ETAS

FlexRay Editor
User's Guide

DRIVING | EMBEDDED EXCELLENCE

Copyright

V2.1.0 R01 EN - 06.2019

Contents

1 Introduction

This manual addresses qualified personnel working in the fields of automobile ECU development and testing. Specialized knowledge in the areas of measurement and ECU technology is required

1.1 About this Manual

This manual contains information about working with FlexRay Editor. The manual contains the following chapters:

- **"About FlexRay Editor" on page 6**

  This chapter contains a description of the FlexRay Editor

  – "FlexRay Modules (Network Integration FlexRay)" on page 6
- **"Working with FlexRay Editor" on page 7**

  This chapter describes how to work on FlexRay Modules.

  – "The FlexRay Editor" on page 7

  – "Import a FlexRay module in COSYM" on page 9

  – "Creating the Necessary Files with EB tresos Busmirror" on page 11

  – "For details of how to create these files, refer to the EB tresos Busmirror documentation." on page 11

  – "Linking to the EB tresos bmc Tool Suite" on page 12

  – "Module Configuration" on page 13

  – "With the "RTPC Compile Optimization Level" switch, the optimization level of the Build environment on the Real-Time PC can be determined. This option corresponds to the optimization level that can be set via the Real-Time PC Web Interface but only applies to the generated source code of the FlexRay module." on page 15

  – "User-Defined Code and Adaptations" on page 15

  – "Linking to the ETAS Bus Communication Monitor (BCM)" on page 22

  – "Target User Modules" on page 25

  – "Experiment Environment" on page 28

  – "Creating a FlexRay Module with the Automation Server" on page 32

1.2 Using this Manual

Presentation of Information

All actions to be performed by the user are presented in a so-called "use-case" format. This means that the objective to be reached is first briefly defined in the title, and the steps required to reach the objective are then provided in a list. This presentation looks as follows:

Definition of Objective

Any preliminary information...

- Step 1

  Any explanation for Step 1...

- Step 2

  Any explanation for Step 2...
- Step 3

  Any explanation for Step 3...

Any concluding remarks...

Specific example:

To create a new file

When creating a new file, no other file may be open.

- Choose **File** → **New**.

  The "Create file" dialog box is displayed.

Type the name of the new file in the "File name" field.

The file name must not exceed 8 characters.

- Click **OK**.

The new file will be created and saved under the name you specified. You can now work with the file.

Typographic Conventions

The following typographic convention are applied:

Choose **File** → **Open**.	Menu options are printed in bold, blue characters.
Click **OK**.	Button labels are printed in bold, blue characters.
Press <ENTER>.	Key commands are printed in small capitals enclosed in angle brackets.
The "Open file" dialog box appears.	The names of program windows, dialog boxes, fields, etc. are enclosed in double quotes.
Select the `setup.exe` file.	Text strings in list boxes, in program code and in path and file names are printed using the `Courier` font.
A conversion between Logic and Arithmetic data types is *not* possible.	Emphasized text portions and newly introduced terms are printed in an *italic* font face.

Important notes for the users are presented as follows:

Note

Important note for users.

2 About FlexRay Editor

This chapter contains a description of the FlexRay Editor

The individual sections contain information on:

2.1 FlexRay Modules (Network Integration FlexRay)

FlexRay Modules (Network Integration FlexRay) is commonly called as COSYM-NIF which is an add-on to COSYM V2.1. It enables testing of ECU functions incorporating FlexRay communication.

The entire FlexRay bus configuration is read in from a data model which has been created previously using software from the company Elektrobit called EB tresos Busmirror. FlexRay Editor then automatically creates source code for the residual bus to be simulated in the form of a FlexRay module. User-specific code can be added to this.

The signals of the FlexRay module which correspond to the signals on the FlexRay bus are available in the Connections view and can be connected with signals of other modules there.

Note

FlexRay Editor *uses the "StringTemplate.NET" and "ANTLR" libraries, please read the license conditions detailed in the COSYM installation guide.*

Hardware Requirements

To be able to run FlexRay bus simulations, you need one or more PCI boards of the type EB 5100 or EB 5200 from the company Elektrobit Automotive on the Real-Time PC simulation target. Mixed operation is also possible – one board can be addressed per FlexRay module.

You can also order this FlexRay interface from ETAS – the following table contains the order data:

Order Name	Order Number
EB 5100 Elektrobit FlexRay Interface Solution	F-00K-106-407
EB 5200 Elektrobit FlexRay Interface Solution	F-00K-108-467

License details

The below listed license is required to work with FlexRay modules:.

- COSYM_NIF

3 Working with FlexRay Editor

This chapter describes how to work on FlexRay Modules.

The individual sections contain information on:

3.1 The FlexRay Editor

This section contains the information on working with the FlexRay Editor.

3.1.1 To create a FlexRay module

- Go to COSYM and open a project.

  The Project dashboard is displayed as shown in the image below.

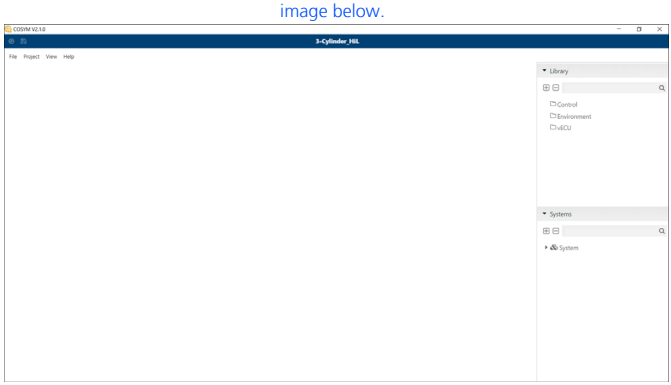
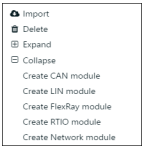


Fig. 3-1 Project dashboard in COSYM

- Go to **View** menu and click **Deployment Management**.
  <or>
  Click **Edit Target** in **Deployment view** in the **System editor**.

