How to create Cdd / IoHwAb with MICROSAR 4

Version 1.0
2018-09-14
Application Note AN-ISC-8-1228

Author Christian Leder

Restrictions Customer Confidential – Vector decides

Abstract Introduction on how to create a CDD or IoHwAb with Vector tools

Table of Contents

1 Overview

This application note shall give a short overview on how to create a Complex Device Driver (Cdd) or a IoHwAb with Vector tools and how to use it. Figure 1-1 shows the AUTOSAR layers. The IoHwAb is part of the ECU Abstraction Layer.

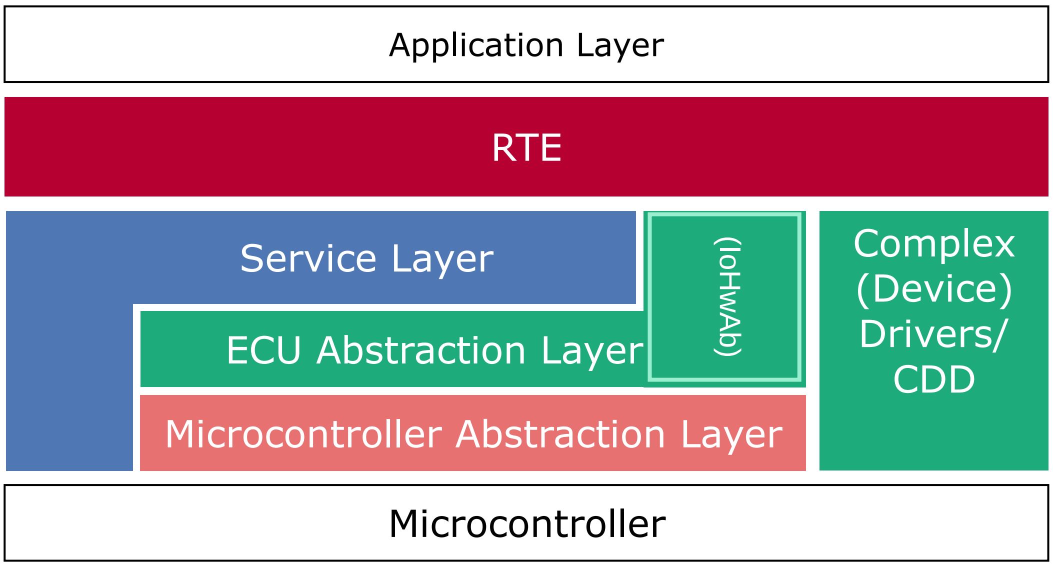


Figure 1-1 Architecture

1.1 Complex Device Driver

Complex Device Drivers are applied to implement software modules that are not standardized by AUTOSAR. They have access to other BSW modules, the Rte and direct hardware access. For example, these modules are not standardized communication drivers or legacy software which have to be included in an AUTOSAR stack.

1.2 IoHwAb

The module IoHwAb is the connection of the Rte to the Microcontroller Abstraction Layer. Within a layered software architecture like AUTOSAR it is not allowed for any software module to have arbitrary access to all modules. Interfaces are defined between certain modules and it is not allowed to bypass these interfaces. The module encapsulates the access to the I/O drivers such as Adc, Dio, Eep. etc. So, the IO signals are available for the application components.

Note
As seen in Figure 1-1 the difference of these modules is the direct hardware access of a Cdd to the microcontroller.

Note
In the following only dealing with Cdds is described in detail. The advantage of this approach is that the Cdd SWC can be designed completely as SWC and is more flexible.
Therefore, only the term **SWC** will be mentioned in the text beyond for a Cdd or IoHwAb.

2 Basics on SWCs

The interfaces between SWCs are defined formally. The AUTOSAR Interface is used hereby. It is application-specific and generated by Rte. The AUTOSAR Interface of a SWC is defined via ports and enables the data exchange between SWCs or between a SWC and a Cdd / IoHwAb.

Generally, the following ports can be used for communication between SWCs:

- Sender / Receiver Ports
- Client / Server Port
- Parameter Ports
- Non-volatile Data Ports
- Trigger Ports
- Mode Switch Ports

In the following only **sender/receiver** and **client/server ports** are described in more detail, because they are most relevant for the implementation of Cdds and IoHwAbs.

