

# MICROSAR Post-Build Loadable

## Technical Reference

Version 1.5.0

Authors	Hannes Haas
Status	Released

## Document Information

### History

Author	Date	Version	Remarks
Hannes Haas, Peter Hajsani, Jochen Vorreiter	2013-05-15	1.0.0	Creation
Hannes Haas	2013-05-24	1.0.1	ESCAN00067647: Review Feedback
Hannes Haas	2013-09-11	1.1.0	AR4-420: Module individual update
Hannes Haas	2014-06-13	1.2.0	AR4-642: Deleting containers at post-build time that had been available at link time: Important update of chapter 6.1.1 (Rules for BSW Changes at Post-Build Time) Minor overall modifications and improvements including notes on a possible runtime impact of post-build changes.
Hannes Haas	2014-10-08	1.3.0	AR4-698: Support of projects with post-build selectable (Identity Manager) and post-build loadable.
Hannes Haas	2014-11-28	1.3.1	Caution box added: Switching the configuration phase from Post-Build back to PreCompile.
Hannes Haas	2015-01-26	1.4.0	ESCAN00080589 changes: <ul style="list-style-type: none"> <li>&gt; Changed access to variant initialization structure in EcuM_DeterminePbConfiguration</li> <li>&gt; rename and split of EcuM_GlobalConfigRoot_Ptr</li> </ul> Global root structure customization now possible by configuring the ECUC module.
Hannes Haas	2015-08-07	1.4.1	Added chapter "System Extract Requirements".
Hannes Haas	2015-11-18	1.4.2	Added note on module configuration variant configuration Added chapter "Known Issues"
Hannes Haas	2016-05-10	1.5.0	Guideline for developing and integrating self-made BSW modules with the post-build loadable tool chain Information on SIP updates in post-build phase

## Reference Documents

No.	Source	Title	Version
[1]	Vector	TechnicalReference_PostBuildHexFileGenerator.pdf	as delivered
[2]	Vector	TechnicalReference_MemoryMapping.pdf	as delivered
[3]	Vector	TechnicalReference_PostBuildXmlGenerator.pdf	as delivered
[4]	Vector	TechnicalReference_EcuM.pdf	as delivered
[5]	Vector	TechnicalReference_BswM.pdf	as delivered
[6]	Vector	TechnicalReference_IdentityManager.pdf	as delivered

## Scope of the Document

This technical reference describes the general aspects of the MICROSAR Post-Build Loadable. The document does not describe BSW module specific functionality or limitations unless this is essential to understand the overall concept. Module specific details can be found in the documentation of each BSW module.



### Caution

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>7</b>
<b>2</b>	<b>Functional Description.....</b>	<b>8</b>
2.1	Features .....	8
2.2	Update Strategies .....	8
<b>3</b>	<b>Memory Management of Post-Build Data.....</b>	<b>9</b>
3.1	Memory Segments for Post-Build Loadable Data.....	9
3.2	Global Root Structure .....	10
<b>4</b>	<b>Post-Build Workflow.....</b>	<b>12</b>
4.1	Overview.....	12
<b>5</b>	<b>ECU Manufacturer Tasks.....</b>	<b>13</b>
5.1	Pre-compile and Link Time Parameter .....	13
5.2	Platform Settings.....	13
5.3	Memory Mapping .....	14
5.3.1	Mapping Post-Build Configuration Data.....	15
5.3.2	Mapping Post-Build Changeable RAM .....	15
5.4	Post-Build Loadable Memory Block Configuration.....	15
5.5	BSW Module Initialization .....	16
5.5.1	Overview .....	16
5.5.2	ECUM Global Root Configuration Structure .....	16
5.5.3	Implement Callout EcuM_DeterminePbConfiguration() .....	17
5.5.4	BSWM “Module Initialization” Auto Configuration .....	17
5.5.5	Manual BSW Initialization.....	17
5.5.6	Global Root Structure Customization .....	18
5.6	Test Post-Build Loadable Update .....	19
5.7	Switching Configuration Phase .....	19
<b>6</b>	<b>Post-Build Time Tasks.....</b>	<b>21</b>
6.1	Project Update during Post-Build Phase .....	22
6.1.1	Rules for BSW Changes at Post-Build Time.....	22
6.1.2	Post-Build Time handling of RTE.....	24
6.1.3	Modification of Start Addresses .....	24
6.2	Code Generation at Post-Build Time .....	24
6.3	Post-Build Loadable XML Generation .....	25
6.4	Check Validity of Post-Build Loadable Configuration .....	25
6.5	Hex File Conversion.....	26

6.6	Test and Verification .....	26
6.7	Measurement of BSW Variables and Constants.....	27
6.8	Updating SIPs and Tools during Post-Build Phase .....	27
6.8.1	SIP Update.....	27
6.8.2	DaVinci Tool Update.....	27
<b>7</b>	<b>Characteristics of Module Individual Update.....</b>	<b>29</b>
7.1	Introduction .....	29
7.2	Characteristics Overview .....	29
7.3	Memory Mapping .....	30
7.3.1	Mapping Post-Build Configuration Data.....	30
7.3.2	Mapping Post-Build Changeable RAM .....	30
7.4	Post-Build Loadable Memory Block Configuration.....	30
7.5	BSW Module Initialization .....	31
7.6	Post-Build Time Update .....	31
<b>8</b>	<b>Post-Build Loadable with MICROSAR Identity Manager .....</b>	<b>33</b>
8.1	Memory Mapping .....	33
8.2	Implement Callout EcuM_DeterminePbConfiguration() .....	33
<b>9</b>	<b>System Extract Requirements .....</b>	<b>34</b>
<b>10</b>	<b>Post-Build Loadable Requirements for Complex Drivers.....</b>	<b>35</b>
10.1	Memory Organization of Post-Build Loadable Data.....	35
10.2	Post-Build Loadable Data Structures .....	35
10.2.1	Not supported code patterns .....	36
10.3	Initialization Function .....	37
10.4	General Rules and Guidelines .....	38
10.5	Configuration and Code Creation.....	38
10.5.1	Guideline for BSWMD File creation.....	39
10.6	BSW Stack initialization .....	40
<b>11</b>	<b>Known Issues .....</b>	<b>42</b>
11.1	Inline assembler in Os.h cause PostBuildXmlGen to throw errors.....	42
<b>12</b>	<b>Glossary and Abbreviations .....</b>	<b>43</b>
12.1	Glossary .....	43
12.2	Abbreviations .....	43
<b>13</b>	<b>Contact.....</b>	<b>45</b>

	Illustrations	
Figure 3-1	Memory management of post-build loadable data .....	9
Figure 3-2	Post-Build data memory management of MICROSAR BSW modules using the global update process .....	11
Figure 4-1	Post-Build Loadable workflow overview .....	12
Figure 5-1	Configuration of post-build RAM and ROM block location.....	16
Figure 5-2	“Module Initialization” Auto Configuration Assistant of the DaVinci Configurator BSWM configuration.....	17
Figure 5-3	Switching Configuration Phase in DaVinci Configurator.....	19
Figure 6-1	Post-Build Time update of an ECU project.....	21
Figure 6-2	Indication of the currently activated configuration phase.....	22
Figure 6-3	The creation phase of containers illustrate the configuration phase the container has been created in.....	23
Figure 6-4	The effective configuration phase shows the latest phase that allows a parameter to be modified.....	24
Figure 7-1	Post-Build data memory management of one MICROSAR BSW module using the module individual update process .....	29
Figure 7-2	Create a new configuration for a module specific update.....	30
Figure 7-3	Module configuration structure configuration for module individual update .....	31
Figure 10-1	EPARM: Define that a BSW module is able to support post-build loadable.....	39
Figure 10-2	EPARM: Define that a container is able to be added and deleted at post- build time .....	40
Figure 10-3	EPARM: Define that a parameter can be modified at post-build time .....	40
Figure 10-4	Announcing a initialization function to ECUM and BSWM for module initialization .....	41

## Tables

Table 2-1	Supported AUTOSAR standard conform features .....	8
Table 2-2	Different post-build loadable update strategies .....	8
Table 5-1	Exemplary global root structure type definition as defined in EcuM_Init_PBCfg.h .....	17
Table 5-2	Exemplary global root structure as defined in EcuM_Init_PBCfg.c.....	17
Table 5-3	Manual BSW initialization using the configuration structure address provided by ECUM.....	18
Table 12-1	Glossary .....	43
Table 12-2	Abbreviations.....	44

## 1 Introduction

This documentation describes the process that is required to update the MICROSAR 4 BSW at post-build time using a flash download tool. In AUTOSAR context, this feature is commonly called post-build loadable.

MICROSAR post-build loadable allows updating defined aspects of the BSW configuration at post-build time without the need of a build environment or the application source code.

The document covers generic aspects common to all BSW modules supporting post-build loadable. It is essential to check for further module specific notes in the technical references of each BSW module.

Post-build loadable does not cover variant handling as defined by the post-build selectable. Variant handling is provided by the MICROSAR Identity Manager that is available as an option. If you want to combine post-build loadable and the MICROSAR Identity Manager in one project please consider the additional notes of chapter 8.

## 2 Functional Description

### 2.1 Features

The following features are supported:

Supported Features
Update of BSW module configuration at post-build time. As defined by AUTOSAR only parameter and containers that are marked as post-build changeable can be updated at post-build time. Post-build loadable is supported by MICROSAR BSW modules that support the Feature “ <b>Post-Build Loadable</b> ” (as it is listed in the BSW Technical Reference chapter “Functional Description → Features”). Please note that this feature requires explicit licensing. The scope of your post-build loadable license will define which BSW modules can undergo post-build loadable update.
ECU configuration update rules are enforced according to AUTOSAR 4.1.2
A module individual update is supported by MICROSAR BSW modules that support the Feature “ <b>Module individual post-build loadable update</b> ” (as it is listed in the BSW Technical Reference chapter “Functional Description → Features”).

Table 2-1 Supported AUTOSAR standard conform features

### 2.2 Update Strategies

MICROSAR post-build loadable supports two different update strategies.

Update Strategies
<b>Global Update:</b> Strategy that is typically applied for the communication stack and supported by all MICROSAR BSW modules. In short, all BSW modules share a single ROM and RAM post-build memory block. As a consequence all of these BSW modules have to be updated together. The advantage is that there is less memory fragmentation and managing of post-build data is simplified.
<b>Module Individual Update:</b> Only supported by MICROSAR BSW modules supporting “Module individual post-build loadable update”. This feature allows BSW modules to be updated individually and thereby not part of the shared RAM and ROM post-build memory block. Individual update can be used for e.g. OBD fault memory that shall be updated independent of the communication stack.

Table 2-2 Different post-build loadable update strategies

The update strategy has to be defined at link time. Both update strategies can coexist within one ECU.

This document first describes the global update process along with the general concept of the post-build loadable solution. Chapter 7 then highlights the characteristics of the module individual update in contrast to the global update process.



### 3 Memory Management of Post-Build Data

This chapter provides an overview of the memory management. A detailed description will be given in chapter 4.

#### 3.1 Memory Segments for Post-Build Loadable Data

All data that can be changed at post-build time is defined in the \*\_PBcfg.c files of the MICROSAR BSW modules that support post-build loadable.