  Deployment Management is displayed.
- Right-click on a target.

  A context menu is displayed as shown in the image below.



- Click **Create FlexRay Module**.

  "Create FlexRay Module" dialog box is displayed.
- Type the name to display in COSYM in the "Name in COSYM" text box.

- Click **Save & Edit**.

  FlexRay Editor is opened for creating a FlexRay module.

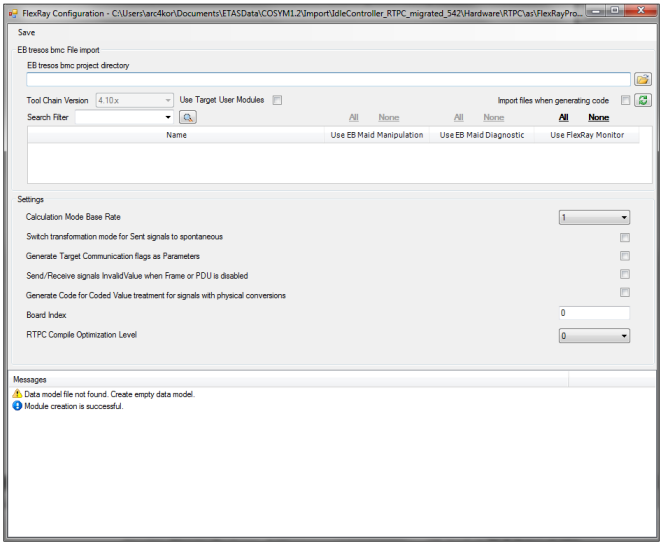


Fig. 3-2 The FlexRay Editor

3.2 Import a FlexRay module in COSYM

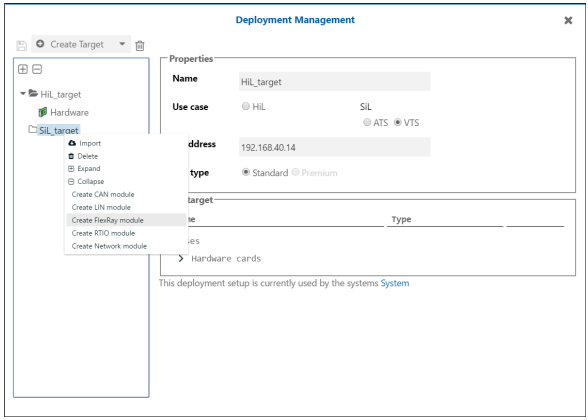
This section describes how to import a FlexRay module (.lmd) in COSYM.

To import a FlexRay module

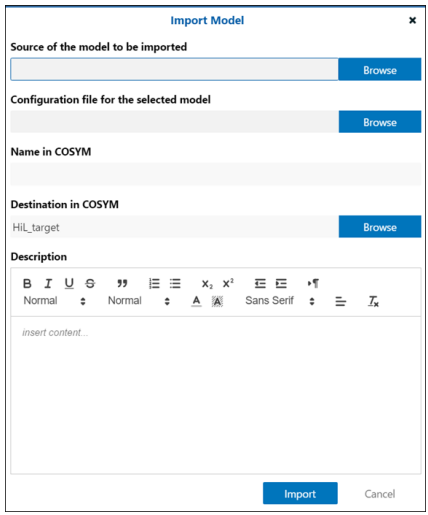
- Go to **View** menu in the Project dashboard (see Fig. 3-1) and click **Deployment Management.**
  <or>
  Click **Edit Target** in **Deployment view** in the **System editor**.

  Deployment Management is displayed.

- Right-click on a target.
  A context menu is displayed.



- Click **Import.**
  The "Import Module" dialog box opens.



- Select the path of the FlexRay module which is to be imported or click **Browse** to select it.

- Click **Browse** to select the destination path in COSYM, if you want to change the target.
- Enter the description which is optional.
- Click **Import**.

  The selected FlexRay module is imported into COSYM.

Note

"Name in COSYM" field is disabled and "Configuration file for the selected model" field is not applicable for importing a FlexRay module.

3.2.1 Edit a FlexRay module

This section describes how to edit a FleyRay module by opening the FlexRay Editor through COSYM.

To edit a FlexRay module

- Go to **View** menu in the Project dashboard (see Fig. 3-1) and click **Deployment Management.**
  <or>
  Click **Edit Target** in **Deployment view** in the **System editor**.

  Deployment Management is displayed.
- Expand the arrow to view the modules in a target.
- Double-click on a FlexRay module to edit.

Note

Only one instance of FlexRay Editor can be opened at a time. If an instance of FlexRay Editor is already running, then close the current instance to open another instance.

The FlexRay Editor is displayed for editing**.**

- Edit/modify the required changes in FlexRay Editor and click **Save** to save the changes made.
- Once you save the changes made in the editor, the same will be reflected in COSYM.
- Close the editor.

3.3 Creating the Necessary Files with EB tresos Busmirror

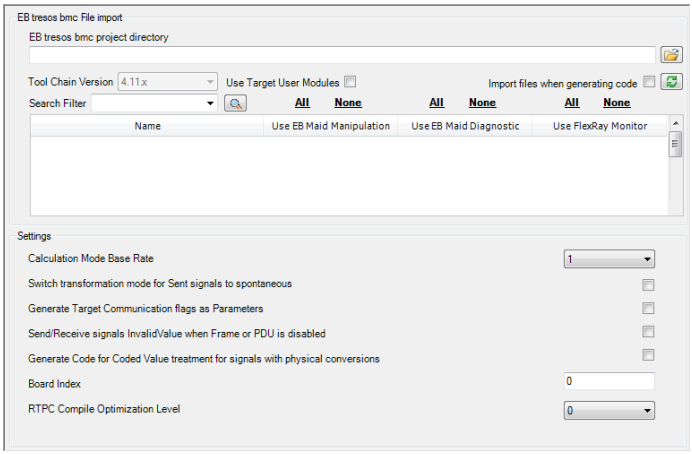
For the integration of the FlexRay bus configuration, the data model and the binary files that can be run on the EB5100 or EB5200 boards have to be imported from an EB tresos Busmirror project.

