

LABCAR-OPERATOR V5.4.4
User's Guide



Copyright

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

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

© **Copyright 2017** ETAS GmbH, Stuttgart

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

V5.4.4 R01 EN - 09.2017

Contents

1	Introduction	9
1.1	About this Manual	9
1.2	Using this Manual	10
2	The LABCAR-IP User Interface	13
2.1	Overview of Functions	13
2.2	The Main Menu of LABCAR-IP	14
2.2.1	The "File" Menu	14
2.2.2	The "View" Menu	15
2.2.3	The "Project" Menu	15
2.2.4	The "Tools" Menu	16
2.2.5	The "?" Menu	16
2.3	The Toolbar	17
2.4	The Main Window	18
2.5	The Project Explorer	19
2.6	The Log Window	19
2.7	Project Settings	20
2.7.1	"General" Tab	20
2.7.2	"Modules" Tab	21
2.7.3	"Events" Tab	22
3	Working with LABCAR-IP	23
3.1	Modular Simulation Code	25
3.1.1	LABCAR Modules	25
3.1.2	Types of Modules that can be Integrated	27
3.1.3	Creating a LABCAR-OPERATOR Project and Integrating Modules	27
3.1.4	Integrating External Code	29
3.1.5	Connection Management	33
3.1.6	Configuration of the Operating System ("OS Settings")	34

3.1.7	Code Generation	34
3.1.8	General Project Options	35
3.2	MATLAB/Simulink Models	38
3.2.1	Simulink Models	38
3.2.2	Restrictions regarding the Usable Simulink Models	40
3.2.3	Creating a Simulink Module	41
3.2.4	Further Settings for the Simulink Model	43
3.3	ASCET Modules	50
3.3.1	General Information on ASCET Modules	50
3.3.2	Preparations	50
3.3.3	Creating an ASCET Module	52
3.3.4	Measure Variables and Parameters in the ETAS Experiment Environment	55
3.4	C Code Modules	57
3.4.1	Tutorial	57
3.4.2	Creating a C Code Module Manually	58
3.4.3	Adding Code	64
3.4.4	Creating a C Code Module with the Automation Server	68
3.4.5	Exporting a C Code Module	68
3.4.6	Adding an Existing C Code Module	68
3.4.7	Changing an Existing C Code Module	70
3.4.8	Linking External C Code	70
3.4.9	Variable Labels	72
3.4.10	Functional Mock-up Units (FMU)	72
3.5	CAN Modules (Network Integration CAN)	83
3.5.1	Introduction	83
3.5.2	The J1939 Extension	85
3.5.3	The CAN Editor	87
3.5.4	Editing the CAN Network	91
3.5.5	Creating a CAN Network	93
3.5.6	The Component Parts of a CAN Network	98
3.5.7	User-Defined C Code	120
3.6	Instrumentation for the CAN Simulation	125
3.6.1	CAN Modules in the "Workspace Elements" Window	125
3.6.2	Instruments for CAN Messages	129
3.6.3	Enabling and Disabling Send Messages	130
3.6.4	The CAN Monitor	131
3.7	LIN Modules (Network Integration LIN)	133
3.7.1	Creating a LIN Module	134
3.7.2	The LIN Editor	134
3.7.3	Editing the LIN Network	137
3.7.4	The Component Parts of a LIN Network	141
3.7.5	User-Defined C Code	155
3.7.6	The LIN Module in ETAS EE	160
3.7.7	Instruments for LIN Frames	166
3.8	Network Module	167
3.8.1	Creating a Network Module	167
3.8.2	Maintaining a Network Module	167

3.8.3	Rest Bus Simulation of a Network Module	170
3.9	NIF Modules (Network Integration FlexRay)	173
3.9.1	Creating the Necessary Files with EB tresos Busmirror	174
3.9.2	Creating a FlexRay Module	174
3.9.3	Linking to the EB tresos bmc Tool Suite	175
3.9.4	Module Configuration	176
3.9.5	Linking the NIF Module to the LABCAR-OPERATOR Project	178
3.9.6	User-Defined Code and Adaptations	179
3.9.7	Linking to the ETAS Bus Communication Monitor (BCM)	186
3.9.8	Target User Modules	189
3.9.9	Experiment Environment	192
3.9.10	Creating an NIF Module with the Automation Server	195
3.10	FiL Modules	197
3.10.1	Configuration in LABCAR-IP	198
3.10.2	Selecting an A2L File	199
3.10.3	Defining Hardware for ECU Access	200
3.10.4	Selecting ECU Variables for the Stimulation	202
3.10.5	Selecting ECU Outputs for Connecting to the Model	204
3.10.6	OS Settings	205
3.10.7	Creating Connections in the Connection Manager	207
3.10.8	Working in ETAS EE	208
3.10.9	The "RT ECU Access" Instrument	209
3.11	Real-Time Plugins	210
3.11.1	The RT-Plugin Builder	211
3.11.2	Example	214
3.11.3	Defining Hooks in the OS Configuration of LABCAR-IP	217
3.11.4	Attaching Plugin Processes to Hooks	217
3.11.5	Label and Mapping	218
3.11.6	The Lifecycle of a Real-Time Plugin	219
3.11.7	Working with Real-Time Plugins in the Experiment Environment	220
3.11.8	The "RT Plugins" Tab	222
3.12	Signal Conversion Modules	225
3.12.1	Inserting Modules	225
3.12.2	Parameters	225
3.12.3	The GUI for Controlling Signal Conversion	226
3.13	Configuring Hardware with the RTIO-Editor	230
3.14	The Connection Manager	231
3.14.1	Real and Virtual Connections	231
3.14.2	Adding Inports and Outports to the Simulink Model	234
3.14.3	Connecting Signals in the Connection Manager	236
3.14.4	Signal Conversion	240
3.15	Configuring the Real-Time Operating System (OS Configuration)	243
3.15.1	Elements of an OS Configuration	243
3.15.2	Working with OS Configurations	250
3.16	Setting Up Multi-RTPC Networks	256
3.16.1	Connecting Hardware	257
3.16.2	Specifying IP Addresses of the Real-Time PCs	258
3.16.3	Configuring the PTP and Data Connections in the Web Interface	259

3.16.4	Creating LABCAR-OPERATOR Projects	262
3.16.5	Merging Projects	263
4	ETAS Experiment Environment - an Overview	265
4.1	The Constituent Parts of the GUI	266
4.1.1	The "Workspace Elements" Window	266
4.1.2	The Experiment Explorer	266
4.1.3	The Main Workspace	266
4.1.4	The "Instruments" Window	267
4.1.5	The "Properties" Window	267
4.1.6	The "Application Log" and "Hardware Output" Windows	267
4.1.7	The Main Menu of ETAS EE	267
4.1.8	The Toolbar	268
4.2	The Experiment Explorer	268
4.2.1	Working with the Experiment Explorer	268
4.3	The "Workspace Elements" Window	270
4.3.1	The Tabs of the "Workspace Elements" Window	271
4.3.2	Working with the Workspace Elements	272
4.3.3	Search and Filter Function	275
4.4	The Main Workspace	276
4.4.1	The "Instrumentation" Tab	276
4.4.2	The "Datalogger" Tab	278
4.4.3	The "Signal Generator" Tab	282
4.4.4	The "RT-Plugins" Tab	300
4.4.5	The "Instruments" Window	300
4.4.6	Adding Signals to the Signal List	301
4.5	The Script Recorder	303
4.6	Working with Parameters	306
4.6.1	Parameters in ETAS EE	307
4.6.2	The Shortcut Menu	308
4.6.3	Modifying Individual Parameters During the Experiment	309
4.6.4	Saving Complete or Partial Parameterizations	313
4.6.5	Managing Parameter Files in the Experiment Explorer	314
4.6.6	Variants and Parameter Combinations	317
4.6.7	Mapping Files	321
4.6.8	Creating and Managing Mapping Files	322
4.7	Editing Parameter Files with LABCAR-PA	326
4.7.1	Managing Parameter Files	326
4.7.2	The Parameter Table	328
4.7.3	A Description of the Menus	333
4.7.4	The Icon Bar	336
4.7.5	Working with LABCAR-PA	337
4.7.6	Setting Options	337
4.7.7	Editing Parameters	342
4.7.8	Reference Files	347
4.7.9	Exporting Parameters to a File	348
4.7.10	Importing Parameters from a File	350
4.7.11	The Filter and Sorting Function	353
4.7.12	The Search Function for Model Parameters	355

4.7.13	Determining the Parameter Scope Binding	358
4.7.14	Online Parameterization	359
4.8	LABCAR-CCI V5.4.4 (Calibration Connector for INCA)	362
4.8.1	System Requirements	363
4.8.2	Working with LABCAR-CCI V5.4.4	363
5	Appendix	367
6	ETAS Contact Addresses	369
	Figures	371
	Index	373

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 LABCAR-OPERATOR V5.4.4. The manual contains the following chapters:

- **"The LABCAR-IP User Interface" on page 13**

This chapter contains a description of the LABCAR-IP user interface.

- "Overview of Functions" on page 13
- "The Main Menu of LABCAR-IP" on page 14
- "The Toolbar" on page 17
- "The Main Window" on page 18
- "The Project Explorer" on page 19
- "The Log Window" on page 19
- "Project Settings" on page 20

- **"Working with LABCAR-IP" on page 23**

This chapter describes the integration of various modules into a LABCAR-OPERATOR project and how to generate code for an experiment which can be run in ETAS Experiment Environment.

- "Modular Simulation Code" on page 25
- "MATLAB/Simulink Models" on page 38
- "ASCET Modules" on page 50
- "C Code Modules" on page 57
- "CAN Modules (Network Integration CAN)" on page 83
- "LIN Modules (Network Integration LIN)" on page 133
- "Network Module" on page 167
- "NIF Modules (Network Integration FlexRay)" on page 173
- "FiL Modules" on page 197
- "Real-Time Plugins" on page 210
- "Signal Conversion Modules" on page 225
- "Configuring Hardware with the RTIO-Editor" on page 230
- "The Connection Manager" on page 231
- "Configuring the Real-Time Operating System (OS Configuration)" on page 243

- **"ETAS Experiment Environment - an Overview" on page 265**

ETAS Experiment Environment (ETAS EE) is used to execute a LABCAR-OPERATOR experiment. This chapter provides an overview of the GUI and the functions of ETAS EE.

- "The Constituent Parts of the GUI" on page 266 "The Experiment Explorer" on page 268
- "The "Workspace Elements" Window" on page 270
- "The Main Workspace" on page 276
- "Working with Parameters" on page 306
- "Editing Parameter Files with LABCAR-PA" on page 326
- "LABCAR-CCI V5.4.4 (Calibration Connector for INCA)" on page 362

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 <code>setup.exe</code> file.	Text strings in list boxes, in program code and in path and file names are printed using the <code>Courier</code> font.
A conversion between Logic and Arithmetic data types is <i>not</i> possible.	Emphasized text portions and newly introduced terms are printed in an <i>italic</i> font face.

Important notes for the users are presented as follows:

Note

Important note for users.

2 The LABCAR-IP User Interface

This chapter contains a description of the LABCAR-IP user interface.

The individual sections contain information on:

- "Overview of Functions" on page 13
- "The Main Menu of LABCAR-IP" on page 14
- "The Toolbar" on page 17
- "The Main Window" on page 18
- "The Project Explorer" on page 19
- "The Log Window" on page 19
- "Project Settings" on page 20

2.1 Overview of Functions

Once you have launched LABCAR-IP and loaded a project, the user interface is displayed as shown below.

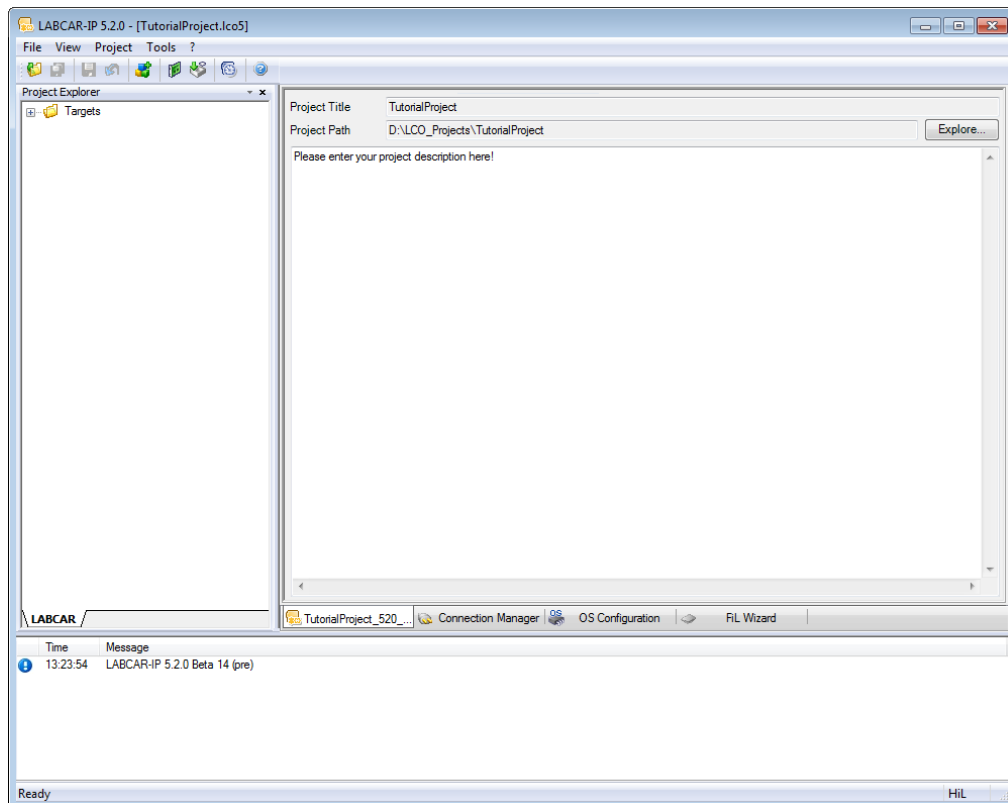


Fig. 2-1 The LABCAR-IP User Interface with an Open Project

The interface has the following control elements:

- the menu bar
For a description of the LABCAR-IP menus refer to the section "The Main Menu of LABCAR-IP" on page 14.

- the toolbar
For a description of the elements in the toolbar refer to the section "The Toolbar" on page 17.
- the main window
The different tabs contain project information, the Connection Manager, the CAN Editor and the OS Configuration (see "The Main Window" on page 18).
- the Project Explorer
See "The Project Explorer" on page 19
- the Log window
See "The Log Window" on page 19

2.2 The Main Menu of LABCAR-IP

The menu bar of LABCAR-IP contains the following entries:

- "The "File" Menu" on page 14
This menu contains all functions for file-specific actions.
- "The "View" Menu" on page 15
This menu is used to control the visibility of the individual windows.
- "The "Project" Menu" on page 15
This menu contains all functions needed for project editing, project management and experiment control.
- "The "Tools" Menu" on page 16
Use this menu provides access to various tools.
- "The "?" Menu" on page 16
This menu contains information on LABCAR-IP.

Note

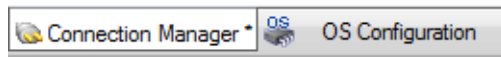
The following description of the LABCAR-IP menu structure is only intended to provide you with an overview. For more detailed information on the individual functions refer to the relevant chapters.

2.2.1 The "File" Menu

This menu contains all functions for file-specific actions.

- **File** → **New Project**
For creating new projects
- **File** → **Open**
Opens an existing project
- **File** → **Close**
Closes the current project

- **File** → **Save All**
Saves the entire project including all changes in the CAN Editor, Connection Manager etc.
- **File** → **Save As**
Saves the entire project under a different name
- **File** → **Save**
Saves the content of the document currently active (CAN Editor, Connection Manager etc.). An asterisk after a tab's name indicates there are unsaved changes.



- **File** → **Discard**
Discards all unsaved changes in the document currently active (see **File** → **Save**)
- **File** → **Recent Projects** →
Displays the list of the last four project files opened
- **File** → **Exit**
Exits LABCAR-IP

2.2.2 The "View" Menu

This menu is used to control the visibility of the individual windows.

- **View** → **Status Bar**
Enables/disables the status bar at the bottom edge of the main window
- **View** → **Tool Bar**
Enables/disables the display of the tool bar
- **View** → **Project Explorer**
Opens/closes the Project Explorer
- **View** → **Log Window**
Opens/closes the "Log Window"

2.2.3 The "Project" Menu

This menu contains all functions needed for project editing, project management and experiment control.

- **Project** → **Add Module**
Opens the "Add Module Wizard".
- **Project** → **RTIO Editor**
Opens the RTIO Editor to configure the connected hardware
- **Project** → **Build**
Opens the dialog window for code generation
- **Project** → **Options**
Opens the dialog window in which the project settings are made

- **Project** → **ECU Pin List**
Opens the "ECU Pin & HW Pin Properties" Dialog
- **Project** → **OS Configuration**
Opens the "OS Configuration" tab

2.2.4 The "Tools" Menu

Use this menu provides access to various tools.

- **Tools** → **HSP Update Tool**
Starts the current version of the HSP Update Tool
- **Tools** → **Open LABCAR-IP Log File**
Opens the log file which contains detailed information
- **Tools** → **Zip Log Files**
Opens a dialog box for zipping the existing log files
- **Tools** → **Zip And Go**
Allows for zipping the current project
- **Tools** → **Open Experiment Environment**
Opens the experiment environment ETAS EE
- **Tools** → **View Version Info**
Shows a list of versions of all software components
- **Tools** → **Options**
Opens a dialog box for editing the LABCAR ID

2.2.5 The "?" Menu

This menu contains information on LABCAR-IP.

- **?** → **Help**
Opens an HTML file with links to the documentation and other information
- **?** → **About**
Information on installed LABCAR software products and their versions
- **?** → **Contact**
Shows ETAS contact information
- **?** → **License**
Opens the ETAS License Manager

2.3 The Toolbar

The LABCAR-OPERATOR V5.4.4 toolbar contains the following functions



1 Open

Opens the document (identical to **File** → **Open**)

2 Save All

Saves the entire project including all changes in the CAN Editor, Connection Manager etc. (identical to **File** → **Save All**)

3 Save

Saves the active document (identical to **File** → **Save**)

4 Discard

Discards all unsaved changes in the document currently active (identical to **File** → **Discard**)

5 Add Module

Opens the Add Module Wizard

6 Edit Hardware Configuration

Launches the RTIO Editor (identical to **Project** → **RTIO Editor**)

7 Build LABCAR Project

Starts code generation (identical to **Project** → **Build**)

8 Open Experiment Environment

Opens the experiment environment ETAS EE (identical to **Tools** → **Open Experiment Environment**)

9 Help

Information on LABCAR-OPERATOR V5.4.4 (identical to **Help** → **Help**)

2.4 The Main Window

The main window contains all the important functions of LABCAR-IP – these functions are divided into individual tabs for a clearer structure.

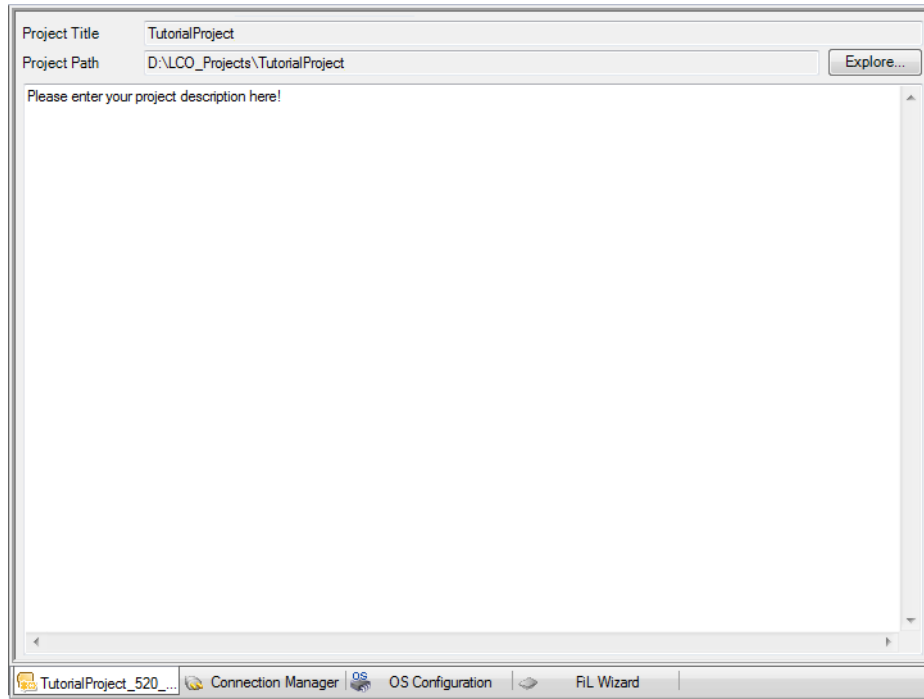


Fig. 2-2 The Main Window of LABCAR-IP

The "Project Info" Tab

This tab displays the project name and indicates where the project files are stored. Information on the project can also be entered here in a text field.

The "Connection Manager" Tab

This tab contains all functions of the Connection Manager (see "The Connection Manager" on page 231).

The "OS Configuration" Tab

All settings for the real-time operating system can be made in this tab (see "Configuring the Real-Time Operating System (OS Configuration)" on page 243).

The "FiL Wizard" Tab

The generation and integration of FiL modules takes place in this tab (see "FiL Modules" on page 197).

2.5 The Project Explorer

All project-relevant objects and documents are managed in the docking window of the Project Explorer.

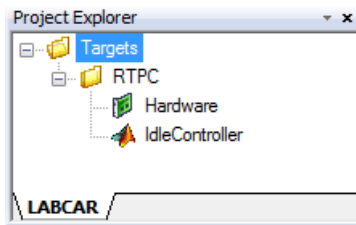


Fig. 2-3 Project Explorer

Targets

The "Targets" folder contains a folder for the experimental target. For each experimental target a subfolder with the targets name is created.

- **Experimental system name ("RTPC")**

The shortcut menu of a file in this folder contains the following functions:

- **Add Module**
- **Edit OS Configuration**

Each of these folders contains:

- All modules (Simulink models, C code, CAN, signal conversion etc.)
- The hardware configuration

The shortcut menu of a file in this folder contains the following functions:

- **Edit**
Opens the relevant editor
- **Update**
The module is opened in the corresponding editor and any changes updated.
- **Remove** (not with hardware configuration)

2.6 The Log Window

"Messages" Tab

This window displays LABCAR-OPERATOR V5.4.4 information and error messages.

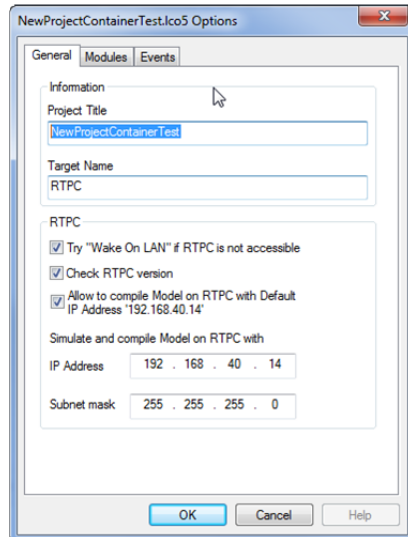
To delete the display in this window, right-click the window and select **Clear Log**.

2.7 Project Settings

There is a range of settings in a LABCAR-OPERATOR project which refer to the different components of LABCAR-OPERATOR. These settings can be accessed via **Project** → **Options**.

2.7.1 "General" Tab

This tab contains information on the "Project Title" and the "Target Name".

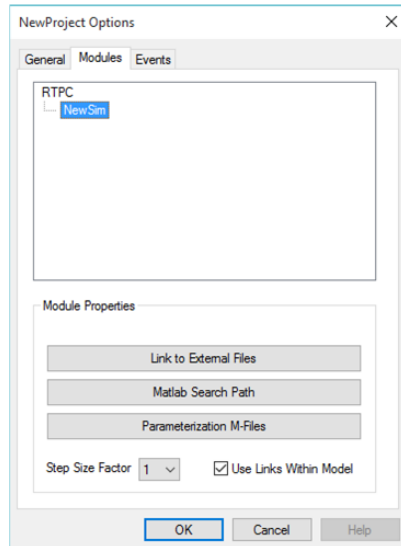


Settings for the Real-Time PC can also be made:

- Try "Wake On LAN" if RTPC is not accessible
If this option has been selected and the Real-Time PC is not responding, Wake On LAN is used to try to start the Real-Time PC.
- Check RTPC version
With this option, the version of the Real-Time PC detected is determined. A check is carried out during the build process to see if this version can work with the current version of LABCAR-OPERATOR.
- Allow to compile Model on RTPC with Default IP Address '192.168.40.14'
If this option has been selected, the IP Address mentioned in the "IP Address" field will be checked first. If the RTPC is still not accessible, then the default IP address is used. If the checkbox is unchecked and the IP address is not reachable, an error occurs.
- IP Address
The IP address of the Real-Time PC assigned to this project (for multi-RTPC projects).
- Subnet Mask
Subnet mask for determining the network prefix

2.7.2 "Modules" Tab

If you are working with the Modeling Connector for Simulink, this is where the modules of the project, such as the Simulink model and external C code, are listed and managed.



For details of the function of these settings, refer to the section "Further Settings for the Simulink Model" on page 43.

2.7.3 "Events" Tab

While a project is running, scripts can be incorporated which are run before or after a build process. A script is any executable which can be run under Windows (*.exe, *.bat, *.cmd, etc.).

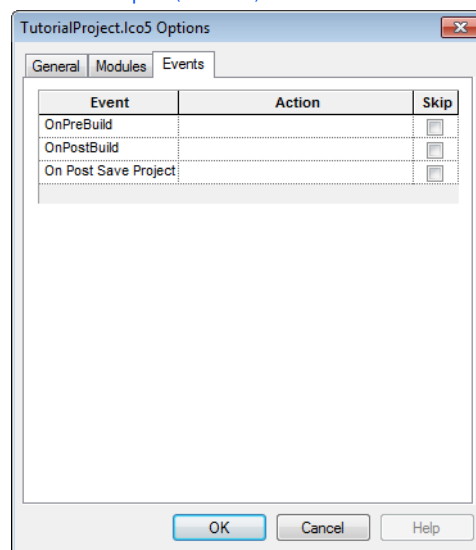
To combine a script with an event

- Select **Project** → **Options**.

The "*Project_name* properties" window opens.

- Select the "Events" tab.

This contains a list of events with which the execution of scripts (Action) can be linked.



- To specify this kind of script, click the cell next to the relevant event.
- Click on the arrow at the edge of the cell and select **<browse ...>**.

A file selector window opens in which you can select the script file to be run.

- Check **Skip** if you do not want to execute a specific script (temporarily).

3 **Working with LABCAR-IP**

This chapter describes the integration of various modules into a LABCAR-OPERATOR project and how to generate code for an experiment which can be run in ETAS Experiment Environment.

The individual sections contain information on:

- "Modular Simulation Code" on page 25
This section provides a short overview on integrating modules of all kinds of origins into a LABCAR-OPERATOR overall project.
- "MATLAB/Simulink Models" on page 38
This section contains information on integrating MATLAB/Simulink models:
- "ASCET Modules" on page 50
This section contains information on integrating ASCET modules in a LABCAR-OPERATOR project.
- "C Code Modules" on page 57
In addition to MATLAB/Simulink models and ASCET modules it is also possible in LABCAR-IP to integrate C code defined by the user into the project.
- "CAN Modules (Network Integration CAN)" on page 83
LABCAR-NIC V5.4.4 (Network Integration CAN) is an add-on to LABCAR-OPERATOR V5.4.4. It enables simple testing of ECU functions which include CAN communication.
- "Instrumentation for the CAN Simulation" on page 125
If a CAN residual bus simulation is executed with LABCAR-NIC V5.4.4, there are special instruments in ETAS EE for CAN messages as well as a "CAN Bus Monitor".
- "LIN Modules (Network Integration LIN)" on page 133
LABCAR-NIL (Network Integration LIN) is an add-on for LABCAR-OPERATOR. It makes it possible to test ECU functions that include LIN communication in accordance with LIN 2.0 and LIN 2.1/LIN 2.2.
- "Network Module" on page 167
This chapter contains information on creating and maintaining Network modules.
- "NIF Modules (Network Integration FlexRay)" on page 173
LABCAR-NIF V5.4.4 (Network Integration FlexRay) is a LABCAR-NIF V5.4.4 module type in LABCAR-OPERATOR V5.4.4. It enables simple testing of ECU functions incorporating FlexRay communication.
- "FiL Modules" on page 197
This chapter describes the creation of FiL modules.
- "Real-Time Plugins" on page 210
This chapter contains information on creating real-time plugins, how to integrate them into a LABCAR-OPERATOR project and how to work with them in the experiment environment ETAS EE.

- "Signal Conversion Modules" on page 225
LABCAR-OPERATOR V5.4.4 comes with a standard signal conversion module with which the generic open-loop configuration can be realized.
- "Configuring Hardware with the RTIO-Editor" on page 230
How to work with the RTIO-Editor and configure the hardware used is described in detail in LABCAR-RTC V5.4.4 - User's Guide.
- "The Connection Manager" on page 231
The Connection Manager enables closed-loop operation by connecting the inputs and outputs of the existing modules accordingly.
- "Configuring the Real-Time Operating System (OS Configuration)" on page 243
When creating a LABCAR-OPERATOR project, a default OS configuration is automatically created and added to the project.
- "Setting Up Multi-RTPC Networks" on page 256
This chapter explains how to set up Multi-RTPC networks.

3.1 Modular Simulation Code

This section provides a short overview on integrating modules of all kinds of origins into a LABCAR-OPERATOR overall project.

It contains information on the following topics:

- "LABCAR Modules" on page 25
- "Types of Modules that can be Integrated" on page 27
- "Creating a LABCAR-OPERATOR Project and Integrating Modules" on page 27
- "Integrating External Code" on page 29
- "Connection Management" on page 33
- "Configuration of the Operating System ("OS Settings")" on page 34
- "Code Generation" on page 34

3.1.1 LABCAR Modules

In LABCAR-OPERATOR, the simulation model code consists of individual "LABCAR modules" that each describe one component of the overall project.

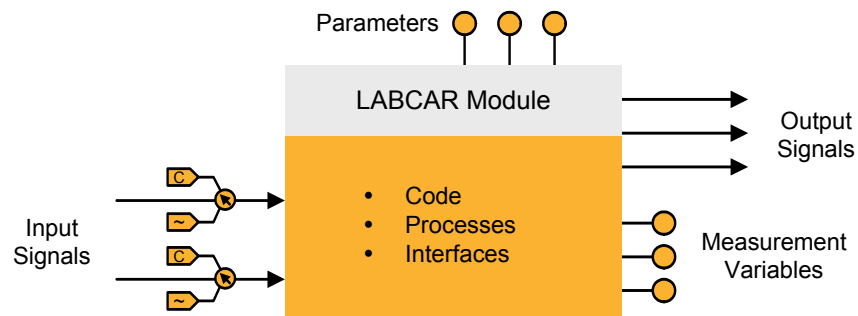


Fig. 3-1 A LABCAR Module

An individual LABCAR module consists of the interface information and the functionality itself (in the form of standard C code).

The interface information contains the following:

- definition of the module's inputs and outputs
- information to access the measure and calibration variables
- processes, that control the module

Fig. 3-2 on page 26 shows the structure of a project consisting of several modules. Each of these modules can be connected with other modules at signal level (using the Connection Manager). In this way, signal paths are defined and control circuits closed.

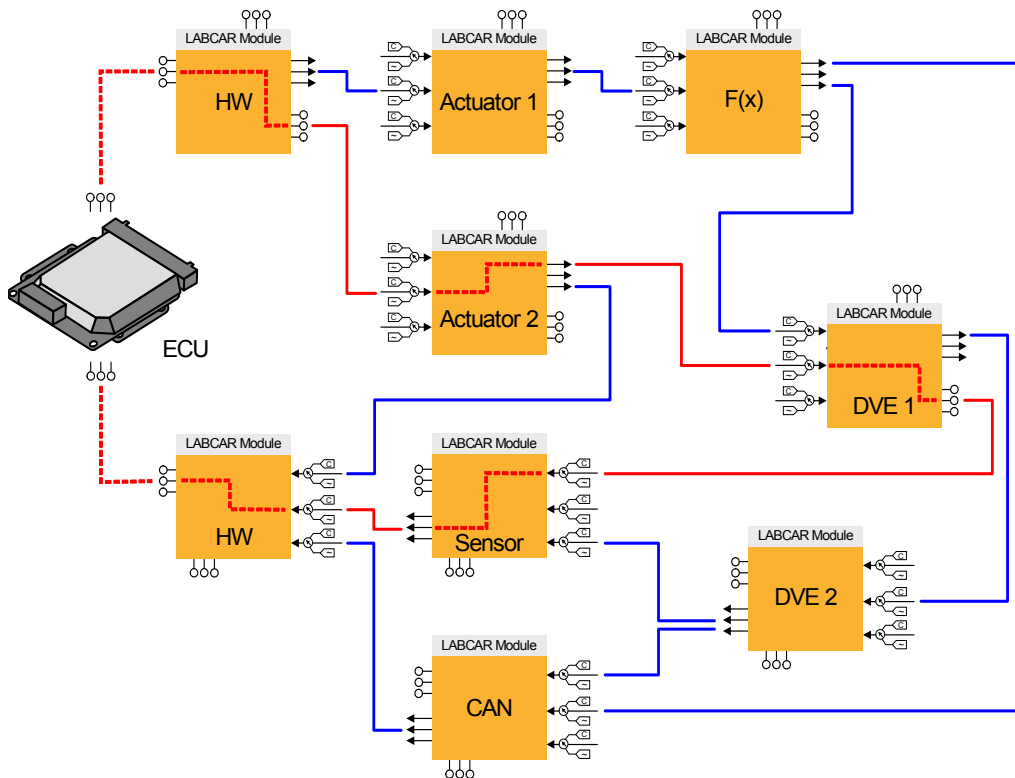


Fig. 3-2 Modules of a LABCAR-OPERATOR Project

Each of the inputs can be interrupted during runtime to supply a constant value, a synthetic signal shape or a recorded signal trace (e.g. of a test drive) (see "Signal Conversion Modules" on page 225).

C code is initially generated from each of the modules during code generation if these are not already available in C code. The individual C code parts are then integrated in the complete simulation code that is later run on the simulation target (RTPC).

This approach offers maximum flexibility because any behavior-describing tool capable of generating C code can be used. This means the domain-specific advantages of various tools can be used specifically, such as using various modeling languages, residual bus simulators or C code libraries.


Calculating the Simulation Code


The real-time calculation of the simulation code takes place on a Real-Time PC, a simulation target based on Intel® processor architecture, that provides generic support for multi-core processors. This makes it possible to distribute the individual modules to several CPU cores.


3.1.2 Types of Modules that can be Integrated


Below is a list of the various modules that can be integrated into a LABCAR-OPERATOR project and notes on where the integrating of these modules is described in this User's Guide.


The individual modules are indicated by the following icons in the Project Explorer.


- 
 - **ASCET and MATLAB/Simulink modules**


For more details on integrating these modules, refer to the sections "ASCET Modules" on page 50 and "MATLAB/Simulink Models" on page 38.
- 
 - **C code modules**

For more details on integrating C code modules, refer to the section "C Code Modules" on page 57.
- 
 - **CAN, LIN and FlexRay modules**

For more details on integrating these modules, refer to the sections "CAN Modules (Network Integration CAN)" on page 83, "LIN Modules (Network Integration LIN)" on page 133 and "NIF Modules (Network Integration FlexRay)" on page 173.
- 
 - **Network modules**

For more details on integrating these modules, refer to the section "Network Module" on page 167.
- 
 - **FiL modules**

For more details on integrating FiL modules, refer to the section "FiL Modules" on page 197.
- 
 - **Modules for open-loop access and signal conversion**

For more details on creating and configuring these modules, refer to the section "Signal Conversion Modules" on page 225
- 
 - **I/O hardware modules**

These modules are only part of the simulation code in the sense that they describe the I/O hardware between the model and the ECU. For a detailed description of how to work with the RTIO-Editor and of the configuration of the hardware involved, refer to the LABCAR-RTC V5.4.4 - User's Guide.

The integration procedure depends on the type of module and is described below.

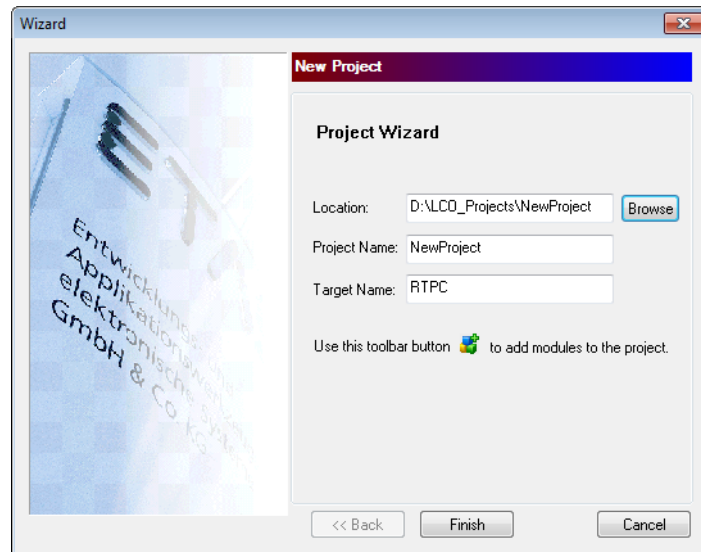
3.1.3 Creating a LABCAR-OPERATOR Project and Integrating Modules

This is where you create a project (with or without a Simulink model) and integrate any additional C-code-based modules.

To create a project

- Select **File → New Project**.
The project wizard opens.
- In "Location" select a directory in which the new project is to be created.
- Enter a project name beside "Project Name".

- If required, assign the target a different name in "Target Name".



- Click **Finish**.
The LABCAR-OPERATOR project is created. You can now add modules.

To add a module

- Select **Project** → **Add Module**.

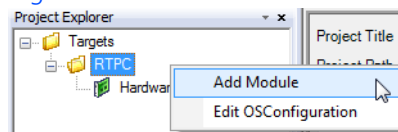
or

- Click the **Add Module** icon.

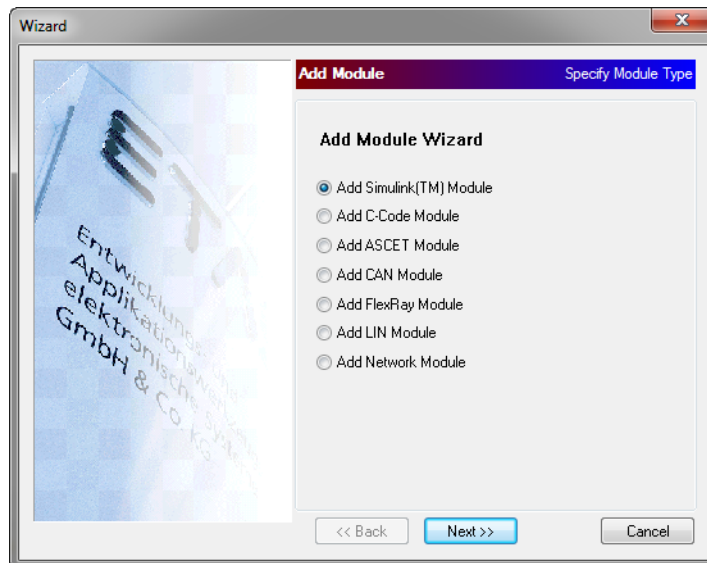


or

- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select the required module.



Different procedures are necessary for integrating additional modules (that cannot be selected in the Module Wizard):

- Hardware module

Every LABCAR-OPERATOR project has a hardware module that is created with the project and that itself contains no hardware at this time apart from the simulation target "RTPC". For more details on the configuration of the module (= integration of I/O hardware), refer to the LABCAR-RTC V5.4.4 - User's Guide.

- CAN modules

CAN modules are created and managed in the "CAN Editor" tab – for detailed information, refer to the section "CAN Modules (Network Integration CAN)" on page 83.

- FiL modules

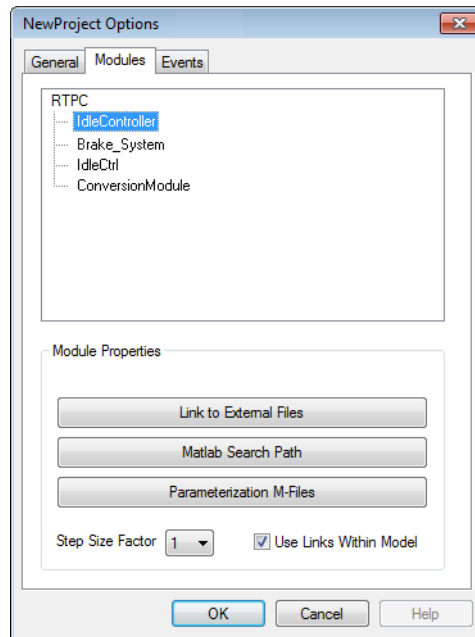
FiL modules are integrated and managed in the "FiL Wizard" tab – for detailed information, refer to the section "FiL Modules" on page 197.

3.1.4 Integrating External Code

In addition to integrating individual modules, you can ensure that additional header, code, object and library files are compiled and linked for some module types (Simulink, CAN and signal conversion modules) during module code generation. This means that frequently used code files and libraries can be exchanged between projects and reused an infinite number of times.

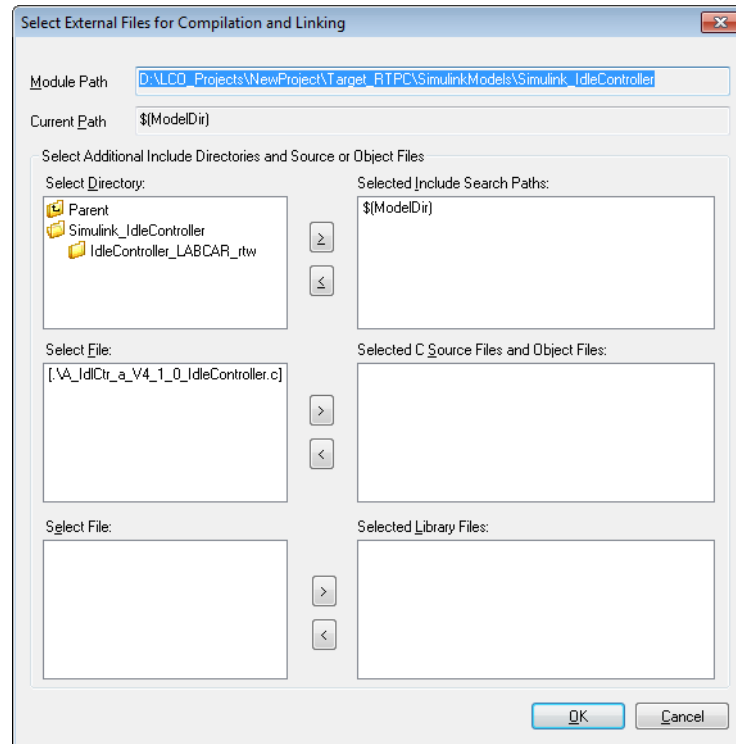
To select external data sources

- Select **Project** → **Options** from the main menu.
- The "<project_name> Options" window opens.
- Select the "Modules" tab.
- Select the module for which further files are to be specified.



- Click **Link to External Files**.

The "Select External Files for Compilation and Linking" window opens.



This is where further directories and files to be considered during compilation and linking can be specified.

These are:

- directories in which header files (*.h) are to be searched for
- files with source and object code (*.c, *.o)
- library files (*.a)

To guarantee the portability of the project and the stability of code generation on the Real-Time PC, only directories and files within the module path (e.g. \$(ModelDir)) can be referenced.

The "user" Directory

In the project directory `<project_name>\Target_RT\PC\` a further `\user` folder is created in which additional files, needed on the real-time PC during the compiler/link procedure, can be stored.

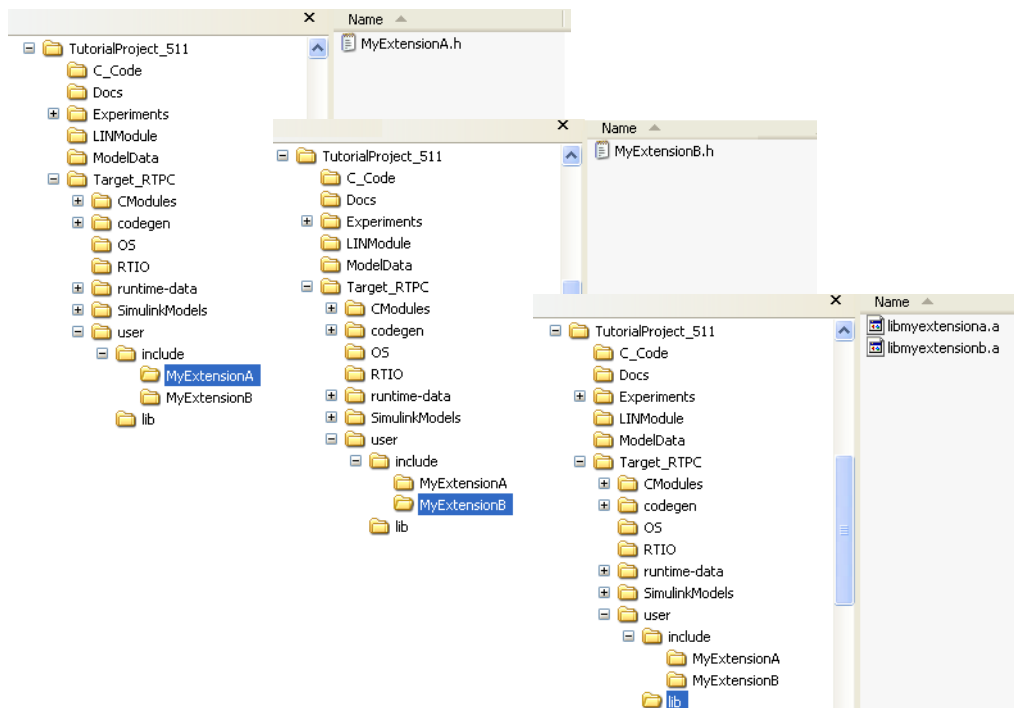
For this purpose, the content of the `\user` folder is transferred to the real-time PC during the build process to the directory

```
/home/labcar/codegen/
```

"labcar" is the default user on the ETAS RTPC operating system.

The folder contains two further subfolders:

- `include`
This is where header files containing the declarations for the routines of the libraries to be used can be stored.
Subfolders in this folder are detected.
- `lib`
This is where libraries whose code can/is to be used during the linking of the project can be stored.
Subfolders in this folder are **not** detected.

Example:

To make the library functions visible for a LABCAR-OPERATOR module (e.g. a C code module), the relevant header files must be referenced in the source code of the corresponding modules.

Example:

```
#include "../MyExtensionA/MyExtensionA.h"
#include "../MyExtensionB/MyExtensionB.h"
```

Unlike the standard method of adding external code to a special module (**Project** → **Options**, "Modules" tab: [Link to External Files](#)), the libraries and header files stored in the `user` directory can be accessed by all modules.

3.1.5 Connection Management

Fig. 3-2 on page 26 shows not only project modules but also connections between their input and outputs. These connections are created in the Connection Manager and enable closed-loop operation and signal tracing.

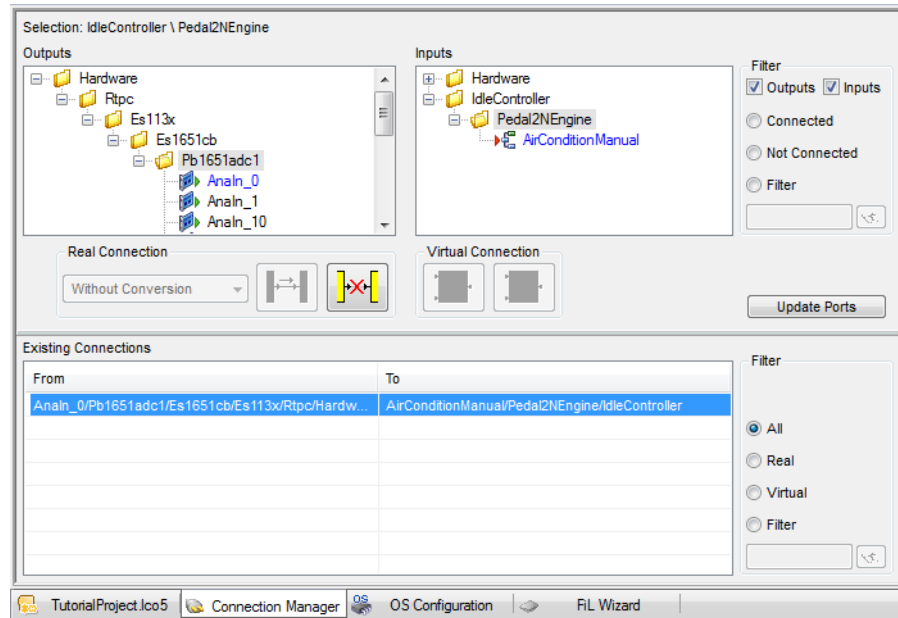


Fig. 3-3 The Connection Manager

For detailed information on connection management, refer to the section "The Connection Manager" on page 231.

When creating a connection, you can also decide whether a signal conversion module should be added at the input of the one module. With this kind of module, you have a choice between different types of signal conversion – in addition, you can interrupt the connection to the other module and instead supply manually specified signals or signals from a signal generator (see "Signal Conversion" on page 240)

These modules are configured in special GUIs in the experiment environment (see "Signal Conversion Modules" on page 225).

3.1.6 Configuration of the Operating System ("OS Settings")

When a module is integrated in the project, the relevant processes are automatically accepted in the OS configuration and assigned to suitable tasks. Once all modules are integrated, it might be necessary to adjust the configuration of the real-time operating system.

This concerns the adding and removing of tasks, the task properties themselves, and the assigning and removing of processes to/from tasks.

For a detailed description of these tasks, refer to the section "Configuring the Real-Time Operating System (OS Configuration)" on page 243.

3.1.7 Code Generation

Once all the modules have been integrated in the project, the signals connected and the task calculation configured, the simulation code for the experimental target can be generated.

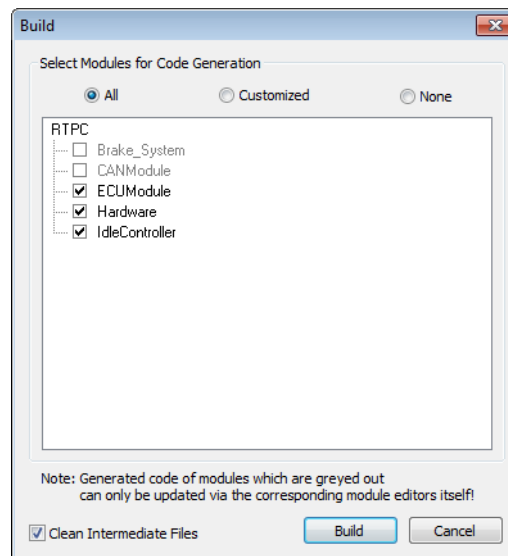
To generate code

- Select **Project** → **Build...**

or



- Click the **Build LABCAR Project** icon.
- Select the modules for which code is to be generated in the following window.



There are two reasons why modules in the list may be grayed out (in other words cannot be selected):

- These are C code modules for which code is generated automatically

- These are Simulink or ASCET modules and the corresponding add-on LABCAR-MCS (Modeling Connector for Simulink) or LABCAR-MCA (Modeling Connector for ASCET) respectively is not installed.

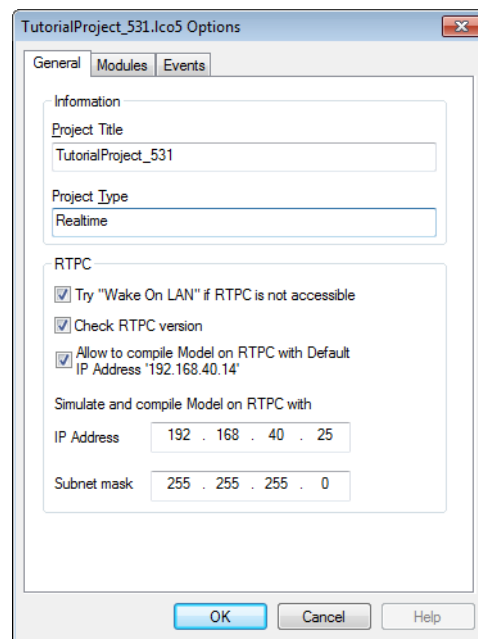
If you activate the option "Clean Intermediate Files", the compiled object code is deleted after the build process. This is necessary when precompiled code is not to be used again.

- Click **Build**.

The code generation process starts and the progress of events is displayed in the "Messages" window.

3.1.8 General Project Options

You will find a few important general settings for the project under **Project** → **Options** in the "General" tab.



Information:

- **Project Title**
The name of the project defined on its creation.
- **Project Type**
Always "Realtime"

RTPC:

- **Try "Wake On LAN" if RTPC is not accessible**

This is used to try to boot the Real-Time PC via the network ("Wake-on-LAN").

For this purpose, you require a corresponding tool on the relevant network PC which sends what are called "magic packets" (e.g. the free tool "WOL - Magic Packet Sender 2007"). The necessary data for configuring this tool can be found on the web interface of ETAS RTPC under "Wake On LAN Settings" ([Main Page >> Power Control](#)).

- **Check RTPC version**

A specific version of LABCAR-OPERATOR basically only supports certain versions of ETAS RTPC (see Release Notes). This option can be enabled to ensure that the simulation target has the right version.

The build is interrupted with an error if the version is not supported.

- **Allow to compile model on RTPC with default IP address '192.168.40.14'**

Select this option if you always want to build a project on a system with the default IP address – regardless of the IP address of the Real-Time PC for this project.

Example: The project is configured for a Real-Time PC with the IP address 192.168.40.25 (see "**Simulate and compile model on RTPC with**" on page 37) which, however, is not available for the build on the system with the default address.

The default address is then used instead of the IP address not found:

```

17:57:40  Checking connection to RTPC with IP Address 192.168.40.25.
17:57:43  Checking connection to RTPC with IP Address 192.168.40.14.
17:57:44  RTPC 192.168.40.14 is connected.

```

The basic procedure is as follows:

- Check the configured IP address

If the Real-Time PC is found under the configured address, it is used for the build.

If not:

- Check fallback IP address 192.168.40.14.

If the Real-Time PC is found under the fallback address, it is used for the build.

If not:

- If "Wake On LAN" is enabled, an attempt is made to wake the Real-Time PC using the configured address.

If the Real-Time PC can be woken using the configured address, it is used for the build.

If not:

Try to wake the Real-Time PC using the fallback IP address.

If the Real-Time PC can be woken using the fallback IP address, it is used for the build.

If not:

- Build not possible

- **Simulate and compile model on RTPC with**

This is where you can specify an IP address (preferred range: 192.168.40.10 ... 192.168.40.40) and the subnet mask of the Real-Time PC on which the project is to be compiled and run.

3.2 MATLAB/Simulink Models

This section contains information on integrating MATLAB/Simulink models:

- "Simulink Models" on page 38
 - "User-Defined C Code: Path Names" on page 38
 - "Function-Call Subsystems" on page 39
- "Restrictions regarding the Usable Simulink Models" on page 40
 - "S-Functions" on page 40
 - "Blocks with Absolute Time" on page 41
 - "Blocks for which no Parameters and Outputs are Generated" on page 41
- "Creating a Simulink Module" on page 41
- "Further Settings for the Simulink Model" on page 43

Software Requirements

If your LABCAR project is to use Matlab/Simulink models, you need

- to install MATLAB
 - A list of supported releases can be found in the Release Notes (via the menu [? → Help](#)).
- a license for LABCAR-MCS V5.4.4 (Modeling Connector for Simulink)

3.2.1 Simulink Models

The Real-Time Workshop generates C code from Simulink models – there are, however, a few points described in this section which have to be observed at all times.

Note

*Please make sure that only **one** version of the Real-Time Workshop is used in code generation! Problems may occur if you want to integrate several models in your project that were created with **different** versions of MATLAB/Simulink!*

User-Defined C Code: Path Names

The Real-Time Workshop makes it possible to integrate code defined by the user. This can be an s-function or other code called by an s-function.

If the project is created for an existing model, the original and its complete directory (with all subdirectories) are either copied to the project directory or simply referenced enabling a version control of the model.

For purposes of compiling and linking, the relevant files are copied to where the build process is to take place. This entails a few restrictions with regard to user-defined code which belongs to a Simulink model:

- In C, the #include command can reference a file with any extension, although *.h is the standard extension. LABCAR-MCS V5.4.4 only supports *.h and *.c as file extensions – if other extensions are used, the compiler may not find the relevant file.

- In C, the #include command can reference a file via a path, although the default is to use just a file name with relevant path information being added via corresponding compiler switches.

For example:

```
#include file_name.h
#include subdirectory\file_name.h
#include ..\parallel_directory\file_name.h
```

LABCAR-MCS V5.4.4 only supports the first two variants – other forms of referencing can lead to the compiler not being able to find the relevant file.

- The content of the various include search paths is copied to a specific directory to be compiled, the same directory to which the files specified when linking external files are copied (see "To link to other files ("Link to External Files")" on page 44).

Make sure that there are not several files of the same name and different content in these directories. Otherwise, depending on the order in which copying takes place, some files may be mistakenly overwritten.

Function-Call Subsystems

The model used may contain function-call subsystems or the user may be using model parts which are not to be executed in the small computing interval and the user therefore creates function-call subsystems. Please take the following into consideration in such cases.

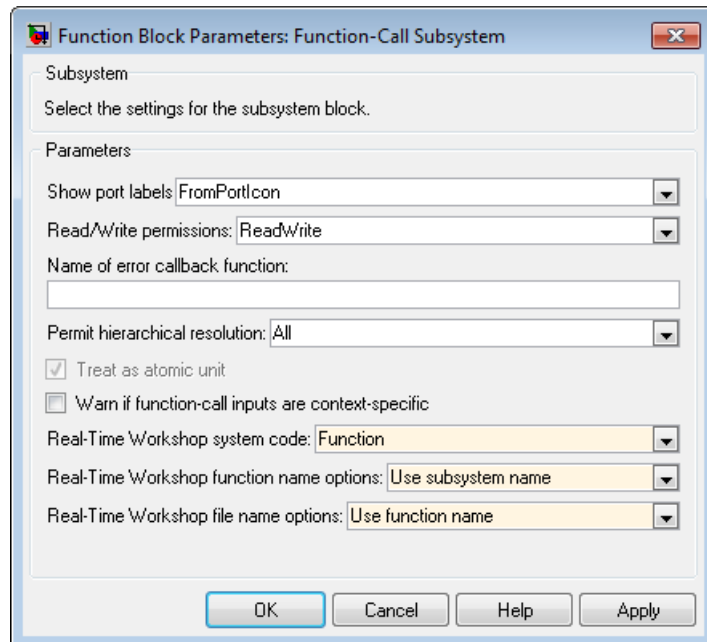
With suitable parameterization of these blocks, the Real-Time Workshop generates a separate process for the model part it contains. This model part is not calculated with the simulation computing interval on the target, or in other words: it is not calculated at all until the user attaches it to a task (which can be time- or event-triggered).