To allow updating the data at post-build time, the post-build configuration data must be mapped to one ROM block ("post-build ROM block"). RAM tables that may change their size at post-build time must also be mapped to a dedicated RAM block in the ECUs memory ("post-build RAM block"). Definition and maintaining these blocks is the task of the integrator, who is in charge of the memory mapping.

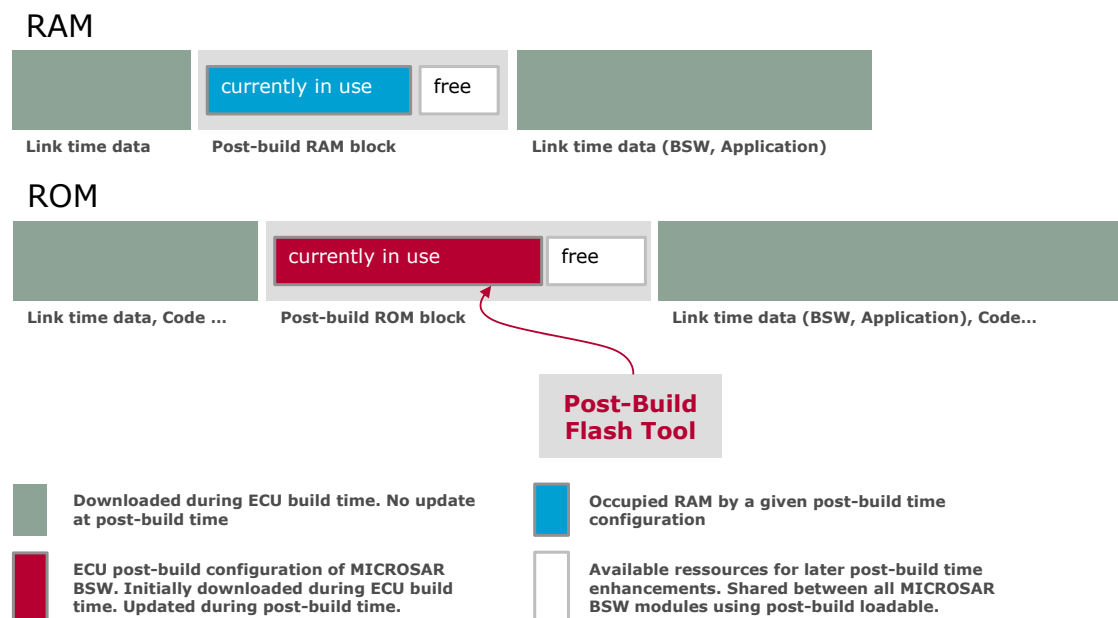


Figure 3-1 Memory management of post-build loadable data

The size of the post-build RAM and ROM block must be at least the size of the data required at link time but is typically larger to allow adding further functionality at post-build time.

The RAM and the ROM block have each one start address and an individual size. The start address for the RAM and the ROM block must be configured in the configuration tool as the addresses are used for address resolution at post-build time. The start addresses are typically fixed at link time and should not change at post-build time.

At link time the integrator maps the post-build data defined in the \*\_PBcfg.c files to the post-build RAM or ROM block (e.g. by using the AUTOSAR Memory Mapping as described in [2]). At post-build time the post-build tool chain maps the data automatically into the (same) post-build RAM and ROM block based on the configured RAM and ROM start address.

The BSW modules use the memory mapping segment SEC\_PBCFG for data that shall be mapped to post-build ROM (FLASH) block and SEC\_VAR\_PBCFG for data that need to be mapped to the post-build RAM block.

**Note**

For better readability this document uses a short notation (e.g. SEC\_PBCFG) to refer to memory sections. The BSW implementation will declare for each memory section BSW specific memory blocks using a start and a stop marker e.g. BSW\_START\_SEC\_PBCFG or BSW\_STOP\_SEC\_PBCFG.

**Edit**

The template based MemMap.h maps the BSW specific start/stop markers to BSW independent markers (e.g. STOP\_SEC\_PBCFG). The BSW independent markers need to be mapped to the linker specific mapping commands (e.g. #pragma) by the integrator. For further details, please see the MICROSAR Memory Mapping documentation [2].

The exact location of the data in the RAM and ROM block is not relevant as the linker and the post-build tool chain resolve the references.

### 3.2 Global Root Structure

In addition to the SEC\_PBCFG and SEC\_VAR\_PBCFG post-build data, one ECUM structure is assigned to the memory segment SEC\_PBCFG\_GLOBALROOT. This structure provides references to all BSW module configuration structures and is therefore called “Global Root Structure”. The structure has an important role in the BSW initialization and must be handled in a distinct way as described in this section.

Figure 3-2 illustrates the different memory mapping segments and their location in the post-build RAM and ROM block.

To keep the implementation simple, the address of the Global Root Structure must not be changed at post-build time. Consequently the linker and the post-build tool chain have to map this structure to the same location. This shall always be the beginning of the post-build ROM block. Configuration data of the SEC\_PBCFG segment shall be mapped directly behind the Global Root Structure to the post-build ROM block.

**Caution**

As a convention it has been defined, that the post-build loadable global root structure is located at the very beginning of the post-build ROM block.

As a consequence, the start address of the post-build ROM block is equal to the address of the Global Root Structure.

The address of the post-build loadable global root structure is provided to the ECUM by the API EcuM\_DeterminePbConfiguration() during initialization (see chapter 5.4). This

address will not change at post-build time as the start address of the post-build ROM block is fixed. Based on the addresses contained in the global root structure all other post-build loadable BSW modules will to be initialized in the following with pointers that may change at post-build time.

The post-build loadable global root structure shall be mapped to the beginning of the post-build ROM block by using the memory mapping segment SEC\_PBCFG\_GLOBALROOT. The start address of this segment shall be configured as Post-Build ROM Start Address in DaVinci Configurator. This will allow the post-build tool chain to map the post-build time data correctly.

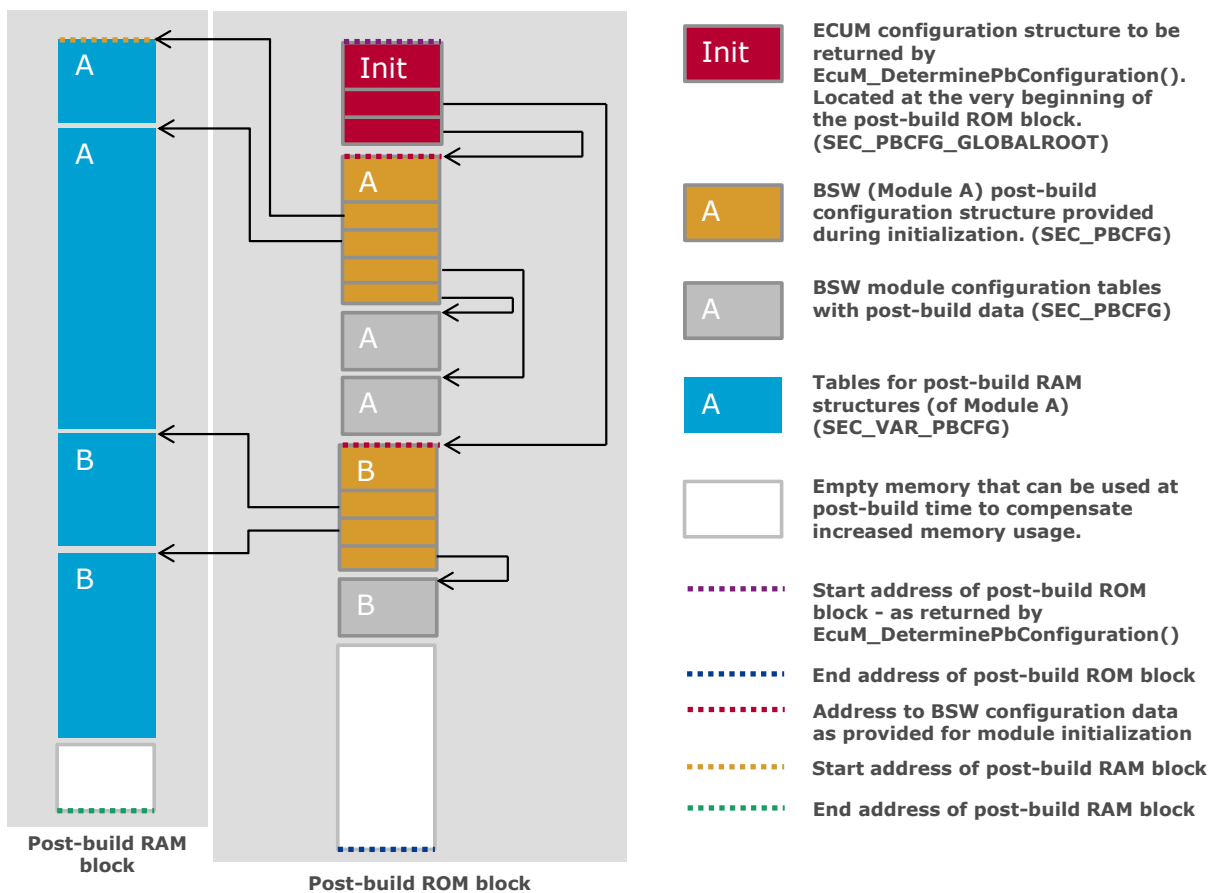


Figure 3-2 Post-Build data memory management of MICROSAR BSW modules using the global update process

## 4 Post-Build Workflow

This chapter describes the tasks to be performed at pre-compile and post-build time to allow a post-build loadable update of an ECU. Typically the pre-compile and link time task are performed by the ECU Manufacturer while the post-build configuration is performed by the OEM.

### 4.1 Overview

The ECU Manufacturer sets up the project and provides the configuration data – which is the DaVinci Project (.dpa) including its associated files – and the MICROSAR SIP to the OEM.

With this data, the OEM is able to perform post-build time updates to the project by generating directly a hex file without using a conventional compiler.

