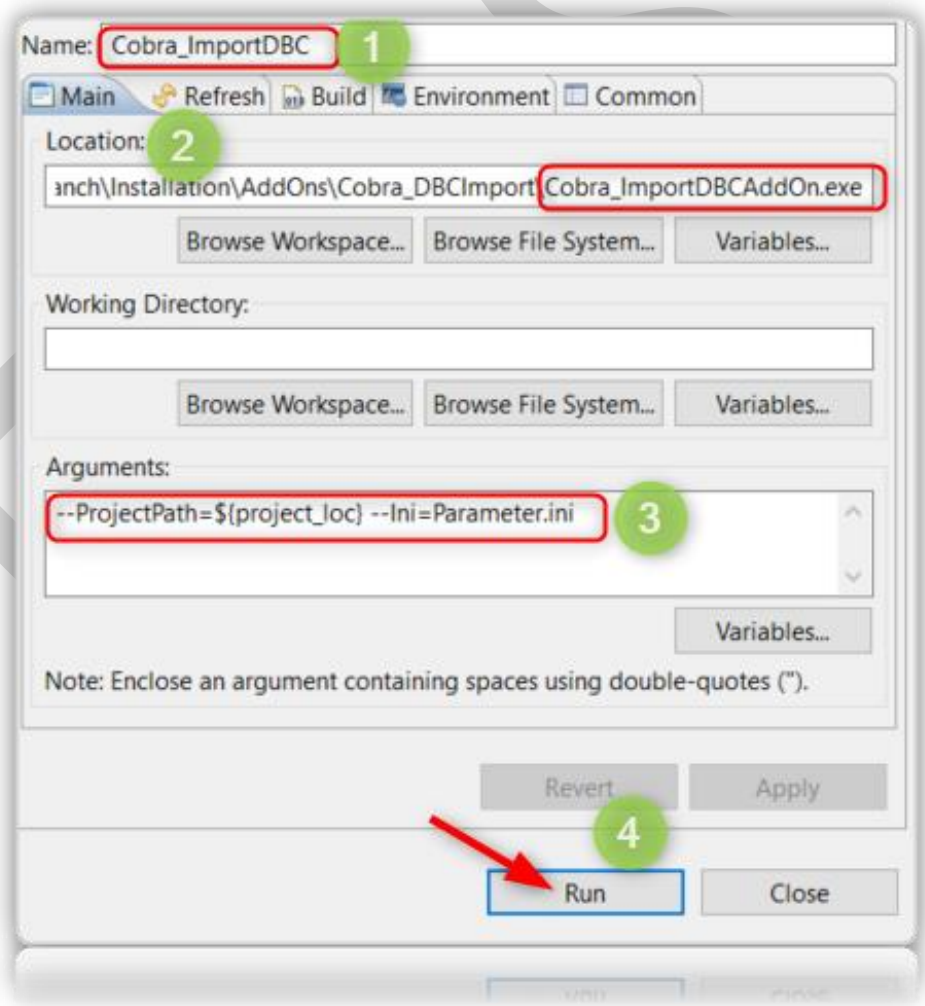


Figure 5 Install Cobra DBC import add-on

As a result of running Cobra import DBC, the console prints success information with system description file generated into BIP configuration root path
`[ProjectRoot]\BasicSoftware`

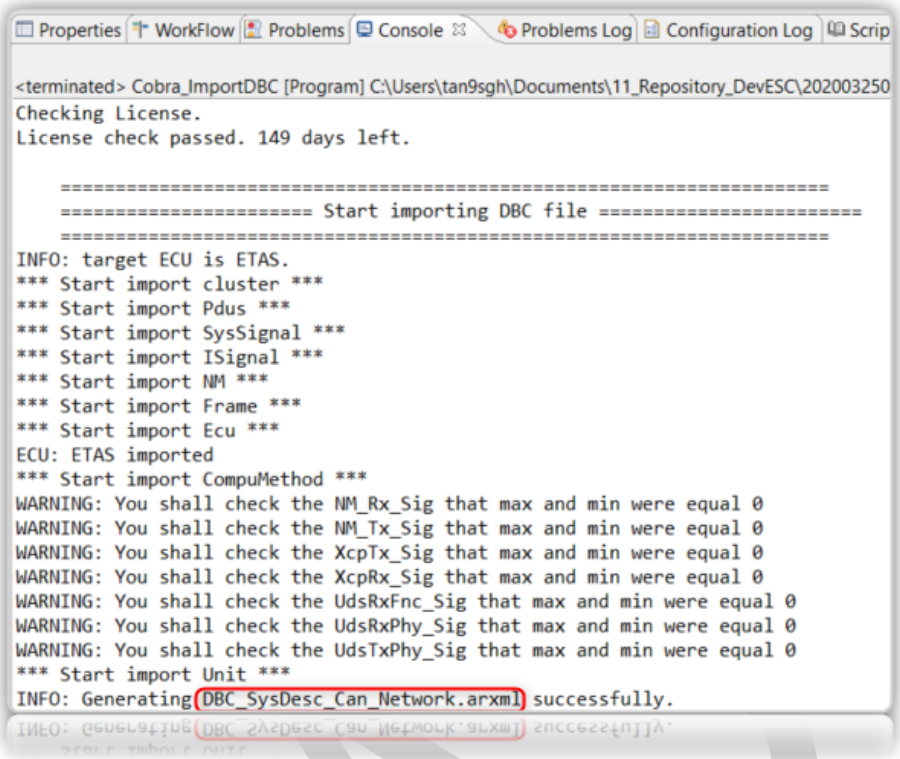


Figure 6 Cobra DBC Import Generation

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

> **NOTE**
> It intends to boost user iteration by following BIP DBC templates `[ProjectRoot]\BasicSoftware\DBC` and integration guide, and it not guarantee working on any RTA-CAR version with any user customized DBC formats.

3.3 How to update Ethernet Network Description

BIP ships with a tool Cobra_EthSysDesc which is bounded to the RTA-CAR version and BIP.

User could iteratively create and update their ISOLAR system description by using Cobra for

- add Ethernet network description
- Configure ethernet properties, e.g., Tcp, Udp, SoAd.

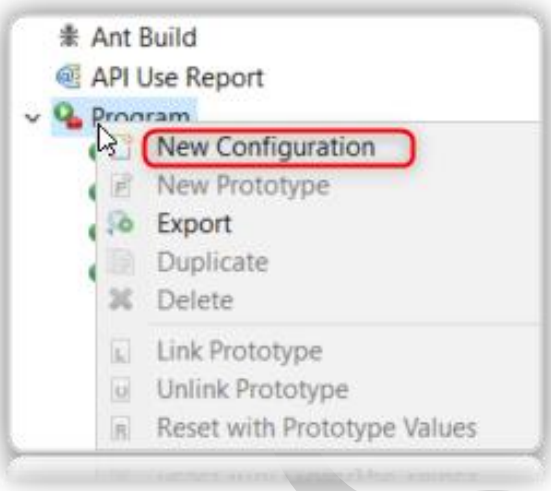
Integration Hints: [$BIP_INTG 003] When user changed any network descriptions, he/she shall

- **Step1**: Update new network parameter into `[ProjectRoot]\Ethernet_Config_Template.xlsx`
- **Step2:** run Cobra_EthSysDesc

Info: regarding how to configure `Ethernet_Config_Template.xlsx, please go to [ProjectRoot]\Docs\IntegrationGuide\How to configure Ethernet communication.pdf`

3.3.1 Install and Run ***Cobra_EthSysDesc***

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_EthSysDesc tool environment and Run it by click on "**Run**".

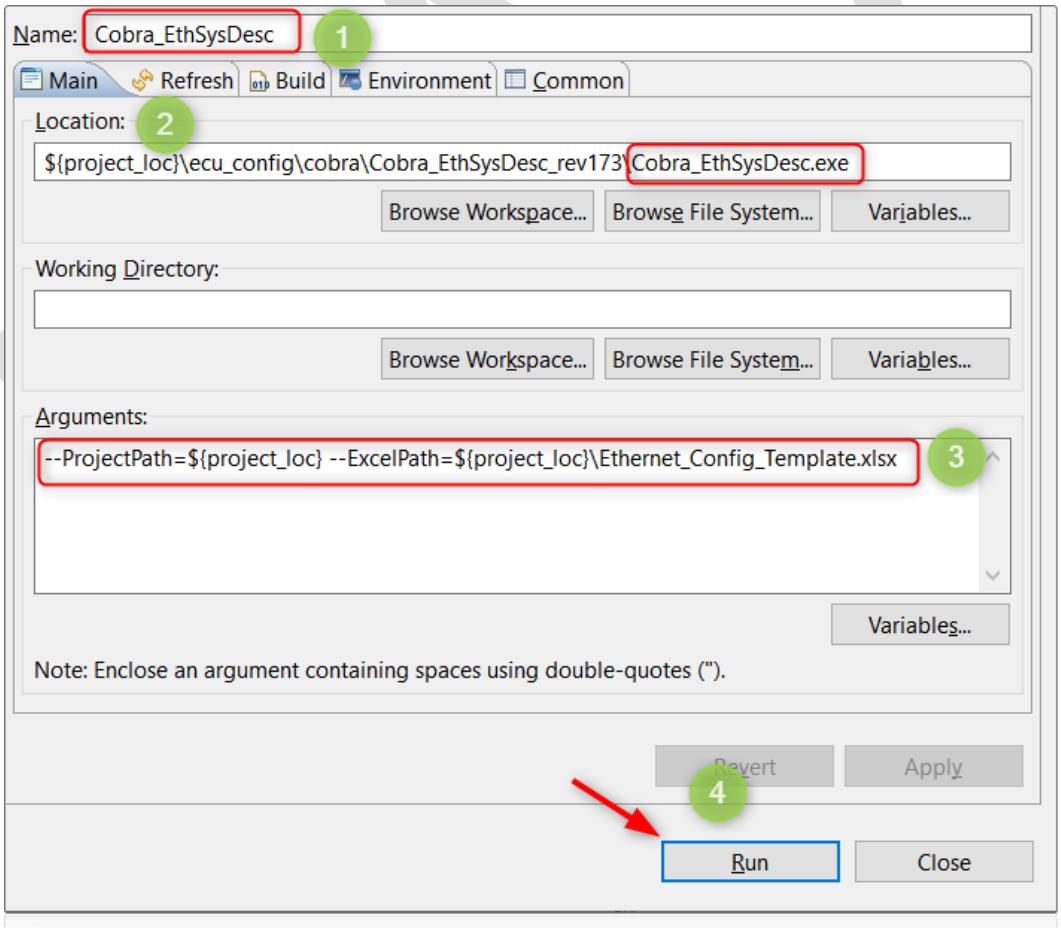


Figure 7 Install Cobra EthSysDesc add-on

As a result of running Cobra EthSysDesc, the console prints success information with system description file generated into BIP configuration root path
`[ProjectRoot]\BasicSoftware`

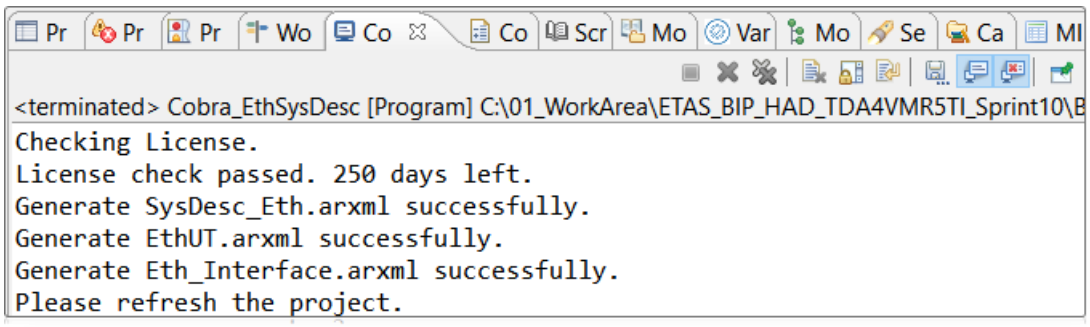


Figure 8 Cobra EthSysDesc Generation

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

3.4 How to update System Description

To construct AUTOSAR system description containing CAN network and Ethernet network with Cobra_DBCImport and Cobra_EthSysDesc, user also need to run Cobra_Before_ConfGen which helps combining network descriptions into ECU under BIP system description.

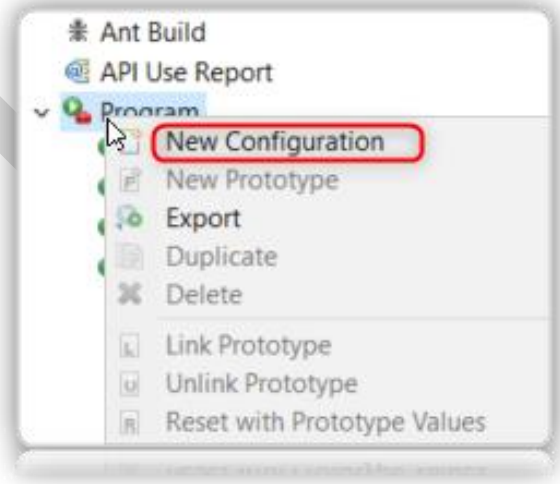
NOTE

Cobra_Before_ConfGen is a target dependent configuration add-on, it is not compatible on another RTA-CAR version and a different BIP target.

If you need to support another target or RTA-CAR version, please contact ETAS at [Contacting ETAS]

3.4.1 Install and Run ***Cobra_Before_ConfGen***

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_Before_ConfGen tool environment and run it by clicking on "**Run**".

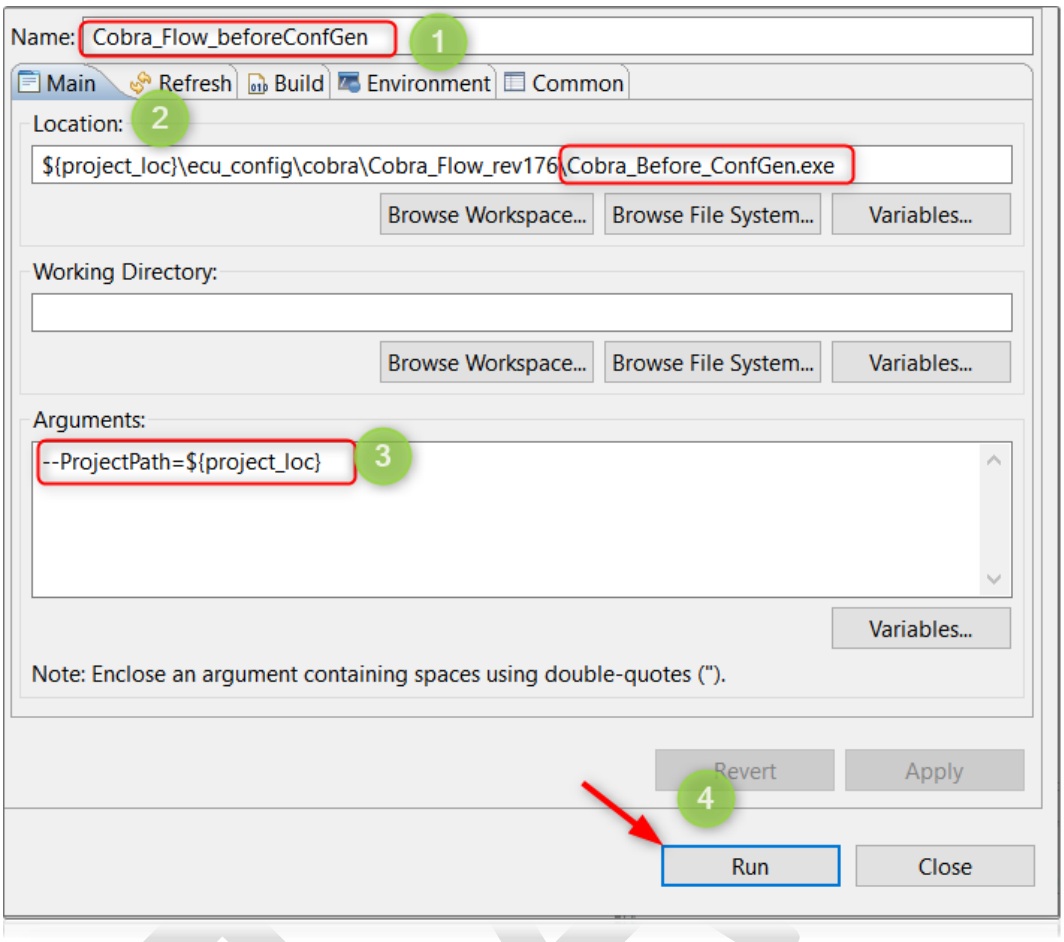


Figure 9 Install Cobra before ConfGen add-on

As a result of running Cobra beforeConfGen, the console prints success information with "*Please refresh your project*"

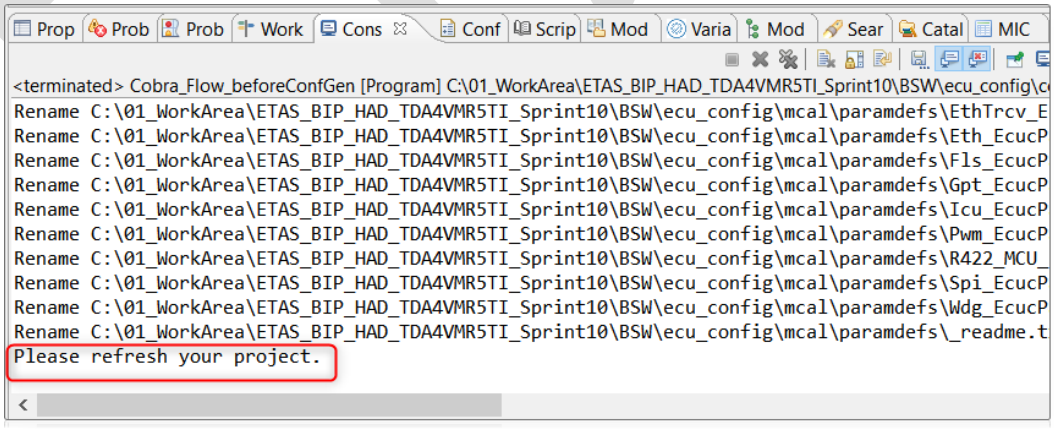


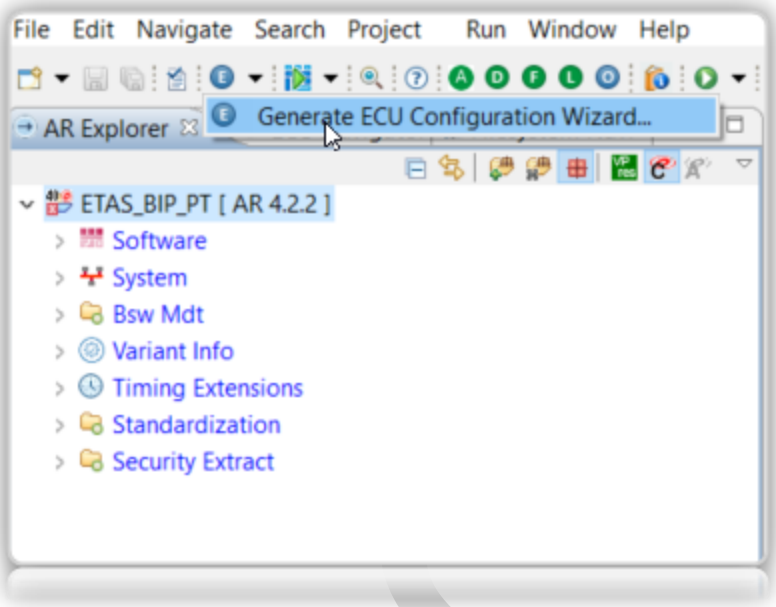
Figure 10 Cobra beforeConfGen Generation

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

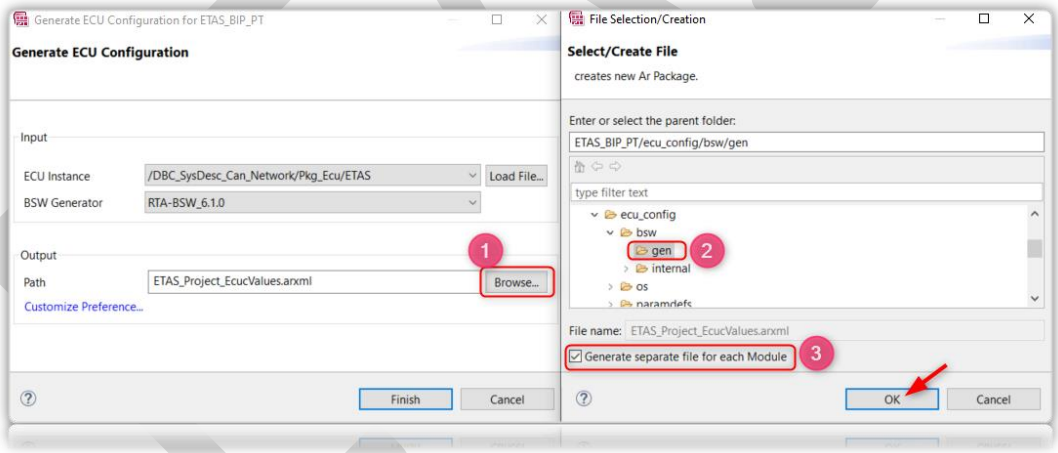
3.5 How to deploy System to EcuC

To deploy ECU "ETAS" referenced system and network information (network, frame, signals, etc.) to ECU, user shall click toolbar <**Generate ECU Configuration Wizard...**> to open configuration generation dialog.

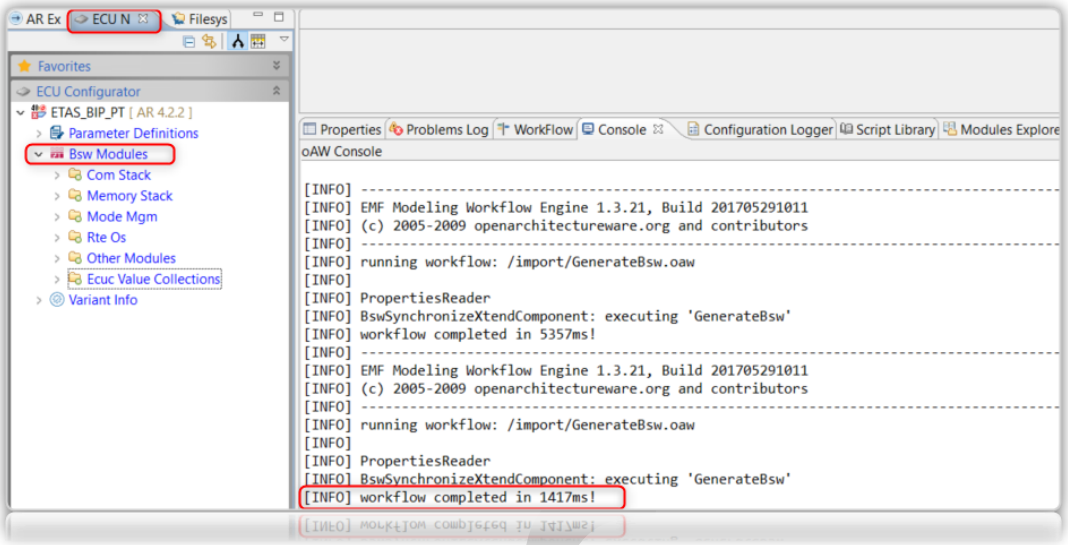


[$BIP_INTG 004] Select the **Output Path** to `[ProjectRoot]\BasicSoftware|ecu_config\bsw\gen` and check <**Generate separate file for each Module**>, then confirm by click <**OK>** to start generating configuration arxml files to the output path.

Info: If you could not find folder 'gen', please create it.



Info: The **Console** window will show a successful deployment of system description to EcuC in ISOLAR-B **ECU Navigator** window.



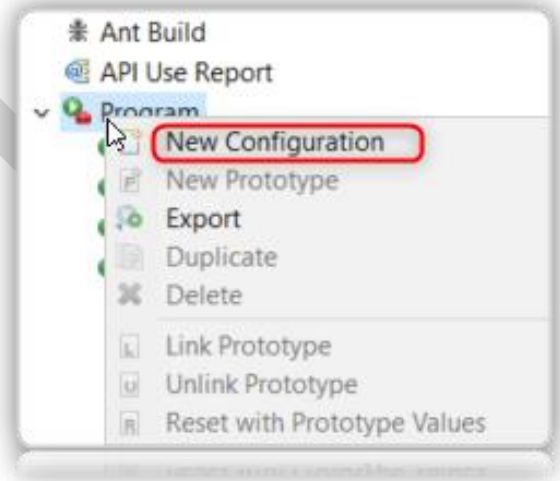
3.5.1 Post-update EcuC Generation by ***Cobra_After_ConfGen***

After ECU Configuration generation, for a specific target BIP user should run **Cobra_After_ConfGen** to post update some target dependent parameter definitions that will need to be configured in [4 ECU Configuration]

> **NOTE**
>
> **Cobra_After_ConfGen** is a target dependent configuration add-on, it is not compatible on another RTA-CAR version and a different BIP target.
>
> If you need to support another target or RTA-CAR version, please contact ETAS at [Contacting ETAS]

*Install and Run **Cobra_After_ConfGen***

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_After_ConfGen tool environment and run it by clicking on "**Run**".

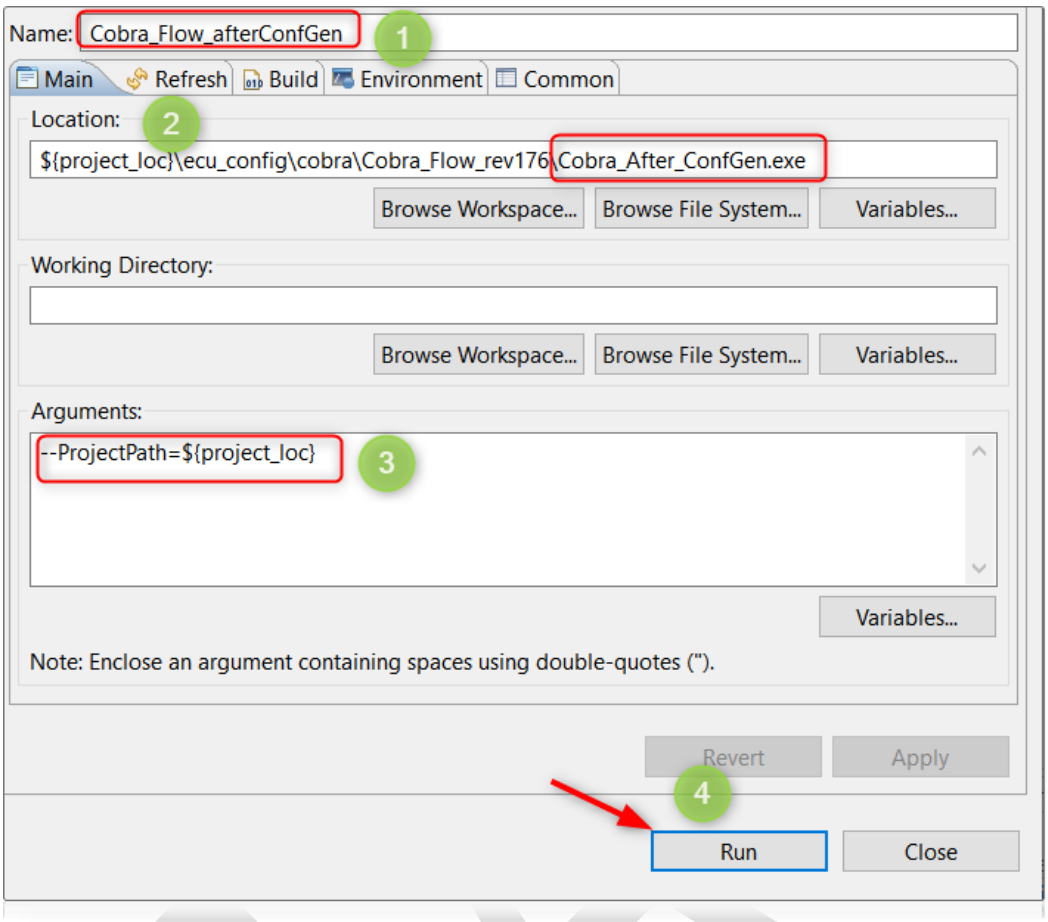


Figure 11 Install Cobra before ConfGen add-on

As a result of running Cobra afterConfGen, the console prints success information with "*Please refresh your project*"

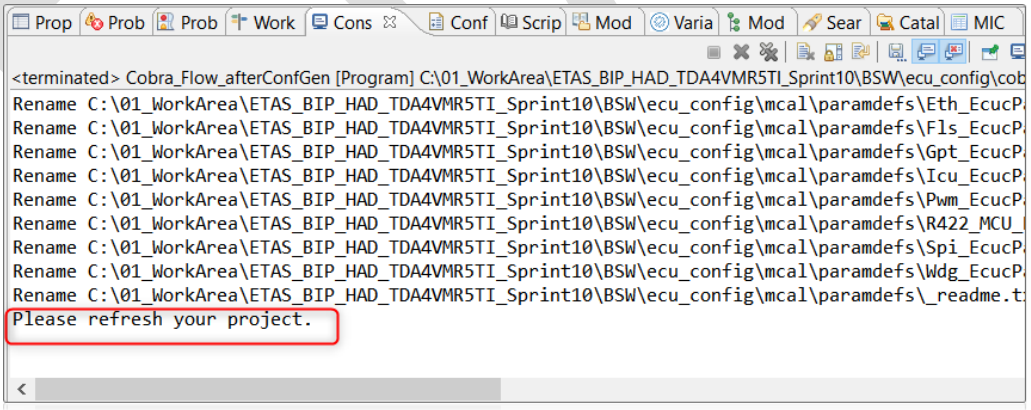


Figure 12 Cobra afterConfGen Generation

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

3.5.2 Customize EcuC Communication Parameters

AUTOSAR Network level communication and Ecu level communication stands on different functional view.

AUTOSAR communication description from network view (**ISOLAR-A System**) may not contain all information syntax completed at Ecu view (**ISOLAR-B ECU Navigator**) for code generation.

As a result, user will need to review and **manually** complete the communication configuration in ISOLAR-B ECU Navigator for missing parameters from network view.

- ❖ This requires comprehensive knowhow on both AUTOSAR BSW standards and RTA-CAR AUTOSAR toolchain features.

Hints: [$BIP_INTG 005] BIP provided a property fie `[ProjectRoot]\BasicSoftware\ecu_config\bsw\settings\algo.properties` that allows customize value of those parameters, user could adapt those parameters as per project needs. You can find more information about grammar of `algo.properties` in RTA-BSW_User_Guide_EN.pdf Section 3.5.2.

3.5.3 Generate ComSignal Receive callbacks by SignalMapping

[Auto-Mapping System Signal] introduces how System signals are mapped to DataElements of SenderReceiverInterfaces automatically under `System\SystemDataMapping`.

When these SenderReceiverToSignalMappings already exist in system description, <**Generate ECU Configuration Wizard...**> will generate additional Com module configuration for **RX** signals.

- `ComNotification`: RTE callback function name called when a signal is received.
- `ComTimeoutNotification`: RTE callback function name called when a signal receive timeout is reached.

Hints: [$BIP_INTG 006] With the **SystemDataMapping** provided in `System`, BIP places the `System` to the same arxml file of communication matrix (PDUs, signals…) so that <**Generate ECU Configuration Wizard...**> will generate `/Com/ComConfig/ComSignal/ComNotification` of RX ComSignals.

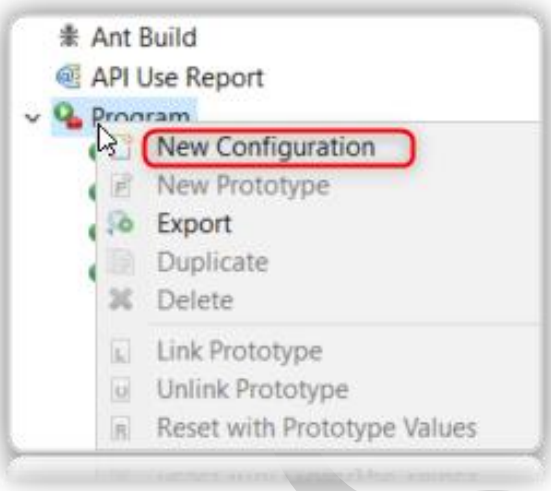
3.6 How to generate advanced ComSignal callbacks with BIP Cobra

AUTOSAR Com module provides below transmit call-backs to application through RTE:

- `ComNotification`: Application been notified by `Com_CbkTxAck` through RTE each time a message successfully transmitted.
- `ComTimeoutNotification`: Application been notified by `Com_CbkTxTOut` through RTE each time transmit timeout occurs
- `ComErrorNotification`: Application been notified by `Com_CbkTxErr` through RTE each time transmission of frame stopped and no further transmit possible for the message.

3.6.1 Install and Run ***Cobra_ComCbk***

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_ComCbk tool environment and run it by click on "**Run**".

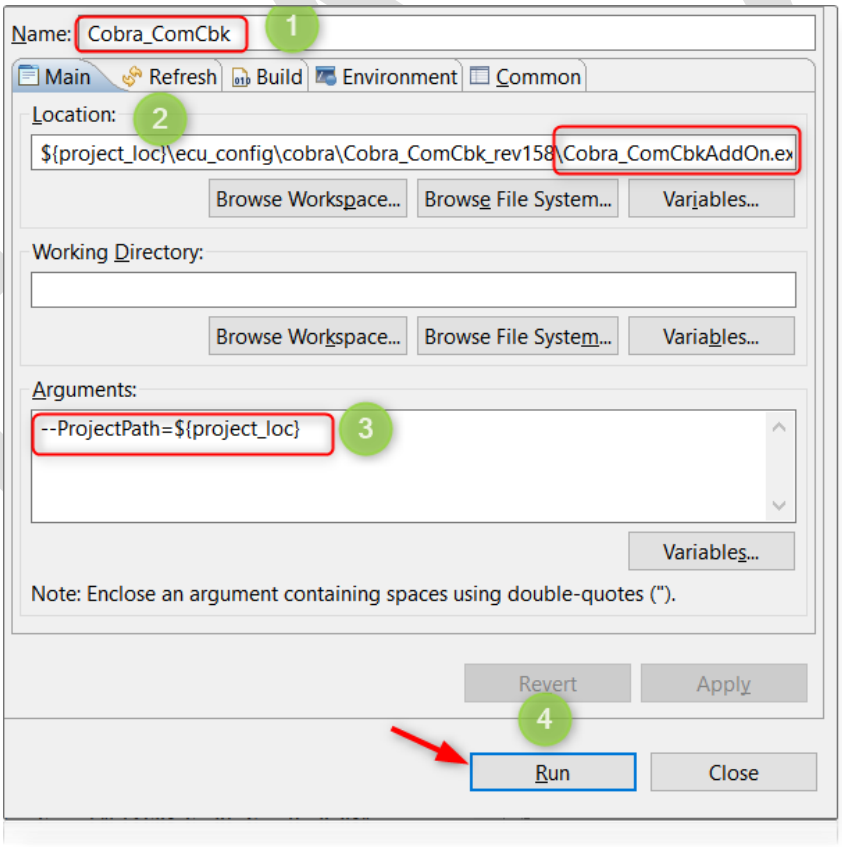


Figure 13 Install Cobra DBC import add-on

Then the console print information asking what kind of callbacks user are expected to configure.

