Identity Manager - Multiple ECUs

Version 1.6
2010-09-09
Application Note AN-ISC-8-1102

Author(s) Christian Weber, Hannes Futter, Marco Wierer

Restrictions Customer confidential - Vector decides

Abstract Configuration of Multiple ECUs with Vector's AUTOSAR BSW stack.

Table of Contents

1.0 Overview

This application note explains how to configure and use the “Multiple ECUs” feature in Vector AUTOSAR stacks.

Contact Information: www.vector.com or +49-711-80 670-0

1.1 What is the “Identity Manager” about?

With the Identity Manager ECUs can be configured to run in different scenarios without major changes[1] in the software. The functionality provided by the Identity Manager is not covered by AUTOSAR explicitly. The use cases are:

- Physical Multiple ECU
- Multiple Configurations ECU
- (Virtual Multiple ECU)

This document explains about how to use the Physical Multiple ECU feature.

1.1.1 Physical Multiple ECU

This aspect covers an ECU within a specific car line of an OEM. If there are several (>1) almost identical instances of this ECU which are connected to the same vehicle network at the same time, they are called “Physical Multiple” ECUs or “Multiple ECUs” in short.

Examples: Door Module (DM), Seat Module (SEAT), Adaptive Damping System (ADS) …

In this application note, we’ll use the Door Modules with the following instances as an example:

- Front Right DM_FR
- Front Left DM_FL
- Rear Right DM_RR
- Rear Left DM_RL

The physical control units are derived from a common class with respect to the functionality. Hence, all control unit instances derived from this class potentially support the same superset of the functional scope. The actual ECU instances will be configured according to the position they will have in the network topology when they are built into the car.

According to the configuration there may be

- A different functional scope
  - E.g. different MMI (man machine interface) for each ECU instance: Mirror control or special window control for all windows only on the Front Left Door Module, not on the other modules (application dependent, not covered by this application note)
- Different communication properties of the instances:
  - Own NM message to transmit
  - Own signals to transmit and to receive
  - Own messages to receive and to transmit

The System Description of the OEM contains the instances of such Physical Multiple ECU. For each instance, the OEM creates a separate ECU Extract of System Description. This set of ECU Extracts is the configuration input of the Physical Multiple ECU.

[1] no major changes means: No changes in the application, but there are changes in the initialization sequence of the basic software which are not covered by AUTOSAR

1.1.2 Multiple Configurations ECU

Across multiple car lines of an OEM, the communication behavior considerably differs. However, the Multiple Configurations ECU is shared among an OEM's car lines. Therefore there must be a mechanism for the adaptation of this ECU to different environments, i.e. different car lines.

Note:	Multiple Configurations ECUs are currently not supported.

1.1.3 (Virtual Multiple ECU)

Virtual Multiple ECUs are control units which are only virtually separate ECUs from a logical point of view but residing inside one ECU hardware board. An example for this is a combined radio / gateway control unit.

Note:	Virtual Multiple ECUs are currently not supported.

1.2 Operation principle of Physical Multiple ECUs

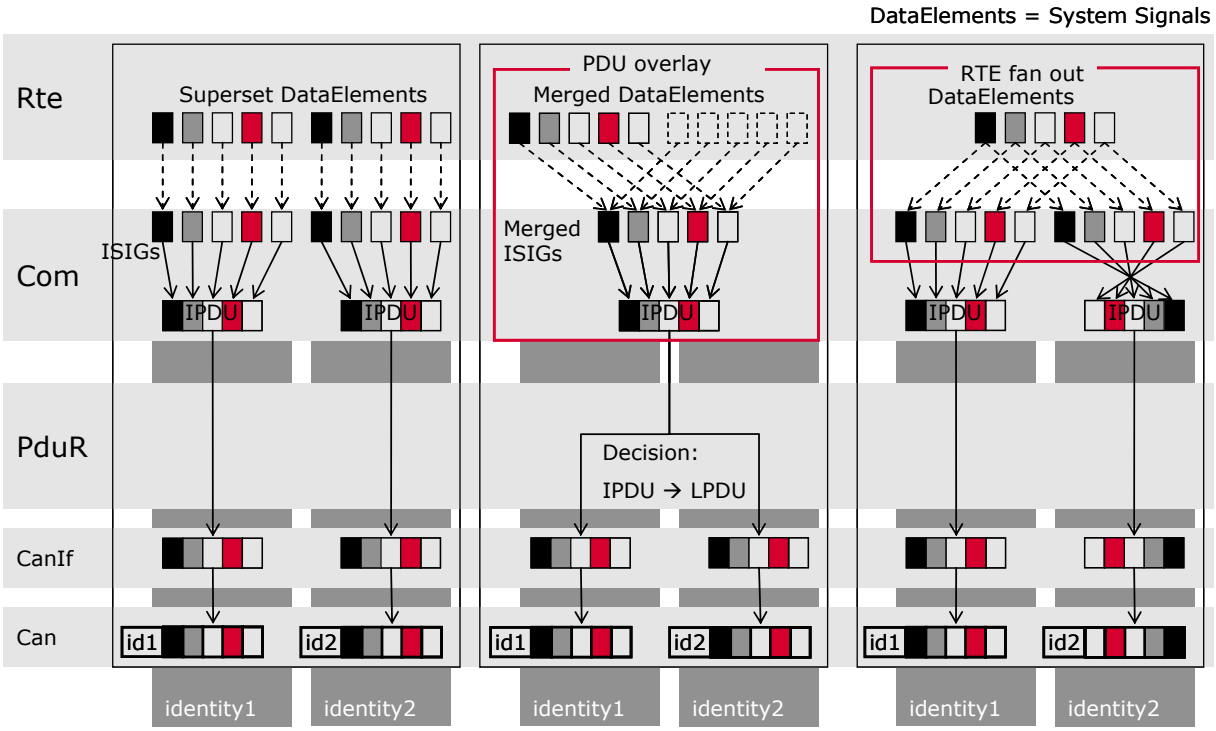

Figure 1 – Overview of the Multiple ECU feature, Tx path shown only.