2.1 Sender/Receiver Ports

Sender/Receiver Ports are used to specify the output or input data of a SWC. Data is defined by so-called data elements of a Sender/Receiver Interface. In a simplified way, the data element principle can be seen as follows:

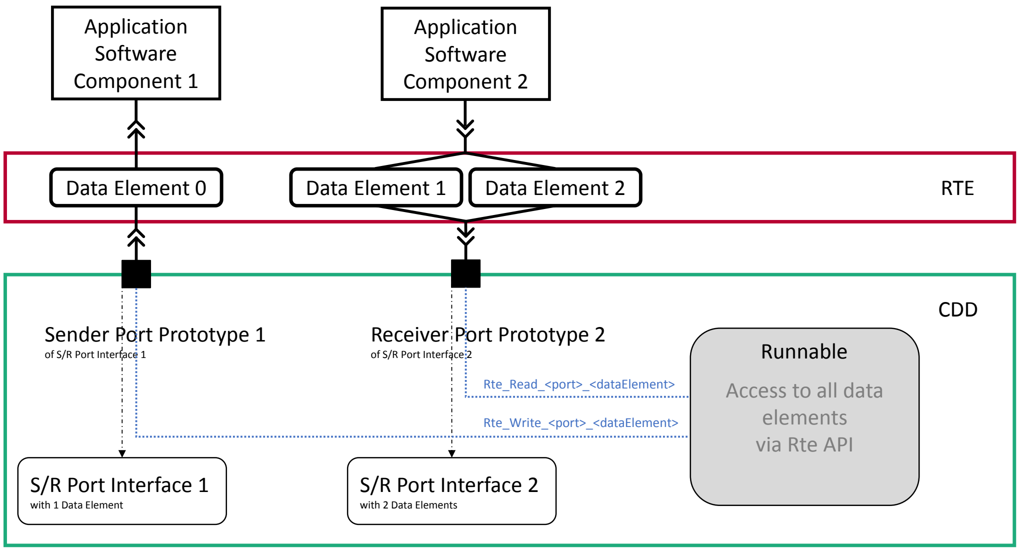


Figure 2-1 Sender-Receiver Concept

Data Elements are defined within the Rte and can be accessed via Rte APIs. Only the part of the Cdd is shown in a more detailed point of view. The application software component must own the counterpart of the port – a Receiver Port Prototype 1 (of S/R Port Interface 1) and a Sender Port Prototype 2 (of S/R Port Interface 2).

2.2 Client/Server Ports

Client/Server Ports can be used to specify the required (client) or provided (server) services of a SWC.

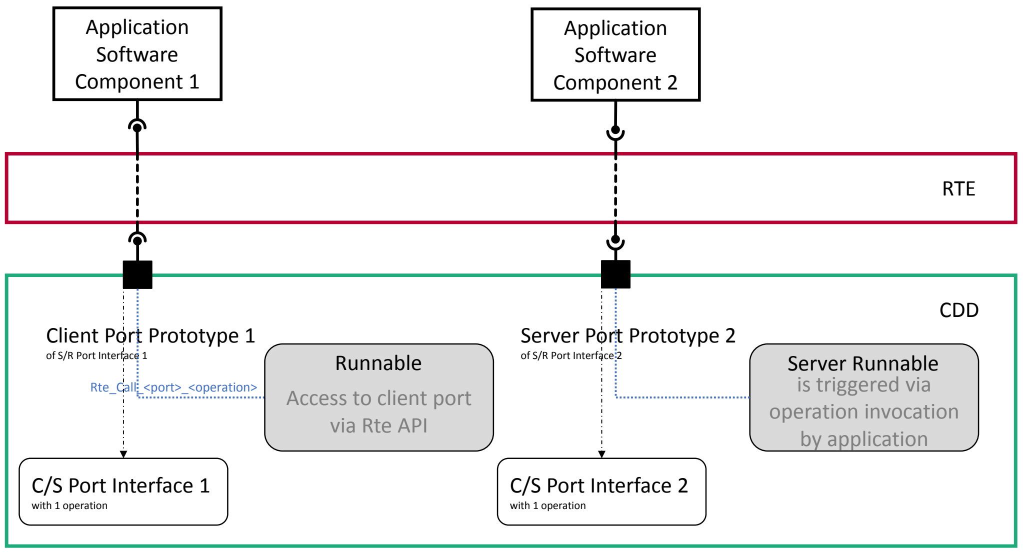


Figure 2-2 Client/Server Concept

Note
Before defining a port prototype an underlying port interface must be created. This concerns Client/Server Ports as well as Sender/Receiver Ports.