- Key in '`Y`' confirm to generate that kind of Com callbacks or key in '`N`' to discard configuration of those Com callbacks.

```
                    ======================================================================
INFO: No timeout is configured in system description (under ecu->signal port).
Do you want to add ComNotification for Rx signals refer to SystemDataMapping? Y | N:
y
Do you want to add ComInvalidNotification for Rx signals who have ComFilters? Y | N:
n
Do you want to add ComNotification for all Tx signals? Y | N:
y
INFO: Please create Transmission Acknowedgement Request under PPort of SWC so that RTE can generate Ack Call-backs for TX signals.
Do you want to add ComErrorNotification for Tx signals? Y | N:
n
INFO: Edit Com Module successfully.Updated in file C:\01_WorkArea\ETAS_BIP_HAD_TDA4VMR5TI_Sprint10\BSW\ecu_config\bsw\gen\ETAS_Pro
```

As a result of running Cobra_ComCbk, the console prints success information with updated `ETAS_Project_Com_EcucValues.arxml` file generated into BIP configuration root path `[ProjectRoot]\BasicSoftware\ecu_config\bsw\gen`

```
=====================
al port).
Mapping? Y | N:

Filters? Y | N:


f SWC so that RTE can generate Ack Call-backs for TX signals.


S_BIP_HAD_TDA4VMR5TI_Sprint10\BSW\ecu_config\bsw\gen\ETAS_Project_Com_EcucValues.arxml
```

Figure 14 Cobra ComCbk Generation

Compatibility

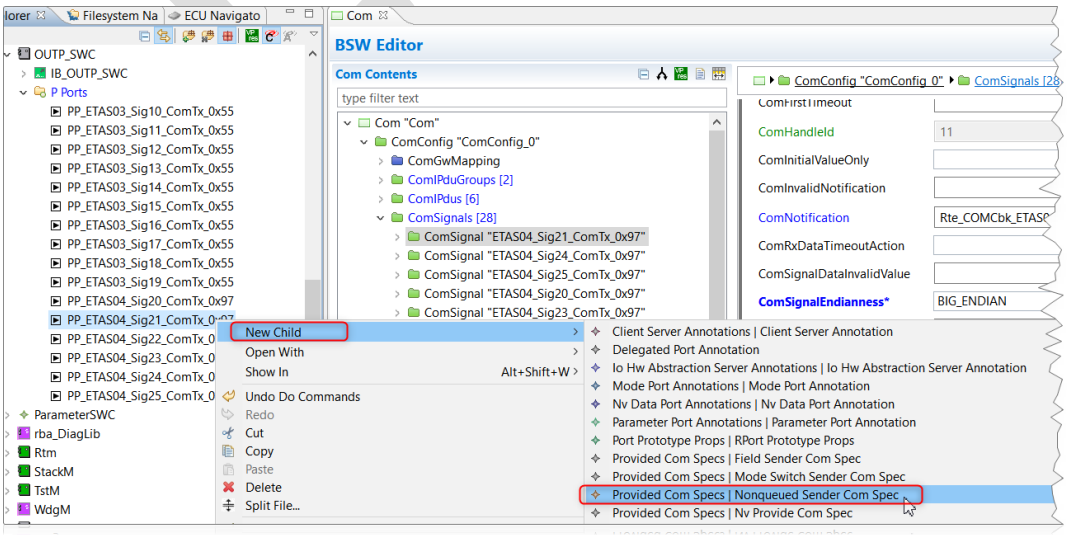
Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

3.6.2 Declare needs of `ComNotification` Rte Call

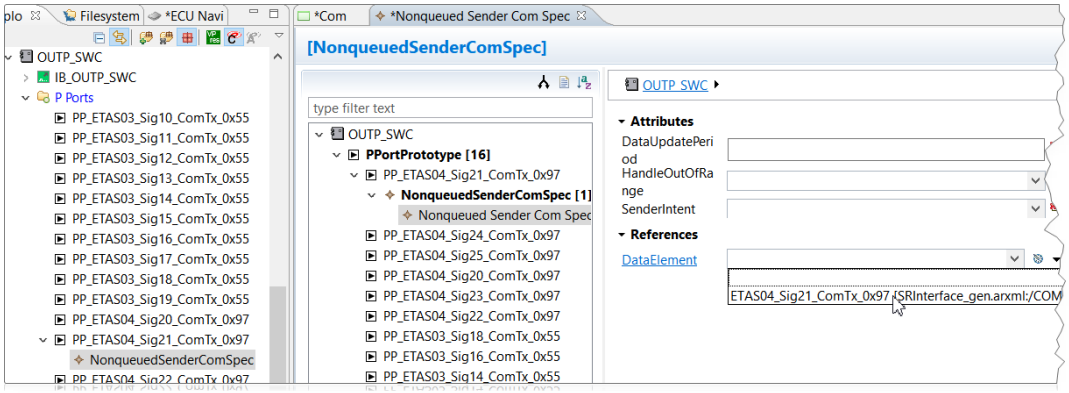
RTE allows to generate **AUTOSAR Interface** and connect to Com module Standardized Interface `ComNotification`.

To let Rte know how to generate AUTOSR Interface, use shall describe which software component will use which provided port to get that notification.

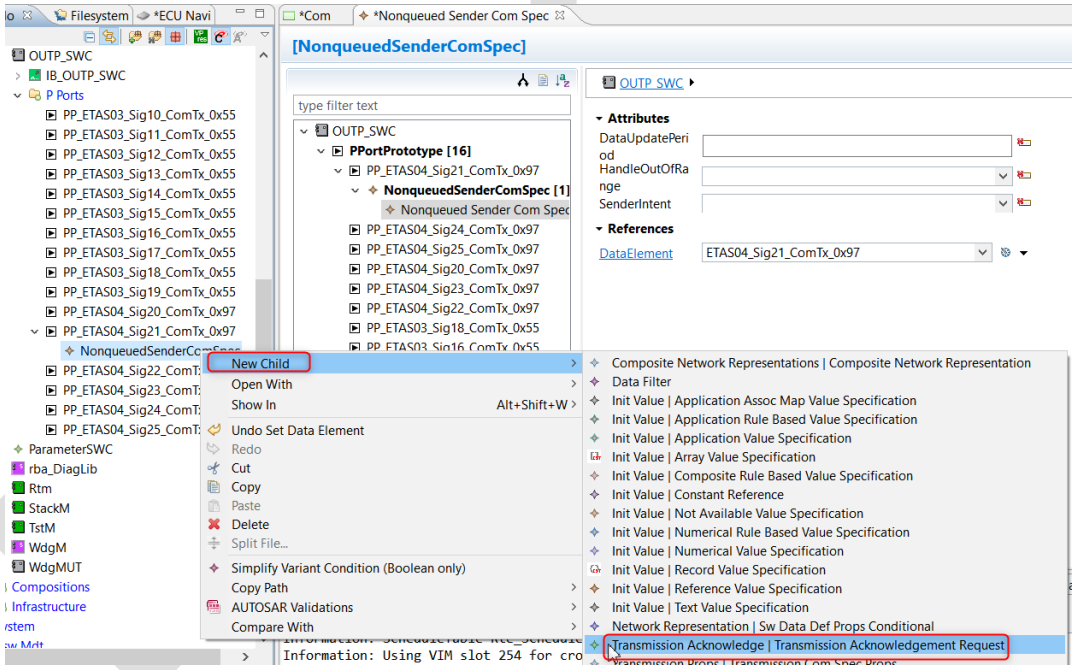
1) First, we need to create Nonqueued Sender Com Spec for PPorts



2) Then refer "`DataElement`" to required element (AUTOSAR Signal)



3) And create "`Transmission Acknowledgement Request`". After this step, RTE can generate Ack Call-back s for TX signals.



NOTE

`ComTimeoutNotification` and `ComErrorNotification` aren't generated by RTE so that their implementation should be designed and written by users.

3.7 Migrate to new RTA-CAR

When migrating to a new RTA-CAR version, user need to double check whether the BIP system description fit for the newer RTA-CAR version by perform [3.5 How to deploy System to EcuC

- If deployment failed with errors, user shall check error information from the **Console** window and correct the BIP system descriptions accordingly.

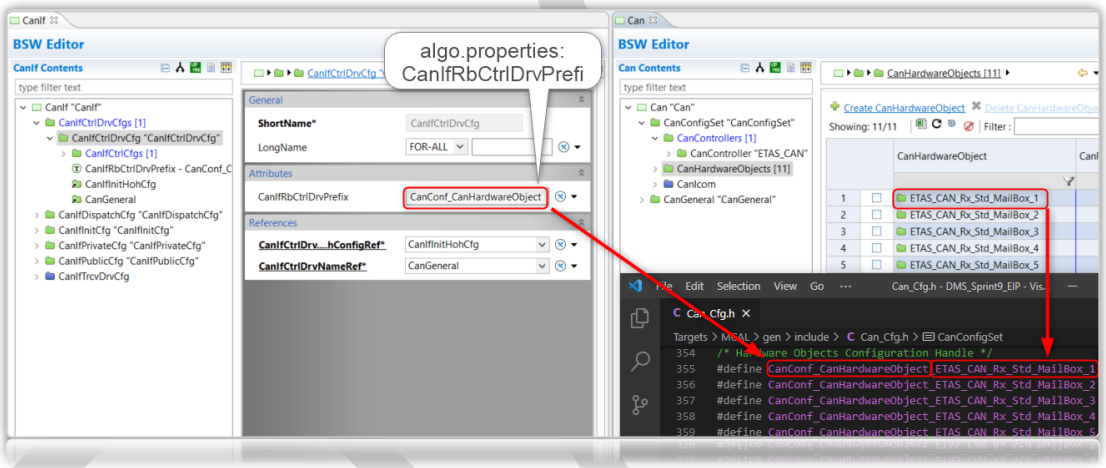
3.8 Migrate to new Target

When migration to a new target with vendor specific MCAL, below modification shall be followed.

3.8.1 CanIf - Can Integration

[$BIP_INTG 007] CanIf-Can: Before perform auto-deployment, user could update specific MCAL dependent CAN hardware mailbox prefix by `CanIfRbCtrlDrvPrefix` of `[ProjectRoot]\BasicSoftware\ecu_config\bsw\settings\algo.properties` to allow those prefixes to be auto-configured to EcuC Can module, for example:

- For IFX MCAL, `CanIfRbCtrlDrvPrefix = Can_17_MCanPConf_CanHardwareObject`
- For Renesas MCAL, `CanIfRbCtrlDrvPrefix = CanConf_CanHardwareObject`
- For NXP MCAL, `CanIfRbCtrlDrvPrefix = CanConf_CanHardwareObject`
- For ARC MCAL, `CanIfRbCtrlDrvPrefix = CanConf`



3.8.2 Can Mailbox Reordering

[$BIP_INTG 008] Many target MCALs and its configuration tool requires that the Can mailboxes are presented in a particular order. This can be supported by setting a mailbox mapping rule in the `algo.properties` file.

- If the `MbSortingPref` rule is specified, the mailboxes will be ordered alphabetically (e.g., A, B, C, D…) with a comma separated list of sorting criteria:

```
MbSortingPref=criteria1, criteria2,...
```

- The criteria can be one of:
  - `canHandleType`
  - `canAddressingMode`
  - `direction`
  - `canControllerName`
  - `mask`
  - `canControllerId`

- The ordering for criteria will be reversed by prepending the optional `~` to the desired criteria. For example: `~controllerName` will search controller name from …, D, C, B, A.

```
MbSortingPref=direction,~controllerName,canHandleType
```

3.9 Migrate to new Compiler

Nothing need to be modified when user migrating project to a new compiler.

ETAS

4 ECU Configuration

This section describes:

- Section 4.2: BSW peer-module **arxml integration** every time after system deployment at [3.5 How to deploy System to EcuC].
- Section 4.3: BSW module static configuration **in BIP as baseline by ETAS experts**.

> **NOTE**
> **As a starter with ETAS AUTOSAR, users are strongly recommanded to get familiar with BIP from configuraiton to generation followed by integration before they start modification.**

4.1 Split BSW Configuration with ISOLAR-B

AUTOSAR BSW configuration is more or less building up **LEGO** where modular is important for iterative development as teamwork.

[$BIP_INTG 009] ETAS BIP ships with split BSW static configurations (a module configuration has been contained in several arxml files) at `[ProjectRoot]\BasicSoftware\ecu_config\bsw\static`.

Modifying BIP

[$BIP_INTG 010] When user would modify the static configurations in **ISOLAR-B** | **ECU Navigator** view, he shall switch to **split view** before the modification so that his configuration is saved in expected configuration arxml file.

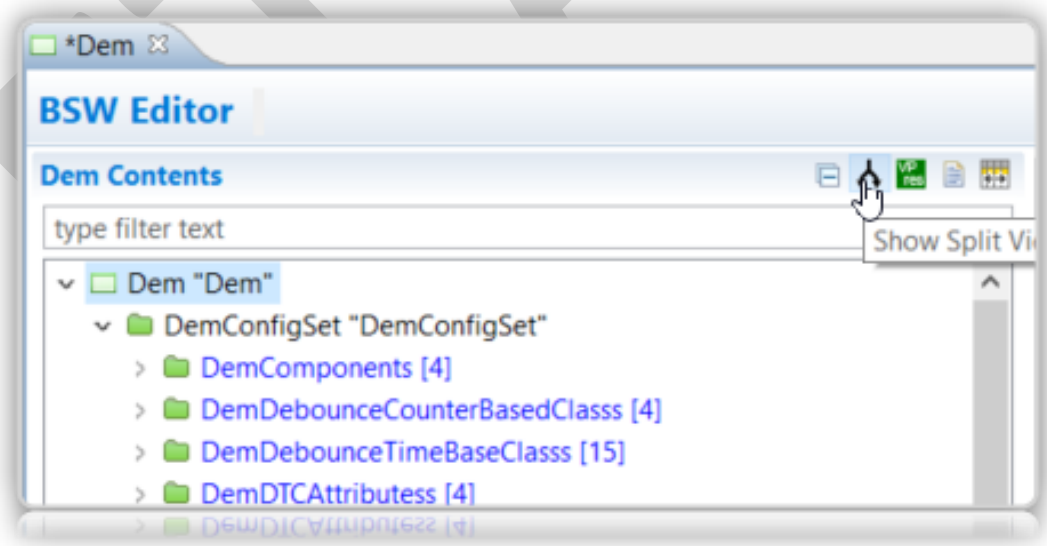


Figure 15 Split BSW Configuration

How to add EcuC Configuration to BIP

After becomes familiar with BIP, user would require modifying BIP to project needs by adding EcuC configuration arxml files.

To do so, user will go to **ISOLAR-B** | **ECU Navigator** view, right click **Bsw Modules** and select the required EcuC module to add.

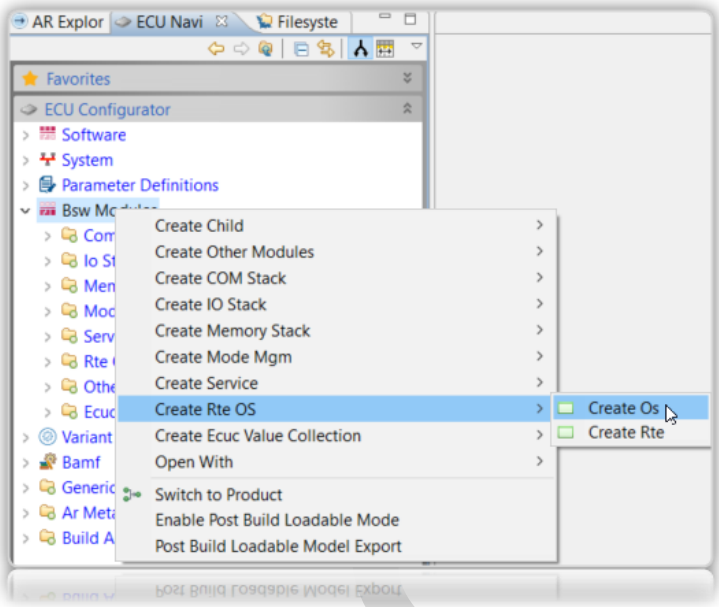
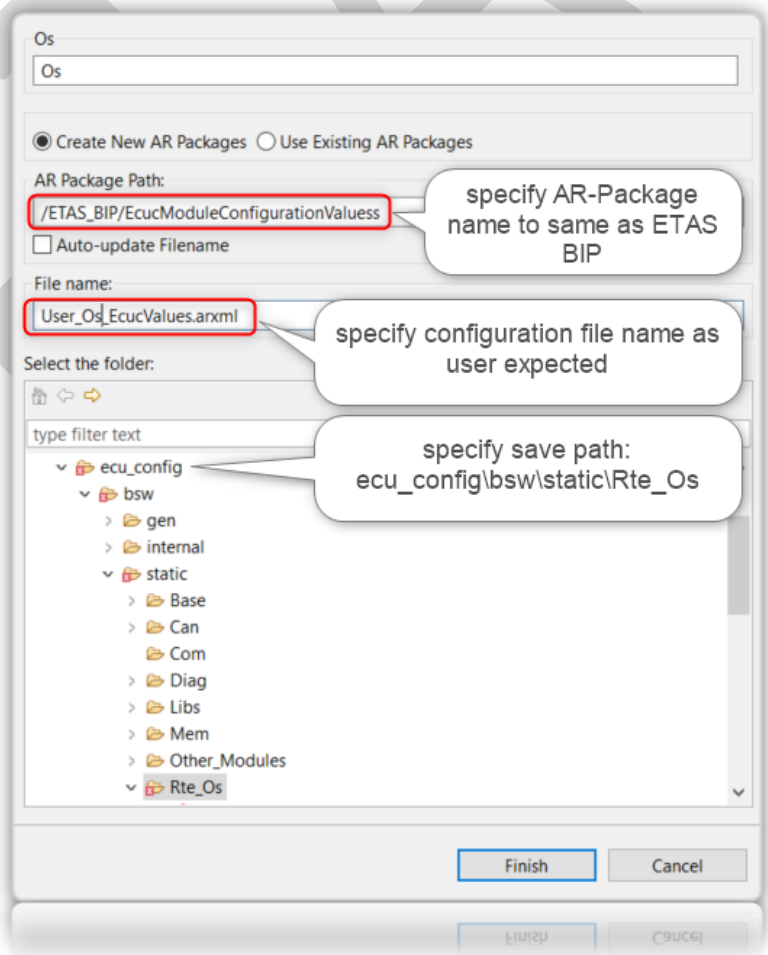


Figure 16 Create BSW Configuration

Then specify

- Component name: same component name as it has in BIP
- AR-package path: Same AR-package path as its already provided in BIP EcuC static configuration file
- File Name: user could name the file as they want



4.2 Integrate Per-module

[3.5 How to deploy System to EcuC] auto-deploy to Ecu view (**ISOLAR-B ECU navigator**) with a set of configuration file `ETAS_Project_[ModuleName]_EcucValues.arxml` for

- Communication related components
- Nv Needs related components

> **NOTE**
> BIP performs below arxml integration that user are recommanded to follow every time after system deployment, expect that project specific configuration needed.

4.2.1 EcuC Pdu Integration

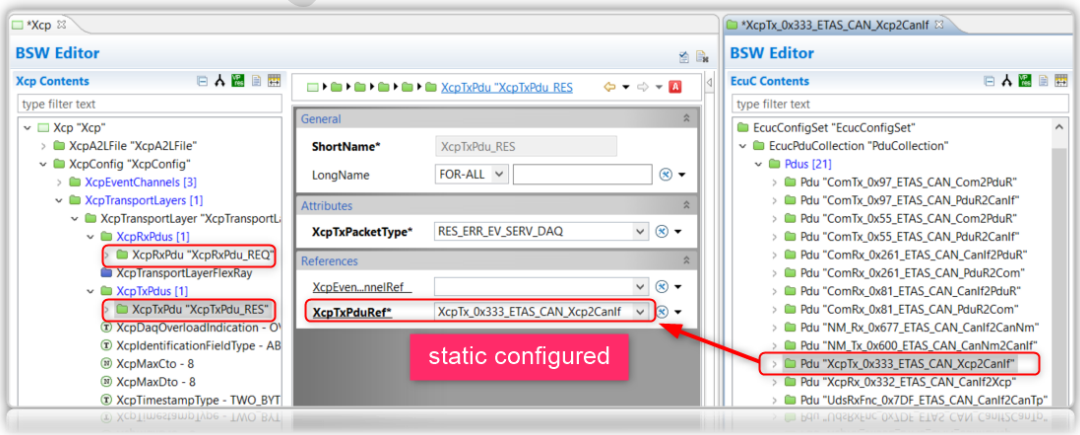
[3.5 How to deploy System to EcuC] auto-deploy a collection of PDUs from system description to EcuC module into `ETAS_Project_EcuC_EcucValues.arxml`.

The static BSW modules which need to reference to the generated EcuC PDUs shall be **staticlly** reference by user.

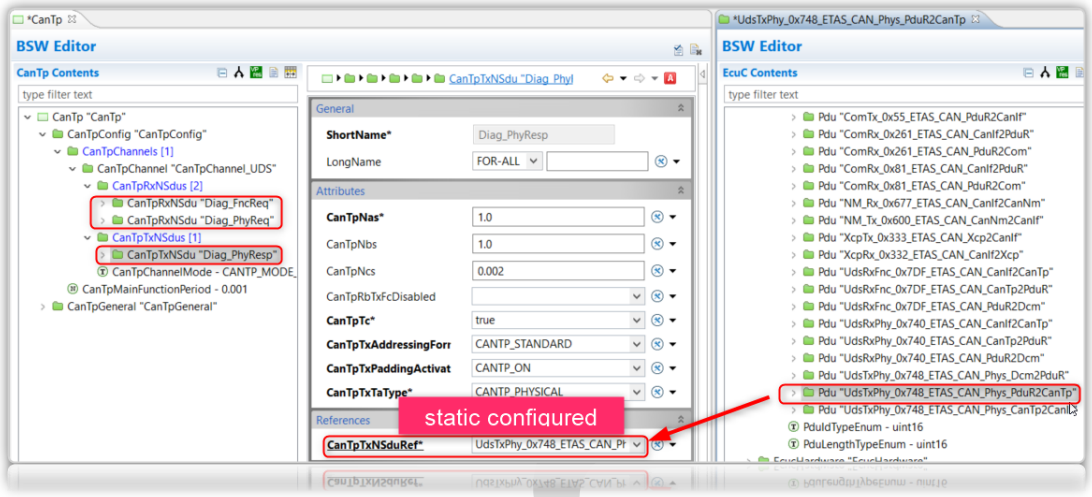
Source Module	Source Module Path	Dest. Module	Dest Module Path
EcuC	*_Xcp2CanIf	Xcp	[$BIP_INTG 011] XcpConfig\XcpTransportlayers**XcpTxPduRef**
EcuC	*_CanIf2Xcp	Xcp	[$BIP_INTG 012] XcpConfig\XcpTransportlayers**XcpRxPduRef**
EcuC	*_CanTp2PduR	CanTp	[$BIP_INTG 013] CantpConfig\CanTpChannels**CanTpRxNSdus**
EcuC	*_PduR2CanTp	CanTp	[$BIP_INTG 014] CantpConfig\CanTpChannels**CanTpTxNSdu**
EcuC	*_PduR2Dcm	Dcm	[$BIP_INTG 015] Dcm\DcmDsl\DcmDslProtocolRows\DcmDslConnection\DcmDslMainConnection**DcmDslProtocolRx**
EcuC	*_Dcm2PduR	Dcm	[$BIP_INTG 016] Dcm\DcmDsl\DcmDslProtocolRows\DcmDslConnection\DcmDslMainConnection**DcmDslProtocolTx**

Table 3 EcuC Pdu Reference Destination

For [GenericPurposeIpdu], BIP connect XcpConfig\XcpTransportlayers\[**XcpRxPduRef**]/[**XcpTxPduRef**] with EcuC Pdus Xcp*.



For [DcmIpdu], BIP connect CantpConfig\CanTpChannels\[**CanTpRxNSdus**]/[**CanTpTxNSdus**] with EcuC PDU Uds*.

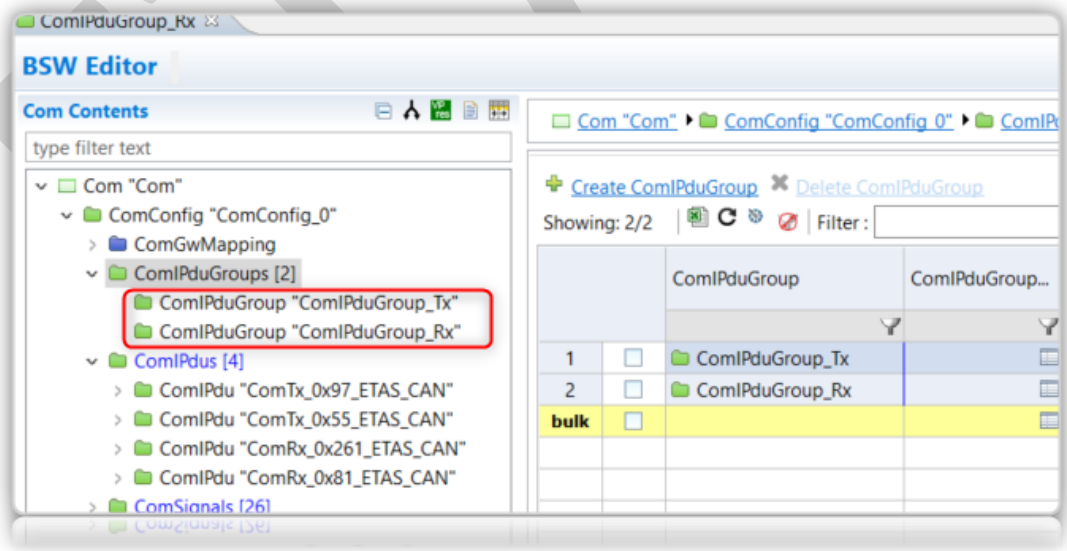


Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.2.2 Com - BswM Integration

[$BIP_INTG 017] In BIP, all `ISignalIPdus` in system are mapped to `ISignalIPduGroups`. Therefore, no default `ComIPduGroup_Rx`/`ComIPduGroup_Tx` will be generated into EcuC Com module during [3.5 How to deploy System to EcuC].

Since BSWM controls **enable**/**disable** send/receive PDU groups, we configure `ComIPduGroup_Rx`/`ComIPduGroup_Tx` in Com module as below.



Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

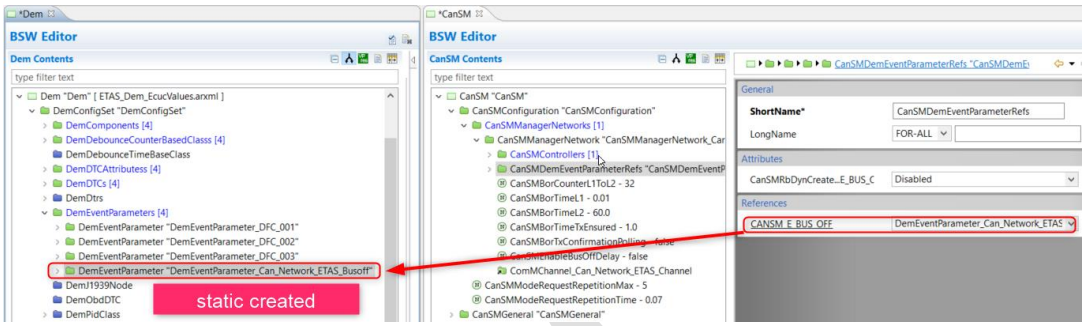
4.2.3 CanSM - Dem Integration

[3.5 How to deploy System to EcuC] generated CanSM contains one CanSM channel for each CAN channel into `ETAS_Project_CanSM_EcucValues.arxml`.

[$BIP_INTG 018] For each CanSM channel, BIP is configured with not create **Dem event** automatically during [4.4 How to generate BSW Code] but rather configure `DemEvent` statically.

To do so:

- Configure `CanSMRbDynCreate_CANSM_E_BUS_OFF` as `Disabled`.
- Created one `DemEventparameter` in Dem and be referenced by CanSM at `CANSM_E_BUS_OFF`.



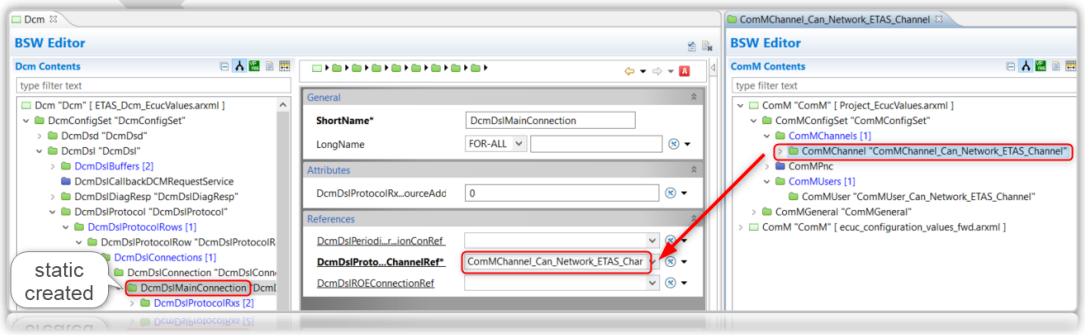
Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.2.4 ComM – Dcm Integration

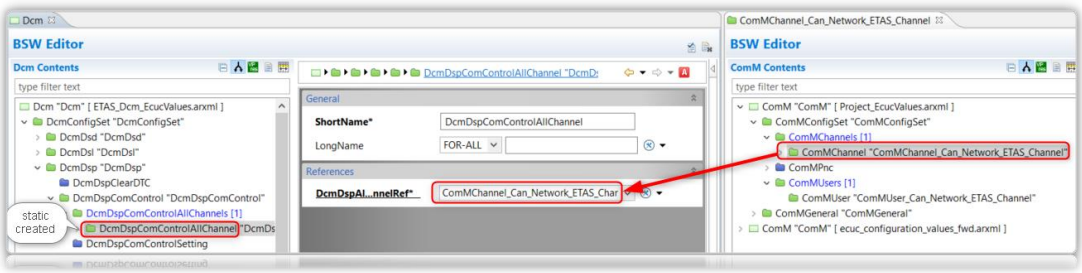
[3.5 How to deploy System to EcuC] generated ComM contains one ComM channel for each CAN channel into `ETAS_Project_ComM_EcucValues.arxml`.

[$BIP_INTG 019] For the ComM channel that the **DcmIPdu** will be received/transmitted, BIP is configured with `DcmDslMainConnection` and referencing to the ComM channel.

Source Module	Source Module Path	Dest. Module	Dest Module Path
ComM	ComMChannel	Dcm	`Dcm\DcmDsl\DcmDslProtocolRows\DcmDslConnection\DcmDslMainConnection\`**`DcmDslProtocolComMChannelRef`**



[$BIP_INTG 020] For the ComM channel that the Dcm will control its status of receive/transmit, BIP is configured with `DcmDspComControlAllChannel` and referencing to the ComM channel.



Source Module	Source Module Path	Dest. Module	Dest Module Path
ComM	ComMChannel	Dcm	`Dcm\DcmDsp\DcmDspComControl\DcmDspComControlAllChannels\DcmDspComControlAllChannel\`**`DcmDspAllComMChannelRef`**

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.2.5 Can - MCAL Integration

[3.5 How to deploy System to EcuCgenerated Can EcuC configuration that contains Can module into `ETAS_Project_Can_EcucValues.arxml`

[$BIP_INTG 021] This Can EcuC arxml file has been further imported to MCAL configuration and generation tool, please refer to section [4.6 How to generate MCAL].

4.2.6 CanIf Integration

[3.5 How to deploy System to EcuC] generated CanIf EcuC configuration that contains CanIf module into `ETAS_Project_CanIf_EcucValues.arxml`

4.3 Integrate Static Modules

BIP provided a set of BSW module static configuration **as baseline by ETAS experts**.

This section provides integration focus when **porting to new target/MCAL**.

Info: This section is not intend to document all module configuration details. User could modify module's static configuration to fit for project specific needs with [consulting to ETAS experts].

4.3.1 Platform Integration

BSW modules **Dcm** and **Xcp** both need to know target byte order for their own data handling.

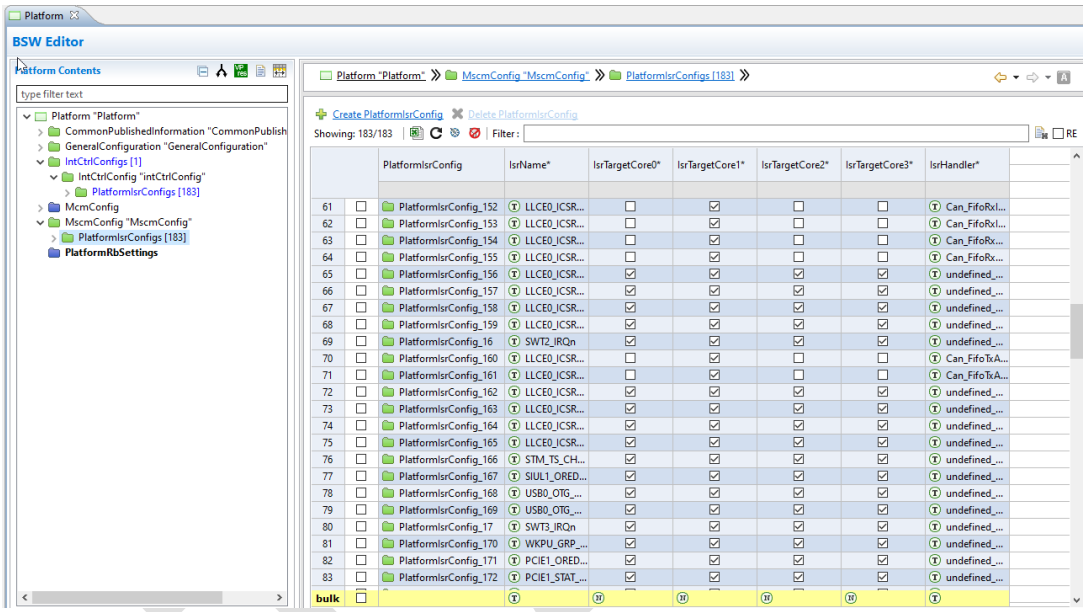
[$BIP_INTG 024] When user porting it to different target, user shall configure this module as MCAL supplied, and confirm `CPU_BYTE_ORDER` shall be `LOW_BYTE_FIRST` in `PlatformTypes.h/Platform_Types.h` so that match the Target hardware byte order.