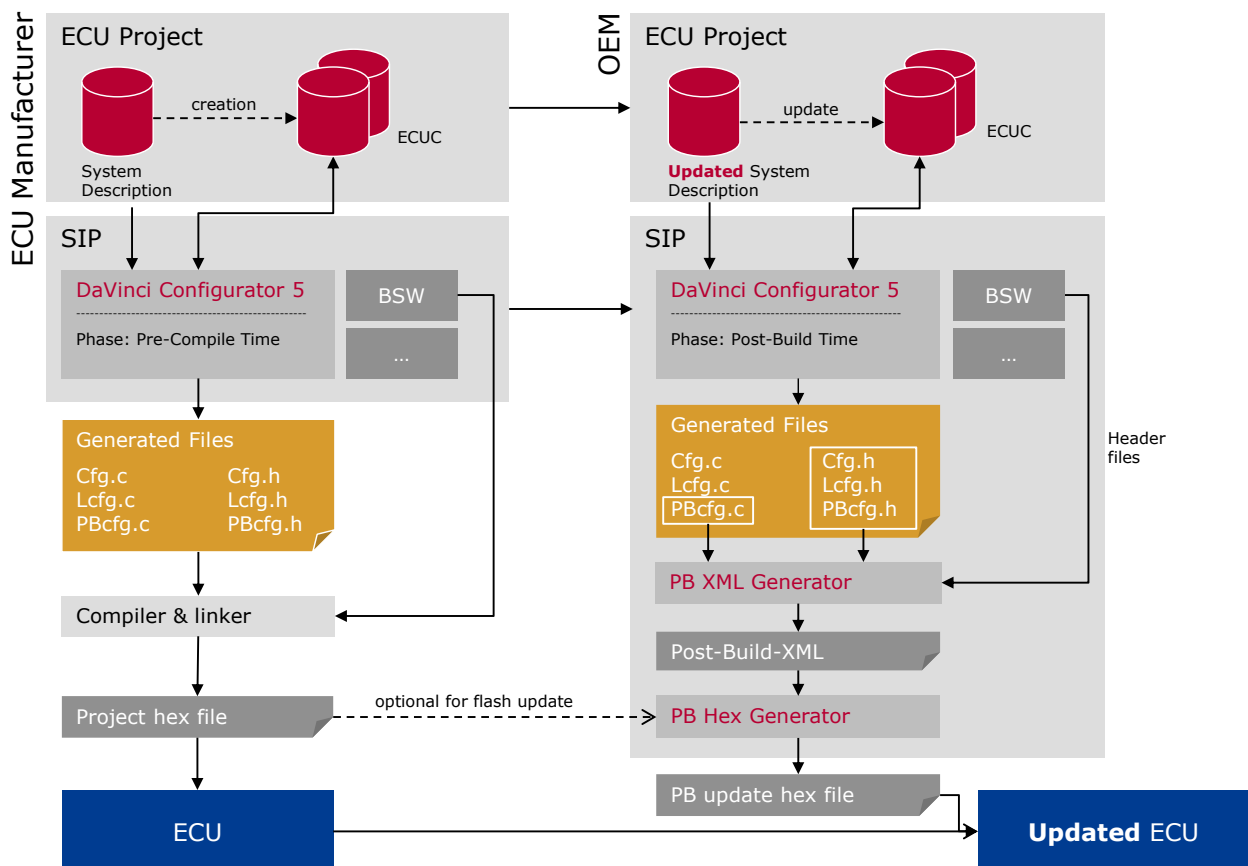


Figure 4-1 Post-Build Loadable workflow overview

## 5 ECU Manufacturer Tasks

Creating an ECU that supports post-build loadable configuration at post-build time requires additional steps and considerations at pre-compile and link time.



### Practical Procedure (overview)

1. Setup your MICROSAR project as usual
2. Set the configuration variant of BSW modules that shall support post-build loadable to POST-BUILD-LOADABLE (Settings Editor -> BSW Modules)
3. Ensure that pre-compile and link time parameter match with the OEM requirements that apply for post-build time (chapter 5.1)
4. Ensure correct configuration of the platform settings (Compiler properties...) (chapter 5.2)
5. Map all post-build data to the post-build RAM and ROM block (chapter 5.3)
6. Configure that location and size of the RAM and ROM block (chapter 5.4)
7. Generate with all MICROSAR code generators
8. Initialize BSW modules (chapter 5.5)
9. Compile, Link and Test the ECU (chapter 5.6)
10. Switch configuration phase to "Post-Build" (chapter 5.7)
11. Ship the MICROSAR SIP, the DPA Project and the ECU to OEM

### 5.1 Pre-compile and Link Time Parameter

Pre-compile parameter and link time parameter cannot be modified at post-build time. Consequently it will not be possible to enable functionality at post-build time that has been disabled in an earlier configuration phase.

This also applies for type definitions (e.g. PduIdType) and their relationship to the maximum number of objects the implementation can handle. These settings cannot be modified post-build time.

It is therefore essential to clarify the post-build time capabilities of the ECU and to configure the pre-compile and link time settings accordingly.

### 5.2 Platform Settings

The post-build time hex file creation process is able to generate the hex file without the need of a conventional compiler. As hex files have a platform and compiler specific layout these properties must be configured before hex files can be created using the post-build loadable tool chain.

As part of a MICROSAR delivery (SIP) supporting post-build loadable, Vector creates and verifies an initial version of the platform settings that define the compiler properties in an abstract manner. The platform settings are included as a recommended configuration to

the EcuC module and are based on the compiler version and compiler options you have indicated to us in the Questionnaire that is exchanged prior the delivery.

If the compiler or its options are changed within the projects these settings may have to be updated to match with the changed compiler behavior.

Due to the variety of compiler settings not all compiler options may be supported by the post-build loadable tool chain. If your compiler options are not compatible this will be communicated during SIP creation process.



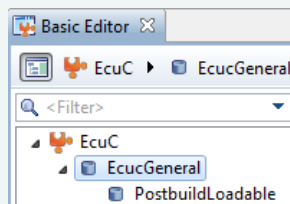
#### Caution

The correctness of the platform settings is crucial for a successful post-build time update. The settings depend on the platform, compiler and compiler options. The settings must be updated and verified each time changes are made to the build environment (such as changed compiler options or version).



#### Reference

The platform settings are configured in the Basic Editor View of DaVinci Configurator. The relevant parameters can be found in the container `EcuC/EcucGeneral` and `EcuC/EcucGeneral/PostbuildLoadable`.



## 5.3 Memory Mapping

The ECU manufacturer shall reserve one RAM and one ROM block for post-build data as introduced in chapter 2.2. The size of these blocks must be sufficient to allow the size of the configuration to increase at post-build time – e.g. because new messages or routing relations are added.

Using linker configuration the ECU manufacturer has to ensure that no other (none post-build) data is mapped to the post-build data RAM or ROM block.

The MICROSAR BSW modules provide AUTOSAR conform memory mapping segments that allow the mapping of RAM and ROM data to defined memory locations at link time. Please see [2] to find out how the memory mapping may be used on your platform.

The post-build RAM and ROM blocks specified at link time shall be reused at post-build time.

**Expert Knowledge**

The exact mapping of data structures within the post-build RAM and ROM block can change at post-build time as the hex file generation process uses a memory mapping strategy that may differ from the compiler. This is usually transparent for the BSW and the application as the structures are linked by pointers.

### 5.3.1 Mapping Post-Build Configuration Data

The MICROSAR BSW modules provide two different types of post-build loadable configuration data (ROM):

- > All data mapped by SEC\_PBCFG shall to be mapped to the post-build ROM block. The addresses are not critical but the data must be mapped behind the data mapped by SEC\_PBCFG\_GLOBALROOT.
- > Modules that support module individual update make use of the memory section SEC\_PBCFG\_ROOT. If the global update process shall be applied the data mapped by SEC\_PBCFG\_ROOT shall be mapped the same way as SEC\_PBCFG. For details on module individual update see chapter 7.
- > The global root structure assigned to SEC\_PBCFG\_GLOBALROOT must be mapped to the very beginning of the post-build ROM block. The address must be equal to the configured Post-Build ROM Start Address.

### 5.3.2 Mapping Post-Build Changeable RAM

Post-build RAM data structures of MICROSAR BSW modules can change their size at post-build time. This RAM data shall be mapped using the memory section SEC\_VAR\_PBCFG to the post-build RAM block. The addresses are not critical.

## 5.4 Post-Build Loadable Memory Block Configuration

As described in chapter 2.2, all post-build RAM and ROM data is mapped to a linear memory area with one start and end address for post-build RAM data and one start and end address for the post-build configuration (ROM) data.

**Caution**

The configuration of the start and end addresses of the post-build RAM and ROM blocks is critical for a successful post-build time update of the ECU.

The configured ROM start address must be equal to the address of the Global Root Structure. The RAM start address must point to an otherwise unused memory area of sufficient size to contain all post-build RAM data.

Based on the configured end addresses, the post-build tool chain performs a consistency check that ensures that the post-build data will not overwrite any other data.



### Reference

The start and stop address of the post-build RAM and ROM block are configured in the Basic Editor View of DaVinci Configurator. The parameters can be found in the container EcuC/EcuGeneral /PostbuildLoadable.

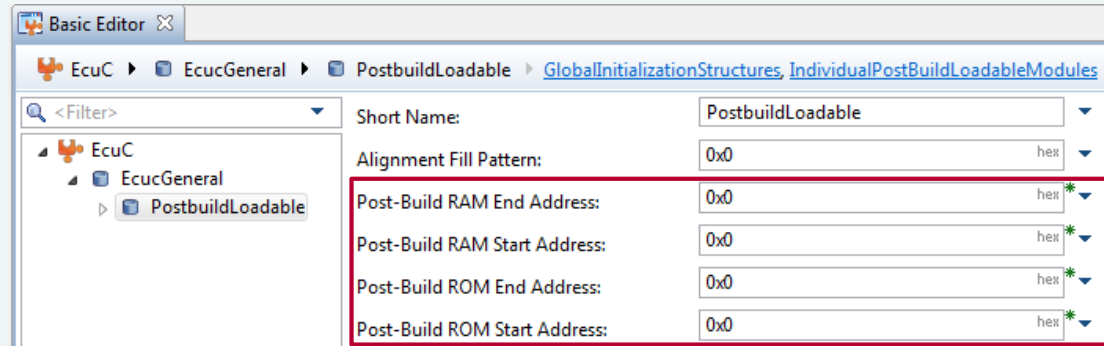


Figure 5-1 Configuration of post-build RAM and ROM block location

## 5.5 BSW Module Initialization

This section summarizes the steps required to initialize post-build loadable BSW modules using the global update process (see chapter 2.2 (Update Strategies)).



### Reference

Please read the Technical References of ECUM ([4]) and BSWM [5] carefully to find out how the BSW initialization needs to be implemented in general terms.

### 5.5.1 Overview

At post-build time the initialization pointer provided to the BSW module initialization function may change. In order to make the address of the BSW modules configuration structure accessible, the ECUM module provides a list with the addresses. This so called global root structure has a static address and is always located at the beginning of the post-build ROM block. The provided addresses pointing to the BSW module configuration structures are updated automatically at post-build time.

### 5.5.2 ECUM Global Root Configuration Structure

The `EcuM_GlobalPBConfigType` must provide one structure element for each BSW module that shall use the global update mechanism. MICROSAR modules add their details automatically through an internal service of DaVinci Configurator. If an additional element shall be added, follow the description in chapter 5.5.6.

```
typedef struct
{
    CONSTP2CONST(BswM_ConfigType, TYPEDEF, BSWM_PBCFG) CfgPtr_BswM_Init;
    CONSTP2CONST(EcuM_PbConfigType, TYPEDEF, ECUM_PBCFG) CfgPtr_EcuM_Init;
    ...
} EcuM_GlobalPBConfigType;
```



Table 5-1 Exemplary global root structure type definition as defined in EcuM\_Init\_PBcfg.h

The global root structure consequently looks as follows:

```
CONST(EcuM_GlobalPBConfigType, ECUM_PBCFG) EcuM_GlobalConfigRoot =  
{  
    BswM_Config_Ptr,  
    EcuM_Config_Ptr,  
    ...  
};
```

Table 5-2 Exemplary global root structure as defined in EcuM\_Init\_PBcfg.c

### 5.5.3 Implement Callout EcuM\_DeterminePbConfiguration()

During initialization the ECUM invokes the callout `EcuM_DeterminePbConfiguration()` to determine the address of the global root structure. In non-variant projects the callout shall return `&EcuM_GlobalConfigRoot.GlobalConfiguraton` published by `EcuM_Init_PBcfg.h`.



#### Edit

A sample implementation of this function is provided by `EcuM_Callout_Stubs.c` as a template. Adapt this implementation if required.