It depends on the parameters of this kind of function-call subsystem whether a separate process is generated or not. If you want this kind of process to be generated, proceed as follows.

To select settings

- Right-click the "Function-Call Subsystem" block in the Simulink model.
- Select **Subsystem parameters**.
The "Block Parameters: Function-Call Subsystems" window opens.

- Select the settings shown in the following figure (R2007b):



The process generated with these settings is assigned the name `<model_name>_<subsystem_block_name>`.

Note

This process is not displayed in the OS configuration (see "Configuring the Real-Time Operating System (OS Configuration)" on page 243) until the project has been newly built in LABCAR-IP ([Project → Build](#)).

3.2.2 Restrictions regarding the Usable Simulink Models

The Real-Time Workshop has some restrictions regarding the Simulink models that can be used: these are described below.

S-Functions

The Real-Time Workshop can only process S-functions which contain parameters of the data type "double". Code generation is aborted if another data type is found.

In addition, the Real-Time Workshop cannot process s-functions which contain outputs with a self-defined data type (declared with "ssRegisterDataType"). In this case, C code is generated successfully, but an error message is issued during code compilation as a macro replaces the invoking of "ssRegisterDataType" with "ssRegisterDataType_cannot_be_used_in_RTW".

Blocks with Absolute Time

The Real-Time Workshop cannot process blocks which use absolute time (e.g. "Sine Wave", "Clock", "Pulse Generator", "Signal Generator", "Transport Delay").

C code is generated, and compiled, successfully for a model that contains blocks of this type, but the blocks return incorrect results on execution.

Blocks for which no Parameters and Outputs are Generated

No parameters or outputs are created for a range of model blocks - this applies to the following blocks:

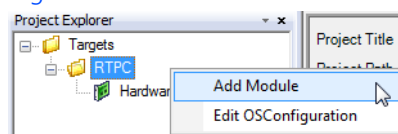
- Scope
- Floating Scope
- Display
- XY Graph
- Inport
- Outport
- Memory
- Goto
- ToWorkspace
- Stop
- FromIf
- Ground
- EnablePort
- GotoTagVisibility
- TriggerPort
- SignalSpecification
- Selector
- Terminator

3.2.3 Creating a Simulink Module

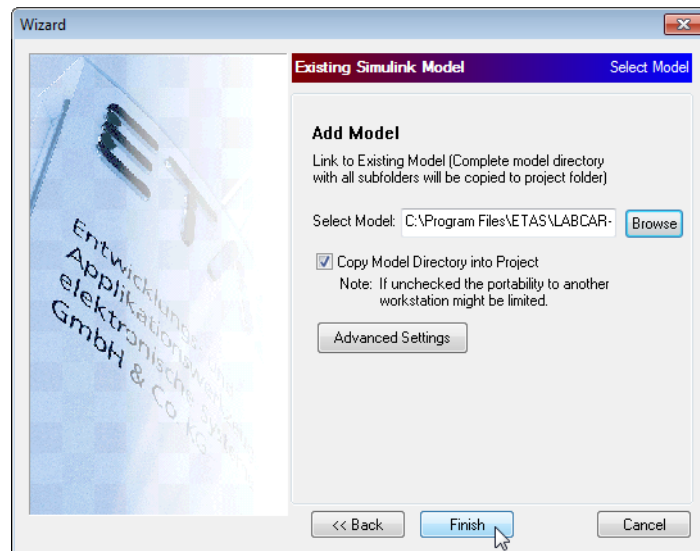
Use the module wizard to add a new Simulink module.

To add a Simulink module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select **Add Simulink(TM) Module**.
- Click **Next**.
- Select **Use existing Simulink(TM) Model**.
- Click **Next**.
- Use **Browse** to select the Simulink model you want to use.



- When the option "Copy Model Directory into Project" is selected, the Simulink model and all corresponding files are copied into the project directory – otherwise the model is only referenced.

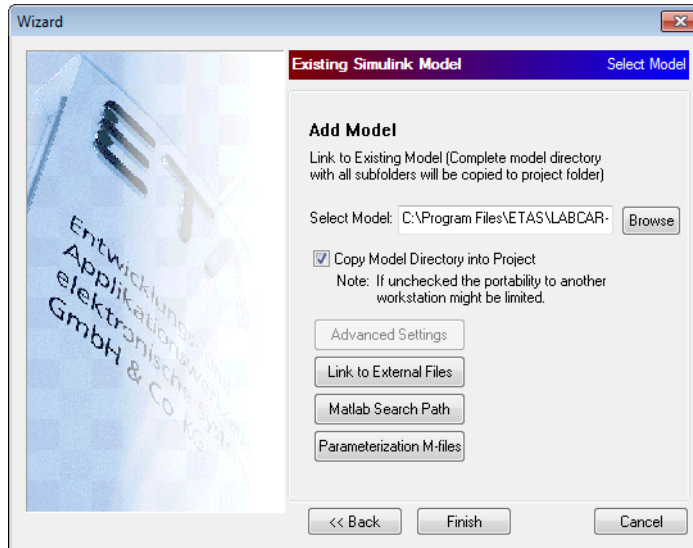
Note

If the model is only to be referenced, this reference has to be adapted for the migration of the LABCAR-OPERATOR project. The (absolute) path to the model directory is in the text file <model_name>.conf (in the directory <project_name>\Target_RTPC\SimulinkModels\Simulink_<model_name>) and can be adapted as a relative or absolute path specification (in accordance with the new position of the LABCAR-OPERATOR project).

- Click **Finish**.
In the "Project Explorer", a new module is added.

3.2.4 Further Settings for the Simulink Model

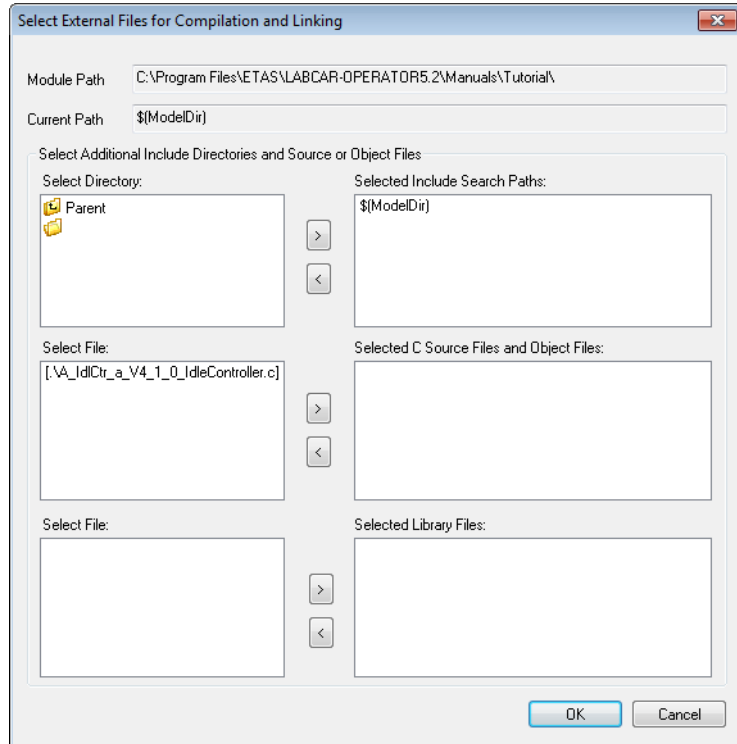
When you specify the Simulink model to be used when creating the project, the **Advanced Settings** button in the same window will allow you to access further settings for the Simulink model.



If additional include directories and source or object files are required for compilation, you can specify these here. You can also select object libraries which are linked in the build process.

To link to other files ("Link to External Files")

- Click **Link to External Files**.
The following window opens.



Note

Library files must have the extension ".a" or ".lib" to be displayed in the list.

- Select the relevant directories and files using the mouse and add these files to the lists on the right-hand side of the window using the > buttons.

Note

You can only select files and directories which are in subdirectories of the directory that contains the model.

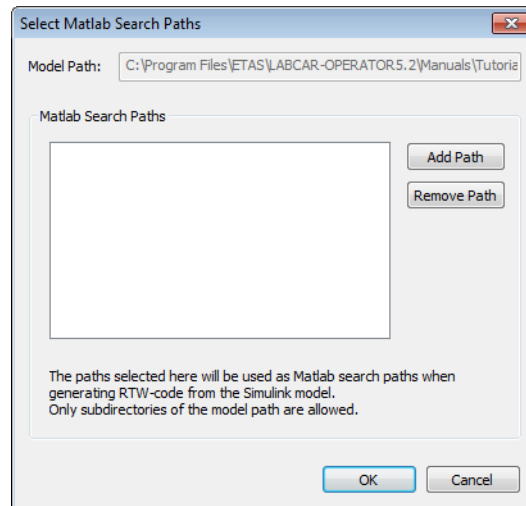
- Click **OK**.

To add MATLAB search paths to the project, proceed as follows:

To add MATLAB search paths ("Matlab Search Paths")

- Click **Matlab Search Paths**.

The following window opens.



- Click **Add Path**.

A directory browser opens from which you can select a directory.

Note

Only directories under the model path can be selected ("Model Path")

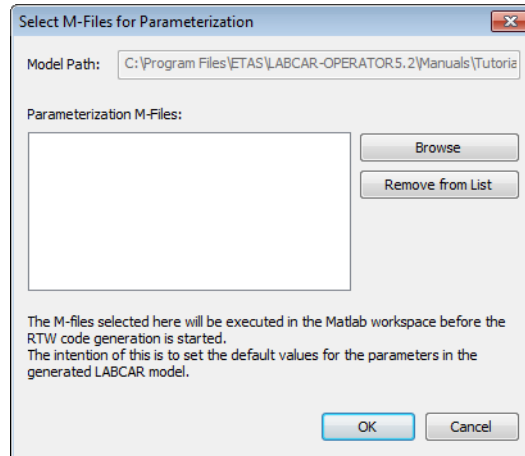
The selected paths are displayed in the list.

- If you want to remove a search path from the list, select it in the list and click **Remove Path**.
- Click **OK**.

To select parameterization files which are evaluated when the Simulink model is opened, proceed as follows:

To select parameterization files ("Parameterization M-Files")

- Click **Parameterization M-Files**.
- The following window opens.

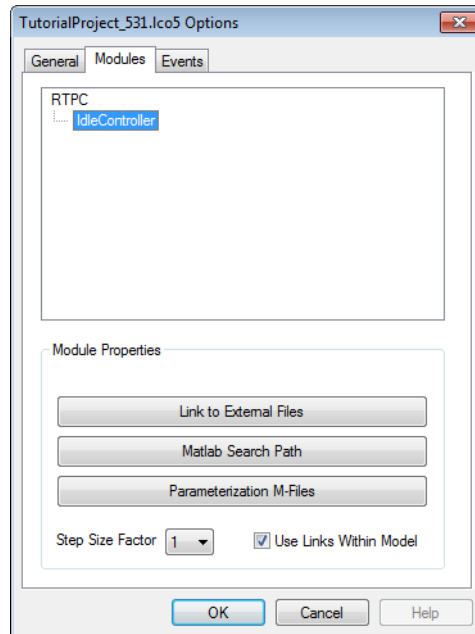


- Click **Browse**.
A file selection window opens in which you can select a file.
The selected file is displayed in the list.
- If you want to remove a file from the list, select it and click **Remove from List**.
- Click **OK**.

To ensure that it is possible to change the settings described above throughout your work with the project and not just during project creation, the dialog boxes described can always be accessed:

To change settings

- Select **Project** → **Options**.
The "*project_name* Options" window opens.
- Select the "Modules" tab.



You can edit all the settings described above for the relevant model (in the window on the left).

Integration step size (Simulink model) and task period (ETAS EE)

As in many cases it is sensible to run the calculation of the model (defined by the fixed-step-size of the Simulink model) much more frequently than the processing/display in the experiment environment (defined by the task period in the OS Configuration), larger values (factor $n = 1..10$) can be configured for the task period dT_{EE} than for the integration step size fss_{SL} :

$$\text{Gln. 3-1} \quad fss_{SL} = dT_{EE} / n$$

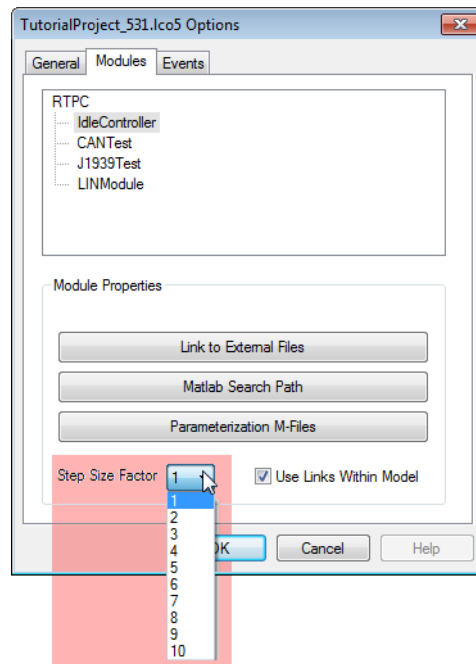
Note

This factor n can be different for each Simulink model of the LABCAR-OPERATOR project!

Proceed as follows:

- Select **Project** → **Options**.
- The "*project_name* Options" window opens.
- Select the "Modules" tab.
- Select the Simulink model you want to make this setting for.

- Specify the required factor.



- Confirm with **OK**.
- Save the change with **File** → **Save**.

This means the step size fss of the Simulink model is set as a fraction (1/n) of the task period defined in the OS Configuration.

To enable/disable Matlab links support

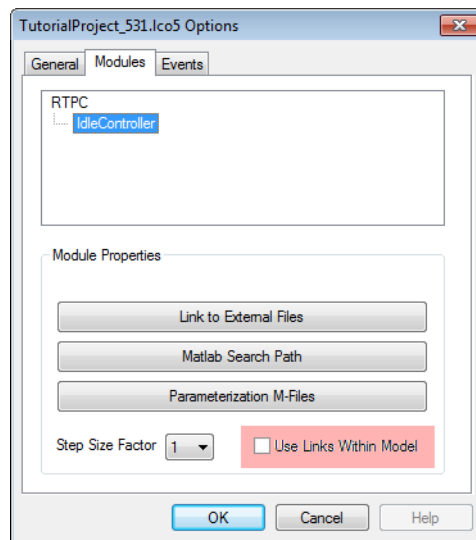
Since Version 5.3.1, Matlab links for outports and inports have been supported by default. To generate code as in earlier versions, you can disable this support.

- Select **Project** → **Options**.
- The "*project_name* Options" window opens.
- Select the "Modules" tab.
- Select the Simulink model you want to make this setting for.

Note

This option can be selected individually for each Simulink model of the LABCAR-OPERATOR project!

- Disable the option **Use Links Within Model**.



- Confirm with **OK**.
- Save the change using **File** → **Save**.

3.3 ASCET Modules

This section contains information on integrating ASCET modules in a LABCAR-OPERATOR project.

- "General Information on ASCET Modules" on page 50
- "Preparations" on page 50
- "Creating an ASCET Module" on page 52
- "Measure Variables and Parameters in the ETAS Experiment Environment" on page 55

3.3.1 General Information on ASCET Modules

After code generation for the "RTPC" target and transfer, and in addition to a range of code and header files, the ASCET project particularly makes the following files available:

- `<project_name>.six`
- `<project_name>.oil`

The `<project_name>.six` file is a SCOOP-IX file¹ that contains a complete interface description of the ASCET module. The ASCET project can be integrated in the LABCAR-OPERATOR project as a module with the information in this file.

The `<project_name>.oil` file contains the configuration of the operating system (tasks, processes etc) that can be included in the integration of the ASCET project if required.

3.3.2 Preparations

Installing ASCET

To ensure you can integrate ASCET projects as modules in LABCAR-OPERATOR, you require an up-to-date ASCET installation (Version 6.0.1 or higher), including the components ASCET-SE, ASCET-MD and ASCET-RP.

Preparing ASCET

Each ASCET version also requires a specific add-on feature to ensure it can export projects for the "RTPC" target.

Note

This feature does not need to be installed manually with later versions of ASCET (V6.1.0 and higher).

The necessary file is on the launch screen of the installation CD under **Installation**.

Click **Install RTPC export for ASCET 6.x.y (LABCAR-ASC)** and follow the instructions of the installation program.

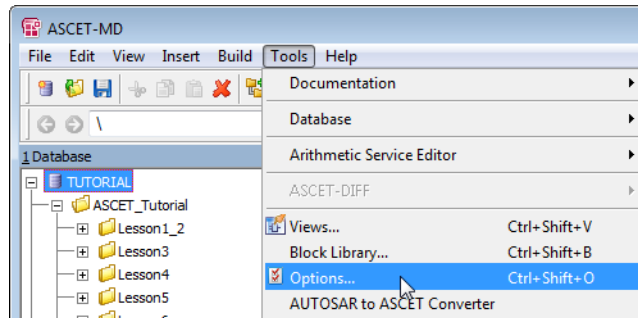
¹ For detailed information on "SCOOP-IX", refer to the INTECRIO V3.0 User's Guide

Settings in ASCET

The transfer of the project from ASCET requires a specific setting for an ASCET option.

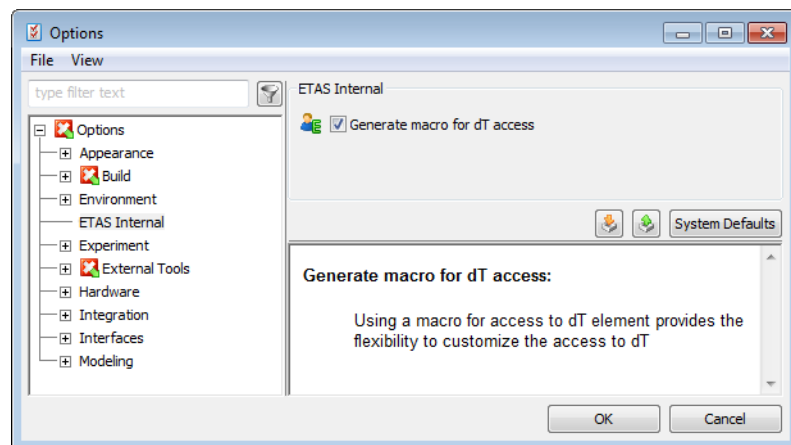
To set options

- Start ASCET.
- Select **Tools** → **Options** from the main menu.



The "Options" window opens.

- Select "ETAS Internal" on the left and activate the option "Generate macro for dT access".



- Click **OK** to close the window.

3.3.3 Creating an ASCET Module

This section describes how to integrate an ASCET project in LABCAR-OPERATOR.

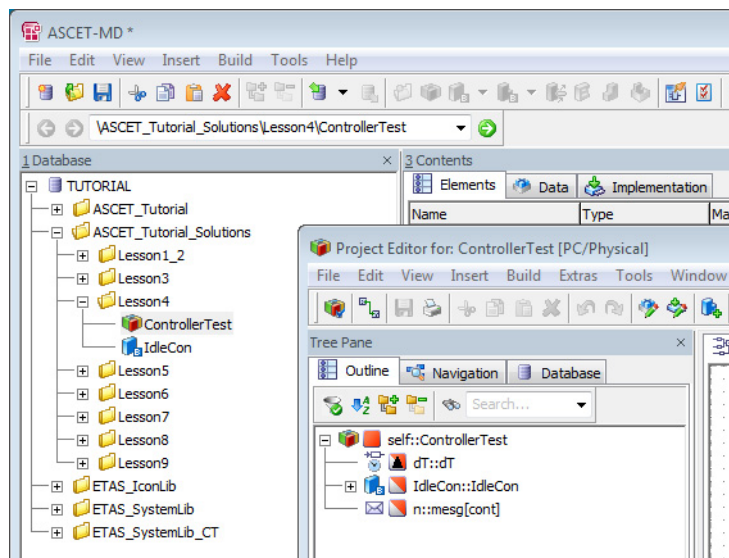
Note

If the project is in a version of ASCET < V6.0.1, you must first export the project (or the entire database) from the older version and import it into the new version.

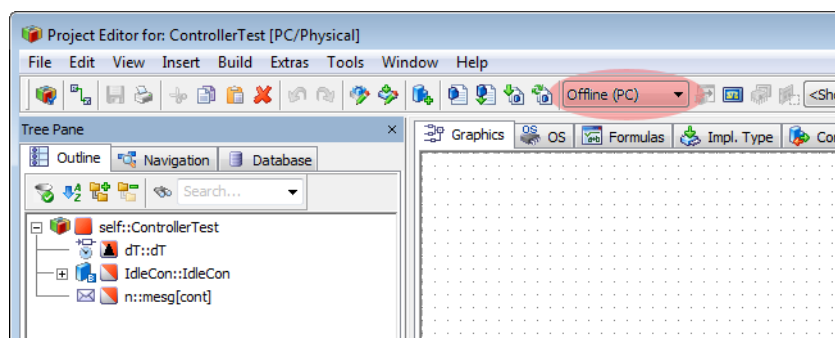
To transfer the ASCET project

- Open the relevant database in ASCET.
- Double-click the required project (in this example this is the "ControllerTest" project in the "Lesson4" folder of the "TUTORIAL" database).

The project opens in the Project Editor.

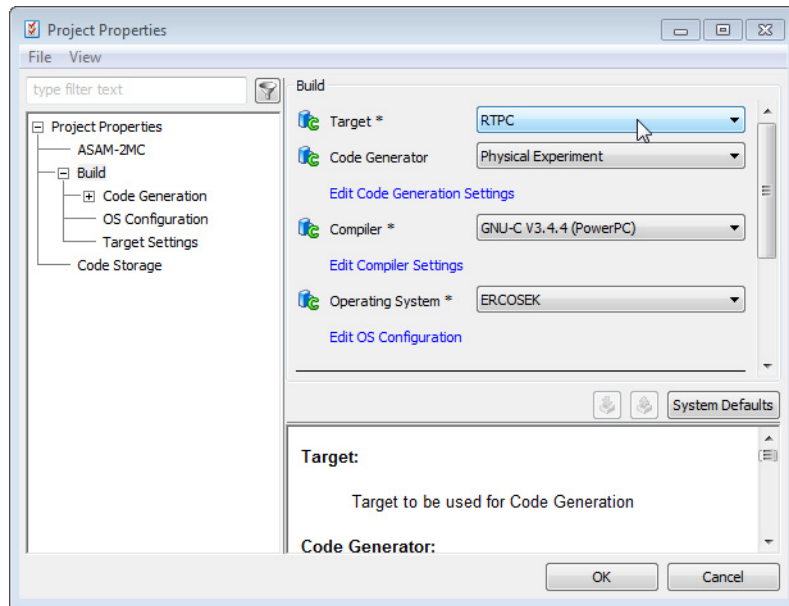


Currently the target for offline simulation on the PC is selected.

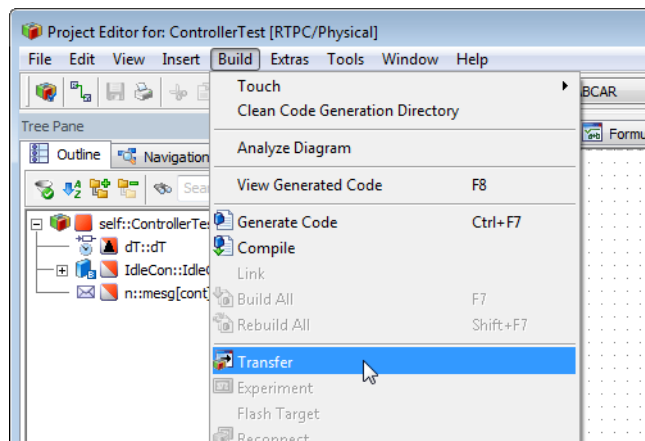


- In the Project Editor, select **File** → **Properties**. The "Project Properties" window opens.

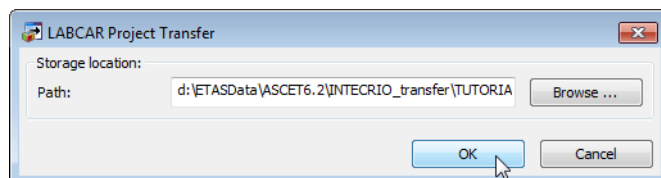
- Select "Build" on the left and "RTOS" beside Target.



- Click **OK**.
- Select **File** → **Save** from the ASCET main window.
- Select **Build** → **Touch** → **Recursive** from the main menu of the ASCET Project Editor.
- Select **Build** → **Transfer**.



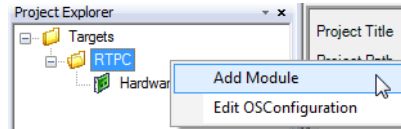
- Select a directory into which the files to be generated should be stored.



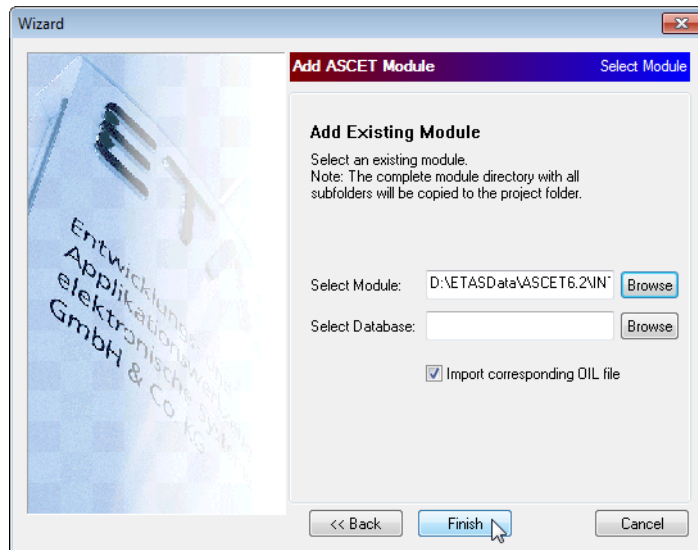
The files *.c, *.h, <project_name>.six and <project_name>.oil are stored in this directory.

To add an ASCET module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select **Add Ascet Module**.
- Click **Next**.
- Select **Use existing ASCET Model**.
- Click **Next**.
- Select the description file `ControllerTest.six` of the module to be added.



- If you specify the ASCET database the exported model comes from (under "Select Database"), it opens when the ASCET module is double-clicked (in the Project Explorer).

If no database is specified here, double-click and then enter the path manually.

- Click **Finish**.
- In the "Project Explorer", a new module is added.
- Save the project.

For those receive messages of the ASCET project that are not connected to send messages of the ASCET project, inports are generated during import that can be connected to other signals of the proj-

ect in the Connection Manager. The same is true of send messages of the ASCET project and outports in the Connection Manager.

- Establish the connections to other modules in the Connection Manager.
- Generate the code for the experiment (**Project → Build**).
- Open the experiment in ETAS EE (**Tools → Open Experiment Environment**).

3.3.4 Measure Variables and Parameters in the ETAS Experiment Environment

The ASCET Project and ETAS EE

The components of the ASCET project (in the "Lesson4" folder of the "TUTORIAL" database), that was used in the above example, are shown in Fig. 3-4.

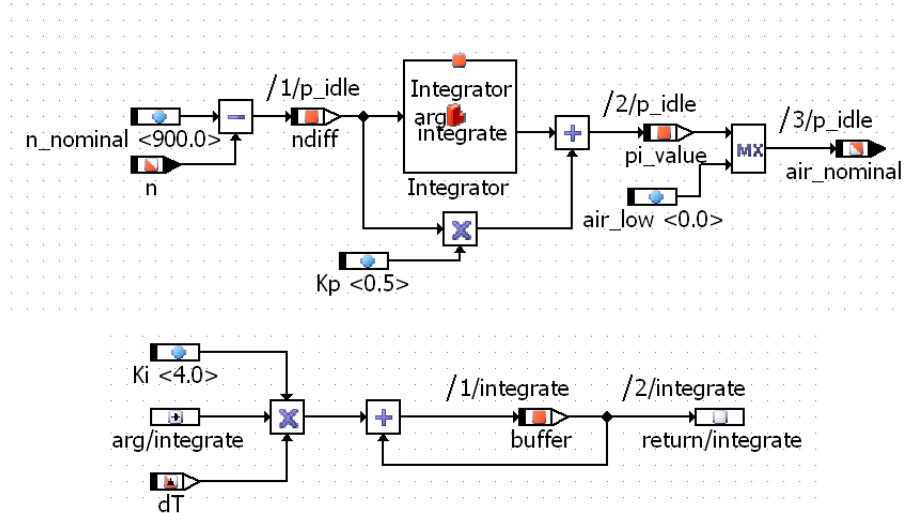


Fig. 3-4 The Components "IdleCon" (top) and "Integrator" (bottom) of the "ControllerTest" Project

The idle controller compares the actual speed "n" with the target speed "n_nominal" and uses these values to control the target air supply "air_nominal".

The components, measure variables and parameters of the ASCET project are shown in Fig. 3-5.

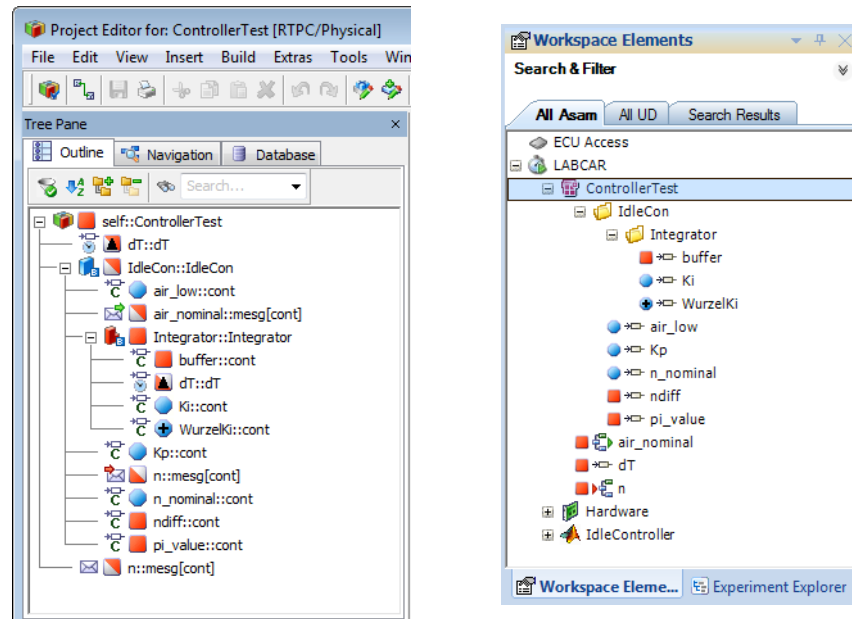


Fig. 3-5 The ASCET Project "ControllerTest" (on the left) and the integrated Module in the "Workspace Elements" Window of ETAS Experiment Environment (on the right).

3.4 C Code Modules

In addition to MATLAB/Simulink models and ASCET modules it is also possible in LABCAR-IP to integrate C code defined by the user into the project.

This particularly enables the following:

- Simple integration of all kinds of functions for inexpensive but flexible user-specific solutions
- Creation of simulation models suitable for LABCAR with third-party tools
- Reusing older LABCAR code
- Integrating functional mock-up units

This chapter contains a tutorial which illustrates the various ways of creating C code modules. It includes information on the following topics:

- "Creating a C Code Module Manually" on page 58
- "Adding Code" on page 64
- "Creating a C Code Module with the Automation Server" on page 68
- "Exporting a C Code Module" on page 68
- "Adding an Existing C Code Module" on page 68
- "Changing an Existing C Code Module" on page 70
- "Linking External C Code" on page 70
- "Variable Labels" on page 72
- "Functional Mock-up Units (FMU)" on page 72

3.4.1 Tutorial

This tutorial describes how to integrate C code in a LABCAR project using C code modules.

This integration includes the following steps:

- Specification of the interface of the C code module
 - either manually (see "Creating a C Code Module Manually" on page 58)

or

- using the automation interface (see "Creating a C Code Module with the Automation Server" on page 68)
- Adding the code to the interface created

These steps are executed below using a simple code example.

The Code Example

The example is a very simple brake system. The inport of the brake system is made available by the brake pedal setting (0...1). The system output consists of four values M_LF, M_RF, M_LR, M_RR which contain the relevant brake torques which have an effect on the four wheels.

The outputs are calculated as follows:

First of all the pedal setting (pedal) is converted to hydraulic brake pressure using a factor (pedal_pressure_factor). Then, the individual brake torques are calculated using further conversion factors (pressure_torque_factor[]).

The module is thus written by the following equations:

$$M_{LF} = \text{pressure} * \text{pressure_torque_factor}[0]$$

$$M_{RF} = \text{pressure} * \text{pressure_torque_factor}[1]$$

$$M_{LR} = \text{pressure} * \text{pressure_torque_factor}[2]$$

$$M_{RR} = \text{pressure} * \text{pressure_torque_factor}[3]$$

where

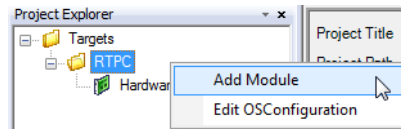
$$\text{pressure} = \text{pedal} * \text{pedal_pressure_factor}$$

3.4.2 Creating a C Code Module Manually

Use the module wizard to add a new C code module manually.

To add a C code module

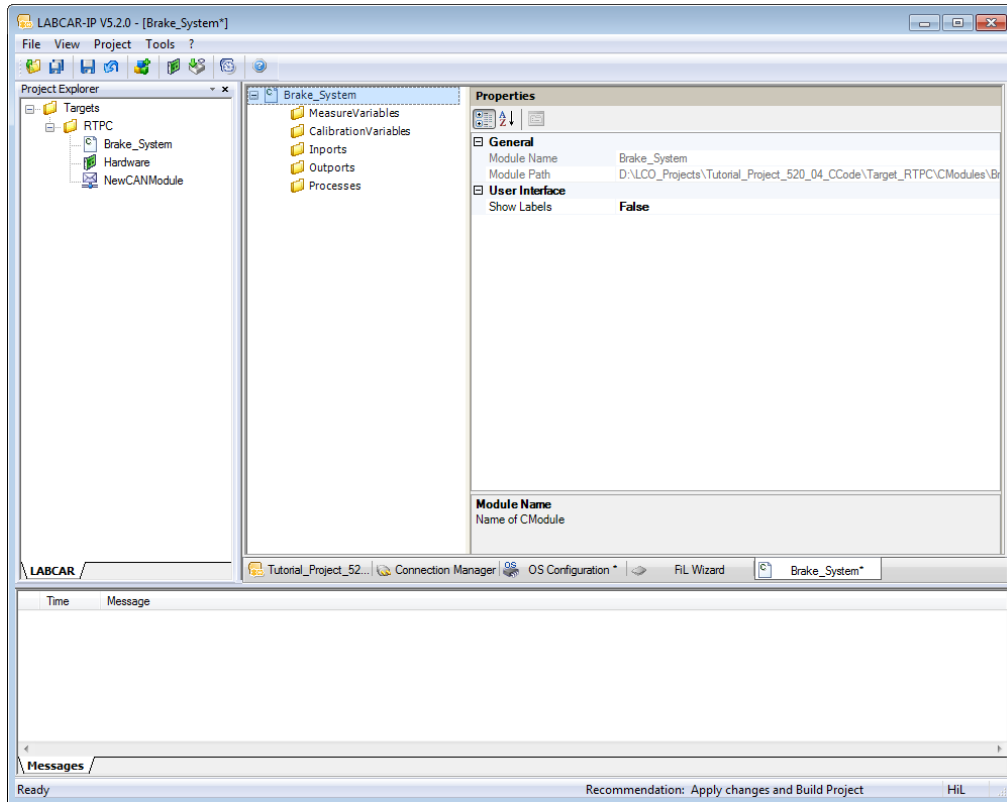
- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select **Add C-Code Module**.
- Click **Next**.
- Select **Create new Module**.
- Click **Next**.
- Enter a module name.
- Click **Finish**.

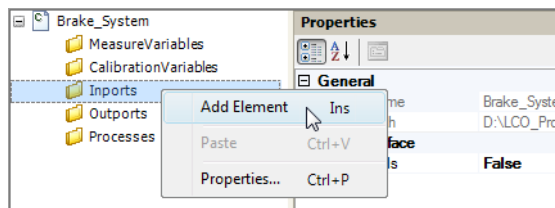
In the "Project Explorer", a new module is added.

- Double-click the module.
- The tab with the C code editor opens.



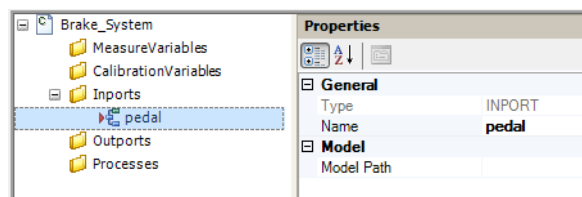
To define a module import

- To define an inport, right-click the "Inports" folder.
- Select **Add Element** from the shortcut menu.



An "Inport" element is created.

- Select the element and specify a name ("pedal") for the inport (in the "Properties" window beside "Name").



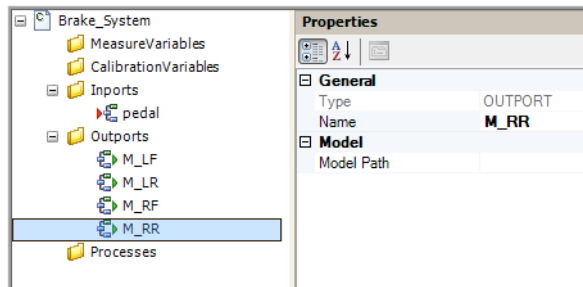
- Click **File** → **Save All**.

Note

Inports and outports are always scalars of the data type "double". This means that no other details are necessary.

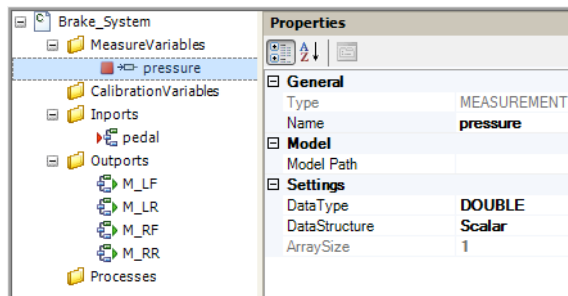
To define module outputs

- To define a module output, right-click in the "Outports" folder.
- Select **Add Element** from the shortcut menu.
- Enter a name ("M_LF") for the output.
- Create three additional outputs with the names "M_RF", "M_LR" and "M_RR" (as described above or via **Copy** and **Paste**).



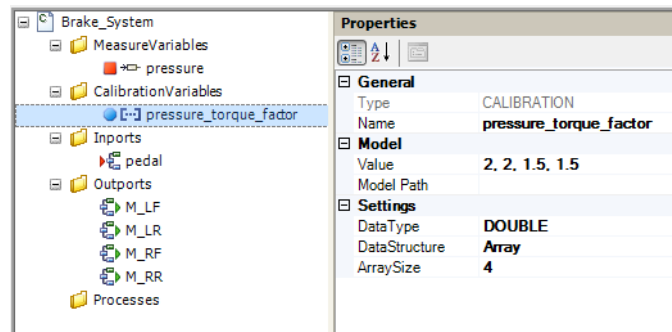
To define a measure variable

- To define a measure variable, right-click the "MeasureVariables" folder.
- Select **Add Element** from the shortcut menu.
- Enter a name ("pressure") for the output.
- Select "DOUBLE" as "Data Type".
- Select "Scalar" under "DataStructure".



To define calibration variables

- To define a calibration variable, right-click the "CalibrationVariables" folder.
- Select **Add Element** from the shortcut menu.
The entries necessary here are basically the same as those for a measure variable except that a calibration variable is assigned a value.



- Name the variable "pressure_torque_factor".
- Select "DOUBLE" as data type.
- Select "Array" with "ArraySize" = 4.
- The value is assigned under "Value", with the individual values being separated by commas.
- Create another calibration variable called "pedal_pressure_factor" and assign it the value 200.

Processes

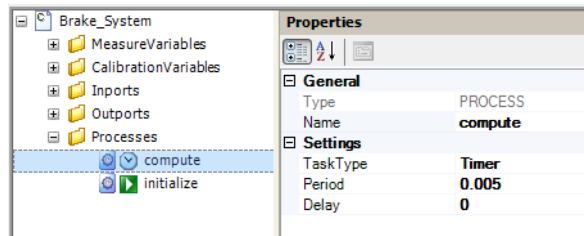
A process is a function which is called by the operating system – the type of call depends on the process type:

- An Init process is called once when the experiment is started.
- An Exit process is called once when the experiment is ended.
- A Timer process is called at a special rate (defined by "Period" and "Delay")
- It is not, however, absolutely necessary to assign a process to a task – this is then referred to as an "Event" process.

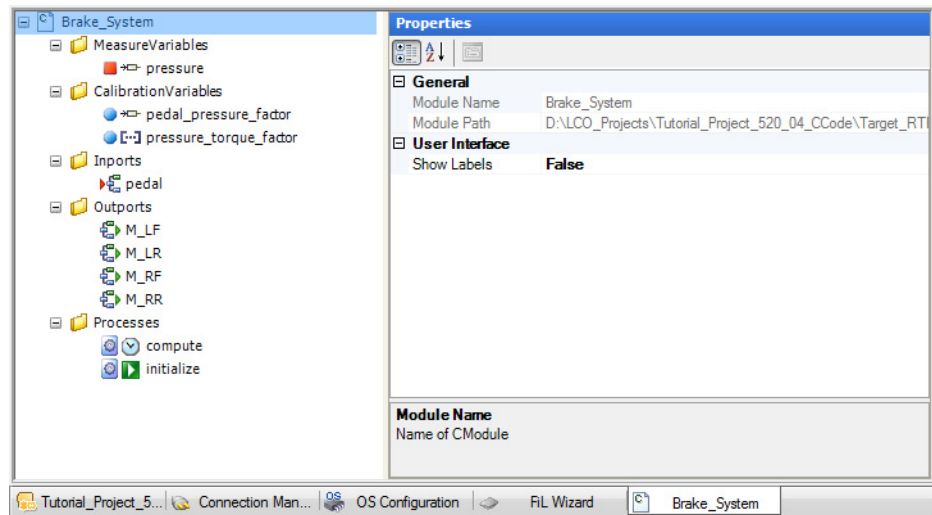
To add a process

- To define a process, right-click the "Processes" folder.
- Select **Add Element** from the shortcut menu.
- Enter "initialize" as "Process Name".
- To assign this new process to a task, select **Specify Task (optional)**.
- Select the Task Type "Init".
- Click **OK**.

- Define another process called "compute".
- Select "Timer" as "Task Type" with the following settings:
 - Period = 0.005 s
 - Delay = 0 s

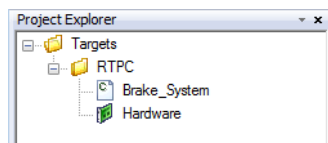


The "Brake_System" tab should now be as follows:



- Select **File** → **Save**.

The module just created is saved and added to the project.

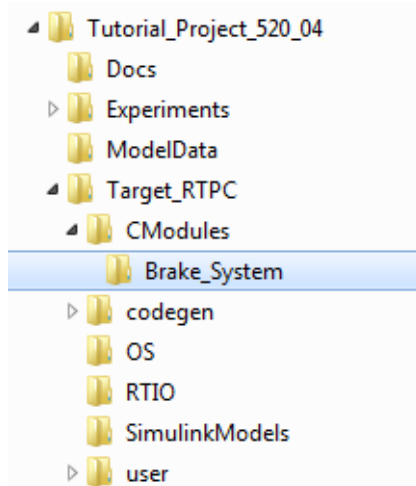


The following steps are executed once you have saved the project:

- Checking the data for consistency
 - This is where a check is executed to see, for example, whether the module name is valid or the names of the variables and processes are unique.

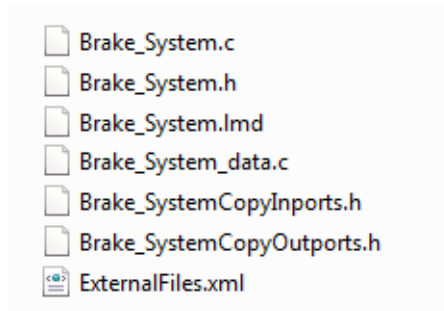
- Creating the directories

A directory is created in the project folder for the module called "Brake_System".



- Generating the files

The following files are created in the above-mentioned directory:



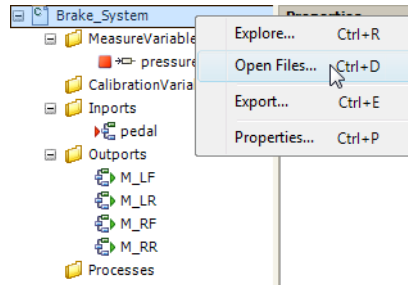
In addition to the files containing the C code and further definitions (*.c and *.h), the *.lmd file is also created.

- Integration in the data model of LABCAR-OPERATOR.

To edit source and include files

To open the source (*.c) and include files of the module in the editor assigned (e.g. Visual Studio), proceed as follows:

- Select **Open Files** in the shortcut menu of the C code module.



The files belonging to the module are opened in the editor selected.

3.4.3 Adding Code

The actual code, as created from the specifications made to date, is in the `Brake_System.c` file. This file is a template with "empty" processes to which code now has to be added.

The content of the file is shown below; the part in bold print is the code added for the brake system.

```
// Add application-specific include statements here:

#include "connect.h"
#include "Brake_System.h"

// Init-triggered function "initialize":
void cmod_initialize_Brake_System()
{
    // Get Inports
    #include "Brake_SystemCopyInports.h"

    // Enter your code here:

    // Set Outports:
    #include "Brake_SystemCopyOutports.h"
}

// Timer-triggered function "compute":
void cmod_compute_Brake_System()
```



```

{
    // Get Inports
    #include "Brake_SystemCopyInports.h"

    // Enter your code here:
    pressure = pedal * pedal_pressure_factor;
    M_LF = pressure * pressure_torque_factor[0];
    M_RF = pressure * pressure_torque_factor[1];
    M_LR = pressure * pressure_torque_factor[2];
    M_RR = pressure * pressure_torque_factor[3];

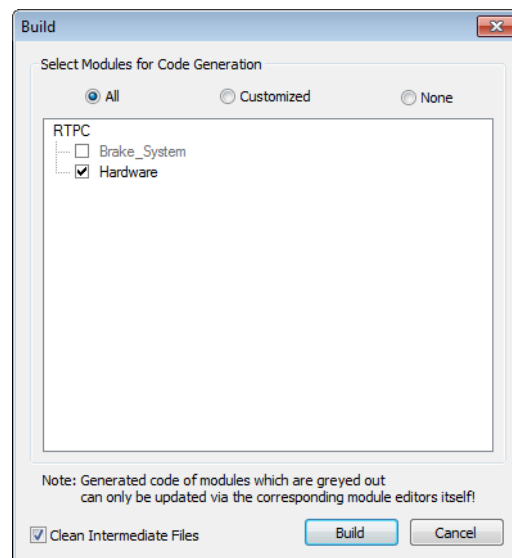
    // Set Outports:
    #include "Brake_SystemCopyOutports.h"
}

```

The variables defined when creating the C code module can be used in the code added. The declaration of these variables is contained in the "Brake_System.h" file. As far as the "initialize" function is concerned, it is not required in the example – it is only used for demonstration purposes here.

This completes the creation of the C code module.

Now select **Project → Build** to create the project.

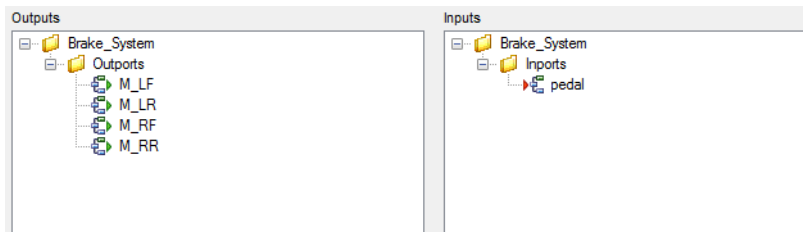


The C code module "Brake_System" cannot be selected in the "Build" window as code no longer has to be generated here. Click **Build**.

Once the build process has been completed successfully, all inports and outports, measure and calibration variables are in the corresponding windows of the LABCAR-OPERATOR user interfaces.

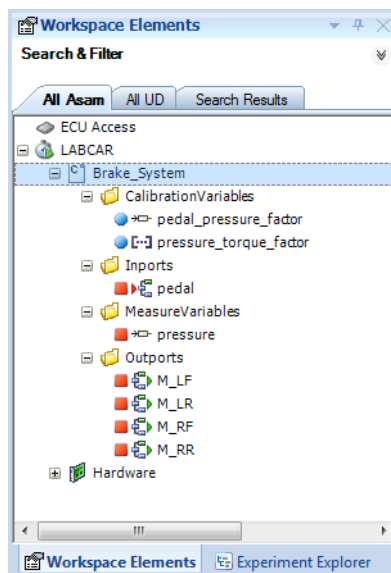
Connection Manager

The inports and outputs are available in the Connection Manager (in LABCAR-IP) for connection to other modules. If necessary, select **Update Ports**.



Workspace Elements

When experimenting in ETAS EE, the inports and outputs as well as the measure and calibration variables of the module are available in the "Workspace Elements" window.

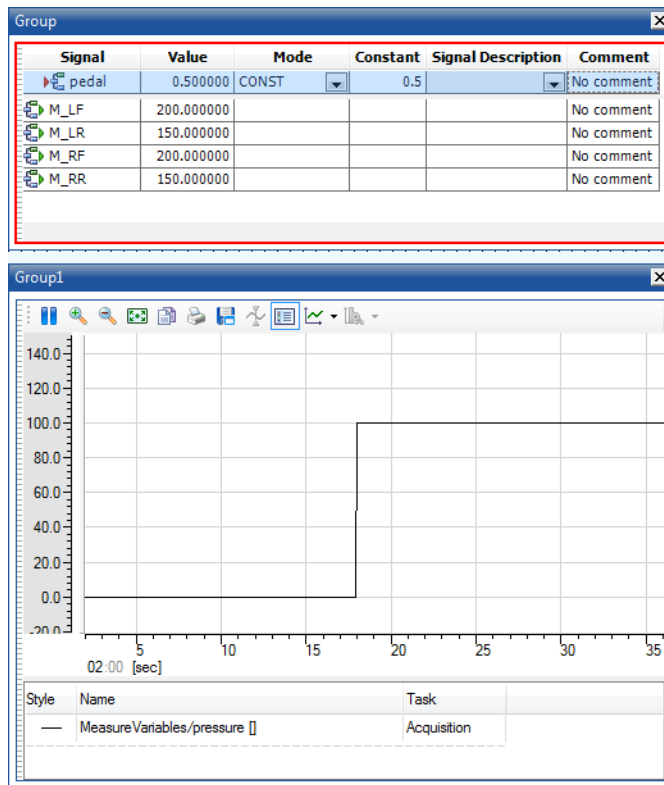


If you want to experiment with the module, execute a build and open the experiment in the experiment environment.

To run a simulation

- Create a Signal List.
- Add the inport and the four outputs to this Signal List.
- Create an oscilloscope and add the measure variable "pressure".
- Start the experiment.
- At the inport "pedal" set "Mode" to "CONST" and assign the inport the value 0.5.

- Press <RETURN>.



"pressure" now has the value "100" – the brake torque of the two front wheels is now 200 and that of the two rear wheels 150.

3.4.4 Creating a C Code Module with the Automation Server

The actions described in the previous section for the creation of a C code module, such as the adding of inports/outputs, variables etc. are also available via API functions. The "ICModuleManager Interface" provides the interface functions for this purpose.

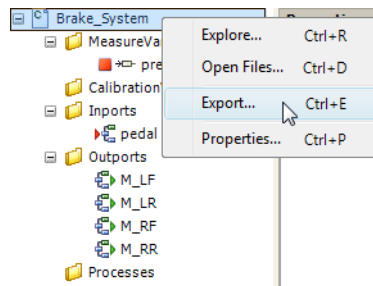
The relevant documentation (chm file) can be found via [? → Help](#) under "API Documentation".

3.4.5 Exporting a C Code Module

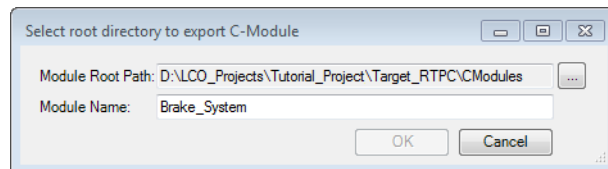
You can also export the C code module (i.e. the relevant files – see "Generating the files" on page 63) with a freely selectable name in order to add it to other projects if necessary.

To export the C code module

- Select **Export** from the shortcut menu of the C code module.



- From the following dialog, select the path and, if necessary, assign the module a different name.



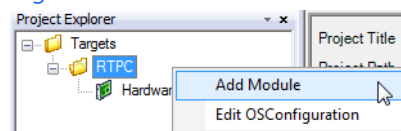
The module is exported under the name specified.

3.4.6 Adding an Existing C Code Module

Use the module wizard to add an existing C code module.

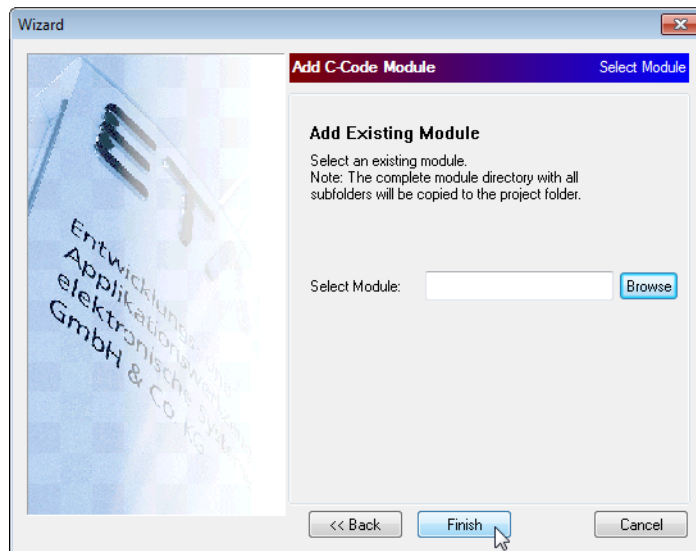
To add a C code module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.

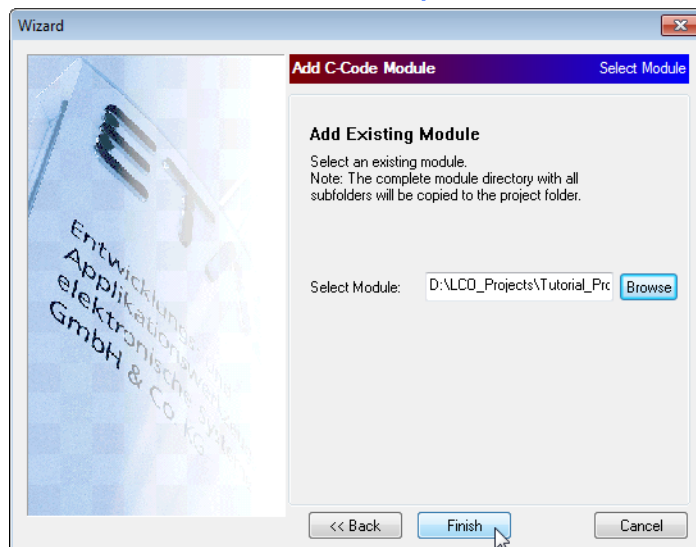


- In the "Add Module Wizard", select **Add C-Code Module**.
- Click **Next**.
- Select **Use existing Module**.
- Click **Next**.

The following window is used to select the module.



- Click **Browse**.
A file selection window opens.
- Select the file and click **Open**.



- Click **Finish**.
In the "Project Explorer", a new module "Brake System2" is added below the existing module "Brake_System".

Once you have again added the equations already added in the first example, this example also has the same simulation result. The only difference is in the changed paths of the measure and the calibration variables in the "Workspace Elements" window in ETAS EE.

3.4.7 Changing an Existing C Code Module

If you modify an existing C code module, you may have to modify the interface, e.g. by adding, editing or removing variables or processes.

If you then select **File** → **Save**, five of the files described above are newly created; the file with the actual code `<Module_name>.c`, which also contains the added code, remains either unchanged or is extended.

The following rules apply:

- If modifications have only been made to the variables, this does not apply to the C file.
- If a new process has been added, the relevant function template is added to the code.
- If a process is removed, the relevant function remains.
- If a process is renamed, this is the same as if it had been removed and a new process added. This means that a new function template is added to the C file, but the function is empty. The code of the old function has to be transferred to the new function by the user by hand.

This procedure ensures that code is never deleted. If the function of a removed process remains in the file, code is generated but has no significance as long as the function is not called.

The only possible case in which code has to be deleted manually is if a process with a specific name is first deleted and then recreated with the same name. Two functions with the same name then exist in C code which is not permissible.

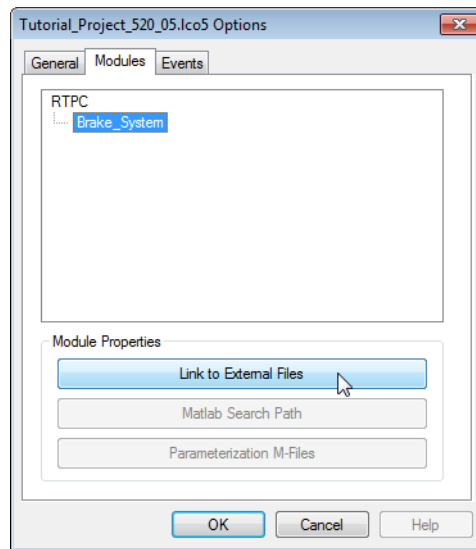
If code is attached to a file, a backup copy is always created beforehand carrying the old file name with an appended, consecutive number.

3.4.8 Linking External C Code

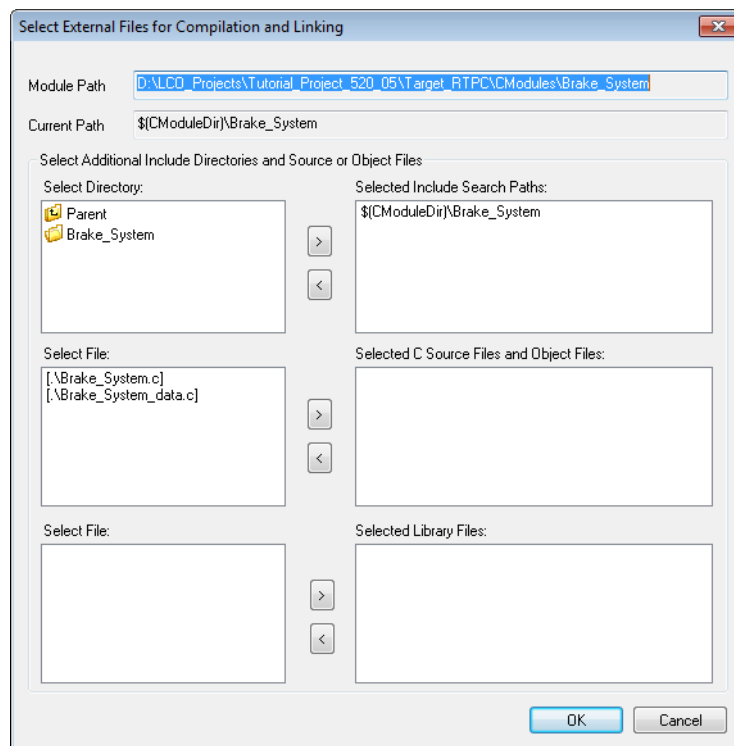
The C code file created by the editor has to be extended with C code defined by the user. This can take place with a piece of code inserted directly; references can also, however, be made to external sources.