In Figure 1, you see an example of the operation principle of a Multiple ECU. Here, the Tx path is shown. The leftmost column displays what happens if you have to configure a Multiple ECU sending out different CAN messages and cannot use any kind of optimization because the signals for each identity are different. For each set of signals which have to be sent, a separate PDU is created and mapped to a corresponding CAN frame which is actually sent out. This situation also applies if you have a PDU which is transmitted in one identity and received via a different identity. Then, despite having the same signals on application level, two PDUs are created, one for transmission in identity1, a second PDU for reception in identity2.

However, if you have semantically equivalent signals sent in different CAN messages, there is room for optimization. This is shown in the middle column. Semantically equivalent signals can be merged into one common PDU, the transmission is done in different CAN frames, but their content is equal.

If the signal layout is not equivalent over the identities, this optimization is not possible (right column of Figure 1). Therefore separate PDUs have to be used. Additionally: There may be only one signal on system level which is sent in different PDUs as shown in the rightmost column. In this case the RTE provides the fan-out to the Com.

Vectors implementation of the Multiple ECU feature incorporates overlaying of PDUs as well as RTE fan in/out (signal overlay).

2.0 Configuring a Multiple ECU

Before you start with the BSW configuration, you have to create a valid initial active ECU configuration based on a collection of all required ECU Extracts of System Configuration. Additionally, the required startup mechanisms in the ECUM have to be provided. The State Manager has to recognize which identity is currently active and has to decide which initialization sequence it has to select for the current identity. Finally, the BSW can be configured based on this ECUC file.

The PDUR controls the communication behavior in each instance of a Multiple ECU configuration. The PDUR implements the reception and transmission behavior as well as the mapping of CANIF PDUs (N-PDUs) to the COM PDUs (I-PDUs). If there is no optimization, the transmission and reception behavior is implemented over a superset of Rx and Tx messages in the CANIF. According to the active identity, the message is either transmitted or received or nothing of this.

For the straight forward case, this can be reached by creating different PDUs for each identity. In an optimized case, the memory consumption can be reduced by merging redundant PDUs.

2.1 ECUC creation (without overlay)

For each ECU identity, the InitialEcuC generator reads in one ECU Extract of System description – see the figure below for an overview of the tool flow:

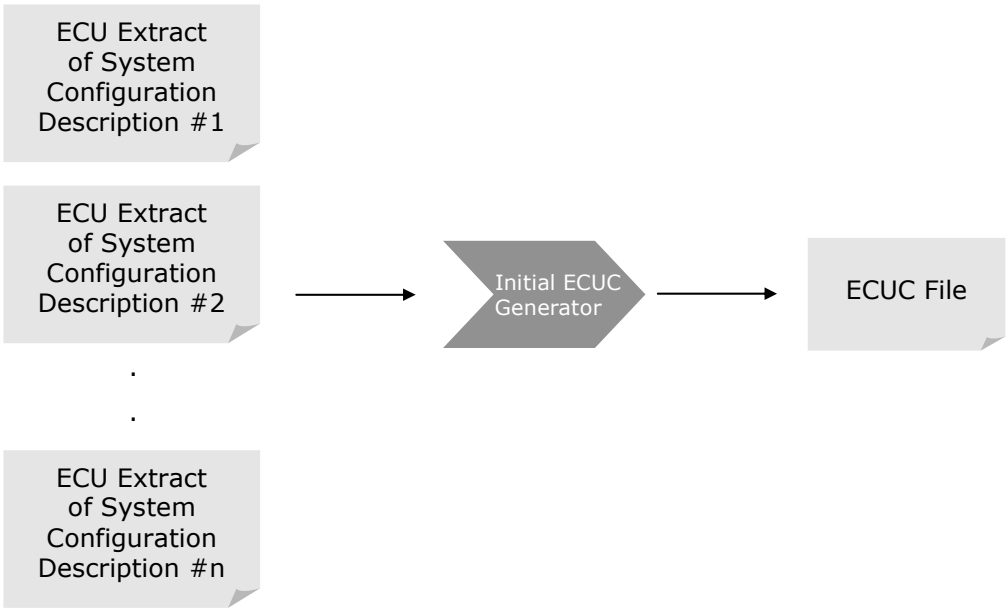

Figure 2 – Creating an ECUC file from Multiple ECU Extracts

In our example, we have the Door Module ECU with 4 physical Multiple ECUs.

That means the OEM will provide four different ECU Extracts of System Configuration Description:

- DM_FL.arxml
- DM_FR.arxml
- DM_RL.arxml
- DM_RR.arxml

The InitialEcuC generator is able to merge these extracts in one ECUC file which provide the superset for the BSW configuration.