Note
A port prototype can be seen as an instance of a specific port interface.

2.3 Runnable Entities

The implementation architecture of a SWC is formally defined via **runnable entities**. They correspond to procedures / sequence of instructions which are started by the Rte – i.e. they are executed if a specific event occurs, e.g. a periodic activation, the reception of a new input value, operation invocation, etc... and run in the context of a OS task.

For each operation of a provide port a runnable entity has to be added / created. A client port, a sender port and a receiver port can only be used within a runnable entity and so port accesses have to be defined for runnable entities.

It is not allowed to call Rte functions from interrupt context (CAT2 and CAT1 ISR). CAT2 ISRs have access to OS functions – e.g. to set events. However, in a CAT1 ISR, it is not allowed to call functions of the Os or the Rte.

Note
It is possible to configure runnable entities which run cyclically, on data reception, on mode change or transition, etc...

These runnable entities can also be used to write or read data on Sender/Receiver Ports or do operation invocations on Client Ports.

Additionally, a so-called **Init Runnable** can be configured if some initialization stuff must be done.

2.4 Exclusive Areas

An Exclusive Area prevents interruption of itself. Runnable entities can be configured to enter a specific exclusive area (realized by an API call within the runnable entities) or that the Exclusive Area has already been started before the runnable entity is started. How the Exclusive Area is realized by the RTE depends on the configuration of the project (e.g. might even be optimized to nothing if runnable entities cannot interrupt each other).

If a dedicated mechanism of the Exclusive Areas is required (e.g. SuspendAllInterrupts() / ResumeAllInterrupts()), create an container **RteExclusiveAreaImplementation** in /MICROSAR/Rte/RteSwComponentInstance and configure it with **RteExclusiveAreaImplMechanism** and make sure the parameter **RteExclusiveAreaOptimization** is created and set to *false*.

2.5 Interrupt Propagation

A runnable entity can be configured which will be triggered by the Rte automatically if an external trigger like an interrupt of the MCAL has been reached. An event has to be set from the interrupt / notification function via OS function SetEvent.

3 Exemplary Configuration with Vector Tools

To create a fully runnable SWC both DaVinci tools are necessary – DaVinci Developer and DaVinci Configurator Pro (version 5.x).

3.1 Using DaVinci Developer

Create a Sender/Receiver and a Client/Server port interface in context of **Application Port Interfaces**:

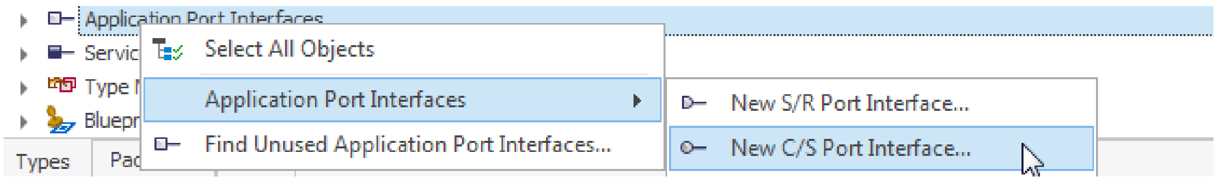
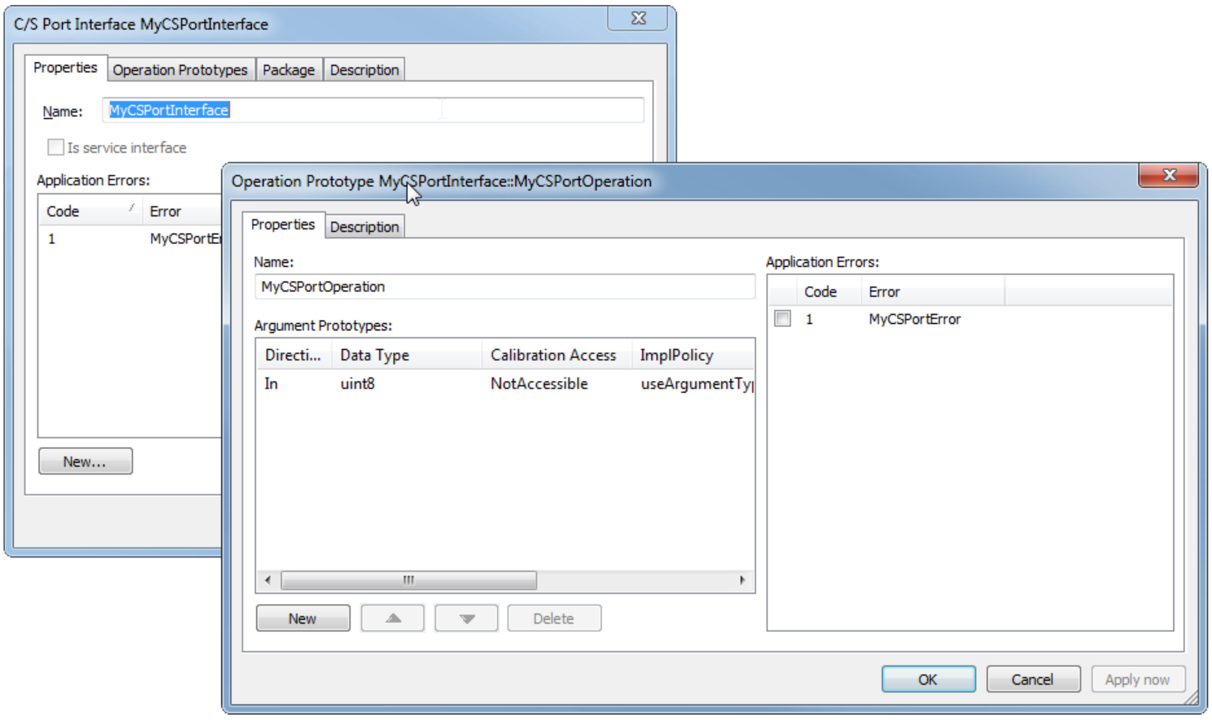