- `BMCfg.tdb`
- `Firmware.ttc`

For details of how to create these files, refer to the EB tresos Busmirror documentation.

3.4 Linking to the EB tresos bmc Tool Suite

The window of a FlexRay module is divided into two sections: "EB tresos bmc File import" and "Settings".



- In the "EB tresos bmc project directory" field select the EB tresos project whose data model and executable binary files you want to import into the FlexRay module.
- Save the FlexRay module.

Synchronizing EB tresos bmc Files and Generating FlexRay Module Source Code

The source code is always generated whenever the module is saved or the external files are synchronized. Working copies of the imported EB tresos bmc files are generally accessed when source code for the FlexRay module is generated. Thanks to a synchronization mechanism, these external files can be re-imported into the FlexRay module.

This procedure can be automated:

- The automatic synchronization is enabled with the "Import files when generating code" option.
  - Enabled: The files are always synchronized before code is regenerated. The code is then generated based on the synchronized files.

– Disabled: The files are only synchronized manually. The manual synchronization and the subsequent code generation are then run manually via the button.

Note

If user code that has been generated manually is used in the FlexRay module, it is not integrated into the FlexRay module source code until the code is generated.

Integration of Target User Modules (EB MAID)

EB tresos bmc makes it possible to integrate additional functionality in the executable binary file using Target User Modules (TUM). The interfaces of these TUMs are stored in TUM descriptions. These TUM descriptions can be used to generate additional source code for the FlexRay module enabling communication with the TUM during runtime.

- Enable the support of TUMs with the option "Use Target User Modules".

Note

If this is disabled, the TUM description is ignored and there is no code generation for the EB MAID support.

3.5 Module Configuration

The window of an FlexRay module is divided into two sections: "EB tresos bmc File import" and "Settings". Edit the relevant options in the "Settings" field and save the FlexRay module.

Adapting the Clock Time of the FlexRay Module on the Real-Time PC

The clock time of the SendReceive task is generally 1 ms. The necessary calls for complete signal exchange between the FlexRay module and the application on the board are distributed evenly over five calculation slots.

Each time the SendReceive task is executed, a single calculation slot is run although its share of the overall exchange is synchronized between the FlexRay module and the board. The calls for exchanging the signals of an individual PDU also modulated with the FlexRay Cycle Repetition of the corresponding FrameTriggerings ensuring that the Real-Time PC and the board are utilized as evenly as possible.

This means that the entire signal exchange is completed after a total of 5 ms. Changing the clock time can result either in optimized performance (clock time > 1 ms) or in an increase in synchronization (clock time < 1 ms).

- In the "Settings" field, select the divisor for the period under "Calculation Mode Base Rate".

  Possible values are: 1/8, 1/4, 1/2. 1, 2, 4 and 8.

  1/8 = increased performance (effectively 8 ms)

  8 = increased synchronization (effectively 125 µs)

Note

The actual execution of read and send communication for a frame or PDU takes place in whole-number multiples of the clock time of the FlexRay module. The relevant clock time of a frame or PDU is taken from the underlying data model.

Optimizing Performance and Avoiding Rival Signal Values

To minimize the number of times the functions for exchanging send signals are called, these are only run as necessary when the value of the signal to be sent has changed since the last call. This can optimize performance.

- Select this option by enabling "Switch transformation mode for Sent signals to spontaneous".

Generating the Target Communication Flags as a Parameter

To control communication for frames or PDUs or to control bus communication, both an "Enable" and "Idle" control signal are generated per frame or PDU; "CommEnable" and "HardBoundaries" are generated for bus communication. This option is used to select whether these signals are generated as inports or as parameters.

The signals for the output of the timestamp of the last change of a receive PDU are always generated as an outport.

InvalidValue Signal with a Disabled Payload

If a frame or PDU has been disabled during runtime via the "Enable" signal, this option controls whether in this case the "InvalidValue" from the FlexRay data model should be sent or received as a substitute value for each signal.

Support of symbolic values in the case of signals with scaling

If a FlexRay signal has scaling for converting the transmitted raw value into physical variables, "symbolic values" are often used (e.g. to transmit an error state). These symbolic values must not be subjected to scaling in the process.

To be able to distinguish a scaled physical value from an unscaled symbolic one, additional inports and outputs can be generated with which this behavior can be controlled at execution time.

- To run this code generation, activate the option "Generate Code for Coded Value treatment for signals with physical conversions".

Supporting Several EB5100 or EB5200 in an Real-Time PC

Several independent FlexRay modules can also be generated – the relevant EB5100 or EB5200 is identified via the board index.

- Enter the index under "Board Index".

Compile Optimization Level

With the "RTPC Compile Optimization Level" switch, the optimization level of the Build environment on the Real-Time PC can be determined. This option corresponds to the optimization level that can be set via the Real-Time PC Web Interface but only applies to the generated source code of the FlexRay module.

3.6 User-Defined Code and Adaptations

As an extension to the FlexRay module source code generated automatically from the data model, manually created source code can also be integrated. The code fragments to be inserted are injected at the relevant points of the generated code using dummies.

Files for Code Fragments

The FlexRay module directory contains a subdirectory for integrating manually created source code:

```
..\<module_name>\UserCode\*.fragment.c
```

Every file in this directory that has the file ending `.fragment.c` is evaluated as a code fragment. The format of these files is as follows:

```
//[Code location Indicator 1]
<user code>
//[Code location Indicator 2]
<user code>
…
```

Note

If manually created user code is used in the FlexRay module, this is not integrated into the FlexRay module source code until code is generated.