### 5.5.4 BSWM “Module Initialization” Auto Configuration

The majority of the BSW modules are initialized by the BSWM. To realize the BSW module initialization use the Module Initialization Auto Configuration wizard provided by the BSW Management editor of DaVinci Configurator. This wizard will realize the BSW initialization as required by post-build loadable using the global root structure as source for the configuration pointers. Find more details in the BSW technical reference [5].

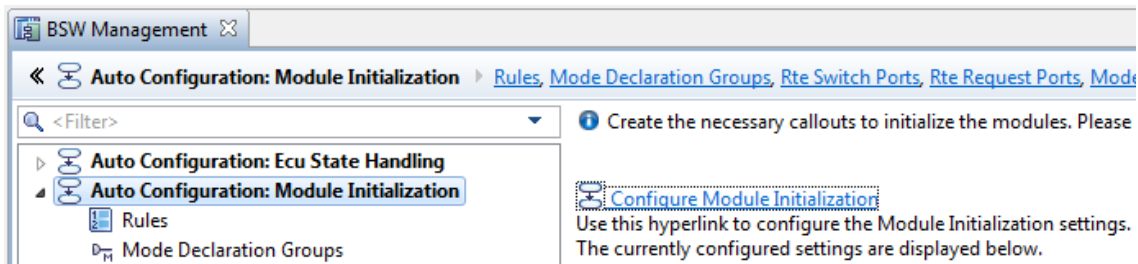


Figure 5-2 “Module Initialization” Auto Configuration Assistant of the DaVinci Configurator BSWM configuration

### 5.5.5 Manual BSW Initialization

If for some reason the initialization through the BSWM “Module Initialization” Auto Configuration assistant is not possible the BSW initialization can also be implemented manually.

For post-build loadable modules the header `EcuM_Init_PBcfg.h` is used to publish initialization relevant data types. To access the module configuration pointers always use `EcuM_GlobalPBConfig_Ptr`.

```
#include "EcuM_Init_PBcfg.h"  
...  
void MyInitFunction()
```

```
{
...
Com_Init(EcuM_GlobalPBConfig_Ptr->CfgPtr_Com_Init);
ComM_Init(EcuM_GlobalPBConfig_Ptr->CfgPtr_ComM_Init);
...
}
```

Table 5-3 Manual BSW initialization using the configuration structure address provided by ECUM

The `EcuM_GlobalPBConfig_Ptr` is initialized by the ECUM during `EcuM_Init()` with the global root structure address returned by the callout `EcuM_DeterminePbConfiguration()`.



#### Caution

The initialization pointers provided by the global root structure can change at post-build time. The implementation must avoid that the compiler replaces the structure access to the configuration pointers with hard coded addresses at link time.

This is done by accessing the module configuration pointer addresses through a RAM pointer (`EcuM_GlobalPBConfig_Ptr`) pointing to the global root structure.

#### Negative Example:

Never do this:

```
Com_Init(EcuM_GlobalConfigRoot.GlobalConfiguration.CfgPtr_Com_Init);
```



#### Expert Knowledge

The pointer to the initialization structure of BSW modules that support post-build loadable (and also the combination of post-build loadable and post-build selectable) is available through the structure `EcuM_GlobalPBConfig_Ptr` (`EcuM_Init_PBcfg.h`).

The pointer to the initialization structure of BSW modules that do not support post-build loadable but e.g. post-build selectable is available through the structure `EcuM_GlobalPCConfig_Ptr` (`EcuM_Init_Cfg.h`).

Modules that support neither post-build loadable nor post-build selectable can be initialized without deriving a special pointer from the ECUM.

## 5.5.6 Global Root Structure Customization

If an additional configuration pointer shall be added to the `EcuM_GlobalPBConfigRoot` structure you can add the initialization details in the ECUC module configuration using DaVinci Configurator: `/MICROSAR/EcuC/EcucGeneral/BswInitialization/InitFunction`.

**Caution**

Within DaVinci Configurator only BSW modules that support the MICROSAR post-build loadable process must be configured as Implementation Variant VARIANT-POST-BUILD-LOADABLE or VARIANT-POST-BUILD. As a consequence these modules are added to the post-build loadable global root structure `EcuM_GlobalPBConfigRoot`.

Variant modules that do not support post-build loadable will be added to `EcuM_GlobalPCConfigRoot` instead.

## 5.6 Test Post-Build Loadable Update

Before committing the ECU project to the OEM it is recommended that the post-build update process is tested by the Tier1. The goal is to ensure that the compiler settings are configured correctly and are that the chosen settings compatible with the post-build tool chain.

Generate the hex file as it is described in chapter 6 with the same configuration as it was used for the normal compile run. Flash the generated hex file to the ECU, keeping in mind that only the post-build data is exchanged in this step. The BSW implementation and link-time configuration remains unchanged.

Execute the ECU test once more to ensure that the ECU configuration has been updated correctly.

## 5.7 Switching Configuration Phase

After the final build of the ECU, the configuration phase must be switched from pre-compile configuration phase to the Post-Build configuration phase. This is required in order to avoid the OEM to modify any parameters that must not be modified at post-build time.

Before switching the configuration phase it is recommended to store the final development version under version management. This version can be used for a later development step of the ECU.

Make sure that all module generators have generated one more time without errors before switching the configuration phase to Post-Build.

When switching the configuration phase all pre-compile and link time settings of the ECU project will be frozen. From this point on, only changes that are allowed at post-build time will be allowed to be made.

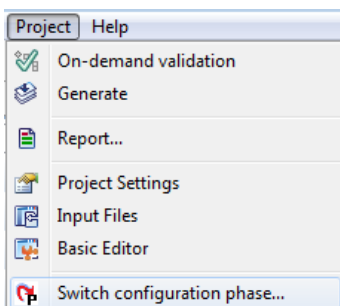


Figure 5-3 Switching Configuration Phase in DaVinci Configurator

The currently active configuration phase is illustrated in the status bar at the bottom of DaVinci Configurator as illustrated in Figure 6-2.

**Caution**

Switching the configuration phase shall be done by the ECU manufacturer before handing over the project to the OEM.

**Caution**

Post-Build compatibility is only granted as long as the configuration phase is not switched back to PreCompile phase.

Switching the configuration phase from Post-Build phase to PreCompile phase is possible but will cause the new Pre-Compile configuration to be no longer compatible with the original Pre-compile configuration. Thus a later Post-Build update of an ECU that implements the original PreCompile configuration will fail.

Exampe:

- 1) PreCompiler configuration (Config1)
- 2) Build ECU (EcuWithConfig1)
- 3) Switch to Post-Build phase (Config1')
- 4) Switch back to PreCompile phase (Config2)
- 5) Switch to Post-Build phase (Config2')
- 6) Update EcuWithConfig1 using Config2' → **Error!**

To avoid this problem the new PreCompile phase configuration (Config2) has to be downloaded to the ECU before updating with Config2'

## 6 Post-Build Time Tasks

Updating a configuration at post-build time involves several steps which require to be executed with great care and in exact order as defined in this chapter.

Figure 6-1 illustrates the main steps to be performed when updating the BSW configuration at post-build time. The figure also indicates the main artifacts involved by each step.

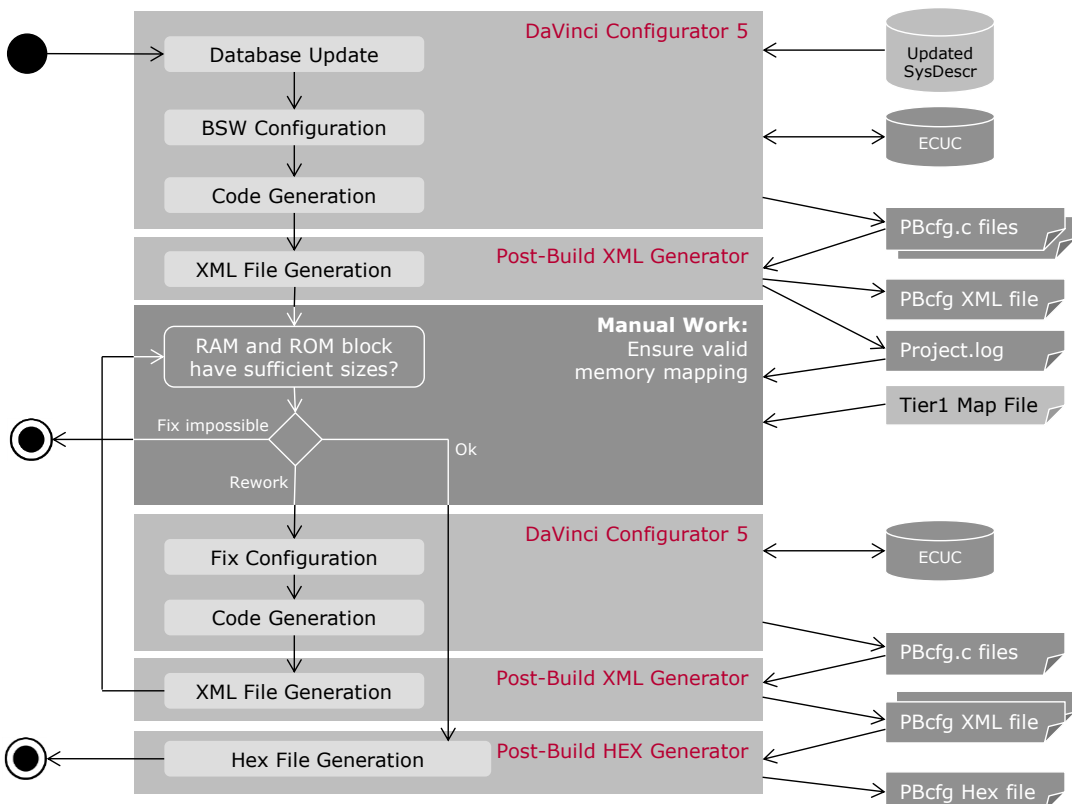


Figure 6-1 Post-Build Time update of an ECU project



### Practical Procedure (overview)

1. Update Project
2. Generate Code for all MICROSAR modules
3. Convert MICROSAR \*\_PBcfg.c files into a post-build loadable XML file
4. Check memory consumption of updated configuration
5. Convert the post-build loadable XML file to hex file
6. Download hex file to ECU
7. Test ECU

## 6.1 Project Update during Post-Build Phase

Before updating the ECU configuration at post-build time, ensure that the configuration phase configured in DaVinci Configurator is set to “PostBuild”. The currently active phase is illustrated in the status bar at the bottom of DaVinci Configurator.

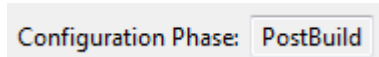


Figure 6-2 Indication of the currently activated configuration phase

Switching the configuration phase to post-build shall be done by the ECU manufacturer once the link time phase is completed and must never be set back for this ECU build.

Now the project can be updated either manually or by using the system description update functionality. Based on the knowledge of the configuration phase, DaVinci Configurator will only allow changes to the data model that are allowed at post-build time. All other changes will be rejected. The Property View of DaVinci Configurator illustrates if and why changes are not allowed to a parameter.

### 6.1.1 Rules for BSW Changes at Post-Build Time

In this chapter the rules for possible changes at post-build time are summarized for a better understanding. The DaVinci Configurator enforces these rules and allows only changes that are allowed by the currently active configuration phase.