Application Port Interfaces
Servic
Type
Bluepr
Types
Pac
Select All Objects
Application Port Interfaces
Find Unused Application Port Interfaces...
New S/R Port Interface...
New C/S Port Interface...

Figure 3-1 Creation of Port Interfaces in DaVinci Developer

Specify at least one **operation** (in context of **Operation Prototypes**) of a Client/Server Port Interface (Figure 3-2) and one **data element** (in context of **Data Element Prototypes**) in a Sender/Receiver Port Interface (Figure 3-3).



Name: MyCSPortInterface

Application Errors:

Code	Error
1	MyCSPortE

Name: MyCSPortOperation

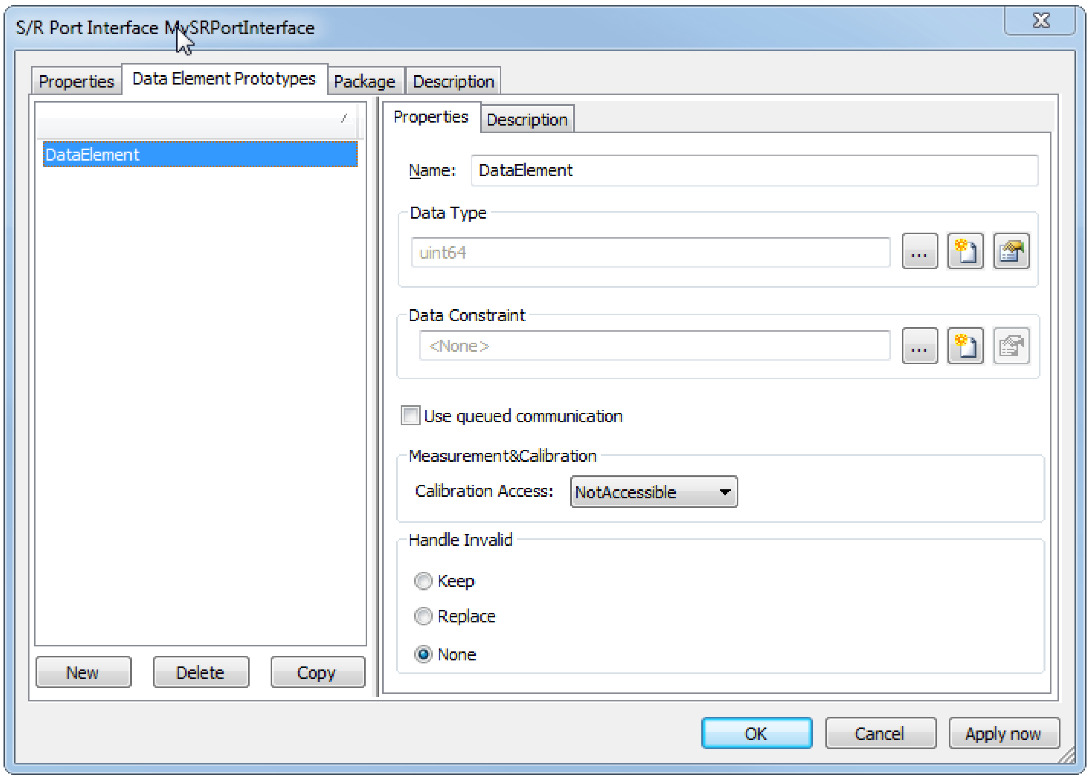
Argument Prototypes:

Directi...	Data Type	Calibration Access	ImplPolicy
In	uint8	NotAccessible	useArgumentTy

Application Errors:

Code	Error
1	MyCSPortError

Figure 3-2 Configuring a Client/Server Operation of Client/Server Port Interface


S/R Port Interface MySRPortInterface

Properties | Data Element Prototypes | Package | Description

DataElement

Properties | Description

Name: DataElement

Data Type: uint64

Data Constraint: <None>

Use queued communication: ☐

Measurement&Calibration — Calibration Access: NotAccessible

Handle Invalid: Keep ○ Replace ○ [✓ None]

New | Delete | Copy

OK | Cancel | Apply now

Figure 3-3 Configuring a Data Element of Sender/Receiver Port Interface

Note
If necessary, additional properties like Data Types, Data Constraints, Handle Invalid, etc. (Sender/Receiver Interface) or Argument Properties (Client/Server Interface) can be set which is not part of that documentation.

Create a new **Application Component Type** as seen in Figure 3-4.

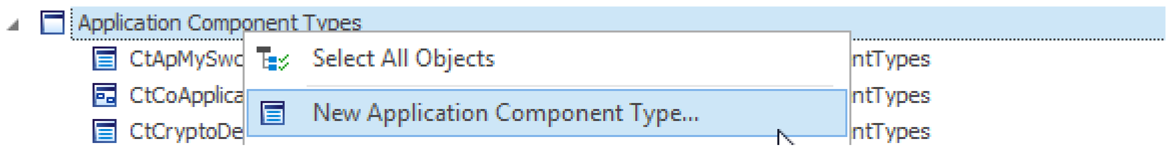
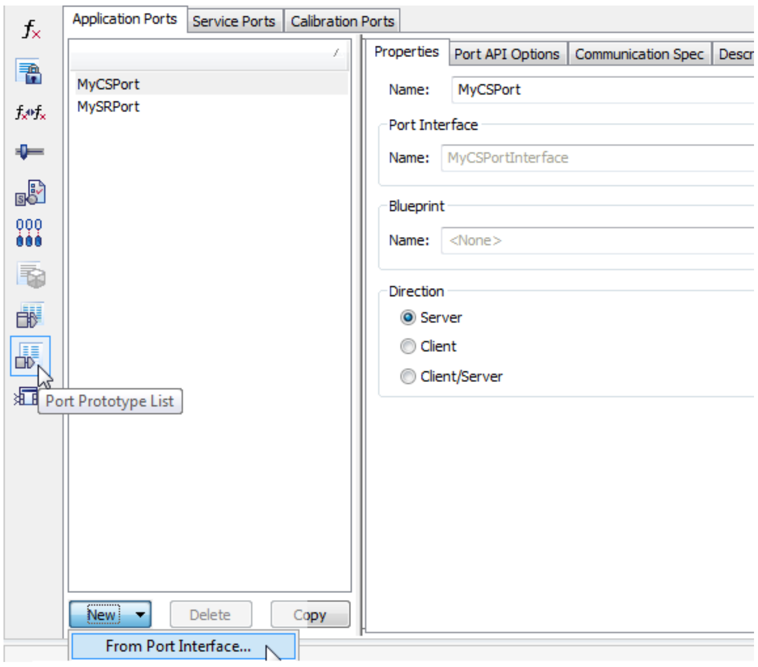


Figure 3-4 Creation of an Application Component in DaVinci Developer

Subsequently, select the type of the component – a **Complex Driver (application layer)**.

Open the newly created component and add a new **Application Port Prototype** from the port interface within the view **Port Prototype List** (Figure 3-5).

Note
The Client/Server Port was configured to act as a server and the Sender/Receiver Interface was configured as a sender.