Code Location Indicator

The start of a code fragment is specified by defining the code location indicator. The code location indicator is enclosed in square brackets and specified as a comment line in the source code.

The code location indicator is followed by the source code to be integrated that is inserted into the FlexRay module by the code generator. The code ends before the next code location indicator – the format of the code location indicator takes the form of one of the following variants:

Specific Code Location Indicators with Global Context

```
//[<CodePoint>]
```

`CodePoint` is a text marker in the generated source code at which the user code is inserted. There are global text markers which only appear once in the generated source code. No further information specifying the location has to be entered for these.

Valid text markers for `CodePoint` in this variant are:

```
//[declarations]
//[beforeInit]
//[afterInit]
//[beforeSend]
//[afterSend]
//[beforeReceive]
//[afterReceive]
//[beforeExit]
//[afterExit]
```

Specific Code Location Indicators with PDU and Signal Context

For other text markers, there are several instances, e.g. per payload or per signal. For these text markers, further identifiers must be specified to indicate the context and object to which the code location indicator refers:

```
//[<CodePoint>:<context>:<object>]
```

Valid text markers for `CodePoint` in this variant are:

```
//[before:context:object]
//[beforeUnlock:context:object]
//[beforeEncode:context:object]
//[beforeSend:context:object]
//[afterReceive:context:object]
//[afterDecode:context:object]
//[afterLock:context:object]
//[after:context:object]
```

The `context` of the text marker can refer to the following operations in the program:

```
SendSignal
ReceiveSignal
SendPdu
ReceivePdu
```

Finally the `object` to which the text marker refers has to be specified. Names of signals and/or PDUs can be used to specify the `object`.

Note

A list of all names is generated in the `SignalList.txt` and `PduList.txt` files in the `UserCode` directory. All valid code location indicators are inserted into the FlexRay module source code as commented text markers in code generation.

Specific Code Location Indicators with TUM Context

For objects that run different functions at different times, a further level of refinement can be used.

This is currently only supported for EB Maid Target User Modules sending "ToHost".

```
//[<CodePoint>:<context>:<object>:<function>]
```

Valid text markers for `CodePoint` in this variant are:

```
//[afterReceive:context:object:function]
```

The `context` of the text marker can currently only refer to the following operations in the program:

```
ReceiveTum
```

Then the `object` the text marker refers to must be assigned. The name of the TUM must be used as specification of the `object`. Then the `function` currently being run by the TUM has to be specified.

The table below shows the valid combinations of `codepoint`, `context` and `function` exist:

		context				
		Send-Signal	**Receive-Signal**	**Send Pdu**	**Receive-Pdu**	**Receive-Tum**
code-point	before	x	x	x	x	
	beforeUnlock			x	x	
	beforeEncode	x				
	beforeSend	x				
	afterReceive		x			x
	afterDecode		x			
	afterLock			x	x	
	after	x	x	x	x	

Tab. 3-1 Valid Combinations of `codepoint`, `context` and `function`

Example:

```
//[declarations]
int counter = 0;
```

The specified code location indicator refers to source code which is inserted after the declaration of the global variables of the FlexRay module. In the above example, a `counter` variable is declared.

```
//[before:SendPdu:Node2_nCommTask1Invocations_I0_ChA]
counter++;
```

The specified code location indicator refers to source code which is run before the specified PDU is sent. The user code increments the `counter` variable declared previously.

Code Location Indicators and Program Flow

This section describes where exactly in the program flow code location indicators are inserted. Names in angled brackets `<text>` refer to corresponding dummies.

Global context:

```
<NIF Module declarations>
//[declarations]

void <modulename>_Init() {
  //[beforeInit]
  <Flexray Init Code>
  //[afterInit]
}

void SendReceive() {
  //[beforeSend]
  <Generated send code>
  //[afterSend]
  //[beforeReceive]
  <Generated receive code>
  //[afterReceive]
}

void <modulename>_Exit() {
  //[beforeExit]
  <Flexray Exit Code>
  //[afterExit]
}
```

SendSignal context:

```
//[before:SendSignal:<<signalname>>]
<retrieve physical value from inport>
//[beforeEncode:SendSignal:<<signalname>>]
<code value from physical to implementation type>
//[beforeSend:SendSignal:<<signalname>>]
<send implementation value to Bus>
//[after:SendSignal:<<signalname>>]
```

ReceiveSignal context:

```
//[before:ReceiveSignal:<<signalname>>]
<receive implementation value from Bus>
//[afterReceive:ReceiveSignal:<<signalname>>]
<decode value from implementation to physical type>
//[afterDecode:ReceiveSignal:<<signalname>>]
<write physical value to outport>
//[after:ReceiveSignal:<<signalname>>]
```

SendPDU context:

```
//[before:SendPdu:<<pduname>>]
< ... >
//[afterLock:SendPdu:<<pduname>>]
<send signals to Bus>
//[beforeUnlock:SendPdu:<<pduname>>]
< ... >
//[after:SendPdu:<<pduname>>]
```

ReceivePDU context:

```
//[before:ReceivePdu:<<pduname>>]
<lock PDU>
//[afterLock:ReceivePdu:<<pduname>>]
<receive signals from Bus>
//[beforeUnlock:ReceivePdu:<<pduname>>]
<unlock PDU>
//[after:ReceivePdu:<<pduname>>]
```

User-Defined Parameters, Measurement Variables, Inports and Outports

To be able to access values calculated in user-defined code, it is possible to extend the FlexRay module with additional parameters and ports. These are made accessible via the Experiment Environment (EE).

Parameters:

Parameters are defined in the following file:

```
..\<module_name>\UserCode\CalibrationVariables.h
```

A parameter is defined by one line:

```
<type> <name> = <value>;
```

Valid values for <type> are

- real64
- real32
- sint32
- uint32
- sint16
- uint16
- sint8
- uint8

It is also possible to declare hierarchical variables by using two underscores as separators. For example, the definition

```
real64 setting__var = 42.0;
```

generates the parameter

```
FlexRay_Bus/User/Measurement/setting/var
```

and initializes it with the specified value.