Tool usage:

```
GenTool_CsAsrInitialEcuC.exe [-p <PrjName>] -m <EcuExtractOfSd1>…<EcuExtractOfSdN>
```

Hint:	The optional parameter –p <PrjName> gives the name of the resulting configuration/project. The name of the resulting configuration and/or project is the first file name given to the generator tool if no – p <PrjName> option is given. In our example, the `PrjName = DM_FRL`.

Open a command line prompt and enter:

```
GenTool_CsAsrInitialEcuC.exe -p DM_FRL -m DM_FL.arxml DM_FR.arxml DM_RL.arxml
DM_RR.arxml
```

A result of the generator run will be:

DM_FRL.ecuc.arxml (active configuration – this one will be iteratively changed during BSW configuration phase and enhanced by vendor-specific attributes)

DM_FRL.Initial.ecuc.arxml (initial configuration holding the basic pre-configuration for the BSW)

DM_FRL.ecuc.vdsxml (this is a file which you can load with GENy; it references initial and active configuration.

DM_FRL.Error.txt (describing errors encountered during the conversion)

Note:	The `GenTool_CsAsrInitialEcuCMultConfMerge.dll` must reside in the same directory like the ECUC generator dll (`GenTool_CsAsrInitialEcuCDsIf.dll`).

2.1.1 What happens in the ECUC?

The ECUC loader creates one network node per identity. Further, it creates a message/PDU list with references to the network node.

The used set of PDUs or signals is a superset of the used PDUs or signals (unique names without duplicates).

For each identity it creates a so called multiple configuration container in the ECUC file below all basic software modules for which a multiple configuration container is defined in their BSWMD files. The multiple configuration container incorporates the ECU name as identification. The container for the ECUM holds references to the init structures of the modules.

The name of the multiple configuration container is composed of

```
<name of the structure in the bswmd/paramdef file>_<EcuShortName>
```

The EcuShortName is taken from the ECU Extract of System Configuration. In the door module example,

ECU short name for the respective multiple configuration containers will be DM_FL, DM_FR, DM_RR, DM_RL.

2.2 ECUC Creation (with overlay)

Multiple ECUs typically have some messages that have the same signal layout and contain semantically equal signals (only with different names) shared at least by some of the identities (see Figure 1 for an overview of the feature).

For this case, a separate PDU is assigned to each semantic group of signals. First, this is not efficient, because you maintain several PDUs which have the same semantic and transport the same signals. The memory consumption for these PDUs is redundant. Second, the application code or the software component should not need to distinguish between the different active identities when handling the contained signals.

So the solution is to configure only one common PDU for all identities and use the potential for an optimization called PDU overlay (see the middle column of Figure 1).

The needed PDU overlay must be configured manually. This means the user has to select the PDUs which can be merged together, and has to provide a respective control file in XML (Overlay Control File).

In the PDUR one such PDU will result in the reception or transmission of a unique L-PDU on interface level.

The configuration of the PDUR at startup will determine to which Rx or Tx L-PDU the I-PDU will be mapped, i.e. which CAN message will be sent with the contents of this I-PDU.

Precondition:

- Existence of multiple ECU Extracts, one for each identity.
- An overlay control file.

The figure below shows the tool flow for creating an ECUC file with an Overlay Control File:

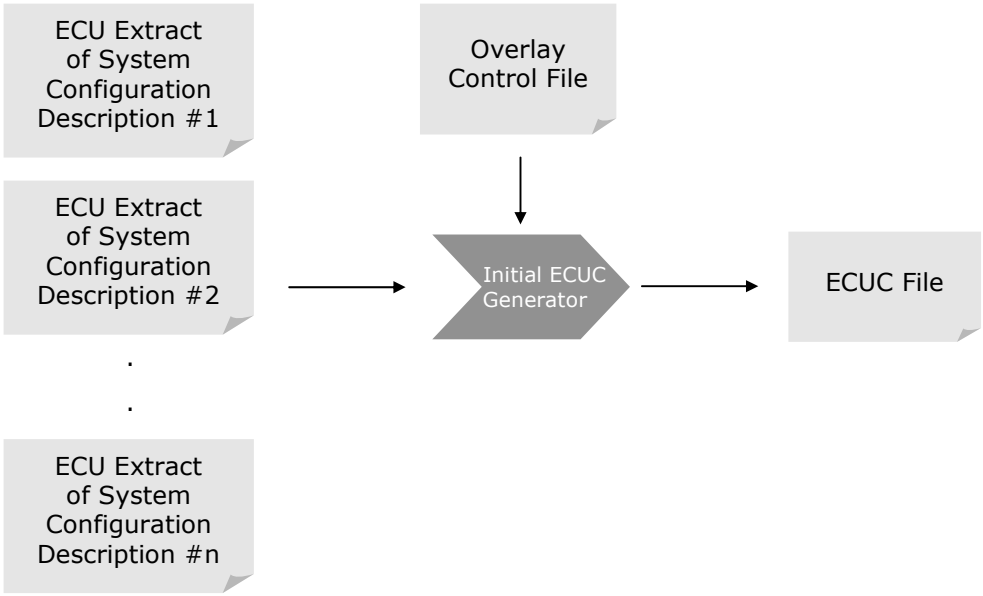

Figure 2 – Creating an ECUC file with Overlay Control File

Tool usage:

```
GenTool_CsAsrInitialEcuC.exe [–p <PrjName>] –m [<EcuExtractOfSd1>…<EcuExtractOfSdN>]
-o <OverlayControlFile>
```

Hint:	The optional parameter –p <PrjName> gives the name of the resulting configuration/project. The name of the resulting configuration and/or project is the first file name given to the generator tool if no – p <PrjName> option is given. In our example, the `PrjName = DM_FRL`. For the PDU overlay case, the PrjName can be given in the control file as well.