For example, for little endian target like ARM, AURIX, S32G and Calterah, `CPU_BYTE_ORDER` shall be `LOW_BYTE_FIRST`.

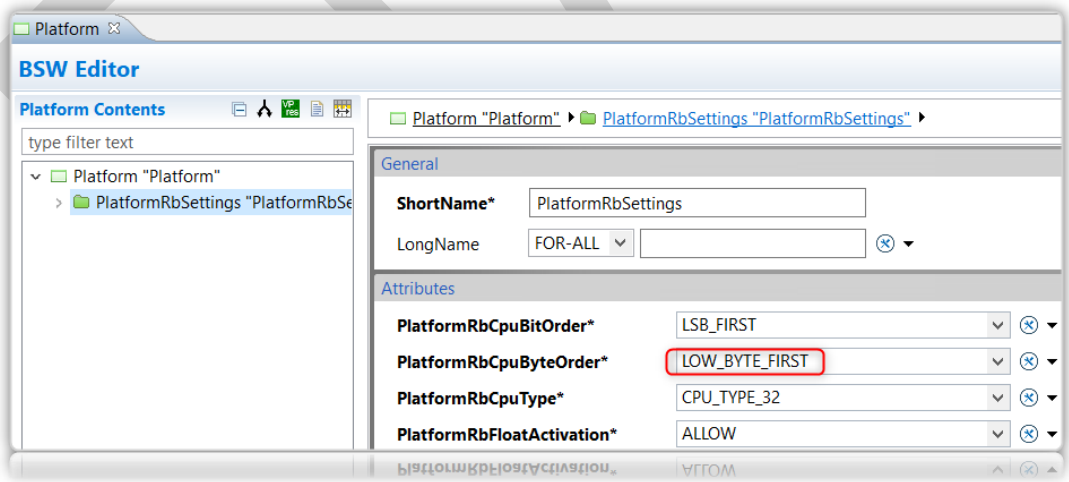
If MCAL supply this file, user should base on it

```
C PlatformTypes.h 4 ×
BasicSoftware > src > integration > types > C PlatformTypes.h > CPU_BYTE_ORDER
160
161   /**
162   * @brief The byte order on memory level shall be indicated in the platform types header file using
163   *        the symbol CPU_BYTE_ORDER.
164   * @implements     CPU_BYTE_ORDER_enum
165   */
166   #define CPU_BYTE_ORDER (LOW_BYTE_FIRST)
167
```

And this module shall be configured as MCAL supplied, such as S32G



If MCAL don't supply this file, user should configure this module in BSW



4.3.2 Det Integration

[$BIP_INTG 025] Det has be enabled for **every** module during development integration phase by set `[Module]DevErrorDetect` to `TRUE` to detect modules configuration and integration error as earlier as possible.

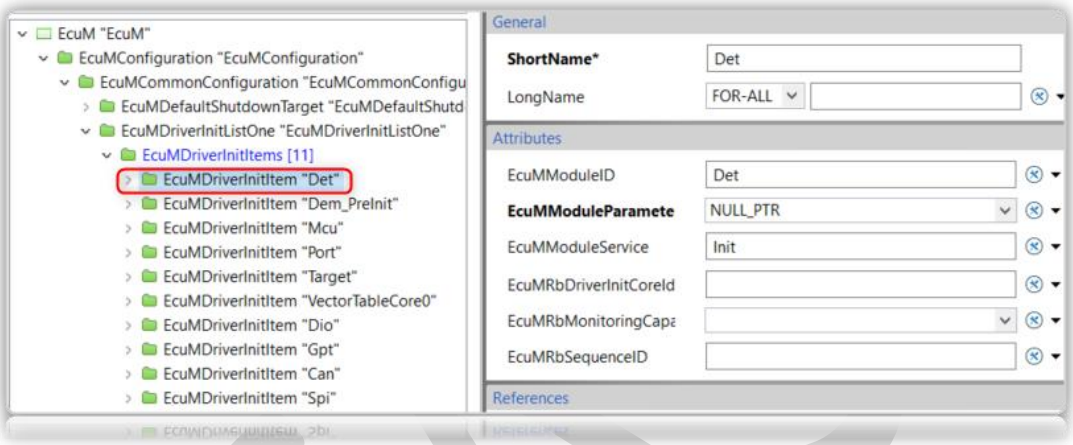
These modules include those generated during [3.5 How to deploy System to EcuC] as well as statically configured ones in `[ProjectRoot]\BasicSoftware\ecu_config\bsw\static`

Hints: Det report can be enabled by setting `all_DevErrorDetect=true` in `[ProjectRoot]\BasicSoftware\ecu_config\bsw\settings\algo.properties`

4.3.3 EcuM Integration

EcuM – Det

[$BIP_INTG 026] BIP configure Det initialization at the first one of EcuM `EcuMDriverInitListOne` list, so that development error report of other modules can be reported even during initialization phase.

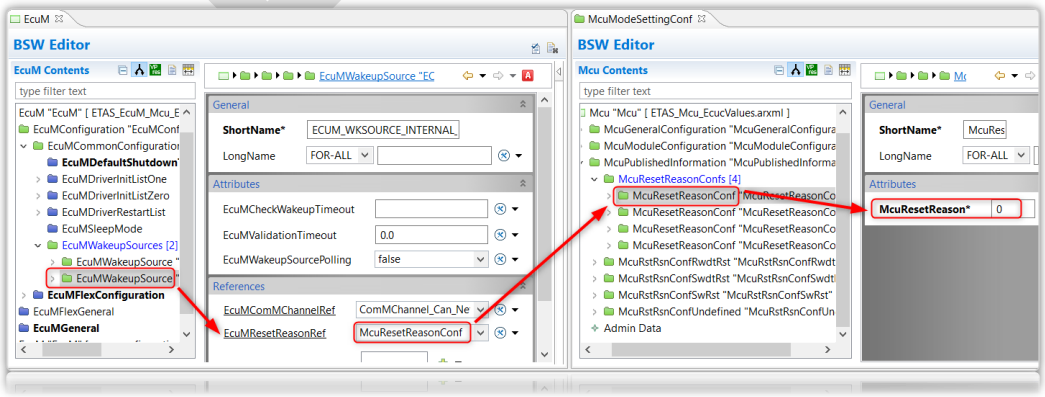


Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

EcuM – Mcu Reset

[$BIP_INTG 027] BIP integrate EcuM wakeup source with Mcu wakeup reason by

- Put MCAL Mcu **Parameter Definition File** into BIP `[ProjectRoot]\BasicSoftware\ecu_config\mcal\paramdefs`.
- Configure this Mcu specific **`McuResetReason`** `McuResetReasonConf` in **ISOLAR-B ECU Navigator** for Mcu module.
- Created `EcuMWakeupSource` to reference corresponding `McuResetReasonConf`.



Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

Porting BIP to new Target

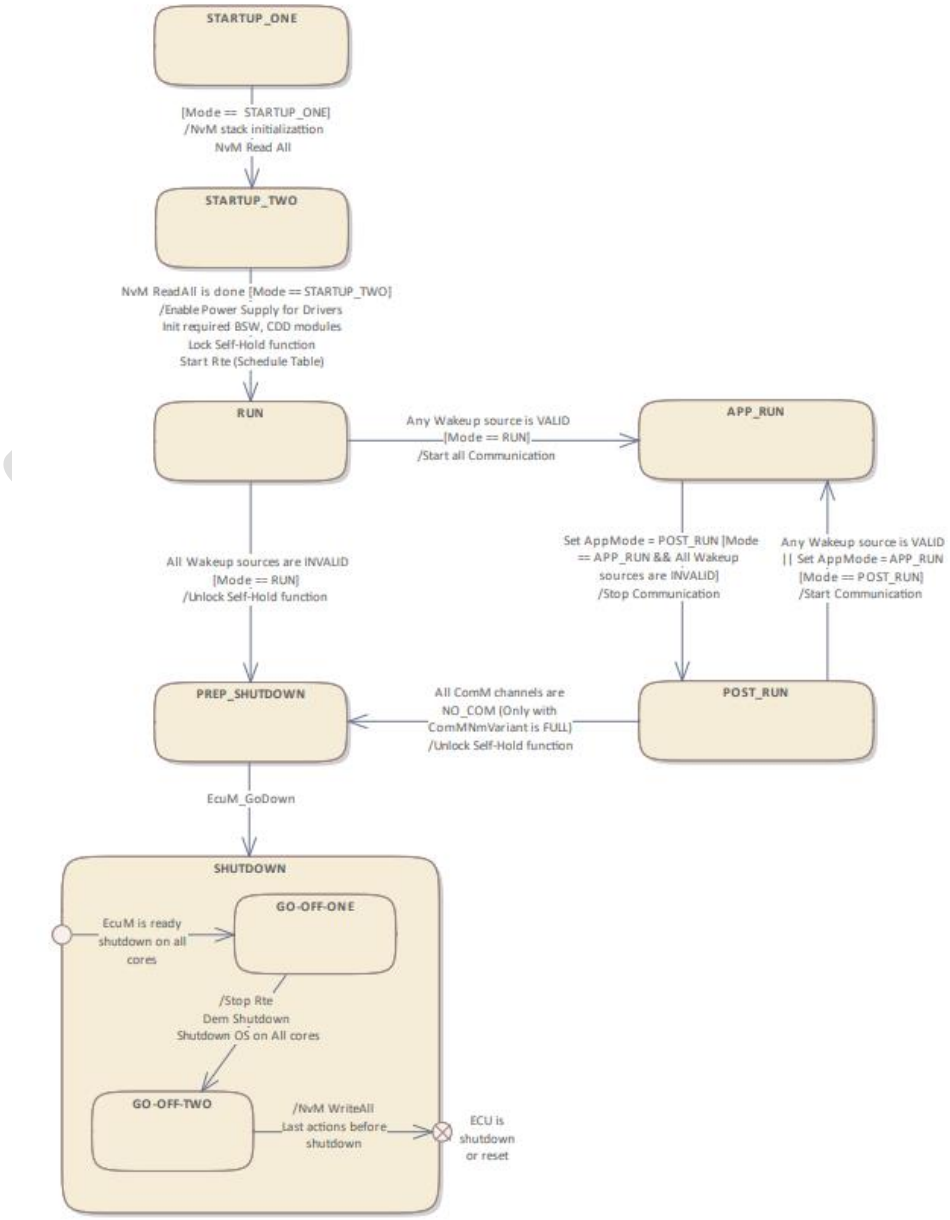
When user porting BIP to different target, user will need to

- Provide MCAL **MCU** parameter definition file (usually shipped within MCAL package) at `[ProjectRoot]\BasicSoftware\ecu_config\mcal\paramdefs`.
- Re-configure this Mcu specific **`McuResetReason`** in Mcu module configuration container `McuResetReasonConf` in ISOLAR-B ECU Navigator.
- Re-configure `EcuMWakeupSource` to reference corresponding `McuResetReasonConf`.

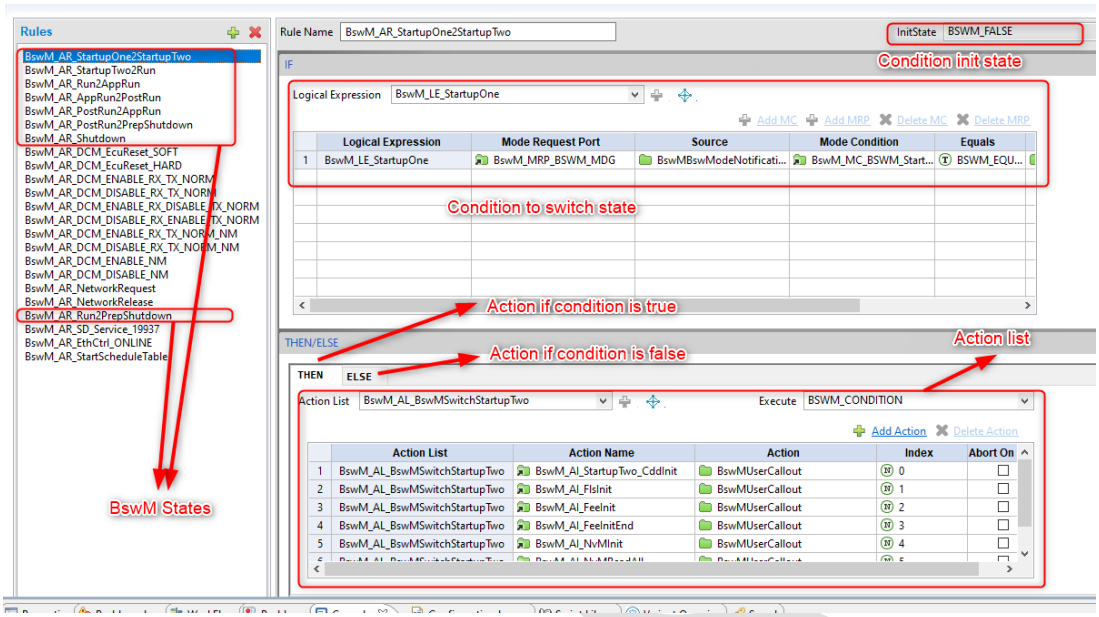
4.3.4 BswM Integration

State Machine

Software will work in 7 states from Start up to Shutdown. With each state, Software will perform certain actions. When actions are all performed, or some conditions are satisfied, Software will switch to another state.



The states can be configured in BSW:



Action list execution

BswM has two way to execute Action list when state condition is correct:

- **BSWM_CONDITION**: Action list shall be executed every time the rule is evaluated.
- **BSWM_TRIGGER**: Action list shall be executed every time the result of the evaluation changes

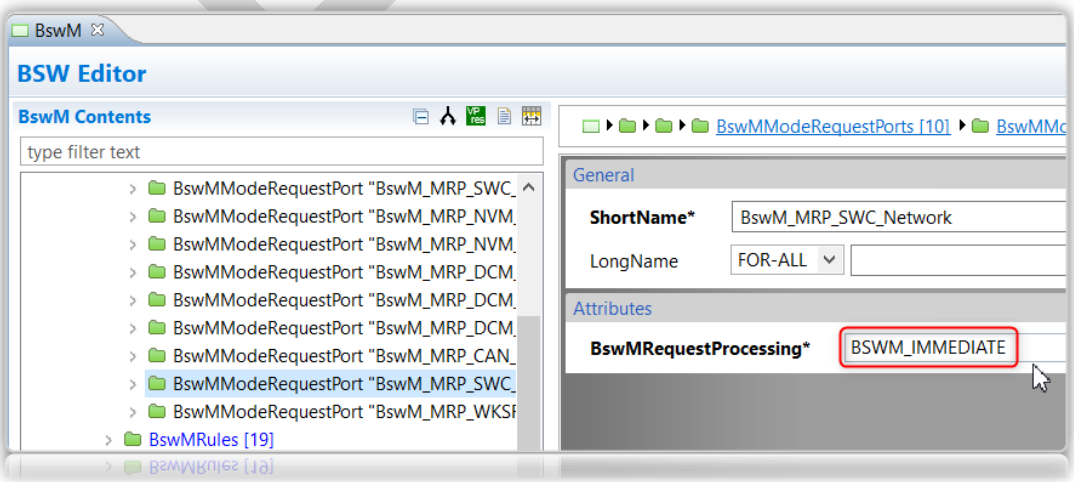
BswM Request Processing

BswM has two mechanisms to evaluate user requests:

- **Immediate**: User request will be immediately evaluated at the point of request.
- **Deferred**: Evaluation of user request in MainFunction which is asynchronized from request.

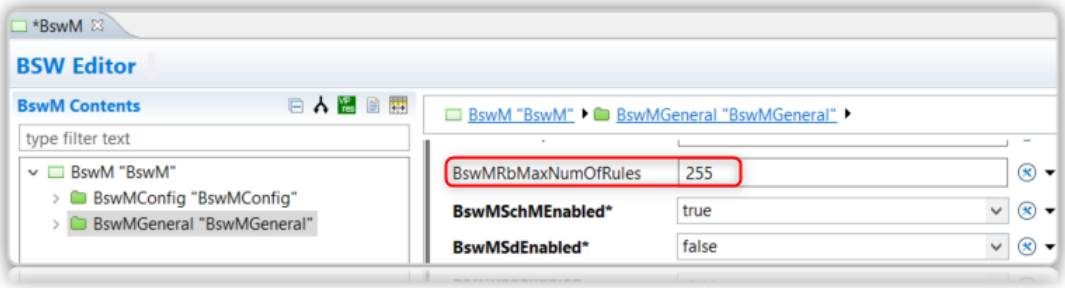
Immediate evaluation of User Request

[$BIP_INTG 028] BIP configure BswM to immediately evaluate consecutive user requests by set `BswMRequestProcessing` as **`BSWM_IMMEDIATE`** so that consecutive request will not be ignored.



Deferred evaluation of User Request

[$BIP_INTG 029] BIP configure BswM with **maximum length** of requested rule buffer so that as many of user requests comes a lot more frequently than BswM evaluation interval will not be lost by BswM and as a result the expected BswM action will not missed.



Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.3.5 Nv Stack Integration

[3.5 How to deploy System to EcuC] generated memory stack configuration for NvBlocks software component with a set of confiugraiton files in `[ProjectRoot]\BasicSoftware\ecu_config\bsw\gen`

- `ETAS_Project_NvM_EcucValues.arxml`
- `ETAS_Project_Fee_EcucValues.arxml`
- `ETAS_Project_MemIf_EcucValues.arxml`

NvM Integration

After [3.5 How to deploy System to EcuC], user shall review generated `ETAS_Project_NvM_EcucValues.arxml` and further adjust the default parameter values to fit for their needs.

[$BIP_INTG 030] At BIP configuration time, below common parameters adjusted to fit for use.

Configuration Parameter	
Change	NvM/NvMCommon/**NvMMainFunctionPeriod**
	NvM/NvMCommon/**NvMSizeStandardJobQueue**

Hints: 2 above configurations can be configured by setting in `[ProjectRoot]\BasicSoftware\ecu_config\bsw\settings\algo.properties`

```
manprop_NvMCommon=\

NvMMainFunctionPeriod:0.01,\

NvMSizeStandardJobQueue:255
```

[$BIP_INTG 031] At BIP configuration time, below **cyclic saving block** specific parameters adjusted to fit for use.

Configuration Parameter

Change	NvM/NvMBlockDescriptor/**NvMInitBlockCallback**
Add	NvM/NvMBlockDescriptor/**NvMNvBlockLength**
	NvM/NvMBlockDescriptor/**NvMSingleBlockCallback**

[$BIP_INTG 032] At BIP configuration time, below **immediate saving block** specific parameters adjusted to fit for use.

Configuration Parameter	
Change	NvM/NvMBlockDescriptor/**NvMInitBlockCallback**
Add	NvM/NvMBlockDescriptor/**NvMNvBlockLength**
	NvM/NvMBlockDescriptor/**NvMSingleBlockCallback**

[$BIP_INTG 033] At BIP configuration time, below **shutdown saving block** specific parameters adjusted to fit for use.

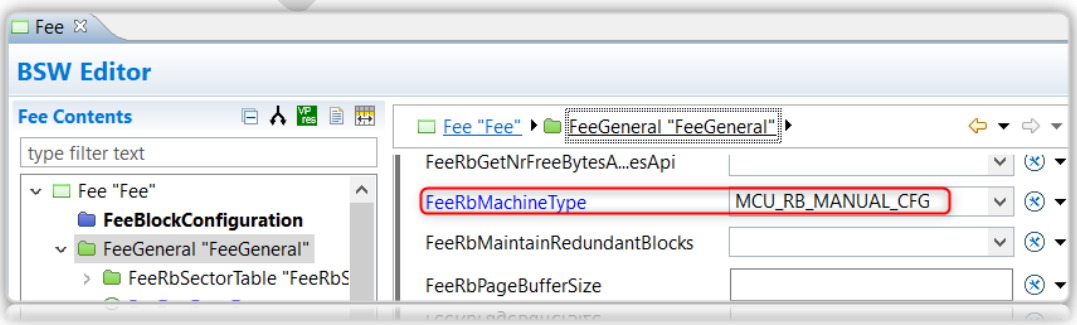
Configuration Parameter	
Change	NvM/NvMBlockDescriptor/**NvMInitBlockCallback**
Add	NvM/NvMBlockDescriptor/**NvMNvBlockLength**
	NvM/NvMBlockDescriptor/**NvMSingleBlockCallback**

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

Fee Integration

During [3.5 How to deploy System to EcuC], there is no way to know which target/MCAL shall be used for SWC required NvBlocks, therefore the generated hardware dependent options need to further update by user in ISOLAR-B **ECU Navigator**.

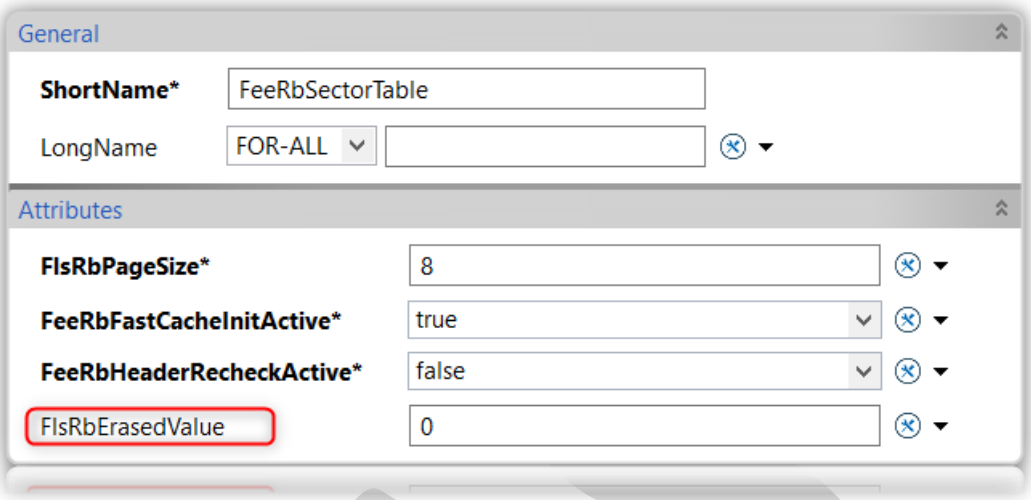
[$BIP_INTG 034] BIP integrates Fee as manual configuration to fit for integration with external MCAL provided Fls module. This is done by specify `Fee\FeeGeneral\FeeRbMachineType` to `MCU_RB_MANUAL_CFG`.



Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

Porting BIP to new Target

[$BIP_INTG 035] When user porting BIP to different target, user need to specify value of `ETAS_Fee_EcucValues.arxml\Fee\FeeGeneral\FeeRbSectorTable\FlsRbErasedValue` according to the Target Data FLASH erased value physically.



4.3.6 WdgM Integration

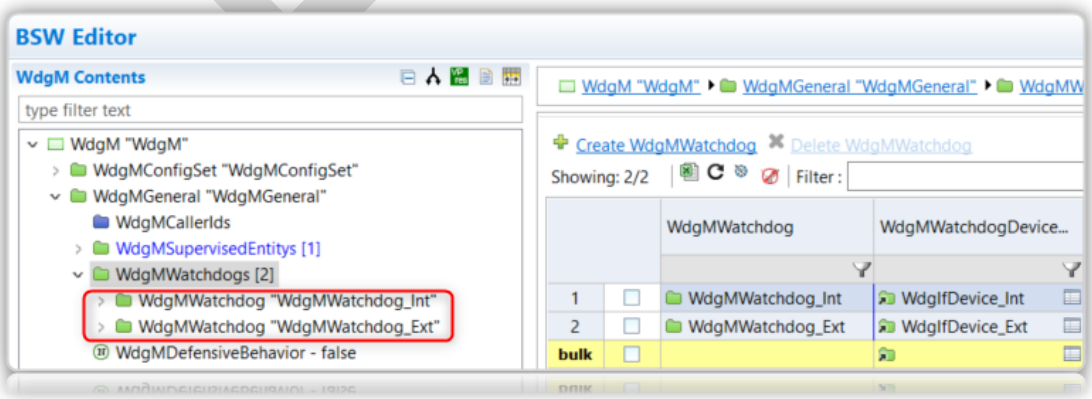
BIP configure WdgM to supervise execution of ECU level software component deployed from SysCluster of section [5 Application Integration].

Its supervision services are further describing in section [5.3.6 Monitoring Software Execution].

WdgM – Watchdog

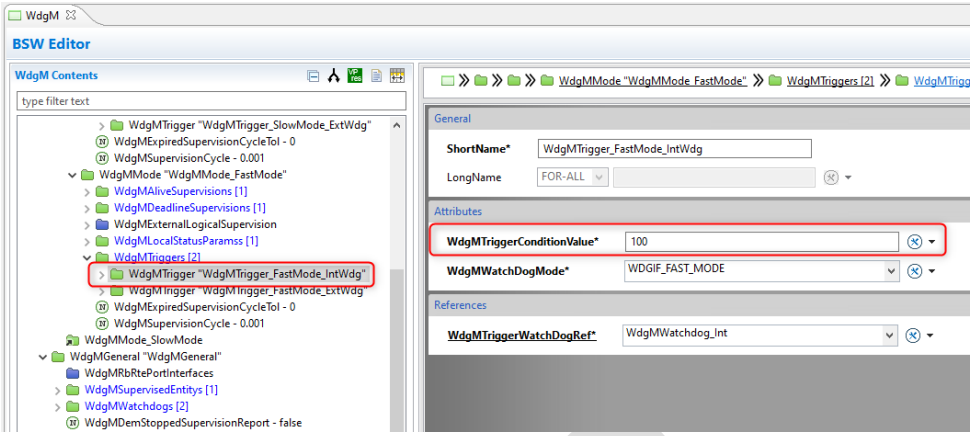
[$BIP_INTG 036] At configuration time, BIP configure WdgM with both internal watchdog and external watchdog as below.

> **NOTE**
> Further **software integration** required as described in section [6.2.3 Wdg - MCAL] for internal watchdog and section [6.3.7 External Watchdog] for external watchdog.



Info: User will need to configure Wdg in MCAL configuration tool with reference to ETAS BIP.

[$BIP_INTG 133] Trigger Condition Value (Timeout for feeding the watchdog) should be configured for each watchdog with corresponding mode (fast mode or slow mode).



NOTE

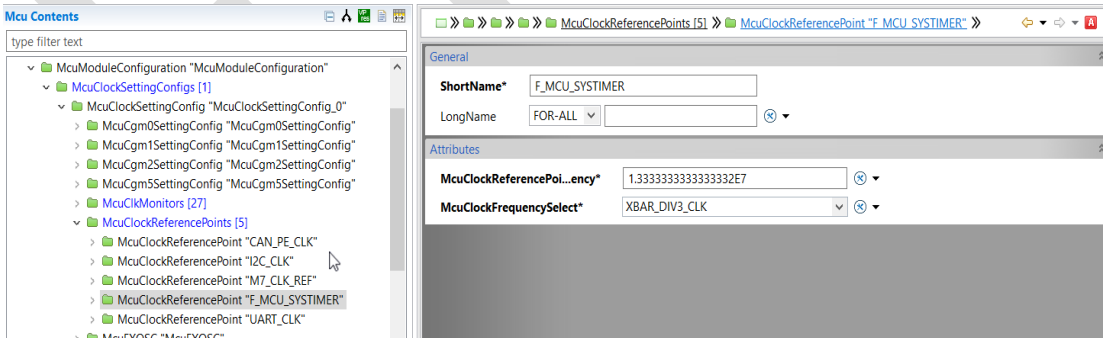
[$BIP_INTG 134] `WdgMTriggerConditionValue` should bigger than "Wdg Timeout Period" which is configured in Wdg-MCAL described for internal watchdog.

WdgM – MCU

[$BIP_INTG 037] BIP configure WdgM to use a **MCU hardware counter** for deadline supervision to provide more supervision precision of supervised entities.

[$BIP_INTG 038] Therefore, BIP configure the MCU hardware counter by `McuClockReferencePoint` configuration `F_MCU_SYSTIMER` integrated as

- A MCAL level configuration `ETAS_Mcu4WdgM_EcucValues.arxml` at `[ProjectRoot]\BasicSoftware\ecu_config\mcal\ecucValues`.
- A MCAL MCU parameter definition file at `[ProjectRoot]\BasicSoftware\ecu_config\mcal\paramdefs`.



Porting BIP to new Target

When user porting BIP to different target, user will need to

- Provide MCAL **MCU** parameter definition file (usually shipped within MCAL package) at `[ProjectRoot]\BasicSoftware\ecu_config\mcal\paramdefs`.
- Configure an intended counter tick frequency in Hz to `F_MCU_SYSTIMER` in configuration file at `[ProjectRoot]\BasicSoftware\ecu_config\mcal\ecucValues`.

NOTE

[$BIP_INTG 039] User shall pay attention to provide the counter ticking at the **same frequency** as configured value of `McuClockReferencePointFrequency` at software integration phase at section [6.3.5 Target Clock], since [4.3.12 Ecu Partitioning with OS Integration] will generate WdgM deadline ticks `DeadlineMax` according to `McuClockReferencePointFrequency`.

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.3.7 Dem Integration

Dem module is mostly automatically configured by deploying AUTOSAR DEXT to EcuC following [3.5 How to deploy System to EcuC].

[$BIP_INTG 040] However Dem has a few configurations information that are not available in AUTOSAR DEXT have been manually configured in `ETAS_Project_Dem_EcucValues.arxml`

Configuration Parameter	
Change	`Dem/DemGeneral/DemAvailabilitySupport`
	`Dem/DemGeneral/DemTaskTime`
	`Dem/DemGeneral/DemDevErrorDetect`
	ExtendedDataRecord: Change `Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/`**`DemDataElementEndiannes`** to `Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/`**`DemDataElementDataType`**
Add	`Dem/DemGeneral/DemPrimaryMemory/DemMaxNumberEventEntryPrimary`
Delete	`Dem/DemGeneral/DemDataElementClass/DemExternalSRDataElementClass/DemDataElementDataSize`

Dem - MCAL Integration

[$BIP_INTG 041] BIP has been configured with `ETAS_Dem_`*`[McalModule]`*`_EcucValues.arxml` for Dem event reported by MCAL modules through `Dem_ReportErrorStatus()`.

Modifying BIP

[$BIP_INTG 042] When modifying BIP to project needs by configuring other MCAL modules that are not in BIP integration, user shall add those module specific Dem events to BSW static configuration.

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

Porting BIP to new Target

[$BIP_INTG 043] Also, different MCAL may report **vendor specific** Dem event for MCAL modules to Dem. When user porting BIP to different target/MCAL, user may need to review

MCAL user manuals to find out which Dem errors are offered by MCAL then add/remove/modify each MCAL modules `ETAS_Dem_[McalModule]_EcucValues.arxml`

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

4.3.8 Dcm Integration

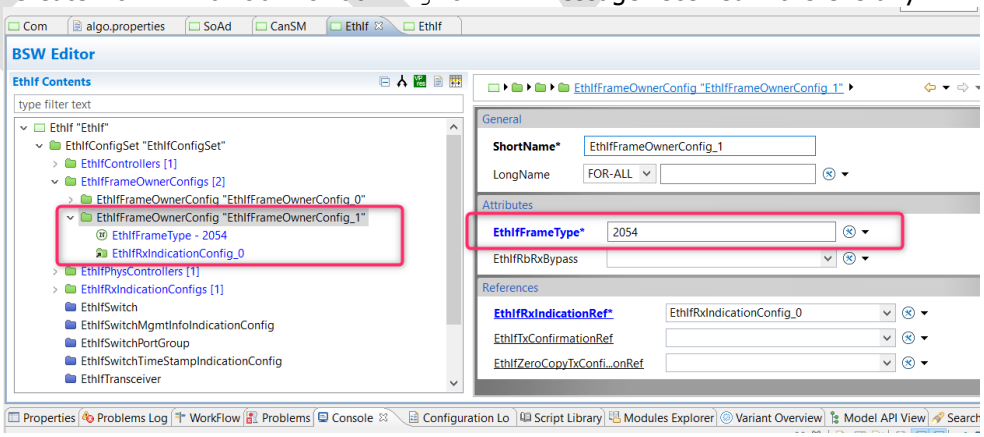
[$BIP_INTG 044] Dcm module is mostly automatically configured by deploying AUTOSAR DEXT to EcuC following [3.5 How to deploy System to EcuC]. However, Dcm has a few configurations information that are not available in AUTOSAR DEXT have been manually configured in `ETAS_Project_Dcm_EcucValues.arxml`

Configuration Parameter	
Change	`Dcm/DcmGeneral/DcmVersionInfoApi`
	`Dcm/DcmGeneral/DcmDevErrorDetect`
	`Dcm/DcmGeneral/DcmRbGeneral/DcmRbRTEsupportsPRportFunctionality`
	`Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidControl/DcmDspDidControlMask`
Add	`Dcm/DcmConfigSet/DcmDsp/DcmDspDidInfo/DcmDspDidControl/DcmDspDidControlMaskSize`

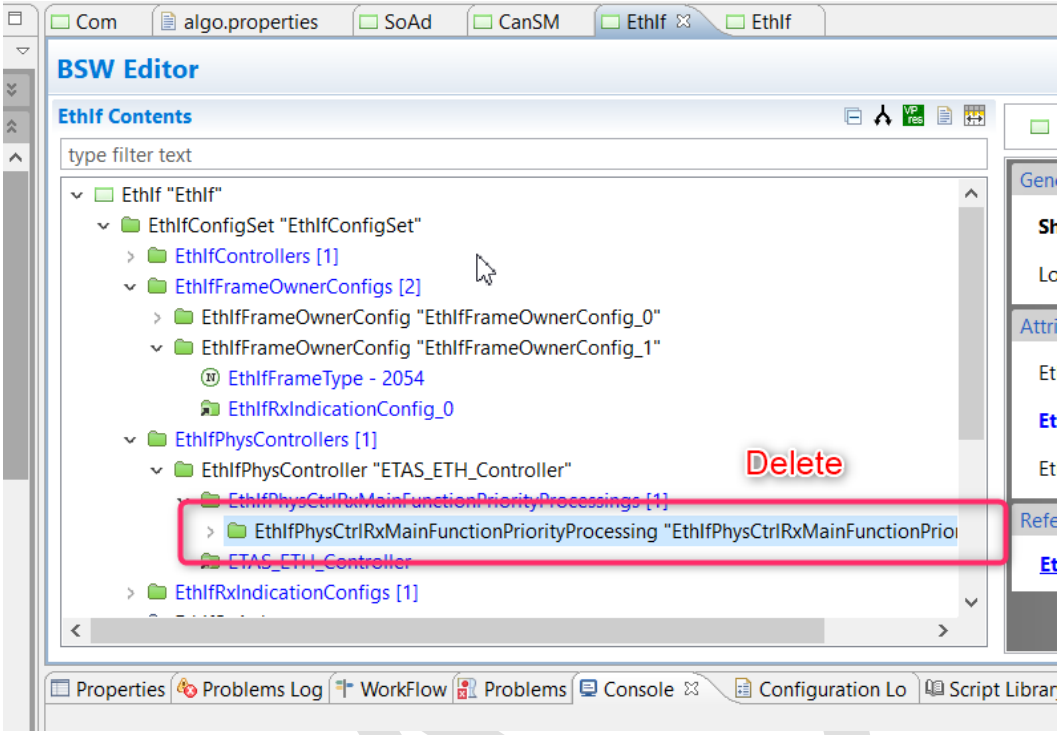
4.3.9 Eth Integration

Almost Eth stack (EthIf, SoAd, Tcp/IP) can be generated by Configuration Generation. But there are some items need to be added and modified.