Note

If names of signals or PDUs from the data model of the EB tresos BMC are used, any double underscores contained are not interpreted as hierarchy levels!

The parameter is accessed in source code via the variable `<name>`.

Measurement variables:

Measurement variables are defined in the following file:

```
..\<module_name>\UserCode\MeasurementVariables.h
```

A measurement variable is defined by one line:

```
<type> <name>;
```

Inports:

Inports are defined in the following file:

```
..\<module_name>\UserCode\Inports.h
```

An inport is defined by one line:

```
TPortObj <module_name>_UserCode_inport_<name>;
```

The inport is accessed in source code via the variable `<name>`. Like variables, inports can also be declared hierarchically.

Outports:

Outports are defined in the following file:

`..\<module_name>\UserCode\Outports.h`

An outport is defined by one line:

`TPortObj <module_name>_UserCode_outport_<name>;`

The outport is accessed in source code via the variable `<name>`. Like variables and measurement variables, outports can also be declared hierarchically.

Example:

The following example shows how user-specific code can be used to add extra functionality to the FlexRay module.

The value of a specific signal

```
sint32 water_temperature
```

is to be read from the bus.

To create a measurement variable:

```
..\<module_name>\UserCode\MeasurementVariables.h

// Raw implementation value of water temperature
sint32 temperatures__raw_water_temperature;
```

To create the code fragment:

```
..\<module_name>\UserCode\temperature_measurement.fragment.c
//[afterReceive:ReceiveSignal:water_temperature]
// Retrieve raw signal from bus
temperatures__raw_water_temperature = val_impl.value_sint32;
```

This calls up the value of the signal last received and assigns it to the variable we declared.

As we want to use the value as soon as the signal is received, `afterReceive` is selected as CodePoint. Since the signal is read by the bus, the correct context is `ReceiveSignal`. The name of the signal for which the code has been inserted is `water_temperature` and comes from the file `Signals.txt`.

After further code generation, the new measurement variable is available in the Experiment Environment (CEE).

Note

If manually created user code is used in the FlexRay module, this is not integrated into the FlexRay module source code until code is generated.

3.7 Linking to the ETAS Bus Communication Monitor (BCM)

The ETAS Bus Communication Monitor enables manipulation of the signal flow within the FlexRay module for example to overwrite the values of a signal with a manually specified one before it is sent.

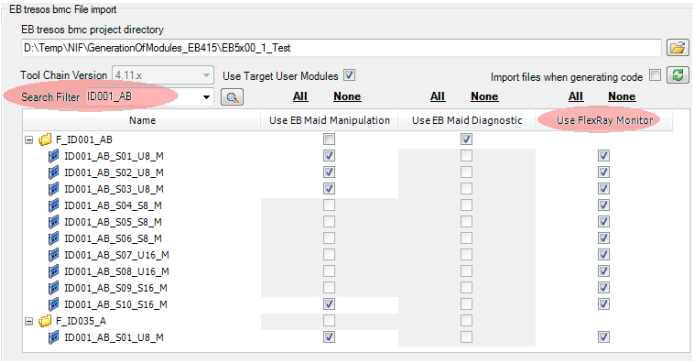
To optimize performance, the selection of signals for code generation to be taken into consideration for manipulation by the ETAS Bus Communication Monitor can be limited on the user interface.

The configuration of the ETAS Bus Communication Monitor is stored in a configuration file:

```
..\<project directory>\Target_RTPC\
        CModules\<module name>\FlexRayViewModel.xml
```

To display signals in the Bus Communication Monitor

- If, for example, you want to make individual signals accessible in the Bus Communication Monitor, you must activate them in the user interface.



Note

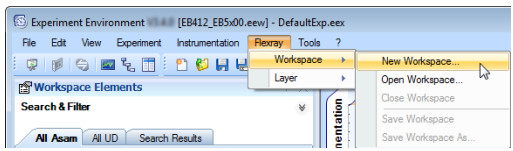
The Bus Communication Monitor only supports scalar signals and signals with a length of max. 32 bits. ByteArray signals or signals longer than 32 bits are not supported!

- To enable Bus Communication Monitor support for individual signals, enable the "Use for FlexRay Monitor" option in the user interface.
  - Specify a regular expression as search filter to limit the selection of signals displayed.

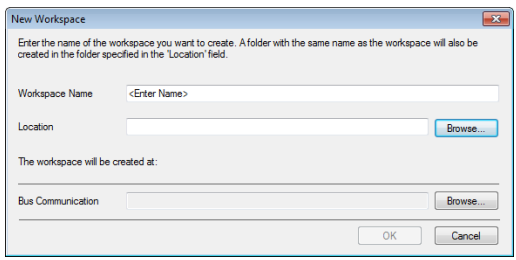
  - Using the "All" and "None" options, all signals which satisfy the search criterion can be enabled or disabled respectively.

    Enabling/disabling affects all instances of the selected signal.

- Save the module.
- Regenerate the code using **Project → Build**.
- To display the FlexRay elements, select **View → Flexray Elements**.
- To open the Bus Communication Monitor, select **View → Bus Communication Monitor**.
- Create a special FlexRay workspace using **Flexray → Workspace → New Workspace**.



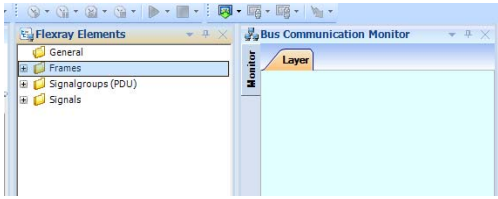
- Select a name and a location for the new workspace.
- Under "Bus Communication", select the file mentioned above `FlexRayMonitor.xml`.