Open a command line prompt and enter:

```
GenTool_CsAsrInitialEcuC.exe -m DM_FL.arxml DM_FR.arxml DM_RL.arxml DM_RR.arxml -o
MultiEcuExample.arxml
```

You can also omit specifying the multiple ECU Extracts since the ECU Extracts are already referenced in the Overlay Control File. The following invocation is equivalent to the one above:

```
GenTool_CsAsrInitialEcuC.exe -m -o MultiEcuExample.arxml
```

2.3 Overlay Control File

An example file is shown at the end of this chapter. Basically you have to define the following:

- The source extracts from which you read-in the different identities and a master extract which defines the default names of the merged entities.
- The instance name for the new ECU
- The PDU overlays (must be members of different identities)
- The signal overlays (for RTE fan in/out of signals with different PDU layout)
- Splitting of PDUs in Rx and Tx direction to explicitly define the overlay: Rx PDUs only, Tx PDUs only or Tx and Rx PDU overlay

2.3.1 Specifying the ECU Extracts to be merged

The ECU Extracts of System Configuration Description which shall be merged are specified within the `<SourceECUExtracts>` tag. You can specify a set of ECU Extracts (instances) to be merged via the `<MergingExtract>` tag and the master ECU Extract via the `<MasterExtract>`.

Note:	If an element (signal/PDU) is not present in the master extract, the default name of the element is defined by the order of the ECU Extracts in the `<SourceECUExtracts>` tag.

See example below for specifying a set of ECU Extracts to be merged:

```
<SourceECUExtracts>
      <MasterExtract> EcuExtractOfSd </MasterExtract>
      <MergingExtract> EcuExtractOfSd1 </MergingExtract>
      <MergingExtract> EcuExtractOfSd2 </MergingExtract>
      <MergingExtract> … </MergingExtract>
</SourceECUExtracts>
```

2.3.2 Specifying the merged ECU instance name

The ECU instance name of the merged ECU is specified using the <NewECUInstanceName> tag:

```
<NewECUInstanceName> MultiIdEcu </NewECUInstanceName>
```

2.3.3 PDU overlay

The PDU overlays are defined inside the <OverlayPDUs> tag, which contains all definitions for the overlays. The overlays consist of equivalent PDU names. The definition which PDUs shall be overlaid (“Rx”, “Tx” or “TxRx”) is specified by the direction parameter within the <OverlayingPDUs> tag.

```
<OverlayPDUs>
      <OverlayingPDUs direction=”Tx”>
            <PDU>PDU1_FL</PDU>
            <PDU>PDU1_FR</PDU>
            <PDU> … </PDU>
      </OverlayingPDUs>
</OverlayPDUs>
```

Note:	For overlaying PDUs, you also have to specify a corresponding signal overlay section in the overlay control file (see description below).

2.3.4 Signal overlay

The signal overlays for specifying the RTE fan in/out are defined inside the tag named <MatchingSignals> which contains all signals which shall be overlaid. Like the PDU overlay, the definition which signals shall be overlaid (“Rx”, “Tx” or “TxRx”) is given in the direction parameter of the <MatchingSignals> tag.

The name of the overlaid signal is specified by the <NewSignalName> tag – see example below:

```
<MatchingSignals direction=”Rx”>
      <Signal>Signal1_FL </Signal>
      <Signal>Signal1_FR</Signal>
      <Signal> … </Signal>
      <NewSignalName>Signal1_XX</NewSignalName>
</MatchingSignals>
```

Please consider the following constraints when specifying signal overlays:

- For overlaying signal groups, you have to specify both the signal group and each group signal in the <Signal> tag.
- The <MatchingSignals> tag is only evaluated by DaVinci Developer. The InitialEcuC Generator doesn’t verify the <MatchingSignals> section.
- The name of the resulting signal, specified by the <NewSignalName> tag shall correspond to the name of the port prototype in the System Description to enable DaVinci Developer to automatically map these signals.

2.3.5 Splitting the direction of PDUs

Within the <SplitTxRxPDUs> tag, you have to provide the PDUs which cannot be overlaid in both communication directions:

```
<SplitTxRxPDUs>
      <PDU>PDU1_FR</PDU>
      <PDU>PDU1_FL</PDU>
</SplitTxRxPDUs>
```