1) Create `EthIfFrameOwnerConfig` for ARP message received if there is any.



2) Remove `EthIfPhysCtrlRxMainFunction` if ETH is in interrupt mode

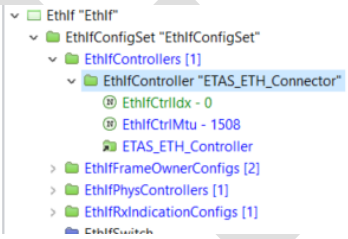


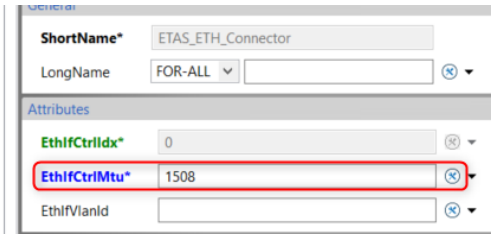
3) Remove EthTrcv configuration (`ETAS_Project_EthTrcv_EcucValues.arxml`) if Hardware isn't required EthTrcv module.

4) `EthIfCtrlMtu` is the maximum transmission unit (MTU).

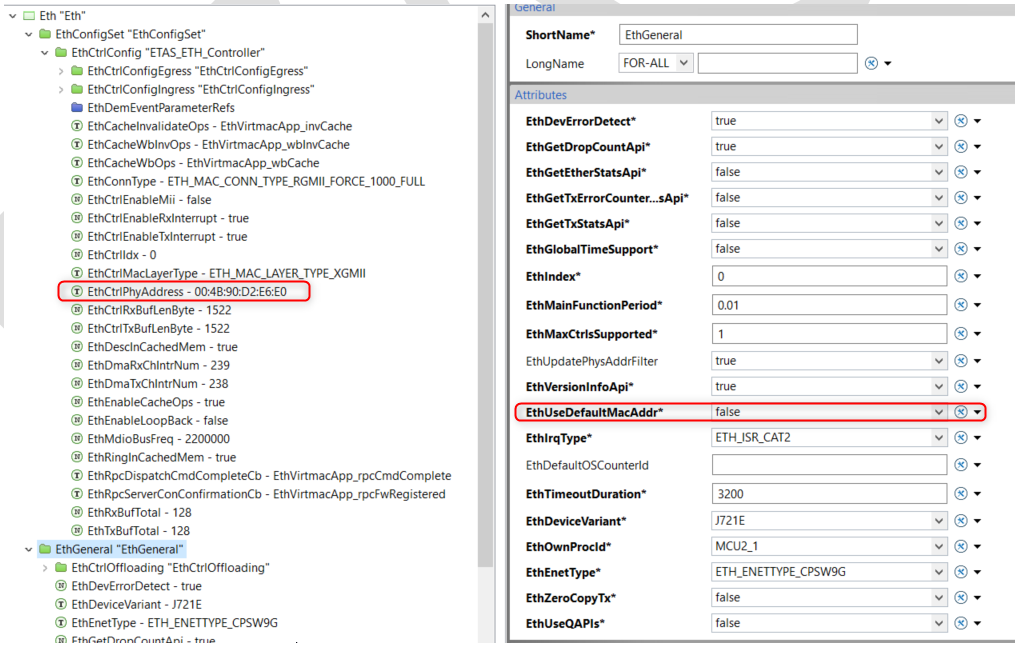
`EthCtrlRx/TxBufLenByte` = [MAC destination 6Bytes] + [MAC source 6Bytes] + [EtherType/length 2Bytes] +MTU = **14 + MTU**.

TCP MSS (Max Segment Size) generated in source code = MTU – [IP Header 20Bytes] – [TCP Header 20 Bytes] = **MTU - 40**.



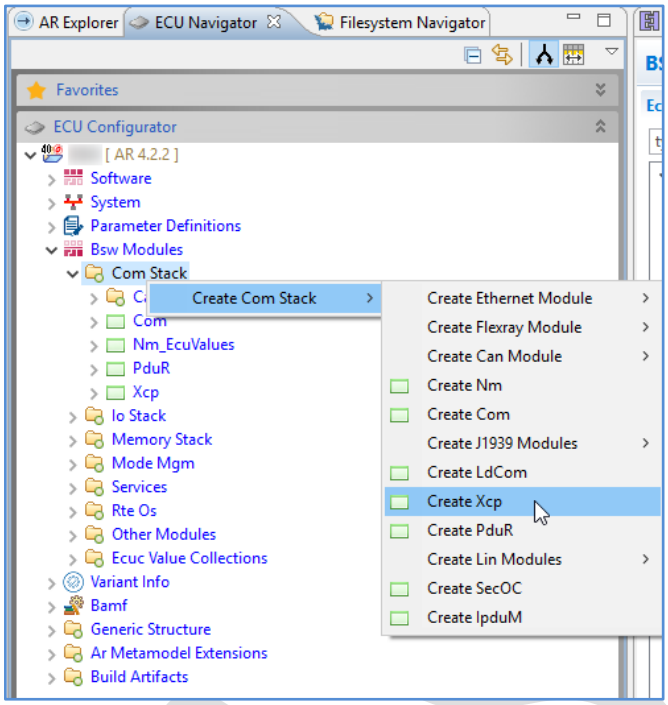


5) To enable the configured `EthCtrlPhyAddress`, `EthUseDefaultMacAddr` shall be false.



4.3.10 Xcp Integration

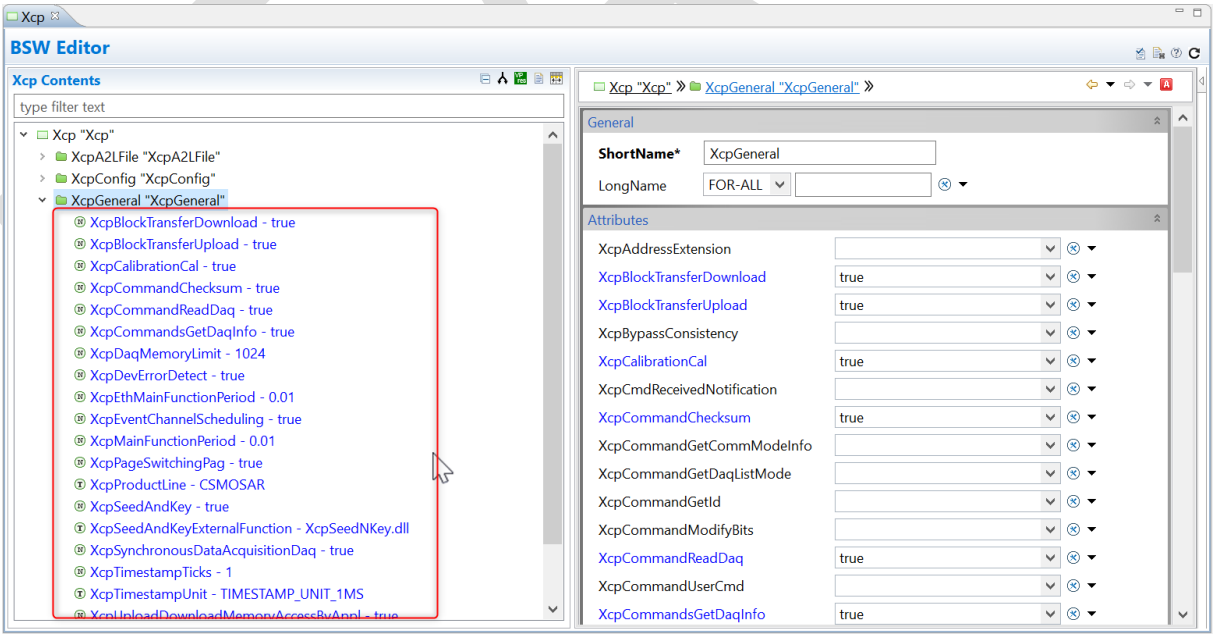
In BSW configuration, XCP can be created and some of configurations are automatically created.



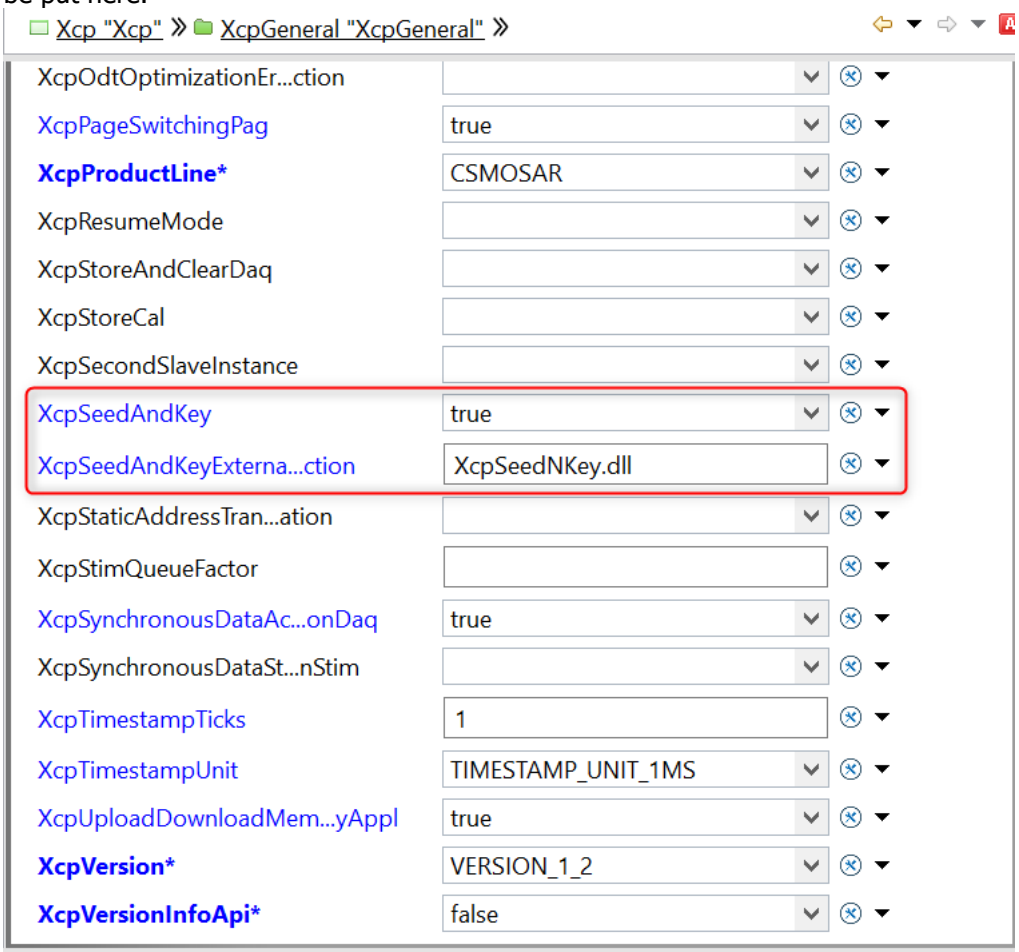
Depend on the requirements, user can customize the configuration to meet the requirements.

XcpGeneral container

XcpGeneral container provides the configuration for list of commands, main function period.



For security algorithm, seed&key function should be set to "true" and name of dll file shall be put here.

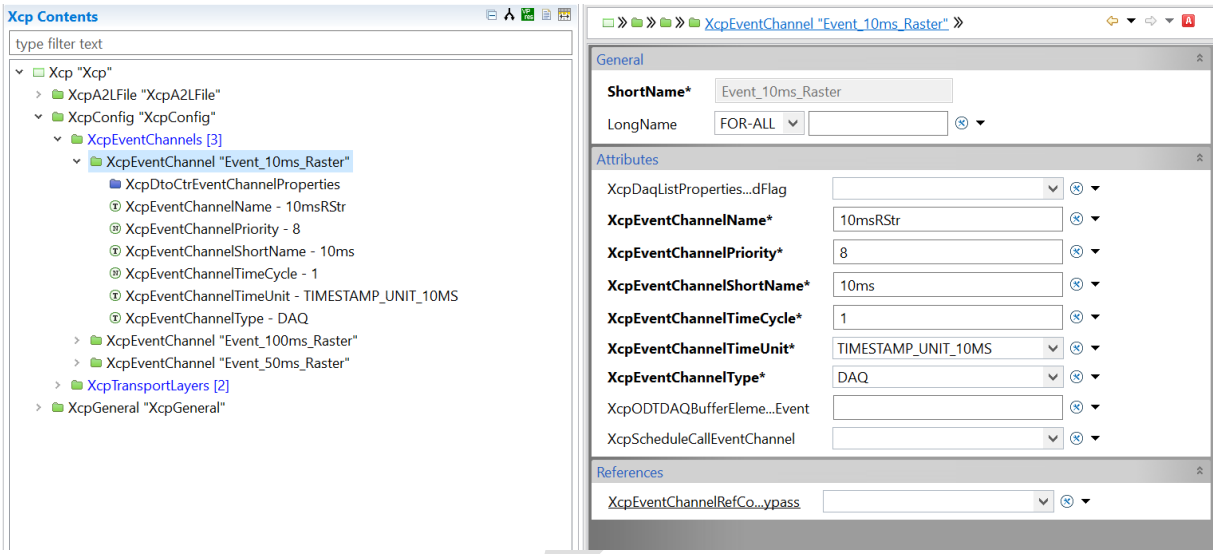


XcpConfig container

XcpConfig container contains the configuration for event channel and transport layer

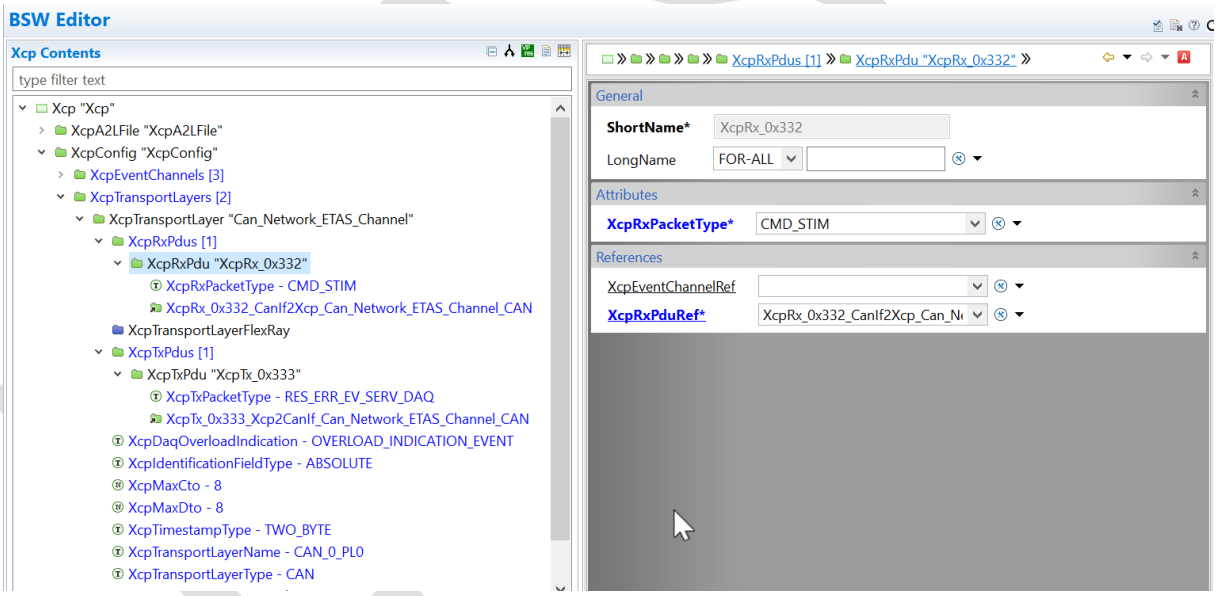
In this section, the user will configure the event channel (Raster) for DAQ:

- XCPEventChannelPriority: The priority of the event
- XCPEventChannelTimeCycle: The number of cycles of time unit
- XCPEventChannelTimeUnit: The unit of the event
- XCPEventChannelType: Type of DAQ list – DAQ or STIM



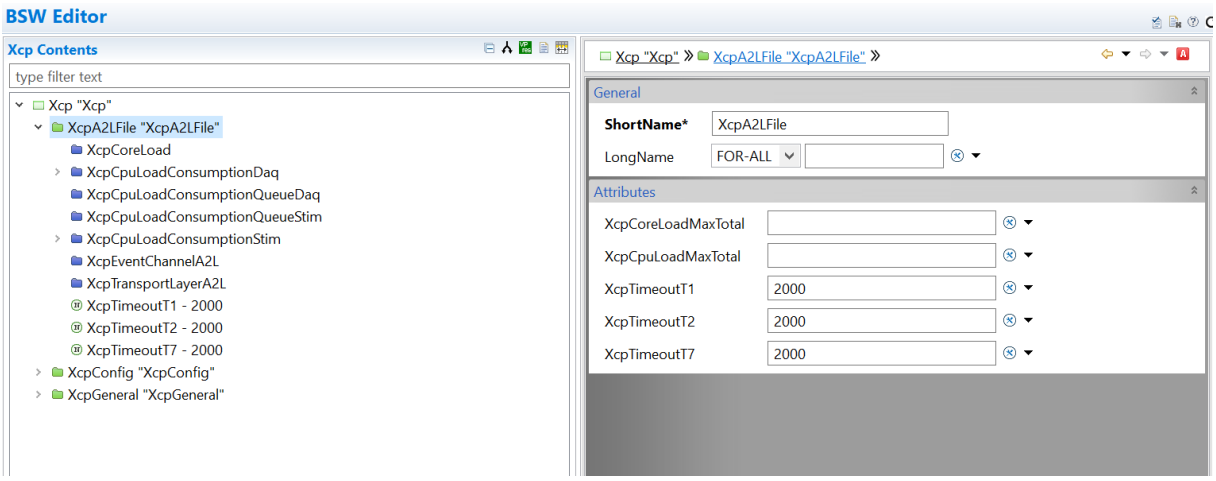
In transport layer configuration, PDUs in XCP should be created and mapped with corresponding PDUs in CanIf. For each one, type of packet can be CMD – command, RES – response, DAQ, STIM – stimulation…

For RxPdu, the packet type shall be either CMD or STIM.



XcpA2LFile

XcpA2LFile is the place where the user configures the timeouts of the connection between Master and Slave.



4.3.11 Rte Integration

BIP has preconfigured BSW MainFunction periods and assign their periodically scheduled events to OS Tasks according to their periods.

[$BIP_INTG 045] In case of BSW module MainFunction period changed, user shall re-map that Task to a proper OsTasks according to the configured period.

Rte – Can – Dcm

BIP Can communication can be configured as either polling mode or interrupt mode.

When configure as polling mode: `Can_MainFunction_Write()` and `Can_MainFunction_Read()` will need to be periodically scheduled at MCAL configured polling period.

[$BIP_INTG 046] Position in Task for BSW scheduled entity `Can_MainFunction_Write()` shall be assigned before `Can_MainFunction_Read()` due to the reason that for Transport layer protocols (`CanTp`) when it received FirstFrame, it needs to response with FlowControl before it can receive ConsecutiveFrame.

When configure as interrupt mode: user shall pay special attention of interrupt vector of CAN reception and CAN transmission interrupts sources. And if they have different interrupt vector, make sure CAN transmission interrupt shall not have lower priority than CAN reception interrupt.

Rte - MCAL

Following AUTOSAR layered architecture, Rte has nothing to do with MCAL regarding interfaces. However, MCAL modules who have MainFunction to be scheduled by OS at runtime need to provide configuration time integration with Rte.

[$BIP_INTG 047] BIP integrate this by putting MCAL BSWMD and SWCD configuration files into BIP `[ProjectRoot] BasicSoftware\integration\mcal\ISOLAR\bswmd`.

Modifying BIP

BIP integrates only the necessary MCAL CAN module's **BSWMD** and **SWCD**.

[$BIP_INTG 048] When modifying BIP to project needs by integrating more MCAL modules BSWMD and SWCD, user will need to

- Put user MCAL module's BSWMD and SWCD (delivered within MCAL package) in `[ProjectRoot] BasicSoftware\integration\mcal\ISOLAR\bswmd`.
- Assign MCAL module's MainFunction to OS tasks in Rte Task Mapping.
- Generate Rte

Porting BIP to new Target

[$BIP_INTG 049] When porting BIP to new target/MCAL, user shall provide the necessary MCAL module's **BSWMD** and **SWCD** either from MCAL delivery package or by manually write according to AUTOSAR standards in case MCAL does not provide them.

NOTE

About which MCAL modules **BSWMD** and **SWCD** are necessary for Rte Integration: Those modules whose **Bsw Schedulable Entity** require scheduling in Tasks.

4.3.12 Ecu Partitioning with OS Integration

ETAS BIP provides three partitions to deploy compositions of [5 Application Integration] to run on either single core or Multicore target.

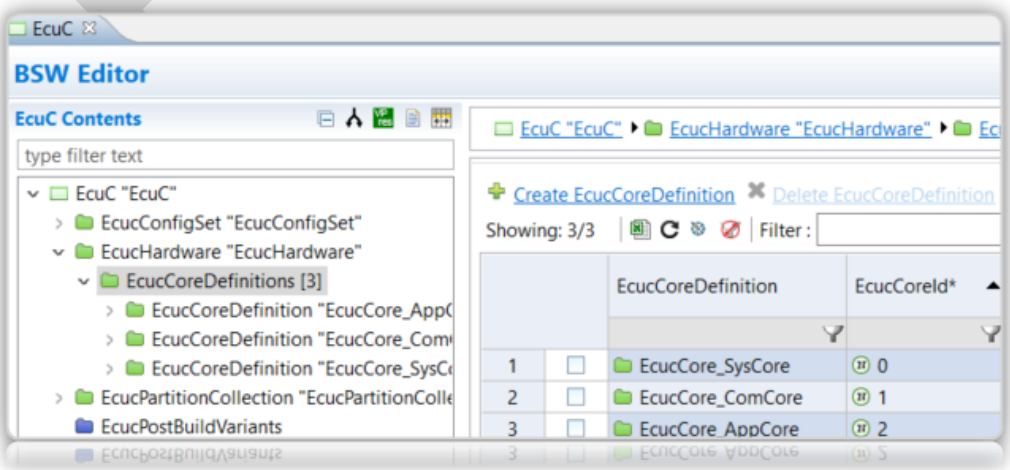
EcuC - OS

1) Specifying AUTOSAR Cores

[$BIP_INTG 050] BIP configures EcuC with a set of `EcucCoreDefinition` for cores that will run AUTOSAR based software.

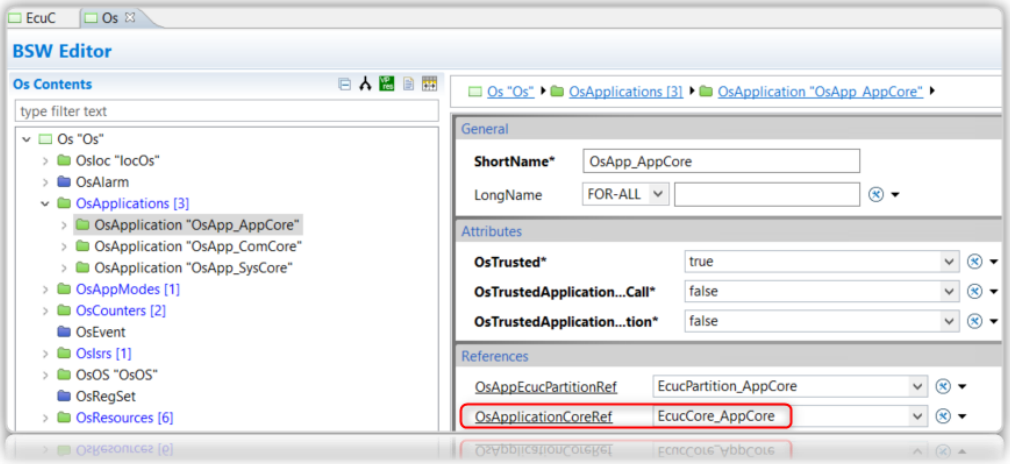
NOTE

There could be other cores on the target which are not necessarily visible in EcuC configuration as they are not required for AUTOSAR scheduling (**non-AUTOSAR Core**). In this case these cores are not necessarily to be configured in `EcucCoreDefinition`.



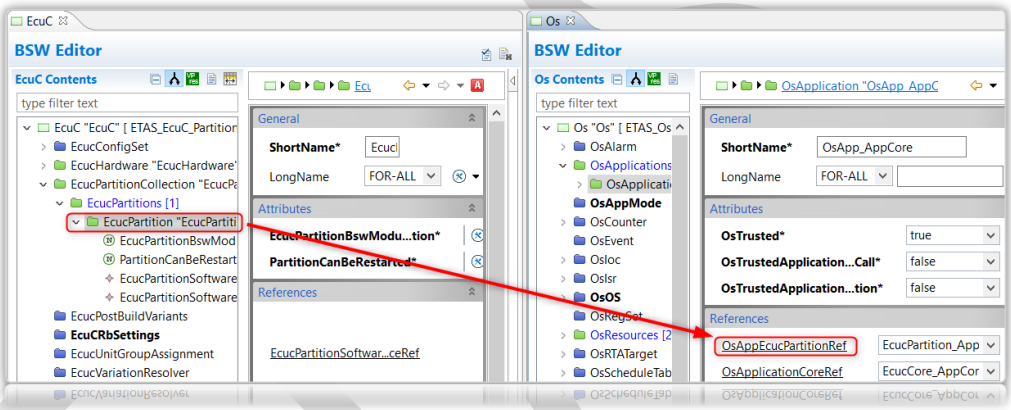
2) Assigning AUTOSAR Cores for Scheduling

[$BIP_INTG 051] BIP assigns those AUTOSAR cores for scheduling by OS with `OsApplicationCoreRef` of `OsApplication`.



3) Scheduling Software by Cores

[$BIP_INTG 052] BIP specifies software components to be schedule by cores by mapping `EcucPartition` to `OsApplication`.

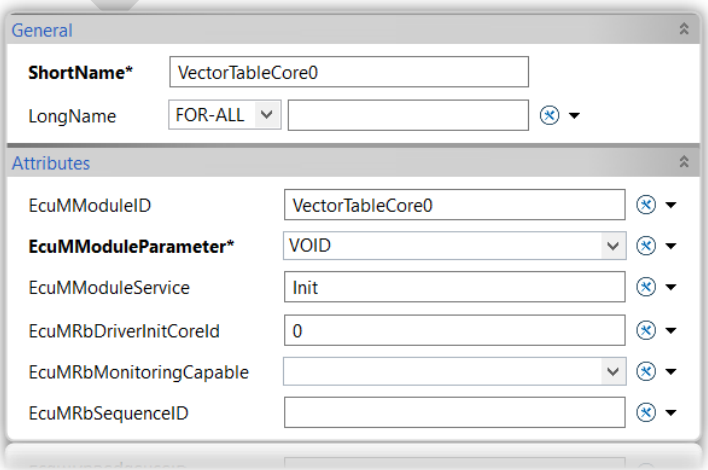


Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

EcuM - OS

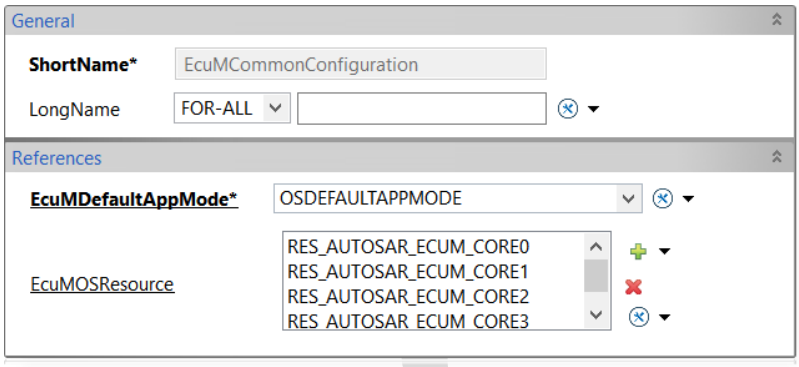
1) EcuM Multicore Startup

[$BIP_INTG 053] Startup of AUTOSAR OS requires core vector table to be initialized before StartOS. BIP configure initialization of core vector table in EcuM Startup-I phase.



2) EcuM Multicore Sleep

[$BIP_INTG 054] To support EcuM coordinating Multicore Sleep (no matter EcuM sleep is required or not), it need to be able to access OS Resources of each AUTOSAR Core specified in section [4.3.12 EcuC - OS]. BIP references these `OSResources` to `EcuMOSResources` for each AUTOSAR core.



Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

OS Target

[$BIP_INTG 055] ETAS BIP configures target specific parameters for OS in `[ProjectRoot]\BasicSoftware\ecu_config\os\static\ETAS_Os_Target_EcucValues.arxml`

Configuration Item	Description
`OsRTATarget\OsRTATargetName`	Name of RTA-OS target port installed from RTA-CAR.
`OsRTATarget\OsRTATargetVersion`	Version of RTA-OS target port.
`OsRTATarget\OsRTATargetVariant`	Target variant to be used by user.
`OsIsr\OsIsrCategory`	The category of ISR
`OsIsr\OsIsrAddress`	The interrupt vector address of the ISR
`OsIsr\OsIsrPriority`	The priority of the configured ISR

Hint: Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

Modifying BIP

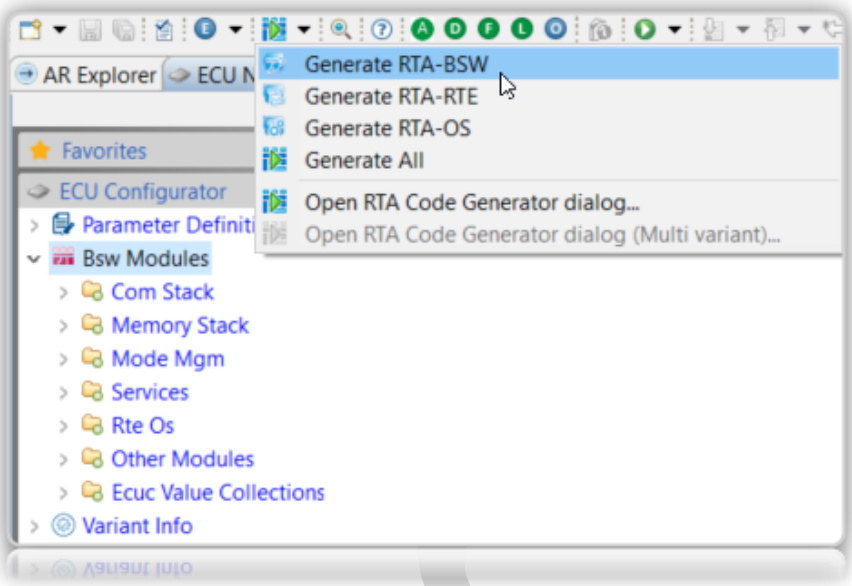
When user modify BIP by changing number of AUTOSAR cores, he/she shall follow the whole section [4.3.12 Ecu Partitioning with OS Integration].

Porting BIP to new Target

When porting BIP to new target which is already supported with RTA-OS port shipped with RTA-CAR, user shall follow section [OS Target] to adapt the target specific parameters.

4.4 How to generate BSW Code

After BIP arxml integration and static configuration, user could generate BSW code by launch **Generate RTA-BSW**



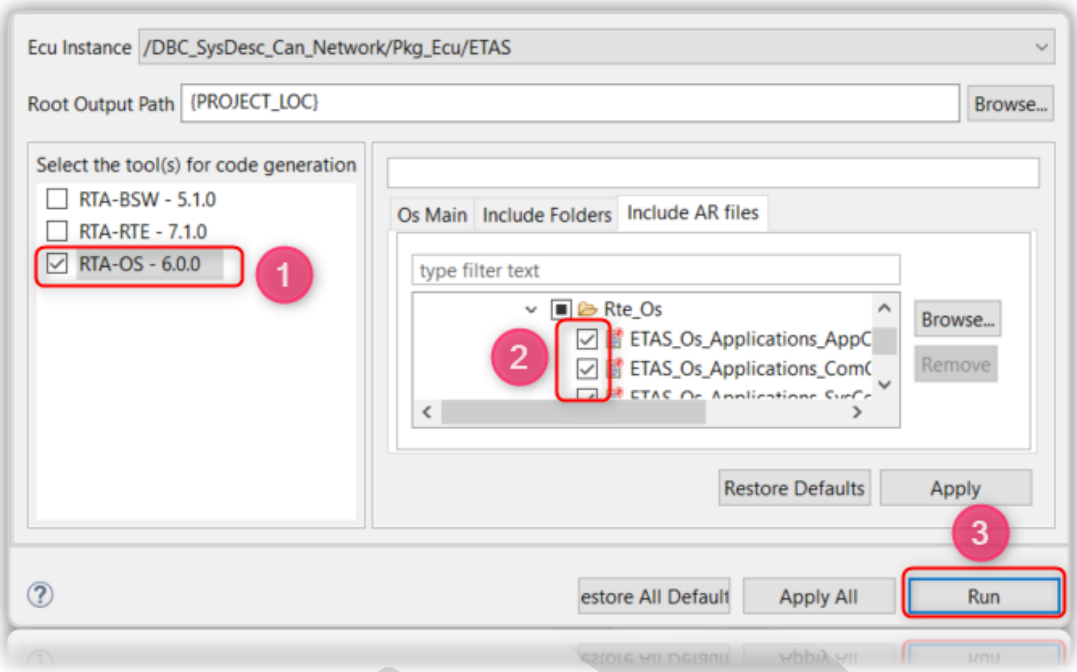
As a result of BSW code generation, user will get below outputs which are used for [5 Application Integration] and [6 Software Integration].