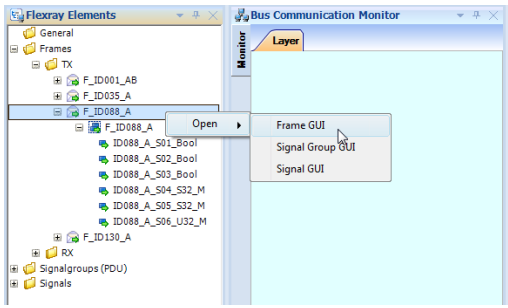
or

- Open an existing workspace (*.bcw) using **Flexray → Workspace → Open Workspace**.

  In Flexray Elements, the frames, signal groups and signals are displayed that have been enabled.

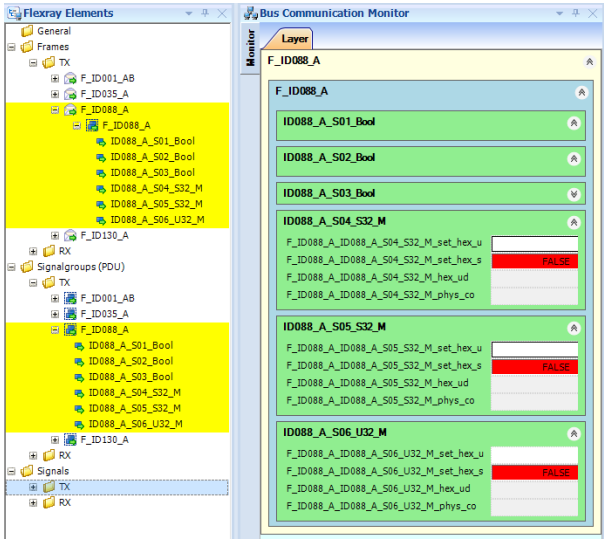


- In Flexray Elements, select the frame to which the signals shown in the Bus Communication Monitor belong.
- Right-click and select **Open → Frame GUI**.



  The frame is displayed in the current layer of the Bus Communication Monitor.

- Show the individual sections. As only three signals of the frame have been "released", these are also the only ones displayed.



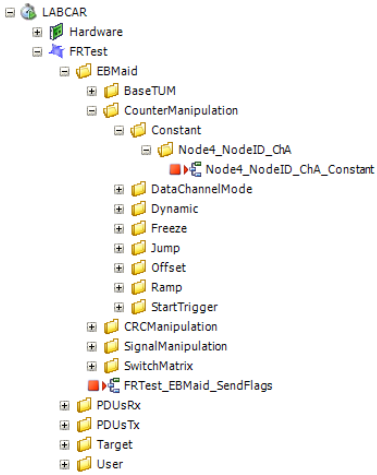
The signals selected for display in the Bus Communication Monitor are highlighted in yellow in Workspace Elements.

3.8 Target User Modules

As certain operations require detailed information about FlexRay communication, the relevant functionality must be implemented directly on the board – examples are alive counters and checksum calculations. The data for example for calculating CRCs is not available on the Real-Time PC – these thus have to be run on the board itself.

User-defined functionality on the board is implemented in the form of Target User Modules (TUM). The TUM-API offers functions for registering callbacks for specific events such as the sending and receiving of frames. It also opens up ways of exchanging messages with the FlexRay module and thus, for example, of communicating with the experiment environment. For more information on the Target User Modules, refer to the relevant Elektrobit Busmirror TUM documentation.

When TUM support is enabled, relevant ports are created for the signals defined in imported TUM descriptions for controlling the TUMs. The following signals are created in the Workspace Elements of the EE under every FlexRay module:



The relative path under this node reflects the functionality mapped in the TUM.

`\<TUM>\<Function>\<Signal>\<Parameter>`

The first level specifies the name of the TUM with the following TUM types being supported:

`SYSTEM`

`PROTOCOL`

`PROTOCOL-SAFEGUARDING`

`MANIPULATION`

`SIGNAL`

Note

The NIF module supports not all TUM-IDs. The following TUM-IDs are used for code generation: 60, 62, 1000, 1001, 1007, 1013, 1040, 1045, 1046, 1062 and 1063.

The second level specifies the functionality mapped in the TUM, e.g. `Counter-Manipulation`. The third level specifies the name of the signal for which the TUM offers the functionality and below that the parameters required for controlling this functionality.

Editing TUM Support for Signals and PDUs

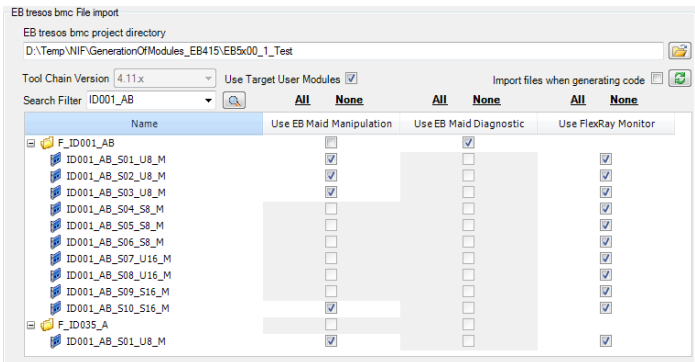
TUM support can be enabled and disabled for individual signals and PDUs of the EB tresos BMC data model. This increases performance and clarity.

All TUM signals independent of the EB tresos BMC data model (e.g. the signals of BaseTum or SwitchMatrix), are always supported providing the relevant TUMs are part of the EB tresos BMC project.

- Edit the EB tresos BMC project to add the application-specific TUM functionality to the project.
- To update the data model, synchronize the FlexRay module if necessary.
- In the user interface, enable the option "Use Target User Modules".
- To make TUM support accessible for individual signals, enable the "Use EB Maid Manipulation" (sending from Real-Time PC to board) or "Use EB Maid Diagnostic" (receiving from board to Real-Time PC)
  - Specify a regular expression as search filter to limit the selection of signals displayed.
  - Using the "All" and "None" options, all signals which satisfy the search criterion can be enabled or disabled respectively.