- Select **Project** → **Options** in the main menu of LABCAR-OPERATOR.
- Select the "Modules" tab.
- Select the module for which the external files are to be linked.

- Click **Link to External Files**.



The window for selecting search paths, source and object files and libraries opens.



- Select the relevant files.
- Click **OK** twice.

3.4.9 Variable Labels

In addition to using standard labels, it is also possible to use labels defined by the user in LABCAR-OPERATOR. This applies both to the specification of a C code module by hand and creation via API functions.

This label corresponds to the path under which variables, inputs and outputs are displayed in the windows "Workspace Elements" (ETAS EE) or in the Connection Manager (inports and outports).

If labels are not specified explicitly, the following default labels are used:

- `<TargetName>/<ModuleName>/MeasureVariables/
<VariableName>`
- `<TargetName>/<ModuleName>/CalibrationVariables/
<VariableName>`
- `<ModuleName>/Inports/<VariableName>`
- `<ModuleName>/Outports/<VariableName>`

If a specific label is specified by the user, e.g. `User/Defined/Path` for a measure variable, the variable is shown in the relevant hierarchy as follows:

`<TargetName>/<ModuleName>/User/Defined/Path/<Variable-
Name>`

3.4.10 Functional Mock-up Units (FMU)

This section describes how to import functional mock-up units (FMU) into a LABCAR-OPERATOR project – special attention is paid to the creation of the shared object file for Linux.

The section contains information on the following topics:

- "Overview" on page 72
- "Adding an FMU in LABCAR-IP" on page 73
- "Generating the Shared Object File (.so)" on page 75
- "Notes and Limitations" on page 82

Overview

A functional mock-up interface (FMI) defines a tool-independent interface standard for the co-simulation and exchange of models from different sources.

By creating a software library called FMU (functional mock-up unit), models of a modeling tool can be run in a different tool environment.

There are various types of FMUs

- Model Exchange
This uses the solvers of the tool into which the FMU has been imported.
- Co-Simulation
This uses the solvers of the simulation tool from which the model comes.
In turn there are two different types of co-simulation:

- Standalone
With this type, the solvers are "embedded" in the FMU.
- ToolExchange
Methods that use the simulation tool solvers are called.

Note

LABCAR-OPERATOR V5.4.4 supports co-simulation (Version 1.0) of the type "Standalone".

Every FMU model (FMU = functional mock-up unit) is made available as a zip file with the extension "fmu". It contains the following:

- `sources`
This folder contains the source files with the C functions of the model equations.
- `binaries`
This folder contains subfolders with the binary files for the various platforms:
 - `win32 / win64`
 - `linux32 / linux64`
- `modelDescription.xml`
This file contains the definition of all variables of the model and further model information.
- Optional further folders with data (such as parameter tables, the user interface etc.) required by the model.

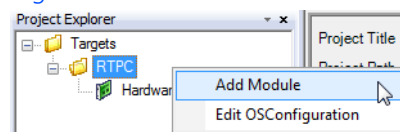
*Adding an FMU in LABCAR-IP***Note**

The FMU module can only be integrated if the FMU source files are available. If a Linux binary file (shared object file) is already available that was created in some other way than the method described below, the LABCAR-OPERATOR project may not work correctly.

We therefore highly recommend that you create the Linux binary file as described in the section "Generating the Shared Object File (.so)" on page 75!

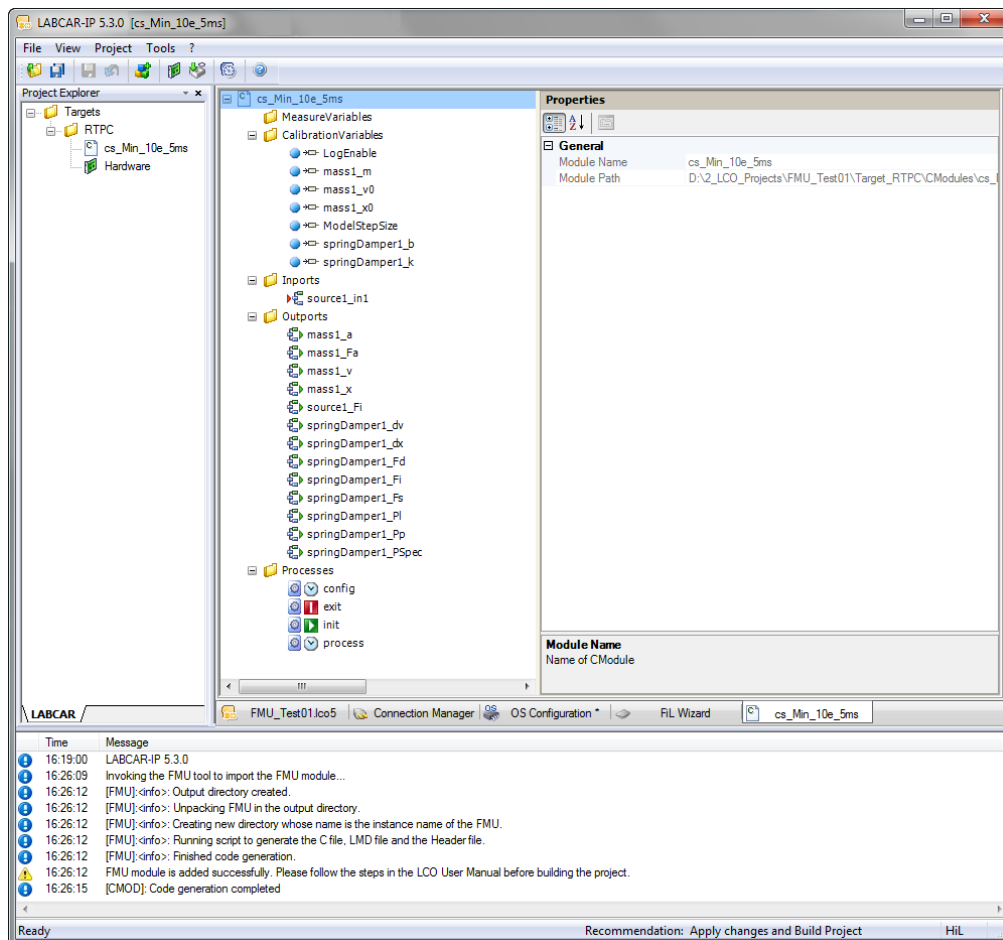
To integrate an FMU, proceed as follows:

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select **Add C-Code Module**.
- Click **Next**.
- Select **Use existing FMU Module**.
- Click **Next**.
- Specify the path to the *.fmu file and enter a module name (must be a valid C identifier).
- Click **Finish**.

In the "Project Explorer", a new module is added..



Before the project is built, the shared object file (.so) to be generated must be copied into the project (see "Generating the Shared Object File (.so)" on page 75).

- Copy the ".so" file to the following directories:
Target_<TargetName>\User\lib
Target_<TargetName>\runtime-data\lib
- The project can now be built.

Generating the Shared Object File (.so)

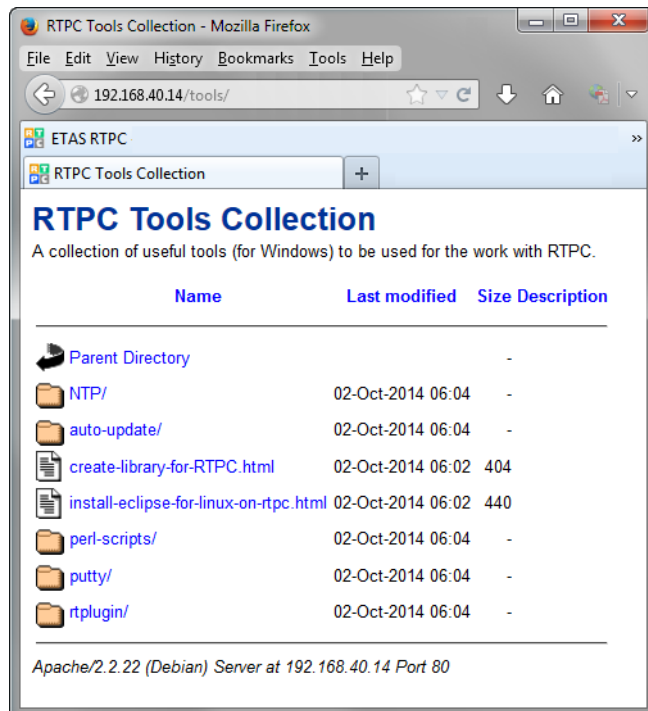
A shared object file is a binary file that contains all functions of the FMU. It must be generated from the source files (as described below).

To generate the shared object file

When integrating an FMU in LABCAR-IP, the C source files are copied to the project directory in the FMU module folder. If the LABCAR-OPERATOR project was created in the D:\Temp directory and the name of the FMU module is "Engine", the source files of the FM are in the directory

```
D:\Temp\<LCO Project Directory>\Target_RTPC\
  CModules\Engine\FMU_Sources\
```

- Zip all files **in the FMU-Sources folder (not the folder itself!)** into one file.
This ensures that the make file `makefile_labcar` is in the main directory.
- Open the web interface of ETAS RTPC.
- Stop the simulation controller.
- Return to the main page and click **Tools**.



- Click the [create-library-for-RTPC.html](#) link.



[Main Page](#) >> [User Library](#)

Build User Library

User Library Files

total 0

Upload Archive File: No file selected.
Supported are zip, tar.gz and tgz files.

Delete:

© 2003-2014 ETAS GmbH

- Click **Delete All Files**.
- Click **Choose File**.
- Select the zip file created previously: `FMU_ - Sources.zip`.
- Click **Upload** to upload the file to the Real-Time PC.
- Click **Build** in "Build User Library".

If the Build process was successful, the source object file created is displayed in "Download Library Files".

If, however, errors have occurred, these are output in the log window. Refer to the section "Trouble Shooting" on page 77 for solutions to possible errors.

- To download the source object file onto your PC, click the relevant link in the section "Download Library File"

Note

The make file `makefile_labcar` provided is only a template in which all C files in the `FMU_Sources` folder are taken into consideration. It also contains specific flags for integrating several FMUs.

Furthermore, it may be necessary to add further compiler flags, definitions and include directories for the build process. It may also be necessary to exclude certain C files from compilation.

Trouble Shooting

Question: The source files do not contain the `fmiFunctions.h` file or the `fmiPlatformTypes.h` file. Where can I get them?

Response: You can download these files from the JModelica homepage:

`fmiFunctions.h`:

<https://svn.jmodelica.org/trunk/ThirdParty/FMI/1.0-CS/fmiFunctions.h>

`fmiPlatformTypes.h`:

<https://svn.jmodelica.org/trunk/ThirdParty/FMI/1.0-CS/fmiPlatformTypes.h>

Question: How do I change the name of a variable?

Response: To change the name of a variable, proceed as follows:

- Change the variable name in the "Properties" window.
- Open the `<FMU_InstanceName>_FMU.c` file (in the `Target_<TargetName>/CModules/instanceName` folder).
- Search for the old variable name in the `cmoc_process_<InstanceName>FMU()` method and replace it with the new name.

Example: If the variable name is to be changed from "value" to "valueone" and the code contains the following line:

```
fmiGetReal(s_batch, inputRealValref_value, nvr_realin, &value);
```

this has to be replaced by:

```
fmiGetReal(s_batch, inputRealValref_value, nvr_realin, &valueone);
```

Note

Before the project build, make sure that the `FMU_Sources` folder (in the `LABCAR-OPERATOR` project directory) has been deleted!

Question: After the `FMU_Sources.zip` file is uploaded, the message
The makefile "makefile_labcar" will be used to build
the library

is not output and furthermore the message

Missing Arguments

is output during the build process.

Response: The cause of this may be an incorrect zip file of the source files in the `\FMU_Sources` folder. If the packed file works, its content is displayed (after the zip file has been uploaded) in the web interface in the "Build User Library" section ([Main Page](#) → [User Library](#)).

Question: The build process is interrupted with a compiler error message.

Response: The build process can be interrupted for a number of reasons. Depending on the tool with which the FMU was created, you should try one of the following solutions.

Note

We generally recommend that the make file of the creator of the FMU be used as starting point (if available) and that the make file `makefile_labcar` be adapted accordingly.

Solution 1: If the FMU was created with MapleSim[®] and the build process was interrupted due to a compiler error, there could be an `fmuTemplate.c` file among the source files. Errors can result from this file being used by other files due to Include instructions.

[Main Page >> User Library](#)

Running "make" on "makefile_labcar" to build the library:

```
make: Warning: File 'makefile_labcar' has modification time 2e+04 s in the future
gcc -fvisibility=hidden -I ./ -c ./fmuTemplate.c
./fmuTemplate.c:29:1: error: unknown type name 'fmiValueReference'
./fmuTemplate.c:29:29: error: 'NULL' undeclared here (not in a function)
./fmuTemplate.c:30:1: error: unknown type name 'fmiValueReference'
./fmuTemplate.c:37:1: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:37:33: error: unknown type name 'ModelInstance'
./fmuTemplate.c:49:1: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:49:32: error: unknown type name 'ModelInstance'
./fmuTemplate.c:63:1: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:63:31: error: unknown type name 'ModelInstance'
./fmuTemplate.c:75:1: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:75:32: error: unknown type name 'ModelInstance'
./fmuTemplate.c:76:2: error: unknown type name 'fmiValueReference'
./fmuTemplate.c:91:1: error: unknown type name 'fmiStatus'
./fmuTemplate.c:91:21: error: unknown type name 'fmiComponent'
./fmuTemplate.c:91:40: error: unknown type name 'fmiValueReference'
./fmuTemplate.c:91:62: error: unknown type name 'fmiString'
./fmuTemplate.c:97:1: error: unknown type name 'fmiComponent'
./fmuTemplate.c:97:51: error: unknown type name 'fmiString'
./fmuTemplate.c:98:2: error: unknown type name 'fmiString'
./fmuTemplate.c:98:18: error: unknown type name 'fmiCallbackFunctions'
./fmuTemplate.c:98:50: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:174:1: error: unknown type name 'fmiStatus'
./fmuTemplate.c:174:47: error: unknown type name 'fmiComponent'
./fmuTemplate.c:175:2: error: unknown type name 'fmiBoolean'
./fmuTemplate.c:175:34: error: unknown type name 'fmiReal'
./fmuTemplate.c:176:2: error: unknown type name 'fmiEventInfo'
./fmuTemplate.c:201:1: error: unknown type name 'fmiStatus'
./fmuTemplate.c:201:46: error: unknown type name 'fmiComponent'
./fmuTemplate.c:216:32: error: unknown type name 'fmiComponent'
./fmuTemplate.c: In function 'fmiGetVersion':
./fmuTemplate.c:243:38: error: 'fmiVersion' undeclared (first use in this function)
./fmuTemplate.c:243:38: note: each undeclared identifier is reported only once for each function it appears in
./fmuTemplate.c: At top level:
```

- [Change the make file by removing the line](#)

```

./fmuTemplate.o: fmuTemplate.c
    $(CC) $(CFLAGS) -c ./fmuTemplate.c
    and
    • remove
./ fmuTemplate.o \
    from the OBJECTS variables in the make file.
    If the make file is as follows:

CFLAGS = -fPIC -fvisibility=hidden $(INCLUDES)
OBJECTS = ./fmuTemplate.o\
./MsimModel.o\
./SeriesHEVFCandSOC.o\
serieshevfcandsoc.so : $(OBJECTS)
    $(CC) $(CFLAGS) -shared -o serieshevfcandsoc.so
$(OBJECTS) -lm
./fmuTemplate.o: fmuTemplate.c
    $(CC) $(CFLAGS) -c ./fmuTemplate.c
./MsimModel.o : MsimModel.c
    $(CC) $(CFLAGS) -c ./MsimModel.c
./SeriesHEVFCandSOC.o : SeriesHEVFCandSOC.c
    $(CC) $(CFLAGS) -c ./SeriesHEVFCandSOC.c
clean :
    $(RM) $(OBJECTS)
    • change it to

CFLAGS = -fPIC -fvisibility=hidden $(INCLUDES)
OBJECTS = ./MsimModel.o\
./SeriesHEVFCandSOC.o\

serieshevfcandsoc.so : $(OBJECTS)
    $(CC) $(CFLAGS) -shared -o serieshevfcandsoc.so
$(OBJECTS) -lm
./MsimModel.o : MsimModel.c
    $(CC) $(CFLAGS) -c ./MsimModel.c
./SeriesHEVFCandSOC.o : SeriesHEVFCandSOC.c
    $(CC) $(CFLAGS) -c ./SeriesHEVFCandSOC.c
clean :
    $(RM) $(OBJECTS)

```

If the error still persists, header files may be missing. These can be obtained from the creator of the FMU.

Solution 2: If the FMU was created with SimulationX® and error messages such as the ones below are being displayed, it could be that definitions are missing from the make file.

[Main Page >> User Library](#)

Running "make" on "makefile_labcar" to build the library:

```
make: Warning: File `makefile_labcar' has modification time 2e+04 s in the future
gcc -fvisibility=hidden -I ./sundials/include/ -I sundials/src/ -c ./ITI_ArrayFunctions.c
gcc -fvisibility=hidden -I ./sundials/include/ -I sundials/src/ -c ./ITI_big_uint.c
gcc -fvisibility=hidden -I ./sundials/include/ -I sundials/src/ -c ./ITI_Cvode_base.c
In file included from ./iti_cvode_sparse.h:9:0,
                 from ./iti_cvode_helpers.h:5,
                 from ./ITI_Cvode_base.h:14,
                 from ./ITI_Cvode_base.c:19:
./ma_sparse.h: In function '__declspec':
./ma_sparse.h:34:14: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:34:52: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:36:1: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:37:64: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:38:45: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:39:26: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:40:38: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:41:26: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:42:14: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:43:2: error: expected declaration specifiers before '__declspec'
./ma_sparse.h:44:31: error: expected declaration specifiers before '__declspec'
```

For the SimulationX® tool, the missing definition

```
-DITI_CVODE_EXT
```

in the CFLAGS variables could be the problem.

- [Add this variable:](#)

```
CFLAGS = -fPIC -fvisibility=hidden $(INCLUDES) -
DITI_CVODE_EXT
```

Question: In the .so file build, the following error message is displayed:

```
fmuTemplate.c:316:25: error: unknown type name 'fmiValueReference'
fmuTemplate.c:316:71: error: unknown type name 'size_t'
fmuTemplate.c:316:83: error: unknown type name 'fmiInteger'
fmuTemplate.c:335:1: error: unknown type name 'fmiStatus'
fmuTemplate.c:335:25: error: unknown type name 'fmiComponent'
fmuTemplate.c:335:25: error: unknown type name 'fmiValueReference'
fmuTemplate.c:335:71: error: unknown type name 'size_t'
fmuTemplate.c:335:83: error: unknown type name 'fmiBoolean'
fmuTemplate.c:354:1: error: unknown type name 'fmiStatus'
fmuTemplate.c:354:24: error: unknown type name 'fmiComponent'
fmuTemplate.c:354:24: error: unknown type name 'fmiValueReference'
fmuTemplate.c:354:70: error: unknown type name 'size_t'
fmuTemplate.c:354:82: error: unknown type name 'fmiString'
fmuTemplate.c: In function 'fmiGetModelTypesPlatform':
fmuTemplate.c:591:12: error: 'fmiModelTypesPlatform' undeclared (first use in this function)
fmuTemplate.c: At top level:
fmuTemplate.c:594:1: error: unknown type name 'fmiComponent'
```


Response: The reason for this error message is that the source code does not contain the macro `#define FMI_COSIMULATION`.

Open the main source file (usually the C file with the name of the FMU) and add the macro at the start of the file as shown below:

```
/* define model size */
#define NUMBER_OF_REALS 215
#define NUMBER_OF_INTEGERS 0
#define NUMBER_OF_BOOLEANS 0
#define NUMBER_OF_STRINGS 0
#define NUMBER_OF_STATES NDIFF
#define NUMBER_OF_EVENT_INDICATORS 0
#define FMI_COSIMULATION
#define TIMESTEP 1.000000e-03
```

Question: The build process of the shared object file was successful but the build process fails when adding several FMUs to the LABCAR-OPERATOR project.

Response: This may be due to conflicts between the methods of different FMUs. To solve this problem, the source files have to be adjusted accordingly.

These adjustments are made automatically by LABCAR-IP if the `\FMU_Source` directory contains a file called `fmiFunctions.h`.

If this file does not exist, manual adjustments are necessary. For this purpose, execute the following steps:

- [Search for the header file that contains the string "fmiInitializeSlave"](#).
- [Open this file in a text editor and replace the line `#define DllExport`](#)

```
/* Export fmi functions on Windows */
#ifdef _MSC_VER
#define DllExport __declspec( dllexport )
#else
#define DllExport
#endif
```

[by the following line:](#)

```
/* Export fmi functions on Windows */
#ifdef _MSC_VER
#define DllExport __declspec( dllexport )
#else
#define DllExport __attribute__ ((visibility ("default")))
#endif
```

Notes and Limitations

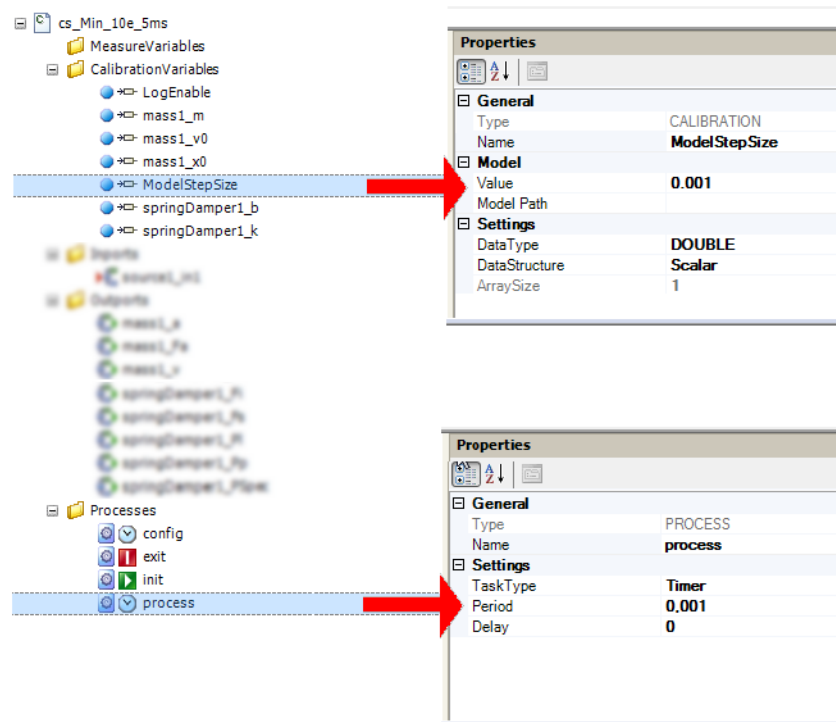
- **Naming conventions for the shared object file**

The name of the shared object file must not begin with the letter combination "lib" followed by the module name. The name must only contain lower case letters.

Example: If the project consists of two modules "gear" and "Engine", shared object files with names such as libgear.so or libEngine.so are not permissible.

- **Task duration and step size of the model**

The period of the "process" process of the module must be identical to the step size of the model (type "CALIBRATION").



- **Calibration variables**

Make sure that the values of the calibration variables are defined in the experiment environment before the experiment is run. These cannot be changed once the experiment is running.

- **Inports and Outports**

LABCAR-IP ignores inports and outports of type "string", "boolean" and "enum" as they are not supported. It also ignores array types. Only scalar types are supported for FMUs in LABCAR-IP.

- **Calibration and measurement variables**

LABCAR-IP does not support strings and enums types for calibration and measure variables.

3.5 CAN Modules (Network Integration CAN)

LABCAR-NIC V5.4.4 (Network Integration CAN) is an add-on to LABCAR-OPERATOR V5.4.4. It enables simple testing of ECU functions which include CAN communication.

The entire CAN bus communication is read in from one or more CANdb files; the user selects the CAN nodes (UuTs) physically available and LABCAR-NIC V5.4.4 automatically creates code for the residual bus to be simulated. User-defined code (e.g. counters or calculation of checksums) can be added to this.

The signals of the selected messages are available in the Connection Manager; this is where they can be connected to model inputs (receive messages) and model outputs (send messages). The signals are also available in the Signal Center with all its OLC functionality.

ETAS EE provides special GUIs for displaying messages and a CAN Monitor.

The J1939 network protocol has been supported since Version 5.0 of LABCAR-NIC (see "The J1939 Extension" on page 85).

Hardware Requirements

To work with Network Integration CAN you require a Real-Time PC with ETAS RTPC V6.3.1 and (at least) one CAN board of the type "iPC-I XC16/PCI" or "CAN-IB200/PCIe" from the company IXXAT.

It is not necessary to integrate the CAN board into the RTIO (LABCAR-RTC V5.4.4) – the configuration of the CAN board(s) is described in the section "Bus" on page 99.

3.5.1 Introduction

As an example, we have selected a CAN network with four nodes (referred to as "Node A, B, C and D") and the CAN messages 1 - 5. The communication described in the CANdb file is shown in Fig. 3-6.

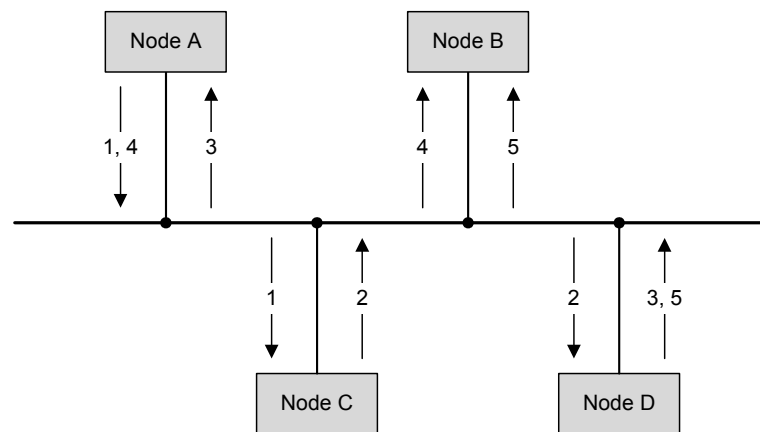


Fig. 3-6 CAN Network with Four Nodes and Five Messages

Two of the four nodes, C and D, are physically available; these are the UuTs. The "counterparts" of messages 1, 3 and 5 have to be simulated.

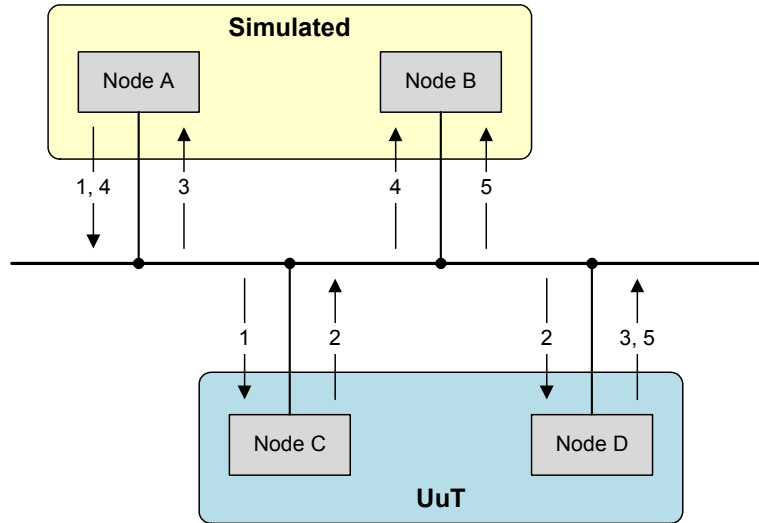


Fig. 3-7 Physically present (UuT) and Simulated Nodes

Message 4 is exchanged within the simulated environment and thus plays no role in the communication with the UuT. The remaining communication to be simulated forms what is referred to as the "residual bus" for which code must be generated.

The simulated residual bus has the send message 1 and the receive messages 3 and 5.

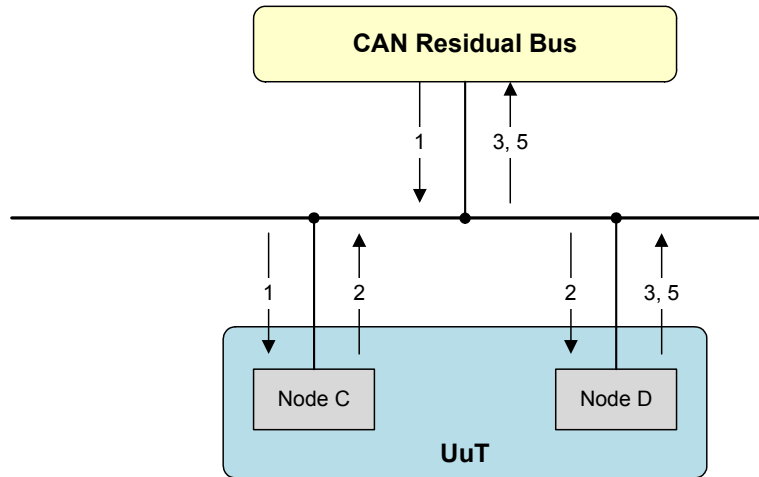


Fig. 3-8 Messages to and from the Residual Bus

3.5.2 The J1939 Extension

The network protocol J1939 is based on a range of standards of the Society of Automotive Engineers (SAE). J1939 describes communication on a CAN bus in commercial vehicles (e.g. drivetrain and chassis) and works on the physical layer with CAN highspeed in accordance with ISO11898.

The special features of J1939 are:

- 29 bit identifiers in accordance with CAN 2.0B (extended format)
- Communication both peer-to-peer and broadcast
- Transport protocols for up to 1785 bytes of data (BAM (Broadcast Announce Message) and CMDT (Connection Mode Data Transfer))
- Decentral network management
- Information is transferred in the form of parameters (signals) which in turn are compiled as parameter groups and identified by a unique number, the Parameter Group Number (PGN).

Features of the J1939 Extension in LABCAR-NIC V5.4.4

LABCAR-NIC V5.4.4 now also processes bus descriptions in accordance with the J1939 standard including:

- the handling of 29 bit message identifiers with their interpretation scheme for J1939
- network management and the corresponding transport protocols
- enabled to import bus descriptions for J1939

Note

The dbc files must be available in the format "J1939 PG (ext. ID)" that is supported by CANalyzer V6.0 and higher – the complete 29 bit identifier is only used with this format.

- configuration of parameters for bus simulation
- and*
- code generation for the IXXAT CAN boards in the Real-Time PC

Long CAN Messages

The transmission of long messages up to and including the protocol maximum 1785 byte payload is supported. These messages can also be created and edited in the CAN Editor of LABCAR-IP – their behavior in the Connection Manager is the same as with standard CAN messages.

Note

To enable J1939 decoding on a CAN controller, at least one long (> 8 byte) J1939 message needs to be configured as send, receive, or gateway message on that controller in the CAN Editor.

The transmission of CAN messages of variable length (DLC) is not supported at this time.

Multiple Session Support (J1939-21)

For reception of long messages, multiple session support is implemented – there is no artificial limit imposed on the number of simultaneous (interleaved) messages that can be received.

For transmission of long messages, each CAN interface supports the transmission of only one long message at a time. If two or more long messages are scheduled such that their transmission would overlap, they are transmitted serially. Simultaneous transmission of long messages on separate CAN interfaces is allowed.

Cycle Time

When scheduling cyclic long messages, cycle time shall be defined as the time between first message packets of long messages.

Example:

- Packet 0 of message 0 is sent at time: t_{00}
- Packet 1 of message 0 is sent at time: t_{01}
- Packet 2 of message 0 is sent at time: t_{02}
-
- Packet 0 of message 1 is sent at time: t_{10}

The cycle time is then $t_{10} - t_{00}$ (time between the first packet in each case).

3.5.3 The CAN Editor

This section contains information on working with the CAN Editor. The CAN Editor is made visible via the "CAN Editor" tab in the main window.

To open the CAN Editor

- In the Project Explorer, double-click the CAN module you want to display or edit in the editor.
The CAN module is shown in the "CAN Editor" tab of the main window.

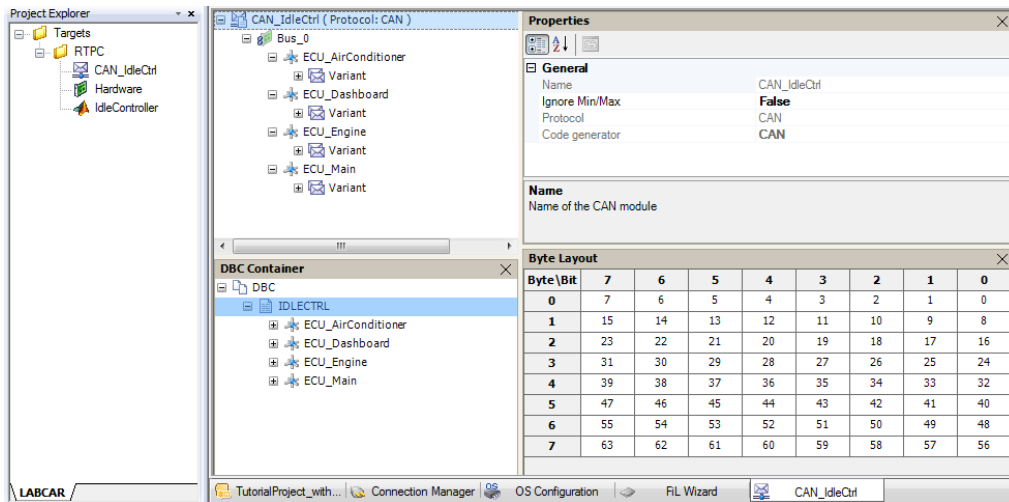


Fig. 3-9 The CAN Editor

The CAN Editor shows

- the CAN network,
 - the DBC container
 - the "Properties" window
- and
- a display for the byte layout of frames.

The CAN Network Window

This view shows the hierarchy of the network created from boards and controllers genuinely available with their subordinate elements.

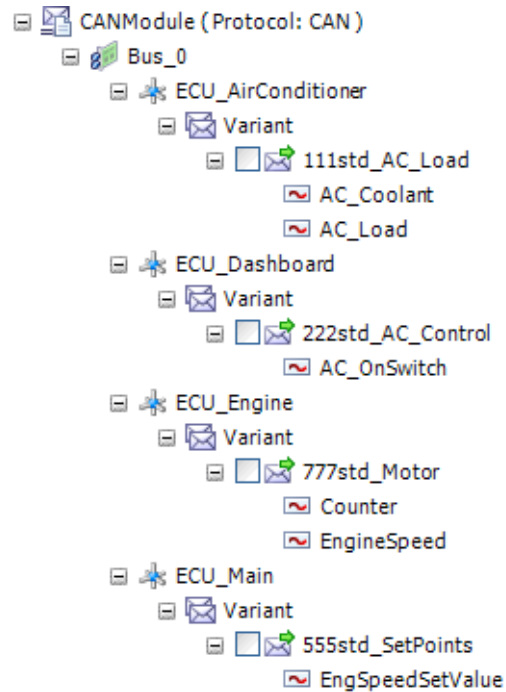


Fig. 3-10 The CAN Network

For information on network structure, refer to the section "Editing the CAN Network" on page 91; on its individual components to "The Component Parts of a CAN Network" on page 98.

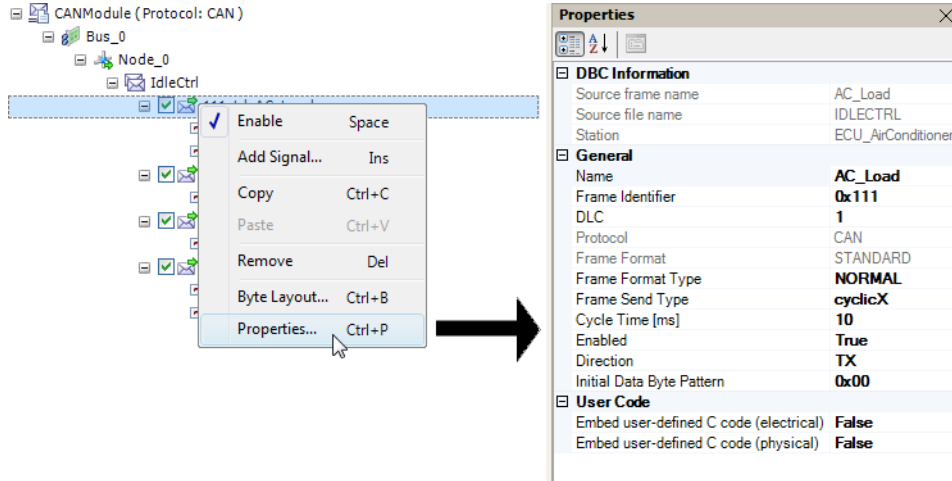
The "Properties" Window

Highlight elements in the tree views to display their properties in the "Properties" window. Editable properties can then be edited directly in the "Properties" window.

To display properties

If the properties are not displayed automatically, proceed as follows:

- Right-click the element whose properties you want to display.
- From the shortcut menu, select **Properties**.



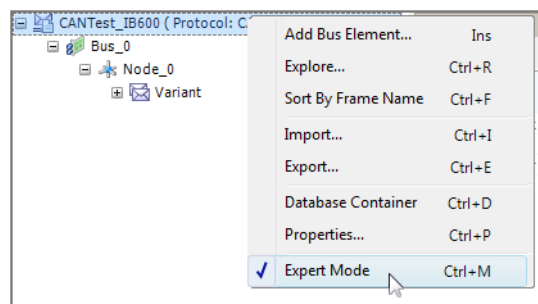
The properties of the selected element are displayed in the "Properties" window.

For more detailed information on editing properties, refer to the section "Editing the CAN Network" on page 91.

The "Expert Mode" Option

If the "Expert Mode" option is selected, rarely used properties are displayed which otherwise remain hidden.

- To enable/disable this option, right-click the root element of the CAN network and click **Expert Mode**.



These properties are described with the relevant elements of the CAN network (see "The Component Parts of a CAN Network" on page 98).

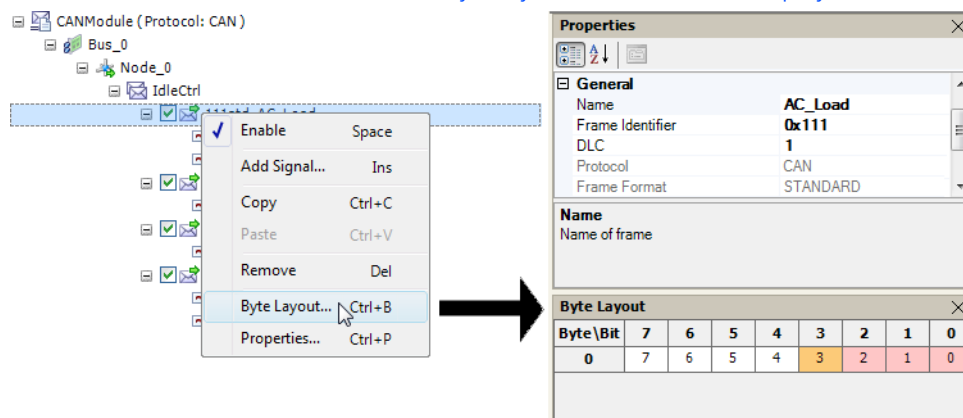
The Byte Layout of Frames

The contents of a frame selected in the network view are shown in this area.

To display the byte layout

To display the byte layout of a frame, proceed as follows:

- Right-click the frame whose layout you want to display.
- From the shortcut menu, select **Byte Layout**.
The byte layout of the frame is displayed.



3.5.4 Editing the CAN Network

The CAN network is edited exclusively via the shortcut menu of the relevant element. The elements contained in a network are described in the section "The Component Parts of a CAN Network" on page 98.

To edit the network, there are a number of items in the shortcut menu of the relevant elements of which several can be used on a multiple selection (of elements of the same kind).

To add elements

- Highlight the higher-ranking element and select **Add (Element)** from the shortcut menu.

Note

Names of parts, frames and signals can only contain characters permissible for ANSI-C identifiers.

To remove elements

- Highlight the element and select **Remove** from the shortcut menu.

To copy elements

- Highlight the source element and select **Copy** from the shortcut menu.
- Highlight the higher-ranking target element and select **Paste** from the shortcut menu.

To copy element properties

- Highlight the source element and select **Copy** from the shortcut menu.
- Highlight the desired target element and select **Paste** from the shortcut menu.

To move elements

- Highlight the source element.
- Use the mouse to drag the highlighted source element to the higher-ranking target element and then release the mouse button.

Note

The actions described above are not always permissible. In these cases, the relevant item in the shortcut menu is either disabled or not available.

Editing Properties

The properties of an element of the CAN network can be edited in the "Properties" window. Some properties of some elements cannot be edited as they are calculated automatically. These properties are grayed out in the "Properties" window.

To display the "Properties" window

- Highlight an element and select **Properties** from the shortcut menu.

The "Properties" window is displayed in the CAN Editor.

To edit properties of several elements

If several elements of the same type are highlighted in the network, the properties that all selected elements share are displayed.

- Open the "Properties" display.
- Use the mouse to select the desired elements while keeping the CTRL key pressed down.

Note

If a property that identifies an element within its higher-ranking element (e.g. a signal name within a frame) is modified when editing several elements, an error message may be issued if the property of another element of the same higher-ranking element is also given the same value.

Validation

Before the CAN module is saved, the CAN network is validated. This includes for example the verification of aspects of the CAN specification that are relevant for code generation. Error messages are displayed in the log window.

Note

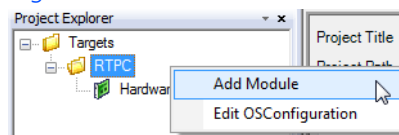
Ignoring error messages can lead to compiler errors or undesired runtime behavior during the build process of the project.

3.5.5 Creating a CAN Network

This section describes how you can create a CAN network by importing a CANdb file. Use the module wizard to add a new CAN module.

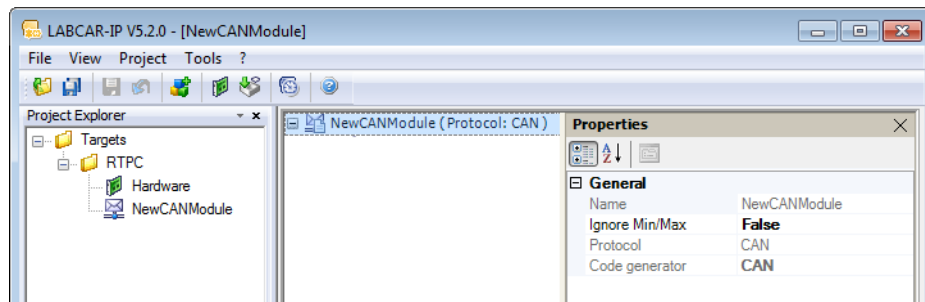
To add a CAN module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



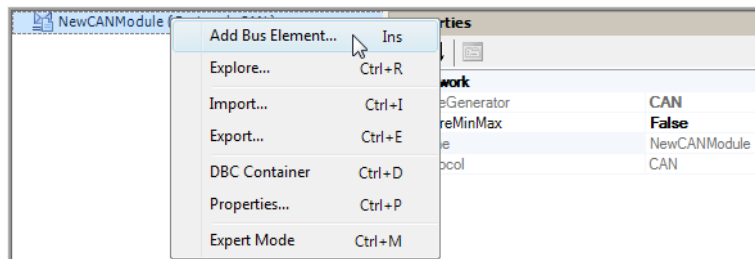
- In the "Add Module Wizard", select **Add CAN Module**.
- Click **Next**.
- Enter a module name and select a protocol.
- Click **Finish**.

In the "Project Explorer", a new module is added.

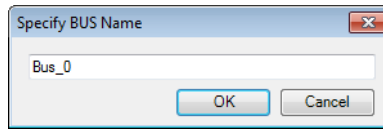


To add a bus

- Select the CAN module and right-click it.
- Select **Add Bus Element**.



- Enter a name for the bus element and click **OK**.



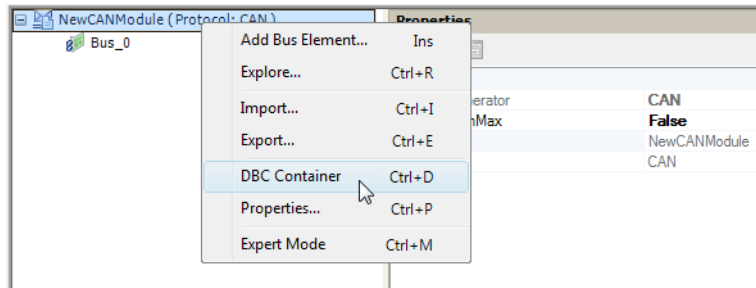
The bus element is created.

- Select **File** → **Save All**.

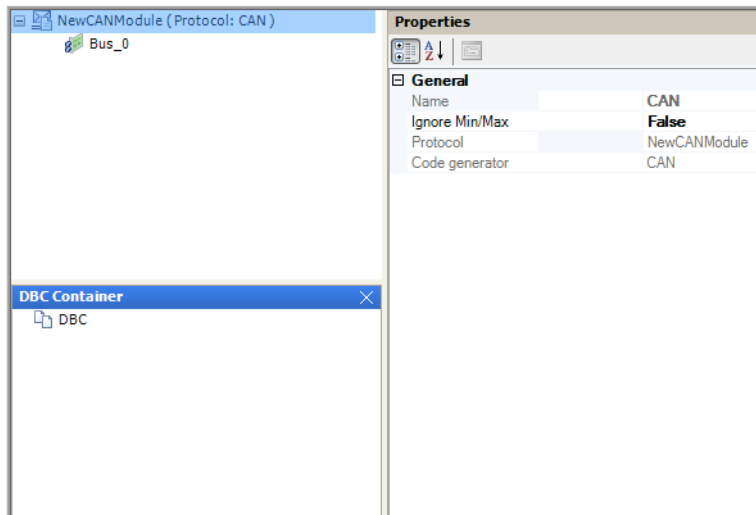
You can now build your CAN module manually (see "Editing the CAN Network" on page 91 and "The Component Parts of a CAN Network" on page 98) or import a DBC file.

To open the DBC Container

- Select the CAN module.
- Select **DBC Container** from the shortcut menu.

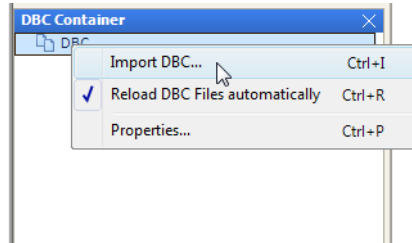


The "DBC Container" space opens.



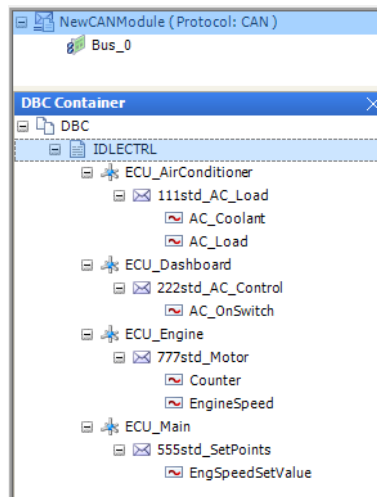
To load a DBC file

- Select the "DBC" element in the DBC Container.
- Select **Import DBC** from the shortcut menu.



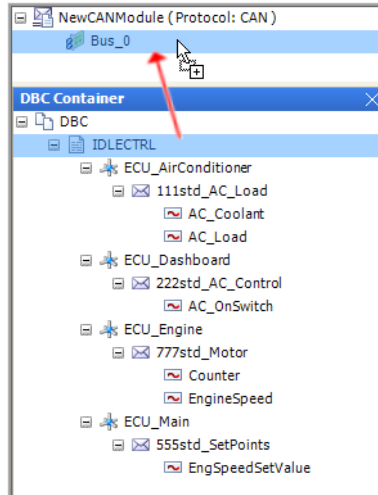
- Select the DBC file from the file selection window and click **OK**.

The information contained in the file is imported into the DBC Container.



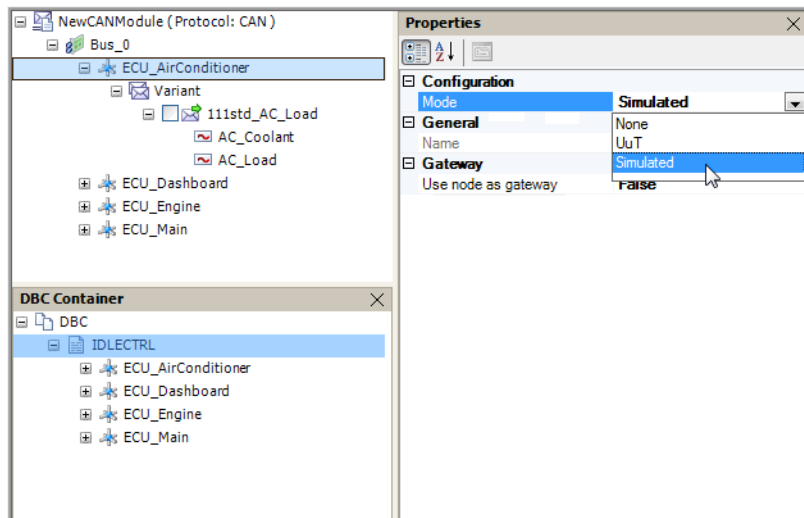
To add the DBC to the bus

- Select the imported DBC and move it to the bus element keeping the left-hand mouse button pressed down.



The nodes of the DBC file are added to the bus. Additionally, a part ("Variant") is added between each node and frame.

- Select the individual nodes and, in the Properties window (Configuration/Mode), define whether the node really exists ("UuT") or is to be simulated ("Simulated").



Drag & Drop in CAN Configuration

When elements (nodes, frames or signals) already exist in a CAN configuration, there are two different possibilities:

1. Versioning

Existing elements are accepted in the configuration with an appended version number.

– **DBC File** → **Bus**

If a node of the same name already exists

– **Node** → **Bus**

If a node of the same name already exists

– **Frame** → **Part**

If a frame of the same name already exists

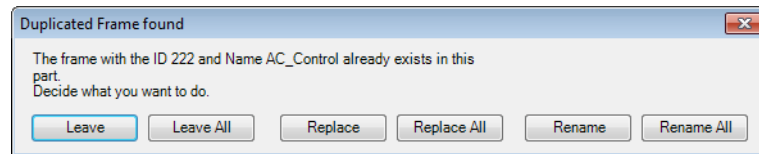
– **Signals** → **Frames**

If a signal of the same name already exists

2. Querying

– **Node** → **Node/Part**

Frames of the original node which do not exist in the target node/part, are copied there – if identical¹ frames are available, there is a query for every frame:



Choose from the following procedures:

- **Leave**

The relevant frame is not imported (with **Leave All**, no other existing frames are imported).

- **Replace**

The existing frame is replaced by the one to be imported (with **Replace All**, all other existing frames are replaced).

- **Rename**

The existing message is assigned a version number and imported (with **Rename All**, all other existing frames are assigned version numbers and imported).

¹. Frames are identical if ID/PGN **and** names are identical

3.5.6 The Component Parts of a CAN Network

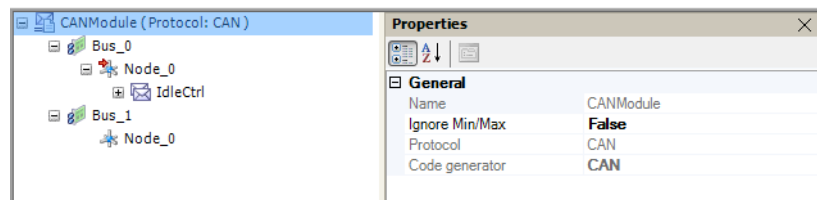
This section contains a description of the component parts of the CAN network in LABCAR-IP.

These are:

- "CAN Network" on page 98
- "Bus" on page 99
- "Nodes" on page 104
- "Parts" on page 107
- "Frames" on page 109
- "Signals" on page 116

CAN Network

The global properties of the CAN network are displayed when it is selected in the "Properties" window.



- **Name**
Name of the CAN module
- **Ignore Min/Max**
This makes it possible to ignore predefined limitations of signal values (see "**Min, Max**" on page 118).
- **Protocol**
CAN or J1939 (with network management)
- **Code generator**
The code generator to be used

Bus

CANModule (Protocol: CAN)	
<ul style="list-style-type: none"> Bus_0 <ul style="list-style-type: none"> ECU_AirConditioner ECU_Dashboard ECU_Engine ECU_Main 	
Properties	
<ul style="list-style-type: none"> Base Rate <ul style="list-style-type: none"> Receive Process Period [ms] 5 Base Rate Selection Mode AUTOMATIC Base Rate Automatic Value [ms] 1000 Base Rate Manual Value [ms] 1000 Base Rate [ms] 1000 Spread Send Frames False BTR <ul style="list-style-type: none"> Manually set Bit Timing Registers (BTR) False Bit Timing Register (BTR) 0 0x00 Bit Timing Register (BTR) 1 0x1C General <ul style="list-style-type: none"> Name Bus_0 Hardware <ul style="list-style-type: none"> Type IXXAT_XC16_CAN Board ID 0 Controller ID 0 High-speed Mode True Baud Rate [kBaud] 500 Number of unspecified frames 10 Use for XCP False Mask <ul style="list-style-type: none"> Use Mask False Standard Identifier Mask 0x00 Standard Identifier Code 0x00 Extended Identifier Mask 0x00 Extended Identifier Code 0x00 STT <ul style="list-style-type: none"> Use Single Transmission Try (STT) False Check Interval 0.5 Initial Check 4 	
Name Name of bus	

Base Rate:

The "Base Rate" property of a controller defines the rate at which communication between the LABCAR-OPERATOR project and the selected CAN module controller takes place.

All CAN send frames are calculated in the fastest task t (e.g. 1 ms). The current transfer task of a frame (cycle time) is triggered when there are whole-number multiples of this "basic task" ($n \cdot t$).

Let's say there are, e.g., three send frames with a cycle time respectively of 5 ms, 10 ms and 2 ms. The basic task must therefore have a base rate in which all these frames can be displayed, i.e. the largest common divisor of these cycle times (here: 1 ms).

If, for example, the cycle time of a frame is to be changed during a running experiment so that it is not compatible to the base rate calculated automatically, this can also be adapted manually.

With automatic calculation ("AUTOMATIC") the least common denominator of all cycle times is assumed as period - the base rate. This automatically calculated rate is divided again by 2 or 4 respectively with the options "AUTOMATIC2x"/"AUTOMATIC4x". This particularly helps in spreading the sending times with the option "Spread Send Frames".

- **Receive Process Period [ms]**

The rate with which the Receive process of the bus is cyclically called.

Only visible if the **Expert Mode** option (see "The "Expert Mode" Option" on page 89) was enabled.

- **Base Rate Selection Mode**

"Base Rate Selection Mode" specifies how "Base Rate" is to be determined. "Base Rate" can be calculated

- manually ("MANUAL")

or

- automatically ("AUTOMATIC")

In automatic calculation, a double or quadruple overclocking can be selected.

- **Base Rate Automatic Value [ms]**

"Base Rate Automatic Value" is calculated from the greatest common divisor of all values of the cycle times of all activated frames under the selected controller.

- **Base Rate Manual Value [ms]**

"Base Rate Manual Value" is the desired rate entered manually by the user.

- **Base Rate [ms]**

The current base rate

- **Spread Send Frames**

As the send buffer of the CAN card can only store a limited number of frames, send frames may be lost if the cycle times selected result in frames to be sent being collected at certain times.

If, for example, 5 frames are to be sent in a 10 ms cycle, 5 in a 20 ms cycle and 5 in a 100 ms cycle, this could lead to a pile-up at $n \cdot 20$ ms ($n = 0, 1, 2, \dots$) and particularly at $n \cdot 100$ ms (see Fig. 3-11).

The "Spread Send Frames" option prevents such pile-ups (and thus possible data loss) by sending individual frames (when being sent for the first time) with a slight delay in comparison to other frames.

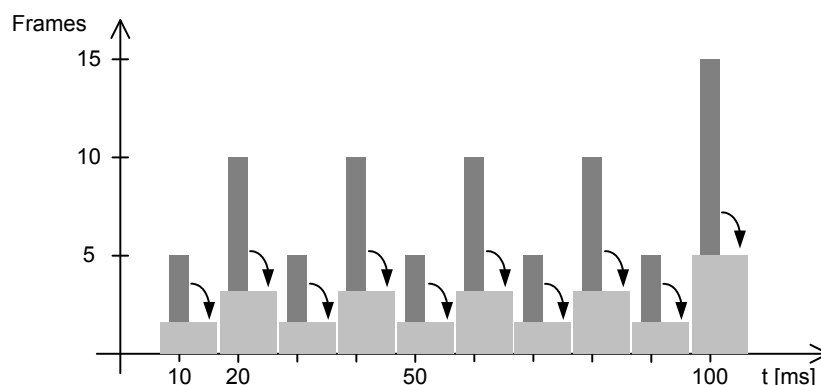


Fig. 3-11 The Effect of the Option "Spread Send Frames" (see Text)

BTR:

- **Manually set Bit Timing Registers (BTR)**

By selecting this option, you can directly access the Bit Timing Registers (BTRs) of the controller. The selection "Baud Rate [kBaud]" is then disabled.

- **Bit Timing Register (BTR) 0**

Bit Timing Register 0

- **Bit Timing Register (BTR) 1**

Bit Timing Register 1

Note

For detailed information of what is in these registers, please refer to the CAN controller data sheet.

CAN-FD - Arbitration Bit Rate:

- **CAN-FD - Arbitration Bit Rate [kBaud]**

The arbitration bit rate of the controller in kBaud (default: 500)

- **Arbitration Timing Mode**

The timing mode of the arbitration bit rate

- **Time Segment 1**

Duration of time segment 1 (in TIME QUANTA)

- **Time Segment 2**

Duration of time segment 2 (in TIME QUANTA)

- **Re-synchronisation Jump Width**

Re-synchronization jump width (in TIME QUANTA)

- **Transceiver Delay Compensation**

Transceiver delay compensation offset (in TIME QUANTA, 0 = disabled)

CAN-FD - Fast Data Bit Rate:

- **CAN-FD - Fast Data Rate [kBaud]**

The fast data rate of the controller in kBaud (default: 2000)

- **Arbitration Timing Mode**

The timing mode of the arbitration bit rate

- **Time Segment 1**

Duration of time segment 1 (in TIME QUANTA)

- **Time Segment 2**

Duration of time segment 2 (in TIME QUANTA)

- **Re-synchronisation Jump Width**

Re-synchronization jump width (in TIME QUANTA)

- **Transceiver Delay Compensation**

Transceiver delay compensation offset (in TIME QUANTA, 0 = disabled)

General:

- **Name**
Bus name

Hardware:

- **Type**
Type of CAN board used ("IXXAT_XC16_CAN", "IXXAT_IB200_CAN" or "IXXAT_IB600_CANFD")
- **Board ID / Controller ID**
Unique identifier of the board and the controllers.
- **High-speed Mode**
This option defines the CAN controller as a high-speed interface.
- **Baud Rate [kBaud]**
This is where you can define the transfer rate of the CAN controller.
- **Number of unspecified frames**
The number of frames not already specified in the experiment that can be received. The corresponding measure values are created for these in LAB-CAR-EE in the "Workspace Elements" tab.
Only visible if the **Expert Mode** option (see "The "Expert Mode" Option" on page 89) was enabled.
- **Use for XCP**
If you want to exclude a controller from the CAN residual bus simulation (e.g. to be used with XCP), select "TRUE".
- **ISO Mode**
If the hardware supports CAN FD ("IXXAT_IB600_CANFD"), you can choose whether the ISO certified protocol ("true") or the original Bosch protocol ("false") shall be applied.