MyCSPort
MySRPort

Name: MyCSPort

Port Interface
Name: MyCSPortInterface

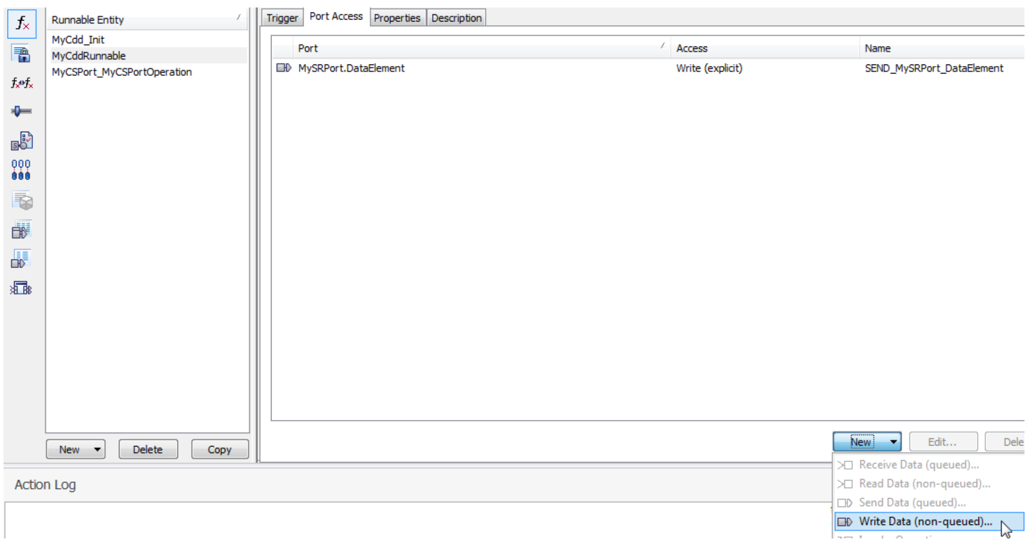
Blueprint
Name: <None>

Direction
[✓ Server]
Client
Client/Server

Figure 3-5 Creation of Port Prototypes from Port Interfaces

After changing to view “Runnable Entity List” one **Server Runnable**, one **Runnable** and one **Init Runnable** were configured:

> For the server runnable, the just now created Server Port Prototype **MyCSPort** was selected as unhandled server port. The associated trigger is created automatically.
> The normal runnable entity **MyCddRunnable** was created with a periodic trigger. This runnable entity shall cyclically update the data element of the sender port. So, the appropriate data element has to be added as Port Access as seen in Figure 3-6.
> The init runnable was added and the appropriate init trigger is set automatically.



Runnable Entity
MyCdd_Init
MyCddRunnable
MyCSPort_MyCSPortOperation

Port	Access	Name
MySRPort.DataElement	Write (explicit)	SEND_MySRPort_DataElement

Figure 3-6 Assigning a Port to have Access to a Sender Data Element.

In case of a desired **propagation** of an arrived **interrupt** to an application above the Rte, the following steps has to be performed:

> Create an additional **Application Port Interfaces** via **New Trigger Interface…**
  - Create a trigger element for this trigger interface
> Create a new **Application Port Prototype** of the newly created trigger interface
  - The port shall receive the trigger and has therefore be configured as a **Receiver**
> Create a new **Runnable** and select as **Trigger**: "**On External Triggering Occurred…**" and choose the created trigger of the receiver port prototype.

Caution
It is assumed that the appropriate counterpart of the newly created SWC does already exist.