- > Post-build time changes to a module configuration are only possible if the implementation configuration class of the BSW module is set to VARIANT-POST-BUILD-LOADABLE. This property overwrites potential settings for parameter and container of that module.
- > Adding a new container (e.g. a ComSignal) to the ECU configuration at post-build time is only possible for containers with the property 'PostBuildChangeable' set to true in their definition.
- > Containers that have been added at post-build time can also be removed at post-build time if these containers are no longer used (i.e. referenced).
- > Containers that have been created at link or pre-compile time can be deleted at post-build time if the definition does not prohibit this using the flag “PostBuildNotDeletable”. It is also mandated that containers that are deleted are no longer referenced by any other object in the ECU configuration. As deleted containers will be removed completely from the ECU configuration the basic software will no longer know these objects. As a consequence e.g. the handle IDs or symbolic names of the deleted objects are no longer valid. Handle IDs may even be reused by other objects added at post-build time. Thus, objects deleted or renamed at post-build time must not be used by the application or by the basic software! Please consider the following warning:

**Caution**

Deleting as well as renaming containers (that have been created at pre-compile or link time) at post-build time is not recommended due to the difficulty of detecting faults introduced with this change to the ECU configuration. If a container that has been deleted at post-build time is still used by some ECU code, unpredictable ECU behavior may occur.

If despite this warning containers (created at pre-compile or link time) are deleted at post-build time great care is required as some link time code may use these objects. Neither the MICROSAR BSW nor DaVinci Configurator are able to detect the usage of objects that have been deleted from the configuration at post-build time if there is no formal reference configured in the ECU configuration.

Accessing objects (such as PDUs) that are no longer part of the ECU configuration will cause unpredictable ECU runtime behavior. There are no formal checks e.g. as provided by a compiler at link time that may detect a corrupted configurations at post-build time!

Important note: This warning applies also for database updates executed at post-build time where containers are removed or renamed due to changes in the input databases. DaVinci Configurator will perform the configuration update according to the rules above and will display a warning for each container that has been removed but was present in the link time phase.

It is the responsibility of the post-build time system integrator to ensure that the objects that have been derived from the deleted containers are not used in any way from the ECU software (this includes application and basic software).

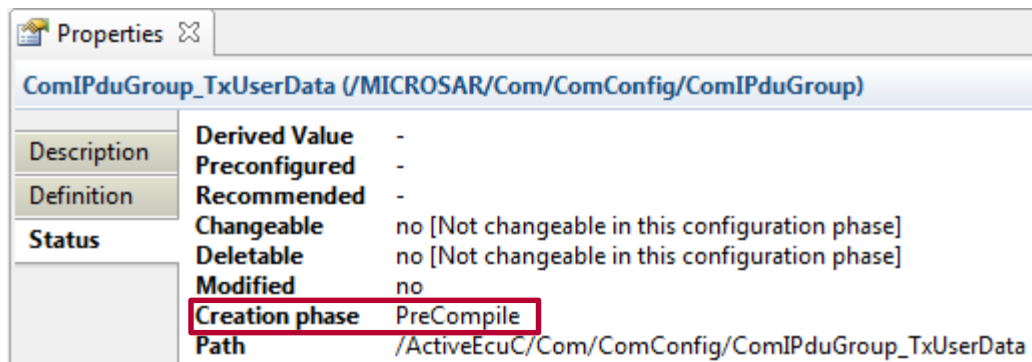


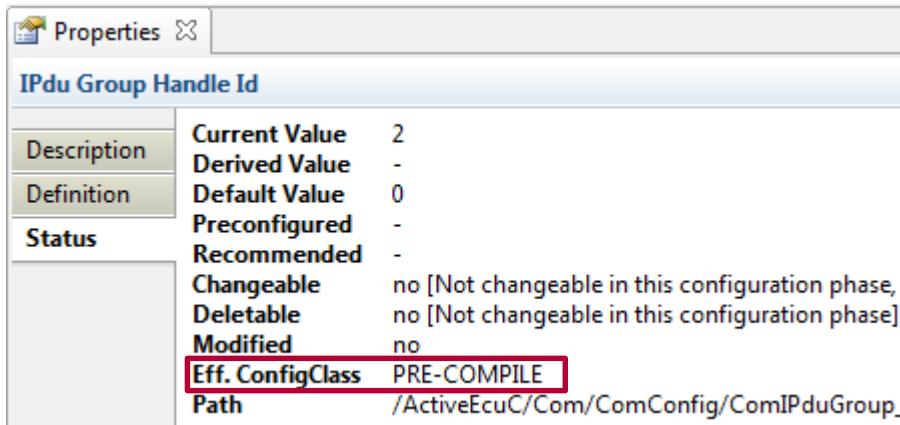
Figure 6-3 The creation phase of containers illustrate the configuration phase the container has been created in

**Example**

An example of containers that cannot be deleted at post-build time is the COM signal interface that provides an important interface to the application. As a consequence COM signals can only be deleted at post-build time if that signal has been added in an earlier post-build time update.

- > Only parameter of the variant 'VariantPostBuildLoadable' can be changed at post-build time. A change is also possible if the parent-container is not post-build changeable.

- > If a parent container has been created at post-build time it is possible to specify all parameters within that container, independent of the configuration variant.



IPdu Group Handle Id		
Description	Current Value	2
	Derived Value	-
Definition	Default Value	0
Status	Preconfigured	-
	Recommended	-
	Changeable	no [Not changeable in this configuration phase,
	Deletable	no [Not changeable in this configuration phase]
	Modified	no
	Eff. ConfigClass	PRE-COMPILE
	Path	/ActiveEcuC/Com/ComConfig/ComIPduGroup_

Figure 6-4 The effective configuration phase shows the latest phase that allows a parameter to be modified.

### 6.1.2 Post-Build Time handling of RTE

The RTE is not post-build capable. Consequently the software component design elements are not changeable in DaVinci Configurator and the RTE generation step will not be executed at post-build time.

### 6.1.3 Modification of Start Addresses

The RAM and ROM start and end addresses are post-build time changeable but must not be modified at post-build time unless this is explicitly allowed and supported by the ECU manufacturer. If an unintentionally modification shall be avoided the values may be pre-configured using the AUTOSAR pre-configuration mechanism.

## 6.2 Code Generation at Post-Build Time

Code generation at post-build time is triggered as in any other configuration phase using the DaVinci Configurator code generation functionality.



#### Caution

The MICROSAR post-build loadable concept requires a post-build update of all modules supporting post-build loadable. This is independent of the changes that have been made. When generating, it is therefore important to enable all module generators in the generation dialogue.

At post-build time the code generator will produce all output files including the pre-compile and link time configuration files. For further processing only the header files (for type definitions) and \*\_PBcfg.c files are of relevance.

Based on the generated \*\_PBcfg.c files a hex file is created that can be downloaded to the ECU. The hex file generation is performed in two steps, each performed by a dedicated command-line tool.



### 6.3 Post-Build Loadable XML Generation

In a first step the \*\_PBcfg.c files are converted into a post-build loadable XML file. In a second step (chapter 6.5) the XML description is converted to the final hex file.



#### Caution

Before creating a post-build loadable XML file, all MICROSAR code generators must have generated the updated configuration successfully. This proceeding is independent of the change performed.

The newly generated \*\_PBcfg.c files are converted to a post-build loadable XML file. The required command line tool for this task is described in [3].

Provide the \*\_PBcfg.c files of all modules that are globally updated as source files. The global root structure defined in EcuM\_Init\_PBcfg.c (see chapter 5.4) must also be provided as source file.



#### FAQ

**Do I have to provide all \*\_PBcfg.c files as input files?**

All MICROSAR modules using the global update mechanism have to be updated together even if there are no changes expected.

The post-build loadable XML file created by the Post-Build XML Generator is an abstract description of the later Hex file. It already contains the binary data that is assigned to addresses in the ECUs memory.



#### Example

A typical call to the command line to create the post-build XML looks as follows:

```
PostBuildXmlGen.exe --source="<GenDataFolder>\<Msn> PBcfg.c" --source="<GenDataFolder>\EcuM Init PBcfg.c" --project="<ProjectPath>\Project.dpa" --include-recursive="..\..\BSW" --include-recursive="<BswGenData>" --include-recursive="<ProjectPath>\Appl\Include"
```

The PostBuildXmlGen.exe tool is located in the SIP folder .\Generators\PostBuildXmlGen. This example will produce the output XML to <GenDataFolder>\PostBuild\Project.xml



#### Basic Knowledge

You can use the DaVinci Configurator “External Generation Steps” to automatize the copy process and the hex file generation process in an easy way.

### 6.4 Check Validity of Post-Build Loadable Configuration

Changes to the configuration during the post-build phase can cause increased RAM and ROM usage. To avoid malfunction of the ECU it is essential to ensure that the updated configuration data fits into the post-build RAM and ROM block reserved at link time.

Based on the start and end addresses that have been configured by the ECU Manufacturer (see chapter 5.4), the Post-Build XML Generator performs a consistency check and issues an error in case memory usage exceeds the specified memory blocks.

If the check fails the post-build configuration occupies RAM and ROM outside the post-build RAM and ROM block. In this case the configuration must not be applied to the ECU as it would destroy the ECU functionality by overwriting other data.

## 6.5 Hex File Conversion

Based on the post-build XML files, hex files are generated using the tool `PostBuildHexFileGenerator.exe` that is part of the SIP. Please check the additional documentation [1] for details on the usage of that tool.



### Example

A typical call to the command line call to create the Hex File looks as follows:

```
PostBuildHexFileGenerator.exe -o <PbFilesFolder>\PostBuild\Project.hex  
<GenDataFolder>\PostBuild\Project.xml
```

The `PostBuildHexFileGenerator.exe` tool is located in the SIP folder `.\Generators\PostBuildHexFileGenerator`.



### Note

Alternatively to the command line tool `PostBuildHexFileGenerator`, the Vector HexView tool can be used to create the hex files based on the post-build XML input.

## 6.6 Test and Verification

Updating an ECU at post-build time introduces new functionality and may change the implementation of existing functionality with respect to the used post-build configuration data.

Post-build time modifications can also alter the runtime behavior of the BSW e.g. by adding more messages and signals to the ECU. But also changes to a parameter can cause increased runtime e.g. when reducing the Tx cycle time of a message.

Extensive test and verification of the updated configuration is therefore obligatory.

Any post-build update of the ECUs configuration causes an update of all post-build capable parameters of all MICROSAR modules supporting post-build (i.e. all MICROSAR modules generating a `*_PBcfg.c` file). This effect is independent of the originally intended scope of the change and must be considered during ECU testing.

**Caution**

In order to ensure ECU functionality after a post-build update extensive tests of the complete ECU functionality are required independent of the scope of the change.

## 6.7 Measurement of BSW Variables and Constants

This section is only relevant if external tools access RAM or ROM data from MICROSAR BSW modules that can be updated post-build time. This is e.g. the case for MICROSAR DBG (Debugging) which is part of MICROSAR AMD. All other (none post-build updated) symbols are not affected by the change described below!

The post-build generation process relocates post-build RAM and ROM data within the assigned memory blocks. The relocation causes the addresses which are documented in the link time MAP or a2l file to be no longer valid.