2.3.6 Example

```
<DVMultiECUConfiguration xmlns="http://www.vector-informatik.de/MultiECUSupport"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.vector-informatik.de/MultiECUSupport
MultiEcuConfig.xsd" version="1.0">
    <SourceECUExtracts>
        <MasterExtract>DoorECU_FL</MasterExtract>
        <MergingExtract>DoorECU_FR</MergingExtract>
        <MergingExtract>DoorECU_RL</MergingExtract>
        <MergingExtract>DoorECU_RR</MergingExtract>
    </SourceECUExtracts>
    <NewECUInstanceName>MyMultiDoorECU</NewECUInstanceName>
    <OverlayPDUs>
        <OverlayingPDUs>
            <PDU>PDU1_FL</PDU>
            <PDU>PDU1_FR</PDU>
            <PDU>PDU1_RL</PDU>
            <PDU>PDU1_RR</PDU>
        </OverlayingPDUs>
        <OverlayingPDUs direction="TxRx">
            <PDU>PDU2_FL</PDU>
            <PDU>PDU2_FR</PDU>
        </OverlayingPDUs>
        <OverlayingPDUs direction="Tx">
            <PDU>PDU3_FL</PDU>
            <PDU>PDU3_FR</PDU>
        </OverlayingPDUs>
        <OverlayingPDUs direction="Tx">
            <PDU>PDU3_RL</PDU>
```

```
            <PDU>PDU3_RR</PDU>
        </OverlayingPDUs>
        <OverlayingPDUs direction="Rx">
            <PDU>PDU3_FR</PDU>
            <PDU>PDU3_RR</PDU>
        </OverlayingPDUs>
        <OverlayingPDUs direction="Rx">
            <PDU>PDU3_FL</PDU>
            <PDU>PDU3_RL</PDU>
        </OverlayingPDUs>
    </OverlayPDUs>
    <MatchingSignals>
        <MatchingSignals direction="TxRx">
            <Signal>Signal1_FL</Signal>
            <Signal>Signal1_FR</Signal>
            <Signal>Signal1_RL</Signal>
            <Signal>Signal1_RR</Signal>
            <NewSignalName>Signal1</NewSignalName>
        </MatchingSignals>
        <MatchingSignals>
            <Signal>Signal2_FL</Signal>
            <NewSignalName>Signal2</NewSignalName>
        </MatchingSignals>
    </MatchingSignals>
    <SplitTxRxPDUs>
        <PDU>PDU3_FL</PDU>
        <PDU>PDU3_FR</PDU>
        <PDU>PDU3_RL</PDU>
        <PDU>PDU3_RR</PDU>
        <PDU>PDU4</PDU>
    </SplitTxRxPDUs>
</DVMultiECUConfiguration>
```

Note:	There is an XML schema file (MultiEcuConfig.xsd) which is part of the SLP10 delivery that helps you to create a valid PDU overlay control file.

2.4 ECUC creation using DaVinci Project Assistant

DaVinci Project Assistant supports you in setting up a project based on an existing control file (see 2.3) or on the ECU extracts of the different identities. It automatically creates the initial active ECU configuration.

When specifying the input files for the project, you may select **Multiple ECU**. You may now either provide the existing control file or the ECU extracts and a location for an automatically created control file. This automatically created control file will contain the specified ECU extract references.

Note:	To define the PDU overlay (see 2.2) you need to manually edit the control file.

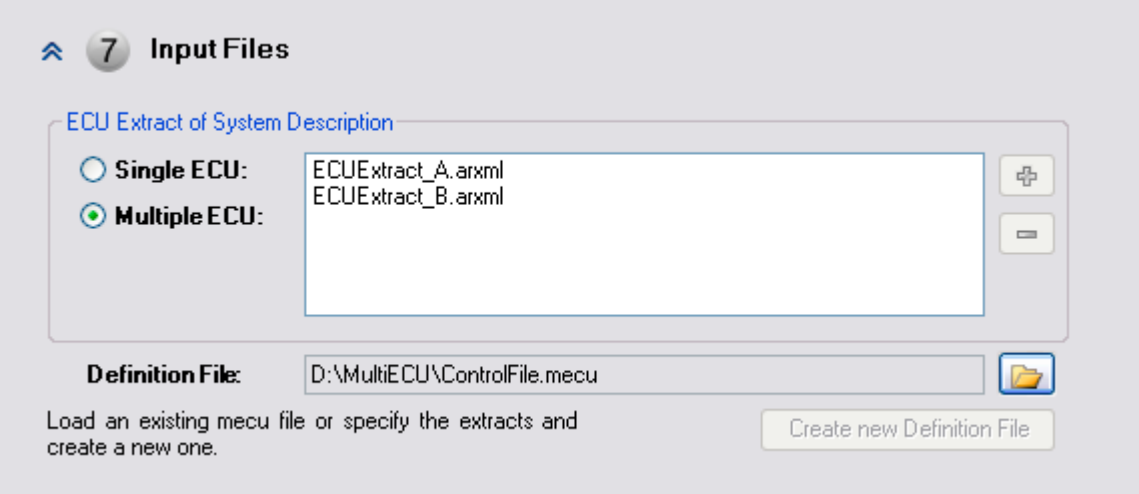

Figure 2 – DaVinci Project Assistant Input Files Specification

If you change the control file after having created the project, you may use the update function of the DaVinci Project Assistant to reflect your changes.

Note:	DaVinci Project Assistant resolves the references to the identities' extracts given in the control file relative to the control file's location.

2.5 Generating the RTE of a Multiple ECU

To generate the RTE of a Multiple ECU, the different ECU extracts – each containing the description of a single identity – need to be merged into a single ECU project representing all identities of the Multiple ECU. The merged ECU project (resp. an ECU Extract of System Description created by DaVinci Developer) is compatible to the ECUC (see 2.0) and can be used as input for the MICROSAR RTE generator.

The merged ECU project is created by the DaVinci Project Assistant. Alternatively, you may import an existing control file (see 2.3) into a DaVinci Developer workspace using the standard XML import mechanism. This will automatically create the merged ECU project.

Note:	DaVinci Developer resolves the references to the identities' extracts given in the control file relative to the control file's location.