Generated Item	Description
`[ProjectRoot]\BasicSoftware\src\bsw`	BSW static code and configuration code to be used for [6 Software Integration]
`[ProjectRoot]\BasicSoftware\ecu_config\bsw\gen\bswmd`	BSW module description provides service entries required by SWCD. -> used for [5 Application Integration]
`[ProjectRoot]\BasicSoftware\ecu_config\bsw\gen\swcd`	BSW service SW-component describing which services provided. -> used for [5 Application Integration]

Table 4 BSW Generation Artefacts

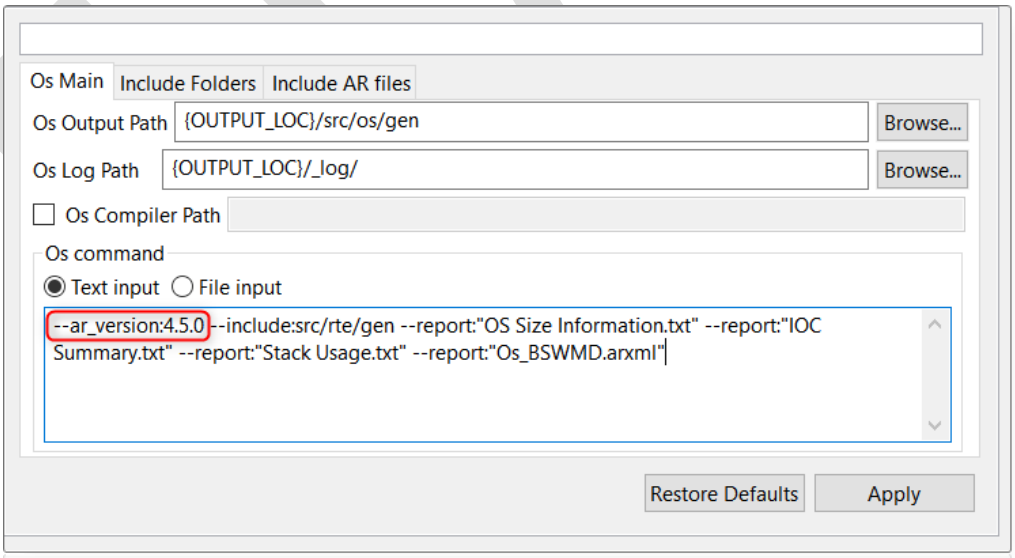
NOTE

If user add OS configuration arxml files as section [How to add EcuC Configuration to BIP], OS generation need to include those as input. To do so user shall [**Open RTA Code Generator dialog...**] and select those arxml files as [**Include AR files**] to run generation.



NOTE

[$BIP_INTG 056] In many cases, user would expect to generate OS of same AUTOSAR version as BSW (e.g., AR 4.2.2 for RTA-CAR 9) **explicitly** to use OS features of AR 4.2.2 if this not generated by default. This could be done by specifying AR version in **Os command box** of RTA-OS Code generator regardless of any version present in the ARXML configuration. e.g. `--ar_version:4.5.0`



4.5 Post-update BSW by Cobra_After_BSWGen

After BSW generation, for a specific RTA-CAR version user should run **Cobra_After_BswGen** to post update some generated files.

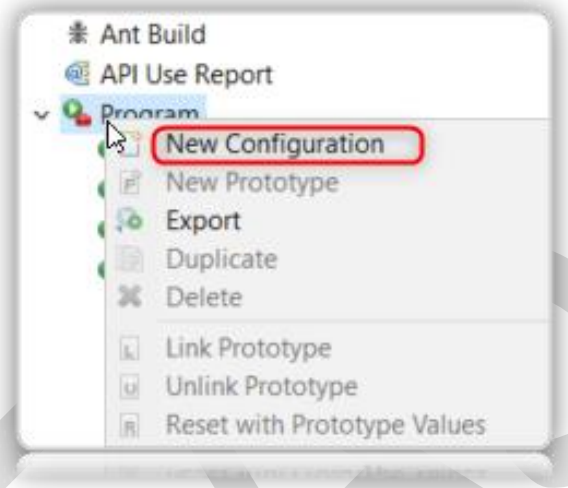
NOTE

Cobra_After_ConfGen is a RTA-CAR version dependent configuration add-on, it is not compatible on another RTA-CAR version.

If you need to support another RTA-CAR version, please contact ETAS at [Contacting ETAS]

4.5.1 Install and Run Cobra_After_BswGen

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup **Cobra_After_BswGen** tool environment and run it by click on "**Run**".

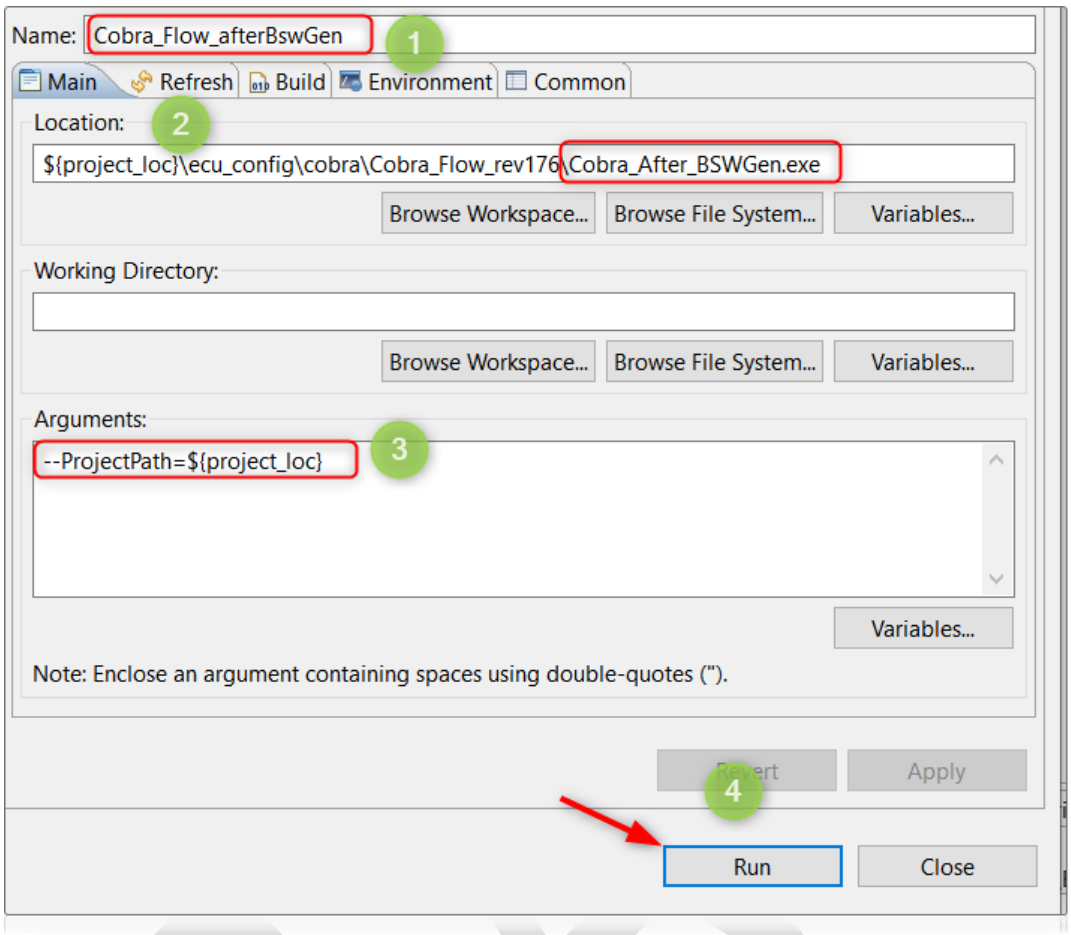


Figure 17 Install Cobra after BswGen add-on

Cobra afterBswGen is more or less a headless add-on, as a result of running it, the console prints nothing interesting information.

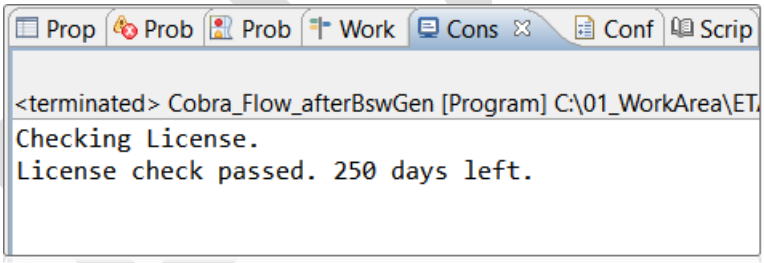


Figure 18 Cobra afterBswGen Generation

4.5.2 Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

4.6 How to generate MCAL

BIP has integrated MCAL configuration and generation with ISOLAR-AB with support of Cobra-MCAL Add-on.

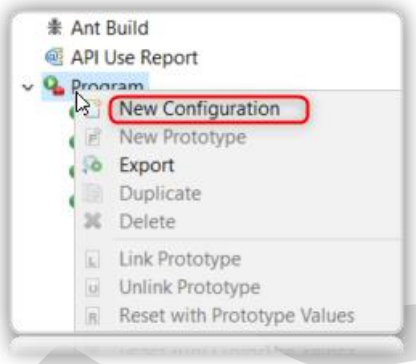
User could be able to configure MCAL modules in a similiar way of BSW EcuC modules described in above sections.

To generate MCAL modules, user could follow below sections.

- McalImporter: [$BIP_INTG 022] Import MCAL configuration into MCAL configuration tool Tresos.
- McalGen: [$BIP_INTG 023] Generation of MCAL configuration code.

4.6.1 Install and Run Cobra Mcal_Importer

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_McalImporter tool environment and run it by click on "**Run**".

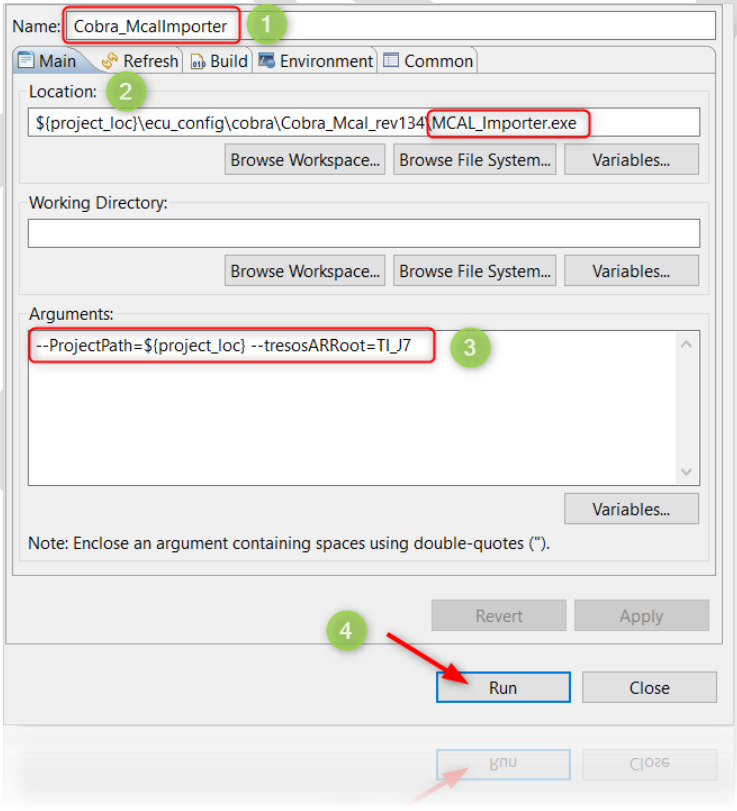
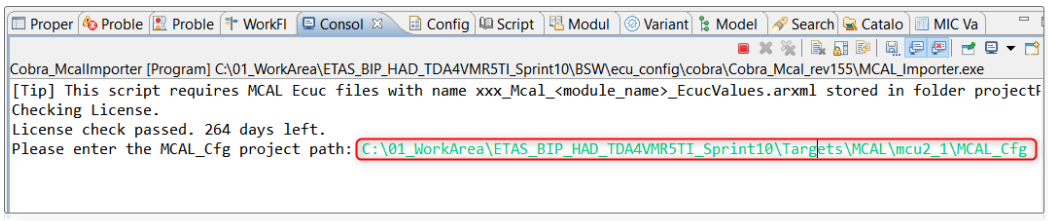


Figure 19 Install Cobra McalImporter add-on

By click run, the console prints information asking you to specify your MCAL project path. Input your path and confirm by click keypad 'Enter'.



As a result of run Cobra_McalImporter add-on, the console prints success information that the set of MCAL configuration file has been exported to MCAL configuration tool Tresos.

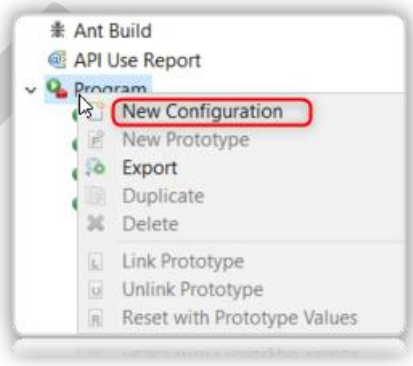
Figure 20 Cobra MCAL Imported

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

4.6.2 Install and Run Cobra Mcal_Gen

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_McalGen tool environment and run it by click on "**Run**".

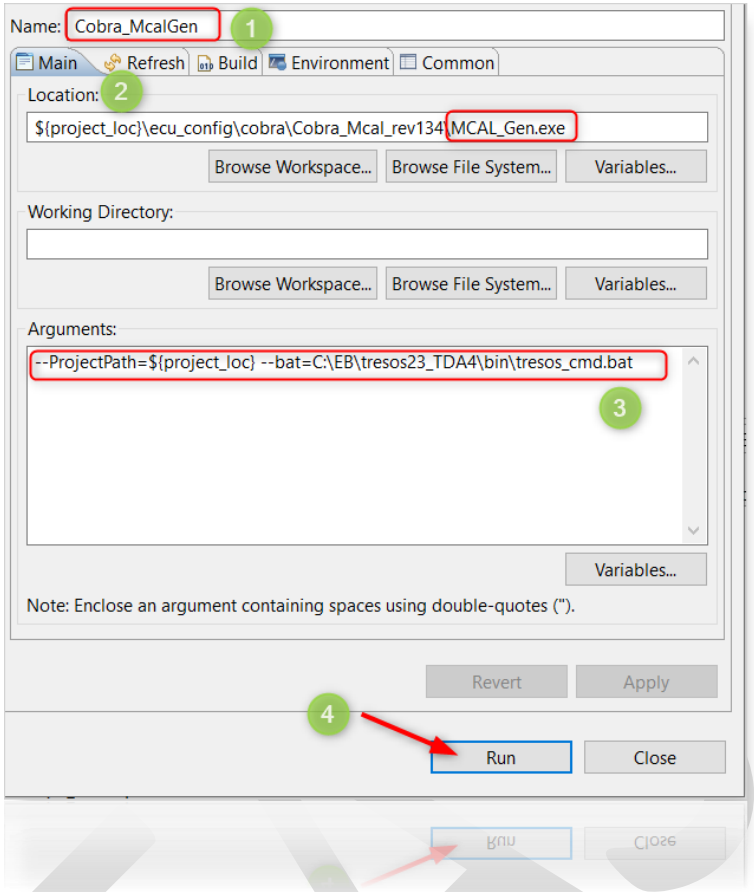
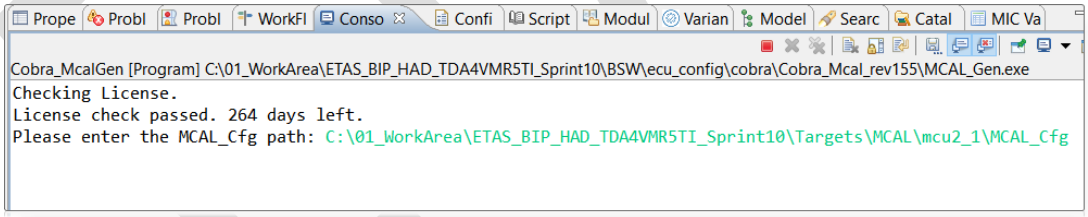
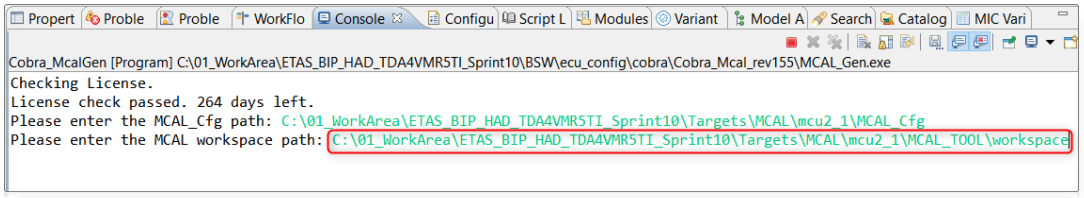


Figure 21 Install Cobra McalGen add-on

By click run, the console prints information asking you to specify your MCAL project path. Input your path and confirm by click keypad 'Enter'.



Then you also need to specify MCAL workspace path. Input your path and confirm by click keypad 'Enter'.



As a result of run Cobra_McalGen add-on, the console prints success information with a set of MCAL configuration source code generated under project path `[ProjectRoot]\BasicSoftware\integration\mcal\src\gen`.

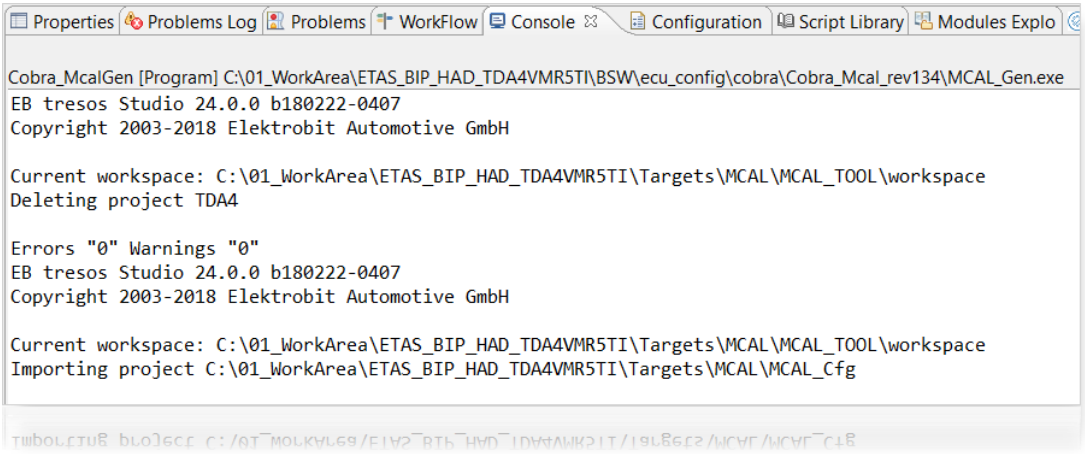


Figure 22 Cobra MCAL Generation

Outputs are shown in figure below.

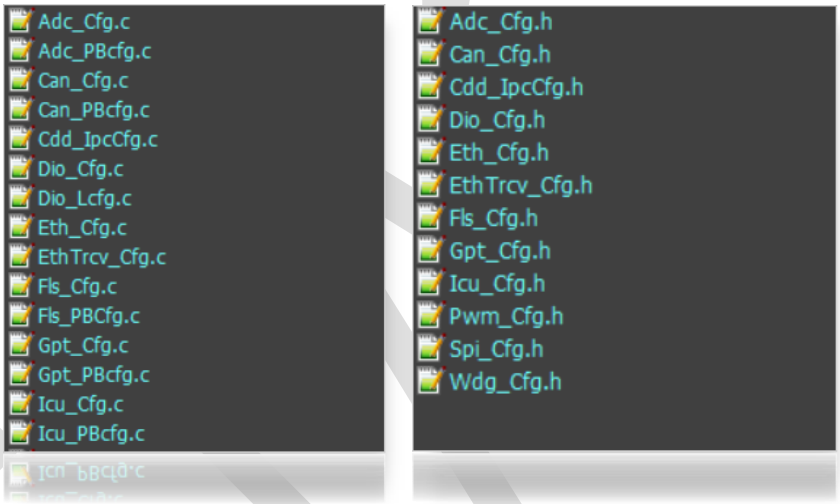


Figure 23 Output of Cobra MCAL Add-On

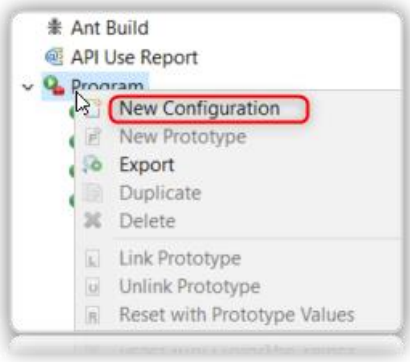
Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

4.7 Install and Run Cobra_Extension

BIP has been provided with Cobra to generate code for Rtm, StackM to assist continuous configuration and integration.

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.1`



Then follow step 1 – 4 to setup Cobra_Extension tool environment and run it by click on "**Run**".

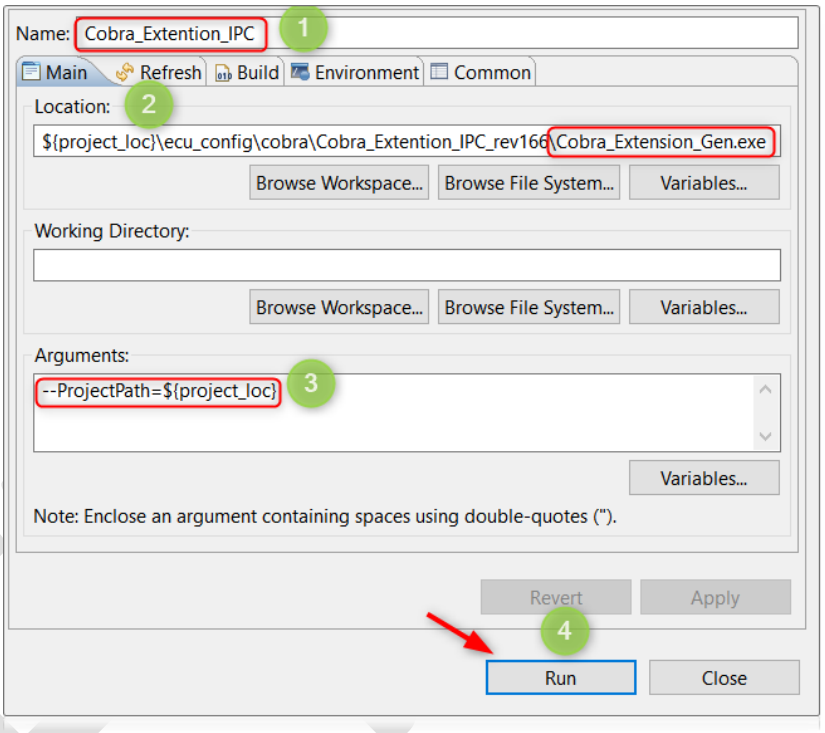


Figure 24 Install Cobra Extension add-on

As a result of run Cobra_Extension add-on, the console prints success information with a set of extension modules configuration source code generated under project path `[ProjectRoot]\BasicSoftware\src\cobra`.

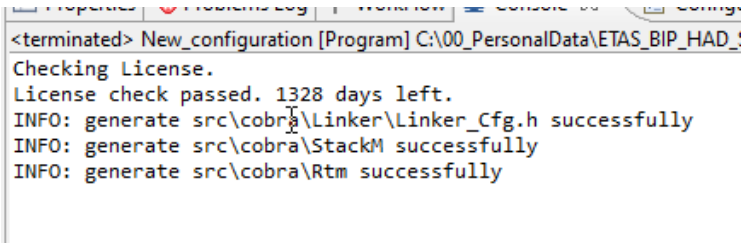


Figure 25 Cobra Extension Generation

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

4.8 Migrate to new RTA-CAR

When migrating to a new RTA-CAR version from BIP baseline

- Pre-condition 1: delete BSW modules [4.3 Integrate Static Modules] and [3.5 How to deploy System to EcuC] under
  `[ProjectRoot]\BasicSoftware\ecu_config\bsw`

  Info: by deleting these artifacts, the
  `[ProjectRoot]\BasicSoftware\ecu_config\bsw\ gen\swcd` and
  `[ProjectRoot]\BasicSoftware\ecu_config\bsw\ gen\bswmd` provided by the older RTA-CAR version are also deleted.

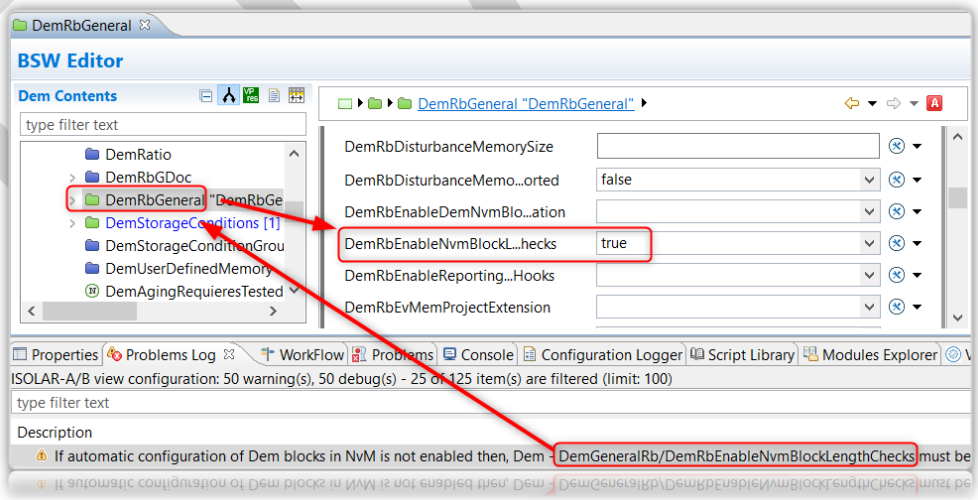
- Pre-condition 2: delete BSW modules parameter definition arxml files under
  `[ProjectRoot]\BasicSoftware\ecu_config\paramdefs`

> **NOTE**
>
> User shall **NOT** keep BIP `[ProejctRoot]\BSW\ecu_config\paramdefs` as parameter definition for configuration in newer RTA-CAR version !

- Peer-module Arxml: user need to start from a complete [3.5 How to deploy System to EcuC].

- BSW module static configuration: user need to **completely** configure EcuC static modules after [3.5 How to deploy System to EcuC] by using new `paramdefs` shipped with this new RTA-CAR.

  Hint: during configuration, RTA-CAR gives you informaiton and configuration advise in Problem Log like screenshot below. Double click the information will navigate you to the corresponding BSW modules where the advise refer to.



- Generate BSW again by referencing section [4.4 How to generate BSW Code].

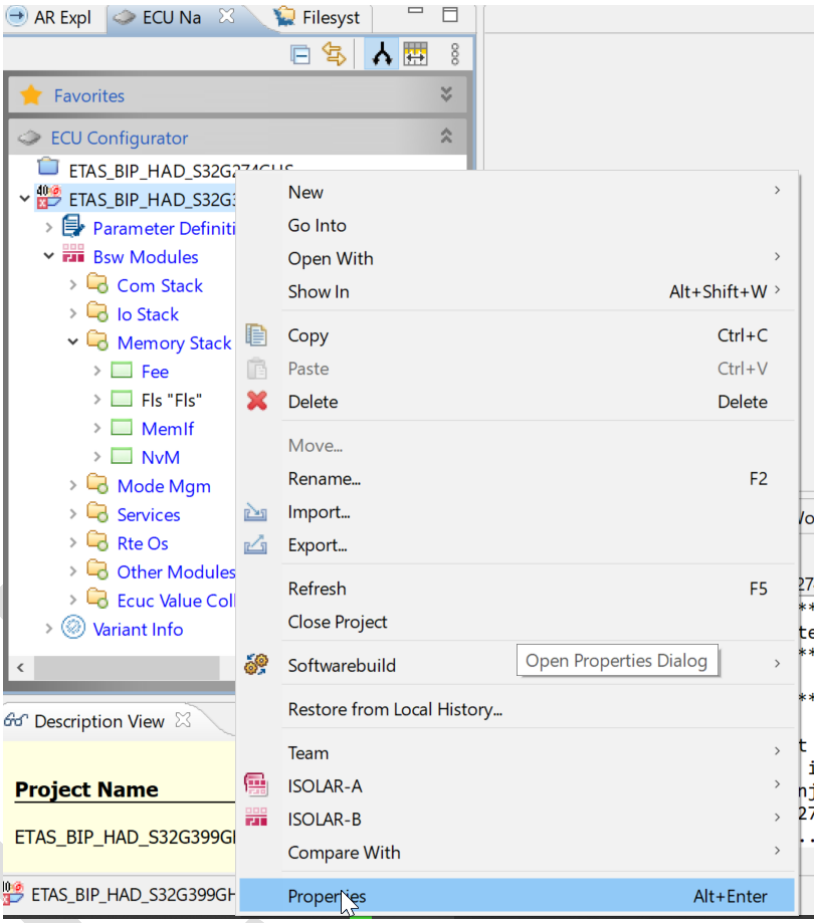
  Info: Newer RTA-BSW is likely providing **changed service interfaces** generated into service description swcd as a result of BSW generation. In this case users are expected to adapt the BSW service interface referencing from their SWCs.

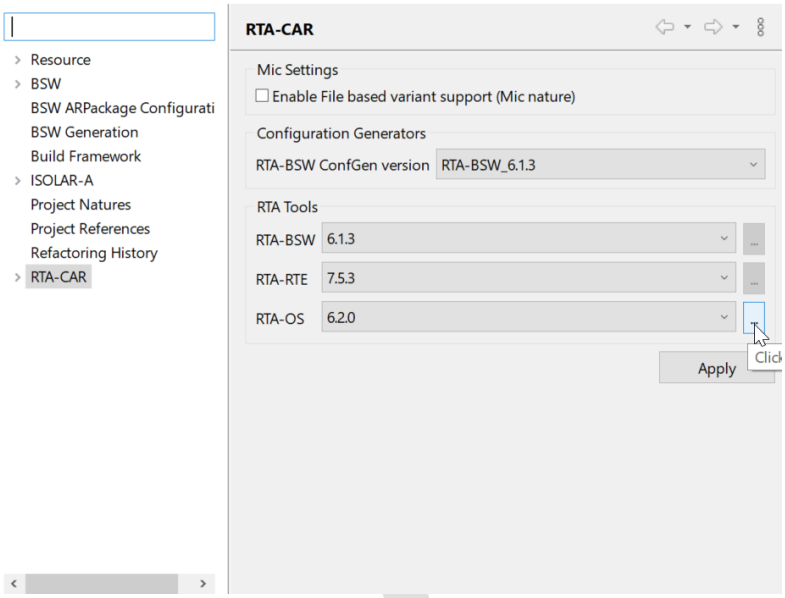
4.9 Upgrade to a newer OS Port

During continuous development and integration, user may receive newer RTA-OS port for **existing target hardware and compiler** in use.

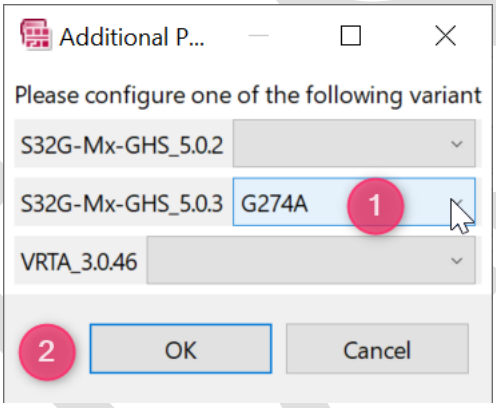
User could follow below steps to replace the 'older' port with the 'newer' port:

1) First install the port into target installation folder, e.g. C:\ETAS\RTA-CAR_9.2.1\RTA-OS_6.2.0
2) Open RTA-CAR toolchain configuration by right click on your ISOLAR project, open RTA-OS target selection window as shown in below figure.

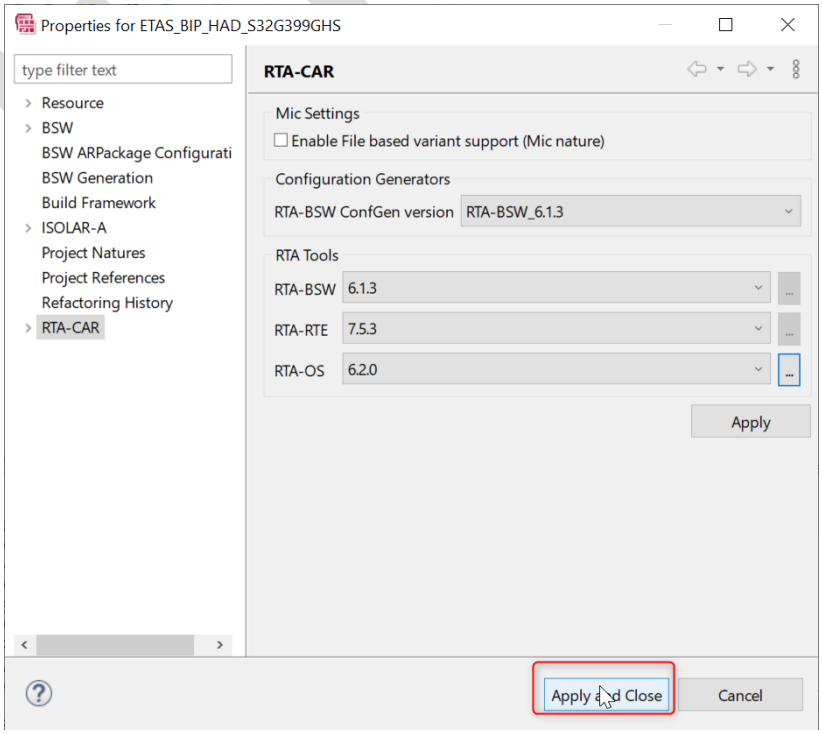




3) Select the 'newer' OS port you received and click **OK** to apply your selection.