If calibration tools require access to post-build RAM and ROM data the addresses of the symbols (e.g. in the a2l file) have to be updated after each post-build generation step. For this purpose the Post-Build XML Generator produces an MAP file indicating the new addresses of all symbols that are updated at post-build time (see [3]). This MAP file can be used to update the existing a2l file(s) with the new addresses by using e.g. the ASAP2 Updater from Vector.

## 6.8 Updating SIPs and Tools during Post-Build Phase

### 6.8.1 SIP Update

A SIP update typically causes inconsistencies between the already linked and downloaded static BSW implementation and the generated post-build data that is generated with the updated generators. Such an update is therefore only possible if only minimum (compatible) changes (i.e. fix of issues) have been made.

**Caution**

Once a project has entered post-build phase the SIP must no longer be updated.

If a BSW generation issue blocks a post-build time update inform Vector on the configuration phase of the project. This information will allow Vector to work on a solution that considers the existing BSW implementation.

### 6.8.2 DaVinci Tool Update

Updating DaVinci Developer is usually not required as this tool plays no significant role at post-build time. A service pack update is however, possible.

Updates of DaVinci Configurator Pro 5 shall not take place at post-build time as the update may introduce changes such as a modified interpretation of input files. This can cause problems when performing a post-build time database update.

**Caution**

Once a project has entered post-build phase the DaVinci Configurator Pro 5 must no longer be updated using service packs.

If a tool issue blocks a post-build time update inform Vector on the configuration phase of the project. This information will allow Vector to work on a solution that considers the existing BSW implementation.

## 7 Characteristics of Module Individual Update

### 7.1 Introduction



#### Note

Please make sure that you have read and understood the previous chapters on global update before start reading the characteristics of the module individual update. This is important as the general approach is similar. This chapter will only highlight the differences.

Module individual update allows you to realize an ECU that allows post-build update of a single BSW module without needing to update other BSW modules. As a consequence such modules cannot be updated or changed with the global update.

### 7.2 Characteristics Overview

Short overview over the main differences of module individual update compared to the global update:

- > Each module that is updated individually use its dedicated post-build RAM and ROM block.
- > The global root structure that refers to the module specific configuration structures is not used for these modules (i.e. does not refer to modules that are updated individually).
- > The module configuration structure has a fixed address located at the very beginning of the module post-build ROM block.

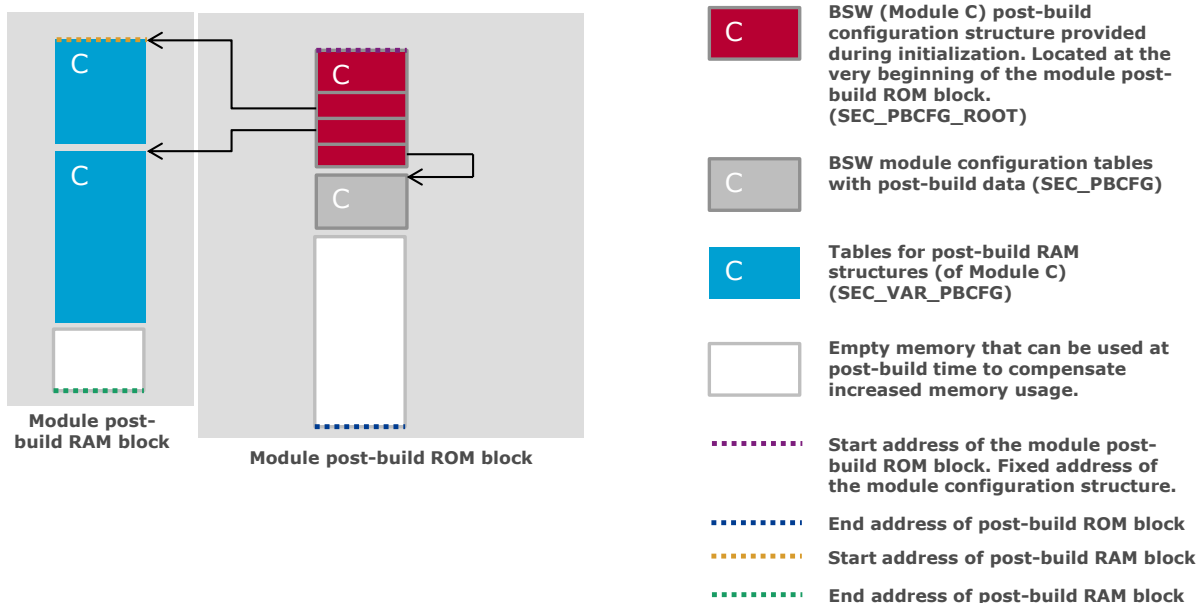


Figure 7-1 Post-Build data memory management of one MICROSAR BSW module using the module individual update process

## 7.3 Memory Mapping

### 7.3.1 Mapping Post-Build Configuration Data

Modules that use module individual update use the memory section SEC\_PBCFG\_ROOT to allow dedicated mapping of the module specific root structure. The module specific configuration structure shall be mapped to the very beginning of the post-build ROM block that has been assigned to that module.

All module configuration data that is mapped by SEC\_PBCFG shall be mapped to the module post-build ROM block behind the data mapped by SEC\_PBCFG\_ROOT.

### 7.3.2 Mapping Post-Build Changeable RAM

All post-build RAM data of one module shall be mapped to the post-build RAM block of that module by using the memory mapping section SEC\_VAR\_PBCFG.

## 7.4 Post-Build Loadable Memory Block Configuration

For each module that uses individual update a dedicated post-build RAM and post-build ROM block need to be defined. This is done by creating a new “IndividualPostBuildLoadableModule” container within the EcuC module.

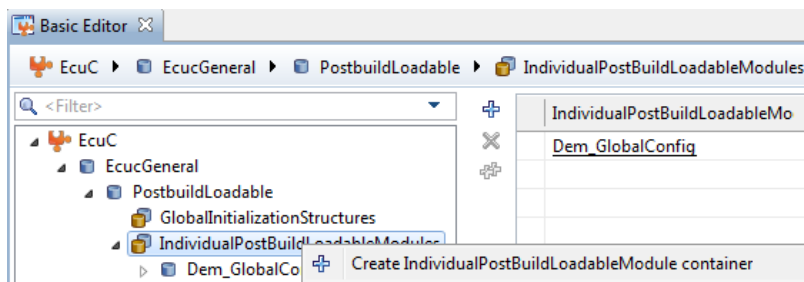


Figure 7-2 Create a new configuration for a module specific update

Rename the container to e.g. the name of BSW module. Configure that start and stop address for the post-build RAM and ROM block for the BSW module. Take care that the memory block does not overlay with any other memory block (e.g. the globally used memory block).

To create the mapping with the module configuration structure, create a sub container “ModuleInitializationStructure” (see Figure 7-3).

Rename the container to the name of the module configuration structure. The configuration structure is generated to <Msn>\_Pbcfg.c and will be within the memory mapping section SEC\_PBCFG\_ROOT. The name is typically <Msn>\_GlobalConfig.

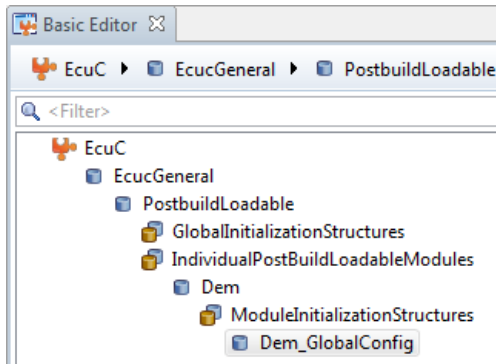


Figure 7-3 Module configuration structure configuration for module individual update

## 7.5 BSW Module Initialization

Ensure that the module configuration structures of modules that are initialized individually are not part of the ECUM global initializations structure (EcuM\_GlobalConfigType). Thus remove these modules from the generated template as described in chapter 5.5.6.

Ensure that the BSWM “Module Initialization” Auto Configuration assistant uses address of your module configuration structure directly without reading this address from the ECUM global root structure where it no longer is defined.

Example: `Dem_Init(&Dem_GlobalConfig)`

As the address handed over during BSW module initialization is fixed at link time, the location of the module configuration structure must not be changed at post-build time.

## 7.6 Post-Build Time Update

For each module that is updated individually the post-build loadable process is executed individually resulting in one hex file for each module. These hex files can be updated independently from the other module configurations or the global update process used by other modules.

To create the module specific hex files for modules that are updated individually follow the steps provided in

- > Chapter 6.2 (Code Generation)
- > Chapter 6.3 (Post-Build Loadable XML Generation): provide the module `<Msn>_PBcfg.c` file as sole “source” file for this update process. The output XML and MAP files will be named according to the module name configured within the ECUC modules `IndividualPostBuildLoadableModule` container. Example:  
`<GenDataFolder>\PostBuild\Dem.xml`
- > Chapter 6.4 (Check Validity of Post-Build Loadable Configuration)
- > Chapter 6.5 (Hex File Conversion)

After the update the overall ECU functionality has to be ensured as described in chapter 6.6 (Test and Verification).

**Caution**

When updating a module individually the user must ensure that the module configuration still matches with the configuration of the other BSW modules that have not been updated.



## 8 Post-Build Loadable with MICROSAR Identity Manager

The same way it is possible to perform a post-build loadable update of a non-variant ECU project it is also possible to update a variant project.

This chapter lists the major differences with respect to post-build loadable when using a multi variant configuration. A general description of the MICROSAR Identity Manager can be found in the technical reference of that feature [6].

### 8.1 Memory Mapping

As there is one RAM and one ROM block for all variants, memory mapping is similar to a non-variant configuration. Also the mapping of `EcuM_GlobalConfigRoot` using `SEC_PBCFG_GLOBALROOT` remains unchanged as this structure now contains one structure element for each variant.

### 8.2 Implement Callout `EcuM_DeterminePbConfiguration()`

The callout `EcuM_DeterminePbConfiguration()` has to return the global root initialization structure address of the variant that shall be used. For variant projects the structure provides one member for each variant. To determine the address use e.g. the following code: `&EcuM_GlobalConfigRoot.<Variant>`.

The symbol is published by `EcuM_Init_PBcfg.h`.

## 9 System Extract Requirements

The post-build loadable update process mandates that specific elements do not change in their name during a post-build time update. As a result, rules that decorate e.g. a signal name using the name of the AUTOSAR ARXML package (e.g. message or channel name) are deactivated for post-build loadable projects.

As a result ARXML system extracts must follow the following rules in order to be compatible with the DaVinci tooling and MICROSAR BSW.

The term unique in the following list requires an element name to be unique within the complete system extract (including all ARXML packages).

### **ComIPdu**

- > Pdu.shortName has to be unique
- > The Pdu is referenced by only one PduToFrameMapping in a Frame with only one triggering for each direction

### **ComSignal, ComSignalGroup, ComGroupSignal:**

- > ISignalToIPduMapping.shortName has to be unique
- > Only one triggering for each direction

## 10 Post-Build Loadable Requirements for Complex Drivers

This chapter summarizes the workflow, embedded software and tooling requirements that allow a post-build time update of complex drivers that are e.g. developed by the OEM or Tier1. The goal is to use the same update process and tooling as for the MICROSAR stack.

To understand this chapter it is mandatory that the general concepts described in this document are understood.