Mask:

- **Use Mask**
This is where you can define whether frames are to be masked or not. If this option is enabled, masks and code can be specified both for "Standard" and for "Extended" identifiers.

Note

For detailed information of the content and on processing the Acceptance Code Registers and the Acceptance Mask Registers, please refer to the CAN controller data sheet.

STT:

This is where you can configure different parameters for error handling.

- **Use Single Transmission Try (STT)**

Enables/disables the operating mode "Single Transmission Mode" of the controller. In this operating mode, a send frame is not put into the queue after a transmission error (missing confirmation).

This operating mode is useful if the controller is not connected to a complete CAN network (if, for example, a connected ECU was temporarily without power).

Usually the controller puts all frames into a queue until they are sent correctly (including confirmation). If an ECU that was temporarily without power is given power again, this results in a sudden pile-up of CAN frames as all 50 frames in the controller's send buffer are sent in quick succession which can lead to a loss of timing between these frames.

- **Check Interval**

This determines a time interval (as a floating point number in seconds) which determines at what intervals the status of the CAN bus is periodically checked and – if the bus is in an incorrect state – rebooted.

If this interval is set to 0, the check is disabled.

- **Initial Check**

Specifies a number of intervals used to determine a failed initialization – the minimum is 2.

The default setting is 4, whereby the check is carried out in the frequency defined by "Check Interval". In this case, an error is signaled when "Initialization" is still the valid state after 4 checks.

Further Properties for the J1939 Protocol

Only visible if the **Expert Mode** option (see "The "Expert Mode" Option" on page 89) was enabled.

DTC:

- **Number of Diagnostic Trouble Codes (DTCs)**

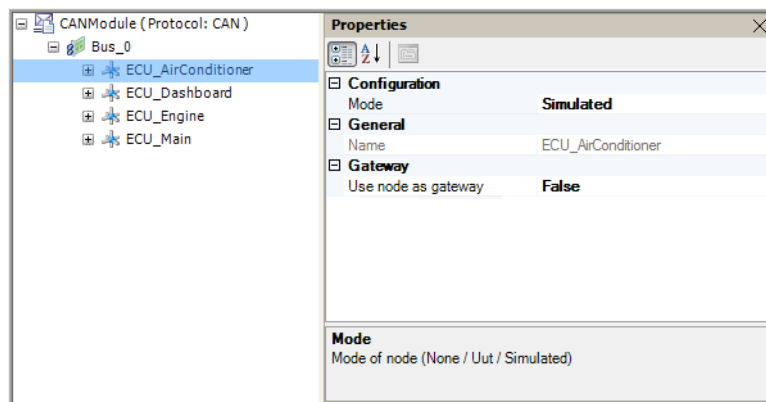
A J1939 node can also handle DTCs - this specifies the number of DTCs which can be buffered. These buffers are only available for dynamically added nodes (see "**Number of unspecified nodes**" on page 103):

As this requires a lot of memory, the maximum number is 50.

- **Number of unspecified nodes**

The maximum number of dynamically added nodes which can register by address claiming in the CAN network.

Nodes



Configuration:

- **Mode**

Configuration of the node (see Fig. 3-7 on page 84):

- **Simulated**

The node is simulated.

- **UuT**

The node is real.

- **None**

None means that the node acts neutrally: It is either controlled entirely in accordance with user definitions or is remotely controlled by the configuration of the other nodes.

General:

- **Name**

Name of the node

Gateway:

- **Use node as gateway**

This is used to configure the node as a gateway.

Further Properties for the J1939 Protocol

Properties	
<input type="checkbox"/> Configuration Mode None	
<input type="checkbox"/> Diagnostics Use as diagnostic node False Number of clients 10 Command data size 128 Response data size 512	
<input type="checkbox"/> DTC Number of Diagnostic Trouble Codes 10	
<input type="checkbox"/> Gateway Use node as gateway False	
<input type="checkbox"/> General Name Node_0	
<input type="checkbox"/> Network management Do Address Claiming False Timeout after power-on [ms] 150 Power-on threshold 0 Address Claim Contention Timeout [ms] 250 Source Address 0x00	
<input type="checkbox"/> Network management - Address Claiming NAME Identity Number 0 Manufacturer Code 0 ECU Instance 0 Function Instance 0 Function 0 Reserved 0 Vehicle System 0 Vehicle System Instance 0 Industry Group 0 Arbitrary Address Capable 0	
Name Name of node	

Diagnostics:

- **Use as diagnostic node**
 Configuration as "Diagnostic Node"

Note

*There can only be **one** "Diagnostic Node" per bus!*

The following three options are only active if "Use as diagnostic node" was selected.

- **Number of clients**
 The number of clients in ETAS EE and LABCAR-AUTOMATION with diagnostic functionality. Two are always reserved for LABCAR-AUTOMATION.
 Only visible when **Expert Mode** (see "The "Expert Mode" Option" on page 89) was enabled.

- **Command data size**

Diagnostic commands are always transmitted as an array. This value specifies the size of the array and thus the number of commands. As commands can contain different quantities of data, the number of possible commands also varies.

The size of the array should only be increased if more commands are genuinely required at the same time.

Only visible when **Expert Mode** (see "The "Expert Mode" Option" on page 89) was enabled.

- **Response data size**

Diagnostic responses are always transmitted as an array. This value specifies the size of the array and thus the quantity of the response data that can be saved.

As the quantity of the required response data depends on the relevant commands, the size of the array should only be increased if not all response data for the relevant commands can be received.

Only visible when **Expert Mode** (see "The "Expert Mode" Option" on page 89) was enabled.

DTC:

- **Number of Diagnostic Trouble Codes**

This specifies the number of DTCs which can be buffered for this node.

Only visible when **Expert Mode** (see "The "Expert Mode" Option" on page 89) was enabled.

Network management:

- **Do Address Claiming**

Enabling/disabling of the "Address Claiming"¹

- **Timeout after power-on [ms]**

The time (in ms) the simulated node has to wait before sending the address claim. Is used to simulate the Power On Self Test (POST).

- **Power On Threshold**

Simulated nodes have an input/inport that is compared with this threshold value. As soon as the threshold is exceeded, address claiming is run.

- **Address Claim Contention Timeout [ms]**

The time (in ms) the node waits after sending its address claim message for a "claim contention" from other nodes. If, during this time, no other node claims the same address, the simulated node assumes that it has been assigned the desired address and proceeds to use it.

- **Source Address**

The source address

¹. Currently only the address claiming process described in the document "SAE J1939-81, Appendix D, Figure D2 - State Transition Diagram for Initialization of Single Address CAs" is supported.

Network management - Address Claiming:

- **NAME**
The identifier of the node (ECU).
"NAME" consists of the following ten fields:
- **Identity Number**
21 bits (defined by the ECU manufacturer)
- **Manufacturer Code**
11 bits identifying the ECU manufacturer
- **ECU Instance**
3 bits for identifying an ECU instance (when, for example, several ABS ECUs exist)
- **Function Instance**
5 bits for identifying the function instance (e.g. ABS #1)
- **Function**
8 bits for identifying the function (e.g. ABS)
- **Reserved**
Reserved for future definition by the SAE
- **Vehicle System**
7 bits for identifying the vehicle system (e.g. trailer)
- **Vehicle System Instance**
4 bits for identifying the instance of a vehicle system
- **Industry Group**
3 bits for defining the particular industry group (e.g. "On-Highway", "Agricultural")
- **Arbitrary Address Capable**
1 bit for specifying whether the ECU – after being turned down in an address claim procedure – can select a different address.

Parts

A part is assigned to a higher-ranking node; frames are arranged under a part. Parts can be imported and exported. Within the module the content of a part is stored in files with the extension `.llp`.

Parts have a unique name within the entire network. If a part with a given name already exists, the name of the part to be added is extended with a consecutive number.

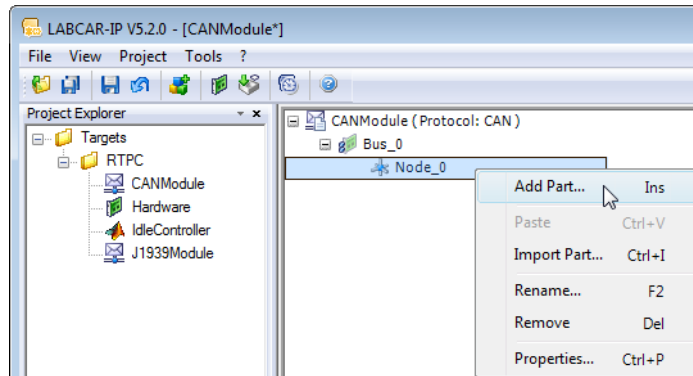
Prioritization, Enabling and Disabling of Parts

Parts are used to model variants of a network node. For this purpose, copies of a part can be created including all frames and signals defined beneath this part.

Parts can be enabled and disabled using "Enabled". The processing of the frames and signals of this part can be controlled using "Priority": Parts with a higher priority overwrite the inputs and outputs of the parts with a lower priority in every simulation cycle.

To create a part

- Right-click the node to which the new part is to be assigned.
- Select **Add Part**.

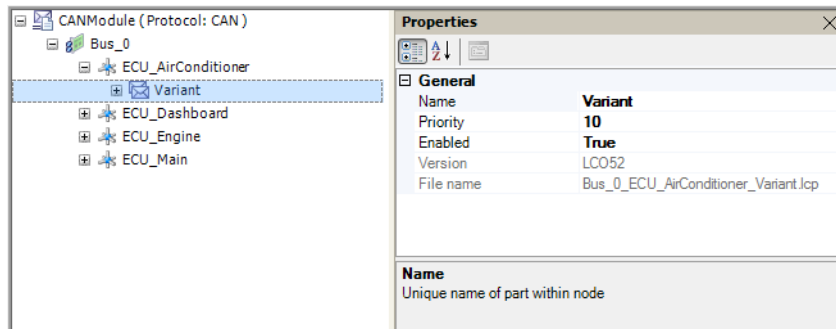


A part named "Variant" is created.

Note

Names of parts, frames and signals can only contain characters permissible for ANSI-C identifiers.

Parts previously exported can be reimported (see "To import a part" on page 109).



General:

- **Name**
Name of the part
- **Priority**
With this property, you can specify the priority of the part. The priority must have a unique value – parts are run according to their priority.
- **Enabled**
Every part can be enabled or disabled (regardless of the other parts) during experiment runtime. If a part is disabled, communication of all frames under the selected part is prevented.

- **Version**
Part version
- **File name**
Name of the file (Bus_name_Node_name_Part_name.lcp) describing the part

Note

Variant handling during experiment runtime is possible with the properties "Enabled" and "Priority".

To export a part

Parts can be exported with all frames and their properties in the form of XML files.

- Select the part to be exported.
- Right-click and select **Export Part**.
- In the file selection window, select a directory and enter a file name.
- Click **OK**.
The part information is exported to the file (*.lcp).

To remove parts

- To remove a part, right-click it and select **Remove**.

To import a part

Parts exported previously (see above) can be reimported into the CAN configuration.



- Select the node to which the part is to be assigned.
- Select **Import Part** from the shortcut menu.
- Select the part to be imported (*.lcp) in the file selection window and click **OK**.
The part is created with the information from the specified file.


Frames

A frame is assigned to a higher-ranking part; signals are arranged under a frame.

In addition to the "Properties" window, the "Byte Layout" window can also be displayed for frames. This shows the byte allocation in the payload of the frame with the signals.

The frames are shown with different icons depending on the direction:

-  – Send frame (TX)
-  – Receive frame (RX)

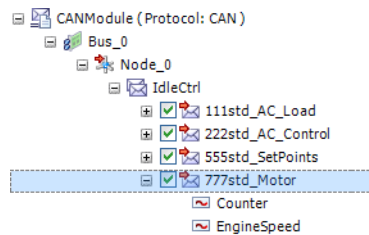
-  – Gateway frame (GW)
 (see "Use node as gateway" on page 104)

Note

Please note that the direction is specified from the point of view of the simulated residual bus (see Fig. 3-8 on page 84)!

To display the byte layout

- Highlight the frame and select **Byte Layout** from the shortcut menu.



Byte \ Bit	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32

In the example, the frame consists of the 16-bit signal "EngineSpeed" (light red) and the 5-bit signal "Counter" (ochre).

- If you double-click on one of the "bit fields", the signal is selected in the frame list and its properties displayed in the "Properties of Signal" field.

Note

If errors occur in the byte layout, e.g. by two signals overlapping or the incorrect specification of the length of a signal or frame, the incorrect bits are shown in red.

To create a user-defined frame

In addition to the possibility of importing a frame from a CANdb file, from a CAN module or part, you can also create user-defined frames.

- Highlight the higher-ranking part and select **Add Frame** from the shortcut menu.

Note

Within a part, the combination of the ID of the frame and its name must be unique!

- Enter the following information (see "General" on page 113).
 - Name of frame (Name)
 - Identifier (ID)
 - Protocol (Protocol)
The protocol was defined when the module was created (see "To add a CAN module" on page 93).
 - Size of frame (DLC)
 - Direction (Direction)
 - Frame type (Type)
- Click **OK**.
The newly created frame is added to the test matrix.

If you are using several CAN boards when configuring your CAN matrix, please read the following note:

Note

If you remove a board to which frames have been assigned, there is no automatic change of this assignment: this has to be executed manually!

To enable and disable frames

Several frames with the same ID may be defined within one part providing they each have a different name. For the purposes of code generation, however, only one frame with a specific ID can be enabled in one part. The other frames therefore have to be excluded from code generation.

- **Using the network tree view:** Enable or disable the checkbox next to the frame icon in the network tree view.
- **Using the shortcut menu:** Highlight the frame and enable/disable it in the **Enable** shortcut menu.
- **Using the "Properties" window:** Open the "Properties" window and edit the "Enabled" property ("True" or "False").

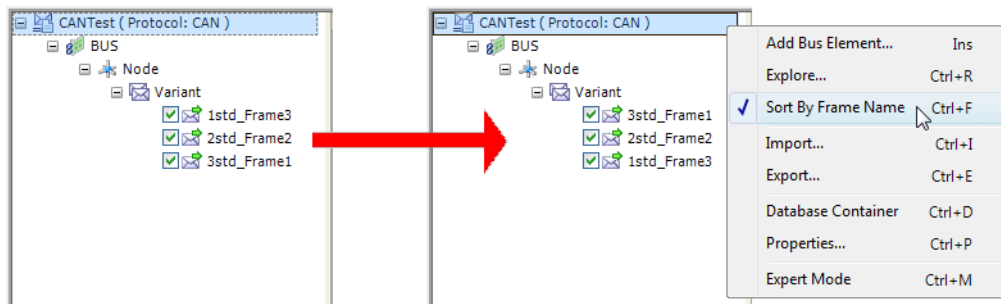
To remove a frame

- Select **Remove** from the shortcut menu of the frame list.
The frame is removed.

To sort frames by name

Frames are normally sorted by frame identifier.

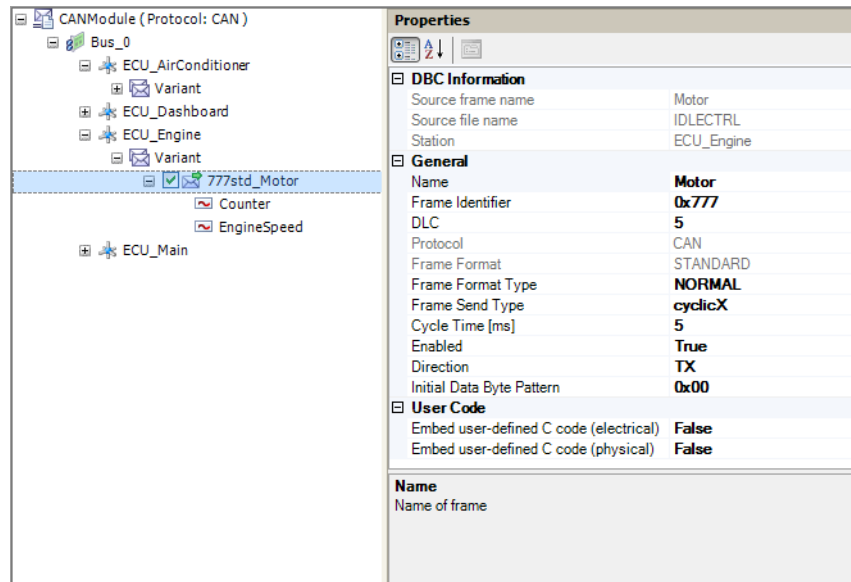
- To sort the frames by name, right-click the module name and select the option **Sort by Frame Name**.



To display properties of a frame

- To display the properties of a frame, right-click it.
- Select **Properties** from the shortcut menu.

The properties of the frame are displayed in the "Properties" window.



DBC Information:

If the frame is from an imported CANdb file, the relevant information is shown here.

General:

- **Name**
Name of the frame
- **Frame Identifier**
The frame identifier (hex)
- **DLC**
The data length code of the frame
- **Protocol**
The protocol ("CAN" or "J1939")
- **Frame Format**
Format of the frame ("STANDARD", "STANDARD_REMOTE", "EXTENDED" or "EXTENDED_REMOTE")
- **Frame Format Type**
"NORMAL" or "REMOTE"
- **Frame Send Type**
The send mode (only with send frames)
 - **cyclicX**
The frame is sent in the cycle time raster

- **spontanX**

The frame is sent if one of the signals has changed

Note

*A frame with the send type "spontanX" is only sent if at least one of the signals is in send mode "On Change" (see "**Send Type**" on page 118)!*

- **cyclicAndSpontanX**

The frame is sent in both of the above cases

- **Cycle Time [ms]**

The cycle time of the frame in ms

- **Enabled**

Activation/deactivation of the frame (see "To enable and disable frames" on page 112)

- **Direction**

The direction from the point of view of the model ("Send (TX)" or "Receive (RX)")

- **CAN-FD**

Frame is interpreted as CAN-FD frame ("True") or standard frame ("False").

- **Bit Rate Switch (BRS)**

Specifies whether (during CAN-FD frame transfer) the increased data bit rate should be used.

- **Initial Data Byte Pattern**

The byte pattern on initialization of the frame

User Code:

- **Embed user-defined C code (electrical)**
Embed user-defined C code (physical)

Here you can specify whether user-defined C code is to be added to the frame (see "User-Defined C Code" on page 120).

Further Properties for the J1939 Protocol

Frame Identifier	
ID	0x1D0A307C
Parameter Group Number (PGN)	68096
Data Page	0x01
PDU Format	0x0A
PDU Specific Field	0x30
Priority	7
Source Address	0x7C
Frame specific source address	False
PGN Type	Standard
PDU Format Type	PDU1
PDU Specific Field Functionality	Destination Address

Frame Identifier:

- **ID**

The complete CAN ID of the message (29 bits) consisting of

- Priority (3 bits)
- Parameter Group Number (18 bits)
- Source Address (3 bits)
- **Parameter Group Number**

The Parameter Group Number consisting of

 - Data Page (1 bit)
 - PDU Format (8 bits)
 - PDU Specific (8 bits)
- **Data Page**

Data Page bit
- **PDU Format**

8 bits for specifying the PDU format (part of the PGN):

 - < 240: PDU Specific Field is "Destination Address" ("PDU1")
 - >= 240: PDU Specific Field is "Group Extension" ("PDU2")
- **PDU Specific Field**

8 bits (see PDU Format)
- **Priority**

3 bits for optimizing the latency of transmission to the bus (is ignored by LABCAR-OPERATOR)
- **Source Address**

Source address of the node to which the frame belongs (8 bits).
- **Frame specific source address**

If "TRUE", the source address of the frame can be set to a value deviating from the address of the node.
- **PGN Type**

PGN type
- **PDU Format Type**

"PDU1" or "PDU2" (see PDU Format)
- **PDU Specific Field Functionality**

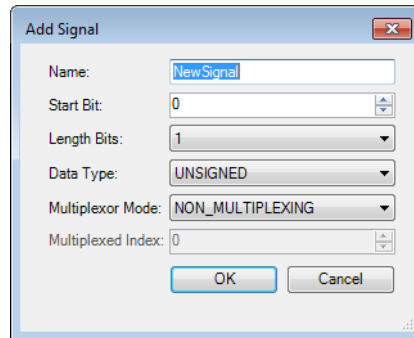
Function of the "PDU Specific" field (see PDU Format)

Signals

A signal is assigned to a higher-ranking frame: The name of a signal under this frame must be unique.

To create a user-defined signal

- Highlight a frame and select **Add Signal** from the shortcut menu.



- Enter the following information (see "Scaling" on page 118 and "General" on page 117).
 - Name
 - Start bit
 - Signal length (Length Bits)
 - Data type
 - Multiplexor mode
 - Multiplexed index
- Click **OK**.

The new signal is added to the frame.

To remove a signal

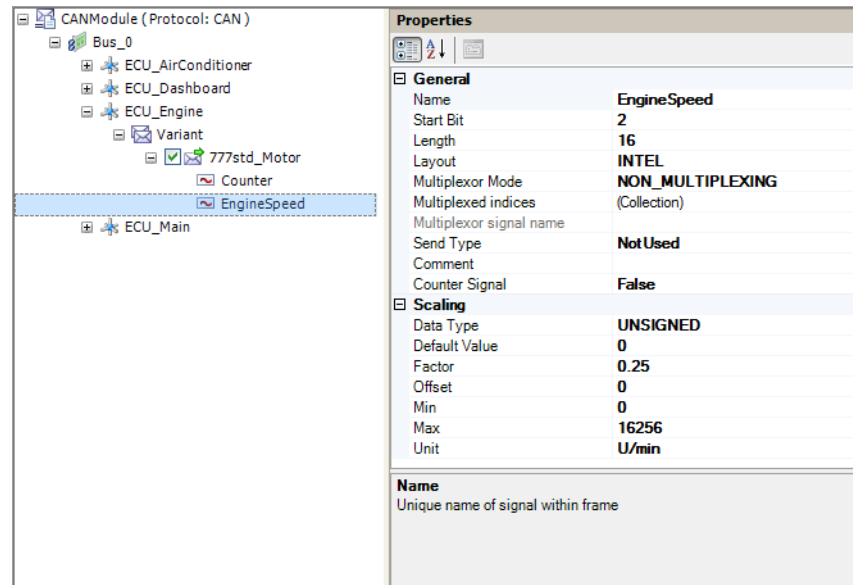
You can remove CAN signals which you have added to a frame:

- Right-click the signal and select **Remove**.

The signal is removed from the frame.

To display the properties of the signal


- To display the properties of a signal, right-click it.
- Select **Properties** from the shortcut menu.



The properties of the signal are displayed in the "Properties" window. This is where the properties of a selected signal are displayed. They can also be edited here, if editing is permissible.

General:

- **Name**
Name of the signal
- **Start Bit**
Start bit of signal
- **Length**
Signal length (in bit)
- **Layout**
Byte order: "INTEL" (little-endian) or "MOTOROLA" (big-endian)
- **Multiplexor Mode**
Characterizing frames
 - NON_MULTIPLEXING
Characterizes a standard (= non-multiplex) frame
 - MULTIPLEXOR
The signal is the control signal of a multiplex frame
 - MULTIPLEXED_SIGNAL
The meaning of the data transferred in this signal is determined by the current value of the control signal.

- **Multiplexed indices**
Coding for the assignment with multiplex frame ("MULTIPLEXED_SIGNAL")
Clicking  opens the Collection Editor with which several value assignments can be made.
- **Multiplexor signal name**
Name of the multiplexer signal that decides whether the signal is to be used or not.
- **Send Type**
Specifies how the sending of a frame is triggered:
 - NotUsed
The signal is not taken into consideration when the send trigger is calculated
 - OnChange
Signal is taken into consideration when the send trigger is calculated
- **Comment**
A comment on the description of the signal
- **Counter Signal**
See "To use a signal as counter" on page 119.

Scaling:

- **Data Type**
Type: "UNSIGNED", "SIGNED", "IEEE754_32bit" or "IEEE754_64bit"
- **Default Value**
A signal of a send frame can be assigned a default value. If the signal is connected to another signal in the Connection Manager, this value is ignored.
The advantage of the default value is in the import/export of CAN modules as the default value is then also available if the CAN module is used again in another LABCAR-OPERATOR project.
- **Factor, Offset**
A factor a and an offset b for the for scaling the signal
- **Min, Max**
This is where limits for a plausibility check are specified (see "Min/Max Values" on page 119).
If you right-click the signal and select **Limit Min/Max to bit length**, "Min/Max" is automatically calculated from "Data Type" and "Length".
- **Unit**
The physical unit of the signal

Min/Max Values

When a CAN test matrix is saved, a check always takes place to see if $\text{Min} < \text{Max}$. If not, a warning is issued unless the option "Ignore Min/Max" (see "**Ignore Min/Max**" on page 98) has been activated.

Note

Please note that there are two checks of the physical value and the value transferred by CAN does not correspond to the original value if this does not adhere to the check conditions.

These checks are:

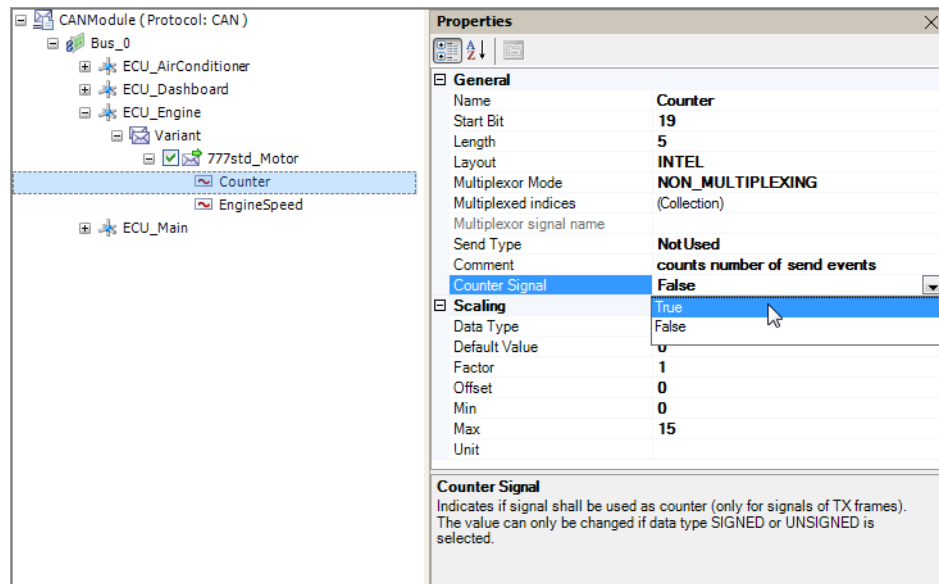
1. Is $\text{min} \leq \text{phys. value} \leq \text{max}$?
If the value does not fulfil this condition, the value is set to min or max.
2. Once the physical value has been converted into a hex value, a check is made to see whether the value corresponds to the size of the data area (taking the data type into consideration). If, for example, 6 bits have been allocated for a "signed integer", the converted value must fulfil the following condition: $-32 \leq \text{value} \leq +31$.

If this condition is not fulfilled, the value is changed to -32 or +31.

To use a signal as counter

There is a simple way of making one signal of a send frame into a counter:

- Select the signal and right-click it.
- Set the property "Counter Signal" to "True".



This signal is now used as a counter. This is indicated by the "123" symbol in the comment field.

Note

The option "Use as Counter" is only available for whole-number data types.

- You can undo this by deselecting this option.

Note

The min/max values of the signal properties are also valid for counter signals!

3.5.7 User-Defined C Code

LABCAR-NIC enables the user to add further user-defined code to the code generated automatically a message to manipulate or extend the content of the message. This is true of both the "physical" part (the signals) and of the byte array ultimately generated.

The situation is illustrated once more below to clarify the access possibilities using user-defined code.

Signal Flow of a Send Message and Possible Access

The entire signal flow of send messages from the model to the CANbus is shown as an overview in Fig. 3-12.

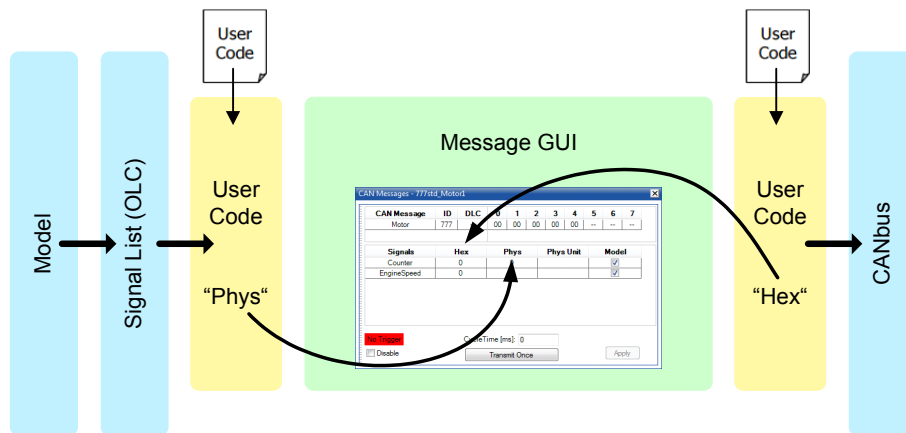


Fig. 3-12 Signal Flow with Send Messages (see Text)

In addition to manipulation possibilities offered by the Signal List, it is possible to influence the content of messages either via the message GUIs described (see "Instrumentation for the CAN Simulation" on page 125) or with user-defined C code (see the following section).

Note

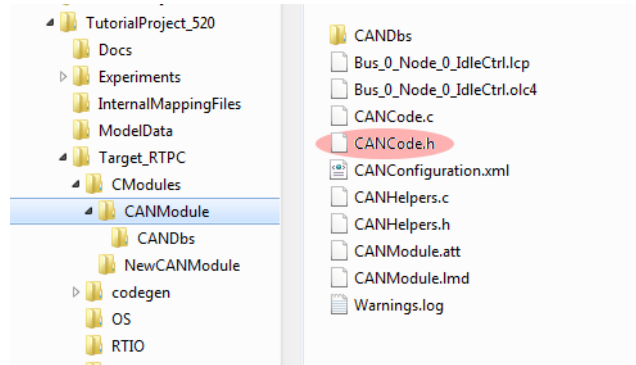
Please note that due to access further right, any code previously changed will be overwritten again!

The situation with receive messages is similar: Changes to the left will overwrite previous changes.

Below, you can find sample code and the description of how you can add this code to the message code.

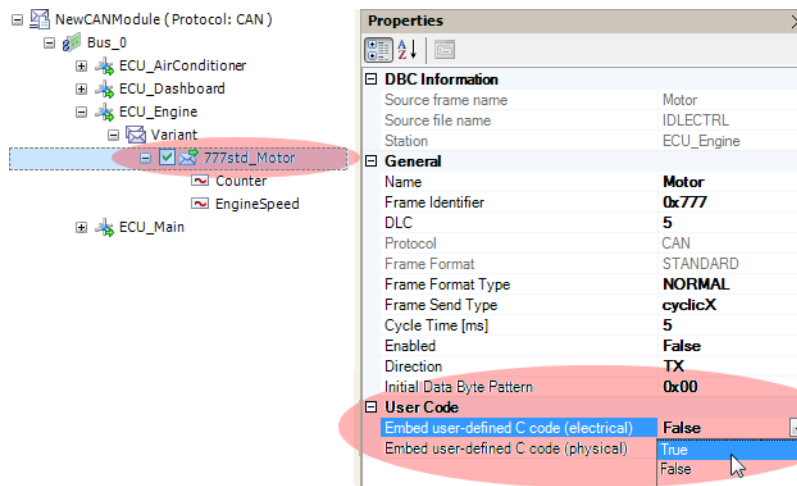
Procedure

Once the CAN configuration has been saved, the directory shown below contains the header file `CANCode.h`, the function declarations and the declaration of the structures for the signals.



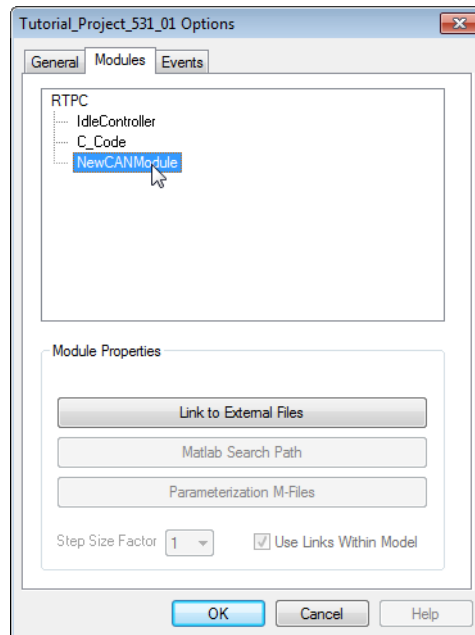
To add user-defined code

- If you want to add your own code, set "Embed user-defined C code (electrical)" or "Embed user-defined C code (physical)" to "True".



- To apply the changes, click **Save**.
- Open the `CANCode.h` file with a suitable editor. This file contains the function declarations for the frames edited above. Using these functions, you can access the physical signals as well as the byte array of the frame. `CANCode.h` also contains the structures for the relevant signals.
- Add a `#include CANCode.h` to your C files so that the function and structuredeclarations are known.

- Select **Project** → **Options** and the "Modules" tab from the main menu.

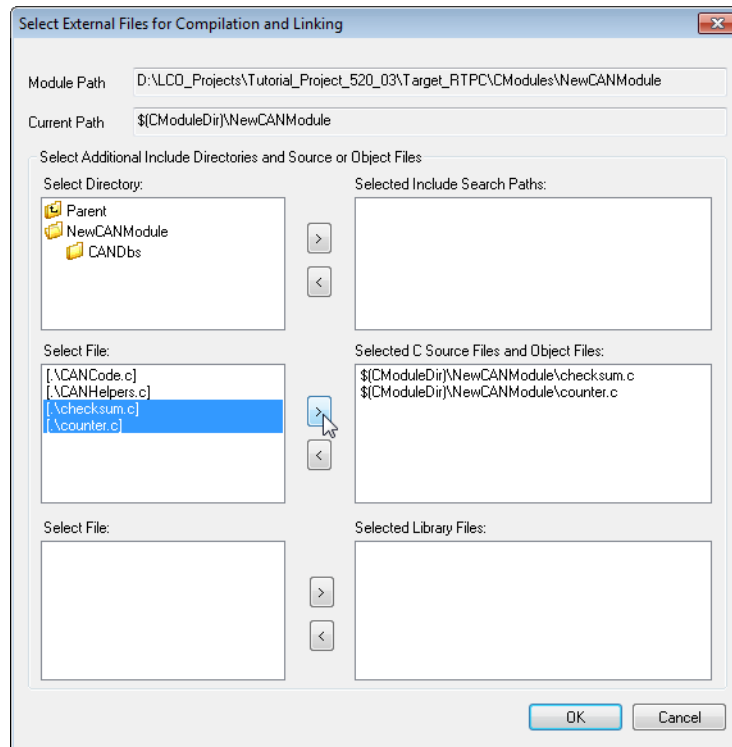


- Select the CAN module and click the **Link to External Files** button.
The "Select External Files for Compilation and Linking" window opens.
- Select the source file(s) to be linked ("Examples" on page 124) and click the > button (multiple selection is possible).

Note

You can only select C files which were stored in the project directory under \Target_RTPC\CModules\CAN_Name_of_CANdb_file or in a subdirectory.

- The file(s) selected is (are) shown in the right-hand window.



- Click **OK**.
These files are always compiled and linked in future code generations.
- Quit the "Modules" tab with **OK**.

Examples

Below, there are two examples of user-defined code:

Message Counter (counter.c):

```

/*
Example file "counter.c" for NIC training © 2001- 2007 by ETAS GmbH
-- All rights reserved --
This file defines the function "userManipulationPhysical_777std_Motor"
The function increments the CAN message signal "Counter" on every
send event of the CAN message "Motor" with the ID 777. If "counter"
scores to 15 it is reset to 0.
*/
#include "CANCode.h"

void userManipulationPhysical_NewCANModule_Bus_0_ECU_Engine_
777std_Motor ( int32_t inputTrigger, NewCANMod-
ule_Bus_0_ECU_Engine_777std_Motor_PhysValues_t* signalStructure )
{
    static int counter = 0;
    signalStructure -> Signal_Counter = counter;
    (counter <= 14) ? (counter++): (counter = 0);
}

```

Note

The "counter.c" example only works with frames sent cyclically (Frame Send Type = cyclicX, see "**Frame Send Type**" on page 113).

Calculation of a Checksum (checksum.c):

```

/*
Example file "checksum.c" for NIC training © 2001- 2007 by ETAS GmbH
-- All rights reserved --
This file defines the function "userManipulationByteArray_777std_Motor"
The function sets the last but not last byte in the byte array[0..4] of
a CAN message "Motor" (ID 777) on every send event of the message
according to a checksum algorithm.
*/
#include "CANCode.h"

void userManipulationByteArray_NewCANModule_Bus_0_ECU_Engine_ 777st-
d_Motor( uint32_t* id, uint64_t* dlc, unsigned char* byteArray )
{
    static unsigned int Checksum = 0;
    unsigned long Motor_ID = *id;
    unsigned long Motor_DLC = *dlc;
    Checksum = Motor_ID + byteArray[0] + byteArray[1]
                + byteArray[2] + byteArray[3];
    Checksum = (Checksum + (Checksum >>8)) & 0xff;
    byteArray[4] = Checksum;
}

```

3.6 Instrumentation for the CAN Simulation

If a CAN residual bus simulation is executed with LABCAR-NIC V5.4.4, there are special instruments in ETAS EE for CAN messages as well as a "CAN Bus Monitor".

3.6.1 CAN Modules in the "Workspace Elements" Window

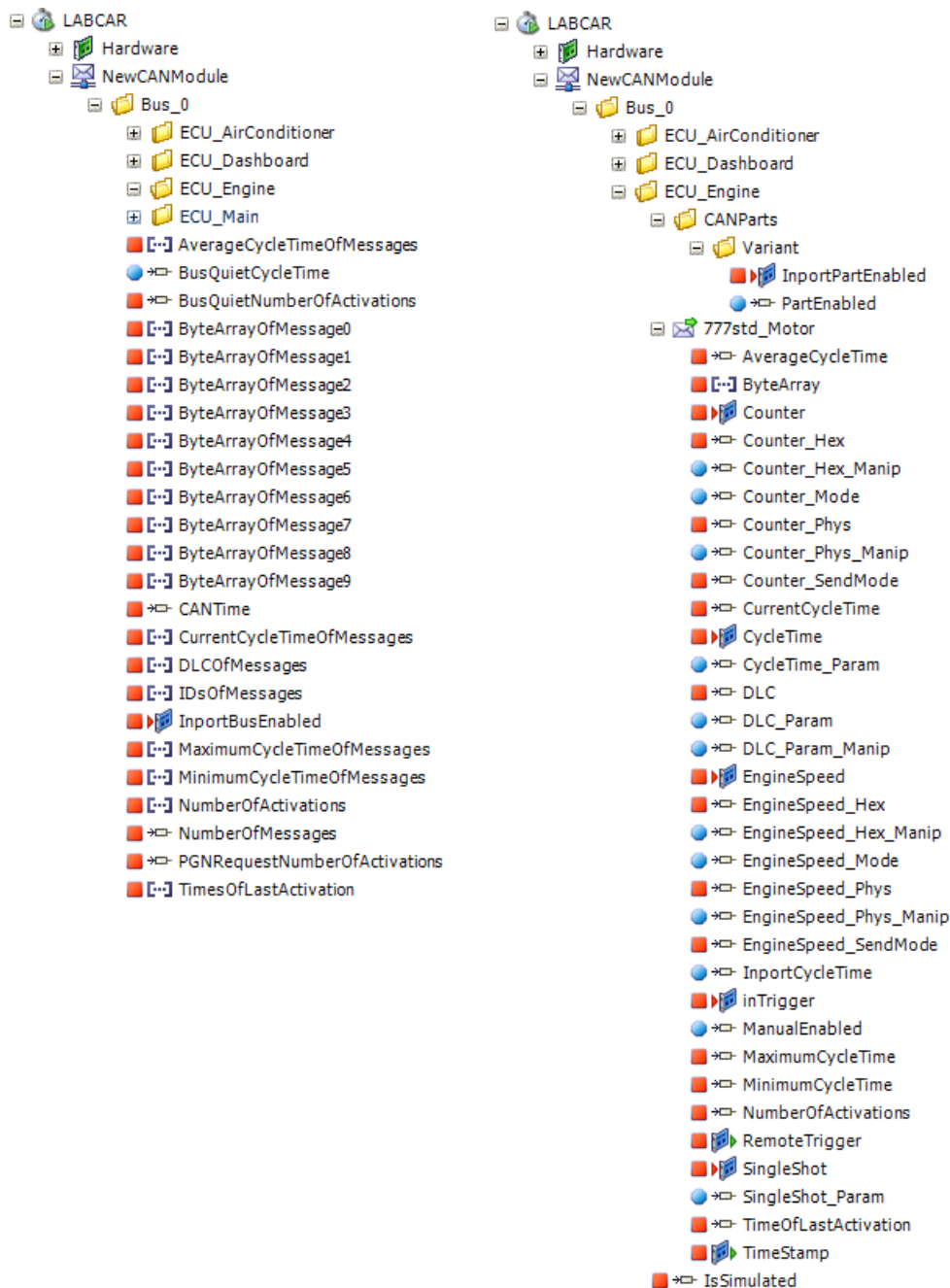


Fig. 3-13 The CAN Module in the Workspace Elements

The CAN module is shown in Workspace Elements with the following hierarchy.

Bus

This folder contains folders for the nodes and the following measure values and parameters (see Fig. 3-13 on page 125, left-hand part):

- **AverageCycleTimeOfMessages**
Array with n elements that contain the average cycle times of the n unspecified messages.
- **BusQuietCycleTime**
The periodic transmit rate of the BusQuiet (DM13) message
- **BusQuietNumberOfActivations**
Number of transmits of DM13 message
- **ByteArrayOfMessage**
Arrays via which the unspecified messages are available. The number n of these messages is defined in the CAN Editor of LABCAR-IP ([Options](#)) under "No. of Unspecified Messages".
- **CANTime**
The time scale for the CAN residual bus simulation – its zero point determined the first time the CAN send process is executed. By default, the CAN residual bus simulation is started together with the experiment itself in ETAS EE.
- **CurrentCycleTimeOfMessages**
Array with n elements that contain the current cycle times of the n unspecified messages.
- **DLCOfMessages**
Array with n elements that contain the DLCs of the n unspecified messages.
- **IDsOfMessages**
Array with n elements that contain the IDs of the n unspecified messages.
- **InportBusEnabled**
Inport for enabling/disabling a bus from the model.
 - $-0.5 < \text{value of the inport} < +0.5 = \text{FALSE}$: bus disabled
 - Otherwise TRUE (default = 1.0): bus enabled
- **MaximumCycleTimeOfMessages**
Array with n elements that contain the maximum cycle times of the n unspecified messages.
- **MinimumCycleTimeOfMessages**
Array with n elements that contain the minimum cycle times of the n unspecified messages.
- **NumberOfActivations**
Array with n elements that contain the number of activations of the n unspecified messages.

- **NumberOfMessages**
Number n of unspecified messages
- **PGNRequestNumberOfActivations**
The number of transmits of a PGN request message
- **TimesOfLastActivations**
Array with n elements that contain the times of the last activation of the n unspecified messages.

Nodes

The folders for the nodes contain the "CANParts" folder with the variants and messages of this node.

- **InportPartEnabled**
Inport for enabling/disabling a part from the model.
 - - 0.5 < value of the inport < + 0.5 = FALSE (default = 0): part disabled
 - Otherwise TRUE: part enabled
- **PartEnabled**
With this parameter, this part can be enabled and disabled during runtime.

Messages

The individual messages have the following measure values and parameters (* only with send messages):

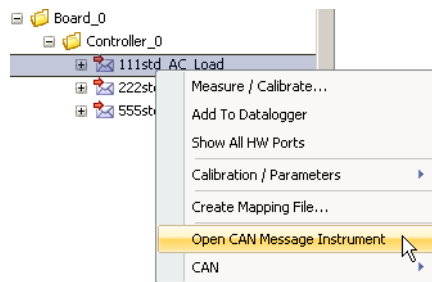
- **Signal name**
The hardware signal
- **Signalname_Hex**
The hex value of a signal
- **Signalname_Hex_Manip***
The modified hex value of a signal
- **Signalname_Mode**
Determines the mode of a signal. The following values are available:
 - The value comes from the model (0)
 - The physical value is modified (1)
 - The hex value is modified (2)
- **Signalname_Phys**
The physical value of a signal
- **Signalname_Phys_Manip***
The modified physical value of a signal
- **Signalname_SendMode**
Measure value containing the send type of the message (see "**Send Type**" on page 118): 0 = NotUsed, 1 = OnChange.

- **AverageCycleTime**
The average cycle time
- **ByteArray**
The byte array of the message
- **CurrentCycleTime**
The cycle time currently measured
- **CycleTime***
Inport for the cycle time (see "**InportCycleTime**" on page 128)
- **CycleTime_Param***
Parameter for specifying the cycle time
- **DLC**
The data length code of the message
- **DLC_param**
Parameter for the specification of a DLC defined by the user for this message
- **DLC_param_manip**
Boolean parameter with which the DLC specified by the user is enabled (1) or the value defined in the CAN Editor of LABCAR-IP (0).
- **InportCycleTime**
Specifies whether the cycle time is defined by the "CycleTime_Param" parameter (= False) or can be stimulated by the inport "CycleTime" (= True)
- **InTrigger***
Enables or disables the sending of messages (see "Enabling and Disabling Send Messages" on page 130). A value = 0 means "FALSE"; otherwise, InputTrigger = "TRUE".
- **ManualEnabled**
Parameter for manually activating send messages. Corresponds to the option "disabled" in the message GUI (see "Enabling and Disabling Send Messages" on page 130)
- **MaximumCycleTime***
The maximum cycle time measured
- **MinimumCycleTime**
The minimum cycle time measured
- **NumberOfActivations**
Number of activations of the message
- **RemoteTrigger***
An output that counts how often the message was triggered remotely.
- **InputTrigger**
Measure variable for the "InTrigger" signal

- **SingleShot***
Enables the one-off sending of a message. The parameter is of the type "Boolean" and is automatically reset to "False" after it was set to "True" for sending.
- **SingleShot_Param***
Parameter for triggering a "single shot". The same as the **Transmit Once** button in a message GUI.
- **TimeOfLastActivation**
Time the message was last activated
- **TimeStamp**
The time stamp that is determined from the current CANTime (see "**CAN-Time**" on page 126 on page 115) when the message is sent or received.

3.6.2 Instruments for CAN Messages

To create an instrument for displaying a message, right-click the message and select **Open CAN Message Instrument**.



An appropriate instrument for the message type is created which displays the content of the selected message.

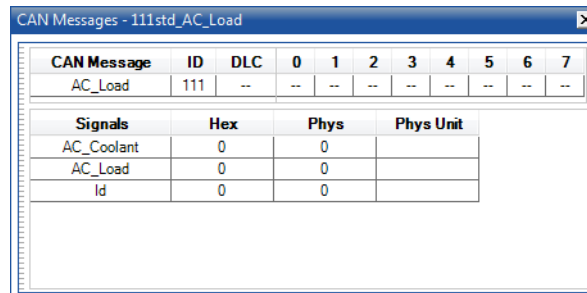


Fig. 3-14 Message GUI for a Receive Message

With a receive message, you cannot edit signals – the message and its signals are simply displayed.

With a send message, either the hex value or the physical value of a signal (the other value can no longer be edited in each case) can be changed in the window. The "Model" option simply has to be disabled for this purpose. Click **Apply** to make this change valid.

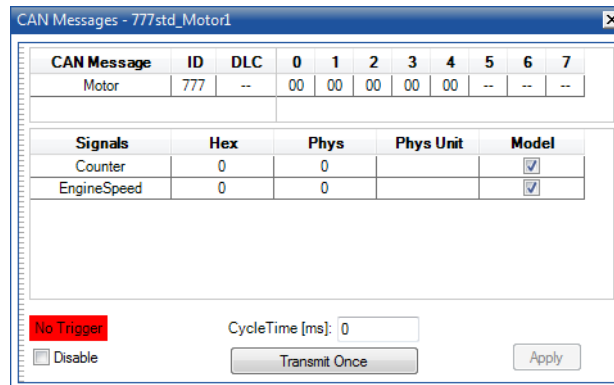


Fig. 3-15 Message GUI for a Send Message

Transmit Once results in the message being sent once on the CAN bus (see "Single Shot" on page 130).

Under "Cycle Time [ms]", you can specify the desired cycle time for the send message.

Note

If you use the "CycleTime" signal from the model, please observe the unit of this parameter (seconds).

3.6.3 Enabling and Disabling Send Messages

If "Trigger" is on a green background, the "inTrigger" signal, which is to be found in every send message, has the logical value "true". This signal is not from the CANdb file but is added by LABCAR-OPERATOR - it is used generally to control send messages from the model.

In the Message GUI itself, you can exert a certain influence on the send message as even a send message released by the "inTrigger" signal can be disabled by selecting the option "disable". Or to put it more precisely, a logical AND operation is applied to the "inTrigger" signal and the "disable" option: the message is only sent when "disable = false" (corresponds to "enable = true") and "Trigger = true".

Single Shot

If the result of this AND operation is "false", this message is not sent controlled by a timer. The **Transmit Once** button becomes active which results in a one-off sending of the message.

Alternatively, a "SingleShot" signal is available for every send message (such as "inTrigger") – a transition from 0 → 1 with this signal triggers a one-off sending of a message.

3.6.4 The CAN Monitor

To monitor traffic on the CAN bus, LABCAR-OPERATOR provides an instrument called a "CAN Bus Monitor".

Name	Message ID (hex)	Time	Cycle Time	Data Bytes (hex)	Details
Send Messages					
Board 0					
Controller 0 (Controller_0)					
Motor	777	0,000000	0,000000	00 00 00 00 00	Open GUI
Controller 1 (Controller_1)					
Receive Messages					
Board 0					
Controller 0 (Controller_0)					
AC_Load	111	0,000000	0,000000	00	Open GUI
AC_Control	222	0,000000	0,000000	00	Open GUI
SetPoints	555	0,000000	0,000000	00 00 00 00 00	Open GUI
Controller 1 (Controller_1)					

Fig. 3-16 The "CAN Bus Monitor" Instrument

In addition to the name of the CAN message, shown in a hierarchy "Bus/Node", the window contains the following columns:

- **Message ID (hex)**

The message ID

- **Time**

The message timestamp (time since the start of the simulation)

- **Cycle Time**

The message cycle time

- **Data Bytes (hex)**

The message data bytes

- **Details**

If the message is known within the project, you can open a message GUI (see Fig. 3-14 on page 129 and Fig. 3-15 on page 130) in this column with [Open GUI](#).

More information is displayed if you select "Extended View" (at the top of the window) instead of "Standard View":

- **Message Count**

Number of messages transferred

- **Mean Cycle Time**

The mean cycle time

- **Min. Cycle Time**

The minimum cycle time

- **Max. Cycle Time**

The maximum cycle time

The send messages listed are all those which were agreed in the CAN Editor. In the two columns to the right, an entry only appears when the message has been sent for the first time.

The receive messages listed are all those which were actually received on the relevant controller. If such a message was also agreed as a receive message in the CAN Editor, its name is known – otherwise "n/a" is specified.

The time span after which the information displayed is updated in each case can be specified under "Update Time Span [sec]" .

3.7 LIN Modules (Network Integration LIN)

LABCAR-NIL (Network Integration LIN) is an add-on for LABCAR-OPERATOR. It makes it possible to test ECU functions that include LIN¹ communication in accordance with LIN 2.0 and LIN 2.1/LIN 2.2.

The specification of LIN bus communication can be read in from one or more LDF² files and also adapted and extended manually. The user defines the LIN frames to be read and written and LABCAR-NIL automatically creates code. User-defined code can be added to this.

The signals of the selected frames are available in the Connection Manager where they can be connected to model inputs (receive frames) and model outputs (send frames).

Hardware Requirements

Working with LABCAR-NIL necessitates a real-time PC with a current version of ETAS RTPC and (at least) one LIN board of the type "iPC-I XC16/PCI"(LIN 2.0 only), "CAN-IB200/PCIe" or "CAN-IB600/PCIe"(LIN 2.0, and LIN 2.1/LIN 2.2) from the company IXXAT. The LIN board does not need to be integrated in LABCAR-RTC (Real-Time Execution Connector).

In LABCAR-IP you configure a LIN bus with a LIN module that contains the infrastructure for the simulation of the complete LIN bus.

Every LIN module supports up to 16 LIN boards each with one (iPC-I XC16/PCI) LIN controller or eight LIN boards each with two (CAN-IB600/PCIe) or four (CAN-IB200/PCIe) LIN controllers. Code generation supports both LIN master and LIN slave nodes.

Alternatively you can assign just one controller per LIN module and create the LIN bus from several different LIN modules together with the existing LIN controllers.

The LIN boards need external power to be supplied via the LIN controller connection.

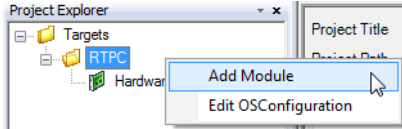
¹. **Local Interconnect Network**

². **LIN Description File**

3.7.1 Creating a LIN Module

Use the module wizard to add a new LIN module.

To add a LIN module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
 - In the "Project Explorer", select a target folder.
 - Right-click and select **Add Module**.
- 
- In the "Add Module Wizard", select **Add LIN Module**.
 - Click **Next**.
 - Enter a module name and select the protocol version.
 - Click **Finish**.
- In the "Project Explorer", a new module is added.

3.7.2 The LIN Editor

To open the LIN Editor, double-click the LIN module in the Project Explorer.

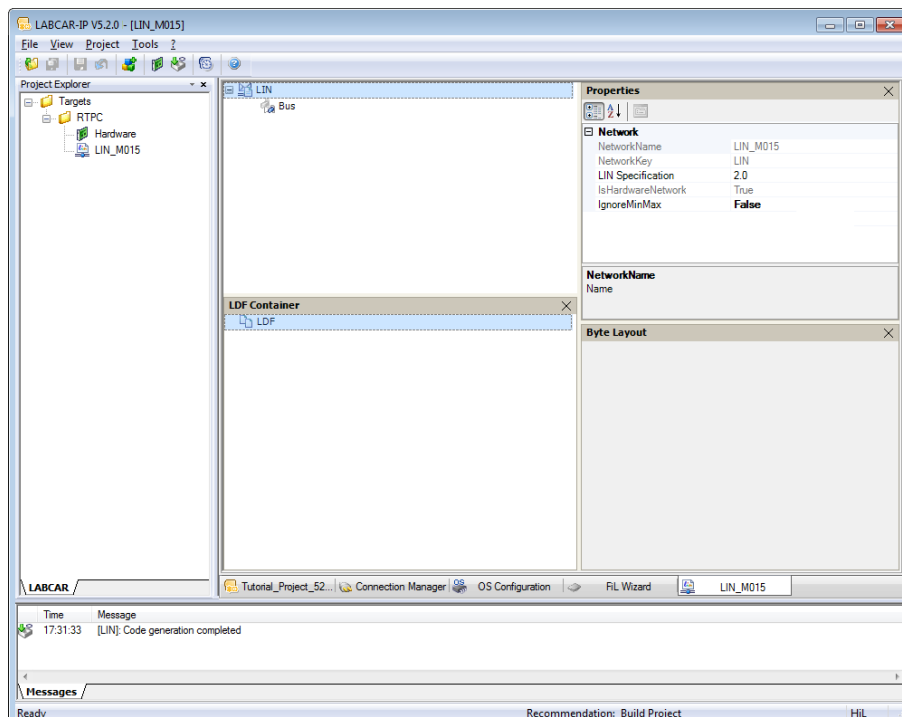


Fig. 3-17 The LIN Editor

The LIN Editor shows

- the LIN network (see page 135),
 - the LDF container (see page 135),
 - the "Properties" window (see page 136)
- and
- a display for the byte layout of frames (see page 137).

The LIN Network Window

This view shows the hierarchy of the network and the subordinate elements.

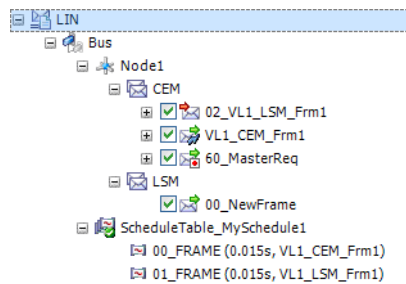


Fig. 3-18 The LIN Network

For information on network structure, refer to the section "Editing the LIN Network" on page 137; on its individual components to "The Component Parts of a LIN Network" on page 141.

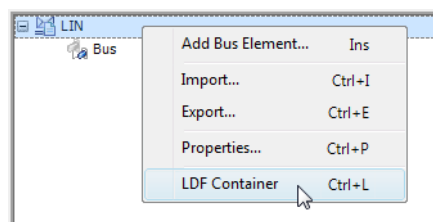
The LDF Container Window

This shows the contents of one or more imported LDF files.

To display the LDF container

If the LDF containers are not displayed automatically in the LIN Editor, proceed as follows:

- Right-click the LIN network node.
- From the shortcut menu, select **LDF Container**.

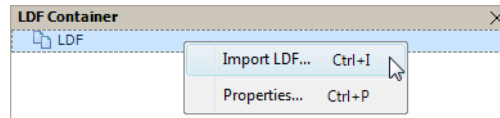


The LDF Container area is displayed.

To import an LDF file

LDF files can be imported via the shortcut menu of the LDF container – the elements inside the LDF container are, however, read-only. Make sure that the protocol version of the imported LDF file matches with the LIN module.

- Right-click the container and select **Import LDF**.



A file selector window opens.

- Select the file to be imported.

The file is imported and its contents displayed under the LDF container.

This process involves the creation of an LDF container that is named after the imported LDF file.

- Repeat the last step as often as necessary.
- LDF files already imported can be removed again using the **Remove** shortcut menu.