4) Click **Apply and Close** to let ISOLAR to import the 'new' OS port properties into ISOLAR OS configuration navigator.



5) Now you will be able to work with the 'newer' port for configuration and generation with the installed 'newer' port.

4.10 Migrate to new Target

BIP integrate BSW and MCAL at both configuration (this chapter) and software level (chapter 6).

When migrating to new target/MCAL during configuration phase, section [4.3.1 Platform Integration] and sub-sections <**Porting BIP to New Target**> of this chapter need to be followed when user migrating project to a new target/MCAL.

4.11 Migrate to new Compiler

Nothing need to be modified when user migrating project to a new compiler.

ETAS

5 Application Integration

BIP ship with three composition clusters to enable user easiler and faster deployment of their application SW-components to dedicated ECU partition.

- **AppCluster**: Collecting function level application SW-components
- **ComCluster:** Collecting communication related SW-components
- **SysCluster:** Collecting ECU level services modules and SW-components

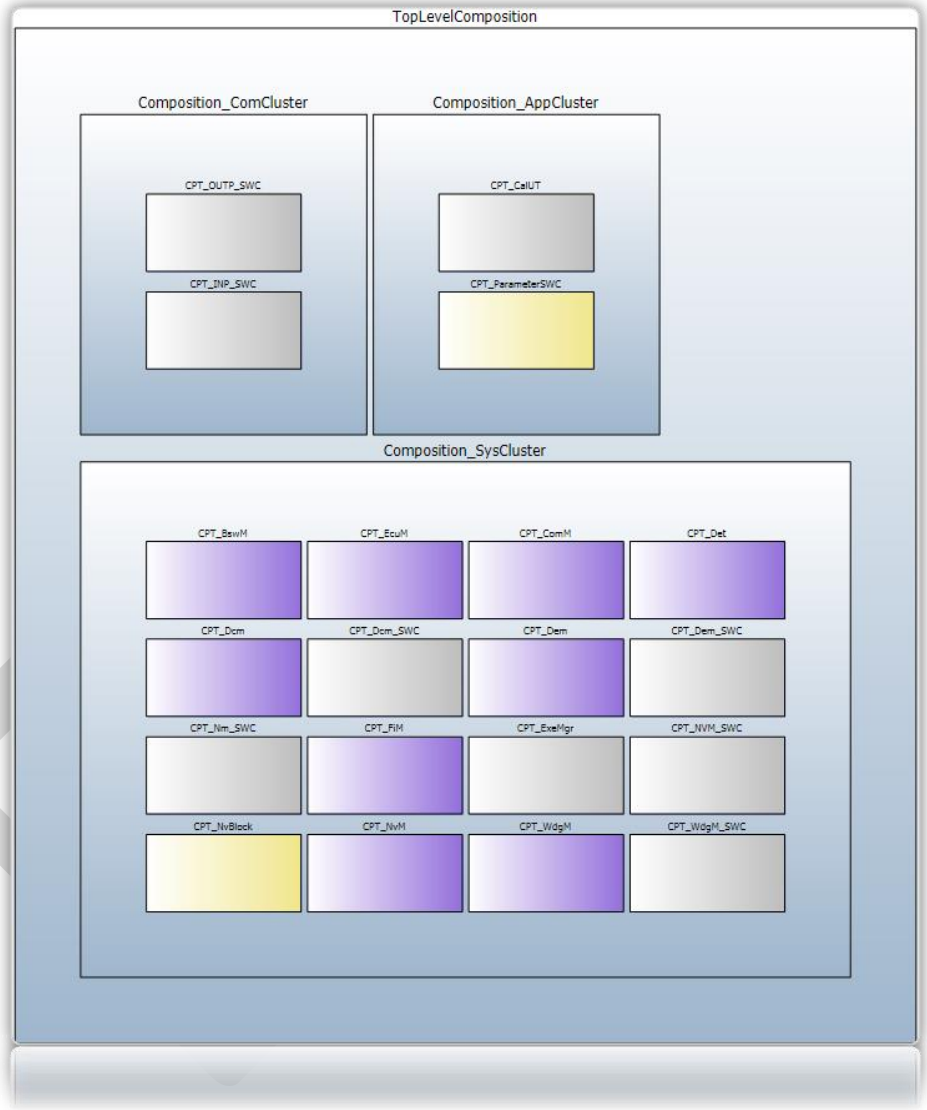


Figure 26 BIP Cluster based Composition

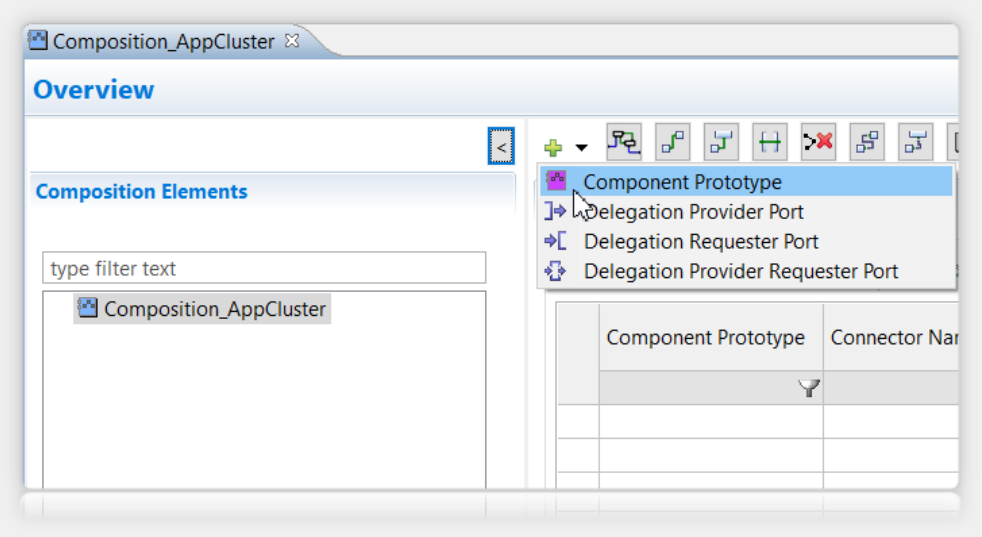
5.1 AppCluster – How to deploy user Application

Application cluster collects user application software components that could be deployed into a singla or multiple ECUs in a system.

With BIP, user have below steps to deploy their application to ECUs.

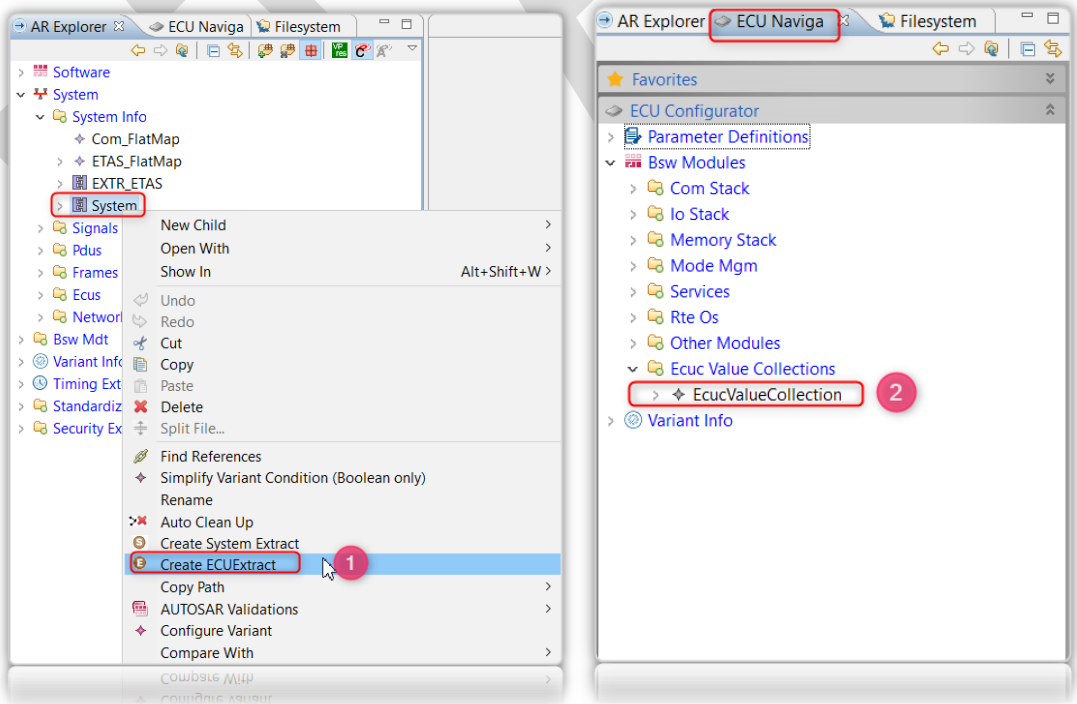
5.1.1 Assign application to BIP AppCluster

[$BIP_INTG 057] Assign user application SW-Components into the composition AppCluster.



5.1.2 Extracting ECU

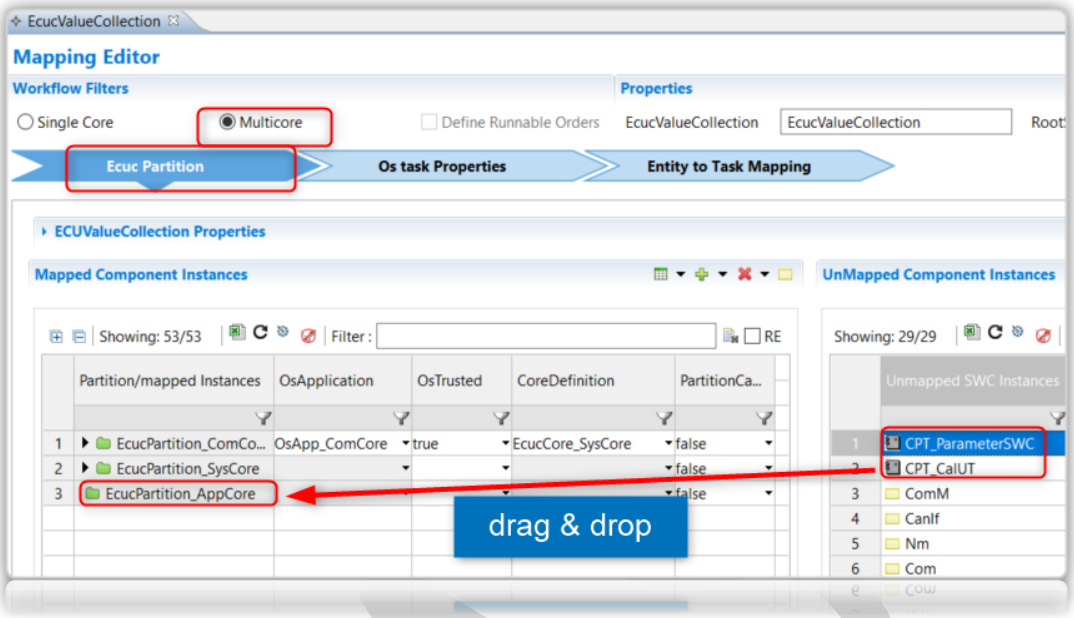
[$BIP_INTG 058] Then user will need to update AUTOSAR EcuCValueCollection with an updated Ecu Extraction by (1) **System** | **Create EcuExtract**, which will forward updated ECU extract information to (2) **BSW** | **Ecuc Value Collections** automatically as we **ONLY add** new elements.



5.1.3 Assign Application to Partition

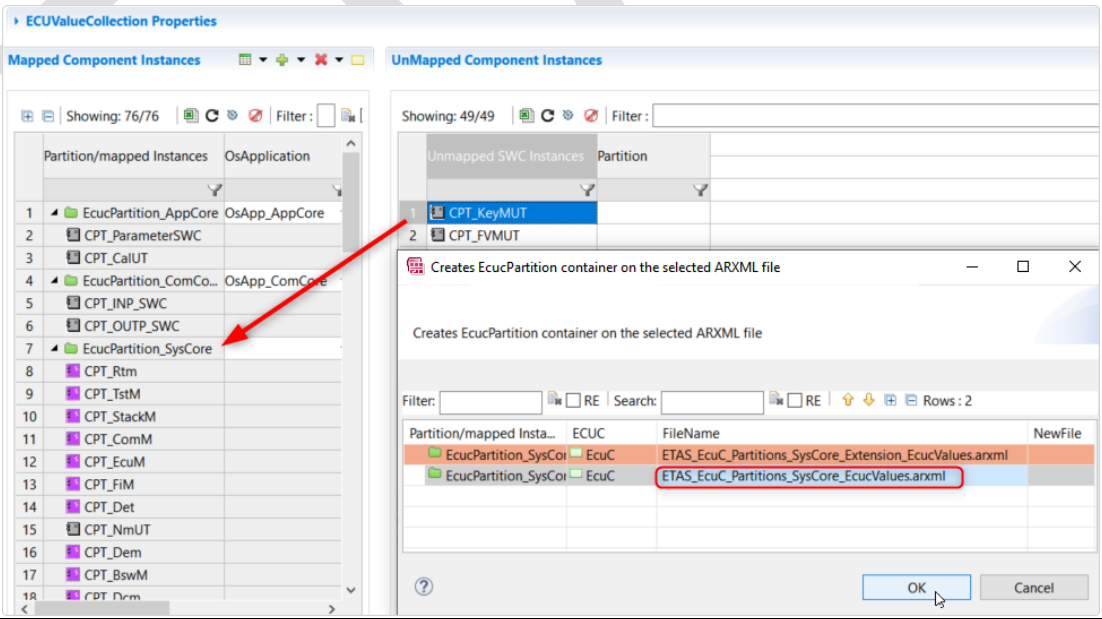
[$BIP_INTG 059] Double click on **Bsw** | **EcucValueCollection** to open EcuCValue description in <**EcucPartition Multicore**> view.

[$BIP_INTG 060] Assign the application SW-Components into Application partition by **drag** & **drop**.



NOTE

In split configuration mode as section [4.1 Split BSW Configuration with ISOLAR-B], you will be asked which partition container (arxml file) will you assign the component by drag & drop a component to a partition. E.g., when assigning `CPT_KeyMUT` to `EcuPartition_SysCore`, ISOLAR prompt asking you which container you will put the component into. Confirm your selection by clicking the container and OK.

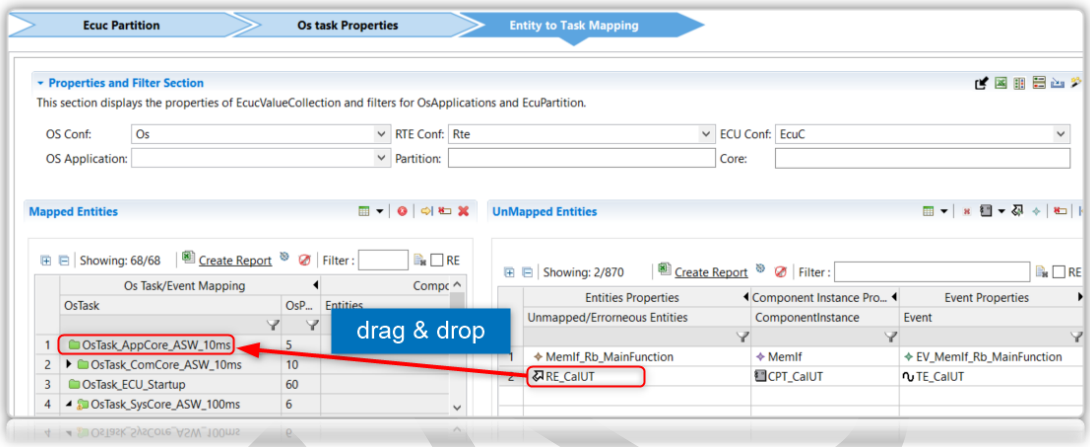


5.1.4 Assign Application to Task

[$BIP_INTG 061] Double click on **Bsw** | **EcucValueCollection** to open EcuCValue description in <**Entity to Task Mapping**> view.

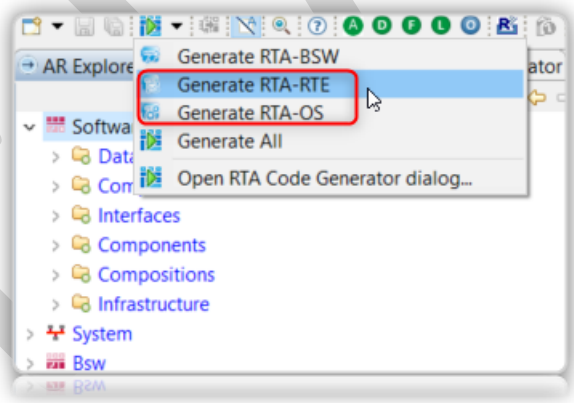
[$BIP_INTG 062] Then assign user application runnables to Os-Aplication tasks by **drag** & **drop**.

***Info**: It will not be described here regarding how to create tasks. User could go to ISOLAR reference manual or consult ETAS expert for those informaiton.*



5.1.5 Generate RTE and OS

[$BIP_INTG 063] After user applciation runnables has be assigned to OS tasks, generate rte and OS by click <**Generate RTA-RTE**> and <**Generate RTA-OS**> respectly.



NOTE

OS and RTE generation configuration of BIP has been pre-configured by ETAS, if you encournter problems during generation, please [9 Contacting ETAS] !

5.2 ComCluster – How to update Signal Interfaces

[3.5 How to deploy System to EcuC] auto-configure I-Signals to ECU Com module.

BIP enable user **one-click** creation of **AUTOSAR interfaces** for communication signals as section [5.2.1 Create signal interfaces with BIP Cobra], and deploy their signal communication to ComCluster with below steps.

5.2.1 Create signal interfaces with BIP Cobra

BIP ships with a tool Cobra_Interface that is bounded to the RTA-CAR version and BIP.

[$BIP_INTG 064] User could iteratively create and update their signal interfaces when their system and network description has been modified on needs.

Cobra_Interface creates below artefacts by one-click.

- Signal interfaces
- INP_SWC
- OUTP_SWC

5.2.2 Setup Cobra_Interface

Input:

System description arxml file, Com EcuC Value arxml file (optional)

Output:

`SRInterface_gen.arxml`, `INP_SWC.arxml`, `OUTP_SWC.arxml`

Arguments:

`--ProjectPath`: The path of the project obtained by selecting the project in navigator.

`--Ecu`: This argument defines the ECU name for which signal interfaces are generated.

Description:

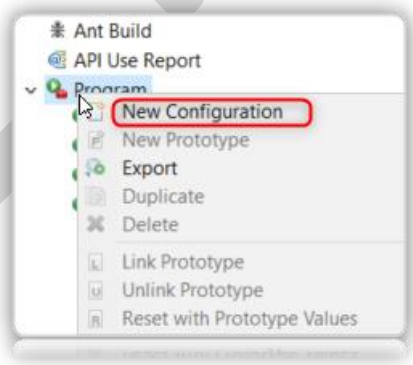
With Interface Add-On, **sender-receiver-interfaces** of signals (contained in ISignalIPdu) in system description will be generated. Two SWCs with R/P Ports of ComSignals can be generated.

Workflow:

Select Ecu for which interfaces shall be generated by editing argument in external tool configuration of section [5.2.3 Install and Run Cobra].

5.2.3 Install and Run Cobra

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_Interface tool environment and run it by click on "**Run**".

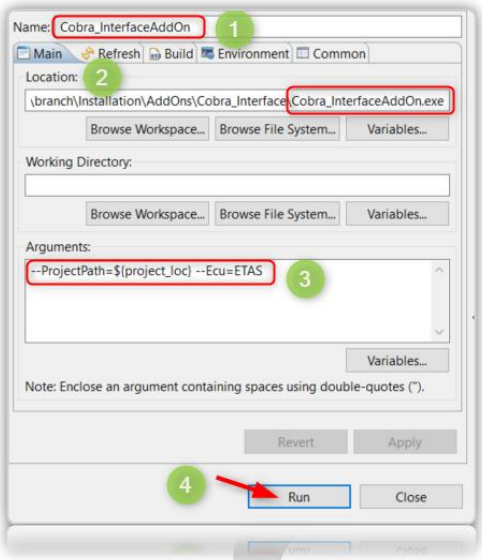


Figure 27 Install Cobra Interface add-on

As a result of run Cobra_Interface add-on, the console prints success information with SRInterface_gen.arxml generated under project path.

NOTE

The base type that is referenced by I-Signal shall have corresponding implementation data type with same name under AR Package `/AUTOSAR_Platform/ImplementationDataTypes` *in file* `Platform_Types.arxml` *because the generated interfaces shall reference valid implementation datatype.*

Hint: In console window, user is asking if you want to generate SWCs with R/P Ports. Input **`Yes`** and press enter key. (Figure 28).

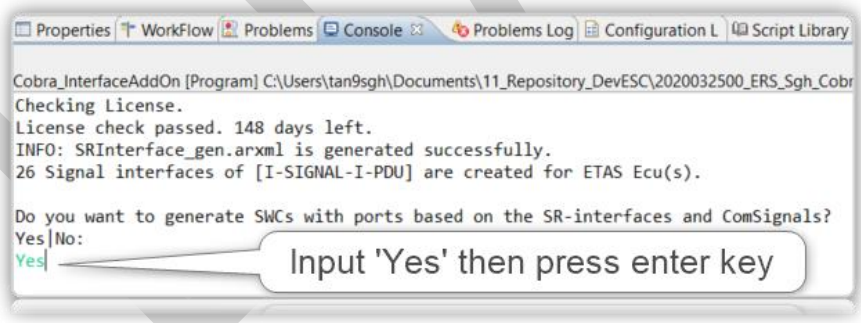


Figure 28 Cobra Interface Generation

Outputs are shown in figure below.

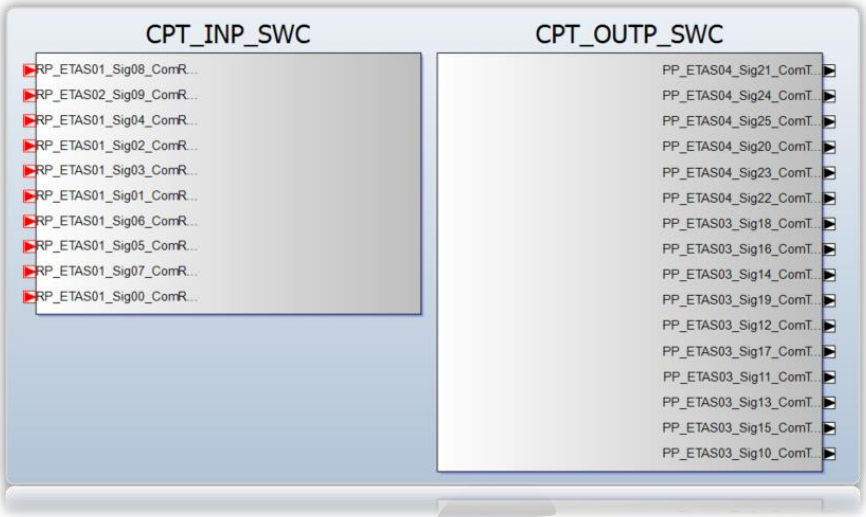


Figure 29 Output of Cobra Interface Add-On

NOTE

After signal interfaces generated, user shall

(1) Create runnable with required RTE **event** within INP and OUTP SW-components.

(2) Assign signal interface data to dedicate runnable by **data access points** as per project needs.

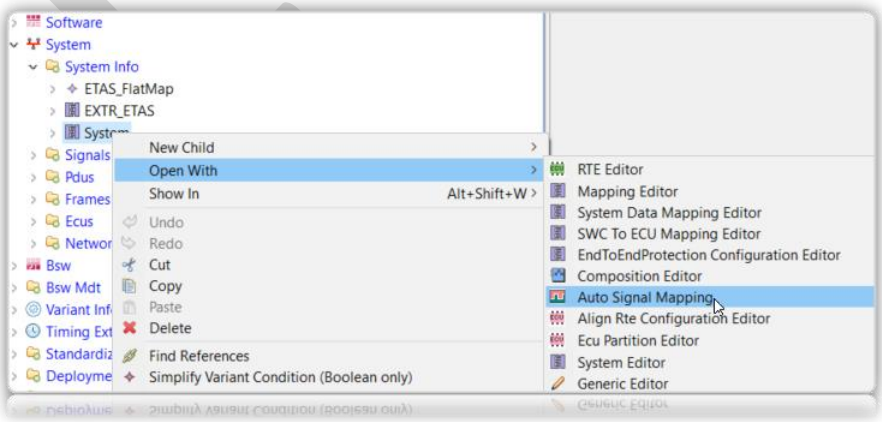
Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

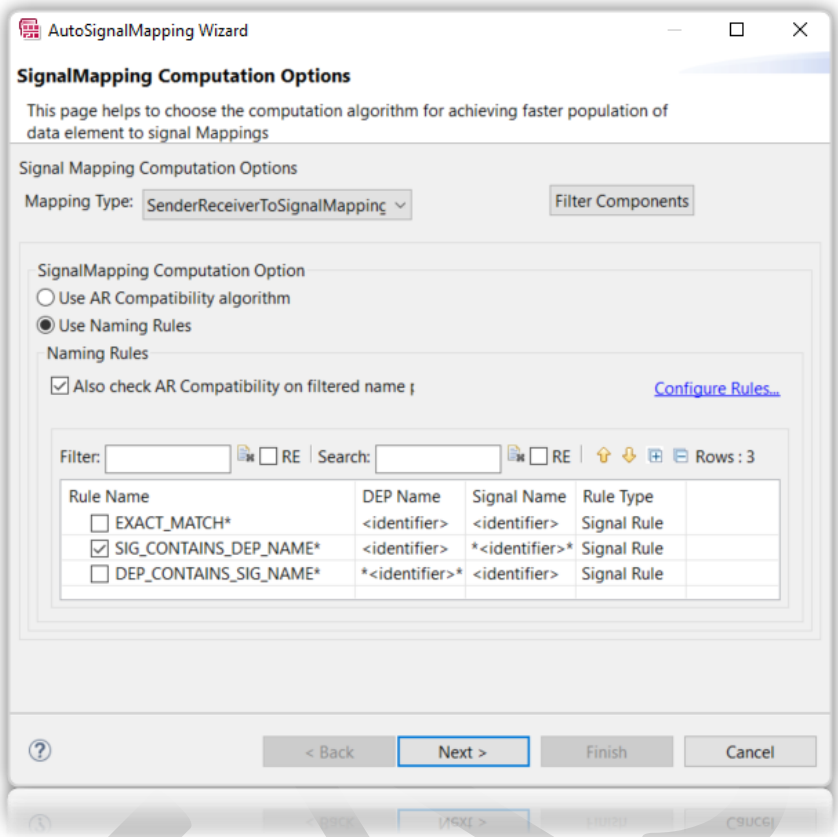
5.2.4 Auto-Mapping System Signal

[3.5 How to deploy System to EcuC] auto-configure I-Signals to ECU Com module. These **AUTOSAR signals** created in section [5.2.1 Create signal interfaces with BIP Cobra] need to be connected to EcuC by following below operation.

[$BIP_INTG 065] ISOLAR-A provide a convenient **AutoSignalMapping** feature by **System** | **Open With** | **Auto Signal Mapping** to open signal mapping wizard.



[$BIP_INTG 066] In the **AutoSignalMapping** wizard, signal interfaces could be connected with I-Signals (system view) with configuring matching rules as below.



After signals mapping, Rte shall be generated following section [5.1.5 Generate RTE and OS], which generates RTE **AUTOSAR Interface** which calling **standardized interface** `Com_SendSignal()`, `Com_ReceiveSignal()`, as example below.

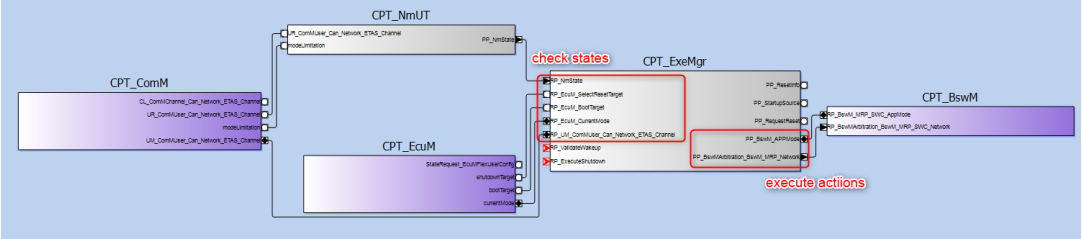
```
FUNC(Std_ReturnType, RTE_CODE)
Rte_ImplWrite_Sig10_ComTx_0x55(VAR(boolean, AUTOMATIC) data)
{
   VAR(Std_ReturnType, AUTOMATIC) rtn = RTE_E_OK;

   /* The signal is ETAS03_Sig10_ComTx_0x55 */
   if ( ((VAR(StatusType, AUTOMATIC))E_OK) != Com_SendSignal(((VAR(Com_SignalIdType, AUTOMATIC))9), &data) )
   {
      rtn = ((VAR(Std_ReturnType, AUTOMATIC))RTE_E_COM_STOPPED);
   }

   /* Send complete */
   return rtn;
}
```

5.3 SysCluster – How to use BIP System Services

5.3.1 Mode Handling

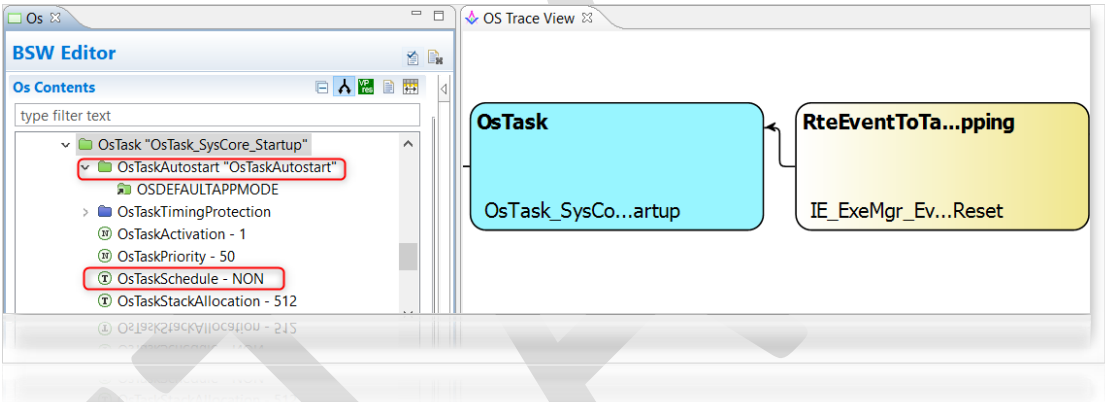
ETAS BIP uses ExeMgr to collect Ecu and Network state during startup, runtime and shutdown to perform particular state change.

- Cyclic read the network state to decide whether requrest FULL/NO communication or not.
- ExeMgr will cyclic check wakeup source to decide whether request run or not.



Evaluate Ecu Reset Reason at Startup

[$BIP_INTG 067] ETAS BIP read the last reset reason that are configured in EcuM by `RE_ExeMgr_EvaluateReset()` at the startup of SysCluster by setting it at begin of the SysCluster non-preemptive auto-activated OS Task with highest priority.



By this, `RE_ExeMgr_EvaluateReset()` is guaranteed to be running after EcuM initialization and OS started but before SysCluster application running.

NOTE

User Integration Required: [$BIP_INTG 068] User is expected to evaluate the reset reason and perform user integrations on project specific needs in `RE_ExeMgr_EvaluateReset()`.

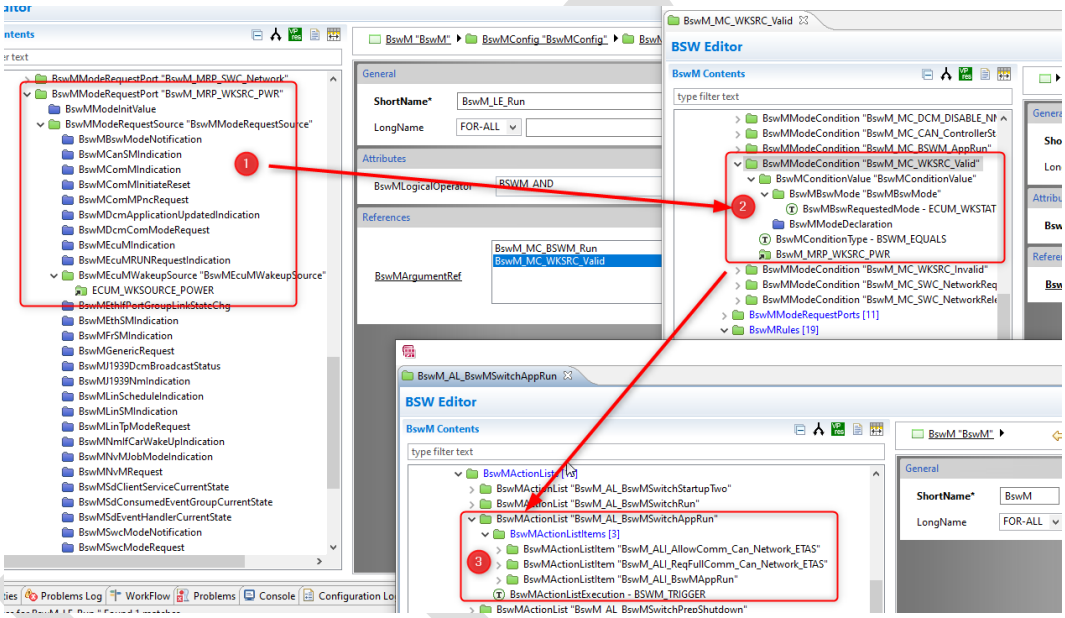
Evaluate Active Wakeup at Startup

[$BIP_INTG 069] ETAS BIP evaluates any active wakeup source by `RE_ExeMgr_EvaluateWakeup()` after evaluation fo Ecu reset reason at the startup of SysCluster immediately after EcuM initialization and OS started.

- [$BIP_INTG 070] If at least one active wakeup source is active, ExeMgr informs this wakeup source to EcuM and continues startup process and scheduling.

```
FUNC (void, ExeMgr_CODE) RE_ExeMgr_EvaluateWakeup(void)
{
    /* Trigger validation of current wakeup source at startup. */
    Rte_IrTrigger_RE_ExeMgr_GetWakeupSource_ITP_GetWakeupSource();

    /* The pending Wakeup Events will be reported to EcuM by EcuM_SetWakeupEvent immediately after StartOS. */
    if(STARTUP_SOURCE_NONE != Rte_IrvIRead_RE_ExeMgr_EvaluateWakeup_ExeMgr_StartupSource())
    {
        EcuM_SetWakeupEvent(EcuMConf_EcuMWakeupSource_ECUM_WKSOURCE_POWER);
        EcuM_ValidateWakeupEvent(EcuMConf_EcuMWakeupSource_ECUM_WKSOURCE_POWER);
    }
    else
    {
        EcuM_ClearWakeupEvent(EcuMConf_EcuMWakeupSource_ECUM_WKSOURCE_POWER);
    }

}
```

[$BIP_INTG 071] Also BswM will catch this notification of state of EcuM wakeup source. Then commnicaiton started actions will be executed.