3.0 Initialization of the BSW

The ECU State Manager (ECUM) implements the support for Multiple ECUs. The scope of this chapter is restricted to the characteristic settings for Multiple ECUs in the ECU State Manager only. Further initialization steps necessary for AUTOSAR systems are not described here and are assumed to be known.

The BSW initialization procedure in AUTOSAR works the following way:

- µC startup code execution (out of AUTOSAR scope).
- Call of the `main()` function → `EcuM_Init()`
- `EcuM_Init()` allows for initialization of the BSW modules on low level
- `EcuM_Init()` starts the OS, `EcuM_Init()` does never return.
- Schedule Manager Task starts and calls the `EcuM_StartupTwo()` in its task body
- `EcuM_StartupTwo()`: calls `SchM_Init()` and calls `DriverInitListTwo`
- The initialization for Multiple ECUs takes place in `DriverInitListTwo`...

The ECU State Manager (ECUM) itself does not allow different initializations in the `EcuM_Init()` function. The following ECUM properties will be defined only once and are commonly valid for all identities inside an ECUC for a Multiple ECU:

- Sleep modes
- Wake-up Sources
- ECUM Users
- COMM channels
- TTII successors and divisors
- DriverInitLists

The ECU firmware (more precisely: a user defined equivalent to an ECU State Manager initialization list) has to decide which identity is presently active and has to call the proper initialization routines for the basic software modules.

This initialization approach will

- **Not** adjust the application logic accordingly. The application implements a superset of all functions and is not necessarily aware of different identities.
- Set up the corresponding BSW modules correctly.

There is no tool-guided way to configure the ECUM for multiple ECU support. Only a manual solution is possible. For this purpose, an initialization function has to be added to the STARTUP II phase of the ECUM. (This is because the communication modules implement this feature. Communication modules are typically initialized during STARTUP II). This can be accomplished by adding a function call to `EcuM_Pbcfg.c` for the ECUM inside the configuration tool (for the SLP10, this can be done inside GENy, for other stacks this is DaVinci Configurator Pro). The necessary steps are described below.

3.1 Creation of the logic for identity selection

The Multiple ECU support requires a user defined function which determines for which identity the ECU shall start up. As described in the paragraph before, this extension has to be built into the STARTUP II.

Further, we assume that you provide this function in the files

```
EcuM_MultiIdentitySupport.h
EcuM_MultiIdentitySupport.c
```

The function which has to be called in STARTUP II itself may be called

```
EcuM_MultiConfigSelection()
```

Note:	The name of the header and C file are only examples, as well as the signature and function name `EcuM_MultiConfigSelection()`. The actual implementation is up to the supplier.

3.2 Prepare the ECUM configuration

Before you can use the Multiple ECU feature, you have to add `EcuM_MultiIdentitySuport.h` to the **Additional Includes** section in the ECUM.

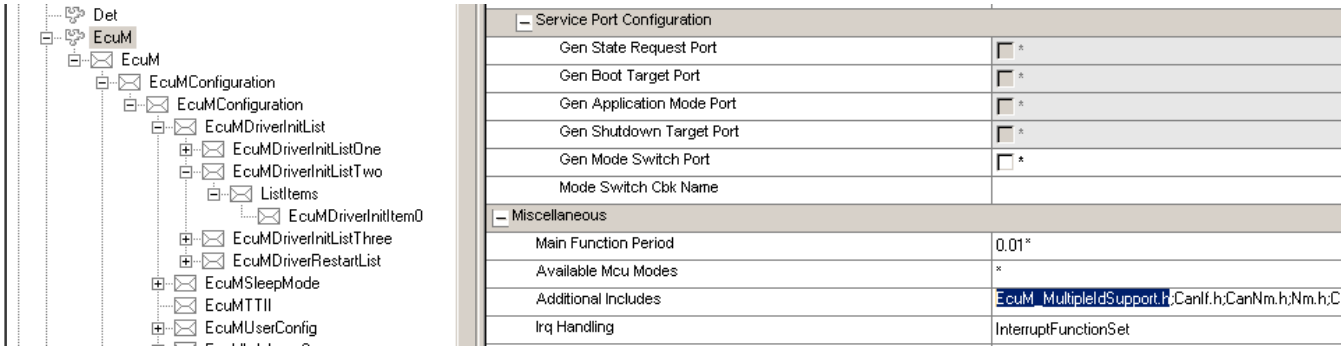

Figure 3 – GENy Configuration for Multiple ECU Feature

Further, you have to create a new entry in the **ECUM Driver Init List Two**, where `EcuM_MultiConfigSelection()` is called.