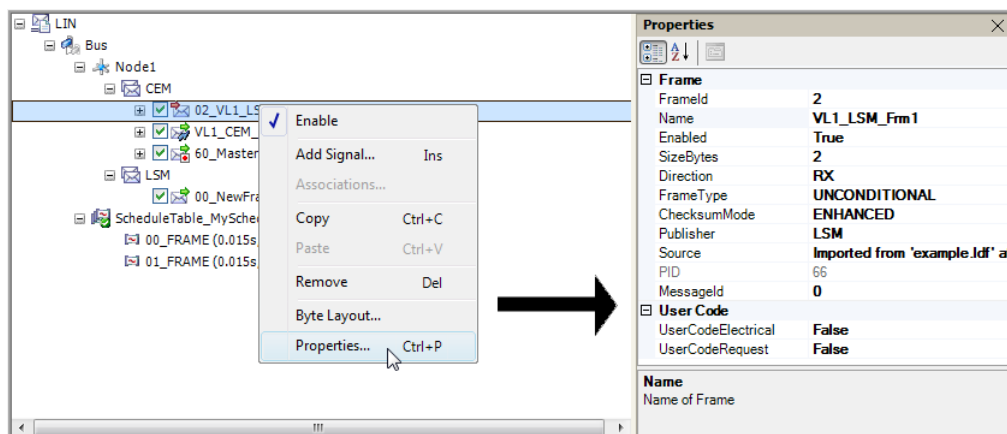
The "Properties" Window

Highlight elements in the tree views to display their properties in the "Properties" window. Editable properties can then be edited directly in the "Properties" window.

To display properties

If the properties are not displayed automatically, proceed as follows:

- Right-click the element whose properties you want to display.
- From the shortcut menu, select **Properties**.



The properties of the selected element are displayed in the "Properties" window.

For more detailed information on editing properties, refer to the section "Editing Properties" on page 138.

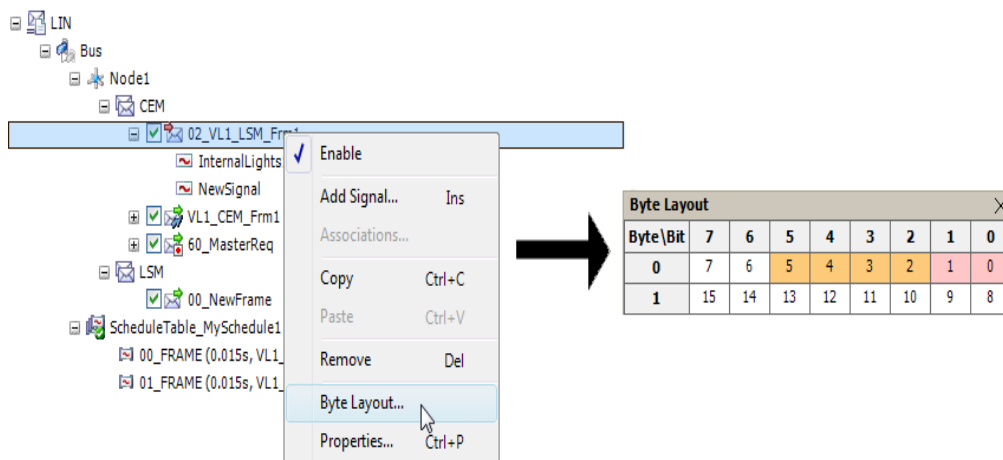
The Byte Layout of Frames

The contents of a frame selected in the network view are shown in this area.

To display the byte layout

To display the byte layout of a frame, proceed as follows:

- Right-click the frame whose layout you want to display.
- From the shortcut menu, select **Byte Layout**.
The byte layout of the frame is displayed.



3.7.3 Editing the LIN Network

The LIN network is edited exclusively via the shortcut menu of the relevant element. The elements contained in a network are described in the section "The Component Parts of a LIN Network" on page 141.

Note

*The properties of elements of the LDF container can **not** be edited, but an LDF container element can be used to create a corresponding element in the LIN network by copying or moving.*

To edit the network, there are a number of items in the shortcut menu of the relevant elements of which several can be used on a multiple selection (of elements of the same kind).

To add elements

- Highlight the higher-ranking element and select **Add (Element)** from the shortcut menu.

Note

Names of parts, frames and signals can only contain characters permissible for ANSI-C identifiers.

To remove elements

- Highlight the element and select **Remove** from the shortcut menu.

To copy elements

- Highlight the source element and select **Copy** from the shortcut menu.
- Highlight the higher-ranking target element and select **Paste** from the shortcut menu.

To copy element properties

- Highlight the source element and select **Copy** from the shortcut menu.
- Highlight the desired target element and select **Paste** from the shortcut menu.

To move elements

- Highlight the source element.
- Use the mouse to drag the highlighted source element to the higher-ranking target element and then release the mouse button.

Note

The actions described above are not always permissible – in these cases, the relevant item in the shortcut menu is either disabled or not available.

Editing Properties

The properties of an element of the LIN network can be edited in the "Properties" window. Some properties of some elements (such as the PID of a LIN frame) cannot be edited as they are calculated automatically. These properties are grayed out in the "Properties" window. The properties of the elements of the LDF container cannot be edited either.

To display the "Properties" window

- Highlight an element and select **Properties** from the shortcut menu.
The "Properties" window is displayed in the LIN Editor.

To edit properties of several elements

If several elements of the same type are highlighted in the network, the properties that all selected elements share are displayed.

- Open the "Properties" display.
- Use the mouse to select the desired elements while keeping the CTRL key pressed down.

Note

If a property that identifies an element within its higher-ranking element (e.g. a signal name within a frame) is modified when editing several elements, an error message may be issued if the property of another element of the same higher-ranking element is also given the same value.

Validation

Before the LIN module is saved, the LIN network is validated. This includes for example the verification of aspects of the LIN specification that are relevant for code generation. Error messages are displayed in the log window.

Note

Ignoring error messages can lead to compiler errors or undesired runtime behavior during the build process of the project.

Modules

Several LIN modules can be created within one LABCAR-OPERATOR project. These modules can be configured independently of one another although a LIN board can only be used in one particular module.

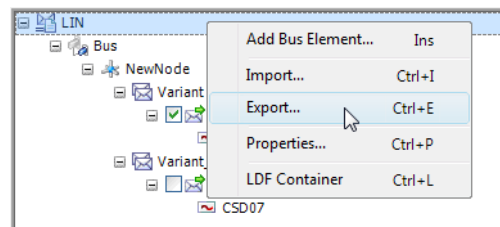
Networks

Within a LABCAR-OPERATOR LIN module, the root node in the tree view corresponds to a LIN network – bus elements are ordered under a network. LIN networks can be imported and exported.

The hardware topology of a module is stored in the `LINConfiguration.xml` file. This file refers to the configuration files of the parts (*.llp) and schedule tables (*.lsc) contained in the network topology. All files are stored within the module directory.

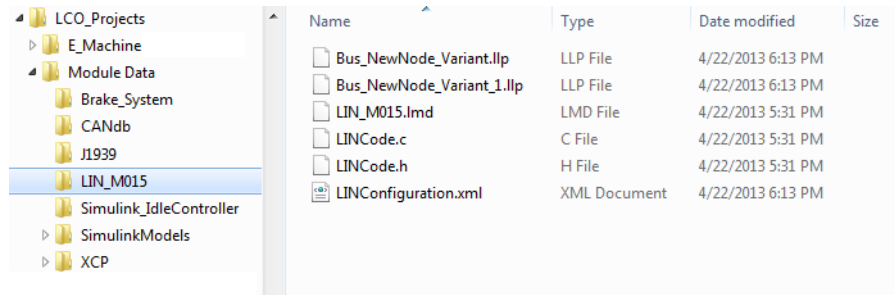
To export a network

- Right-click the LIN network and select **Export**.



- Select an existing folder or create a new one and then click **OK**.

The files are created in a directory named after the LIN module.



To import a network

- Right-click the LIN network and select **Import**.
A file selector window opens.
- Select the file `LINConfiguration.xml`.

Note

Files containing user-defined C code (*.c and *.h) (see "User-Defined C Code" on page 155) are not exported or imported. These have to be transferred manually if they are needed.

3.7.4 The Component Parts of a LIN Network

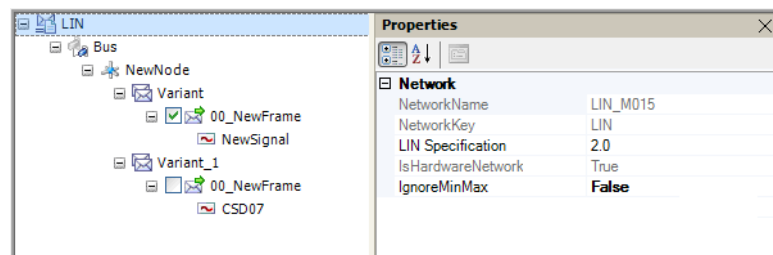
This section contains a description of the component parts of the LIN network in LABCAR-IP.

These are:

- "LIN network" on page 141
- "Bus" on page 142
- "Nodes" on page 144
- "Schedule Tables" on page 144
- "Commands" on page 145
- "Parts" on page 145
- "Frames" on page 148
- "Signals" on page 151

LIN network

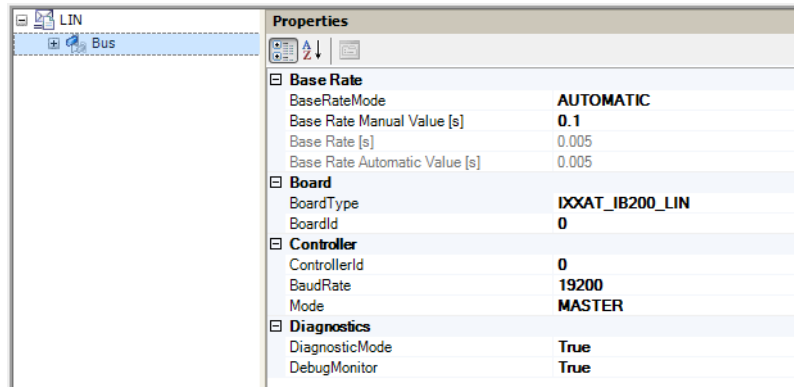
The global properties of the LIN network are displayed when it is selected in the "Properties" window.



Network:

- **NetworkName**
Editable name of the LIN network
- **NetworkKey**
Unique network ID (is generated automatically)
- **LIN Specification**
LIN 2.0 or LIN 2.1/2.2
- **IsHardwareNetwork**
Specifies whether this is a real network with configured hardware or a network in an LDF file.
- **Ignore Min/Max**
Ignores the Min/Max restrictions of signal values. If this option is set, the physical signal values are not limited by the Min/Max restriction in the signal properties (before these are converted to electrical signal values).

Bus



Base Rate:

The "BaseRate" property defines the rate at which communication between the LABCAR-OPERATOR project and the selected LIN module controller takes place.

The current value of BaseRate is shown in the "Properties" window of the controller – it is determined in accordance with the following controller properties:

- **BaseRateMode**

"BaseRateMode" specifies how "Base Rate" is to be determined. "Base Rate" can be calculated

- manually ("MANUAL")

or

- automatically ("AUTOMATIC")

In automatic calculation, a double or quadruple overclocking ("AUTOMATIC2x" / "AUTOMATIC4x") can be selected.

- **Base Rate Manual Value [s]**

"Base Rate Manual Value" is the desired rate entered manually by the user.

- **Base Rate [s]**

The base rate

- **Base Rate Automatic Value [s]**

"Base Rate Automatic Value" is calculated from the greatest common divisor of all values of the "Delay" property of all commands as well as all values of the "Base Rate" property of all parts under the selected controller. The "Base Rate" property of a part is defined by the parameter "Master Time Base" from the imported LDF files. This property of a part can also be modified later.

Board:

- **BoardType**

Type of board used ("iPC-I XC16/PCI", "CAN-IB200/PCIe" or "CAN-IB600/PCIe")

- **BoardID**

Unique identifier of the board within the network

Controller:

- **ControllerID**

Unique identifier of the controller on a board

Note

On the CAN-IB200, channel 1 is mapped to controller 1 and channel 2 is mapped to controller 0!

- **BaudRate**

The transfer rate of the controller

- **Mode**

"MASTER" or "SLAVE".

Note

There can only be one controller in "MASTER" mode within a module!

Diagnostics:

The following properties are available to support the execution of diagnostic tasks:

- **DiagnosticMode**

If this property is enabled ("DiagnosticMode" property = TRUE), Boolean calibration variables are generated allowing you to prevent communication on non-diagnostic frames (calibration variable = TRUE) or to allow it (calibration variable = FALSE) during experiment runtime.

- **DebugMonitor**

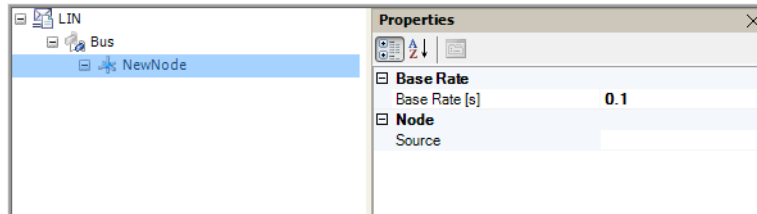
If this property is enabled, measure variables are generated that make it possible to observe communication between the module and the available controllers during experiment runtime.

Corresponding measure variables are created for sent and received frames (master and slave) as well as for sent frame headers of requested frames (master only). These show the complete contents of all bytes transferred recently.

Note

As the communication between the module and the controller takes place at a higher rate than with the base rate set (the data of several frames is exchanged in every cycle), these measure parameters are not suitable for recording communication without loss.

Nodes



Base Rate:

The following property is available for a part for configuring the time behavior:

- **Base Rate []**

The base rate of a part is defined by the parameter "Master Time Base" from the imported LDF files.

This property of a part can also be modified later. The "BaseRateAutomaticValue" property (see "**Base Rate Automatic Value [s]**" on page 142) of a controller is calculated using the base rate.

Node:

- **Source**

Source of the node (after import)

Schedule Tables

A schedule table is assigned to a higher-ranking controller (in Master mode). Commands are arranged under a schedule table (see "Commands" on page 145). Within the module the content of a schedule table is stored in files with the extension `.lsc`.

Schedule tables have a unique name within the network. If a schedule table with a given name already exists, a number is added to the name of the schedule table to be added as a suffix.

To modify startup behavior

- Highlight the schedule table and enable or disable **Use as Startup** in the shortcut menu.

Note

Only one schedule table can be marked as "Startup" within a controller. If a different schedule table is marked as "Startup", the previous highlighting is removed.

Commands

A command is assigned to a higher-ranking schedule table. Commands are identified by a unique position within a schedule table. Commands are processed in the order of their position.

To change position

- Highlight the required command and select **Move Up** or **Move Down** from the shortcut menu.

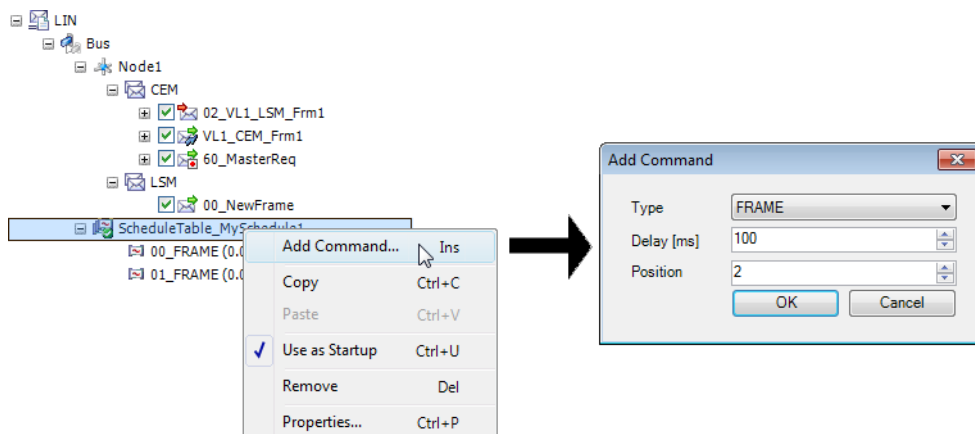
Note

The position of a command can also be modified via the "Properties" window. If the position of a command changes, the positions of other commands may well be adjusted automatically accordingly. The "Position" property of a command is unique for every command of a schedule table.

To create a user-defined command

In addition to the possibility of importing a command from an LDF file, you can also create user-defined commands.

- Highlight the higher-ranking schedule table and select **Add Command** from the shortcut menu.



Note

When creating a user-defined command, you initially only have to specify the general parameters – "Delay" and "Position". The specific parameters for the type of command can then be modified in the "Properties" field.

Parts

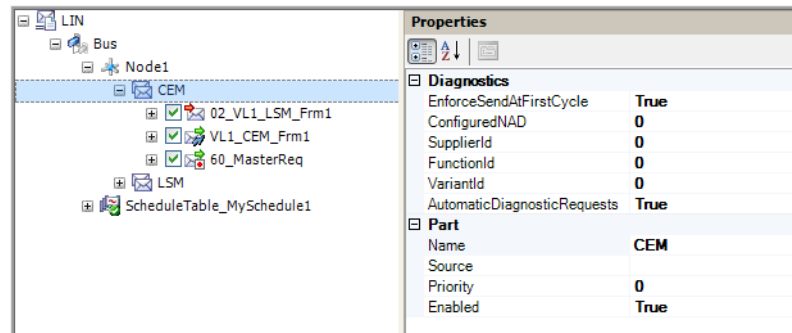
A part is assigned to a node; frames are arranged under a part. Parts can be imported and exported. Within the module the content of a part is stored in files with the extension `.llp`.

Parts have a unique name within the entire network. If a part with a given name already exists, the name of the part to be added is extended with a consecutive number.

Prioritization, Enabling and Disabling of Parts

Parts are used to model variants of a network node. For this purpose, copies of a part can be created including all frames and signals defined beneath this part.

Parts can be enabled and disabled using "Enabled". The processing of the frames and signals of this part can be controlled using "Priority": Parts with a higher priority overwrite the inputs and outputs of the parts with a lower priority in every simulation cycle.



Part:

- **Name**
The name of the part
- **Source**
Source of the part (after import)
- **Priority**
With this property, you can specify the priority of the part. The priority must have a unique value – parts are run according to their priority.
- **Enabled**
Every part can be enabled or disabled (regardless of the other parts) during experiment runtime. If a part is disabled, communication of all frames under the selected part is prevented.

Note

Variant handling during experiment runtime is possible with the properties "Enabled" and "Priority".

Diagnostics:

The following properties are available to support the execution of diagnostic tasks:

- **EnforceSendAtFirstCycle**
To optimize performance, the code is generated so that the data of a frame to be sent is only written to the controller transmission buffer if at least one frame signal has changed since the last transmission.

In the first cycle of the simulation, the frame with the configured start values of the signals is written to the transmission buffer. If the "EnforceSendAtFirstCycle" property is disabled ("FALSE"), this initialization of the

transmission buffer is prevented. This can be used to simulate one part not yet taking part in network communication at the beginning of the simulation.

- **ConfiguredNAD**
- **SupplierId**
- **FunctionId**
- **VariantId**

These properties identify the node defined by the part in the network. This is necessary for the automatic handling of the `ASSIGN_NAD` and `LIN PRODUCT ACTIVATION` commands.

- **AutomaticDiagnosticRequest**

This property makes it possible to enable code generation for the automatic handling of the commands `ASSIGN_NAD` and `LIN PRODUCT ACTIVATION` in accordance with the LIN specification for this part. The automatic handling of these commands only takes place on controllers in slave mode.

Note

If this property is enabled, a measure variable is created for this part showing the current value of the network address "Current_NAD".

To export a part

- Highlight the part to be exported and select **Export Part** from the shortcut menu.
A file selector window opens.
- Select a directory and a file name and click **OK**.
The part is exported.

To import a part

- Highlight the higher-ranking controller and select **Import Part** from the shortcut menu.
A file selector window opens.
- Select a file and click **OK**.
The part is imported.

Frames

A frame is assigned to a higher-ranking part; signals are arranged under a frame (depending on the type of frame).

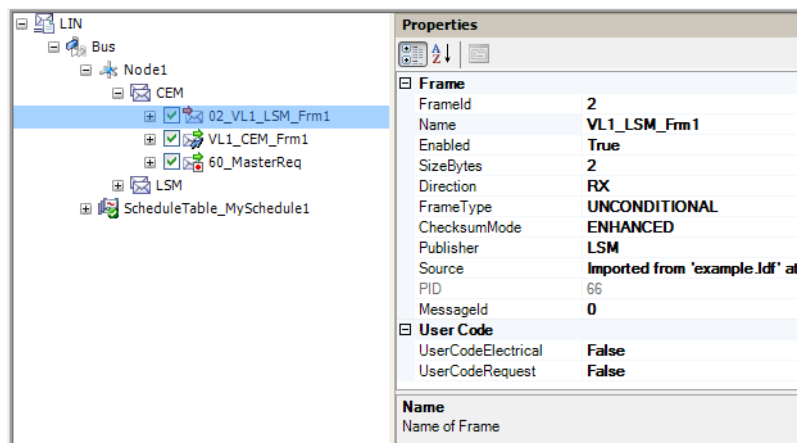
In addition to the "Properties" window, the "Byte Layout" window can also be displayed for frames. This shows the byte allocation in the payload of the frame with the signals.

To display the byte layout

- Highlight the frame and select **Byte Layout** from the shortcut menu.

Note

If errors occur in the byte layout, e.g. by two signals overlapping or the incorrect specification of the length of a signal or frame, the incorrect bits are shown in red.



Frame:

- **FrameId**
ID of the frame
- **Name**
Name of the frame
- **Enabled**
Frame enabled in simulation ("TRUE") or ("FALSE")
- **SizeBytes**
Size of the frame (in bytes)
- **Direction**
Transport direction (TX = send, RX = receive)

- **FrameType**
Type of frame:
 - UNCONDITIONAL
 - SPORADIC
 - EVENTTRIGGERED
 - DIAGNOSTIC
 - USERDEFINED
- **ChecksumMode**
Mode for checksum calculation
 - CLASSIC
 - ENHANCED
- **Publisher**
Editable name of the sending node (is taken from LDF)
- **Source**
Source of the element
- **PID**
Protected identifier of the frame (calculated value)
- **MessageId**
Optional message ID of the frame for using the command ASSIGN_
FRAME_ID

User Code:

This is where you can specify whether user-defined C code should be added to the frame (see "User-Defined C Code" on page 155).

- **UserCodeElectrical**
This code is called before sending or receiving a frame and enables access to the frame payload.
- **UserCodeRequest**
This code is called after a frame request has been received.

To create a user-defined frame

In addition to the possibility of importing a frame from an LDF file, module or part, you can also create user-defined frames.

- Highlight the higher-ranking part and select **Add Frame** from the shortcut menu.

Note

Within a part, the combination of the ID of the frame and its name must be unique!

Note

Not all combinations of the parameters of a frame are permissible!

For example, the combination of the properties

- *Type = DIAGNOSTIC,*
- *the send direction*
- and*
- *the controller mode*

automatically results in the ID of the frame.

*If, with certain combinations, individual parameters can no longer be freely selected or only selected with a limited value range, the relevant input fields are disabled or their value range limited accordingly. If the input were to result in serious errors in the configuration, the input mask cannot be closed with **OK**.*

To enable and disable frames

Several frames with the same ID may be defined within one part providing they each have a different name. For the purposes of code generation, however, only one frame with a specific ID can be enabled in one part. The other frames therefore have to be excluded from code generation.

- **Using the network tree view:** Enable or disable the checkbox next to the frame icon in the network tree view.
- **Using the shortcut menu:** Highlight the frame and enable/disable it in the **Enable** shortcut menu.
- **Using the "Properties" window:** Open the "Properties" window and edit the "Enabled" property ("True" or "False").

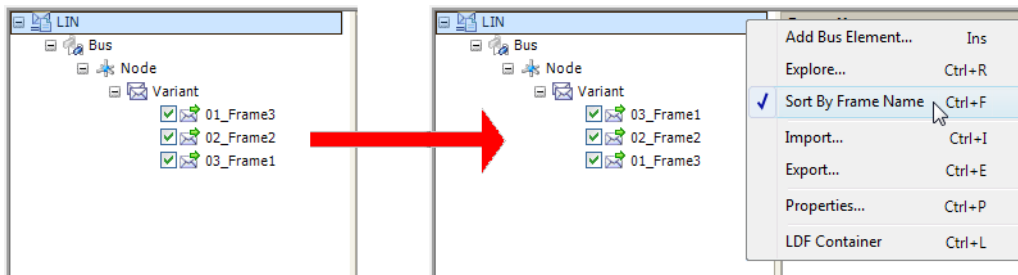
To remove a frame

- Select **Remove** from the shortcut menu of the frame list.
The frame is removed.

To sort frames by name

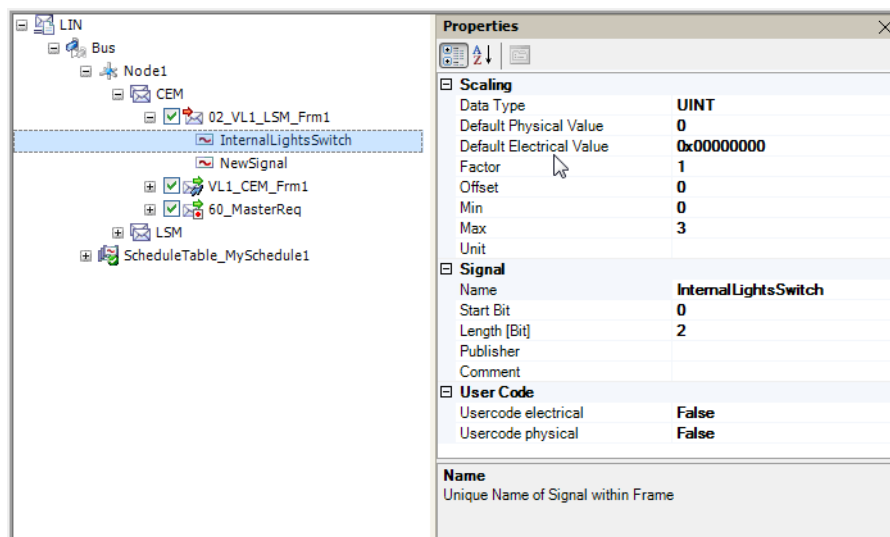
Frames are normally sorted by frame identifier.

- To sort the frames by name, right-click the module name and select the option **Sort by Frame Name**.



Signals

A signal is assigned to a higher-ranking frame: The name of a signal under this frame must be unique.



Scaling:

- **Data Type**
"UINT", "BOOLEAN" or "BYTEARRAY"
- **Default Physical Value**

- **Default Electrical Value**

A signal of a frame can be assigned a default value. If the signal is connected to another signal in the Connection Manager, this value is ignored.

The advantage of the default value is in the import/export of LIN modules as the default value is then also available if the LIN module is used again in another LABCAR-OPERATOR project.

- **Factor, Offset**

A factor a and an offset b for the for scaling the signal

- **Min, Max**

This is where limits for a plausibility check are specified (see "Min/Max Values" on page 119)

- **Unit**

The physical unit of the signal

Signal:

- **Name**

Name of the signal

- **Start Bit**

Start bit of signal

- **Length**

Signal length (in bit)

- **Publisher**

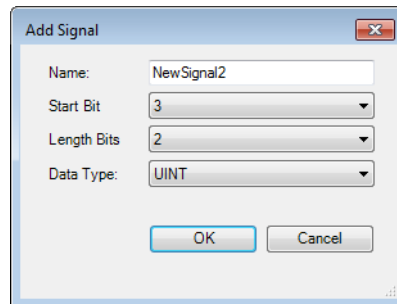
Editable name of the sending node (taken from LDF)

- **Comment**

An (optional) comment

To create a user-defined signal

- Highlight a frame and select **Add Signal** from the shortcut menu.



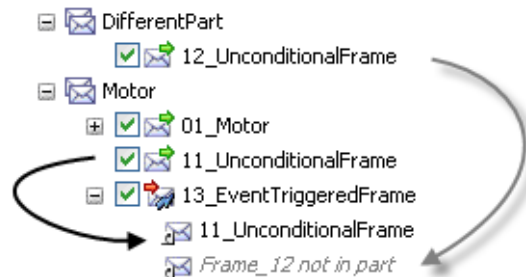
Note

*Not all combinations of the parameters of a signal are permissible! For example, the property "Signal Type = BOOLEAN" automatically results in the length of the signal. If, with certain combinations, individual parameters can no longer be freely selected or only selected with a limited value range, the relevant input fields are disabled or their value range limited accordingly. If the input were to result in serious errors in the configuration, the input mask cannot be closed with **OK**.*

Associations

An "association" is the link of an unconditional frame to an event triggered frame or sporadic frame. An association is assigned to a higher-ranking frame.

The association is shown in the network tree view as a link of the unconditional frame under the assigned event triggered frame or sporadic frame.



The above example shows an event triggered frame (ID 13) with two associations with the two unconditional frames ID 11 and ID 12.

An event triggered frame or sporadic frame can have associations with several unconditional frames, but only one unconditional frame can be in the same part as the event triggered frame or sporadic frame. Associations to unconditional frames that are defined in a different part are thus shown grayed out.

To generate an association

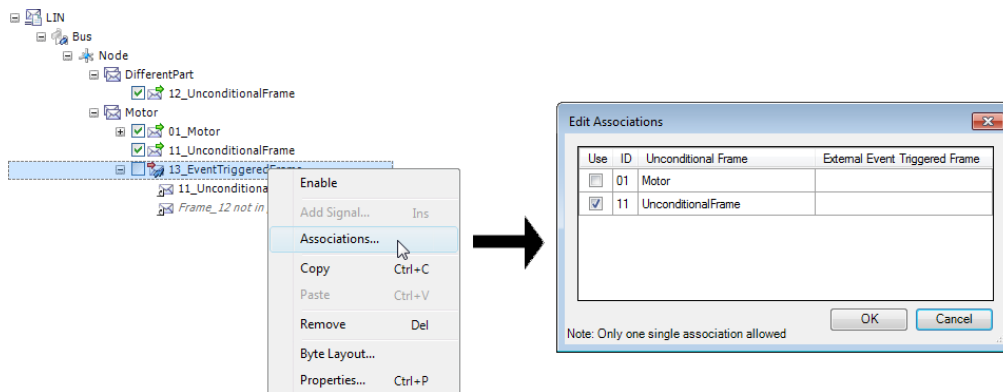
- Use the mouse to drag the unconditional frame to an event triggered frame or a sporadic frame and then release the unconditional frame.

To remove an association

- Highlight the association to be removed under the higher-ranking event triggered frame and select **Remove** from the shortcut menu.

To edit an association

- Highlight the higher-ranking event triggered frame and select **Associations** from the shortcut menu. The "Edit Associations" window opens.



- Select the unconditional frame (within the same part) to be associated with the event triggered frame or sporadic frame. Only one single unconditional frame can be selected.

Note

Associations of unconditional frames of the higher-ranking part with other event triggered frames or sporadic frames are shown grayed out. These associations can only be modified by editing the other event triggered frame.

Event Triggered Frames

- In the case of an association with an event triggered frame, both the master and the affected slaves must be aware of this relationship. The association between an event triggered frame and an unconditional frame must therefore be specified in every affected part.
- For parts under controllers in master mode, the direction of the event triggered frame and the unconditional frame must be RX (reading), under controllers in slave mode TX (sending).

Sporadic Frames

- In the case of an association with a sporadic frame, only the sending part has to be aware of this relationship.
- For all parts in which this kind of relationship occurs, the direction of the sporadic frame and the unconditional frame must be TX (sending).

3.7.5 User-Defined C Code

In LABCAR-NIL it is possible to add further user-defined code to the automatically generated code to manipulate or extend the content of frames or signals. With signals, this applies both to the physical representation (scaled and with physical units) and for the electrical representation (unscaled and without physical unit).

- **Send Frames**

The payload of the frame can be manipulated just before sending – use the "UserCodeElectrical" property of the frame for this purpose.

- **Receive Frames**

The payload of the frame can be manipulated right after receiving – use the "UserCodeElectrical" property of the frame for this purpose.

- **Frame Header**

The requested frame header can be manipulated just before sending (only for controllers in Master mode) – use the "UserCodeRequest" property of the frame for this purpose.

- **Send Signals**

The signal can be manipulated right after reading the inport as a value of the type "double" – use the "UserCodePhysical" property of the signal for this purpose.

The signal can be manipulated just before being added to the payload of the frame as an electrical value (unscaled and without a physical unit) as a value of the type "uint64" – use the "UserCodeElectrical" property of the signal for this purpose.

- **Receive Signals**

The signal can be manipulated right after being extracted from the payload of the frame as an electrical value (unscaled and without a physical unit) as a value of the type "uint64" – use the "UserCodeElectrical" property of the signal for this purpose.

The signal can be manipulated just before writing the output of the LIN module as a physical value (scaled and with a physical unit) as a value of the type "Double" – use the "UserCodePhysical" property of the signal for this purpose.

Procedure

After the LIN configuration has been saved, the module directory contains the header file `LINCode.h` that contains the function declarations for inserting the user code.

At the end of the `LINCode.h` file, there is a comment line

```
// Declarations for user code insertion:
```

The function calls of the user-defined code are declared below this comment line – these functions must be implemented in the user-defined code.

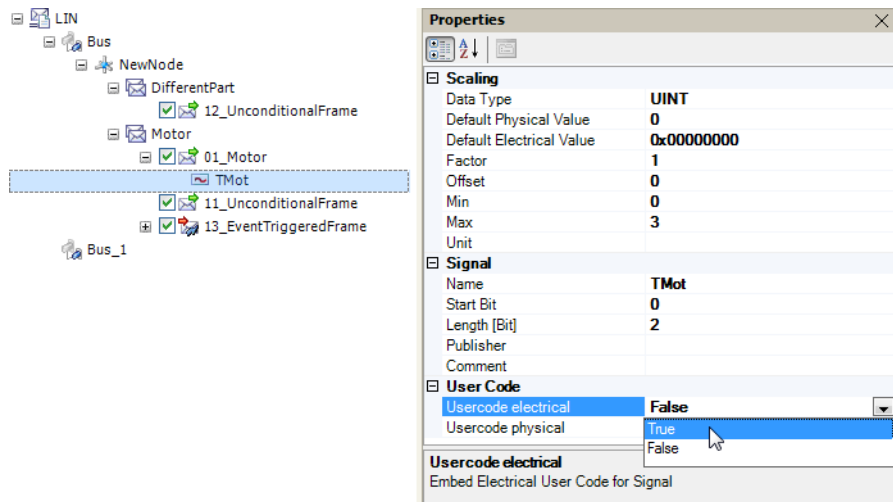
To declare user-defined code

- Edit the properties
 - "UserCodePhysical"
 - "UserCodeElectrical"

and

- "UserCodeRequest"

of the frames and signals to which you want to add user-defined code.

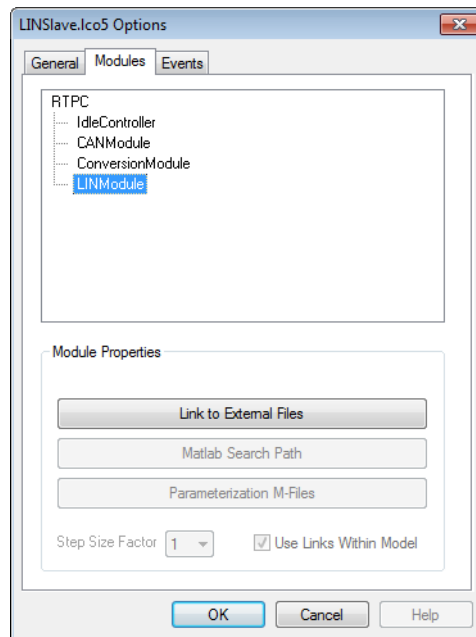


- Select **File** → **Save** to save the module.
- Use an appropriate editor to open the `LINCode.h` file in the module directory.
- Copy the function declarations of the user-defined code into a text file.
- Name this text file for example `UserCode.c` and store it within the LIN module directory.
- To your C-file `UserCode.c` add the line


```
#include "LINCode.h"
```

 so the function and structure declarations are known.
- Implement the user-defined code in `UserCode.c`.
- In the main menu, select **Project** → **Options**.

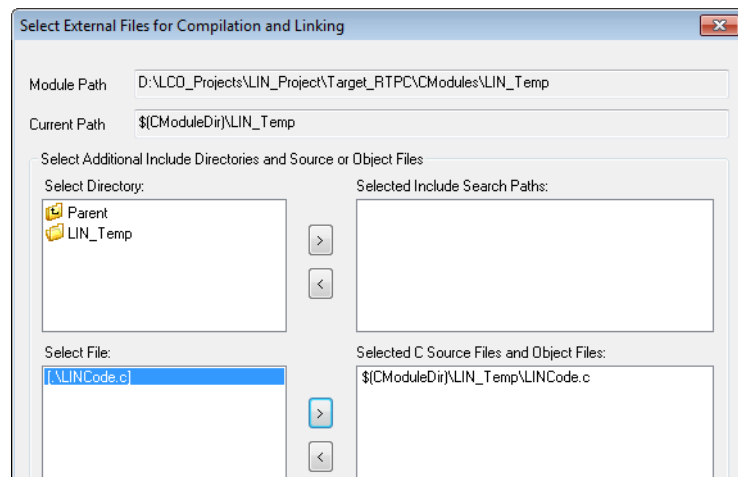
- In the Options window, select the "Modules" tab.



- Select "LINModules" and click **Link to External Files**.

The "Select External Files for Compilation and Linking" window opens.

- Select the source file(s) to be integrated.



Note

Only files in the module directory can be selected.

The selected files are shown in the "Selected C Source Files and Object Files" field.

- Click **OK**.

These files are then always compiled and linked when code is generated.

- Exit the "Options" tab using **OK**.

Data Types and Structures

The following data types and structures are used in the function declarations of the user-defined code for manipulating a frame:

Constants

```

/* The following constants are used for the "mtype" field in ixx-
at_lin_header_t: */
#define IXXAT_LIN_MTYPE_DATA      0 // Standard message data frame
#define IXXAT_LIN_MTYPE_INFO     1 // Info message type
#define IXXAT_LIN_MTYPE_ERROR    2 // Error message type
#define IXXAT_LIN_MTYPE_WAKEUP   3 // Error message type
#define IXXAT_LIN_MTYPE_REQUEST  0x10 // Request id
#define IXXAT_LIN_MTYPE_EVT_TRIG_DATA 0x11 // Event triggered data
                                     // frame (for slave only)

/*
The following constants are used for the "crcmodel" field in ixx-
at_lin_data_t:
*/
#define IXXAT_LIN_CRCMODEL_CLASSIC 0
#define IXXAT_LIN_CRCMODEL_ENHANCED 1

```

Data transmission: sending and receiving

```

typedef struct {
    ixxat_lin_header_t hdr;
    union {
        ixxat_lin_load_data_t      data;
        ixxat_lin_load_request_id_t request;
        ixxat_lin_load_error_t     error;
        ixxat_lin_load_status_t    status;
    };
} ixxat_lin_data_t;

typedef struct {
    unsigned long time_stamp; // Time stamp (for receive messages)
                             // One tick corresponds to 125 microsec
    unsigned char mtype;     // Message type
    unsigned char minfo;     // Message info (for receive messages)
} ixxat_lin_header_t;

```

Data transmission: receiving: return error

```

typedef struct {
    unsigned short errorcode; // LIN error code (receive only)
} ixxat_lin_load_error_t;

```

Data transmission: receiving: return status

```
typedef struct {
    unsigned short status;          // LIN status (receive only)
} ixxat_lin_load_status_t;
```

Data transmission: receiving: requests

```
typedef struct {
    unsigned char id;              // LIN message ID
    unsigned char crcmodel;       // CRC model (classic / enhanced)
    unsigned char length;         // Slave buffer data length
} ixxat_lin_load_request_id_t;
```

Data transmission: sending and receiving: frame payload

```
typedef struct {
    unsigned char id;              // LIN message ID.
                                   // For event-triggered frames:
                                   // The unconditional frame id.
    unsigned char evt_trig_id;    // For event-triggered frames:
                                   // The event triggered frame id.
    unsigned char crcmodel;       // CRC model (classic / enhanced)
    unsigned char length;         // Message data length
                                   // (0 disables this message)
    union {
        unsigned char data[8];    // LIN data bytes.
                                   // Standard byte access.
        // Some alternative access mechanisms:
        unsigned short uwddata[4];
        unsigned int udwdata[2];
        unsigned long long uqwddata;
        signed char sdata[8];
        signed short swdata[4];
        signed int sdwdata[2];
        signed long sqwdata;
    };
} ixxat_lin_load_data_t;
```

3.7.6 The LIN Module in ETAS EE

This section explains how to work with a LIN module in the experiment environment ETAS EE.

Once the project has been created and compiled on the target, a number of inputs and outputs for control and communication are available for every LIN module in the project. The inputs and outputs of a LIN module are shown under the module in the Experiment Explorer.

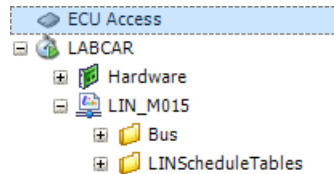
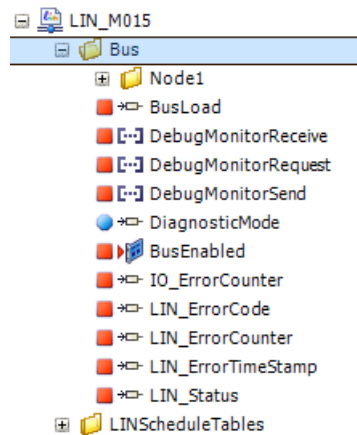


Fig. 3-19 The LIN Module in the Workspace Elements

In the Workspace Elements, every LIN module is displayed in its own node ("Name of the module"). Under this module node, a bus element node ("Bus") is displayed for every configured bus element. All inputs and outputs for controlling the bus element or the elements contained therein are displayed under this bus element node.

Bus

Inputs and outputs for controlling and monitoring communication with the configured LIN controller are shown under "Bus".



- **BusLoad**

Shows the current load of the LIN bus in percent.

Note

If this feature is not supported by the board, a value of "-1" will be displayed instead!

- **DebugMonitorReceive**
- **DebugMonitorRequest**
- **DebugMonitorSend**

Measure variables for displaying the bytes last transferred between the LIN module and controller.

Note

As communication between the module and the controller takes place at a higher rate than with the base rate set (the data of several frames is exchanged per cycle), these measure variables are not suitable for recording communication without loss.

- **DiagnosticMode**

Calibration variable for enabling/disabling the diagnostic mode of the controller

- **BusEnabled**

Inport for the enabling/disabling of a bus.

- $-0.5 < \text{value of the inport} < +0.5 = \text{FALSE}$: bus disabled
- Otherwise TRUE (default = 1.0): bus enabled

Note

The activation/deactivation of a bus is controlled by the respective board. The change of the communication state for a certain controller can cause interferences on the buses of other controllers. These interferences last as long as the change of state itself.

- **IO_ErrorCounter**

Error counter for the communication with the selected controller.

Incorrect specifications of the "BoardId" and "ControllerId" properties and errors in the LIN board firmware are frequent causes of error.

- **LIN_ErrorCounter**

Error counter for errors in LIN communication with the IXXAT driver.

- **LIN_ErrorTimeStamp**

Timestamp for the last time a "LIN_ErrorCode" occurred. The timestamp is specified in microseconds since the start of LIN communication and determined by the LIN board.

- **LIN_ErrorCode**

Error code for communication via the LIN protocol.

Incorrect specifications of the time behavior and other properties of the elements of the LIN network are frequent causes of error.

```
#define BCI_LIN_NO_ERROR 0x00
#define BCI_LIN_BIT_ERROR 0x01
#define BCI_LIN_CHECKSUM_ERROR 0x02
#define BCI_LIN_ID_PARITY_ERROR 0x03
#define BCI_LIN_SLAVE_NOT_RESPONDING_ERROR 0x04
#define BCI_LIN_SYNCH_BREAK_ERROR 0x05
#define BCI_LIN_INCONSISTENT_SYNCH_FIELD_ERROR 0x06
#define BCI_LIN_MORE_DATA_EXPECTED 0x07
#define BCI_LIN_TIME_OUT_AFTER_START_SYNCH_BREAK 0x08
#define BCI_LIN_TIME_OUT_AFTER_SYNCH_BREAK 0x09
#define BCI_LIN_TIME_OUT_AFTER_SYNCH_FIELD 0x0a
#define BCI_LIN_NOT_CONNECTED 0x0b
#define BCI_LIN_PARAMETER_ERROR 0x0c
#define BCI_LIN_BUS_NOT_FREE 0x0d
#define BCI_LIN_UNKNOWN_ERROR 0x0e
```

- **LIN_Status**

Error code for the status of communication with the LIN board

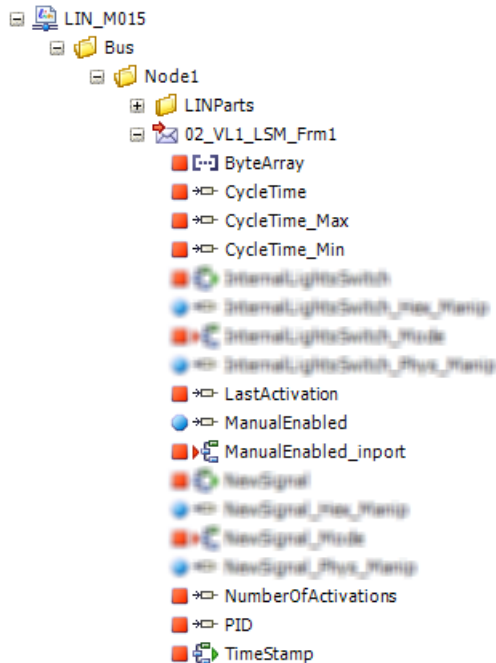
```
// status 0x00 means communication is running
#define BCI_LIN_STATUS_OVRRUN 0x01 /* data overrun occurred */
#define BCI_LIN_STATUS_ININIT 0x10 /* init mode active */
```

Nodes

Under the bus node, an individual node is displayed for every node of the LIN network. All inputs and outputs for controlling the LIN node or the elements contained therein are shown here.

Frames

The frames of a node are displayed under the node. The frames shown are the union of sets of all frames in all parts of the LIN node.



- **ByteArray**

The payload of the frame as a byte array.

- **CycleTime**

- **CycleTime_Max**

- **CycleTime_Min**

The average/maximum/minimum cycle time of the frame in μs . The cycle time is the time between two activations of the frame.

- **ManualEnabled**

- **ManualEnabled_inport**

A calibration variable and an inport with which the frame can be enabled/disabled during experiment runtime.

- **LastActivation**

- **NumberOfActivations**

Timestamp of the last activation of the frame in s and the number of activations.

A frame is said to be active if the frame is read by the LIN bus or written to the LIN bus. The processor clocks of the real-time PC are used to determine "LastActivation".

- **PID**

The protected identifier of the frame (is calculated from the "FrameId").

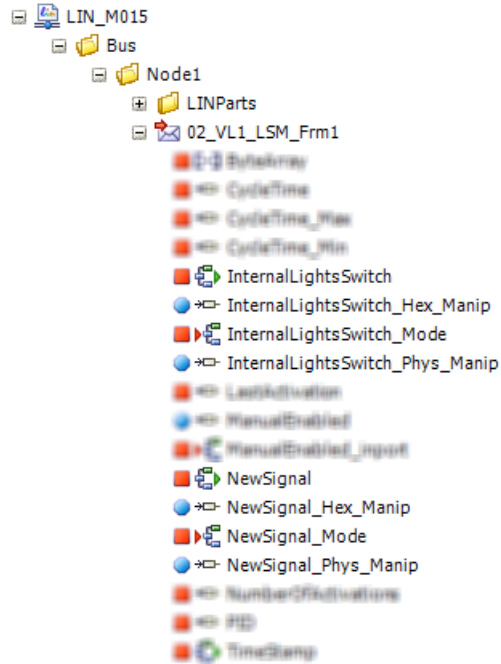
For receive frames, the "TimeStamp" signal is also created.

- **TimeStamp**

Timestamp of the last receipt of the frame in μ s. TimeStamp is determined by the LIN board.

Signals

The inputs and outputs for controlling the signals are created under a frame node.



- **<SignalName>**

Specifies the physical value of the signal and can represent either the value of the signal on the LIN bus or a substitute value entered manually.

- **<SignalName>_Mode**

Selection mode for the determination of the value of <SignalName>

```
typedef enum {
    MODEL = 0, /* Use Model value */
    CONFIG_PHYSICAL = 1, /* Use Phys_Manip */
    CONFIG_ELECTRICAL = 2, /* Use scaled value of Hex_Manip */
} eRedirectMode;
```

- **<SignalName>_Hex_Manip**

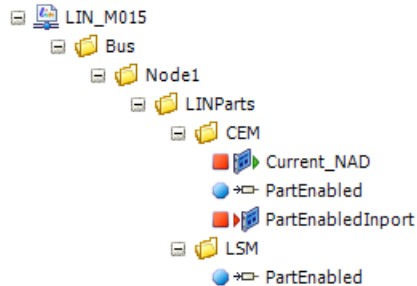
The electrical manual substitute value for <SignalName>

- **<SignalName>_Phys_Manip**

The physical manual substitute value for <SignalName>

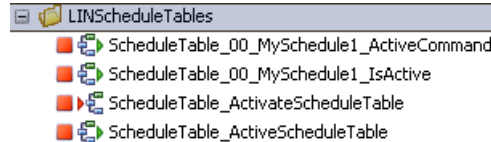
LINParts

An individual folder with the name of the part is displayed here for every LIN Part.



- **Current_NAD**
Current value of the "NAD" property for parts with an enabled "AutomaticDiagnosticRequest" property.
- **PartEnabled**
Calibration variable for enabling/disabling a part
- **PartEnabledInport**
Inport for the activation/deactivation of a part

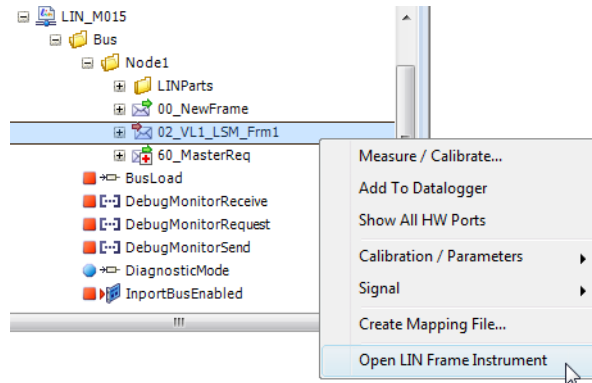
LINScheduleTables (Only for Controllers in Master Mode)



- **<ScheduleTableIndex>_<ScheduleTableName>_ActiveCommand**
Shows the position of the command currently being run of the schedule table.
- **<ScheduleTableIndex>_<ScheduleTableName>_IsActive**
Shows the current execution status of the selected schedule table (0 = inactive, 1 = active).
- **ActivateScheduleTable**
Entry of the "Schedule Table Index" of the schedule table to be activated.
- **ActiveScheduleTable**
Output of the "Schedule Table Index" of the schedule table currently active.

3.7.7 Instruments for LIN Frames

To create an instrument for displaying a message, right-click the message and select **Open LIN Frame Instrument**.



An instrument is created that shows the content of the selected message.

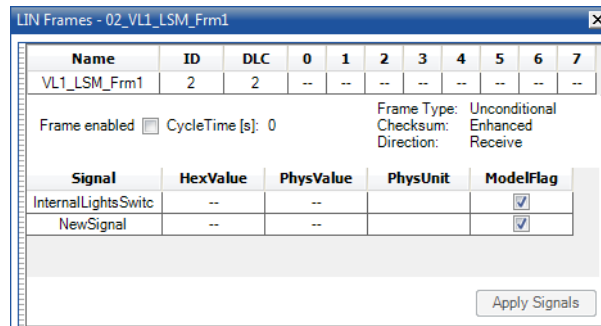


Fig. 3-20 Instrument for a LIN Frame

In a frame, either the Hex value or the physical value of a signal can be modified in the operating window (the other value can then no longer be edited).

For this purpose, only the "ModelFlag" option has to be disabled. This change becomes valid when **Apply Signals** is activated.

3.8 Network Module

The Network module provides configurations of residual bus simulations using different network types. It allows configurations of the bus type "CAN/CAN FD". The configuration is based on the Autosar standard.

Hardware Requirements

To be able to run Network bus simulations you need at least one CAN board.

Use CAN boards of the following type:

- CAN-IB600
- CAN-IB200
- XC16

Software Requirements

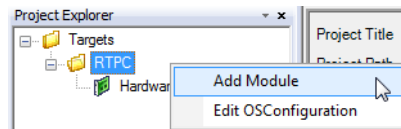
To be able to run Network modules you need an LCS_LCO_NIC licence.

3.8.1 Creating a Network Module

Use the module wizard to add a new Network module.

To add a Network module

- Create a new or open an existing LABCAR-IP project.
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.



- In the "Add Module Wizard", select **Add Network Module**.
- Click **Next**.
- In the next window, enter a module name.
- Click **Finish**.

In the "Project Explorer", a new module is added.

3.8.2 Maintaining a Network Module

On the Network modules, you can perform the following actions:

- "To edit a Network module" on page 167
- "To edit the `NetworkConfiguration.xml` file" on page 169
- "To update a Network module" on page 169
- "To remove a Network module" on page 170

To edit a Network module

- In the Project Explorer, select a Network module.

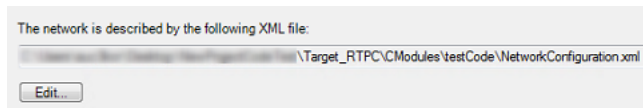


- Double-click the Network module.

or

- Right-click and select **Edit**.



The network editor opens in the editor panel. Your individual path to the `NetworkConfiguration.xml` file is shown.



- Click **Edit**.
An external editor opens.
- In the external editor edit the `NetworkConfiguration.xml` file.

Note

You find an explanation how to edit the .XML file below and in the `NetworkConfiguration.xml` file itself.

- Save the file in the external editor with **File** → **Save**.
 - Return to LABCAR-IP.
 - Click **Save** 
- or
- Click **Save All**  .
- The changes at the Network module are saved. The code is generated.

Note

If you edit the `NetworkConfiguration.xml` file in the external editor, LABCAR-IP recognizes this and enables the **Save** button and the **Save All** button in LABCAR-IP automatically.

To edit the `NetworkConfiguration.xml` file

Tags in the file have the following meaning:

<i>version</i>	Indicates the version of the Network Module.
<i>file name</i>	Points to the ARXML file location.
<i>ecu name</i>	ECU names in the ARXML file which you wish to simulate.
<i>hardware-access:</i>	Bus Type and its required configuration. If the user has not selected any values for the "hardware-access", default values would be taken into consideration. For CAN-Board ID and Controller ID is to be defined. Default Board ID is "0" and Controller ID is "0".
<i>clusters</i>	CAN Cluster Names in the ARXML file which you wish to simulate.
<i>code-gen</i>	Generates code either for simulating the ECU or for monitoring purpose. For monitoring set the simulateEcu to "false". The default code is always generated for simulation.

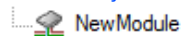
Complete example:

```
<networkmodule version="1">
  <autosarfiles>
    <file name="C:\test.arxml" />
    <ecus>
      <ecu name="EIS" />
      <ecu name="LRR" />
    </ecus>
    <hardware-access type="CAN" boardid="0" controllerid="0" />
    <bus type="CAN" clusters="RADAR_F" />
    <code-gen simulateEcu="true" />
  </autosarfiles>
</networkmodule>
```

To update a Network module

If a referenced Autosar file (ARXML file) changes, it is necessary to update the network configuration in LABCAR-IP.

- [In the Project Explorer, select a Network module.](#)




- Right-click and select **Update**.
The Network module is updated. The new code is generated.

Note

If errors occur during the update process, it is shown in the "Application Log" window.

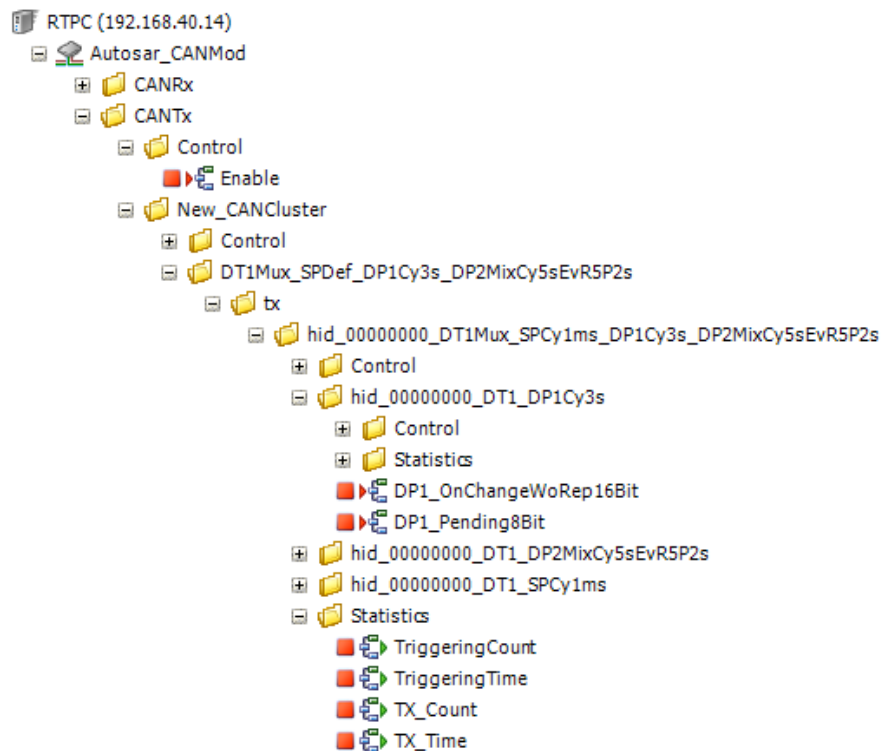
To remove a Network module

- In the Project Explorer, select a Network module.
 NewModule
- Right-click and select **Remove**.

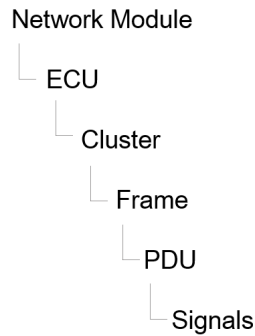
3.8.3 Rest Bus Simulation of a Network Module

Network Module Hierarchy in ETAS Experiment Environment

The following figure shows the hierarchy of components in a Network module shown in the ETAS Experiment Environment:



The Network modules components are arranged as shown in the following figure:



Special Ports in a Network Module

Trigger:

Triggers have typical behaviors. If there is any value change with respect to a 'Trigger' inport, it will cause a transmission of the respective PDU. By default, the value of a Trigger is '0'.

- If the user changes the value of a Trigger to '1', it will cause a transmission once.
- If the user changes the value of a Trigger from '1' to '2', it will cause a transmission once.
- If the user changes the value of a Trigger from '2' to '0', it will cause a transmission once.

Enable:

This inport is used to enable or disable the component. If it is set to "1", then the component is enabled. If it is set to "0", then the component is disabled. The default value for this inport at ECU level will be "0". In all other components (Frame\PDU), the default value is "1".

Statistics:

This section provides the details about the statistics of the communication. This section will be available at Module\ECU\PDU levels.

PDU Level Statistics:

This section provides information to the user as to how many triggers (cyclic\spontaneous\mixed) have occurred for PDU and how many transmissions and receptions occurred for PDU.