### 10.1 Memory Organization of Post-Build Loadable Data

Data that shall be changeable at post-build time need to be separated from data and code that is fixed at link time. This requires two steps:

- > Separation via file: All post-build loadable data is defined in an own files following the naming rule `<Cdd>_PBcfg.c`. This file must only contain data that is updated at post-build time.
- > Separation via memory sections: All post-build loadable data must be mapped to the post-build RAM and ROM block using dedicated memory sections. Thereby the post-build data can be mapped to the same RAM and ROM block as the post-build data of the MICROSAR stack. Typically the memory sections are:



#### Example

```
RAM data open: <CDD>_START_SEC_VAR_PBCFG
RAM data close: <CDD>_STOP_SEC_VAR_PBCFG
CONST data open: <CDD>_START_SEC_PBCFG
CONST data close: <CDD>_STOP_SEC_PBCFG
```

If module individual update is being used to separate the configuration data of the complex driver from the MICROSAR stack the complex driver initialization structure need to be mapped to the very beginning of the complex driver specific ROM block. This is done by a dedicated ROOT memory section.



#### Example

```
CONST root structure open <CDD>_START_SEC_PBCFG_ROOT
CONST root structure open <CDD>_STOP_SEC_PBCFG_ROOT
```

### 10.2 Post-Build Loadable Data Structures

Data structures that are updated at post-build time need to be compatible with the MICROSAR post-build update tool chain. This is a compiler frontend that interprets the post-build data and generates Hex data that can be downloaded to the ECU.

Post-build data is typically organized as follows:

- > One initialization structure that is handed over with `<Cdd>_Init` (Example: `Cdd_PBConfig`). This structure includes:

- > Data elements with fixed size but flexible values such as the number of handled elements.
- > Pointers to configuration arrays that can change their size and values at post-build time.
- > No data structures shall exist that are not referenced by the configuration initialization structure



### Example

```
#define CDD_START_SEC_VAR_PBCFG
#include "MemMap.h"

STATIC VAR(Cdd_ARamTp, CDD_VAR_PBCFG) Cdd_RamFeatureA [2];

#define CDD_STOP_SEC_VAR_PBCFG
#include "MemMap.h"

#define CDD_START_SEC_PBCFG
#include "MemMap.h"

STATIC CONST(Cdd_ADataTp, CDD_PBCFG) Cdd_FeatureAData [2] = {
    { 1, 0, FALSE } /* User 1 of feature A */
    { 1, 2, TRUE } /* User 2 of feature A */
};

STATIC CONST(Cdd_BDataTp, CDD_PBCFG) Cdd_FeatureBData [1] = {
    { 0, 0, FALSE } /* Dummy Element */
};

CONST(Cdd_PBConfigType, CDD_PBCFG) Cdd_PBConfig = {
    Cdd_FeatureAData /* Pointer to an array that defines data for Feature A */
    Cdd_FeatureBData /* Pointer to an array that defines data for Feature B */
    Cdd_RamFeatureA /* Pointer to runtime data of Feature A (Ptr to RAM) */
    0x02 /* No. of elements in Cdd_FeatureAData and Cdd_RamFeatureA */
    0x00 /* No. of elements in Cdd_FeatureBData. */
    0xCABA /* FinalMagicNumber. Helps to ensure PB data validity */
};

#define CDD_STOP_SEC_PBCFG
#include "MemMap.h"
```

## 10.2.1 Not supported code patterns

The following patterns are not supported by the MICROSAR post-build loadable update tool chain:

- > Pointers to array elements such as `&Cdd_FeatureAData[3]`. It is only possible to reference the array root element (`Cdd_FeatureAData`).
- > Pointer to data that is not part of the post-build update. Examples:
  - > Pointer to link time data of the same module
  - > Pointer to post-build loadable data of BSW modules that is located in a different post-build ROM block.
- > All kind of code as the post-build loadable tool chain is only able to process data
- > Data types must not be changed at post-build time as the access to post-build data is implemented in code that is fixed at link time

- > Bit fields are not supported
- > Arrays must be initialized completely (i.e. a value must be defined for each element)

As part of the complex driver test it is mandatory to check if the MICROSAR post-build tool chain is able to process the created structures correctly.

### 10.3 Initialization Function

The complex driver is typically initialized by BSWM or ECUM using a pointer that references the module initialization structure.



#### Caution

ConfigPtr (in the example below) is a pointer to const. A compiler may assume that the referred data never changes and may inline some data directly to optimize the code.

To avoid this it is recommended to never access ConfigPtr directly but to create a copy of the pointer in RAM. Some compilers may also disable such optimization based on the key words “volatile const” or similar.



#### Example

```
STATIC P2CONST(Cdd_PBConfigType, CDD_VAR_ZERO_INIT, CDD_PBCFG) Cdd_ConfigSet_pt =
NULL_PTR;

...

FUNC(void, CDD_CODE) Cdd_Init(P2CONST(Cdd_PBConfigType, AUTOMATIC, CDD_PBCFG) ConfigPtr)
{
    /* Initialize RAM pointer to avoid compiler optimization */
    Cdd ConfigSet pt = ConfigPtr;

    /* Do what ever Cdd Init (and other functions) shall do using the RAM copy */
    Cdd_ConfigSet_pt -> CddData...
}
```

Any access to post-build data must be done using the RAM pointer (Cdd\_ConfigSet\_pt in the example above).

It may be helpful to check if the referenced configuration data is correct by comparing a “final magic number”. This random number is hard coded in the initialization function and also within to the post-build configuration structure. This allows detecting problems that cause invalid configuration data to be handled over to the BSW module.

**Example**

```
FUNC(void, CDD CODE) Cdd Init(P2CONST(Cdd PBConfigType, AUTOMATIC, CDD PBCFG) ConfigPtr)
{
    ...
    /* Initialize RAM pointer to avoid compiler optimization */
    Cdd ConfigSet pt = ConfigPtr;
    ...
    /* Check if configuration data are valid */
    if(Cdd_ConfigSet_pt -> FinalMagicNumber != 0xCABA)
    {
        /* BSW module cannot be initialized. E.g. report Problem to ECUM */
        Ecum BswErrorHook(CDD MODULE ID, ECUM BSWERROR MAGICNUMBER);
    }
}
```

## 10.4 General Rules and Guidelines

The following rules have to be considered when creating code for post-build loadable modules

- > Handle IDs shall be fixed: IDs of elements that can be referenced in embedded code using a handle ID (resp. a defined) must not be changed at post-build time if the users cannot be updated with the changed handle IDs at the same point in time.
- > By default it is possible to delete elements at post-build time. Deleting elements introduces new problems such as gaps in the handle ID range and the possibility (that in turn could be forbidden by a guideline) that an API is used with a handle ID that is no longer present. Whether such invalid accesses can be detected at runtime depends on e.g if new elements are allowed to “reuse” empty slots for sake of optimization.
- > Pre-compile time parameters that are included in containers that can be added at post-build time: Generated data that is based on such parameters must consider post-build time update as such (pre-compile) parameters can be added when creating a new container at post-build time.
- > Pointer between post-build data and non-post-build data are not supported as the addresses cannot be resolved. Consequently alternative patterns such as an index are required to be used.

## 10.5 Configuration and Code Creation

The above mentioned <Cdd>\_PBcfg.c/h files can either be edited manually (including updates at post-build time) or be generated by any kind of code generator that follows the rules mentioned above.

If code is generated it is recommended to use DaVinci Configurator Pro to configure the complex driver based on the AUTOSAR methodology. This has the following advantages:

- > Convenient user interface (through Basic editors) including persistency using ARXML
- > Possibility to create own BSWMD files using DaVinci Configurator Pro.MD
- > DaVinci Configurator supports configuration phases and disables configuration options no longer available (i.e. as fixed in the pre-compile phase)

- > MICROSAR ECUM checks a pre-compile configuration data hash value at ECU startup to ensure pre-compile configuration data consistency of all included modules.
- > Possibility to integrate and access the complex driver with MICROSAR BSW modules

### 10.5.1 Guideline for BSWMD File creation

To include a complex driver into DaVinci Configurator Pro a BSWMD file need to be created. The file describes the configuration parameters and containers as well as the configuration phase in which modifications are allowed.

The following rules need to be applied in order to allow a BSW module to provide container and parameter that can be modified at post-build time.

This description is based on the Vector tool EPARM that is part of DaVinci Configurator Pro.MD

- > Within the EcuModuleDef enable the Implementation Configuration. Variant for Post-Build Loadable by adding this element to the list. This will allow parameters and container to support post-build loadable.

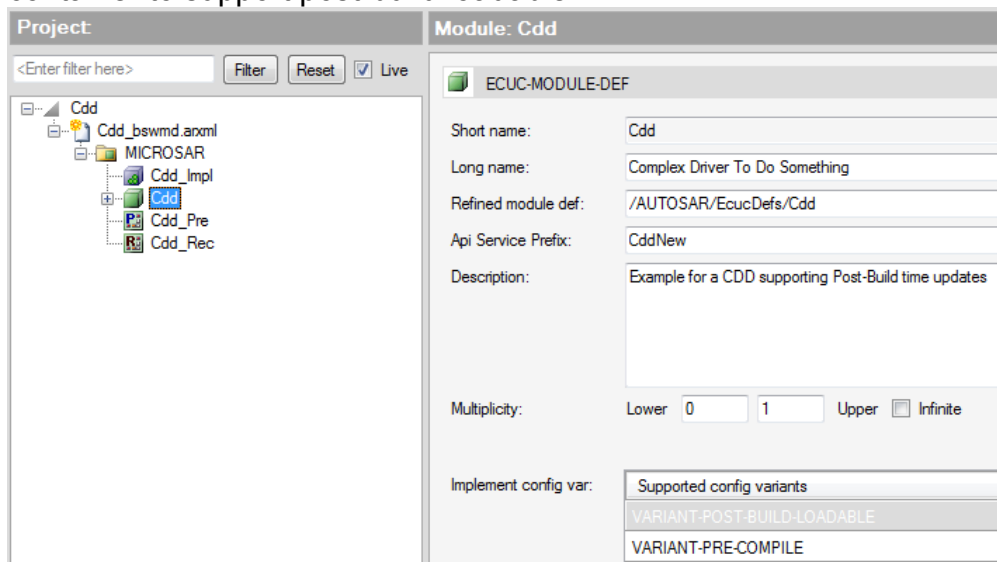


Figure 10-1 EPARM: Define that a BSW module is able to support post-build loadable

- > For container that shall be changeable at post-build time, set the flag “Post-build changeable” to true. By default AUTOSAR and EPARM also allow a container to be deleted at post-build time. If this is not supported by the implementation, a Vector specific BSWMD extension allows overwriting this property by disabling the flag “Post-build deleteable”.  
If the container shall not be changeable (add, delete, rename) at post-build time remove the flag “Post-build changeable”.