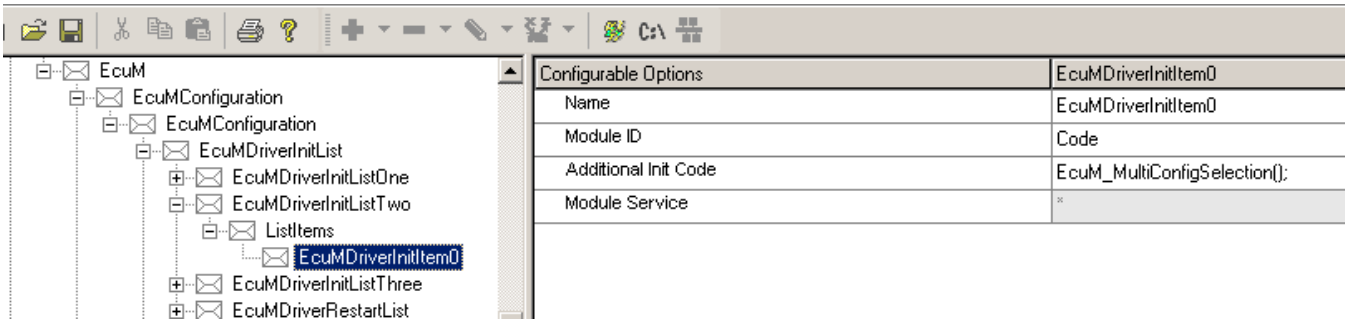

Figure 4 - GENy Configuration for Init List Two

3.3 Provide an user defined ECUM initialization function

An implementation for the MultiConfigSelection initialization function could look like that:

```
…
#include <EcuM_MultiIdentitySupport.h>
…
void EcuM_AL_DriverInitTwo () /*
{
  Dio_Init();
  EcuM_MultiConfigSelection();
}
```

You will need to provide a similar implementation in `EcuM_MultiIdentitySupport.c`:

```
FUNC(void, ECUM_CODE) EcuM_MultiConfigSelection(void)
{
  channelGroupIdType dioDipSwitch;

  switch (Dio_ReadChannelGroup(&dioDipSwitch))  /* read in DIP switch */
  {
    case MULTI_CONFIG_FL: /* initialization variant for door front left */
      Can_Init(NULL_PTR);
      CanIf_Init(NULL_PTR);
      CanSM_Init();
      PduR_Init(&PduRGlobalConfig_DM_FL);
      Com_Init(NULL_PTR);
      Nm_Init(NULL_PTR);
      CanNm_Init(&CanNmGlobalConfig_DM_FL);
      ComM_Init();
      CanTp_Init(&CanTpConfigSet_DM_FL);
      Dcm_Init(&DcmConfigSet_DM_FL);
      Vmm_Init();
      Dem_Init();
    break;
    case MULTI_CONFIG_FR: /* initialization variant for door front right */
      Can_Init(NULL_PTR);
      CanIf_Init(NULL_PTR);
      CanSM_Init();
      PduR_Init(&PduRGlobalConfig_DM_FR);
      Com_Init(NULL_PTR);
      Nm_Init(NULL_PTR);
      CanNm_Init(&CanNmGlobalConfig_DM_FR);
      ComM_Init();
      CanTp_Init(&CanTpConfigSet_DM_FR);
      Dcm_Init(&DcmConfigSet_DM_FR);
      Vmm_Init();
      Dem_Init();
    break;
    default:
    break;
  }
```

```
  return;
}
```

Inside each initialization sequence, there have to be instance-specific calls into the basic software init functions like

`<BSWM>_Init(&<configSetPtrName>_<EcuShortName>)`

Each module has to provide the instance-specific `<configSetPtrName>_<EcuShortName>` in its actual configuration obtained by the BSW configuration step performed in GENy.

The name of the parameter used for the init functions is formed according to the following rule:

`<configSetPtrName> comes from the structure name from bswmd/paramdef file.`

`<EcuShortName> is derived of the Ecu Names used in the ECU Extract of System Description`

Note:	You do not have the same scheme for the init functions of all BSW modules, some take no arguments some take a mandatory pointer. The following section provides an overview.

3.4 Where do I find the initialization structures?

For each BSW module, you can find the init structures in the generated code. Simply have a look at the following files in the \gendata folder:

`<ModuleName>_lcfg.c, <ModuleName>_pbcfg.c, <ModuleName>_cfg.c`

3.5 Overview of BSW module initialization

Considering the Multiple ECU feature, the AUTOSAR selectable post build approach is not fully supported. The AUTOSAR selectable post build approach is based on different unique initialization structures which are passed to the BSW via an initialization pointer in order to configure it.

Resource optimization would not be possible if unique initialization structures for each identity were provided. For this reason, the AUTOSAR selectable post build initialization mechanism is not applied for all modules.

For the majority of the modules, only one merged post build configuration/superset of initialization data will be created in order to save resources. For the remaining modules **DCM, CANTP, CANNM, IPDUM, PDUR,** individual initialization data sets are supported.

The following table gives an overview of the configuration pointer usage.

	Used init pointer	
BSW module	Not instance specific	Instance specific
COMM	☑	
ECUM	☑	
VMM	☑	
COM	☑	
IPDUM		☑
PDUR		☑
NM	☑	
CANNM		☑
CANSM	☑	

CANIF	☑	
CANDRV	☑	
CANTP		☑
DCM		☑
DEM	☑	

Table 1 – Overview of the module configuration for multiple identities

4.0 Diagnostics and Multiple ECU Support

The DCM/DEM module, developed by Vector, implements two dynamic configuration modes that allow switching among the available configuration sets at runtime (without need to reprogram the ECU). This requires that the superset of all variants must be pre-enabled during the software compilation and link process.

In Multiple ECU mode, DCM allows you to use:

- Multiple diagnostic configuration sets – a use case where the ECU always communicates over the same connection, but shall implement different functionality depending on some hardware (jumper) setting.
- Multiple communication configuration sets – typical use case where the ECU shall have the same functionality, but depending on its location on the network uses different communication parameters (communication message IDs i.e. door ECUs).
- Both multiple diagnostic and communication sets.