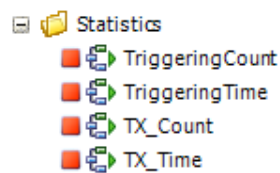


Fig. 3-21 PDU Tx Level Statistics

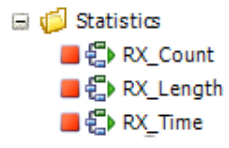


Fig. 3-22 PDU Rx Level Statistics

3.9 NIF Modules (Network Integration FlexRay)

LABCAR-NIF V5.4.4 (Network Integration FlexRay) is a LABCAR-NIF V5.4.4 module type in LABCAR-OPERATOR V5.4.4. It enables simple 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. LABCAR-NIF V5.4.4 then automatically creates source code for the residual bus to be simulated in the form of a NIF module. User-specific code can be added to this.

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

Note

LABCAR-NIF V5.4.4 uses the "StringTemplate.NET" and "ANTLR" libraries – please take the license conditions detailed in "Appendix" on page 367 into consideration.

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 NIF 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

Software Requirements

A license for the "EB tresos Busmirror" software (4.6.x or higher) is required from Elektrobit to generate the files to be imported for residual bus simulation.

Note

Due to the conversion of the ETAS RTPC operating system to 64 bits, versions < 4.6 are no longer supported. When migrating older LABCAR-OPERATOR projects to the current version, NIF modules based on EB tresos bus mirror versions no longer supported are not changed. These modules can then no longer be edited or run on the Real-Time PC, and must be replaced by new NIF modules.

3.9.1 Creating the Necessary Files with EB tresos Busmirror

For the integration of the FlexRay bus configuration in LABCAR-OPERATOR, 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

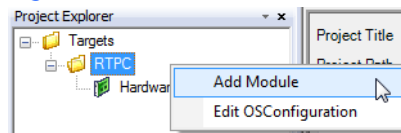
For details of how to create these files, refer to the EB tresos Busmirror documentation.

3.9.2 Creating a FlexRay Module

Once the data model and the executable binary files have been generated with EB tresos Busmirror, they can be integrated into the LABCAR-OPERATOR project. Use the module wizard to add a new NIF module.

To add a FlexRay module

- Create a new or open an existing LABCAR-IP project (see 3.1.3 on page 27).
- In the "Project Explorer", select a target folder.
- Right-click and select **Add Module**.

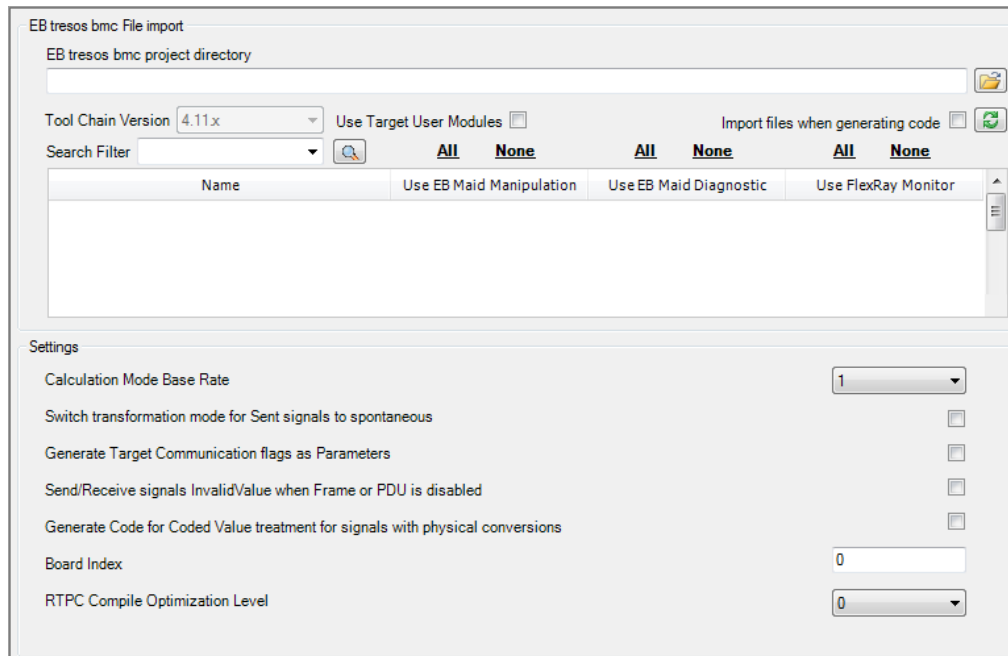


- In the "Add Module Wizard", select **Add FlexRay Module**.
- Click **Next**.
- Enter a module name.
- Click **Finish**.

In the "Project Explorer", a new module is added.

3.9.3 Linking to the EB tresos bmc Tool Suite

The window of an NIF 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 NIF module.
- Save the NIF module.

Synchronizing EB tresos bmc Files and Generating NIF 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 NIF module is generated. Thanks to a synchronization mechanism, these external files can be re-imported into the NIF 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 NIF module, it is not integrated into the NIF 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 NIF 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.

3.9.4 **Module Configuration**

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

Adapting the Clock Time of the NIF 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 NIF 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 NIF module and the board. The calls for exchanging the signals of an individual PDU also modulated with the FlexRay Cycle Repetition of the corresponding Frame-Triggerings 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 μ s)

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 NIF 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 output.

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 a Real-Time PC

Several independent LABCAR-NIF modules can also be generated – the relevant EB5100 or EB5200 (within the directory structure of the LABCAR-OPERATOR-project) is identified via the board index.

- Enter the index under "Board Index".

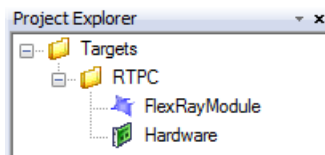
Compile Optimization Level

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 NIF module.

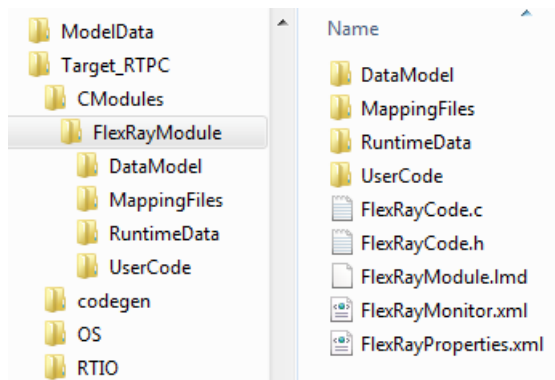
3.9.5 Linking the NIF Module to the LABCAR-OPERATOR Project

To save the module configuration

- Select **File** → **Save**.
The module configuration is saved and the source code of the module generated. The module is then integrated into the LABCAR-OPERATOR project and shown in the Project Explorer.



After successful code generation, a directory with the following files and directories is created for the NIF module.



To update the OS Configuration and Connection Manager

- Toggle to the "OS Configuration" tab.
- Click **Update Processes**.
The tasks and processes created for the NIF module are displayed.
- Toggle to the "Connection Manager" tab.
- Click **Update Ports**.
All inputs (send payloads) and outputs (receive payloads) of the data model are displayed.

To compile the module

- Select **Project** → **Build**
- or
- Click the **Build LABCAR Project** icon.



3.9.6 User-Defined Code and Adaptations

As an extension to the NIF 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 NIF 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 NIF module, this is not integrated into the NIF 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 NIF 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 NIF 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 NIF 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 NIF 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.

Outputs:

Outputs are defined in the following file:

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

An output is defined by one line:

```
TPortObj <module_name>_UserCode_output_<name>;
```

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

Example:

The following example shows how user-specific code can be used to add extra functionality to the NIF 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 in LABCAR-IP, the new measurement variable is available in the experiment environment LABCAR-EE.

Note

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

3.9.7 [Linking to the ETAS Bus Communication Monitor \(BCM\)](#)

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

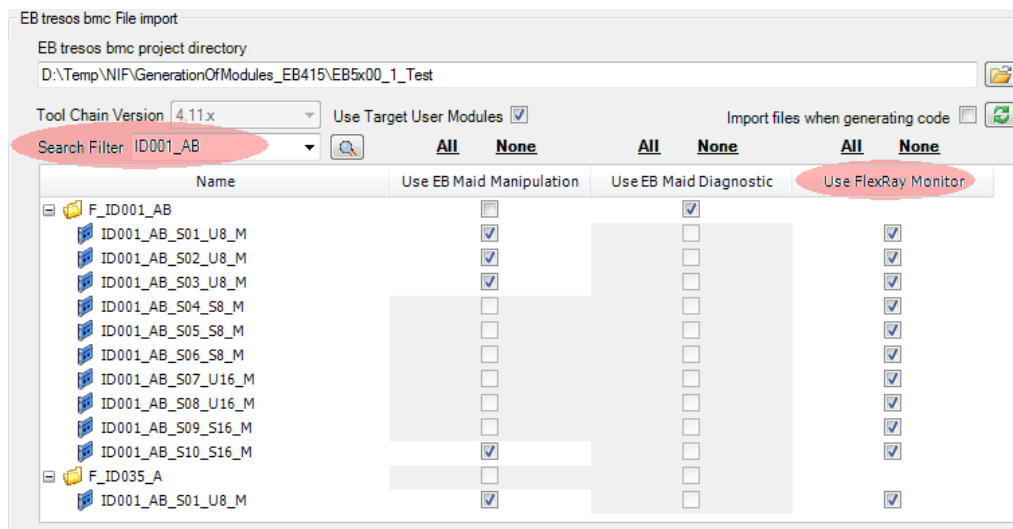
To optimize performance, the selection of signals for code generation to be taken into consideration for manipulation by the ETAS Bus Communication Monitor can be limited on the user interface.

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.



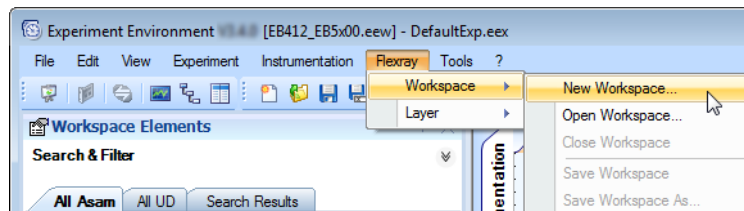
Note
The Bus Communication Monitor only supports scalar signals and signals with a length of max. 32 bits. ByteArray signals or signals longer than 32 bits are not supported!

- 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.

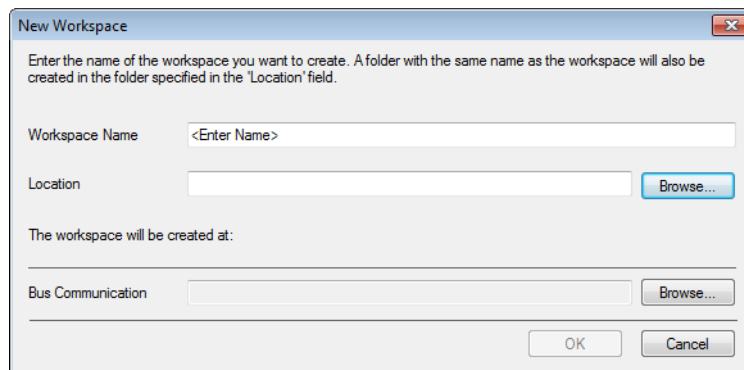
- Using the "All" and "None" options, all signals which satisfy the search criterion can be enabled or disabled respectively.

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**.



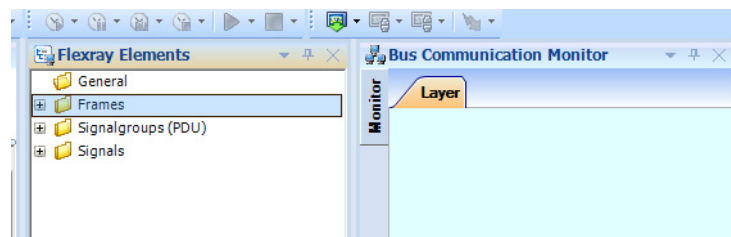
- Select a name and a location for the new workspace.
- Under "Bus Communication", select the file mentioned above `FlexRayMonitor.xml`.



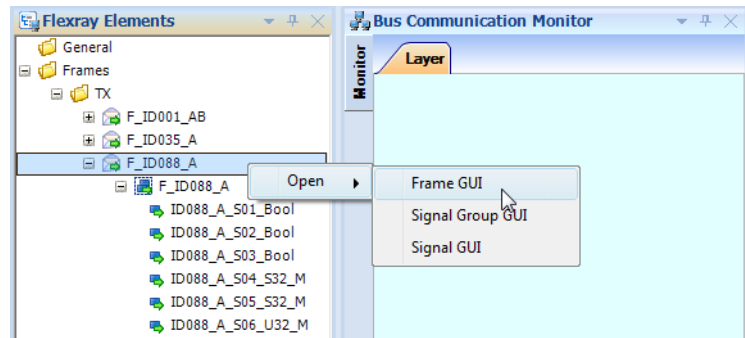
or

- 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.

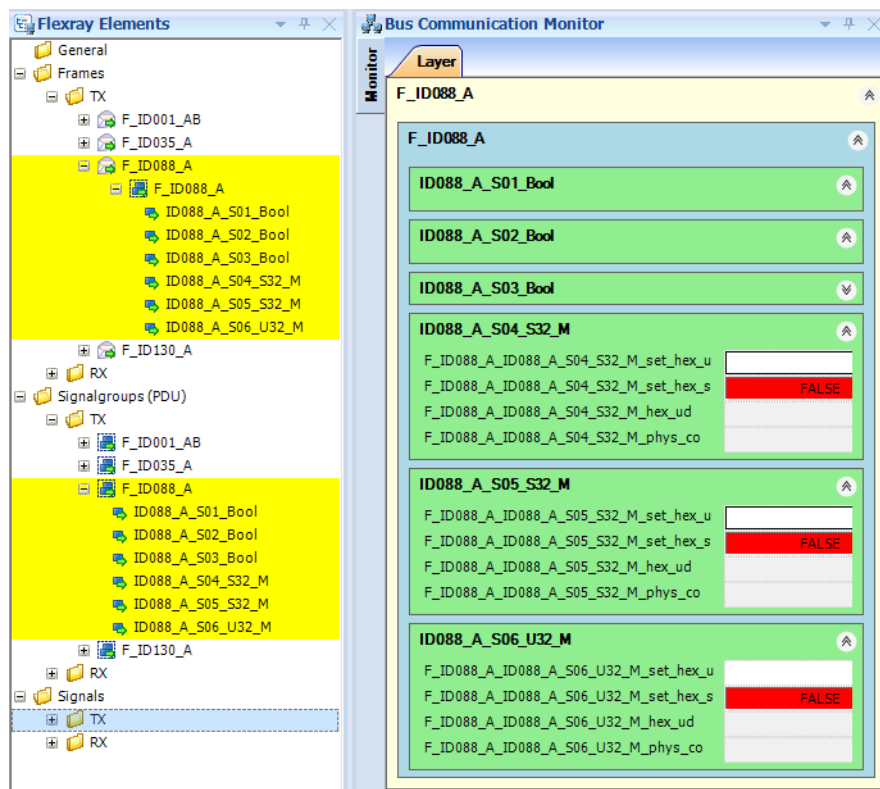


- In Flexray Elements, select the frame to which the signals shown in the Bus Communication Monitor belong.
- Right-click and select **Open** → **Frame GUI**.



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.



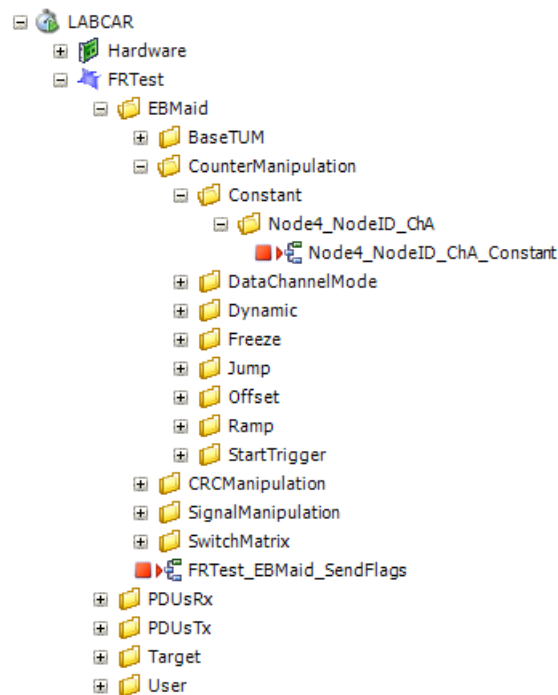
The signals selected for display in the Bus Communication Monitor are highlighted in yellow in Work-space Elements.

3.9.8 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 NIF 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 NIF 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

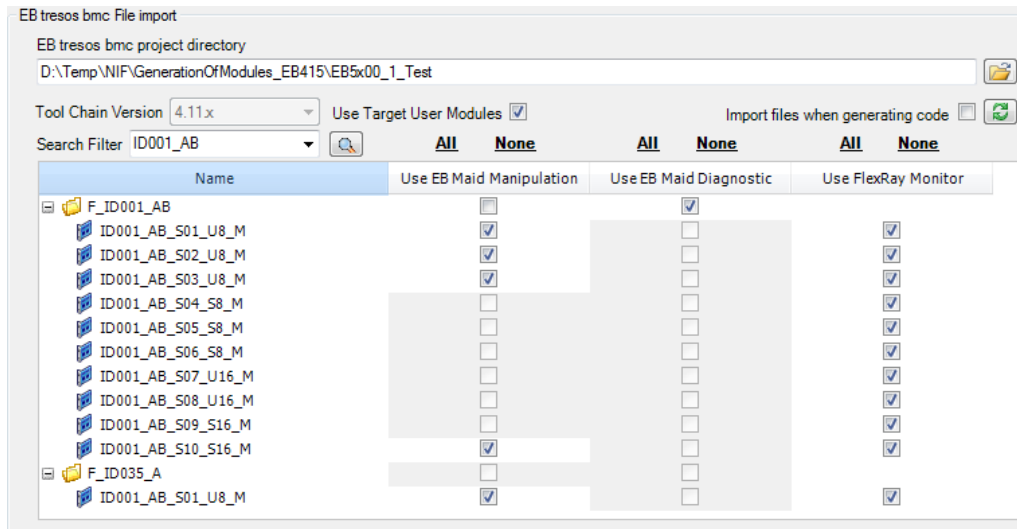
TUM support can be enabled and disabled for individual signals and PDUs of the EB tresos BMC data model. This increases performance and clarity.

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 NIF 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

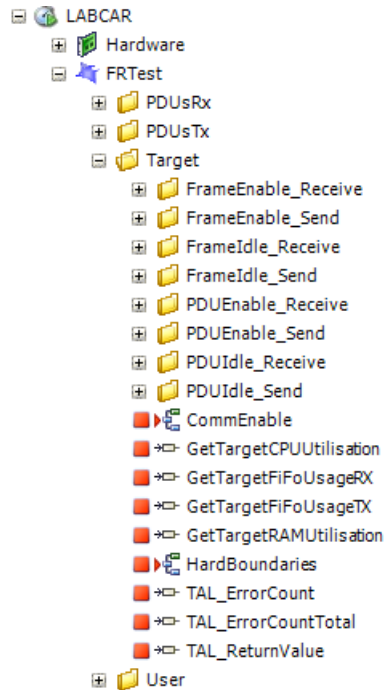
A TUM is not run cyclically. All parameters the TUM needs for execution are transferred in an individual function call. Then TUM execution begins until it is stopped by another call. To start the execution of a function, the following inport can be used:

```
\<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.

3.9.9 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 NIF 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 the bus communication. If the CommEnable parameter is set to 0 at simulation start, the FlexRay controller does not start the bus communication. To start the bus communication, activate the CommEnable parameter.
- `<module_name>/Target/CommEnableAfterExit`
To avoid that the FlexRay controller stops the bus communication at simulation stop, activate the CommEnableAfterExit parameter.
- `<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 transmission of frames between the NIF module and 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 is sent.
 - Value = 0: The frame 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.
 - Value = -1: A NULL frame is sent.
- `<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]_[CycleRepetition] 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 NIF 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/Frameldle_Receive/Send

There are various signals available under this node to enable and disable the transmission of signals between the LABCAR-OPERATOR project and NIF module.

- `<module_name>/Target/Frameldle_Send/Frameldle_Send_<frame_name>`
Enables or disables the accepting of the signal values to be sent from the LABCAR-OPERATOR 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/Frameldle_Receive/Frameldle_Receive_<frame_name>*
Enables or disables the transmission of the signal values received to the LABCAR-OPERATOR 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 LABCAR-OPERATOR project and NIF module.

- *<module_name>/Target/PDUIdle_Send/PDUIdle_Send_<PDU_name>*
Enables or disables the accepting of the signal values to be sent from the LABCAR-OPERATOR 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 LABCAR-OPERATOR 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.

3.9.10 *Creating a NIF Module with the Automation Server*

An NIF module can also be created using API functions of LABCAR-OPERATOR. 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 LABCAR-OPERATOR in the case of automation.

There is a whole range of Set/Get methods with which the configuration of an NIF 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 FiL Modules

This chapter describes the creation of FiL modules.

Function-in-the-Loop (FiL)

A traditional HiL system (Fig. 3-23) consists of an ECU and a simulation model connected to each other in a closed-loop control circuit. The ECU outputs are connected to a board to acquire signals for the simulation PC. A real-time-capable simulation model runs on the simulation PC. It calculates the relevant output values based on the input values measured. In turn, these are converted to electric signals and connected to the ECU inputs.

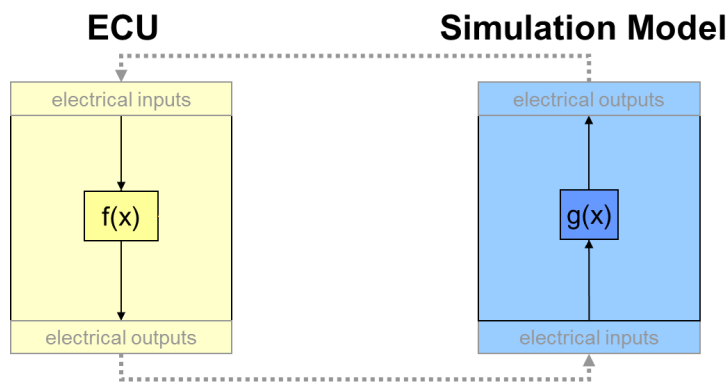


Fig. 3-23 Structure of an HiL System

In a FiL system (Fig. 3-24), some (or all) of these electric connections are bypassed and the ECU memory is accessed directly – the ECU must be equipped with an ETK or XETK for this purpose¹.

This means that the input and output stages and the hardware connections to and from the simulation model are bypassed.

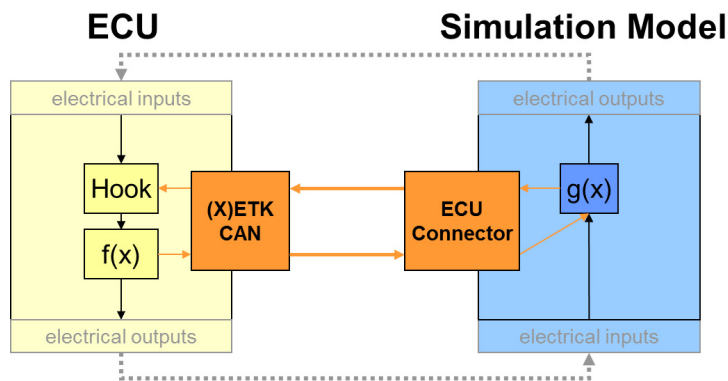


Fig. 3-24 FiL System

To create an FiL system, you need the following:

- The ECU must be equipped with an ETK or XETK.
- ECU outputs that are measured by the ETK/XETK must be connected to the simulation model.

¹. Access via XCPonCAN is also possible – its speed is, however, too low for a real FiL system.

- Inputs for the ECU functions that are to be stimulated by the model must have a software hook. A hook is a switch with which the input of an ECU function can be separated from the electrical input or from the previous ECU function and connected to the model using ETK/XETK.
- Sound knowledge of the internal function blocks of the ECU.
- To enable correct stimulation of the function inputs, it may be necessary to disable the diagnostic functionality of the ECU.

The FiL system is configured in LABCAR-IP; communication with the FiL module takes place in the experiment environment ETAS EE.

3.10.1 Configuration in LABCAR-IP

The following figure shows the start page of the FiL Wizard.

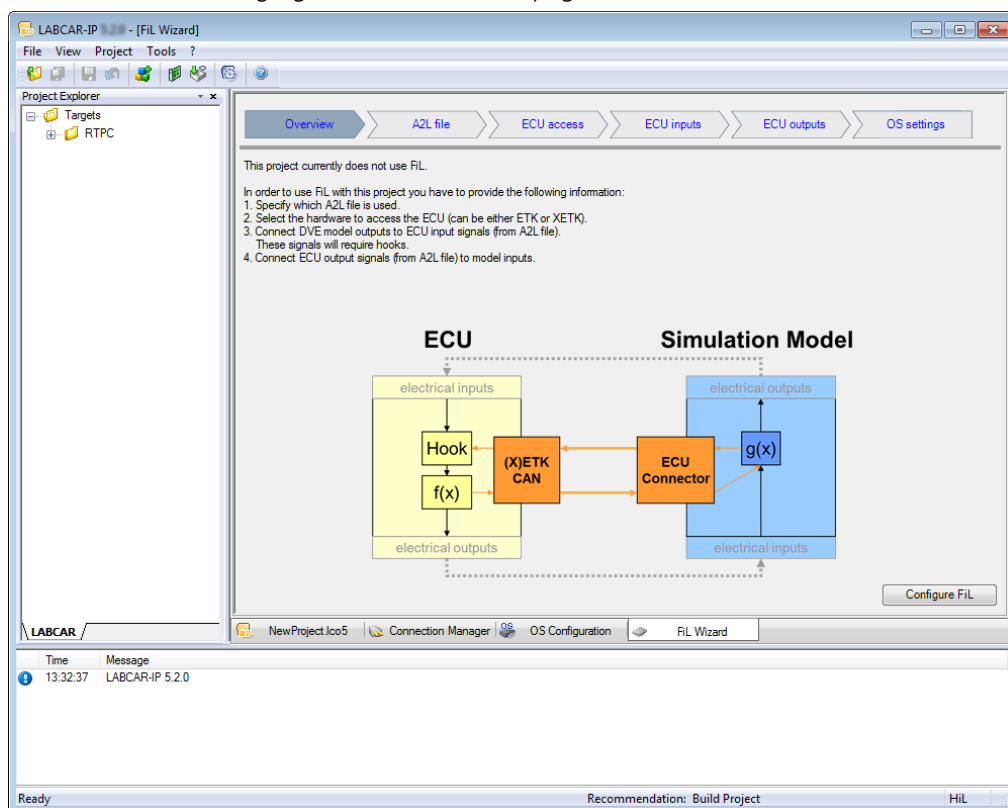


Fig. 3-25 The Start Page of the FiL Wizard

This is where you will find a short overview of the steps to be executed.

1. Definition of the ECU file (see "Selecting an A2L File" on page 199)
2. Definition of the connection from the Real-Time PC to the ECU (see "Defining Hardware for ECU Access" on page 200)
3. Definition of the ECU values which are to be stimulated by the model (see "Selecting ECU Variables for the Stimulation" on page 202).
4. Definition of the ECU values which are to be connected to inputs of the simulation model (see "Selecting ECU Outputs for Connecting to the Model" on page 204).

5. Assignment of the ECU rasters to tasks in the OS configuration (see "OS Settings" on page 205)

This order is reflected in the order of the tabs:



State and Error Display

Each tab (icon) also indicates the state of the relevant step (OK, warning or error). Detailed warning and error messages during the configuration process are issued in the "Messages" window and written to the log file.

3.10.2 Selecting an A2L File

The ECU description file must be defined at the beginning of the configuration.

To select the ECU description file (A2L file)

- Select the "A2L file" tab.
- Click **Browse** and select the A2L file.

The file is loaded and its contents (parameters, measure values and available hooks) displayed.

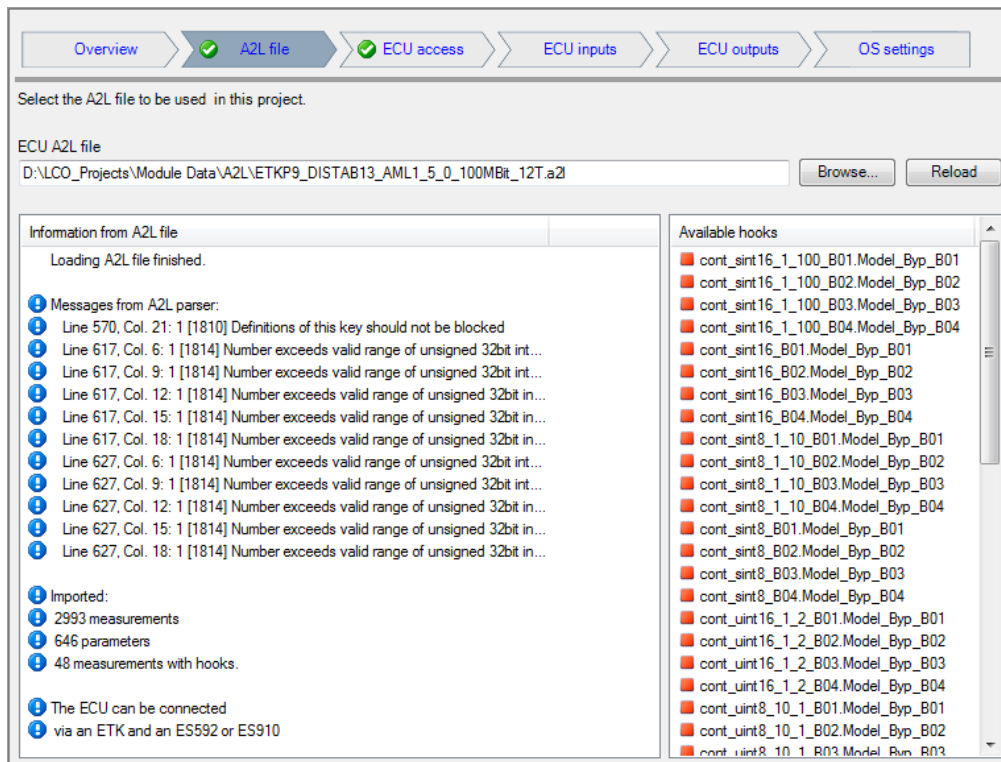


Fig. 3-26 Selecting the A2L File

The following information is displayed once the file has been read:

- **Information from A2L file**

A range of parser messages is issued here.

If the file could not be loaded, a list of all errors is displayed in the "Messages" window.

- **Available hooks**

All signals that have had hooks defined for them (in the A2L file) are listed here.

If the A2L file has been changed (but name and directory have stayed the same), the file can be read in again with **Reload**.

3.10.3 Defining Hardware for ECU Access

The next step is to define the hardware via which ECU access is to take place:

- **XETK**

For an ECU with XETK – the XETK is connected directly to the Real-Time PC via Ethernet.

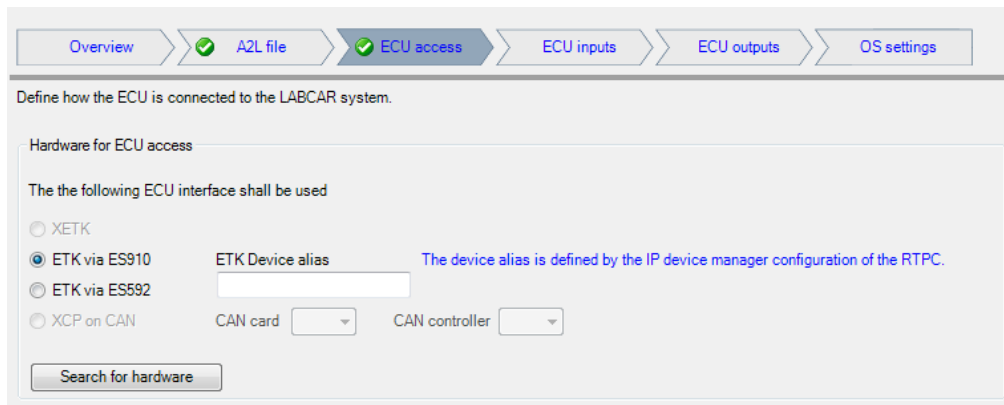
- **ETK**

For an ECU with ETK – the ETK is connected to an ES592 or ES910 and these are connected to the Real-Time PC via Ethernet.

- **XCP on CAN**

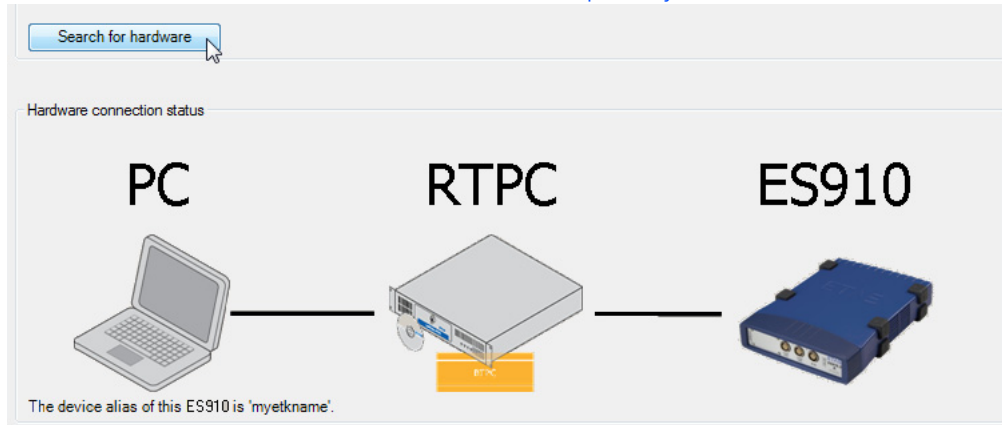
To select hardware

- Go to the "ECU access" tab.
- Select the hardware for ECU access.



- When accessing via ES910 or ES592 enter the "ETK Device Alias" (see "ETK Device Alias" on page 201).

- To check whether the selected hardware is connected, click **Search for hardware**.
If the selected hardware is available, it is shown by an icon (and possibly the device alias).



Note

The hardware search is limited to the selection made in the "Hardware for ECU access" field. Other connected devices are not displayed!

ETK Device Alias

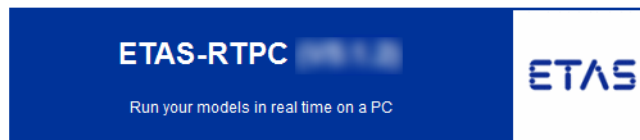
The "ETK Device Alias" replaces the "hard wiring" with the IP address of the connected ES592 or ES910 and thus facilitates porting projects.

Note

When accessing via ETK, an "ETK Device Alias" has to be specified here (and be defined previously in ETAS RTPC)!

To assign a device alias in ETAS RTPC

- Clicking the link opens a browser instance and in it, the page "IP Device Manager" of the ETAS RTPC web interface.



[Main Page >> System Info >> IP Device Manager](#)

Search for Hardware			
Eth	Device	Serial	Alias
eth1 (192.168.40.20)	ES592	9900026	myES592
[not connected]	ES910	101060	myES910

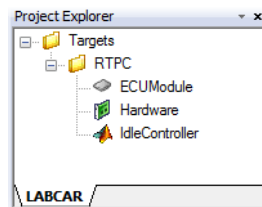
- Click **Search for Hardware** to search for Ethernet connections which were configured for real-time communication ("rtudp_n").

After a successful scan, the following information is displayed:

- **Eth**
The Ethernet adapter of the Real-Time PC where the hardware is connected.
- **Device**
The device type
- **Serial**
The serial number of the device
- Enter an identifier under **Alias** (C-conform, max. 32 characters).
- Select **Reset** to reset changes made.
- To accept the entry made, select **Apply Changes**.
- Exit the web interface and return to LABCAR-IP.

To save the project

- Save the configuration with **File** → **Save**.
- The FiL module is saved and displayed in the Project Explorer ("ECUModule").



3.10.4 Selecting ECU Variables for the Stimulation

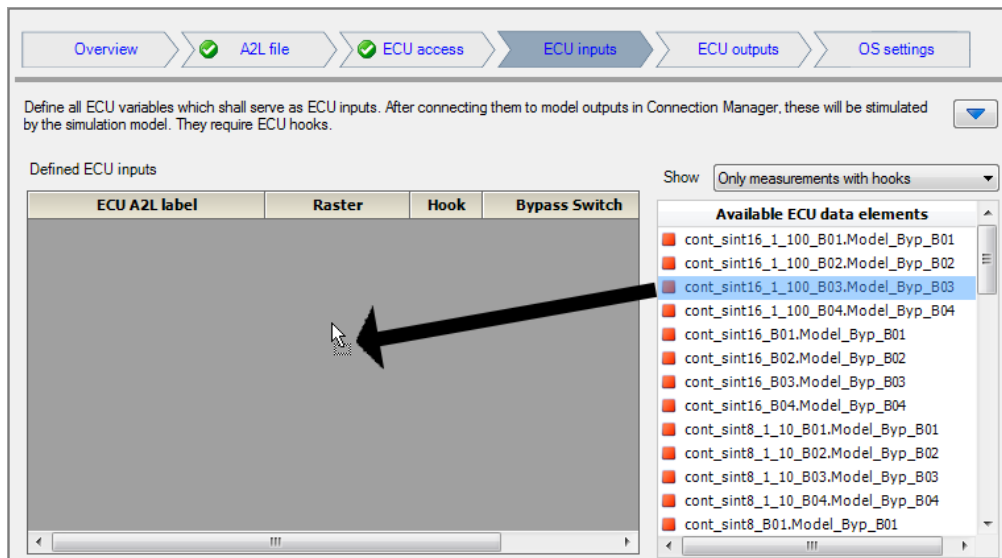
In this step, you define ECU variables to act as inputs. These are stimulated by outputs of the simulation model and require ECU hooks for this purpose.

These hooks separate the relevant function block from the outputs of the block before (or even from the ECU input) and connect it to the input values provided by the connected ETK/XETK. The hooks are thus the switches with which the external bypass function is enabled/disabled.

To select inputs

- Select the "ECU inputs" tab.
- Select ("Show") **Only measurements with hooks** as filter.
- Select a measure value from the "Available ECU data elements" list.

- Move it to the list on its left by Drag & Drop.



The measure value is added to the list.

Defined ECU inputs			
ECU A2L label	Raster	Hook	Bypass Switch
cont_sint16_1_100_B03.Model_By		Available	n/a

To delete a measure value from the list

- To remove a measure value from the list, right-click it and select **Delete**.

Meaning/Function of the Individual Columns

- **ECU A2L label**
The label of the data element
- **Raster**
This is where you select the ECU raster in which the data element is acquired.
- **Hook**
This displays whether a hook is available for this data element.
- **Bypass Switch**
A parameter with which the Bypass can be enabled/disabled. This can be selected from the "Available ECU data elements" list (filter: "Characteristics") by Drag & Drop.

- **Initial Value**

Initial value for the parameter in the "Bypass Switch" column.
In the experiment environment, there is an instrument for operating communication that has a function (see "[Apply bypass switch settings](#)" on page 209) with which all bypasses can be enabled with a default initial value.

The data elements selected here are available as inputs after saving in the Connection Manager (if not, click [Update Ports](#) in the Connection Manager).

3.10.5 [Selecting ECU Outputs for Connecting to the Model](#)

In this step, you select data elements of the ECU that later can be connected for measuring with model inputs.

The following figure shows the "ECU outputs" tab.

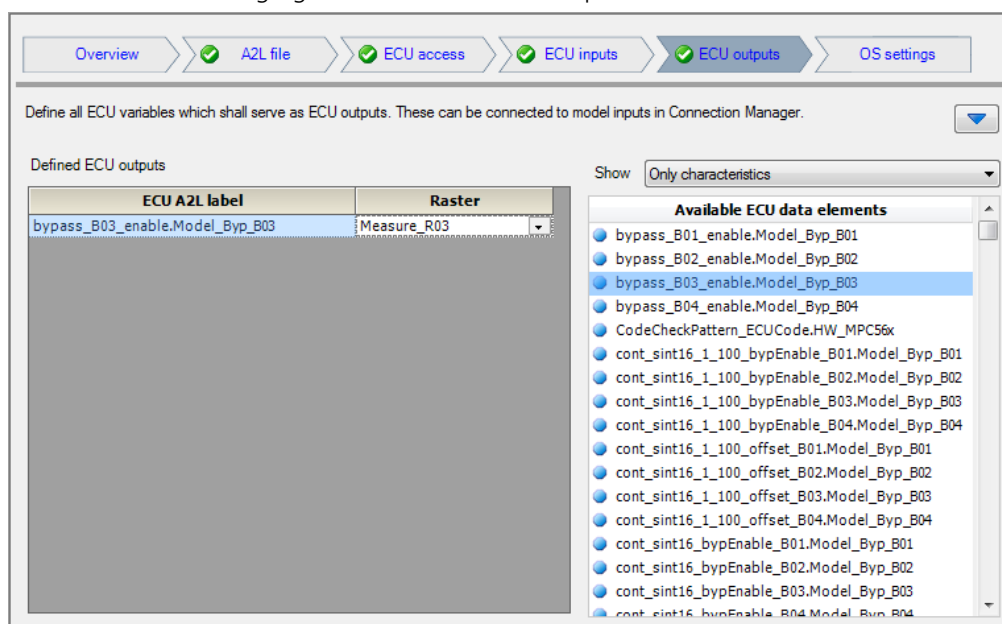


Fig. 3-27 Making ECU Outputs Available for Connecting with Model Inputs
To select a data element

- Select a data element in the list on the right and drag it to the list on its left.
The selected element is added to the "ECU A2L label" column.
- Select a raster from the "Raster" column.

The data elements selected here are available as outputs after saving in the Connection Manager (if not, click [Update Ports](#) in the Connection Manager).

3.10.6 OS Settings

In this step, you can link ECU rasters to Real-Time PC tasks. This enables a close coupling of the simulation to the ECU raster.

The following figure shows the "OS settings" tab.

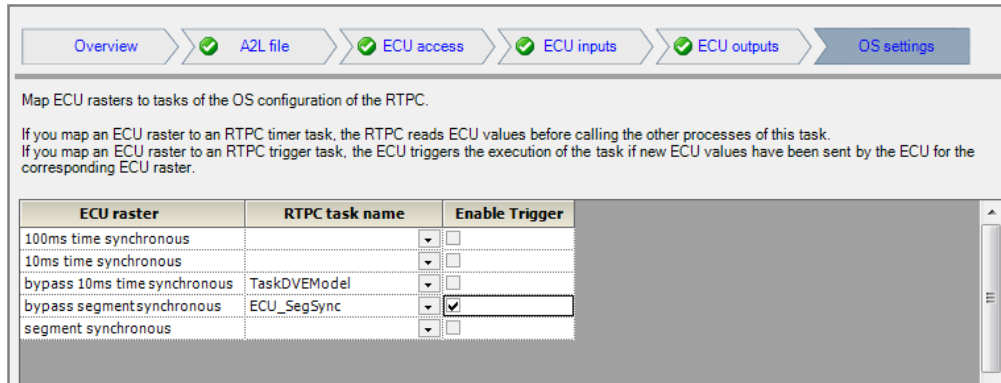
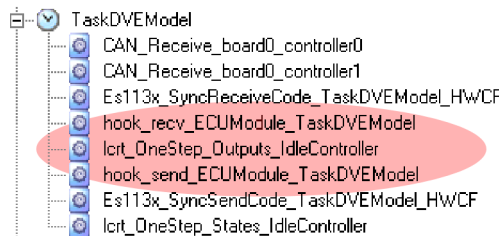


Fig. 3-28 Assignment of Tasks to ECU Rasters

This applies to both timer tasks and trigger tasks.

Timer Task

If an ECU raster is mapped to the timer task of the Real-Time PC, in which the model is calculated, prior to the calculation step dt data is read from the ECU (hook_rcv_ECUModule_[task name]), then calculation takes place (lcr_OneStep_Outputs...) and then it is written to the ECU (hook_send_ECUModule_[task name]),.



Trigger Task

Mapping an ECU raster to a trigger task enables a close coupling: The ECU can be made the timer of the simulation – the Real-Time PC starts the model calculation only after having received data from the ECU.

If the ECU is to become a clock generator, you must assign a trigger task and activate the "Enable Trigger" option (Fig. 3-28 on page 205). If this option is not selected, the Real-Time PC task is not triggered although the ECU sends data.

This is why the "Enable Trigger" option is activated in most cases – it is, however, sensible not to activate it in the following case: If several ECU rasters were assigned to the same trigger task, the "Enable Trigger" option should only be activated for one raster (see figure).

ECU raster	RTPC task name	Enable Trigger
100ms time synchronous	ECU_Sync	<input checked="" type="checkbox"/>
10ms time synchronous	ECU_Sync	<input type="checkbox"/>
bypass 10ms time synchronous	TaskDVEModel	<input type="checkbox"/>
bypass segmentsynchronous	ECU_SegSync	<input checked="" type="checkbox"/>
segment synchronous		<input type="checkbox"/>

Otherwise, the ECU would trigger the trigger task ("ECU_Sync" in the figure) several times during a period dT leading to a time response difficult to comprehend (several, non-equidistant read cycles of the ECU).

The "Auto Trigger" Option

If there is no data from the ECU, the simulation comes to a standstill. To avoid this, there is the option "Auto Trigger" for tasks of the type "Trigger".

If you select the corresponding trigger task in the "OS Configuration" tab, you can enable the option "Auto Trigger" at the bottom of the window.

Task Settings

Task ID 9 Delay 0 s Priority 1 CPU-Core 1 Exclusive Core Usage

Disable Task Stop On Floating Point Exception

Auto Trigger Period 0.01 s Event Timeout 0.012 s

OS Settings

Max. Number Of Priorities 40 Automatic Process Assignment Enable OS Monitoring

Example: If, after 12 ms ("Event Timeout"), no data has arrived from the ECU, calculation continues at intervals of 10 ms ("Period") until data starts coming from the ECU again.

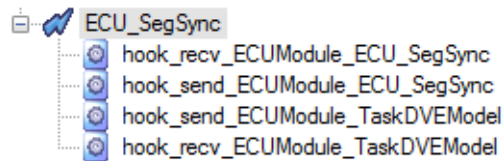
The assignment of an Real-Time PC task in the "RTPC Task Name" column (see Fig. 3-28 on page 205) adds the processes

"hook_rcv_ECUModule_[Taskname]"

and

"hook_send_ECUModule_[Taskname]"

to the respective task (tab "OS Configuration" in the main window of LABCAR-IP).



Note

At the end of the configuration, check whether the order of processes in the Timer/Trigger tasks used for ECU communication is correct.

3.10.7 Creating Connections in the Connection Manager

Once you have selected the data elements for stimulation and acquisition and have mapped ECU rasters to tasks, configuration of the FiL module is complete.

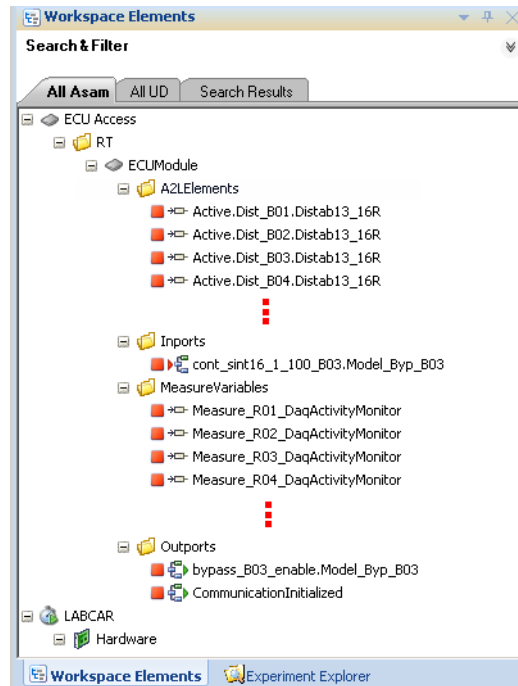
To be able to work with the module, its inputs and outputs must then be connected to the inputs and outputs of the DVE model in the Connection Manager.

The "CommunicationInitialized" Output

In addition to the outputs you have configured, there is also the "CommunicationInitialized" output in the Connection Manager. It is a "Boolean" value whose transition from 0 → 1 signals that the ECU is ready to communicate.

3.10.8 Working in ETAS EE

Once the project has been rebuilt, it can be opened in the experiment environment. The following figure shows the representation of the FiL module in the Workspace Elements.



The module has the following subfolders:

- **A2LElements**
This folder contains all A2L labels (measure values and parameters) of the ECU description file.
- **Inports**
The inputs of the module defined during configuration.
- **MeasureVariables**
One variable per ECU raster that is incremented when this raster is triggered by the ECU.
- **Outports**
The outputs of the module defined during configuration. This folder also contains the output "CommunicationInitialized" (see "The "CommunicationInitialized" Output" on page 207).

3.10.9 The "RT ECU Access" Instrument

Measure values and parameters can be displayed and modified using the standard instruments. There is a special "RT ECU Access" instrument for controlling the FiL function.

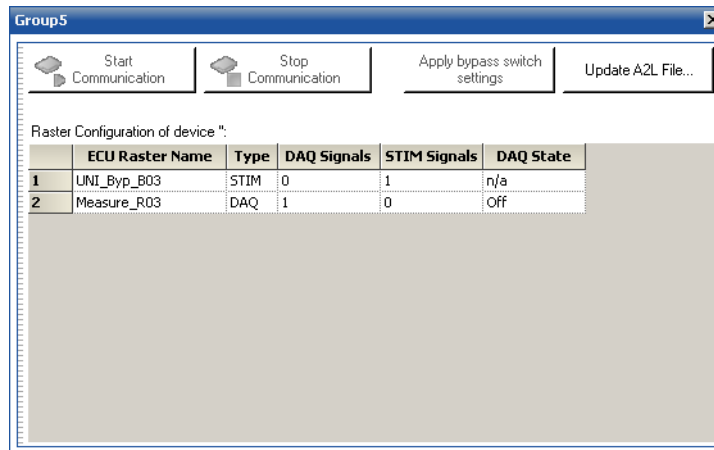


Fig. 3-29 Instrument for Controlling the FiL Function

The upper part of the instrument contains buttons for controlling communication:

- **Start Communication / Stop Communication**
Is used to start and stop communication of the FiL module with the ECU.
- **Apply bypass switch settings**
Is used to set the default values for the bypass switches (see "**Initial Value**" on page 204).
- **Update A2L File**
If you use a new A2L file due to a new software status, this function helps you carry out an update without opening the configuration in LABCAR-IP.

Note

The update only works if the new file contains all rasters and ECU labels of the module!

The configuration of the ECU rasters is displayed in the bottom part of the instrument. The meaning of the individual columns is as follows:

- **ECU Raster name**
The name of the raster
- **Type**
Type: "DAQ" or "STIM"
- **DAQ Signals / STIM Signals**
Number of the measured/stimulated signals in the relevant task
- **DAQ State**
Displays whether even values are delivered by the ECU for the DAQ raster. Possible values are "n/a", "Off" and "Running".

3.11 Real-Time Plugins

This chapter contains information on creating real-time plugins, how to integrate them into a LABCAR-OPERATOR project and how to work with them in the experiment environment ETAS EE.

Real-Time Plugins

A real-time plugin is a dynamically loadable program library for the Real-Time PC whose functions can be integrated into the real-time operating system, and executed, during runtime.

The RT-Plugin Builder is a tool for creating a real-time plugin project and managing the associated source code. It also controls the compiling and linking of the plugin on the Real-Time PC and generates a packet file for the real-time plugin.

ETAS RTPC

The plugin source code is compiled and linked to a dynamically reloadable program library (Shared Object Library) on the Real-Time PC that provides the necessary functionality and infrastructure required for real-time plugins.

This includes in particular

- loading and unloading real-time plugins
- generating hook processes
- linking functions (of the plugin) to hook processes
- resolving data element identifiers in memory addresses
- invoking hook processes in a real-time context

Configuring the Operating System

Hooks are added to tasks of the real-time operating system in the "OS Configuration" tab in LABCAR-IP.

Experiment Environment (ETAS EE)

Real-time plugin packet files can be added, enabled, disabled and removed in the experiment environment.

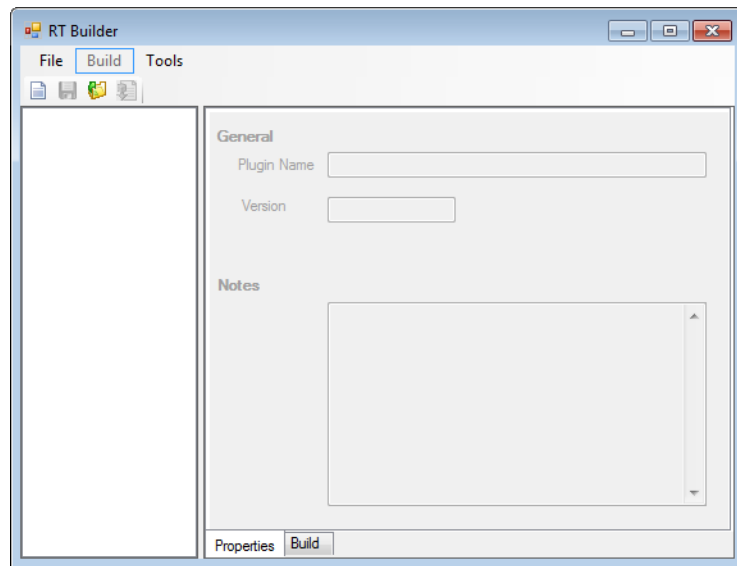
3.11.1 The RT-Plugin Builder

The RT-Plugin Builder is an application used to create real-time plugins. The RT-Plugin Builder generates a directory structure and files with function roots for the necessary callback functions of the plugin.

To create real-time plugin projects

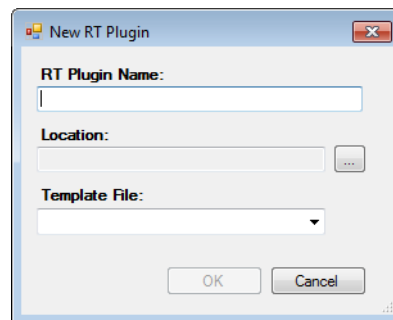
- In the Windows Start menu, select **Programs** → **ETAS** → **LABCAR-OPERATOR X.Y** → **RT-Plugin Builder**.

The RT-Plugin Builder opens.



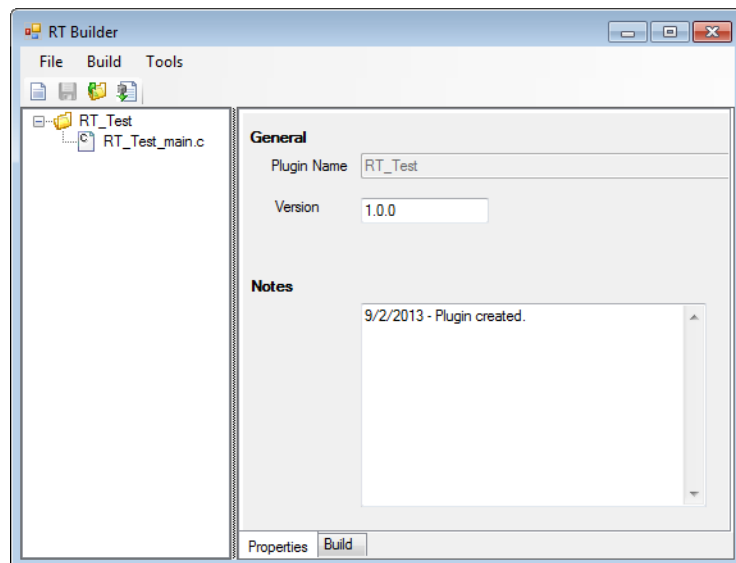
- Select **File** → **New**.

The "New RT Plugin" window opens.



- Enter the name of the RT plugin and the directory in which the project files are to be saved.
- Select a template file.

- Click **OK**.
The project is created.

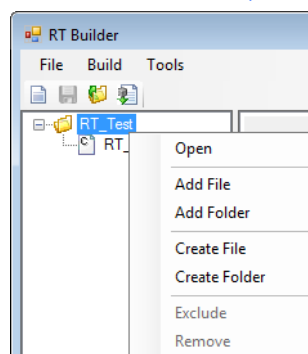


- If necessary, change the version number ("Version").
- Save the project using **File** → **Save**.

To manage real-time plugin source code

All files that belong to a plugin are managed in the RT-Plugin Builder in the form of a logical tree structure.

- Right-click the project folder.
The shortcut menu opens.



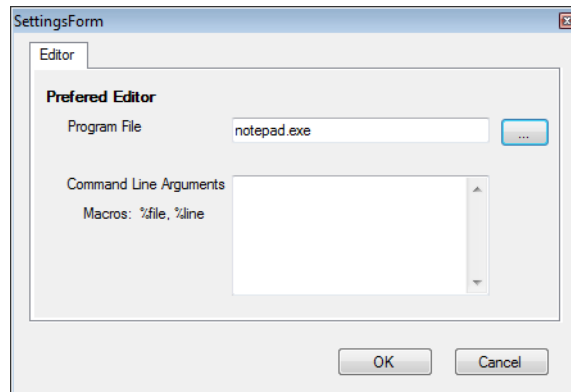
- Use **Create...** to create new files or folders.
- Use **Add...** (in the project directory) to add existing files or folders.
- Select **Exclude** to remove a file or subfolder from the project.

The files/folders in the project folder are retained and can be added again using **Add File / Add Folder**.

- To delete a file or a subfolder, select **Remove**.
The files/folders are deleted physically.

To edit files

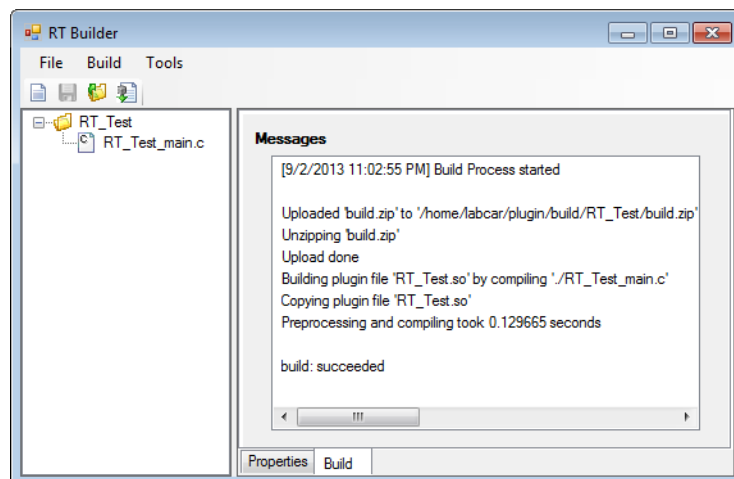
- Double-click a file in the tree to edit it.
The file opens in the preset editor.
- Select **Tools** → **Settings** to define another editor.



- If you want to transfer arguments when the editor is invoked, you can specify them in the field "Command Line Arguments".

To compile and link a real-time plugin

- Toggle to the "Build" tab.
- Select **Build** → **Build Plugin**.
The files are transferred to the Real-Time PC and processed there.



The results of the Build process are displayed under "Messages".

Real-Time Plugin Packages

A Shared Object Library is generated from the files containing source code during the build process. The RT-Plugin Builder puts these files, together with other files, into an archive file referred to below as a real-time plugin package.

A real-time plugin package contains:

- A metadata file (xml) with information on tools, versions etc.
- The Shared Object Library of the plugin

3.11.2 Example

The following example shows the implementation of a simple counter that can be controlled using the commands "enable", "disable" and "reset".

The plugin interacts with a C code module of the LABCAR-OPERATOR project by reading the scaling factor from the module and writing the (scaled) counter value into a module parameter "In". The C code module issues "In" via the outputport "Out".