The creation of the counterpart is not part of this documentation

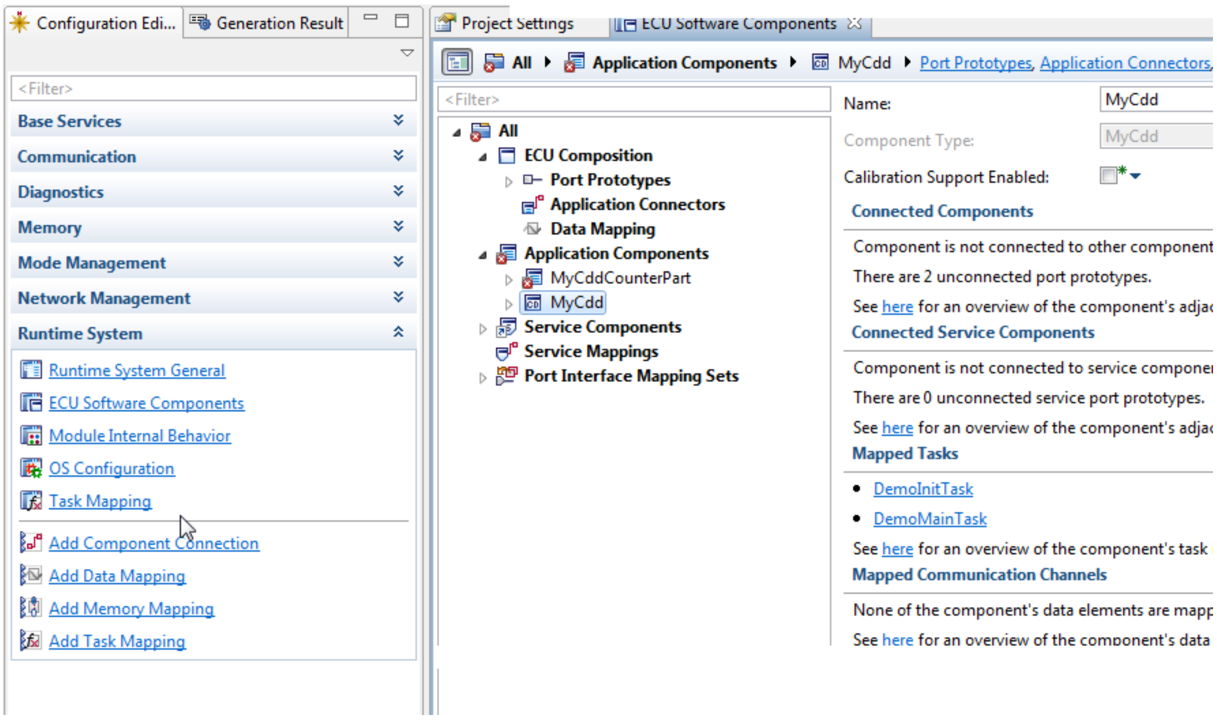
3.2 DaVinci Configurator Pro

The final connection of the newly created SWC with the still existing counterpart is done in within the DaVinci Configurator Pro.

At first the new component has to be added to **Application Components** within the comfort editor as seen in Figure 3-7 via **ECU Software Components**. The counterpart **MyCddCounterPart** is already available, but unconnected.

Additionally, all triggered **runnable entities** (**MyCdd_Init** and **MyCddRunnable** (and possibly the runnable entity which has to be triggered by an interrupt) has to be mapped to a valid OsTask.

Note
The documentation on how to configure the MICROSAR OS is not part of this documentation. Take a look at the technical reference of MICROSAR OS.



Name: MyCdd

Component Type: MyCdd

Calibration Support Enabled:

Connected Components

Component is not connected to other component

There are 2 unconnected port prototypes.

See here for an overview of the component's adja

Connected Service Components

Component is not connected to service compone

There are 0 unconnected service port prototypes.

See here for an overview of the component's adja

Mapped Tasks

- DemoInitTask
- DemoMainTask

See here for an overview of the component's task

Mapped Communication Channels

None of the component's data elements are mapp

See here for an overview of the component's data

Figure 3-7 Adding the New Component to Application Components

The port prototypes of the SWCs have to be connected by using the Component Connection Assistant (**Add Component Connection** in Figure 3-7).

Select **Application connector prototypes** as connector type and then the newly created SWC **MyCdd**. Now choose all ports which should be connected. If both sides of SWCs / ports have a matching configuration the automatic matching feature within that assistant should work and provide the appropriate connections.