    Enabling/disabling affects all instances of the selected signal.

    Signals with more than 4 bytes are not transmitted as outports. The generated code makes a byte array available for these signals; the array can be further processed with user code.



Note

TUM support is only available for signals and PDUs which are contained in at least one TUM of the EB tresos BMC project.

- Save the module.

Selecting TUM Functionality for a Signal

For signals in TUMs with several functions, such as Dynamic Value, Hold, Ramp etc., only one individual function can be carried out at any one time. This function is selected via the corresponding inport under the node

`\<TUM>\<Function>\DataChannelMode\DataChannelMode_<Signal>`

The value of this inport specifies the function to be run. Refer to the TUM documentation for details on valid values and their functionality.

The value 0 always specifies the default state of the function. This is `OFF`, i.e. the function within the TUM is not executed.

Starting and Ending TUM Functionality for a Signal

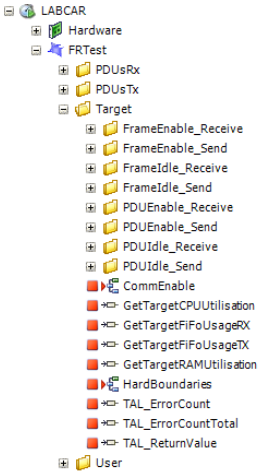
A TUM is not run cyclically. All parameters the TUM needs for execution are transferred in an individual function call. Then TUM execution begins until it is stopped by another call. To start the execution of a function, the following inport can be used:

`\<TUM>\<Function>\StartTrigger\StartTrigger_<Signal>`

When the value changes from 0 to 1, the function is started. To end a function, the following setting needs to be made: `DataChannelMode` = 0. Then a change from 0 to 1 has to be entered in the StartTrigger.

3.9 Experiment Environment

In the Experiment Environment, the "Workspace Elements" window (depending on the EB tresos bmc version used) offers a range of measurement variables, parameters and inports for communicating with the FlexRay module.



Target

There are various signals available under this node to control communication. These are generated as parameters or inports depending on the "Generate Target Communication flags as Variable" option.

- <*module_name*>/Target/CommEnable

  Enables or disables bus communication. If this parameter is set to 0, there is no communication on the bus.

- <*module_name*>/Target/HardBoundaries

  Enables or disables "hard" or "soft" boundaries. If this parameter is set to 0, the MinVal and MaxVal values described in the data model are used as "soft" boundaries of the signal. If this parameter is set to 1, the values calculated (using the bit length specified in the data model) are used as "hard" boundaries of the signal. The signal value is limited to the relevant boundaries in every cycle.

- <*module_name*>/Target/GetTargetCPUUtilisation

  The current utilization of the board's CPU in percent.

- <*module_name*>/Target/GetTargetRAMUtilisation

  The current utilization of the RAM memory of the board in percent. Only the memory dynamically allocated during runtime is taken into consideration - the share programmed in executable code ("static memory") is ignored which means that the value of "GetTargetRAMUtilisation" in an extreme case can even be 0%.

- <*module_name*>/Target/TAL_ErrorCount

  This counter reflects the number of TAL errors during a communication cycle during the simulation.

- <*module_name*>/Target/TAL_ErrorCountTotal

  The total number of all errors from TAL calls since the start of the simulation. Restarting communication resets the counter.

- <*module_name*>/Target/TAL_ReturnValue

  The first error code from a TAL call in the last communication cycle. Refer to the TAL user documentation for more information on the TAL function error codes.

- <*module_name*>/Target/GetTargetFiFoUsageRX

  The current usage of the receive FiFo between Real-Time PC and RAM memory of the board in percent.

- <*module_name*>/Target/GetTargetFiFoUsageTX

  The current usage of the send FiFo between Real-Time PC and RAM memory of the board in percent.

Target/FrameEnable_Receive/Send

There are various signals available under this node to enable and disable the frametriggerings on the FlexRay bus.

- <*module_name*>/Target/FrameEnable_Send/ FrameEnable_Send_<*frame_name*>

  Enables or disables the sending of the specified frame on the bus. This function is run using TAL calls:

  – Value = 1: The Frame-Triggering is sent according to the configuration of the Rest Bus Simulation.

  – Value = 0: The Frame-Triggering is no longer sent. The signal values are not changed on the send ports or the signal values are replaced by the InvalidValue of the signal if the "Signal InvalidValue" option is enabled.

  – Value= -1: A NULL frame is sent for the the Frame-Triggering.

Note

Wert = 0 does not work for sync frames, because the FlexRay controller does not support this feature. Please either use Value = -1 instead or change the sync frame in the configuration project.

Note

Wert = -1 does not work for the dynamic segment of FlexRay, because the FlexRay controller does not support this feature.

Note

If disableMethod value = 0 the API works on a slot level. This means disabling only one Frame-Triggering of a slot leads to undefined behaviour if there is another Frame-Triggering available for the same slot. You have to call Disable (Disappear) for all Frame-Triggerings of the according slot.

  – For more information see EB TAL manual.

- <*module_name*>/Target/FrameEnable_Receive/FrameEnable_Receive_<*frame_name*>

  Enables or disables the receipt of the specified frame by the bus. This function is run using TAL calls:

  – Value = 1: The frame is received.

– Value = 0: The frame is no longer received or the signal values are replaced by the InvalidValue of the signal if the "Signal InvalidValue" option is enabled with a disabled payload.

Note

If more than one FrameTrigger is assigned to a frame, an individual signal is generated for each FrameTrigger for controlling and timing. The name of the signal is extended by the suffix `ft_[SlotID]_[BaseCycle]_[CycleRepetion]` *of the FrameTrigger. Furthermore, a signal is generated for the common control of all FrameTriggers of the frame with the original name. If this common signal has a valid value [-1, 0, 1], its value is accepted. In all other cases, the values of the control signals of the individual FrameTriggers are used.*

Target/PDUEnable_Receive/Send

There are various signals available under this node to enable and disable the transmission of signals in PDUs between the FlexRay module and FlexRay bus.