- [$BIP_INTG 072] If no valid startup source detected, BIP starts BswM managed shutdown process with scheduling.

NOTE

User Integration Required: `RE_ExeMgr_EvaluateWakeup()` evaluates the active wakeup source by `RE_ExeMgr_GetWakeupSource()` where BIP set hardwire wakeup available by default as:

```
FUNC (void, ExeMgr_CODE) RE_ExeMgr_GetWakeupSource(void)
{
    ExeMgr_StartupSourceType startupSource = ~STARTUP_SOURCE_NONE;
```

User is expected to add their project specific hardwire wakeup sources detection code into `RE_ExeMgr_GetWakeupSource()`.

Polling Active Wakeup during Runime

[$BIP_INTG 073] ETAS BIP polls any active wakeup source by `RE_ExeMgr_MainFunction()` during runtime.

> **NOTE**
>
> **User Integration Required**: `RE_ExeMgr_MainFunction()` polls the active wakeup source by `RE_ExeMgr_GetWakeupSource()` provided IRV.
>
> [$BIP_INTG 074] User is expected to add their project specific hardwire wakeup sources detection code into `RE_ExeMgr_GetWakeupSource()`, otherwise it always return available wakeup source and no shutdown request will be proceed in [5.3.1 Handling Shutdown Request during Runime].

Network State Check

[$BIP_INTG 075] ETAS BIP cyclicly check any user request of FULL/NO communication in dicated by variable `Nm_Test[]`.

```
if(Nm_Test[index_chanel].request_b)
{
    /* S/R to request COMM_FULL_COMM mode by ExeMgr */
    Rte_Write_PP_NmState_uint8(APP_READY_FOR_NM_REQUEST);
    Nm_Test[index_chanel].request_b = 0;
}
else
{
    Rte_Write_PP_NmState_uint8(APP_NM_INIT);
}
if(Nm_Test[index_chanel].release_b)
{
    /* S/R to request COMM_NO_COMM mode by ExeMgr */
    Rte_Write_PP_NmState_uint8(APP_READY_FOR_NM_RELEASE);
    Nm_Test[index_chanel].release_b = 0;
}
```

[$BIP_INTG 076] For any user requests for network indicated by variable `Nm_Test[].request_b` or network release request indicated by variable `Nm_Test[].request_b`, NmUT informs to ExeMgr who cyclicly reads network state provided by NmUT to execute mode request of FULL/NO communication.

```
Rte_Read_RP_NmState_uint8(&NmState_u8);
if (NmState_u8 == APP_READY_FOR_NM_REQUEST)
{
    Rte_Write_PP_BswMArbitration_BswM_MRP_Network_uint8(RTE_MODE_ComMMode_COMM_FULL_COMMUNICATION);
}
else if (NmState_u8 == APP_READY_FOR_NM_RELEASE)
{
    Rte_Write_PP_BswMArbitration_BswM_MRP_Network_uint8(RTE_MODE_ComMMode_COMM_NO_COMMUNICATION);
}
else if (NmState_u8 == APP_NM_BUS_SLEEP)
{
    AppMode_TC = RTE_MODE_MDG_ECUM_STATE_ECUM_STATE_POST_RUN;
}
```

NOTE

User Integration Required: User is expected to perform further integration of `Nm_Test[].request_b` by either replace it as **AUTOSAR Interface** or directly modify NmUT to user specific requirements (i.e. CAN network message has received or Can network message has gone).

Handling Shutdown Request during Runime

[$BIP_INTG 077] ETAS BIP handling shutdown request by `RE_ExeMgr_MainFunction` during runtime.

[$BIP_INTG 078] At runtime, if user application detected shutdown condition (i.e. network is in sleep state), ETAS BIP provides a global variable `StateMachine_Test` which shall be set to `APP_MODE_REQUEST_POST_RUN` (set SW at post run) and then `INVALID_WAKEUP_SOURCE` (to invalid wakeup source), then shutdown sequence will be executed. (BIP State machine 4.3.4)

NOTE

User Integration Required: User is expected to perform further integration of `StateMachine_Test` by either replace it as **AUTOSAR Interface** or directly modify state machine of ExeMgr to user specific requirements (i.e. you might need your ECU to stay awake if Run Crank is still active even if CAN has gone down.)

Get Ecu Reset Information during Runtime

[$BIP_INTG 079] ETAS BIP provide a service operation by AUTOSAR Interface `CSI_ExeMgr_ResetInfo` with service operation `RE_ExeMgr_ResetInfo()` which user application software could read the EcuM configured reset reason at any time after startup.

NOTE

User Configuration Required: If user have specific reset reasons that need to be saved during shutdown and be extracted after next start up, you shall go to EcuM configurations.

For how to perform EcuM static configuration, that is not intended in this document and please [9 Contacting ETAS].

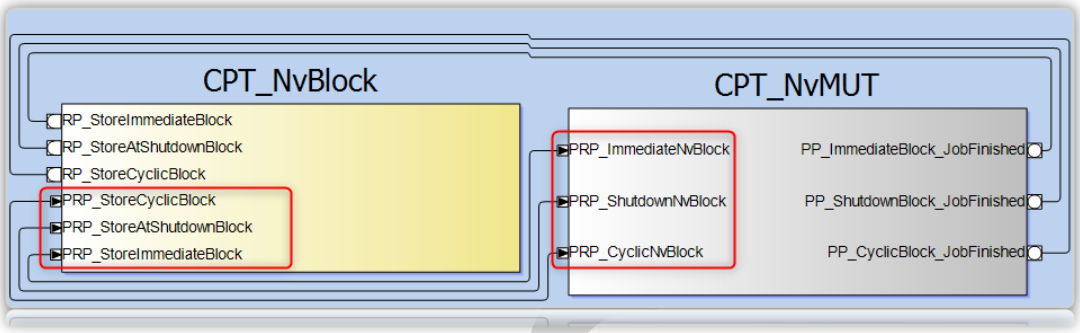
5.3.2 Add Non-volatile Data

ETAS BIP provide three types of NVBlocks in `NvBlock` Sw Component Type SWC to store user non-volatile RAM data need to be saved in non-volatile memory when ECU Power-off.

`NvBlock`	**Description**
`StoreCyclicBlock`	Application data changes during runtime and is expected to be stored periodically
`StoreImmediateBlock`	Application data that are expected to be stored immediately during operation to prevent loss.

`StoreAtShutdownBlock`	Application data changed during operation and is expected to be saved when the ECU is powered off

To use the three types of NvBlock to store user application data, user will need to connect their SWCs to NvBlock SWC by **AUTOSAR Interface** as example of NvMUT.



Immediately Store

Immediate storage is expected to be stored directly to the Nvm without waiting.

[$BIP_INTG 080] When the user needs to store application data, application software could directly call `Rte_Write_<P>_<O>`, `<p>` is the port name, `<O>` is NvData Prototype name.

Such as in NvMUT:

```
/* APP code */
. . .
/*APP RAM DATA running Change*/
Write_Block_Immediate_u32[0] = Write_Block_Immediate_u32[0]++;
/* APP code */
. . .
/* test block write immediately  */
Rte_Write_PRP_ImmediateNvBlock_ImmediateNvBlock(Write_Block_Immediate_u32);
. . .
```

NOTE

The actual immediate storage is executed immediately, but the actual writing to flash of the STORAGE to the Nvm is done through background task 1ms scheduling `MemIf_Rb_MainFunction()`.

Periodic Store

Periodic storage application data is expected to be periodically stored in non-volatile memory with user defined calling period.

[$BIP_INTG 081] When the user needs to store application data, application software could periodic call `Rte_Write_<P>_<O>`, `<p>` is the port name, `<O>`is NvData Prototype name.

Such as in NvMUT, it is periodic called every 100ms:

```
FUNC (void, NVM_SWC_CODE) RE_NVM_SWC_100ms
(
        void
)
{
    /* APP code */
    . . .
    /*APP RAM DATA Write_Block_Cyclic_u8 running Change */
    Write_Block_Cyclic_u8[0] = Write_Block_Cyclic_u8[0]++;
    /* APP code */
    . . .
    /* test block call write in Periodic timer is 100ms */
    Rte_Write_PRP_CyclicNvBlock_CyclicNvBlock(Write_Block_Cyclic_u8)
;
    . . .
}
```

NOTE

The actual Periodic storage is executed Periodic timer, but the actual writing to flash of the STORAGE to the Nvm is done through background task 1ms scheduling `MemIf_Rb_MainFunction()`.

Shutdown Store

Application data is saved to a non-volatile memory before power-off.

[$BIP_INTG 082] When the user needs to store application data, application software could directly call `Rte_Write_<P>_<O>` , `<p>` is the port name, `<O>`is NvData Prototype name.

Such as in NvMUT:

```
    /* APP code */
    . . .
    /*APP RAM DATA  Write_Block_Cyclic_u8  running Change  */
    Write_Block_ShutDown_u8[0] = Write_Block_ShutDown_u8[0]++;
    /* APP code */
    . . .
    /* test block call write shutdown block  */
    Rte_Write_PRP_ShutdownNvBlock_ShutDownNvBlock(Write_Block_ShutDown_u8);
    . . .
```

NOTE

[$BIP_INTG 083] The actual shutdown storage is executed during ECU shutdown by calling function `EcuM_OnGoOffTwo`, which call `NvM_Integration_WriteAll()` to performs block writes to non-volatile memory.

```
void NvM_Integration_WriteAll(void)
{
```

```
    NvM_Rb_StatusType    NvM_Status = NVM_RB_STATUS_UNINIT;
    MemIf_StatusType     MemIf_Status = MEMIF_UNINIT;

    /* Schedule table has been stopped before NvM_WriteAll. */
    NvM_WriteAll();

    do{
        /* Request Memory layers handle request */
        NvM_MainFunction();
        MemIf_Rb_MainFunction();

        /* Get the status of request */
        (void)NvM_Rb_GetStatus(&NvM_Status);
        MemIf_Status = MemIf_Rb_GetStatus();

    } while((NvM_Status == NVM_RB_STATUS_BUSY) || (MemIf_Status == MEMIF_BU
SY));
}
```

Power On Restore

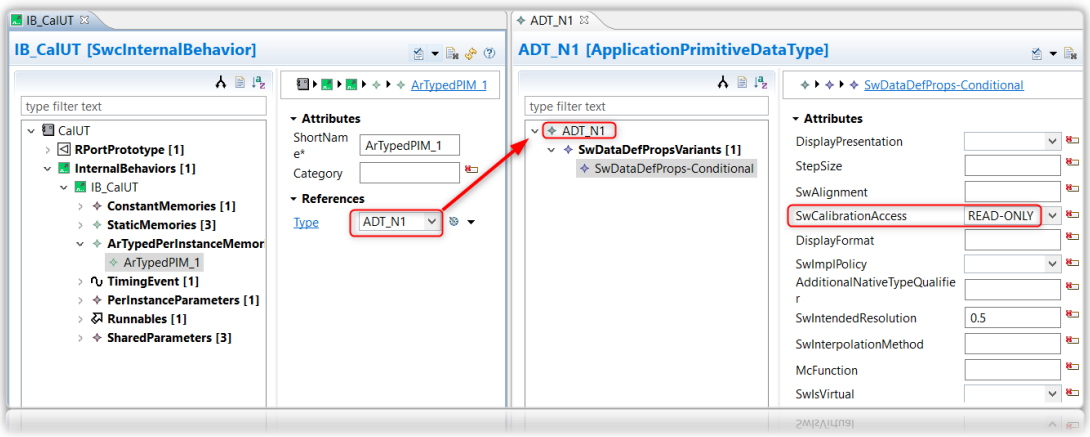
[$BIP_INTG 084] Application data Power-on restore data from non-volatile memory, need call `Rte_Read_<P>_<O>` in startup task, `<p>` is the port name, `<O>` is NvData Prototype name.

Such as in NvMUT:

```
FUNC (void, NVM_SWC_CODE) RE_NVM_SWC_Init (void)
{
   /* test cyclic store block init read */
   (void)Rte_Read_PRP_CyclicNvBlock_CyclicNvBlock(Read_Block_Cyclic_u8);
    /* test immediate store block init read  */
    (void)Rte_Read_PRP_ImmediateNvBlock_ImmediateNvBlock(Read_Block_Immedia
te_u32);
    /* test store at shutdown block init read  */
    (void)Rte_Read_PRP_ShutdownNvBlock_ShutDownNvBlock(Read_Block_ShutDown_
u8);
}
```

5.3.3 Add Measurement Variables

Cobra-BIP has pre-configrued AUTOSAR measurement variables by CalUT SWC **ArTypedPerInstanceMemory** with **SwCalibrationAccess** `READ-ONLY`.



When user would like to add their own measurement variables, you could define your variables in the same way of CalUT.

5.3.4 Add Calibration Data

There are several ways to describe calibration parameters by AUTOSAR Authoring tool e.g. ETAS ISOLAR-A:

Type of Parameter	**Description**
Parameter SWC	Provides a **calibration data pool** that can be accessed by all SWCs
SWC inner parameter data: **Per Instance Parameter** **Shared Parameter** **Constant Memory**	Calibration parameter that can be used in the context of the internal behavior of a SWC.

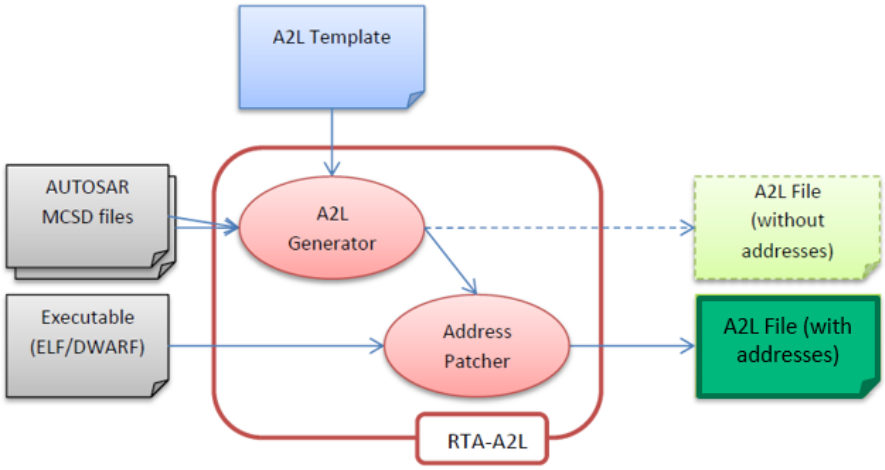
Cobra-BIP has pre-configrued AUTOSAR calibration parameter with both ways.

When user would like to add their own calibration parameters, you could define your variables in the same way of CalUT.

5.3.5 Generate A2L File

[$BIP_INTG 085] After user added measurement variables and calibration parameters, a RTE generation will automaticlly collect their application variables and parameters into AUTOSAR MCSD file `Rte_McSupportData.arxml`

ETAS BIP support one click generation of A2L file as senario below.



[$BIP_INTG 086] To do so, user could double click `[ETAS-BIP]\TOOLS\A2L\run0.bat` which invoke RTA-A2L from command-line and generate A2L into `[ETAS-BIP]\TOOLS\A2L\ETAS_BIP.a2l`

NOTE

It is assumed that you have installed RTA-CAR to your default path at `C:\ETAS\RTA-CAR_x.y.z`. User may need to adjust variable `%RTAA2L%` of `run0.bat` to match your RTA-CAR installation path.

```
set RTAA2L=C:\ETAS\RTA-CAR_9.1.0\RTA-RTE_7.4.1\Tools\RTA-A2L\bin\RTAA2L.exe
```

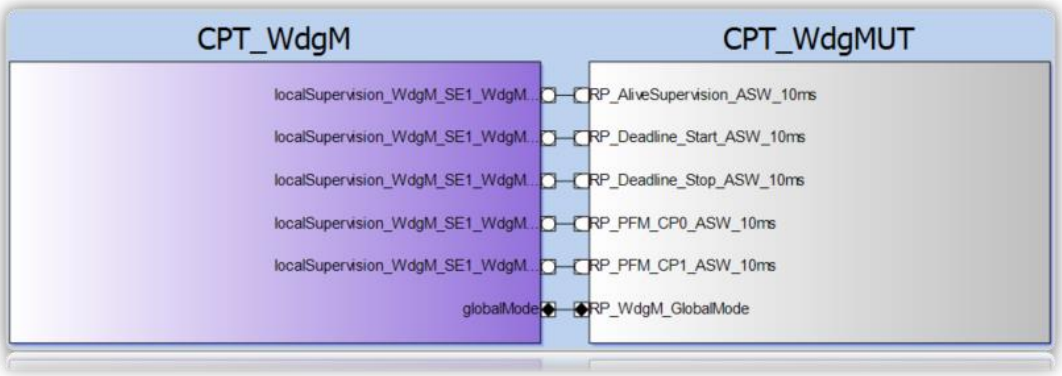
An example of generated calibration parameter as below.

```
/begin CHARACTERISTIC SpeedLimitUpper
      "This is ETAS BIP AUTOSAR PerInstanceParameter Scalar example 2"
      VALUE
      0xe63375a4
      RTAA2L_Internal_Scalar_UnsignedLong
      0
      ETAS_BIP_CompuMethods_N1
      0
      16383.5
      DISPLAY_IDENTIFIER  SpeedLimitUpper
      MAX_REFRESH  1000
                   1
      PHYS_UNIT "rpm"
/end CHARACTERISTIC
```

5.3.6 Monitoring Software Execution

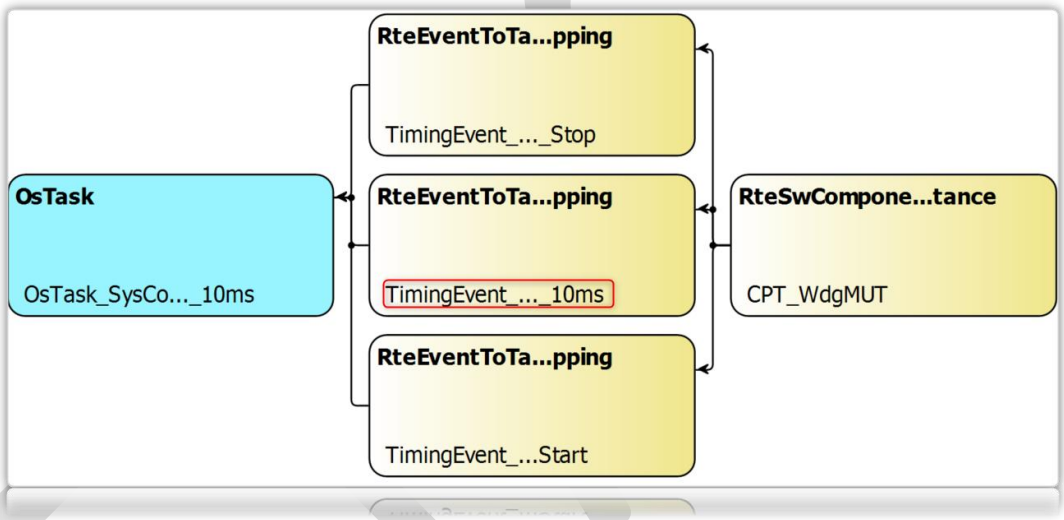
After [4.4 How to generate BSW Code], WdgM service SW-component is generated into `[ProjectRoot]\BasicSoftware\ecu_config\bsw\gen\swcd` describing which **standardized AUTOSAR interfaces** services provided provide to user.

BIP configure WdgM with one supervision entity `WdgM_SE1` to supervise software execution of partition SysCore by **VFB** connection below.



Use Alive Supervision

BIP connect WdgM alive supervision to software execution of `OsTask_SysCore_ASW_10ms` by Rte standardized AUTOSAR interface wrappered in runnable `RE_WdgMUT_ASW_10ms()` .



NOTE

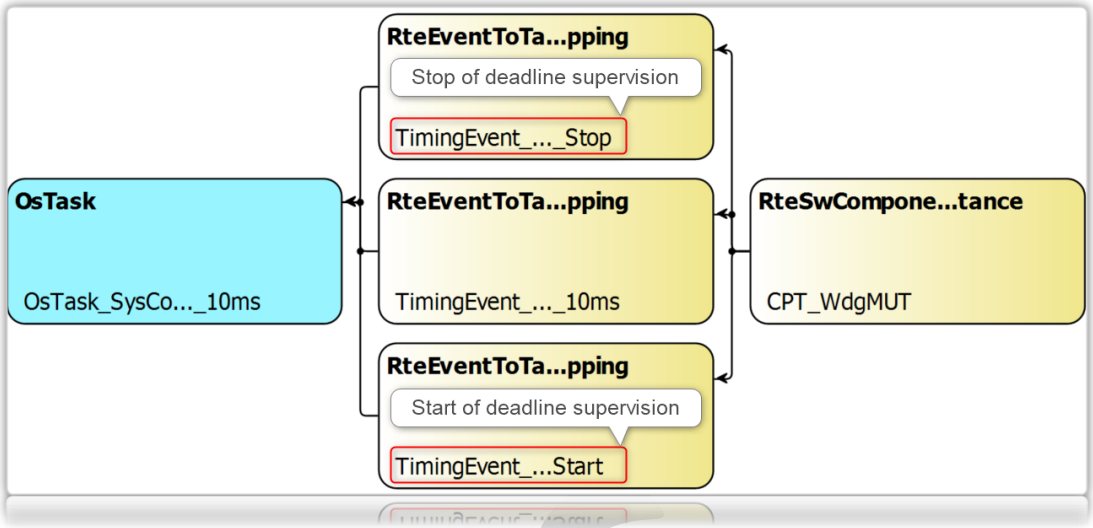
If user expect to supervise alive execution of other entity at **same execution frequency** senario, they shall map `RE_WdgMUT_ASW_10ms()` to that execution.

NOTE

If use expect to supervise an entity executing at a **different frequency** senario, they shall modify profile of `WdgMAliveSupervision` within WdgM EcuC ref. [4.3 Integrate Static Modules].

Use Deadline Supervision

BIP connect WdgM deadline supervision to worst case execution time of Task `OsTask_SysCore_ASW_10ms` by Rte standardized AUTOSAR interface wrappered in runnable `RE_WdgMUT_ASW_10ms_Start()` and `RE_WdgMUT_ASW_10ms_Stop()`.



NOTE

If user expect to supervise deadline execution of **other entity** with **same worst case** execution time configured in BIP, they shall map `RE_WdgMUT_ASW_10ms_Start()` and `RE_WdgMUT_ASW_10ms_Stop()` to that execution.

NOTE

If use expect to supervise entity executes at a **different worst case time**, they shall modify profile of `WdgMDeadlineSupervision` within WdgM EcuC ref. [4.3 Integrate Static Modules].

Use Logical Supervision

ETAS BIP configure a local graph for the WdgM supervision entity `WdgM_SE1` by providing two checkpoints `WdgM_SE1_PFM_CP_0` and `WdgM_SE1_PFM_CP_1`.

BIP connect WdgM logical supervision

- checkpoint 0 to execution of Task `OsTask_SysCore_BSW_10ms` by Rte standardized AUTOSAR interface wrappered in runnable `RE_WdgMUT_PFM_CP_0()`.
- checkpoint 1 to execution of Task `OsTask_SysCore_ASW_10ms` by Rte standardized AUTOSAR interface wrappered in runnable `RE_WdgMUT_PFM_CP_1()`.

NOTE

If user expect to supervise logical execution of **other entity** with **same internal graph** as configured in BIP, they shall map `RE_WdgMUT_PFM_CP_0()` to start of a graph and `RE_WdgMUT_PFM_CP_1 ()` to end of **the same** graph.

NOTE

If use expect to supervise entity that transion in a **different graph** (different number of checkpoints or different transitions), they shall modify profile of `WdgMLocalStatusParams` within WdgM EcuC ref. [4.3 Integrate Static Modules].

5.4 Migrate to new RTA-CAR

When migrating BIP application integration to a new RTA-CAR version, user could directly copy BIP `[ProjectRoot]\BasicSoftware\swc_config` to the same location of newer RTA-CAR project.

- **AppCluster**: no need to be modified
- **ComCluster**: no need to be modified
- **SysCluster**: may need to be modified by remapping BSW service interfaces in composition `Composition_SysCluster` if newer RTA-BSW module provides **changed service interfaces** generated into service description swcd during [4.4 How to generate BSW Code].

5.5 Migrate to new Target

Nothing need to be modified when user mograting project to a new target/MCAL.

5.6 Migrate to new Compiler

Nothing need to be modified when user migrating project to a new compiler.

6 Software Integration

The BIP is organized in a folder structure distinguishing which user shall modify and which shall keep no change.

User shall refer to section ***Folder Structure*** of <ETAS BIP Release Notes.pdf> to understand which files/folders are

- **Static code**: Avoid manual modification on those
- **Generated code**: Replace these after generation from modified configuration.
- **Integration code**: BSW callback and callout functions that are not defined by AUTOSAR but helpful during user software integration.

This section focuses on **Integration code** that software integrator to integrate when

- Modify BIP baseline integration to fit for project needs
- Porting BIP to new Target (MCAL)

> **NOTE**
> **About BSW Integration Code: The Integration Code are provided by RTA-BSW as templates and require user software integration throughout the rest of this section.**

6.1 Generate BSW Integration Code

[$BIP_INTG 087] When BIP generate BSW modules for the first time, <**Generate Integration Code**> has been select in [**RTA code generator**] as shown below.

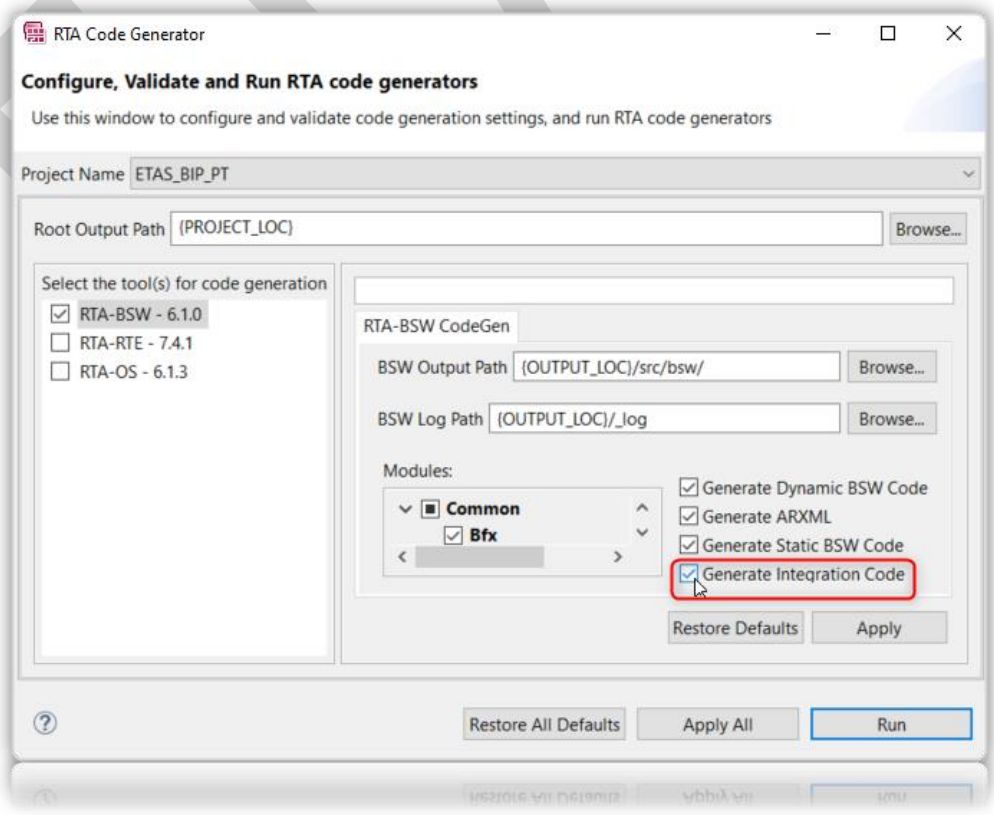


Figure 30 Generate BSW Integration Code

- [$BIP_INTG 088] The **BSW Integration Code** for every **configured** module have been generated into `[ProjectRoot]\BasicSoftware\src\bsw\[BswModules]\integration`, and then moved into location `[ProjectRoot]\BasicSoftware\integration\src\bsw` for software integration purpose.

- [$BIP_INTG 089] BSW generic integration folder `[ProjectRoot]\BasicSoftware\src\bsw\integration` has been moved into `[ProjectRoot]\BasicSoftware\integration\src\bsw` for including of Rte and OS code generation.

NOTE

<**Generate Integration Code**> shall and have to be only enabled for the first time with user migrating to a new RTA-CAR version, as it never changed for a RTA-CAR version.

6.2 Integrate BSW-MCAL

Following AUTOSAR layed architecture, interfaces between BSW and MCAL are **standardized interface** which are direct API calls. For most standardized interface, most MCAL vendor has standard API that are directly integrated with BSW.

However there could be a few interfaces that need a **tinny** wrapper to integrate BSW and MCAL, e.g. MCAL API name contain Vendor ID.

6.2.1 EcuM - MCU

MCU Initialization

[$BIP_INTG 090] BIP integrates MCAL modules to EcuM startup-I initialization by including MCAL integration header files `[MCAL_Module]_Integration.h` from `[ProjectRoot]\BasicSoftware\integration\src\bsw\EcuM\user\EcuM_PBDefine.h`

```
#include "MCAL_Integration.h"
#include "EcuM.h"
```

Porting BIP to new Target

[$BIP_INTG 091] When porting BIP to new target, user shall adapt and add MCAL modules include file into `Mcal_Integration.h` to fit for MCAL specific configuration definition.

MCU Reset

[$BIP_INTG 092] BIP has integrated EcuM Reset interface `EcuM_AL_Reset()` with MCAL provide **standardized interface** `Mcu_PerformReset()` .

If user would change the MCU reset methods, then they could change implementation of `EcuM_AL_Reset()`.

6.2.2 CanIf - CAN

AUTOSAR defines CAN module API `CanIf_RxIndication()` differently between AR 4.2.2 and AR 4.4.0.

ETAS CanIf module of RTA-BSW(NG) works with both API.

Porting BIP to new Target

[$BIP_INTG 093] When integrating a specific MCAL, it's AUTOSAR release version shall be specified in `[ProjectRoot]\BasicSoftware\integration\src\bsw\CanIf\integration\CanIf_Integration.h` to let CanIf choose the right API to use.

```
/*
 * Set the MCAL minor release version here.
 * Example:
 * For AR 4.0.3, the minor version is 0U
 * For AR 4.2.2, the minor version is 2U
 * For AR 4.3.1, the minor version is 3U
 * Default is 2U for compliance with AR 4.2.2
*/
#define CANIF_RX_INDICATION_VERSION (2U)

#define CANIF_WRITE_INTEGRATION_VERSION (2U)

#define CANIF_CONTROLLERMODEINDICATION_VERSION (4U)

#define CANIF_SETCONTROLLERMODE_INTEGRATION_VERSION (4U)//MCAL version is 4.4.0
```

[$BIP_INTG 094] When porting BIP to new target, user shall adapt MCAL **standardized interface** and configuration definitions in BIP `[ProjectRoot]\BasicSoftware\integration\mcal\src\api\Can_Integration.h (or Can.h)` to fit for MCAL vendor specific API definition and configuration.

> **NOTE**
> In some specific MCAL, the module driver files is like Module_VendoerIndex.c/.h. (e.g. Can_17_McmCan.c/ Can_17_McmCan.h), but the others are like Module.c/.h (e.g. Can.c/Can.h).

6.2.3 Wdg - MCAL

AUTOSAR defines Wdg module with below three APIs

- `Wdg_Init`
- `Wdg_SetMode`
- `Wdg_SetTriggerCondition`

MCAL vendor normall provide these services with vendor IDs `Wdg_[vendorId]_xxx`.

[$BIP_INTG 095] BIP intend to integrate Wdg API in `[ProjectRoot]\BasicSoftware\integration\mcal\src\integration\api\Wdg.h`

Porting BIP to new Target

[$BIP_INTG 096] When porting BIP to new target, user shall adapt MCAL **standardized interface** and configuration definitions in BIP `[ProjectRoot]\BasicSoftware\integration\mcal\src\integration\api\Wdg.h` to fit for MCAL vendor specific API definition and configuration.

```
#define Wdg_SetTriggerCondition          Wdg_43_Instance0_SetTriggerCondition
#define Wdg_SetMode                      Wdg_43_Instance0_SetMode
#define Wdg_Init                         Wdg_43_Instance0_Init
```

6.2.4 Fee - Fls

[$BIP_INTG 097] ETSA BIP implemented non-volatile memory feature with MCAL Fls. To accelerate the read speed, BIP integrates Fee for synchronous data read from data flash by memory copy in function `Fee_Fls_SyncRead()`.

`Fee_Fls_SyncRead()` need to know the base address of data flash which is configured as Fls base address.

Porting BIP to new Target

[$BIP_INTG 098] When porting BIP to new target, user shall adapt the Fls base address according to their MCAL configuration by specify the definition `FLS_BASE_ADDRESS` in `[ProjectRoot]\BasicSoftware\integration\mcal\src\integration\api\Fls_Integration.h/Fls.h` as below.

```
#define FLS_BASE_ADDRESS        0U      /* User Integration. */
```

[$BIP_INTG 099] When porting BIP to new target, user shall adapt MCAL **standardized interface** and configuration definitions in BIP `[ProjectRoot]\BasicSoftware\integration\mcal\src\integration\api\Fls_Integration.h/Fls.h` to fit for MCAL vendor specific API definition and configuration.

6.3 Integrate BSW-Non MCAL

Also there are non-AUTOSAR MCAL standardized modules need user to integrated with BSW for their project specific needs.

6.3.1 Det Error Hook

[$BIP_INTG 100] BIP provides Hooks notification to user to support debugging and error tracing during development and runtime in `[ProjectRoot]\BasicSoftware\integration\src\bsw\Det\user\Det_Integration.c`

NOTE

User Integration: The implementation of Det Hooks are not AUTOSAR standardized and user shall handle them at project specific needs.

Det Hook	Description
`Det_ReportErrorHook()`	This function notify user for reported BSW error from BSW modules.
`Det_ReportRuntimeErrorHook()`	This function notify user for reported BSW runtime error from BSW modules.
`Det_ReportTransientFaultHook()`	This function notify user for reported BSW

	transient fault from BSW modules.