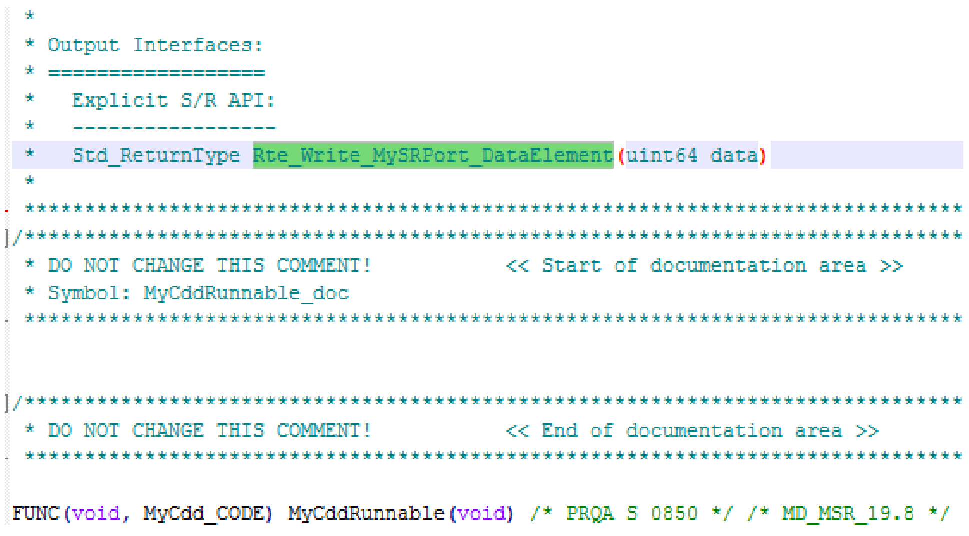
The software component template has to be generated which can be then filled with ECU specific implementation within the so-called user section as seen in Figure 3-8.

```
FUNC(void, MyCdd_CODE) MyCSPort_MyCSPortOperation(uint8 port_argument) /* PRQA S 0850 */ /* MD_MSR_19.8 */
{
/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << Start of runnable implementation >>             DO NOT CHANGE THIS COMMENT!
 * Symbol: MyCSPort_MyCSPortOperation
 *********************************************************************************************************************/

 /* Add here your ECU specific implementation */

/**********************************************************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << End of runnable implementation >>               DO NOT CHANGE THIS COMMENT!
 *********************************************************************************************************************/
}
```

Figure 3-8 Example of Generated Software Component Template.

Directly above of each runnable entity all accesses to data elements or operation invocations via Rte of the corresponding runnable entity are listed in comments. This can be applied to make sure all port accesses were configured correctly.

For example, the runnable entity **MyCddRunnable** is configured to have a write access to the data element of the underlying Sender/Receiver Port Interface via port prototype **MySRPort** as seen in the Figure 3-9 and the API to be used is mentioned in the function header.



```
 *
 * Output Interfaces:
 * ==================
 *   Explicit S/R API:
 *   -----------------
 *   Std_ReturnType Rte_Write_MySRPort_DataElement(uint64 data)
 *
 **********************************************************************************
/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << Start of documentation area >>
 * Symbol: MyCddRunnable_doc
 **********************************************************************************


/**********************************************************************************
 * DO NOT CHANGE THIS COMMENT!           << End of documentation area >>
 **********************************************************************************

FUNC(void, MyCdd_CODE) MyCddRunnable(void) /* PRQA S 0850 */ /* MD_MSR_19.8 */
```

Figure 3-9 Example of Generated Comments to Rte APIs

Note

If interrupt propagation is used, the OS function **SetEvent** has to be called from the notification function with appropriate TaskID and EventId.

The OS event for that trigger can be found in the generated Rte report (Rte.html) in the chapter **Trigger List**. Take a look at **ExternalTriggerEvent** of the configured runnable entity and use the referenced Task and Event to set the event within your notification:

```
SetEvent(DemoTask, Rte_Ev_Run1_MyCdd_MyCddExtTriggeredRunnable);
```

Summary / Practical Procedure

One possible workflow for SWC designing could be:

DaVinci Developer

1. Create Port Interfaces
   - With data element(s) in Sender/Receiver Interfaces
   - With operation(s) in Client/Server Interfaces
2. Create Application Component Type
3. Assign port interfaces to component types by creating application ports
   - select whether the port is a sender or a receiver of the corresponding data of this port
   - select whether the component provides services with this port (sender) or requires services with this port (client)
4. Create the necessary runnable entities of the component
   - at least one (server) runnable entity for each server operation
   - at least one runnable entity if Sender/Receiver or client ports are used with the appropriate port access and trigger condition.

(Assumption: The counterpart of the SWC still exists)

DaVinci Configurator Pro

5. Add the component connection between the newly created component and the existing counterpart (application connection)
6. Generate BSW (Rte) and Software Component Templates

Any IDE

7. Add ECU specific implementation in the user sections of the generated software component templates.

4 Abbreviations

Abbreviation	Description
Adc	Analog Digital Converter (unit)
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
Cdd	Basic Software Module
Dio	Digital Input Output (unit)
ECU	Electronic Control Unit
Eep(ROM)	Electrically Erasable and Programmable Read Only Memory
IoHwAb	Input / Output Hardware Abstraction
ISR	Interrupt Service Routine
MCAL	Microcontroller Abstraction Layer
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Os	Operating System
Rte	Runtime Environment
SW-C, SWC	Software Component

Table 1 Abbreviations

5 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.