- <*module_name*>/Target/PDUEnable_Send/PDUEnable_Send_<*PDU_name*>

  Enables or disables the sending of the specified PDU by the bus. This function is run using TAL calls:

  – Value = 1: The PDU is sent.

  – Value = 0: The PDU is no longer sent. The value last sent remains on the bus or the signal values are replaced by the InvalidValue of the signal if the "Signal InvalidValue" option is enabled with a disabled payload.

- <*module_name*>/Target/PDUEnable_Receive/PDUEnable_Receive_<*PDU_name*>

  Enables or disables the receiving of the specified PDU by the bus. This function is run using TAL calls:

  – Value = 1: The PDU is received.

  – Value = 0: The PDU is no longer received or the signal values are replaced by the InvalidValue of the signal if the "Signal InvalidValue" option is enabled with a disabled payload.

Target/FrameIdle_Receive/Send

There are various signals available under this node to enable and disable the transmission of signals between the COSYM project and FlexRay module.

- <*module_name*>/Target/FrameIdle_Send/FrameIdle_Send_<*frame_name*>

  Enables or disables the accepting of the signal values to be sent from the COSYM project.

  – Value = 0: Signal values of the frame are accepted by the project.

  – Value = 1: Signal values of the frame are not accepted by the project. The value last accepted is retained.

- <*module_name*>/Target/FrameIdle_Receive/FrameIdle_Receive_<*frame_name*>

  Enables or disables the transmission of the signal values received to the COSYM project.

  – Value = 0: Signal values of the frame are transferred to the project.

  – Value = 1: Signal values of the frame are not transferred to the project. The signal values last transferred are retained.

Target/PDUIdle_Receive/Send (Only EB tresos Busmirror 4.1.x and higher)

There are various signals available under this node to enable and disable the transmission of signals between the COSYM project and FlexRay module.

- <*module_name*>/Target/PDUIdle_Send/PDUIdle_Send_<*PDU_name*>

  Enables or disables the accepting of the signal values to be sent from the COSYM project.

  – Value = 0: Signal values of the PDU are accepted by the project.

  – Value = 1: Signal values of the PDU are not accepted by the project. The value last accepted is retained.

- <*module_name*>/Target/PDUIdle_Receive/PDUIdle_Receive_<*PDU_name*>

  Enables or disables the transmission of the signal values received to the COSYM project.

  – Value = 0: Signal values of the PDU are transferred to the project.

  – Value = 1: Signal values of the PDU are not transferred to the project. The signal values last transferred are retained.

Target/PDUTiming_Receive

One signal is available under this node for each PDU for issuing the timestamp for which a signal of the specified PDU was last received. The timestamp is issued in milliseconds.

These signals are always created as an output regardless of the "Generate Target communication Flags as Parameters" setting.

Send/ReceiveFrames

Under this node, inports are available for the signal values to be sent and outports are available for the received signal values. In this version, the signals are arranged under the relevant PDUs.

User

Under this node, the user-defined measurement variables, parameters, inports and outports are available depending on the user-defined module code.

3.10 Creating a FlexRay Module with the Automation Server

A FlexRay module can also be created using API functions of COSYM. The API maps all functions which the user can run via the GUI. The automatic insertion of user-defined code or adaptations is not supported but must be implemented outside COSYM in the case of automation.

There is a whole range of Set/Get methods with which the configuration of a FlexRay module can be specified or queried. In the case of an error, these set methods return "False" and in the case of success "True".

The relevant documentation (chm file) can be found via **? → Help** under "API Documentation".

3.10.1 Global Time synchronization

This feature is supported for the EB 5200 only, with the scope currently limited to Master mode.

If this feature is activated, the internal time of the RTPC is automatically communicated to the card as system time when the simulation is started in the experimental environment. The card distributes the system time as master time in the network.

In the experiment environment, two new measurement signals are also available under "GlobalTimeSync". "FrRx_GlobalSyncTime" includes the time read from the card and "FrTx_GlobalSyncTime" the time communicated to the card. If everything works correctly, the two times should be identical.

To activate this feature in a NIF project, you still have to go through the <EbUserDefinedToolVersion.xml> of the respective NIF module. In the file the value of the node <GlobalTimeSync> should be set to true.

Note

So that the user-specific settings are taken into account when loading the module, the nodes <EbVersionMajor> and <EbVersionMinor> must be filled with the values of the BMC version of the project and furthermore <NifEnabled> must be true.

For more information about EB HW or SW please have a look into the manuals and tutorials.

4 The FlexRay Editor APIs

The Scripting APIs are the programmable interface for remote controlling of the FlexRay module.

The following `.dll` files are found at <COSYM Installation directory>`\FlexRayStandaloneEditor\API` folder.

- `StandaloneEditorsUtils.dll`
- `FlexRay_API.dll`

A FlexRay module can also be created using API functions. These APIs are mapped with all the functions where you can run via the GUI of FlexRay Editor. There is a wide range of Set/Get methods with which the configuration of a FlexRay module can be specified or queried.

The relevant documentation to use these APIs are documented in the `FlexRay_API.chm` file and it can be found at <COSYM Installation directory>`\FlexRayStandaloneEditor\Manual\` folder.

Please refer to the scripting example which is available in the `FlexRay_API.chm` file itself.

5 ETAS Contact Addresses

ETAS HQ

ETAS GmbH

Borsigstraße 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	WWW:	www.etas.com

ETAS Subsidiaries and Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

ETAS subsidiaries	WWW:	www.etas.com/en/contact.php
ETAS technical support	WWW:	www.etas.com/en/hotlines.php

Index