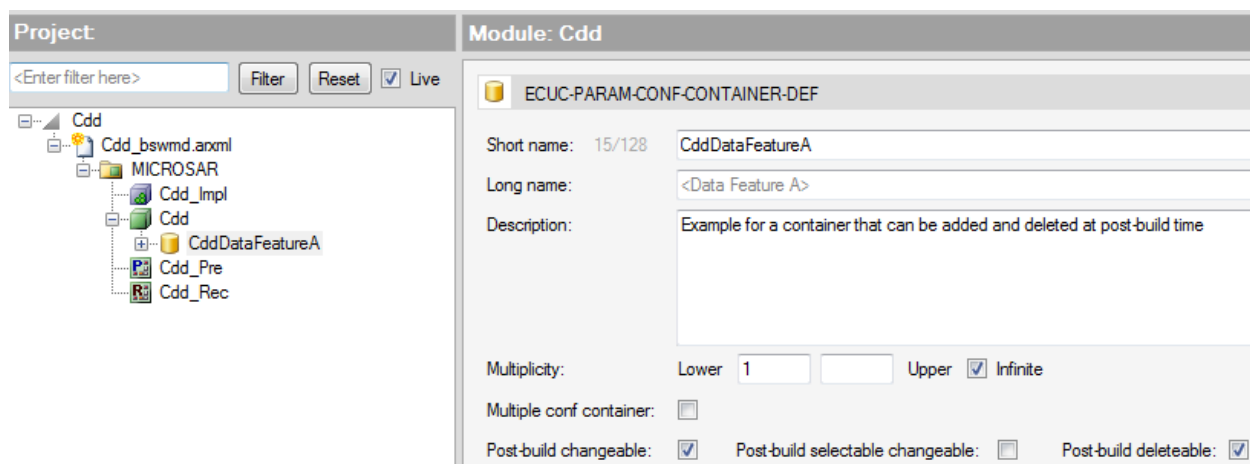


Figure 10-2 EPARM: Define that a container is able to be added and deleted at post-build time

- > For parameters that shall be changeable at post-build time, set the Implement. Config Class for the Config Variant Post-Build Loadable to the Config Class POST-BUILD. If the parameter shall not be changeable at post-build time, set the Config Class to PRE-COMPILE for all Configuration Variants.

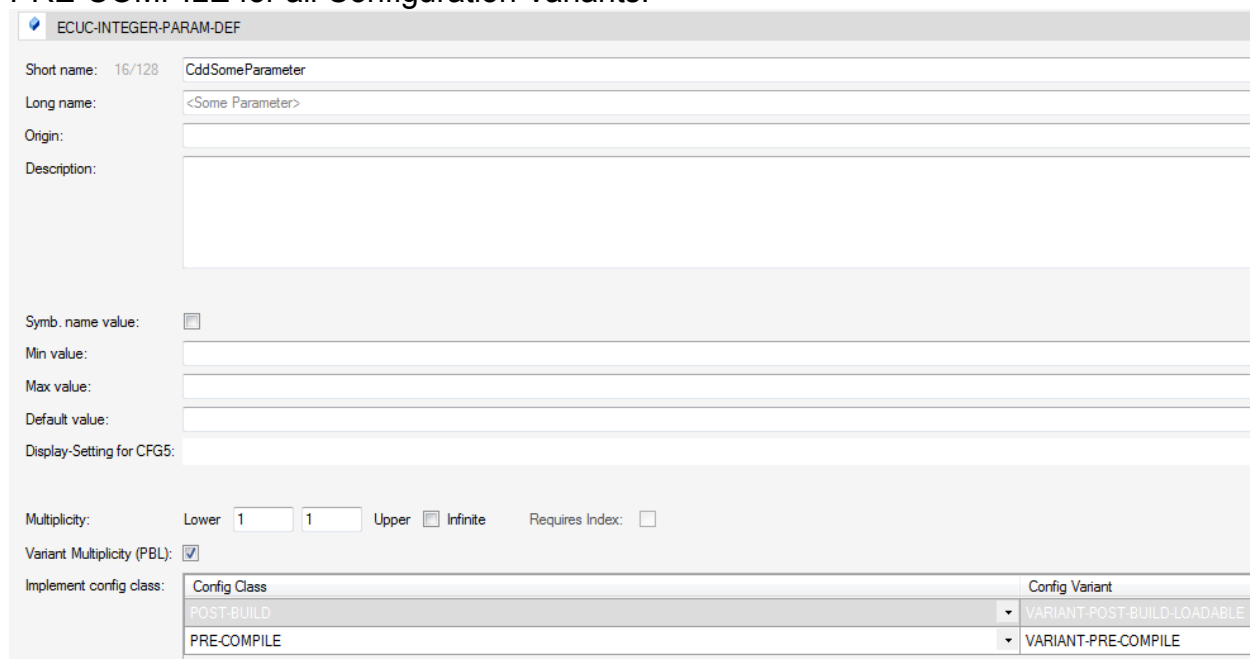


Figure 10-3 EPARM: Define that a parameter can be modified at post-build time

## 10.6 BSW Stack initialization

Once the complex driver is enabled in the BSW configuration, the MICROSAR ECUM and BSWM provide a service to automatically initialize BSW modules. If the BSW module supports post-build loadable (and is not updated individually) the following steps are required to publish the existence of the complex driver to ECUM and BSWM. As a consequence the configuration pointer to the initialization structure of the complex driver will be added to the global root structure.



Open the Basic editor and navigate to the EcuC module. Add a new Initialization function and configure it accordingly.

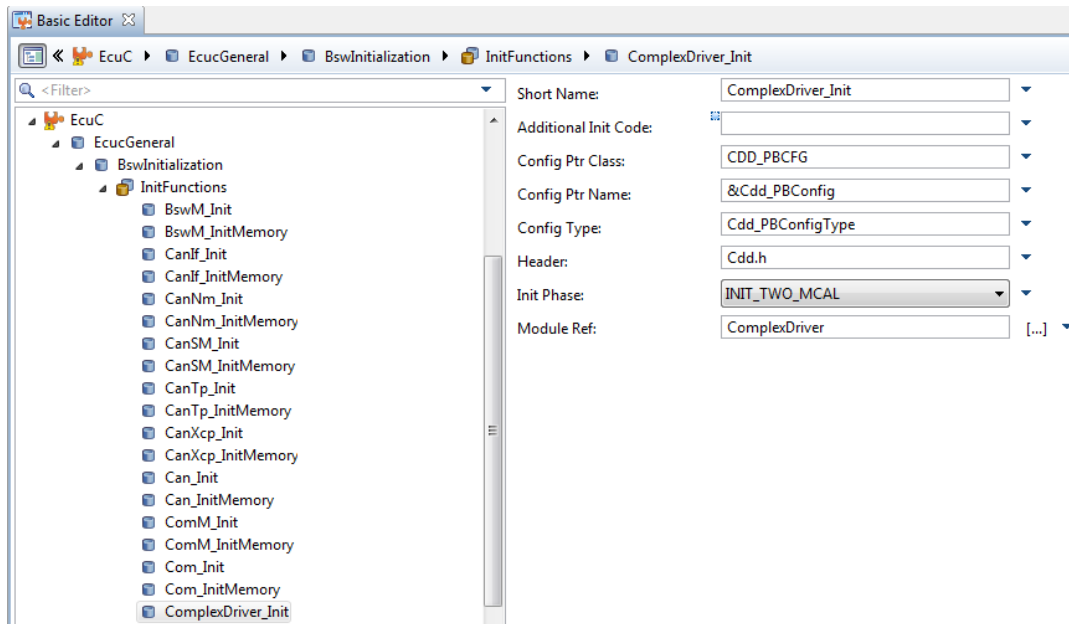


Figure 10-4 Announcing a initialization function to ECUM and BSWM for module initialization

The configured header (Cdd.h) shall publish the configured Config Ptr Name (declaration of the modules Post Build root variable) and Config Type (type definition of the modules Post Build root variable).

Once the initialization function is announced in the ECUC module, use e.g. the BSW Management editor to create the necessary BSWM initialization actions. The Module Initialization Auto Configuration Assistant will propose an initialization sequence for the complex driver.

## 11 Known Issues

This chapter provides help for issues that cannot be resolved directly by adapting the MICROSAR post-build loadable solution.

### 11.1 Inline assembler in Os.h cause PostBuildXmlGen to throw errors

Some operation systems publish compiler specific inline assembler through Os.h. This will lead to an error when executing the post-build loadable process. The Post-build XML Generator (PostBuildXmlGen.exe) cannot process such compiler specific none ANSI-C directives.

To overcome this problem during the post-build loadable process, replace the original Os.h file by a custom Os.h file that satisfies all requirements of the post-build loadable process.

As the post-build loadable process only handles constants and variable, there is no OS implementation required. The (adapted) Os.h file is only required by PostBuildXmlGen.exe to allow its compiler frontend to build up a complete include tree.



#### Practical Procedure

Add the adapted Os.h file to a folder which is not an include folder for the original compiler. Configure the PostBuildXmlGen.exe to use this file by the command `--include="<PathToFolderWithModified\Os.h>"` as first include information in the list of include files and directories.

In this case the original Os.h can remain unmodified in its original location (where it will be used by the original compiler at pre-compile and link time). The adapted (second) Os.h file is used by the post-build update process.



#### Caution

When using the original compiler e.g. at pre-compile and link time, the original Os.h file content must be used without modification.

The content of the (modified) Os.h may be similar to the following example:

```
typedef uint8 StatusType;
typedef uint32 ResourceType;
#define E_OS_ACCESS 1
#define E_OS_NOFUNC 5
#define E_OS_DISABLEDINT 12
void SuspendOSInterrupts(void);
void ResumeOSInterrupts(void);
void SuspendAllInterrupts(void);
void ResumeAllInterrupts(void);
void DisableAllInterrupts(void);
void EnableAllInterrupts(void);
StatusType GetResource(ResourceType ResId);
StatusType ReleaseResource(ResourceType ResId);
```

## 12 Glossary and Abbreviations

### 12.1 Glossary

Term	Description
DaVinci Configurator	Configuration and generation tool for MICROSAR BSW
Identity Manager	The MICROSAR Identity Manager is the Vector implementation of post-build selectable.
Memory Mapping Section	AUTOSAR concept that allows mapping RAM or ROM data to certain addresses. For this purpose the BSW modules encapsulate all data with definitions that are resolved by MemMap.h. Each section has a start and a stop key e.g. BSW_START_SEC_PBCFG / BSW_STOP_SEC_PBCFG. For better readability the corresponding memory section is referred to as SEC_PBCFG (i.e. the BSW and START/STOP part is omitted).
Post-build loadable	Post-build loadable allows updating the configuration at post-build time by downloading an updated configuration as a hex file. As a consequence the configuration can be modified in the configuration phase PostBuild.
Post-build RAM block	Linear RAM area reserved for post-build loadable MICROSAR modules. This memory area is used for data structures that can change their size at post-build time (e.g. by adding a new message). The memory area need to be assigned by the integrator at link time and must be of sufficient size to support increased RAM consumption at post-build time.
Post-build ROM block	Linear memory area in FLASH reserved for post-build loadable MICROSAR modules. This memory area is used for configuration data that can change their size at post-build time (e.g. by adding a new message). The memory area is assigned by the integrator at link time and must be of sufficient size to support increased configuration data size at post-build time.
Post-build selectable	Post-build selectable allows storing several alternative configurations within the ECU. At startup time one of these configurations is chosen and used for BSW initialization.

Table 12-1 Glossary

### 12.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)

SWS	Software Specification
-----	------------------------

Table 12-2 Abbreviations

## 13 Contact

Visit our website for more information on

- > News
- > Products
- > Demo software
- > Support
- > Training data
- > Addresses

[www.vector.com](http://www.vector.com)