```
#include <stdio.h>
#include <math.h>
#include "rtos_hook.h"
#include "rtos_rtplugin.h"

// Label definition
//   raw label
#typedeflabel double cmodIn CModule/CalibrationVariables/In

//   mapped (user defined label)
#typedeflabel double cmodScale UDModule/UDScale

// Data structure
typedef struct
{
    unsigned int enable;
    unsigned int count;
}
TData;
```

```

// Callback functions for the hooks
/**
 * Function to be called from the hook.
 * @param arg  Datapointer as passed to rtos_hook_attach
 * @param t_ns  The current time in nanoseconds
 * */
static void Hook_Callback(void *arg, rtos_time_t t_ns)
{
    if( arg )
    {
        if( ((TData*)arg)->enable )
        {
            ((TData*)arg)->count++;
        }
        cmodIn =  cmodScale * ((TData*)arg)->count;
    }
}

static rtos_handle_t hObj;
static TData data;

// Command handler for additional plugin commands
/**
 * Function to be called as command handler
 * @param argc  Number of argv[] strings
 * @param argv  Array of strings
 * @param stream  The output stream
 * */
static int cmd(int argc, char *argv[], FILE *stream)
{
    if( strcmp( argv[0], "enable" ) == 0 )
    {
        data.enable = 1;
    }
    else if ( strcmp( argv[0], "disable" ) == 0 )
    {
        data.enable = 0;
    }
    else if ( strcmp( argv[0], "reset" ) == 0 )
    {
        data.count = 0;
    }
    else
    {
        rtos_log( LOG_ERR, "unknown command" );
    }
}

```

```
        return -1;
    }
    return 0;
}

int on_load(void)
{
    return 0;
}

int on_initialize(void)
{
    rtos_rtplugin_command(cmd);
    data.count = 0;
    data.enable = 1;
    hObj = rtos_hook_attach("Hook", Hook_Callback,
                           (void*)&data );

    return 0;
}

void on_terminate(void)
{
    rtos_hook_detach(hObj);
}

void on_unload(void)
{
}
```

See the following sections for explanations of this example.

3.11.3 Defining Hooks in the OS Configuration of LABCAR-IP

A hook acts as a wildcard for one or more processes in the OS configuration of a LABCAR project. Like a process, a hook can be added to a task at any point, moved and even deleted. Unlike processes, the name of a hook can be freely defined (and also changed) by the user.

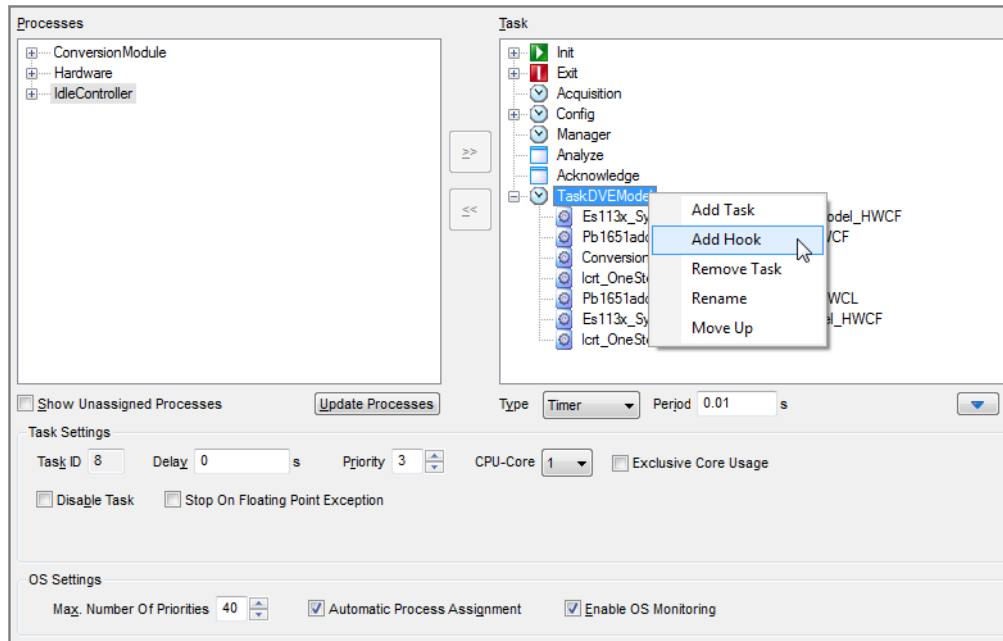


Fig. 3-30 Adding a Hook to the OS Configuration

A hook can also be used several times, i.e. it is permissible to add a hook with the same name at different positions within one or more tasks.

3.11.4 Attaching Plugin Processes to Hooks

Attaching processes to a hook takes place via the real-time plugin itself. When loading a real-time plugin, the

```
on_load
```

method of the plugin is invoked.

Within it, you can use the function

```
rtos_hook_attach
```

to attach processes to specific hooks.

If a necessary hook is not available, the implementation of the `on_load` method determines whether loading the real-time plugin has to be interrupted or whether it can be continued.

It is possible to attach several processes to one hook. By default, new processes are always attached to the end of the process list of a hook. It is not possible to add a process at any position in the hook process list.

When unloading a real-time plugin, all attached processes in the

```
on_unload
```

method must be removed from the process lists of the hooks using the function

```
rtos_hook_detach
```

3.11.5 Label and Mapping

Every data element in a LABCAR-OPERATOR project has a label that is used to enable unique addressing.

Addressing Data Elements

Data elements of a LABCAR-OPERATOR project are addressed using a compiler directive in real-time plugins.

The syntax of the directive is as follows:

```
#typedlabel <type> <variable identifier> <data element label>
```

The data element is accessed in the source code of the plugin using the specified variable identifier. A label is resolved into the memory address of the data element during runtime when the plugin is loaded – before the plugin function `on_load` is invoked.

If the label is unknown within the LABCAR-OPERATOR project or the data type of the data element is not the type expected, the real-time plugin is not loaded and corresponding error entries are made in the ETAS RTPC log file.

Mapping Labels

User-defined labels for data elements are realized in LABCAR-OPERATOR using mapping files and SuT mapping files. Using user-defined labels in a real-time plugin enables project-independent plugins to be created.

A user-defined label is resolved into the memory address of the data element during runtime when the plugin is loaded – before the plugin function `on_load` is invoked.

If the label is unknown within the LABCAR-OPERATOR project or cannot be resolved due to a missing mapping or the data type of the data element is not the type expected, the real-time plugin is not loaded and corresponding error entries are made in the ETAS RTPC log file.

Background Services

Functions that are not to be run in a real-time context (e.g. accessing files) can be assigned to the background tasks of the operating system using the function `rtos_hook_attach` – the corresponding hook identifier is `rtos_hook_background`.

3.11.6 The Lifecycle of a Real-Time Plugin

The activity diagram in Fig. 3-31 shows the sequence of the individual activities when loading or unloading the real-time plugin.

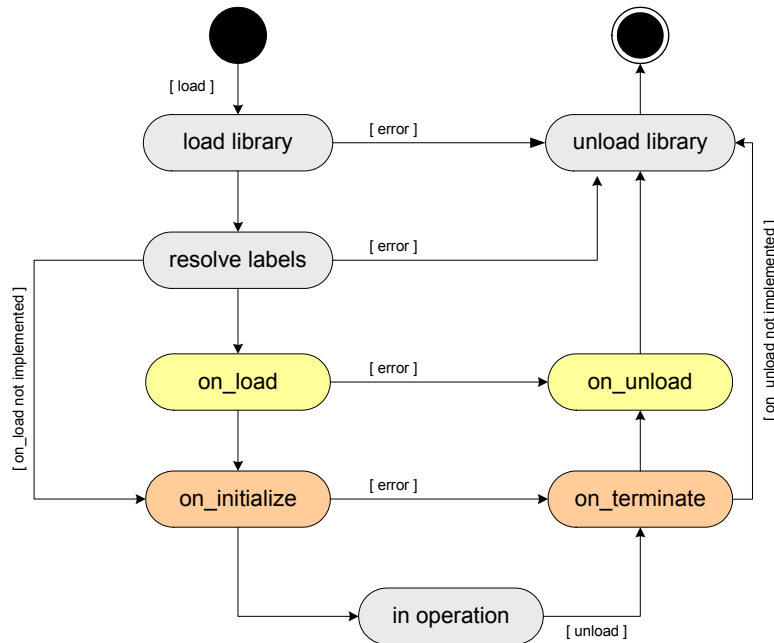


Fig. 3-31 Activity Diagram

- **load library**

Loads the Shared Object Library and resolves the function pointers for the callback functions of the plugin

- on_load (optional)
- on_initialize (necessary)
- on_terminate (necessary)
- on_unload (optional).

Error case: The function pointers for necessary callback functions cannot be resolved.

- **resolve labels**

Resolves labels for data elements in memory addresses.

Error case: Label of a data element cannot be resolved or the type of data element is incompatible.

- **on_load**

Invokes the callback function `on_load`, if implemented. This function is used to treat dependencies and reserve system resources.

Error case: Return value of the callback function is < 0.

- **on_initialize**

Invokes the callback function `on_initialize`. In this function, real-time functions are assigned to specific hooks, background functions and command handlers are registered.

Error case: Return value of the callback function is < 0 .

- **in operation**

Invokes the registered real-time and non-real-time functions by the operating system of the Real-Time PC.

- **on_terminate**

Invokes the callback function `on_terminate`. In this function, the functions registered previously in `on_initialize` are terminated.

- **on_unload**

Invokes the callback function `on_unload`, if implemented. System resources that were previously reserved in `on_load` are rereleased in this function.

- **unload library**

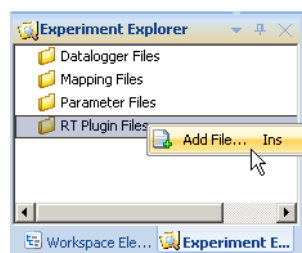
The Shared Object Library of the real-time plugin is released and unloaded.

3.11.7 Working with Real-Time Plugins in the Experiment Environment

Real-time plugins are managed in the experiment environment in exactly the same way as parameter or mapping files, i.e. adding, removing, enabling and disabling take place in the Experiment Explorer of ETAS EE.

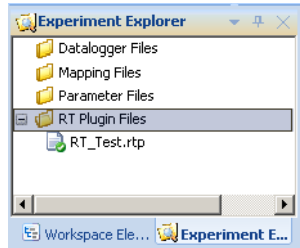
To add RT plugins to the experiment

- Right-click the "RT Plugin Files" folder in the Experiment Explorer.
- Select **Add File**.

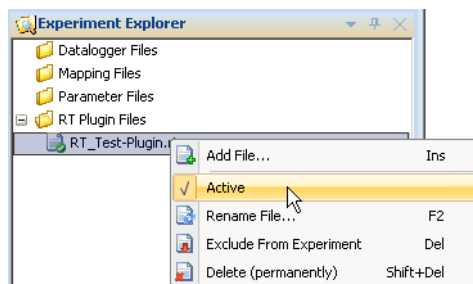


- Select the plugin file (*.rtp) in the file selection window.

The real-time plugin is added to the experiment.



- If the plugin is not active, right-click the file and select **Active**.



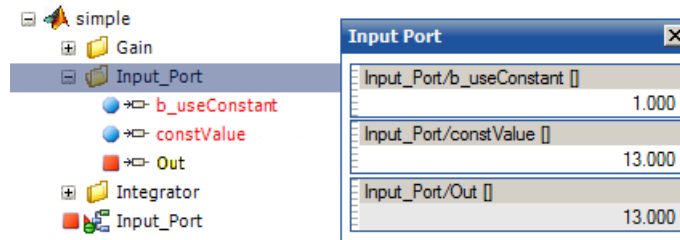
- To remove a file from the "RT Plugin Files" folder, select **Exclude From Experiment**.
- To delete a file, select **Delete (permanently)**.

To run an experiment with a real-time plugin

- Select **Experiment** → **Download** → **LABCAR** to download the experiment to the simulation target.
The state of the plugin is now "Initialized".
- Select **Experiment** → **Start Simulation** → **LABCAR** to start the simulation.
The experiment is run.
- Toggle to the "RT Plugins" tab.

Note on Using LABCAR Ports in Simulink Models

LABCAR input ports of Simulink models should not be written directly by real-time plugins. Instead, set the parameter "B_useConstant" to "1.0" and write the parameter "constValue" from the real-time plugin.



This results in the measure value "Out" ("simple/Input_Port/Out") and all subsequent Simulink blocks being influenced reliably if the corresponding hook has been added accordingly in the OS configuration.

LABCAR output ports should not be written by real-time plugins either. Writing input ports of C code modules is, however, possible.

3.11.8 The "RT Plugins" Tab

The following figure shows the "RT Plugins" tab.

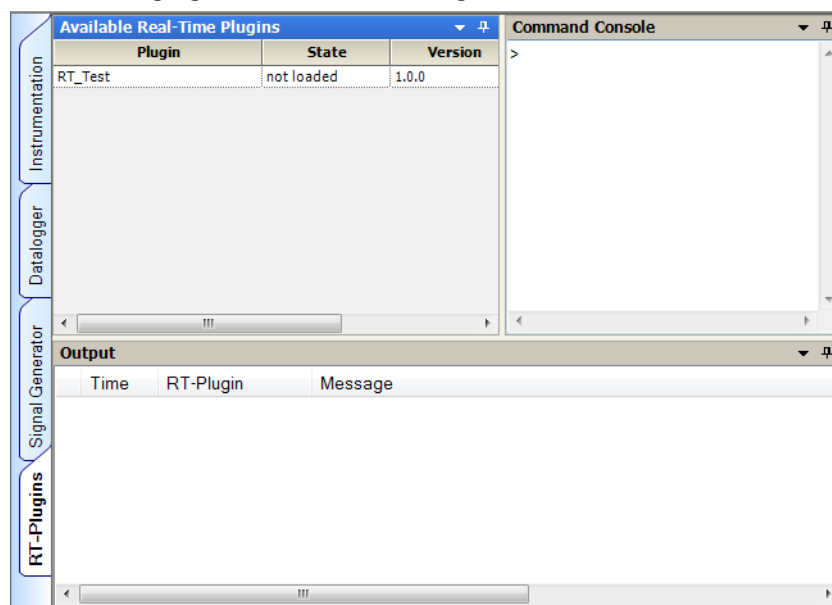


Fig. 3-32 The "RT Plugins" Tab in ETAS EE

The area consists of three parts:

- Available real-time plugins
- Output (see "Output" on page 223)
- Command console (see "Command Console" on page 223)

Available Real-Time Plugins

This is where all real-time plugins of the experiment are listed.

- **Plugin**

The name of the plugin

- **State**

The "State" column shows the current state of the individual plugin. The query is cyclical in a raster of around 500 ms.

The states "loaded" and "not loaded" are predefined and show the load state of a plugin. The definition of user-defined states is possible and takes place in the source code of the plugin.

- **Version**

The version of the plugin (is assigned during creation in the RT-Plugin Builder)

Output

The "Output" tab (in the bottom half) shows the log messages generated by the real-time plugins.

Command Console

The command console makes it possible to implement an individual command protocol (based on ASCII character strings) for each real-time plugin. For this purpose, the plugin must make a protocol handler available to interpret the transferred strings.

This handler must be registered when the real-time plugin is loaded using

```
rtos_plugin_command
```

The following commands are available:

at or @

Delegates the command specified subsequently to the named RT plugin
Example: `at ExamplePlugin disable`

clear

Clears the command console

info

Lists all plugins loaded (on the target)

help

Provides help

load

Loads the named plugin (→ `on_load`)
Example: `load ExamplePlugin`

unload

Unloads the named plugin (→ `on_unload`)
Example: `unload ExamplePlugin`

init

Initializes a plugin (→ `on_initialize`)
Example: `init ExamplePlugin`

terminate

Terminates a plugin (→ `on_terminate`)
Example: `terminate ExamplePlugin`

<Tab>

Auto-completion (case-sensitive)

<Arrow up>

Command history

<Arrow down>

Command history

3.12 Signal Conversion Modules

LABCAR-OPERATOR V5.4.4 comes with a standard signal conversion module with which the generic open-loop configuration can be realized.

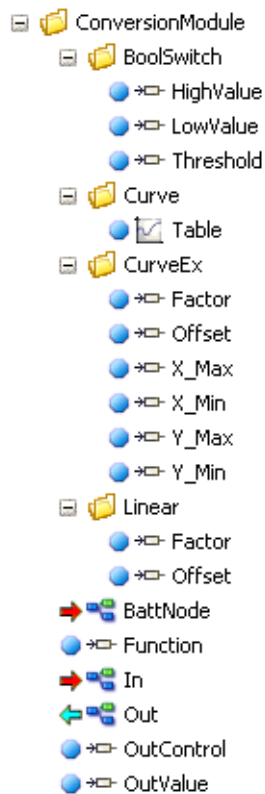
This module can be inserted between all ports which can be connected in the Connection Manager and particularly enables sensor/actuator modeling (between the I/O hardware and the DVE model).

3.12.1 Inserting Modules

How to insert this kind of module for signal conversion is described in the section "Signal Conversion" on page 240.

3.12.2 Parameters

If a module for signal conversion has been inserted, all inputs, outputs and parameters are also available in ETAS EE in the "Workspace Elements" window.

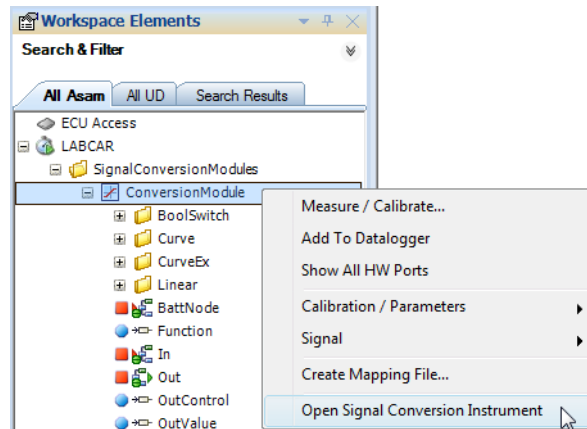


In addition to the parameterization via parameter files, the instrumentation of ETAS EE also provides a GUI via which the parameters of the module can be calibrated.

The meaning of the parameters (and thus the functioning of the modules) is described below using the GUI.

3.12.3 The GUI for Controlling Signal Conversion

To create this kind of window, right-click the relevant module in the tree view of the "Workspace Elements" window and select **Open Signal Conversion Instrument**.



In the GUI which is created in the active layer (in the "Instrumentation" tab), the following conversion types can be selected under "Type":

- Boolean
- Curve
- CurveEx
- Identity
- Linear

Controlling the Output Signal

What all types have in common is that the output can be controlled – the options in the "Out Control" field are used for this purpose.



The following settings are possible:

- On
The output is always calculated.
- Off
The output is not calculated - the output is set to the value specified.

- Node

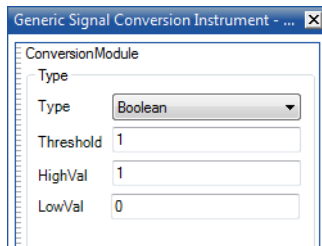
The calculation of the output depends on the state of a battery node ("BattNode" input in the Workspace Elements and in the Connection Manager). If the selected battery node is not switched, the curve is not calculated.

Note

If you doubt the values of any signal, always check whether the calculation of the signal depends on the status of a battery node or has been completely switched off.

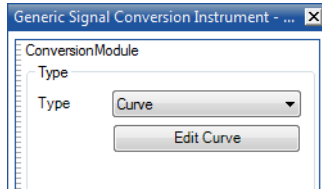
"Boolean"

A "Threshold" for the input signal can be entered here under which the output value assumes the value "LowVal" and above which it assumes the value "HighVal".

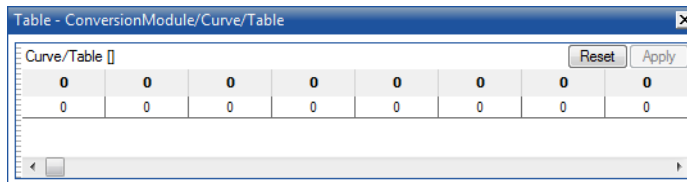


"Curve"

Use the **Edit Curve** button to edit a table for a curve.

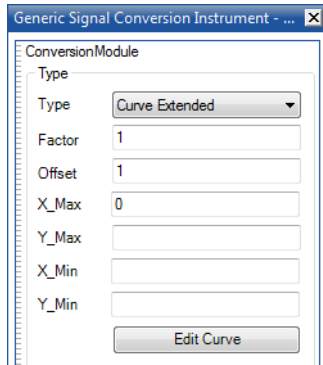


A GUI opens in which the value pairs for the parameter "ConversionModule/ Curve/Table" can be edited.



"Curve Extended"

The extended form of "Curve" which permits a further modification of the curve.



The signal can be modified here in several levels:

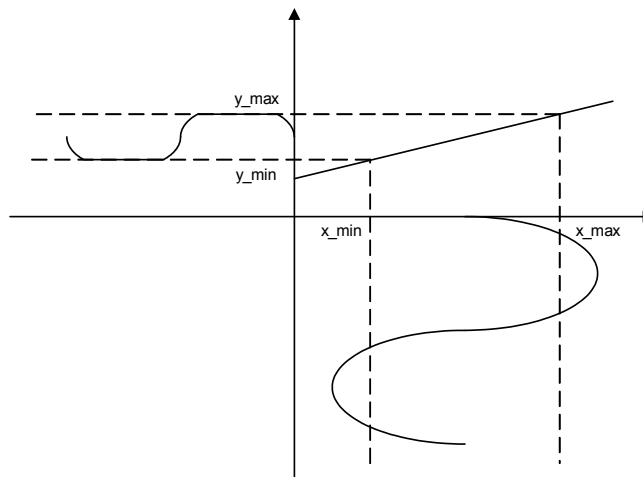
1. Mapping via a linear function $v = a \cdot u + b$ which results in a gain or attenuation a ("Factor") and an offset b ("Offset") of the signal. Presetting: Offset = 0 and Factor = 1, i.e. no change.

Note

*This mapping acts upon the **input values** of the characteristic defined in the next step!*

2. Mapping via a characteristic defined in a table with **Edit Curve** (see "Curve" on page 227.)
3. A signal limitation (specified via "y_min" and "y_max") plus a linear mapping (by specifying corresponding "x_min", "x_max", "y_min" and "y_max" values).

The following figure shows the effect of the delimiter function which is defined by the values x_{min} , x_{max} , y_{min} and y_{max} .



The input signal is limited by x_{min} and x_{max} - specifying x_{min} , x_{max} , y_{min} and y_{max} determines a linear equation which assigns the signal a gain and an offset.

If the signal is not to be limited, select the minimum and maximum values correspondingly small/large.

If no further gain and no offset are required, just set $y_{min} = x_{min}$ and $y_{max} = x_{max}$.

Otherwise, there is the following connection between factor (slope a of linear equation (1)), offset (parameter b of the linear equation (1)) and the values x_{max} , x_{min} , y_{max} and y_{min} :

$$y = a \cdot x + b \quad (1)$$

$$y = \left(\frac{y_{max} - y_{min}}{x_{max} - x_{min}} \right) \cdot x - \left(\frac{y_{max} - y_{min}}{x_{max} - x_{min}} \right) \cdot x_1 + y_1$$

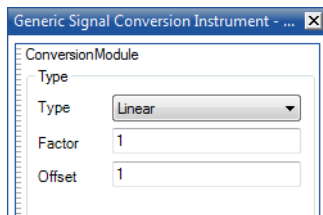
where one of the two value pairs (x_{max}, y_{max}) or (x_{min}, y_{min}) is to be used for x_1 and y_1 .

"Identity"

Signal is transferred unchanged 1:1.

"Linear"

Used to map the input value to the output value using a linear equation (expressed with "Gain" and "Offset").



3.13 Configuring Hardware with the RTIO-Editor

How to work with the RTIO-Editor and configure the hardware used is described in detail in LABCAR-RTC V5.4.4 - User's Guide.

Refer to the Tutorial in the "LABCAR-OPERATOR V5.4.4 - Getting Started" manual for an example of how to configure a hardware board.

3.14 The Connection Manager

The Connection Manager enables closed-loop operation by connecting the inputs and outputs of the existing modules accordingly.

Its properties are:

- displaying all inputs and outputs that can be connected
- connecting individual inputs and outputs and interrupting them
- creating virtual connections for signal tracing
- highlighting existing connections
- possibility for export/import of the connection list into/out of other projects
- sorting and display options for connection list
- filter for selecting special connections

The following sections describe how to assign models inputs and outputs, create connections and insert modules for signal conversion.

3.14.1 Real and Virtual Connections

Signals must be known to be displayed in the Connection Manager. This is a matter of course for modules integrated with LABCAR-IP as the inputs and outputs of a module are part of the module description. This concerns the modules already described such as ASCET and Simulink models, CAN and FlexRay modules, C code modules provided by the user, modules for adjusting the signal path (signal conversion and opening the closed-loop) and the hardware configuration in LABCAR-RTC.

Their signals are available in the Connection Manager and also in the experiment in ETAS EE after code generation. The following figure is a schematic of an HiL experiment with the typical components (hardware, signal conversion) and a model with actuator and sensor simulation.

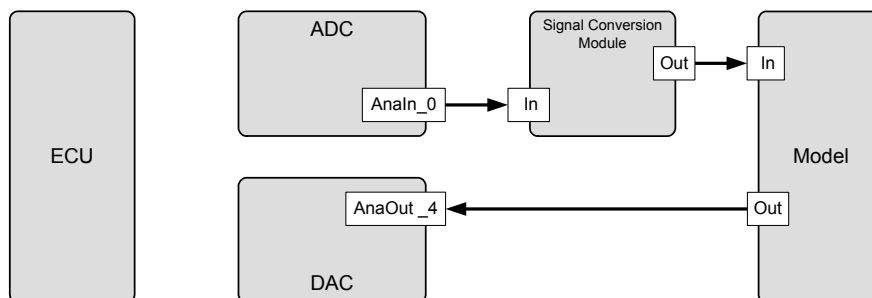


Fig. 3-33 The Experiment Signals

The actuator signal from the ECU appears here for the first time as an output signal of an ADC module and is then routed to a model input via a module for signal conversion. In the case of the sensor, the known signal path starts at a model output and ends at the input of a DAC module.

The following are unknown to date:

- the connections between the ECU and the RTIO hardware and
- the connections within the modules

This means important information on the complete signal path in a closed-loop experiment is missing: the experiment environment requires it to be traceable.

The first point is addressed by the provision of lists of ECU pins and HW pins; the second via what are referred to as virtual connections.

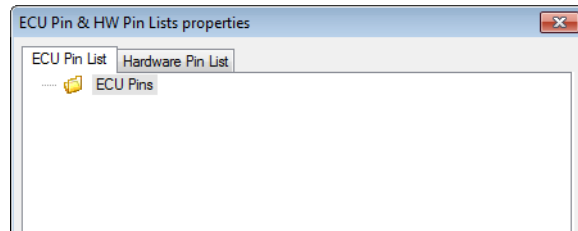
ECU and HW Pins

You can create and edit the ECU Pin List or the Hardware Pin List as described below. This can also be automated using the LABCAR-IP Scripting API. Take a look at the help section (? → **Help**) for documentation on this API.

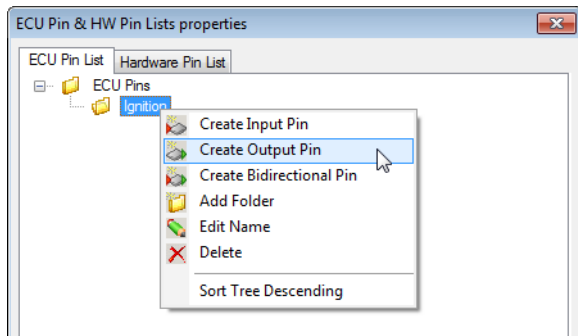
To create a list of ECU pins

- In the main menu of LABCAR-IP, select **Project → ECU Pin List**.

The "ECU Pin & HW Pin Lists properties" window opens.



- Select the "ECU Pin List" tab.
- Right-click on the root entry "ECU Pins".
- Select **Add Folder**.
- A new folder is created whose name you can change here.
- Right-click this folder and select, for example, **Create Output Pin**.



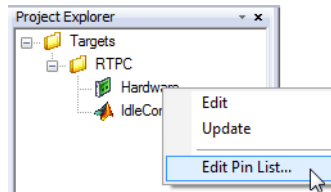
To create a list of HW pins

- Select **Project → ECU Pin List** and then the "Hardware Pin List" tab.

or

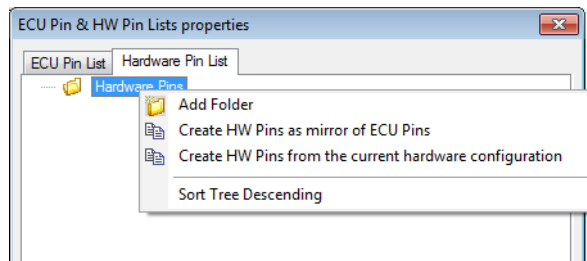
- Select the hardware module in the Project Explorer.

- Right-click and select **Edit Pin List**.



The "ECU Pin & HW Pin Lists properties" window opens.

- Select the "Hardware Pin List" tab.
- In the shortcut menu of the root entry "Hardware", you now have the following possibilities.



- To create a folder in which you have all possibilities as already described for the ECU Pin List.
- To create the Hardware Pin List as a mirror of an existing ECU Pin List.
- To create the Hardware Pin List from the signals of the current hardware configuration. With this variant virtual connections between hardware pins and the relevant hardware signals are also created automatically.

The result is that connections of ECU and hardware are available in the Connection Manager and can be connected.

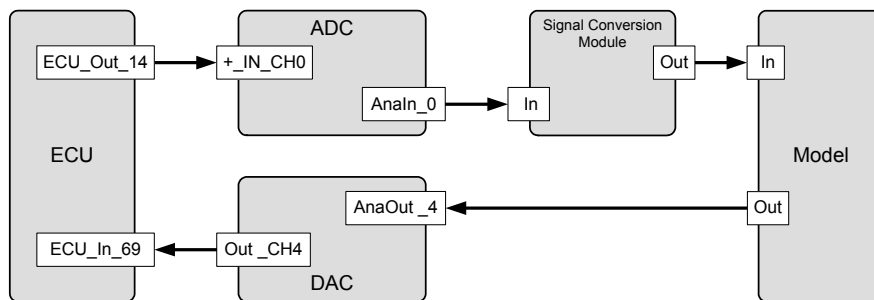


Fig. 3-34 The Signals and Pins of ECU and Hardware

Virtual Connections

To enable signal tracing the connections within a module also have to be known. In simple cases, for example a hardware board, in which the list of HW pins was created automatically from the existing configuration, this can take place automatically as the pin and the signal are effectively hard-wired. It is also natural in the case of a signal conversion module that input and output are connected internally with each other.

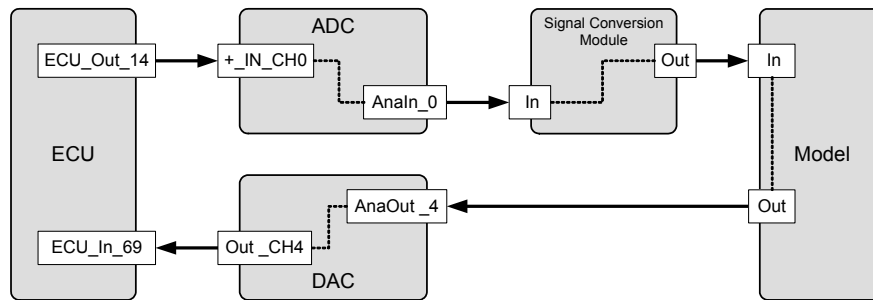


Fig. 3-35 Signals, Pins and Virtual Connections (Dashed Lines)

Note

The connections between the ECU and the LABCAR hardware are real to the extent that lines are connected between the ECU and the hardware. Nevertheless, these connections also have to be created as virtual connections in the Connection Manager ("ECU_Out_14" and "+_IN_CH0" as well as "ECU_In_69" and "Out_CH4" in Fig. 3-35). This enables full tracing of the signals between the ECU and the LABCAR hardware.

For information on how to create virtual connections see the section "To create a virtual connection" on page 238.

3.14.2 Adding Inports and Outports to the Simulink Model

Model signals must be accessible if they are to be connected to hardware signals for HiL applications. This takes place using what are referred to as LABCAR inports and outports.

These blocks can be positioned at any hierarchy level of the Simulink model.

In MiL applications, an inport can be assigned a constant value so that it behaves like a Simulink "constant block"; LABCAR outports then behave like Simulink "terminator blocks".

These inports and outports are in the "LABCAR Port Blocks" library (Fig. 3-36).

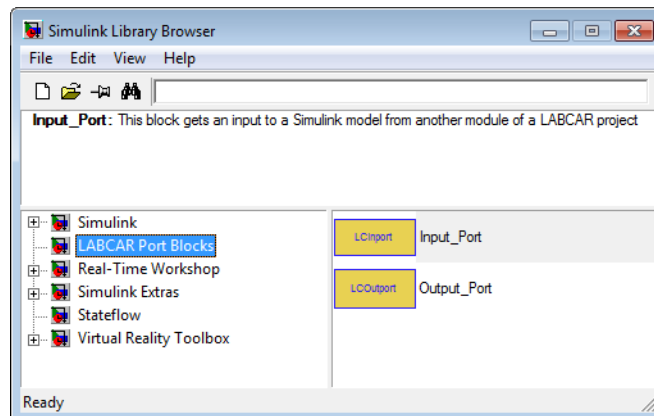


Fig. 3-36 LABCAR Port Blocks

The connections are created in the Connection Manager which provides an easy-to-operate user interface for this purpose (see "Connecting Signals in the Connection Manager" on page 236).

The advantage of this procedure is that this definition of the interface does not depend on the hardware used each time and is thus also valid for MiL applications. Hardware connections can also be changed after model code has been generated, meaning the code does not have to be regenerated.

Note

If these port blocks do not exist in the Library Browser, read the following section.

Adding LABCAR Port Blocks to the Library Browser

When the Simulink Browser opens, MATLAB searches all paths for files called `sliblocks.m`. These M files contain a description of the extensions of the Simulink Model Browser with corresponding elements.

In the case of LABCAR-OPERATOR, this file is in the directory
`<drive>:\Program Files\ETAS\LABCAR-OPERATORX.Y\SiCo.`

If you did not open your Simulink model from LABCAR-OPERATOR, the LABCAR port blocks are not available in the Library Browser. Add this path in the MATLAB Command Window with