The combination of diagnostic and communication configuration is configured statically in GENy.

The details of Multiple ECUs and diagnostics are described in the Technical Reference for the DCM, chapter 7.6 and in the Technical Reference for the DEM, chapter 7.1.

5.0 Supported use cases

Based on the PDU overlay and RTE fan in/out approach, the current implementation of the Multiple ECU supports the following use cases:

- Overlaying of RX/TX PDUs with identical signal layout, respective PDUs with gaps in the signal layout. Each signal within the PDU has to have the same length and has to be of the same type: Single signals can only be overlaid by single signals, signal groups can only be overlaid by signal groups.
- RTE fan in/out of signals which are located in different PDUs.

6.0 Restrictions and limitations

LIN is invariant to the Multiple ECU feature that means it is not actively supported. As a consequence, the same LIN network exists in all identities. Multiple ECU support for FlexRay is planned for future releases.

Please also note the for RTE fan in, the signals to be overlaid have to be located in different PDUs

If a PDU contains signals for different identities, a wrapper software component needs to be implemented to select the according signal based on the current identity – see example below:

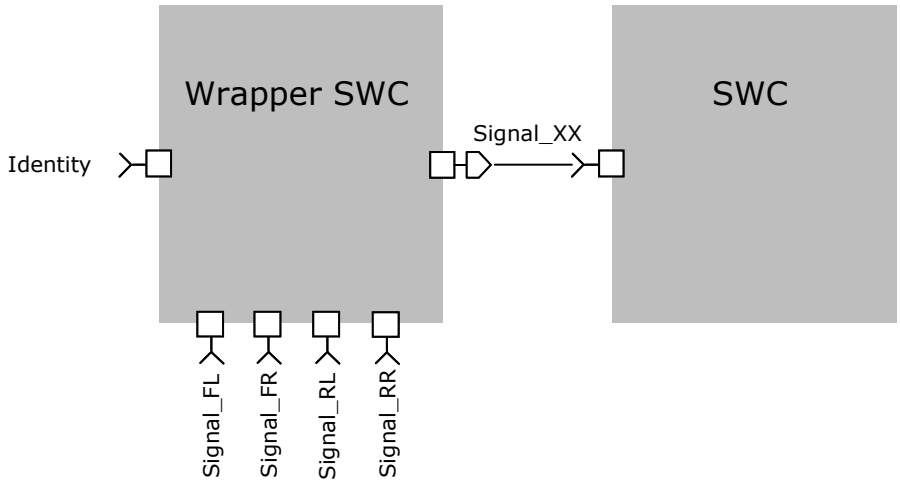

Figure 4 – Wrapper software component for switching identity specific signals

7.0 Additional resources

VECTOR TECHNICAL REFERENCES
MICROSAR DCM Technical Reference (based on ASR 3.0) Version 3.14

MICROSAR Diagnostic Event Manager (DEM) Technical Reference Daimler Version 2.2

8.0 Contacts

Germany
and all countries not named below:
Vector Informatik GmbH
Ingersheimer Str. 24
70499 Stuttgart
GERMANY
Phone: +49 711-80670-0
Fax: +49 711-80670-111
E-mail: info@de.vector.com

France, Belgium, Luxemburg:

Vector France SAS
168 Boulevard Camélinat
92240 Malakoff
FRANCE
Phone: +33 1 42 31 40 00
Fax: +33 1 42 31 40 09
E-mail: information@fr.vector.com

Sweden, Denmark, Norway,
Finland, Iceland:
VecScan AB
Theres Svenssons Gata 9
41755 Göteborg
SWEDEN
Phone: +46 31 764 76 00
Fax: +46 31 764 76 19
E-mail: info@se.vector.com

United Kingdom, Ireland:
Vector GB Ltd.
Rhodium
Central Boulevard
Blythe Valley Park
Solihull, Birmingham
West Midlands B90 8AS
UNITED KINGDOM
Phone: +44 121 50681-50
E-mail: info@uk.vector.com

China:
Vector Informatik GmbH
Shanghai Representative Office
Suite 605, Tower C,
Everbright Convention Center
No. 70 Caobao Road
Xuhui District
Shanghai 200235
P.R. CHINA
Phone: +86 21 - 6432 5353 ext. 0
Fax: +86 21 - 6432 5308
E-mail: info@vector-china.com

India:
Vector Informatik India Private Ltd.
4/1/1/1 Sutar Icon
Sus Road
Pashan
Pune 411021
INDIA

Phone: +91 9673 336575
E-mail: info@vector-india.com

USA, Canada, Mexico:
Vector CANtech, Inc.
39500 Orchard Hill Pl., Ste 550
Novi, MI 48375
USA
Phone: +1 248 449 9290
Fax: +1 248 449 9704
E-mail: info@us.vector.com

Japan:
Vector Japan Co. Ltd.
Seafort Square Center Bld. 18F
2-3-12, Higashi-shinagawa,
Shinagawa-ku
Tokyo 140-0002
JAPAN
Phone: +81 3 5769 7800
Fax: +81 3 5769 6975
E-mail: info@jp.vector.com

Korea:
Vector Korea IT Inc.
#1406, Mario Tower,
222-12 Guro-dong, Guro-gu
Seoul, 152-848
REPUBLIC OF KOREA
Phone: +82 2 807 0600
Fax: +82 2 807 0601
E-mail: info@kr.vector.com