Table 5 Integrating Det Error Hook

6.3.2 BswM Action

[$BIP_INTG 101] BIP provides several BswM user integration functions in `[ProjectRoot]\BasicSoftware\integration\src\bsw\BswM\user\BswM_Integration.c` to ease user integration based on BIP baseline.

CDD Initialization

NOTE
User Integration: During BswM start-up phase **StartupTwo** and **Run**, user could add their CDD initialization functions in BswM action container.

BswM action	Description
`BswM_StartupTwo_CddInit()`	Initialize CDDs need to initialize before sensor supply, load supply and Dem initialization.
`BswM_Run_CddInitZero()`	Initialize CDDs will be initialized after Dem initialization but before sensor supply, load supply
`BswM_Run_CddInitOne()`	Initialize CDDs will be initialized after Dem initialization and sensor supply, but before load supply

Table 6 Integrating CDD Initialization

Power Supply Initialization

NOTE
User Integration: During BswM start-up phase **Run**, if user have power supply control required, user could use BswM action container as below.

BswM action	Description
`BswM_Enable_PeripheralSply()`	Enable power supply to peripherals, e.g. ADC conditional pull-up circuit, etc.
`BswM_PerformSelfHold()`	After peripheral supply enabled, if user need to perform self-hold for power supply, they could add their function here.

Table 7 Integrating Power Supply Initialization

Power Supply De-Initialization

NOTE
User Integration: During BswM **Shutdown** phase, if user have power supply control to release, user could use BswM action container as below.

BswM action	Description
`BswM_SelfHold_Unlock()`	Action to unlock power supply.

Table 8 Integrating Power Supply De-Initialization

Can Transceiver Initialization

NOTE

User Integration: During BswM start-up phase **Run**, if user have CAN transceiver configured and its initialization need to be called before Can stack initialization, user could use BswM action container as below.

BswM action	Description
`BswM_CanTrcv_GotoNormal()`	Action to call CAN transceiver initialization function to set Can transceiver go to normal operation mode.

Table 9 Integrating Can Transceiver Initialization

Can Tranceiver De-Initialization

NOTE

User Integration: During BswM **Shutdown** phase, if user have CAN transceiver configured and it's intended to be setup after Can communication has been stopped, user could use BswM action container as below.

BswM action	Description
`BswM_CanTrcv_GotoSleep()`	Action to call CAN transceiver De-initialization function to set Can transceiver go to sleep.

Table 10 Integrating Can Transceiver De-Initialization

6.3.3 Can Transceiver

Can Transceiver integration function are called by BswM in section [6.3.2 BswM Action].

NOTE

User Integration: BIP provides a user integration function in `[ProjectRoot]\BasicSoftware\src\integration\bsw\CanTrcv\user\CanTrcv_Integration.c` for user to put their can transceiver integration based on BIP baseline.

6.3.4 EcuM Shutdown

BIP has configured EcuM shutdown with three phases executing from step 1 to step 3:

1) `EcuM_OnGoOffOne`: Shutdown phase I before OS scheduling stopped.
2) `EcuM_OnGoOffTwo`: Shutdown phase II after OS stopped.
3) `EcuM_AL_Reset/EcuM_AL_SwitchOff`: Shutdown phase III for execution.

EcuM_OnGoOffOne

[$BIP_INTG 102] BIP configures `EcuM_OnGoOffOne` with four steps:

1) `EcuM_GoOffOne()`
2) De-initialize BswM
3) De-initialize SchM

4) `ShutdownOS()`

[$BIP_INTG 103] BIP integrated `EcuM_GoOffOne()` in `[ProjectRoot]\BasicSoftware\integration\src\bsw\EcuM\user\EcuM_Integration.c` to stop OS schedule table by `Rte_Stop()` and `Dem_Shutdown()`.

NOTE

Scheduling and Peripheral ISRs are already disabled after calling `ShutdownOS()` so user shall not enable interrupts at this point of time.

EcuM_OnGoOffTwo

[$BIP_INTG 104] BIP configures `EcuM_OnGoOffTwo` with three steps:

1) `TstM_PostRun`
2) `NvM_WriteAll`
3) `ExeMgr_ExecuteShutdown`

NOTE

User Integration: Please refer to section [6.3.8 Test Management] for `TstM_PostRun()`.

NOTE

User Integration: User is expected to implement, if any, project specific SWC level shutdown handlings in `ExeMgr_ExecuteShutdown()`.

EcuM_AL_SwitchOff

[$BIP_INTG 105] BIP integrates `EcuM_AL_SwitchOff` by integration function `Ecu_SwitchOff()` in `[ProjectRoot]\integration\src\ecu\StartupShutdown\src\Ecu.c`

NOTE

User Integration: User is expected to implement project specific go off strategy in `Ecu_SwitchOff()`, e.g. shutdown by SBC via SPI.

NOTE

[$BIP_INTG 106] It's not expected to return from `Ecu_SwitchOff()` to normal software execution.

EcuM_AL_Reset

[$BIP_INTG 107] `EcuM_AL_Reset` has one input parameter of type `EcuM_ResetType`. The parameter gives the user requested reset type that is configured in EcuM static configuration, refer to section [4.3.3 **Hint:** Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

EcuM – Mcu Reset].

During software integration, BIP implements `EcuM_AL_Reset` with interfaces as reference to section [0 MCU Reset].

Modifying BIP

[$BIP_INTG 108] BIP implements `EcuM_AL_Reset` with only the configured necessary reset type. When modifying BIP to project needs by adding additional reset type, user shall integrate reset handler in `EcuM_AL_Reset`.

Porting BIP to new Target

[$BIP_INTG 109] When porting BIP to new target, user shall adapt implementation of `EcuM_AL_Reset` with proper reset handler intended for all the configured `EcuM_ResetType` referencing section [4.3.3 **Hint:** Please refer to [4.1 Split BSW Configuration with ISOLAR-B] when modify the configurations.

EcuM – Mcu Reset].

6.3.5 Target Clock

RTA-CAR has several BSW modules need to get elapsed counter values (ticks) from system during runtime.

- WdgM: Requires this tick for deadline supervision.
- Xcp: Runtime measurement with Xcp.
- CanTp: CanTp segment data transmission & reception timing
- Os: Os time monitoring with `Os_Cbk_GetStopWatch()`

[$BIP_INTG 110] BIP provides an integration API `Target_GetSysTicks32Bits()` in `[ProjectRoot]\BasicSoftware\integration\src\target\src\Target.c`.

[$BIP_INTG 111] BIP also provide an integration 32bit width CAT2 timer interrupt fire at `1ms frequency in [ProjectRoot]\BasicSoftware\integration\src\target\src\Target.c` who will provide software tick for OS task scheduling.

Porting BIP to new Target

NOTE

User Integration: When porting BIP to new target, user shall provide a value of a 32bit free-running hardware counter to `Target_GetSysTicks32Bits()`.

NOTE

As fast as the counter ticks at frequency of CPU instruction cycle, the OS time monitoring will be much more previse. However, User shall evaluate the longest time the counter could tick before a wraparound, which would be the longest timing interval for measurable deadline for WdgM deadline supervision, e.g.
*When a counter tick at 500MHz, the maximum measurable deadline supervised entity is 2e32 * 2ns = 8.58s*.

6.3.6 Xcp

BIP has integrated Xcp for synchronized data acquisition and overlay based calibration data management.

Access Authorization

[$BIP_INTG 135] Xcp support user access authorization by SeedNKey when user attempts to perform calibration, DAQ acquisition, stimulation, or programming. When establish a Xcp session, measurement and calibration tools (e.g., INCA) will use the SeedNKey algorithm described in A2L file to unlock Ecu for the intended access.

BIP provide a reference SeedNKey algorithm both in `XcpAppl_Unlock()` and in SeedNKey.dll.

When user need to apply own algorithm, both XcpAppl_Unlock() and in SeedNKey.dll could be updated and rebuild.

Timestamp Data Acquisition

[$BIP_INTG 112] Xcp support synchronized DAQ data acquisition by Xcp timestamp function. BIP has integrated this by implementing an OS software counter `Millisecond_TickCounter` return by `Target_Get1msTicks()` in `[ProjectRoot]\BasicSoftware\integration\src\target\src\Target.c`.

Porting BIP to new Target

NOTE

User Integration: When porting BIP to new target, user shall provide a 1ms free-running counter to tick OS software counter `Millisecond_TickCounter`.

Calibration Data Management

[$BIP_INTG 113] Although calibration management strategy is not standardized AUTOSAR modules, BIP provide implementation baseline with overlay based double page (RP, WP) CDD in `[ProjectRoot]\CDD\OvrlyM`.

Function	Description	Scheduling Point
`Overlay_Init`	Initialize overlay registers necessary for address redirection.	BswM initialization phase **StartupTwo**
`Overlay_Sync`	Copy initial values of calibration parameters from calibration FLASH to calibration RAM	BswM initialization phase **StartupTwo** and XCP command `COPY_CAL_PAGE`
`Overlay_Enable`	Enable overlay to redirect CPU access of calibration parameters from calibration FLASH to calibration RAM	User request switching calibration page to WP by Xcp command `SET_CAL_PAGRE`
`Overlay_Disable`	Disable overlay to direct CPU access of calibration parameters to calibration FLASH	User request switching calibration page to RP by Xcp command `SET_CAL_PAGRE`

Porting BIP to new Target

NOTE

User Integration: When porting BIP to new target, user shall implement overlay CDD functions as described above.

6.3.7 External Watchdog

[$BIP_INTG 114] Similiar with [Wdg - MCAL], BIP pre-configured with below three APIs for external watchdog in in `[ProjectRoot]\CDD\ExtWdg\src\ExtWdg.c`

- `ExtWdg_Init`

- `ExtWdg_SetMode`
- `ExtWdg_SetTriggerCondition`

> **NOTE**
> **User Integration:** When porting BIP to user projects to fit for project where user have external watchdog to use, user shall provide software implementation for these three functions.

6.3.8 Test Management

BIP pre-configured with two function container for user to add their software test features in `[ProjectRoot]\BasicSoftware\src\cobra\TstM\src\TstM.c`.

TstM_PreRun

[$BIP_INTG 115] BIP schedules `TstM_PreRun()` during EcuM initialization, before interrupt initialization and OS initialization.

> **NOTE**
> **User Integration:** User is expected to implement, if any, project specific Pre-runtime tests in `TstM_PreRun()` provided in `[ProjectRoot]\BasicSoftware\src\cobra\TstM\src\TstM.c`.

TstM_Run

[$BIP_INTG 116] BIP schedules `TstM_Run()` during runtime in an 100ms non-pre-emptive lowest priority background task so that execution of this task will not blocking other software from execution.

> **NOTE**
> **User Integration:** User is expected to implement, if any, project specific runtime tests in `TstM_Run()` provided in `[ProjectRoot]\BasicSoftware\src\cobra\TstM\src\TstM.c`.

> **NOTE**
> User shall evaluate fault detection time and fault reaction time with the software they put in `TstM_Run()` so that it meet project needs.

TstM_PostRun

[$BIP_INTG 117] `TstM_PostRun()` is called during shutdown at the point where task scheduling and maskable peripheral ISRs are already disabled.

NOTE

User Integration: User is expected to implement, if any, project specific shutdown tests in `TstM_PostRun()` provided in `[ProjectRoot]\BasicSoftware\src\cobra\TstM\src\TstM.c`.

6.3.9 Startup Code

[$BIP_INTG 118] BIP integrates target startup by providing startup code under `[ProjectRoot]\BasicSoftware\integration\src\target\src`.

NOTE

User Integration: User is expected to review and adapt these start-up code to fit for project specific needs.

6.3.10 Initialize Interrupts

AUTOSAR requires both category 1 and category 2 interrupts to be setup by an AUTOSAR OS during `EcuM_Init()` before system scheduling for each AUTOSAR cores.

[$BIP_INTG 119] BIP integrates interrupts initialization in `VectorTableCore[x].c` under `[ProjectRoot]\BasicSoftware\integration\src\os\src` for each AUTOSAR cores with OS service API `Os_InitializeVectorTable()`.

According to AUTOSAR, this is all required before OS scheduling start with `StartOS()`. However, for some silicon (especially non release version silicon) which have peripheral interrupt controllers (e.g. ARMv7 VIM) which need to be connected to CPU cores before CPU could respond on peripheral events, there could be chip level interrupt sources might start up with interrupts pending and latched into the VIM from some interrupts sources. This could lead to unexpected interrupts might come in before you are really ready to deal with them with `Os_InitializeVectorTable()`.

6.3.11 Trap Handling

[$BIP_INTG 120] BIP pre-configures unknown traps of target hardware as a AUTOSAR Category 1 interrupt `DefaultInterruptHandler()` in target dependent integration source code `[ProjectRoot]\BasicSoftware\integration\src\target\src\Target.c`.

When a trap occurs, user software will be dropped into `DefaultInterruptHandler()` endless loop for debug purpose.

NOTE

User Integration: User is expected to implement these traps to fit for project specific needs.

6.4 Exclusive Area

[$BIP_INTG 121] BIP provides a default SchM exclusive area access as pre-integration in `[ProjectRoot]\BasicSoftware\integration\src\SchM\SchM_Default.h` by suspend and resume CPU maskable category 1 and category 2 interrupts with OS API `SuspendAllInterrupts()`/`ResumeAllInterrupts()`.

SchM API	Pre-Integration
`SCHM_ENTER_DEFAULT()`	`SuspendAllInterrupts()`

SCHM_EXIT_DEFAULT()	ResumeAllInterrupts()

Table 11 Integrating Exclusive Area

6.4.1 User Integration

> **NOTE**
>
> **User Integration**: [$BIP_INTG 122] When user iterating BIP baseline to fit for project specific needs, user shall evaluate and adapt implementation of exclusive are accordingly.

6.5 How to generate Memory Map and Linker File

[$BIP_INTG 123] Memory map specifies output sections into linker file for every software module. BIP uses `MemMap.h` for the whole project with each module maintains it's own `[Module]_MemMap.h`

[$BIP_INTG 124] BIP pre-integrated memory map under `[ProjectRoot]\BasicSoftware\src\cobra\MemMap.`

- `[SWC]_MemMap.h`: Cobra generated MemMap for software components according to BIP partitioning.
- `Rte_MemMap.h`: Cobra generated MemMap for Rte code and data according to BIP partitioning.
- `Bsw_MemMap.h`: Cobra generated Bsw modules MemMap according to BIP partitioning.
- `MemMap.h`: Cobra generated MemMap with compiler abstraction for BIP module sections.
- `[EcuCPartitionName]` linker file: linker sections for BIP modules, e.g. BSW cleared variables located to partition SysCore.

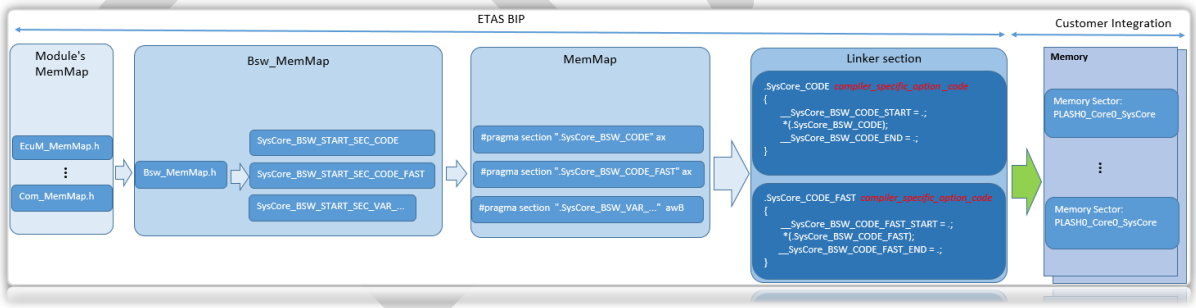
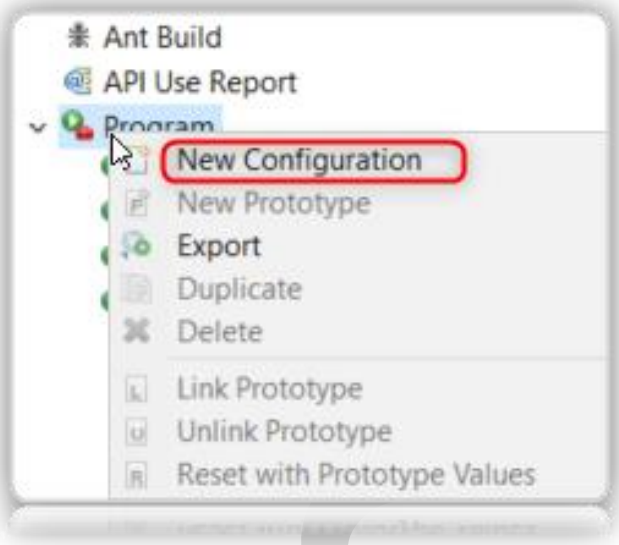


Figure 31 BIP MemMap and Sections

6.5.1 Install and Run Cobra_MemMap

To install the tool, user shall go to toolbar **ISOLAR-AB | Run | External Tools | External Tools Configuration**, right click on "**Program**" to add a New Configuration.



Then follow step 1 – 4 to setup Cobra_MemMap tool environment and Run it by click on "**Run**".

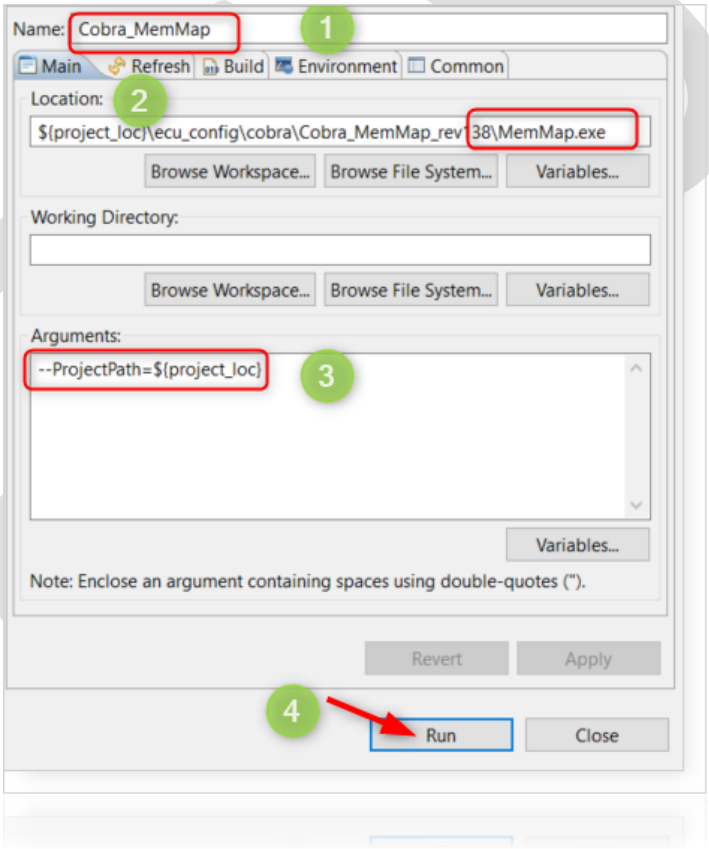


Figure 32 Install Cobra MemMap add-on

As a result of run Cobra_MemMap add-on, the console prints success information with a set of MemMap generated under project path `[ProjectRoot]\BasicSoftware\src\cobra\MemMap`.

Hint: In console window, user is asking if you want to generate SWCs MemMap. Input `Yes` and press enter key.

```
MemMap [Program] C:\00_PersonalData\ETAS_BIP_HAD_      \BSW\ecu_config\cobra\Cobra_MemMap_rev148\MemMap.exe
Checking License.
License check passed. 1328 days left.
Target Compiler is      .
Bsw module partition is EcucPartition_SysCore
Update file C:\00_PersonalData\ETAS_BIP_HAD_      \BSW\src\cobra\MemMap\Bsw_MemMap.h successfully.
Update file C:\00_PersonalData\ETAS_BIP_HAD_      \BSW\src\cobra\MemMap\Rte_MemMap.h successfully.
Generate ASW_MemMap.h? Y|N:
y
Generate C:\00_PersonalData\ETAS_BIP_HAD_      \Targets\Linker\GHS_ARM_EcucPartition_SysCore.ld successfully.
Generate C:\00_PersonalData\ETAS_BIP_HAD_      \Targets\Linker\GHS_ARM_EcucPartition_AppCore.ld successfully.
Generate C:\00_PersonalData\ETAS_BIP_HAD_      \Targets\Linker\GHS_ARM_EcucPartition_ComCore.ld successfully.
Generate C:\00_PersonalData\ETAS_BIP_HAD_      \Targets\Linker\GHS_ARM_shared.ld successfully.

Enter to exit
```

Figure 33 Cobra MemMap Generation

Outputs are shown in figure below.

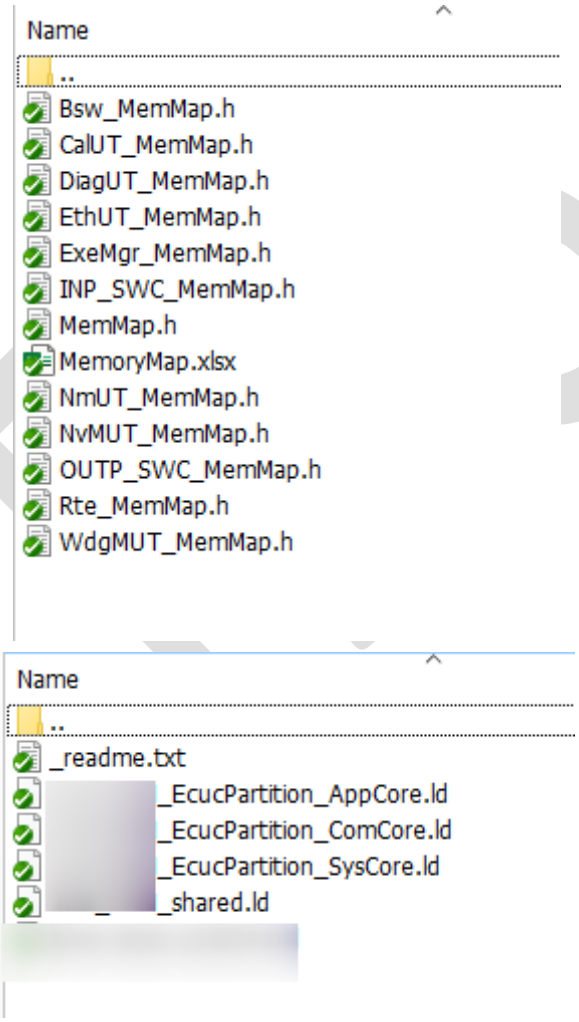


Figure 34 Output of Cobra MemMap Add-On

NOTE

User Integration: When user iterating BIP baseline to fit for project specific needs, user may need to adjust the **Compiler Abstraction** for **Sections** in `MemMap.h` accordingly.

Compatibility

Please refer to section [7 Compatibility] for Cobra compatibilities with RTA-CAR.

6.5.2 Memory Sector

[$BIP_INTG 125] BIP provides top level linker file in `[ProjectRoot]\BasicSoftware\integration\linker` contains memory sectors for the target as example fit for BIP only.

> **NOTE**
>
> **User Integration:** When user iterating BIP baseline to fit for project specific needs, user shall adjust the memory sections according to their memory design.

6.5.3 Add User Sections

User Integration: For user integration of their application into the BIP AUTOSAR project baseline, you may need to create additional sections for your own code.

You shall follow the steps below:

1) Define or maintain their own *`[Module_Name]`*`_MemMap.h` under `[ProjectRoot]\BasicSoftware\integration\src\`*`[Sub-Folders]`*`\integration` folder.
2) Include `MemMap.h` from your `[Module_Name]_MemMap.h` for each sections.
3) Maintain and update `MemMap.h` for sections from `[Module_Name]_MemMap.h`
4) Update linker file

> **NOTE**
>
> [$BIP_INTG 126] When defines sections in `MemMap.h`, user shall pay special attention for alignment definition for your sections.

6.5.4 Porting BIP to new Target

On a Multicore target, the physical volatile memory is usually distiguished as global RAM and core local RAM.

When porting BIP to new multicore targets, several integration principles shall be followed with linker file and startup code.

Explicit Initialization of OS Objects

[$BIP_INTG 127] For an AUTOSAR Multicore target, OS related variables are recommended to be linked into local RAM of master core or global RAM where the master core (as AUTOSAR start-up core) will be able to explicitly initialize OS Multicore Startup synchronization semaphore by `StartOS()`.

Implicit Initialization of Cores Local RAM

A Multicore target usually start cores in a different time frame.

[$BIP_INTG 128] For BIP Multicore Startup, if implicit initialization (clear/copy) of core local RAM is carried by slave cores, user need to pay attentipon that slave core shall not trying to initialize global RAM or other cores local RAM.

This is to avoid curruption of RAM belongs to other cores which start at a previous time frame, e.g. slave cores currupt OS startup synchronization semaphore been explicitly set by master core. As a result of which, the Multicore Startup got hang up at `StartOS()`.

Implicit Initialization of Global RAM

[$BIP_INTG 129] The whole RAM area shall be clearned on startup before copying initialized variables to RAM except for bootloader (if presented) Protected RAM area which could be project specific.

Explicit Initialization of Calibration Memory

[$BIP_INTG 130] RAM area of calibration working page (WP) shall be cold reset cleared or warm reset initialized (non-initialized safe) to intended values so that as soon as user application start using these variables, they contain intended values (e.g. initial values from Reference Page).

Linking and Locating of Calibration Memory

[$BIP_INTG 131] Calibration data reference page (RP) shall be allocated on Flash area which allow read-while-write Flash programming operation in parallel with Flash area of Nv Blocks.

This is to avoid memory confliction between runtime Nv Block write (immediately store or periodic store) and calibration data read from the same read-while-write Flash memory.

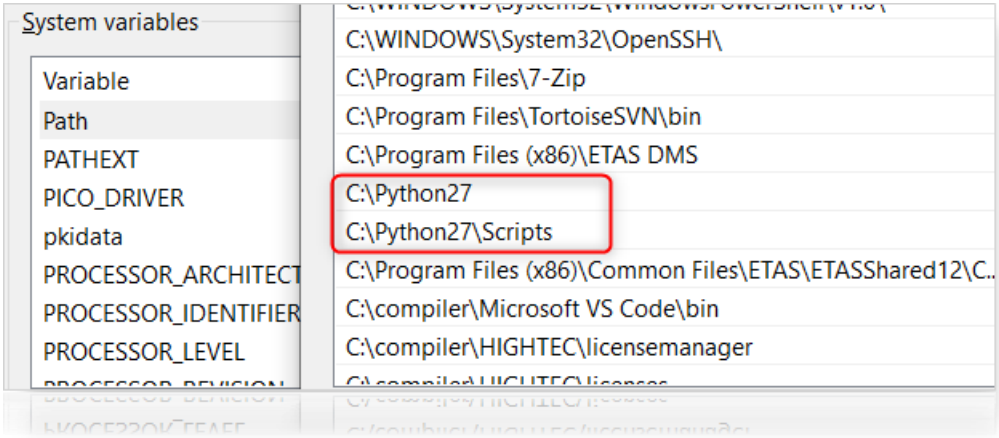
6.6 Setup Build Environment

6.6.1 Install Build Environment

BIP is build using below open source scripting toolchains.

Build Tool	Version
Python	python-2.7.amd64
scons	scons-3.0.0

1. To Install Python, user will need to double click **python-2.7.amd64.msi** and select the install path, e.g. by default at `C:\Python27`
2. Then configure below environments variables into your `%PATH%`
   1) `C:\Python27`
   2) `C:\Python27\Scripts`



3. After python installed, you will need to install scons into Python by
   1) unzip scons-3.0.0.zip
   2) Run the scons installation script **setup.py** inside the installation folder by typing "`python setup.py install`" on the Command prompt, Scons will then be installed into the folder C:\Python25\scons-x.x.x

By now, you will be able to buid your software by refer to section [2.2 How to Build].

6.6.2 Change Build Options

BIP is build with a set of compiler and linker options details in section ***Build options*** of <ETAS BIP Release Notes.pdf>

User could open `[ProjectRoot]\settings.py` with a notepad, then change those build options by `COMPILER_FLAGS`, `ASSEMBLER_FLAGS` and `LINKER_FLAGS`.

> **NOTE**
> [$BIP_INTG 132] When specify build options, user shall pay special attention to not use parameters that are forbidden by RTA-OS in section ***Forbidden Options for Application Code*** of < RTA-OS Port Guide.pdf>

6.7 How to add source code to Build

- Add source code to BIP existing srouce file folders: directly drop source code and build.

  Info: This will automaticly build user source into executable.

- User create sub-folders under BIP source file folders: user need to put a **SConscript** file at the same level of the created sub-folder.

  Info: The SConscript file can be copied from any other folder and place at the same level of the created sub-folder.

- User create top level folders: not recommand.

6.8 Migrate to new Target

If user will migrate BIP to new target/MCAL, user shall change below target/MCAL dependent files:

Folder	Subfolder	Description	Purpose for user
Targets	Linker	Linker control file	**User shall** change linker file on project needs.
	MCAL**config**	MCAL configuration files	**User shall** configure MCAL on project needs.
	MCAL**gen**	MCAL configuration code	**User shall** generate MCAL configuration code into this folder.
	MCAL**modules**	MCAL static code	**User shall** replace MCAL static code with new target MCAL.
Integration	MCAL\Memmap	MCAL module memmap file	**User shall** replace MCAL memmap files with new target MCAL.
	MCAL\SchM	MCAL module SchM	**User shall** replace MCAL SchM files with new target MCAL.
	MCAL\api	MCAL Integration header file	**User shall** put their MCAL software integration header files.
	MCAL\src	MCAL Integration source file	**User shall** put their MCAL software integration source files.
	Target\api	Target integration header file	**User shall** put their target software integration header files.
	Target\src	Target integration source file	**User shall** put their target software integration source files.

Table 12 Migrate to new Target

Hint: User may also need to know how to make their source code to build by ref [6.7 How to add source code to Build]

6.9 Migrate to new Compiler

If user will migrate BIP build environment to new compiler, user shall change below compiler dependent files and scripts.

6.9.1 Compiler dependent Files

To migrate BIP build environment to new compiler, user shall change below compiler dependent files:

Folder	Subfolder	Description
BSW	src\integration\integration	Change compiler abstraction in `Compiler.h`
		Change memory and pointer classes in `Compiler_Cfg.h`
BSW	src\cobra\MemMap	Change `section` compiler abstractions in `MemMap.h`
Integration	Target\src	Change startup code
Target	Linker	Change the linker file syntax and options

Table 13 Compiler dependent Files

6.9.2 Compiler dependent Scripts

To migrate BIP build environment to new compiler, user shall change below compiler dependent scripts.

NOTE

The shipped `site_scons` scripts provides as example without maintenances.

When adapting to a compiler that is not yet included in the shipped `site_scons` tool, user shall create the scons scripts fit for intended compiler themselves.

Files	Parameters	Description
settings.py	`PROJ_NAME`	Set name of your project
	`COMPILER_CHOICE`	Select the compiler name, which has predefined within settings.py. e.g. set it to 'TASKING' if you use tasking compiler.
	`COMPILER_TOOL`	Name of the compiler which must match the file name located in the [ProjectRoot]\site_tools directory
	`COMPILER_ROOT`	Set to your compiler executable location
	`CPU_TARGET`	Set which CPU in this project you use
	`CPU_USED`	Set which CPU in this project you use
	`COMPILER_FLAGS`	Set attribute of compiler, it need to modify according to section 6.2 of the specific RTA-OS Port Guide that is supplied with the target of RTA-OS.
	`ASSEMBLER_FLAGS`	Set attribute of assembler, it need to modify according to section 6.2 of the specific RTA-OS Port Guide that is supplied with the target of RTA-OS.
	`LINKER_FLAGS`	Set attribute of linker, it need to modify according to section 6.2 of the specific RTA-OS Port Guide that is supplied with the target of RTA-OS.
	`LINKER_FILE`	Set the main linker file in this project
SConstruct	`[xxx]_path`	Specify which source code folder/sub-folder to build.
	`[xxx]_build_dir`	Specify build output directories for sub-systems.
	`objects +=`	Compile each subsystem's SConscript file and collect all objects.
[ProjectRoot]\site_scons\site_tools*.py	N.A.	Compiler scons scripts to build BIP.

Table 14 Compiler dependent Scripts

7 Compatibility

Cobra is bounded to the RTA-CAR version and BIP that is delivered to user.

It intend to boost user iteration by following integration guide, and it does not guarantee working on any RTA-CAR version.

Software	Version	Description
ISOLAR-AB	9.1.0	Used for SWC and RTE configuration and code generation for this release.
RTA-BSW	6.1.0	Used for BSW module configuration and code generation

Table 15 Cobra Compatible Tool

8 Rights of Use

The following definitions are used in the remainder of this section:

- **Software Tool:**

  ETAS Software Program, including AUTOSAR Basic Software Modules and code generators, which is running on a personal computer for designing, configuring, integrating AUTOSAR-compliant basic software and generating code.
- **AUTOSAR Basic Software Modules:**

  Add-ons from the ETAS RTA basic software collection that provide AUTOSAR middleware for advanced automotive ECUs.
- **Generated Code:**

  Embedded code (source code or library files) generated by the Software Tool code generator based on the configuration of AUTOSAR Basic Software Modules.
- **Generated Configuration:**

  Files containing configuration data generated by the Software Tool, typically used by further code generators and development tools.
- **Executable Image:**

  Executable binary image that is ready to run on a specific embedded system, generated through a respective compiler and linker from the Generated Code.

8.1 Development License for ETAS Standard Software

In the case that a development license has not been purchased, Basic Integration Package (BIP) shall not be entitled to copy, modify, distribute, transfer and sublicense the Generated Code or Executable Image for commercial production purposes.

8.2 Production License for ETAS Standard Software

In the case that a production license has not been purchased, Basic Integration Package (BIP) shall not be entitled to copy, modify, distribute, transfer and sublicense the Generated Code or Executable Image for commercial production purposes.

9 Contacting ETAS

ETAS HQ

ETAS GmbH
Borsigstraße 14
70469 Stuttgart
Germany

Phone: +49 711 3423-0
Fax: +49 711 3423-2106
WWW: www.etas.com

ETAS Automotive Technology (Shanghai) Co., Ltd.

333 Fuquan Road North, IBP
Changning District
Shanghai
China

Phone: +86 21 2218 5858
Fax: +86 21 5037 2221
WWW: www.etas.com

ETAS

Figures

Tables