```
>> addpath('<drive>:\Program Files\ETAS\
          LABCAR-OPERATORX.Y\SiCo');
```

If LABCAR-OPERATOR has never been installed on the system on which you are editing your Simulink model, please copy the entire `\SiCo` directory to it and make the relevant path known to MATLAB (as described above).

Data Types

As LABCAR-RTC only processes data of the type "double", an automatic casting from and to the basic Simulink data types has to be take place:

- double (LABCAR Inport) → double, float, boolean, signed/unsigned int8, int16, int32, int64
- double, float, boolean, signed/unsigned int8, int16, int32, int64 → double (LABCAR output)

3.14.3 Connecting Signals in the Connection Manager

The Connection Manager obtains all its information from an XML file which contains information on all inputs and outputs of all modules of the project.

The following figure shows the Connection Manager GUI (accessible via the corresponding tab in the main window).

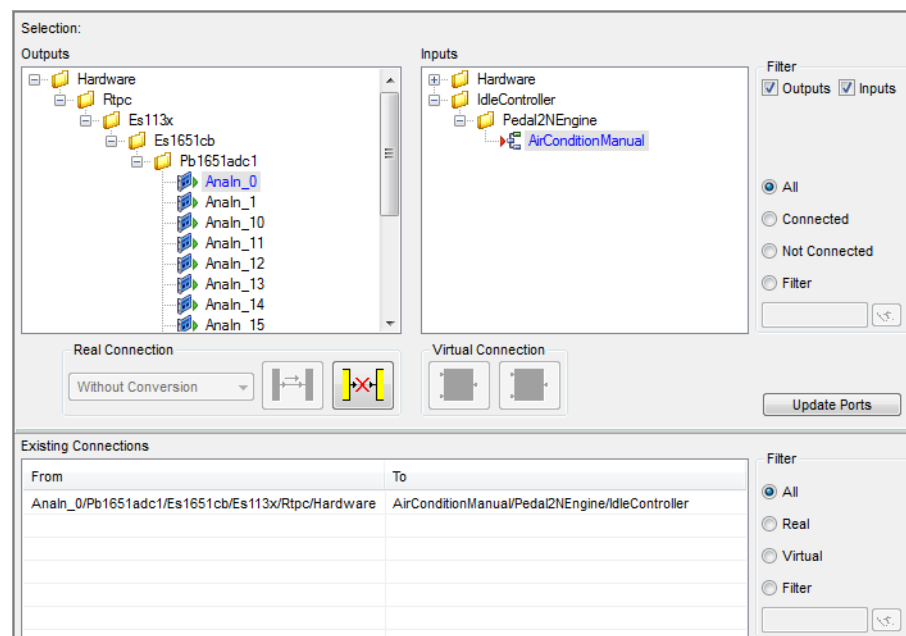


Fig. 3-37 The Connection Manager User Interface

The top part of the window shows the inputs and outputs of all the modules present.

To display a signal in the relevant editor

- Select a signal from the list of outports or inports.
- Right-click and select **Goto...**

The editor for signals of the relevant type opens and the selected signal is highlighted.

Filter on the Display

You can specify a filter to be used when displaying signals ("Filters" field) simplifying the signal overview according to the type of task to be executed.

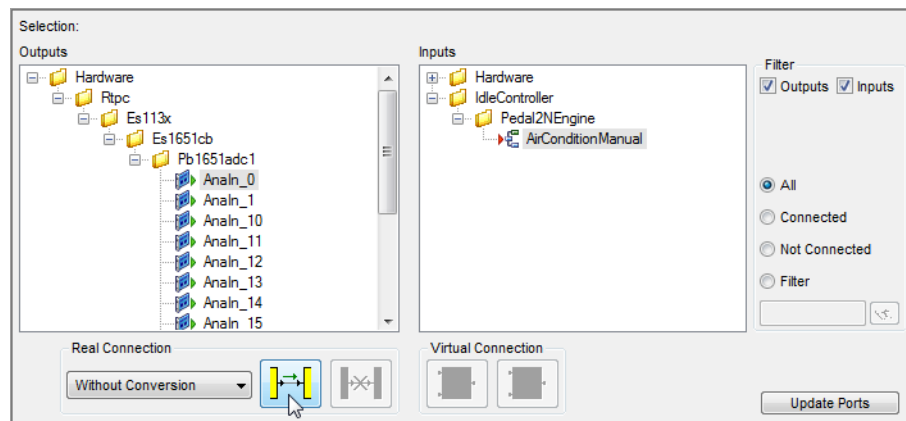
The following filter settings are possible:

- **All**
All available inputs and outputs are displayed.
- **Connected**
With this setting, only the connected inputs and outputs are displayed.
- **Not Connected**
With this setting, only the inputs and outputs which are not connected are displayed.
- **Filter**
A character string to filter the names can be entered here. No distinction is made between upper and lower case.

You can also select whether the filter condition only applies to outputs ("For Outputs"), inputs ("For Inputs") or both.

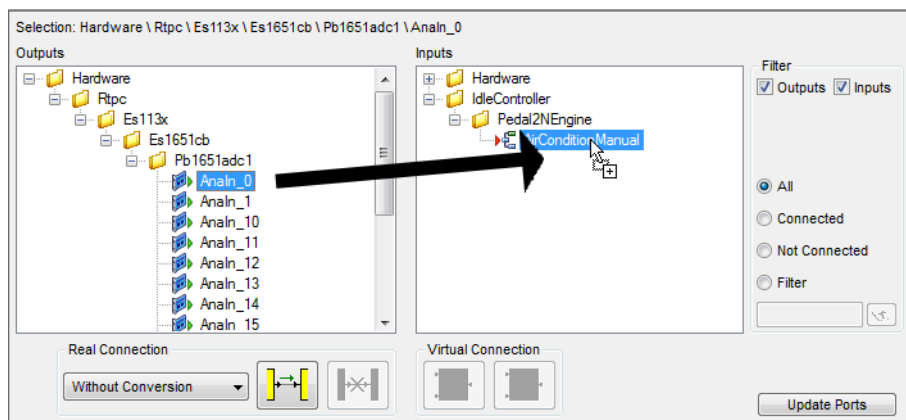
To connect signals

- Use the mouse to select an input and an output.
- To connect the signals click the icon.

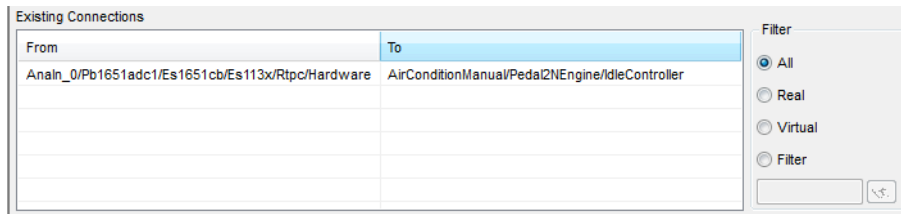


or

- Use the mouse to select an input/output and move it to the desired output/input.

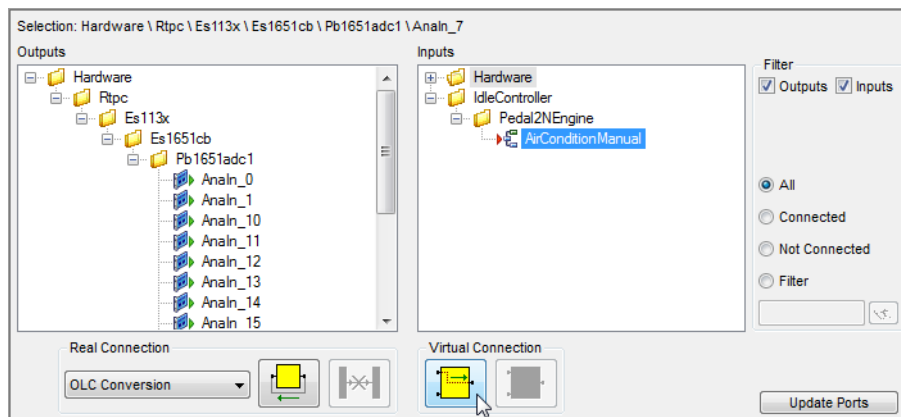


The connection is created and shown in the field "Existing Connections" (providing no filter is active preventing newly created connections from being displayed).



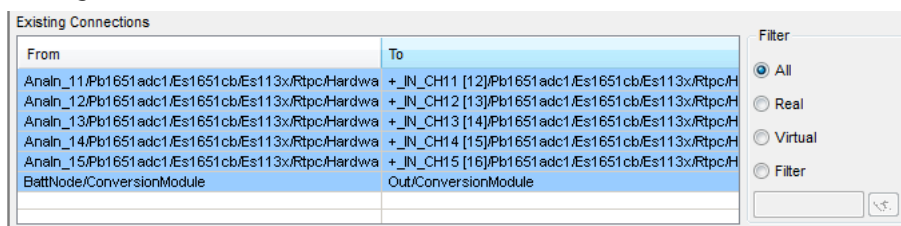
To create a virtual connection

To create a virtual connection between an input and an output of one and the same module, proceed in the same way as for real connections (see above) – the icon for creating the virtual connection (within a module) just looks different from the icon for creating real connections (between two modules).



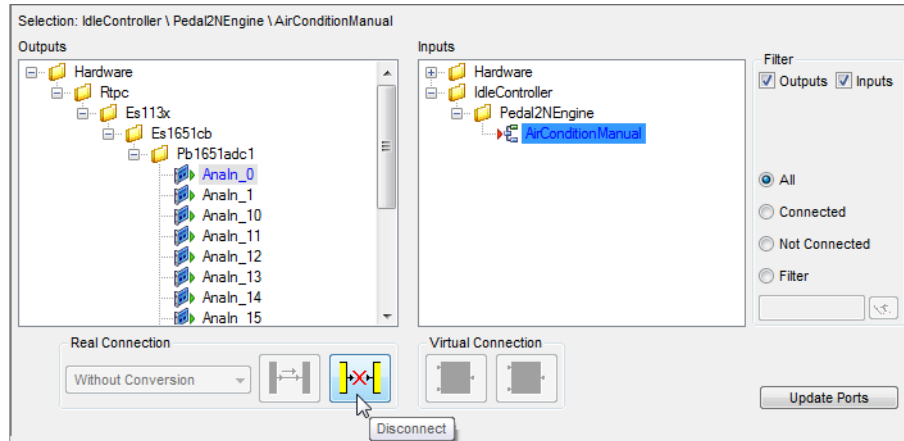
Beside it there is the icon for creating a real connection between an input and an output of the same module which can be used for feeding back signals.

Virtual connections are shown on a blue background in the Connection Manager.



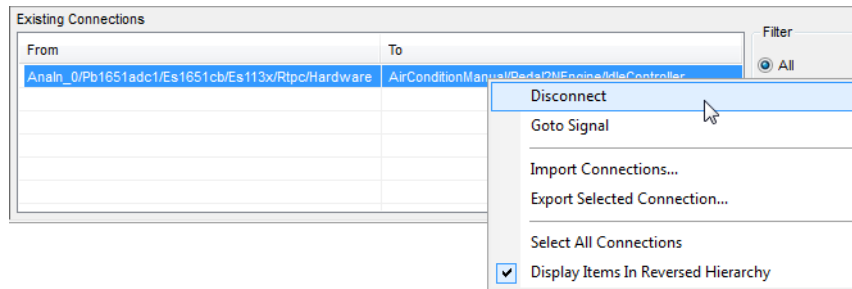
To disconnect connections

- Select the one of the signals whose connection is to be interrupted.
- To disconnect click the icon.



or

- Select an existing connection under "Existing Connections".
- Press the right-hand mouse button and select **Disconnect**.



The connection is interrupted again.

If changes are made a module, the display can be refreshed at any time.

To refresh the display

- Select **Update Ports**.
The display is updated.

To sort the connection list

- Double-click the column headings ("Outputs" or "Inputs") in the connection list.
The relevant column is sorted alphabetically (either in standard or in reverse order).

To reverse name hierarchy

- Select **Display Items in Reversed Hierarchy** from the shortcut menu of the connection list.

This reverses the hierarchy of the signal names. It now starts with the signal name and ends with the module name.

To export connection settings

- Use the mouse to select one or more connections (multiple selection is possible by pressing the CTRL- or SHIFT key).
- Select **Export Selected Connection** from the shortcut menu.

A file selection window opens.

- Enter a file name.

The selected connections are saved in the file.

This kind of saved connection settings can be imported again.

To import exported settings

- Select **Import Connections** from the shortcut menu of the connection list.

A file selection window opens.

- Select a file with connection data.

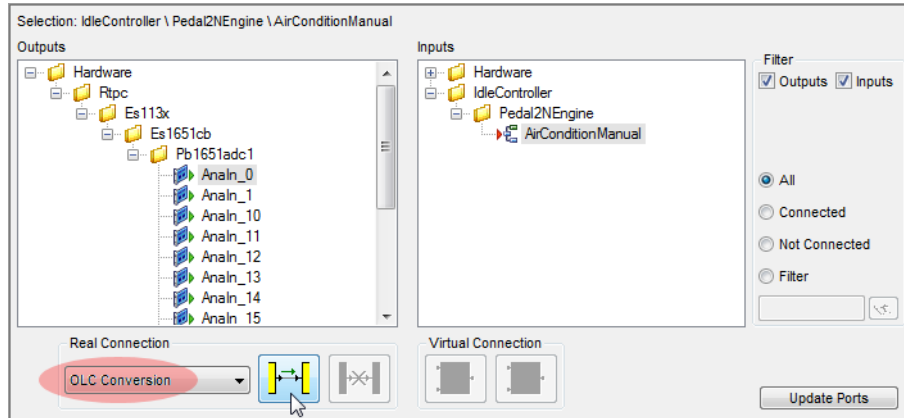
The selected connections are imported.

3.14.4 Signal Conversion

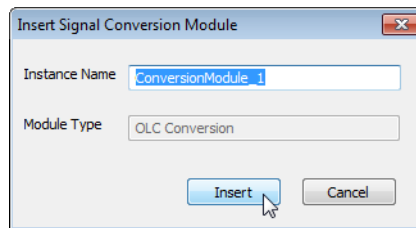
In the Connection Manager, generic signal conversion models (actuator/sensor modules) can be inserted between the signals to be connected (see "Signal Conversion Modules" on page 225) – the configuration of these modules then takes place in a special GUI in ETAS EE.

To create a connection with the signal conversion module

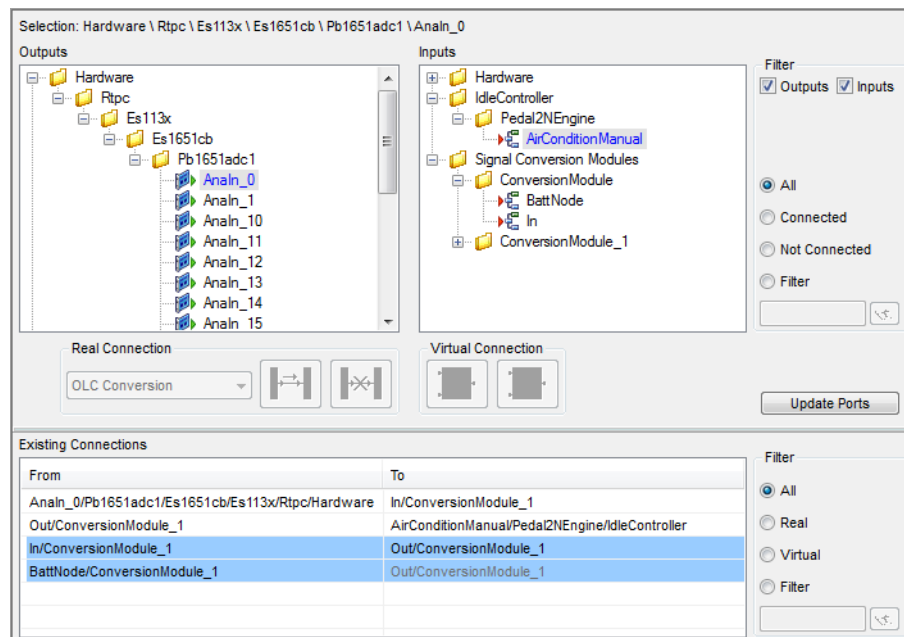
- Select the signals to be connected.
- Select **OLC Conversion** and click the icon.



- Enter a name for the module in the following window and click **Insert**.

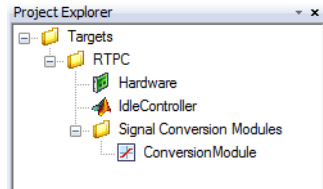


The two signals are connected using this kind of module by, for example, connecting the hardware output with the module input and the module output with the model input.

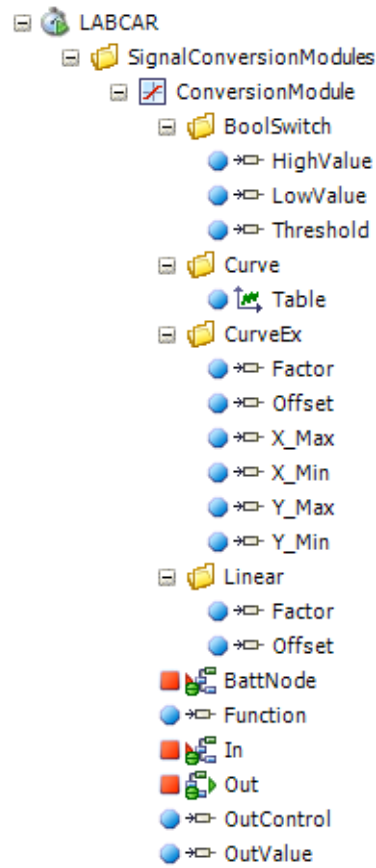


The virtual connections within the conversion module are also displayed (on a blue background).

The inserted module is as well displayed in the Project Explorer.



All inputs, outputs and parameters of the module are then accessible in the "Workspace Elements" window in ETAS EE.



3.15 Configuring the Real-Time Operating System (OS Configuration)

When creating a LABCAR-OPERATOR project, a default OS configuration is automatically created and added to the project.

The real-time operating system is configured in LABCAR-IP in an interface in which the following actions can be carried out:

- adding and removing tasks
- editing task properties
- assigning and removing processes to/from tasks

Working with the OS configuration interface is described in the section "Working with OS Configurations" on page 250.

If a new module is linked, the relevant processes are automatically integrated into the OS configuration and suitable tasks assigned.

3.15.1 Elements of an OS Configuration

The following figures each show sections of the "OS Configuration" tab in which specific settings are made.

Global Settings

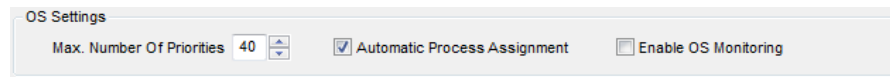


Fig. 3-38 Global Settings

- **Max. Number of Priorities**

Maximum number of priority levels

- **Automatic Process Assignment**

When adding a new module to the LABCAR-OPERATOR project, this option means that all processes of the newly added module are automatically assigned to the relevant task.

Note

*Existing assignments of processes of other modules are not affected by this! If errors occur due to manual changes made to automatic assignments, these can only be remedied by removing **all processes** from their tasks so that the next project build triggers a new automatic allocation.*

- **Enable OS Monitoring**

When this option is activated, a range of additional measure variables are added to the Workspace Elements in ETAS EE after the subsequent build process (under "OSMonitoring"). These variables provide information on the runtimes of the individual tasks.

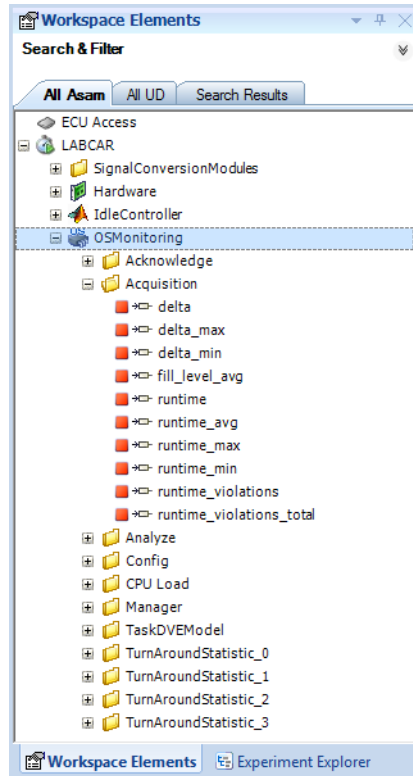


Fig. 3-39 Additional Measure Variables with "Enable OS Monitoring"
OS Monitoring of Tasks

The significance of the measure variables is as follows:

- delta (delta_max, delta_min)
Current time dt (in ns) between two tasks being activated (maximum, minimum time)
- fill_level_avg
The quotient of the average duration divided by the entire period duration.
Example: If a task with a period of 100 ms is invoked and has an average (!) duration of 20 ms, the fill level is 20%.
- runtime (runtime_avg, runtime_max, runtime_min)
Current runtime (in ns) of the task (average/maximum/minimum runtime)
- runtime_violations
Integer incremented by one whenever the current runtime ("runtime") exceeds the current period ("delta").
This variable is reset periodically – the length of a period until the measure

variable is reset is defined in the web interface of ETAS RTPC ([Main Page](#) → [Configuration](#)) via the RTPC_TASK_TIMING_STATISTIC parameter ("RTPC_TASK_TIMING_STATISTIC = 0" disables recording).

- runtime_violations_total
Number of "runtime violations" during the entire runtime of the experiment.

TurnAroundStatistic_n

The "TurnAroundStatistic_n" folders (n is the number of the VME chassis and corresponds to the Ethernet adapter "ETHn" with which it is connected to the Real-Time PC) contain the four variables "runtime", "runtime_avg", "runtime_min" and "runtime_max" that can be found in every task.

The following figure shows how "runtime" is calculated.

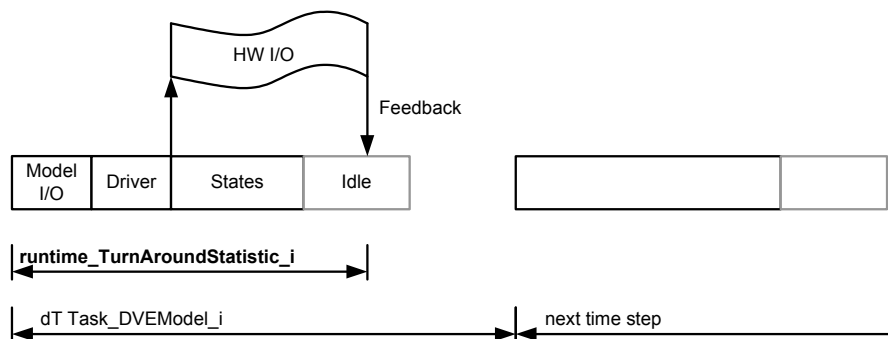


Fig. 3-40 Calculating "runtime_TurnaroundStatistic"

CPU Load

The "CPU Load" folder contains the measure variables "Core n" that in turn contain the load of each individual processor core. This helps optimize the distribution of tasks to the various cores (see "**CPU Core**" on page 246).

Task Settings

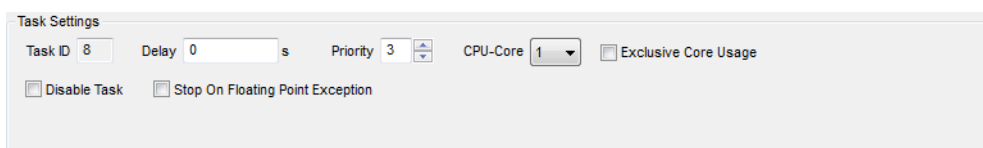


Fig. 3-41 Task Settings

- **Show Unassigned Processes**
If this option is enabled, only those processes are displayed in the "Processes" field (see "Processes" on page 247) that have not been assigned to any task.
- **Update Processes**
A search once again takes place in all modules of the project for the existing processes.

- **Type**
Task type (see "Tasks and Their Types" on page 248)
- **Period** (timer tasks only)
Period of the timer task in seconds
- **Task ID**
The ID of a task (cannot be edited) – software tasks, for example, are invoked using this ID (see "Tasks and Their Types" on page 248)
- **Delay**
Delay of task activation in seconds
- **Priority**
This is where a priority can be assigned to a specific task – the greater the number, the higher the priority in processing. The number of priority levels available can be set in the field "OS Settings" under "Max. Number of Priorities" (see "Global Settings" on page 243).

Note

The simulation target RTPC works with preemptive task scheduling – the task with the higher priority is always processed in the relevant processor core in specific time slices.

- **CPU Core**
The processor core that calculates this task.

Note

For distributed simulation (i.e., assigning different tasks to different cores) more than two cores are required!

- **Exclusive Core Usage**
This option ensures that the processor core selected previously is reserved for exclusive use by this task.

Note

*"Exclusive Core Usage" is intended to be used to run tasks that have very short cycle times ($< 100 \mu\text{s}$) and as low a latency as possible (e.g. hardware I/O). The "Idle" time of a task is filled with "Busy-Waiting" in this operating mode. Consequently the relevant processor core is always used to capacity (100%).
In particular, never operate the DVE task (or other slow tasks) with this setting!*

- **Disable Task**
State of the task: "enabled" or "disabled"

- **Stop Simulation on Floating Point Exception**

This option is only available if a Real-Time PC is used as a simulation target.

If this option is enabled, a floating-point exception is generated when an unusual value of a floating point number ("Not-a-number") occurs and the target is stopped. For more details on what might have caused the error, refer to the ETAS RTPC manual.

In the case of **"Trigger" tasks**, there is a further option:

- **Auto Trigger**

This ensures that a Trigger task is calculated even if there is no trigger (which would otherwise lead to the simulation coming to a standstill).

- **Period**

As long as there is no trigger event, the task is calculated with this period.

- **Event Timeout**

If there has been no trigger event once this time has elapsed, the Auto Trigger function is started.

Processes

The following figure shows a list of processes as shown in the user interface.

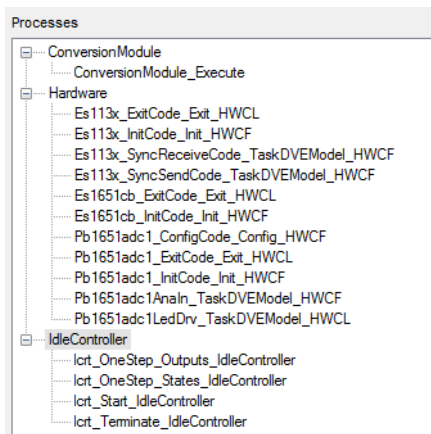


Fig. 3-42 The "Processes" Field of the "OS Configuration" Tab

Tasks and Their Types

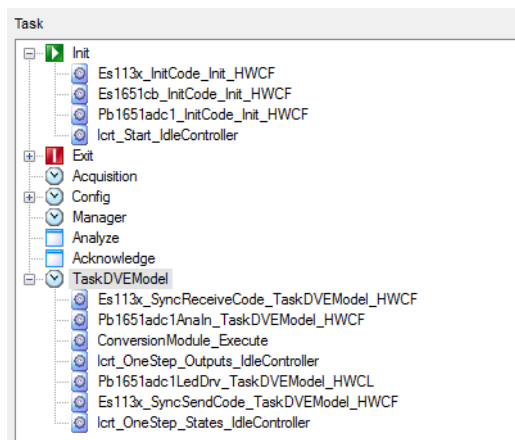








Fig. 3-43 The "Tasks" Field of the "OS Configuration" Tab

- 
 - **Init**
Is always executed at the start of simulation
- 
 - **Exit**
Is always executed at the end of simulation
- 
 - **Timer**
This task type is run in a specific periodic time frame (determined by "Period")
- 
 - **Event**
Special task which can be used by ETAS RTPC for event-controlled communication. A process within this task must execute a "blocking call" that waits for input from a hardware or the network.

A typical example is CAN access with a timeout. When `rtos_comm_read()` is invoked, the task waits (blocked) until a CAN message arrives (signaled by an interrupt). This makes it possible to react very quickly to incoming CAN messages.
- 
 - **Software**
Task which can be invoked with `activateTaskWithId(uint32 taskId)`
- 
 - **Trigger**
Task which can be triggered by external events (hardware, network, ...)

The following table contains the tasks of a default OS configuration, its types and further settings.

Task	Type	Period [sec]	Prio	Comment
Acquisition	Timer	0.1	1	For measuring
Init	Init	-		RTIO
Exit	Exit	-		RTIO
Config	Timer	0.1	1	RTIO
Manager	Timer	0.01	3	RTIO

Tab. 3-2 Tasks of a Default OS Configuration

Processing Sequence

Basically the processes are processed in the order in which they were added under the relevant processes (see e.g. "The "Tasks" Field of the "OS Configuration" Tab" on page 248)

The runtime may be able to be optimized by changing the sequence, e.g. if different Simulink models are calculated.

Note

*"Outputs" always have to be calculated **before** "States"!*

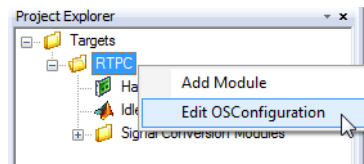
3.15.2 Working with OS Configurations

This section contains instructions on how to work with OS configurations. The individual sections are:

- "To edit an OS configuration" on page 250
- "To display extended settings" on page 251
- "To determine the assignment to a task" on page 252
- "To add a task" on page 252
- "To rename a task" on page 253
- "To assign task settings" on page 253
- "To remove a task" on page 253
- "To change the order of the tasks" on page 254
- "To add a process to a task" on page 254
- "To remove a process from a task" on page 255

To edit an OS configuration

- Select the "OS Configuration" tab
- or*
- Select the target in the Project Explorer (e.g. RTPC).
 - Right-click and select **Edit OS Configuration**.



The "OS Configuration" tab opens which contains the current OS configuration of the project.

- To show all processes and all tasks with the assigned processes, expand the relevant items.

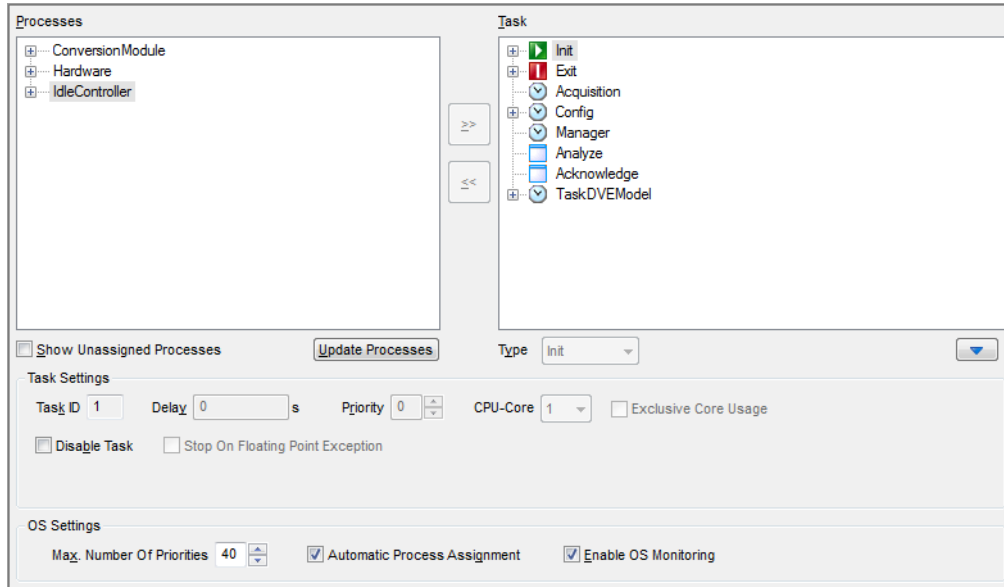


Fig. 3-44 The "OS Configuration" Tab (extended view)

To display extended settings



- Click the chevron button.
The fields "Task Settings" and "OS Settings" are added to the window.

The "Processes" Field

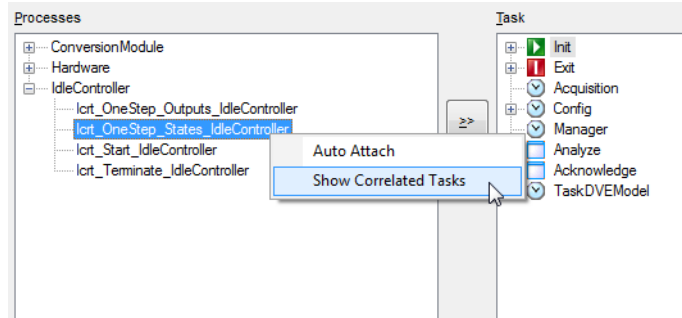
The tree view in this field shows all available processes of the hardware as well as of the model. If existing processes are not to be assigned to a task (field to the right of it), these can be made visible by activating the "Show Unassigned Processes" option.

If processes have not been assigned to tasks, select the function **Auto Attach** from the shortcut menu (see below).

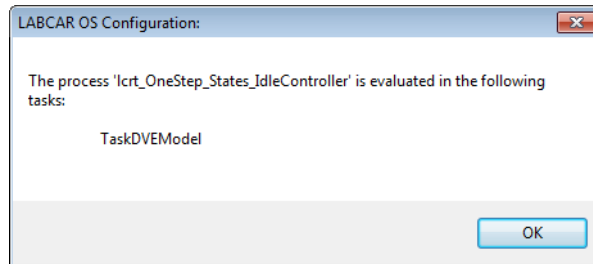
If you want to find out which task a particular process has been assigned to, proceed as follows.

To determine the assignment to a task

- Select a process in the "Processes" window.
- Select **Show Correlated Tasks** from the shortcut menu.



A window opens specifying the task to which the selected process is assigned.



The "Tasks" Field

This field shows all available tasks and the processes assigned to them.

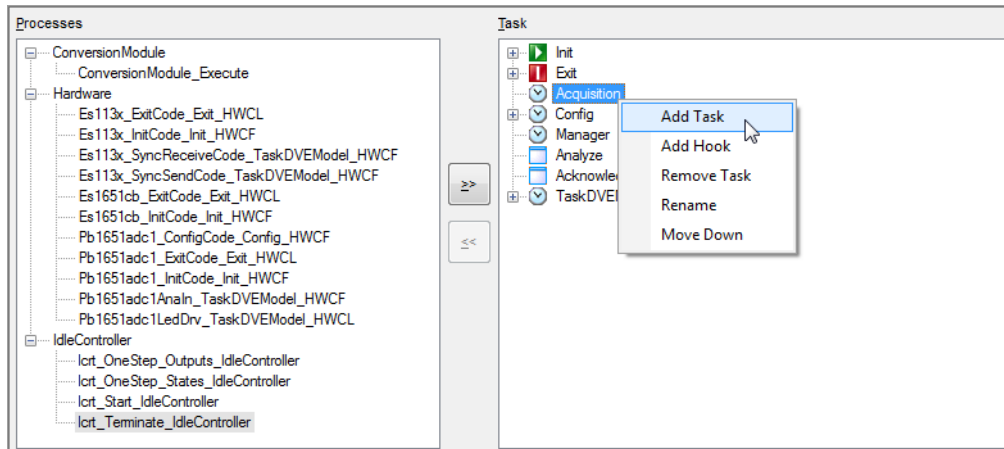
To add a task

- Select the task under which the new task is to be added.

Note

You can also change the order of the tasks later (see "To change the order of the tasks" on page 254).

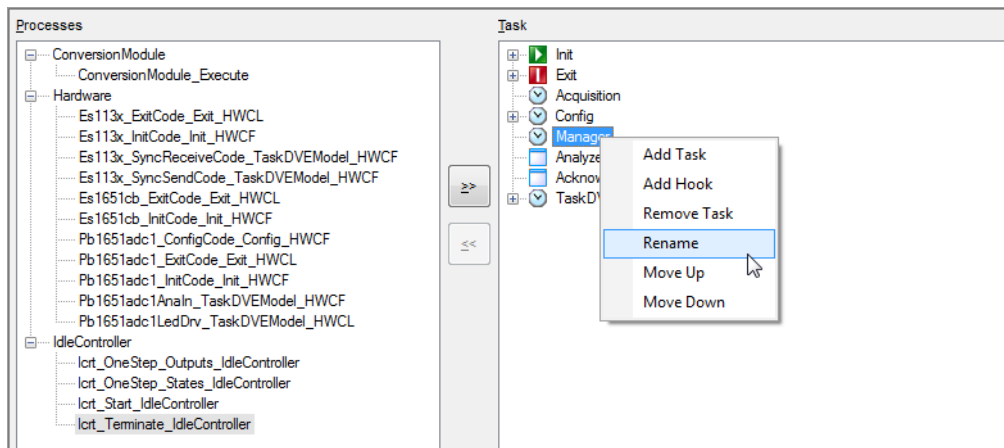
- Select **Add Task** from the shortcut menu.



A new task with the name "Task_n" (n is a sequential number) is added.

To rename a task

- Select the task to be renamed.
- Select **Rename** from the shortcut menu.



The task name can now be edited.

To assign task settings

- Assign the new task a type under "Type".
- Specify the "Task Period".

To remove a task

- Select the task to be removed.
- Select **Remove Task** from the shortcut menu.
The task is removed.

To change the order of the tasks

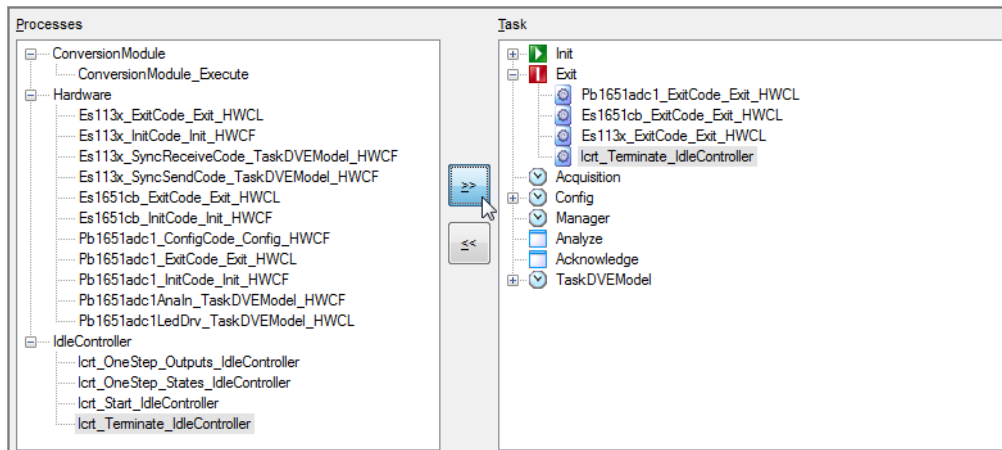
- Select the task to be moved.
- Select **Move Up** or **Move Down** from the shortcut menu.

The selected task is moved accordingly.

To add a process to a task

For example, to assign the process "Icrt_Terminate_IdleController" to the "Exit" task, proceed as follows:

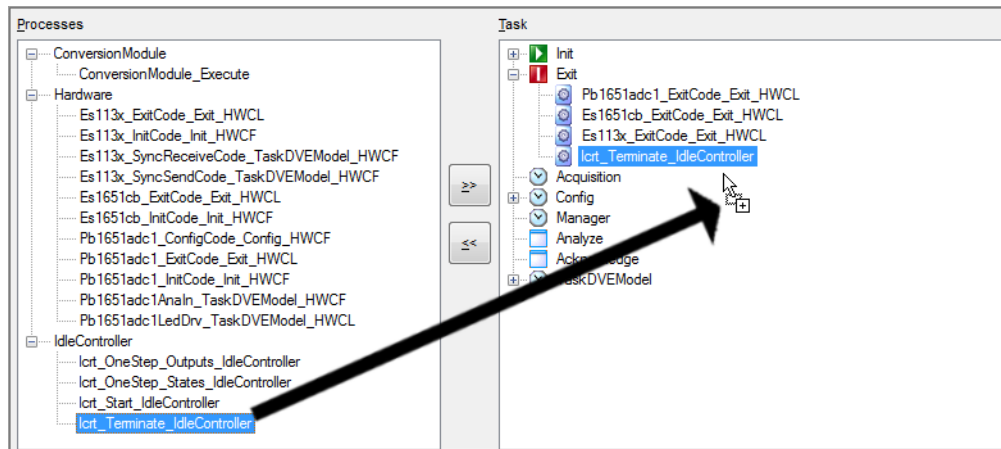
- Select the process to be assigned in the "Processes" field.
- In the "Tasks" field, select the process of a task under which the process is to be assigned.
- Click the >> button.



or

- Select the process to be assigned.

- Drag it, keeping the mouse button pressed down, to the process under which it is to be assigned.



The process is added under the selected process.

To remove a process from a task

- Select the process to be removed.
- Select **Remove Process** from the shortcut menu.
The selected process is removed from the task.

To create a hook

To create a software hook within a task, proceed as follows:

- Select the task within which the hook should be created.
- Select **Add Hook** from the shortcut menu.
The hook is created.

To rename a hook

- To change the name of the hook, select **Rename**.

To remove a hook

- To remove the hook, select **Remove**.

3.16 Setting Up Multi-RTPC Networks

Fig. 3-45 shows the setup and the components of a Multi-RTPC network

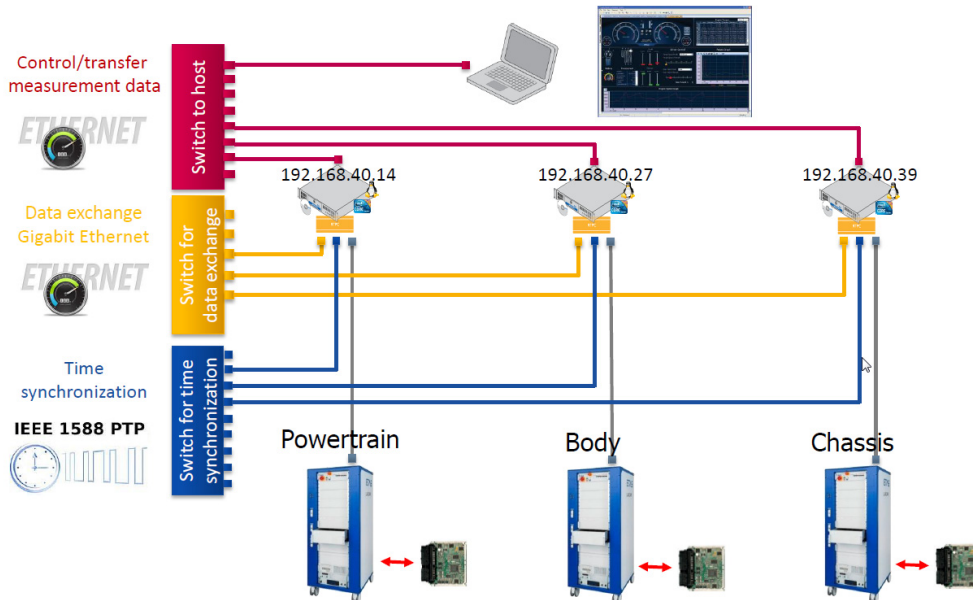


Fig. 3-45 Architecture of a Multi-RTPC System

The following components and functions are necessary for this:

- Control and transfer of measurement data between the user PC and the Real-Time PCs
- Data exchange between the Real-Time PCs
- Time synchronization between the Real-Time PCs

In terms of hardware, the relevant functions/components are connected using network switches.

Synchronizing the Clocks of the Real-Time PCs with PTP 1588

The Precision Time Protocol (PTP, defined in IEEE 1588 and included as part of IEC 61588) is used to synchronize the clocks. It enables devices in a network to be synchronized with greater precision. With PTP-capable network cards, time deviations are under 1 μ s. With software implementations, the deviations are under 1 ms.

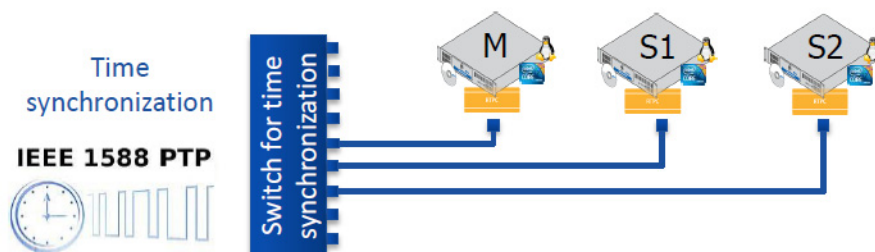


Fig. 3-46 Clock Synchronization with PTP

Real-Time Data Exchange between the Real-Time PCs

Data is exchanged between the models/projects run on the various Real-Time PCs using real-time interfaces - a library for UDP data transfer in real time is part of the delivery scope of ETAS RTPC.

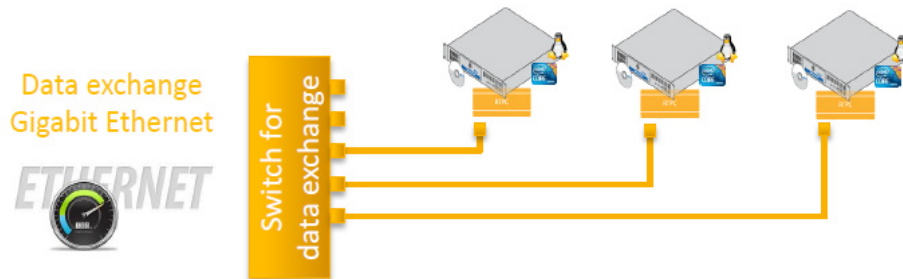


Fig. 3-47 Data Exchange between Real-Time PCs

The possible transfer rates vary according to the number of PCs and message type (point-to-point, multicast / broadcast).

Hardware Requirements

This results in the following hardware requirements for a Multi-RTPC network:

- As many Real-Time PCs as necessary – each one equipped with a PTP-capable Ethernet board each with at least three Ethernet ports.
- Three Ethernet switches each with as many ports as the "number of Real-Time PCs + 1".
- Ethernet cable (2 x number of Real-Time PCs + 1)
- A user PC with LABCAR-OPERATOR installed on it

3.16.1 Connecting Hardware

Switch for Connection between the Real-Time PCs and the User PC

Connect the eth0 port of every Real-Time PC with the corresponding switch and connect it to the user PC.

Switch for Time Synchronization (via PTP) between the Real-Time PCs

Connect one Ethernet port each of every Real-Time PC (e.g. eth1) with the relevant switch.

Switch for Data Transfer between the Real-Time PCs

Connect one Ethernet port each of every Real-Time PC (e.g. eth2) with the relevant switch.

3.16.2 Specifying IP Addresses of the Real-Time PCs

The Ethernet adapter "eth0" is used for the connection to the user PC.

Note

Follow the instructions below to specify IP addresses for all Real-Time PCs in succession!

- Connect the relevant Real-Time PC to the user PC.
- Start the Real-Time PC.
- Open the ETAS RTPC web interface and navigate to the section "RTPC Configuration".

RTPC Configuration

Host Ethernet Configuration (ETH0).
 The Ethernet adapter eth0 is used to connect the host to the RTPC.
 Note: Changes of these parameters may lead to an unaccessible RTPC!
 Any change requires a reboot to be effective. [\(Help\)](#)

Eth	IP Address	Netmask	DHCP	Ethernet Negotiate
ETH0	192.168.40.30	255.255.255.0	no	auto

Note

192.168.40.14 is a "standard address" for a Real-Time PC – assign addresses from 192.168.40.30 onward!

- To ensure a better overview, names should be assigned to the individual Real-Time PCs.

General Parameters

Parameter	Value
RTPC_POWER_UP_MODE The power up operation mode of RTPC. Help	simulate
RTPC_LOG_LEVEL Filter the log messages from RTPC. Help	warning
RTPC_NAME An user defined name of the RTPC. Help	192.168.40.30 - RTPC1

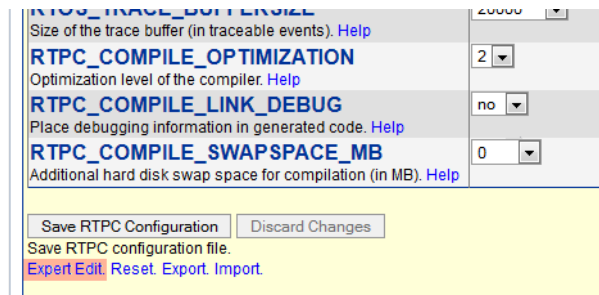
- Reboot RTPC ([Main Page](#) → [Power Control](#) → [Reboot RTPC](#)).

3.16.3 [Configuring the PTP and Data Connections in the Web Interface](#)

To enable PTP in ETAS RTPC, proceed as follows:

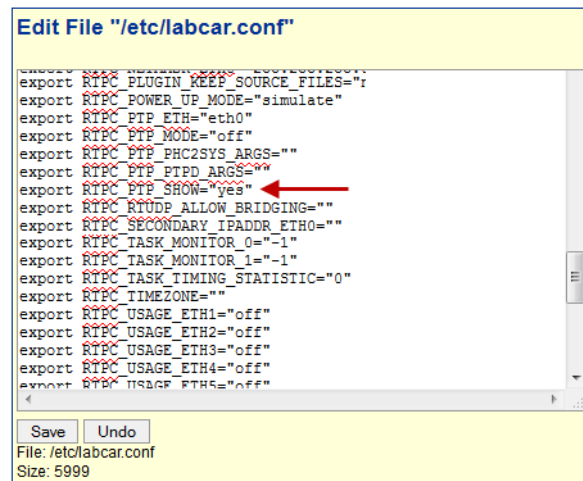
To enable PTP

- Navigate in the ETAS RTPC web interface to the section "RTPC Configuration".
- Click **Expert Edit**.



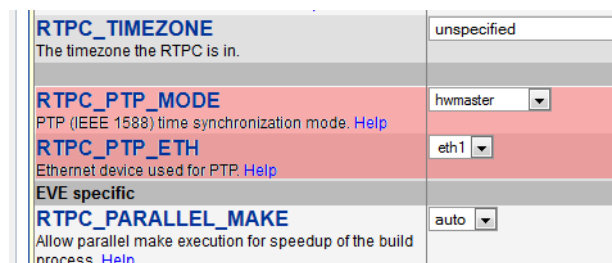
The configuration file opens in the editor.

Configuration >> Expert Edit



- Set the "RTPC_PTP_SHOW" parameter to "yes".
- Click **Save** and return to **Configuration**.

Two additional parameters are now visible.



- For further information click **Help**.

To enable PTP and data connections in ETAS RTPC, proceed as follows:

To configure PTP and data connection

- For "RTPC_PTP_MODE" select the option "hwmaster" for the PTP-Master and "hwslave" for the other Real-Time PCs.
- Under "Realtime Ethernet Configuration", assign the port for the PTP switch an IP address and "Usage" = "up".

In the example below this is "ETH1" with the IP address 192.168.50.30.

Realtime Ethernet Configuration.
 The realtime capable Ethernet adapters are used to connect external devices to the RTPC.
 (Help)

Eth	Usage	IP Address	Ethernet Negotiate
ETH1 (Blink)	up	192.168.50.30	auto
ETH2 (Blink)	rtudp_0	192.168.60.30	auto

Show options to control the order of Ethernet adapters based on MAC addresses.
 Note: Changing these values may lead to an unaccessible RTPC!

- The "Usage" for the Ethernet port for data exchange (here: ETH2, 192.168.60.30) should be set to "rtudp_0" for all Real-Time PCs.

Note

As all PTP and data ports are each connected with the same switch, these must have different IP addresses. It is helpful if the addresses of all ports on a Real-Time PC have the same end digit (here: 30).

- Click **Save RTPC Configuration.**
- Reboot.

Fig. 3-48 on page 261 is a summary of the configuration parameters.

RTPC Configuration

Host Ethernet Configuration (ETH0).
 The Ethernet adapter eth0 is used to connect the host to the RTPC.
 Note: Changes of these parameters may lead to an unaccessible RTPC!
 Any change requires a reboot to be effective. [\(Help\)](#)

Eth	IP Address	Netmask	DHCP	Ethernet Negotiate
ETH0	192.168.40.30	255.255.255.0	no	auto

Realtime Ethernet Configuration.
 The realtime capable Ethernet adapters are used to connect external devices to the RTPC. [\(Help\)](#)

Eth	Usage	IP Address	Ethernet Negotiate
ETH1 (Blink)	up	192.168.50.30	auto
ETH2 (Blink)	rtudp_0	192.168.60.30	auto

Show options to control the order of Ethernet adapters based on MAC addresses.
 Note: Changing these values may lead to an unaccessible RTPC!

General Parameters

Parameter	Value
RTPC_POWER_UP_MODE The power up operation mode of RTPC. Help	simulate
RTPC_LOG_LEVEL Filter the log messages from RTPC. Help	waming
RTPC_NAME An user defined name of the RTPC. Help	192.168.40.30 - RTPC1
RTPC_TIMEZONE The timezone the RTPC is in.	unspecified
RTPC_PTP_MODE PTP (IEEE 1588) time synchronization mode. Help	hwmaster
RTPC_PTP_ETH Ethernet device used for PTP. Help	eth1

Connection to Host

PTP Port Data Port

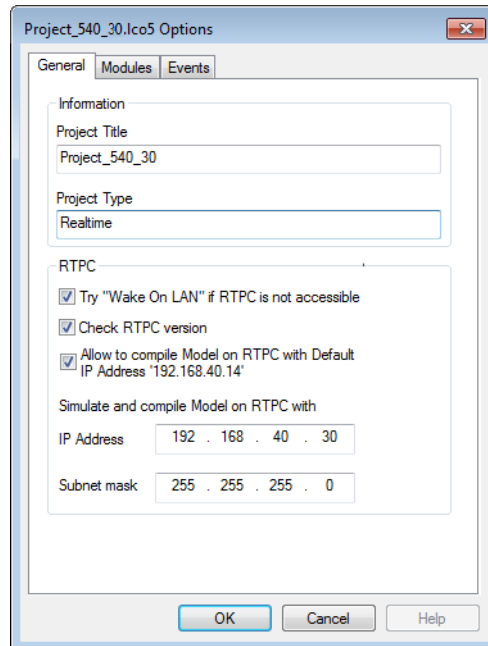
PTP Settings

Fig. 3-48 Settings for a Real-Time PC in the Web Interface

3.16.4 Creating LABCAR-OPERATOR Projects

A LABCAR-OPERATOR project must be created for every Real-Time PC in the network.

Under **Project → Options**, you can set the IP address of the Real-Time PC on which the project is to be created and run.



If network targets are not available, the "Allow to compile..." option permits project compilation on the "standard target" with the IP address 192.168.40.14.

Note

Please note that the modules of all projects (including the hardware modules) must have different names.

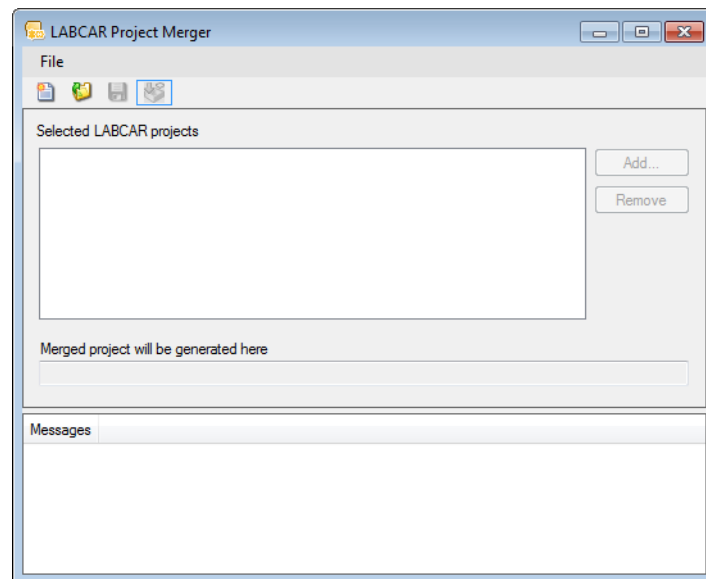
3.16.5 Merging Projects

To merge the individual projects, proceed as follows:

To create a project configuration

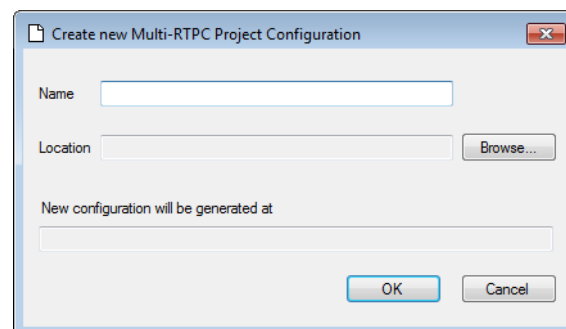
- Start the program `<LABCAR-OPERATOR Installation directory>/Project-Merger/ProjectMerger.exe`.

The LABCAR Project Merger is launched.



- Select **File** → **New**.

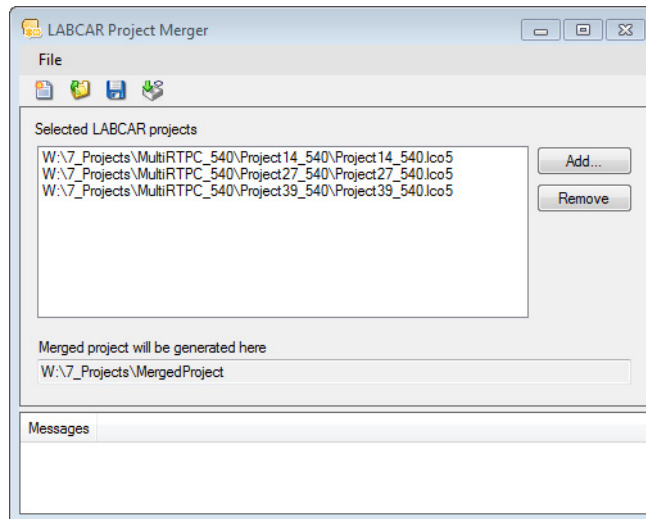
The "Create New Multi-RTPC Project Configuration" window opens.



- Specify a name and a directory for the project configuration.
- Click **OK**.

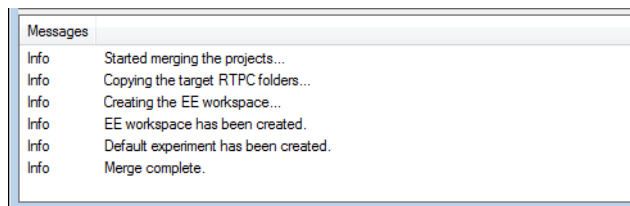
In the selected directory path, a directory with the specified name is created and the configuration file `<Name>.multirtpc` is created in it.

- Use **Add** to select the LABCAR-OPERATOR projects you want to merge to create a Multi-RTPC project.



- Select **File** → **Save** to save the configuration.
- Select **File** → **Merge** to generate the Multi-RTPC project.

The project is created and information (particularly error messages) is listed in the "Messages" section.



4 **ETAS Experiment Environment - an Overview**

ETAS Experiment Environment (ETAS EE) is used to execute a LABCAR-OPERATOR experiment. This chapter provides an overview of the GUI and the functions of ETAS EE.

Note

This chapter is only intended to provide an overview – for details, please refer to the online help in ETAS EE.

The followings sections contain information on:

- "The Constituent Parts of the GUI" on page 266
- "The Experiment Explorer" on page 268
- "The "Workspace Elements" Window" on page 270
- "The Main Workspace" on page 276
 - "The "Instrumentation" Tab" on page 276
 - "The "Datalogger" Tab" on page 278
 - "The "Signal Generator" Tab" on page 282
 - "The "Instruments" Window" on page 300
 - "Adding Signals to the Signal List" on page 301
- "The Script Recorder" on page 303
- "Working with Parameters" on page 306
- "Editing Parameter Files with LABCAR-PA" on page 326
- "LABCAR-CCI V5.4.4 (Calibration Connector for INCA)" on page 362

4.1 The Constituent Parts of the GUI

If you start an experiment from LABCAR-IP in the experiment environment, the ETAS EE GUI looks similar to this.

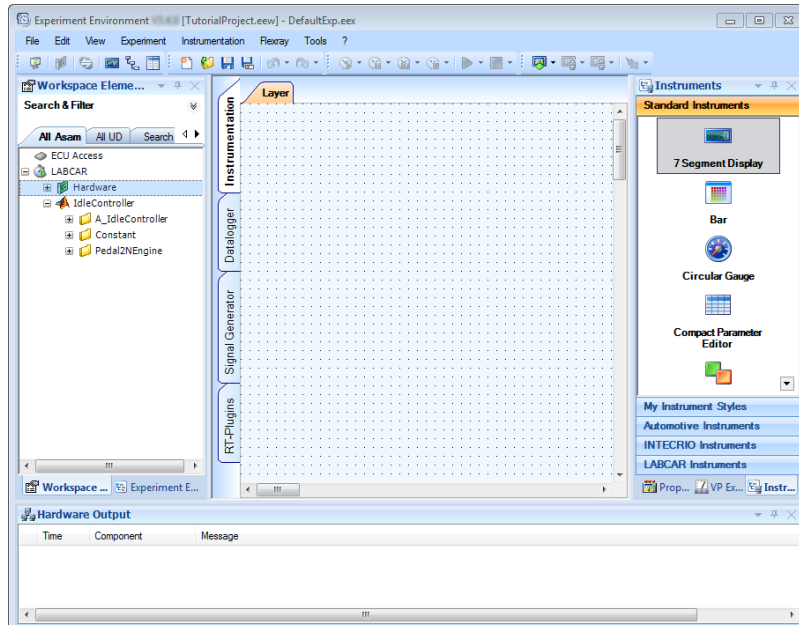


Fig. 4-1 The ETAS EE GUI

The individual parts of this GUI are described below.

4.1.1 The "Workspace Elements" Window

All measure variables, parameters, signal inputs and outputs of the modules of the experiment are made accessible in the "Workspace Elements" window.

In addition, this window also provides filter functions relating to hierarchies, element, data and object type, and search functions with regular expressions.

4.1.2 The Experiment Explorer

The Experiment Explorer contains all files required for configuration or when running an experiment, such as parameter files, configuration files for the Datalogger, signal generator sets etc..

4.1.3 The Main Workspace

The important GUIs of the experiment environment are made accessible in the main workspace (via several tabs).

The "Instrumentation" Tab

This is where you create the layers which in turn contain the various instruments (combined in groups) for executing the experiment.

The "Signal Generator" Tab

The signal generators of the experiment are configured and operated in this tab.

The "Datalogger" Tab

The Datalogger is configured and operated in this tab.

4.1.4 The "Instruments" Window

This window contains all the instruments you can use in the experiment.

4.1.5 The "Properties" Window

This window displays the properties and meta data of every object selected in the GUI, e.g. parameters or instruments.

4.1.6 The "Application Log" and "Hardware Output" Windows

These windows display information, warnings and error messages from the application.

4.1.7 The Main Menu of ETAS EE

Note

The following description of the menu structure of ETAS EE is only intended to provide an overview – for more detailed information on the individual functions, refer to the online help on ETAS EE.

The "File" Menu

This menu includes all file-specific actions (workspace and experiment).

The "Edit" Menu

Provides "Undo" and "Redo" functions.

The "View" Menu

This menu allows you to hide or make visible the component parts of the GUI described above. You can also maximize the screen size (<F11>).

The "Experiment" Menu

This menu combines all functions to do with downloading the simulation code to the target and controlling the simulation itself.

The "Instrumentation" Menu

This menu contains functions for managing the layers in the "Instrumentation" tab.

The "Tools" Menu

The various options for the experiment are accessible here.

The "?" Menu

This menu provides you access to the online help, licensing and to program and contact information.

4.1.8 The Toolbar

The toolbar consists of four individual groups.

File

Contains functions for managing experiments (New, Open etc.)

Experiment

Contains functions for downloading and connecting to the target.

Simulation

Contains functions for controlling simulation and measuring.

Instrumentation

Contains functions which are frequently used when working with instruments.

4.2 The Experiment Explorer

The Experiment Explorer contains all files required for configuration or when running an experiment, such as parameter files, configuration files for the Datalogger, signal generator sets etc..

This data can be activated, edited or removed from the experiment again here.

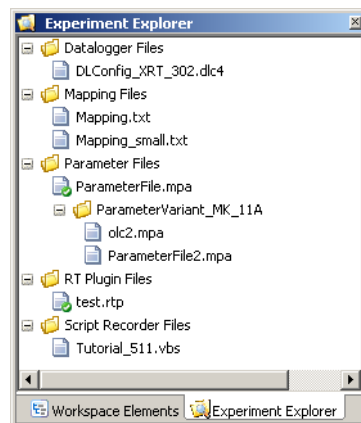


Fig. 4-2 The Experiment Explorer

4.2.1 Working with the Experiment Explorer

Shortcut menus (right-click) of the relevant objects are used to work with folders and files of the Experiment Explorer.

Functions for all Folders

- **Add File**

Opens a file selector window to select a file to be added

Functions for Specific Folders

"Parameter Files" folder:

- **Set Display Order**

Opens a dialog box in which the order can be determined in which the existing parameter files are displayed (i.e. downloaded to the target).

- **Download all active files**
Downloads all active parameter files to the target (when there is a connection to an experiment).
- **Add New Parameter Variants**
Creates a subfolder and adds a parameter file (variant) to it.

Functions for all Files

- **Add File**
Opens a file selector window to select a file to be added
- **Rename File**
Renames the particular file
- **Exclude From Experiment**
File is excluded (not deleted) from the experiment
- **Delete (permanently)**
File is excluded from the experiment and deleted

Functions for Specific Types of Files

- **Edit File**
 - Files in the "Mapping Files" folder (*.txt, *.smf)
Opens the mapping file in the SUT Mapping File Editor
 - Files in the "Parameter Files" folder
 - *.mpa: Opens the parameter file in LABCAR-PA (Parameterization Assistant) (see "Editing Parameter Files with LABCAR-PA" on page 326)
 - *.dcm: Opens a text editor for editing the file
- **Active**
This marks a file as being "active". This file is downloaded in this process and in the future is always downloaded with all other active files when the experiment is downloaded to the target.
 - Files (*.dlc4) in the "Datalogger Files" folder
Activates the selected Datalogger configuration
 - Files (*.txt) in the "Mapping Files" folder
Activates the selected mapping file
 - Files (*.mpa, *.dcm *.cdfx or *.olc4) in the "Parameter Files" folder
Activates the selected parameter file or open-loop configuration
- **Set Display Order**
 - Files in the "Parameter Files" folder
Opens a dialog box in which the order can be determined in which the existing parameter files are displayed (i.e. downloaded to the target).
- **Reload Mapping**
 - Files in the "Mapping Files" folder
Updates the mapping according to the content of the selected file

4.3 The "Workspace Elements" Window

The "Workspace Elements" window is one of the most important windows in the experiment environment ETAS EE. The organization of the objects in this window takes place in folders for

- Simulink and ASCET models
- C modules (incl. signal conversion modules)
- CAN and FlexRay modules
- FiL modules
- Hardware
- ECU Access

In these folders, you have

- access to all parameters and measure variables
- access to all inputs and outputs of the modules (signals)
- access to hardware and ECU ports

and you can influence the view by setting filters and selecting search criteria.

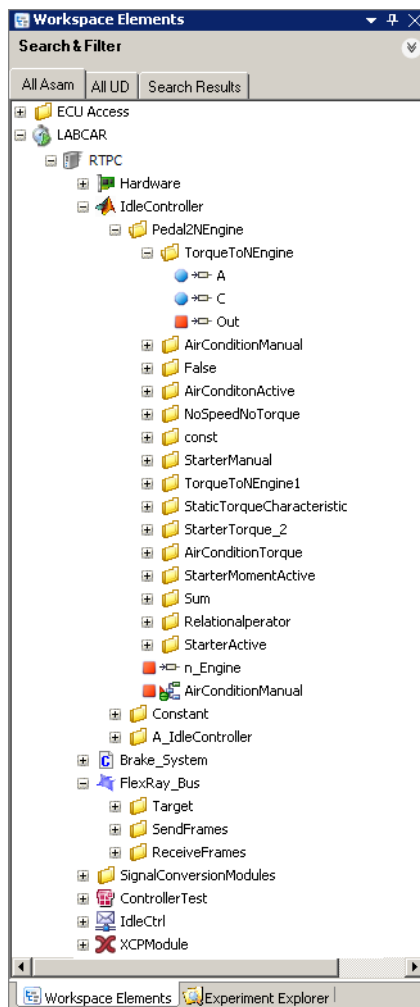


Fig. 4-3 The "Workspace Elements" Window

4.3.1 The Tabs of the "Workspace Elements" Window

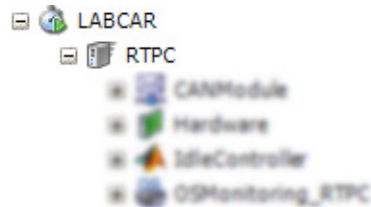
The "Workspace Elements" window has three tabs which contain different views of the elements:

- "All ASAM" Tab
Standard tree view with all elements in hierarchies which are based on the ASAM label or the module specification.
- "All UD" tab
In this tab, the display is as described above, although the user-defined names from the mapping file are used here.
- "Search Result" Tab
This tab displays the results of the last search (see "Search and Filter Function" on page 275).

4.3.2 Working with the Workspace Elements

In Fig. 4-3 on page 271 different elements are shown with different icons.








The "LABCAR" element is the top layer, followed by the layer that symbolizes the Real-Time PC (here: "RTPC") the project is assigned to (several Real-Time PCs can exist in a multi-RTPC project).



The various project modules are under the relevant Real-Time PC.





Modules









The different types of modules are shown using the following icons:

-  • ASCET and MATLAB/Simulink modules
-  • C Code modules
-  • CAN, LIN and FlexRay modules
-  • CAN, LIN and FlexRay modules
-  • FiL modules
-  • Modules for open-loop access and signal conversion
-  • I/O hardware module

Measure Variables and Parameters of Modules

The measure variables and parameters of the individual modules are shown using the following icons:

-  • Measure variable
-  • Parameter
-  • Module output (measure variable)
-  • Module input (measure variable)

-  • Hardware output (to module) (measure variable)
-  • Hardware input (from module) (measure variable)
-  • Hardware pin (output)
-  • Hardware pin (input)
-  • ECU pin (output)
-  • ECU pin (input)
-  • CAN send message
-  • CAN receive message

Note

How to work with parameters is described in detail in "Working with Parameters" on page 306.

Shortcut Menus

For working with the elements of the window, there is a shortcut menu for every element which has a varying number of functions according to the type of element:

- **Measure / Calibrate**
Opens the "Create Instruments" dialog box to assign this element to an existing instrument or an instrument to be newly created
- **Find in Instruments**
Determines whether measurement variables/parameters have been assigned to an instrument and outputs the result in the "Application Log" window. The layer the measure variable or parameter is found in can be displayed in the foreground by double-clicking the relevant element.
- **Show All HW Ports**
It is unusual for all hardware signals (e.g. monitoring signals or signals for creating measure modes) to be displayed in the "Workspace Elements" window. This option ensures they are all displayed.
- **Add to Datalogger**
Adds the selected element to a Datalogger. If several Dataloggers have been configured, a specific Datalogger can be selected.

- **Calibration / Parameters**

These functions are only available with parameters and after the experiment has been downloaded to the target.

 - **Copy Values**

Copies the value of the parameter to the clipboard
 - **Paste Values**

Assigns the value in the clipboard to the selected parameter
 - **Save**

Opens a dialog box to save parameters in mpa files. Several parameters are selected using <SHIFT>.
 - **Set State To Modified/Unmodified**

Changes the state of a parameter from "modified" (red) to "unmodified" (black)
- **Signal**

This function is only available with inputs of modules and hardware and has the following submenus:

 - **Trace**

Opens the dialog box for signal tracing
 - **Save Input Signal Settings**

Saves the settings and values for an input in a file (*.olc4)
 - **Reset Full Path to "Model"**

Sets the full path with the relevant signal to "MODEL" to close any open loops
 - **Reset Input Setting To**

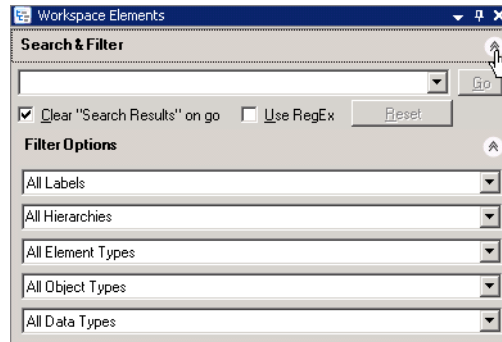
To change the OLC settings, a specific value ("MODEL", "CONST", etc.) can be selected in a dialog box
- **CAN**

During simulation, sending can be enabled or disabled with **Disable/Enable all Messages** (in the context of CAN messages)
- **Reload Mapping**

Reloads the active mapping file

4.3.3 Search and Filter Function

In the "Workspace Elements" window, elements can be searched for or a filter can be applied to the view



The Search Function

To run a search, enter the search text and click **Go**. Those elements which correspond to the search criteria are displayed in the "Search Results" tab.

If you activate the option **Clear Search Results on Go**, the list of search results is deleted from the "Search Results" tab for every new search, otherwise the results are added to the list. With the option **Use RegEx** you can use regular expressions for the search patterns.

The Filter Function

With the filter function, the list can be filtered according to various criteria and properties:

- Label Type
 - Filters according to label type
 - All Labels
 - All ASAM
 - All UD
- Hierarchy
 - Only shows the selected part of the hierarchy
- Element Type
 - Filters according to element type
 - Inputs
 - Outputs
 - Parameters
 - Measure Elements
 - Pins
- Object Type
 - Filters according to object type
 - Scalar
 - Array
 - Matrix

- 1D-Table
- 2D-Table
- Data Type
 - Filters according to data type
 - Logical
 - Discrete
 - Unsigned Discrete
 - Continuous
 - Enumeration

Saving Results in a New Tab

You can save the results shown in the "Search Results" tab in a new tab; to do this, right-click on the tab title, "Search Results", and select **Export Result to New Tab**. You can also specify the name of the new tab.

4.4 The Main Workspace

Three main functions are spread over four tabs in the main workspace of ETAS EE:

- "The "Instrumentation" Tab" on page 276
This is where the instrumentation (display and control elements) of the experiment is created and operated.
- "The "Datalogger" Tab" on page 278
This is where the Datalogger is configured and operated.
- "The "Signal Generator" Tab" on page 282
In this tab signal generators are created, managed and controlled.
- "The "RT-Plugins" Tab" on page 300
In this tab RT-Plugins are managed.

4.4.1 The "Instrumentation" Tab

The "Instrumentation" tab consists of one or more layers which in turn contain groups of display and control elements.

Layer

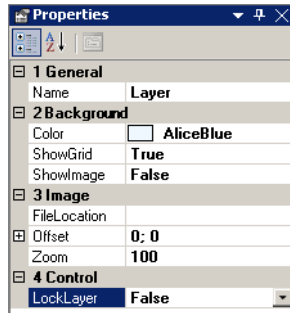
If you right-click the title of a layer, a shortcut menu opens which contains the following functions:

- **Create Layer**
Creates a new layer
- **Import Layer**
Creates a layer on the basis of a previously exported layer
- **Export Layer**
Saves a layer configuration in a file

- **Rename Layer**
Renames a layer
- **Delete Layer**
Deletes the selected layer

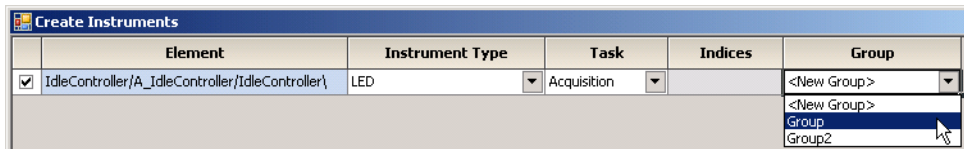
Layer Properties

If you select the title of a layer, its properties are displayed in the "Properties" window and can also be edited there.



Groups

If you want to display measure variables or calibrate parameters using the **Measure / Calibrate** function of the shortcut menu of the "Workspace Elements" window, you can select in the "Create Instruments" dialog box whether you want to create a new group for the instrument or whether you want to add it to an existing group.



If you select the group title and right-click, a shortcut menu opens containing the following functions:

- **Bring to Front**
Brings the group to the front
- **Send to Back**
Sends the group to the back
- **Cut / Copy / Paste**
Cuts/copies a group of a layer and pastes it to another layer
- **Move to Layer**
A different layer can be selected here to which the group can be moved
- **Hide Group Frame / Show Group Frame**
Hides a group frame or shows a group frame again
- **Delete**
Deletes the selected group

Group Properties

If you select the title of a group, its properties are displayed in the "Properties" window and can also be edited there.

Instruments

An instrument is added to a group/layer either by drag-and-drop from the "Instrumentation" window or in the context of the "Workspace Elements" window.

Properties

If you select the title of an instrument, its properties are displayed in the "Properties" window and can also be edited there.

4.4.2 The "Datalogger" Tab

The "Datalogger" tab is used to create and configure Dataloggers.

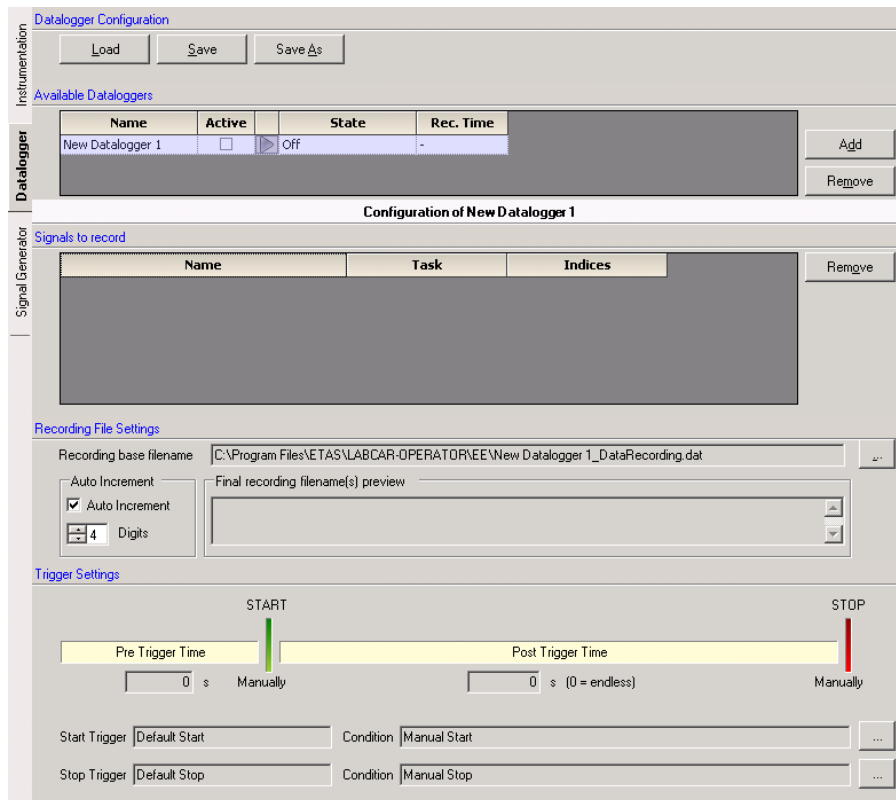


Fig. 4-4 The "Datalogger" Tab

Below, you will find a description of the operating possibilities open to you in this tab.

The "Datalogger Configuration" Field

This is where you can save the entire configuration shown of the Dataloggers and load configurations saved previously – these (file extension *.dlc4) are primarily managed in the "Datalogger Files" folder in the Experiment Explorer.

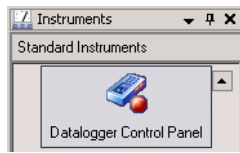
The "Available Dataloggers" Field

This is where all available Dataloggers are displayed – use **Add** to create a new one; **Remove** to remove a Datalogger.

Note

*A **maximum of two** Dataloggers can be created! One of these is used to record variables from the modules of the LABCAR-OPERATOR project – a second can be used for the remote control of the INCA Datalogger (if LABCAR-CCI V5.4.4 (Calibration Connector for INCA) is being used).*

The actual operation of a Datalogger does not, however, take place here but from a layer of the "Instrumentation" tab. Create a "Datalogger Control Panel" instrument there (from the list of "Standard Instruments").



The display and operating possibilities in this instrument are then the same as those in the "Datalogger" tab.

Group12				
	Name	Active	State	Rec. Time
✓	New Datalogger 1	<input type="checkbox"/>	Off	-

The green tick in the first column shows that the Datalogger is "valid" – this is the case when a signal is added to the "Signals to record" list (see below).

If the experiment is running, the Datalogger must be activated in the "Active" field. It can then be started manually by clicking the blue triangle, or alternatively starts when the trigger condition occurs.

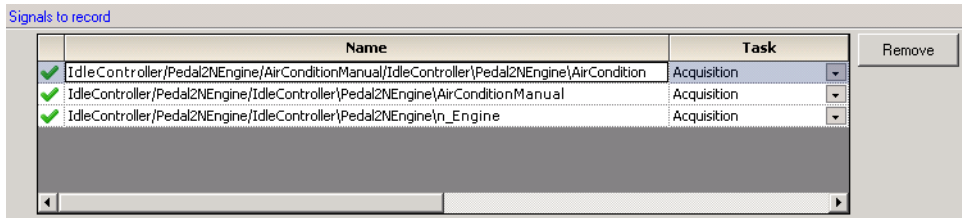
Group12				
	Name	Active	State	Rec. Time
✓	New Datalogger 1	<input checked="" type="checkbox"/>	WaitingForTrigger	-

When the Datalogger is running, the blue triangle becomes a square with which the Datalogger can also be stopped manually again.

The current state of the Datalogger ("State") and (when the Datalogger is running) the recording time ("Rec. Time") are also displayed.

The "Signals to record" Field

This list contains the signals to be recorded by the relevant Datalogger which can be added from the "Workspace Elements" window by drag-and-drop. In addition, a task for recording can be assigned to every signal here.



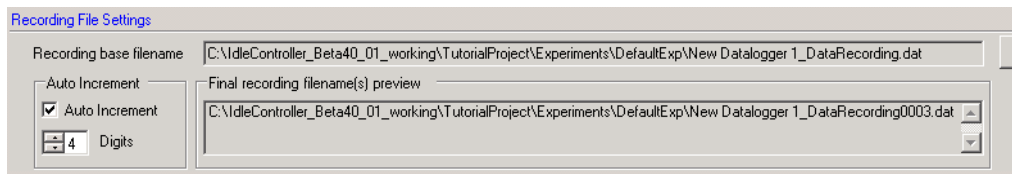
Signals are removed by selecting the signal and clicking **Remove**.

The "Recording file settings" Field

This is where the settings for the names of the log files are determined. This happens by

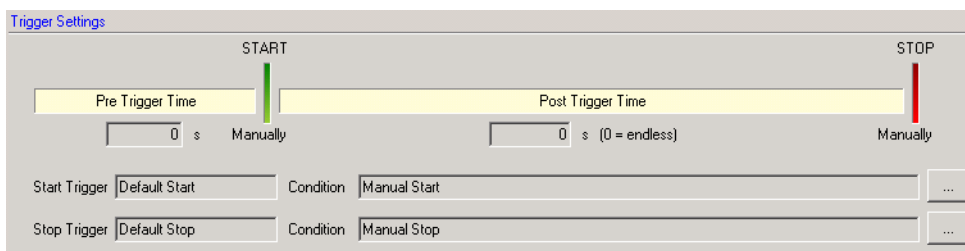
- defining a "template" file in a specific path ("Recording base filename") and
- agreeing a convention that a number added to this file name is incremented ("Auto Increment").

This leads to the file name used for the next recording ("Final recording filename preview").



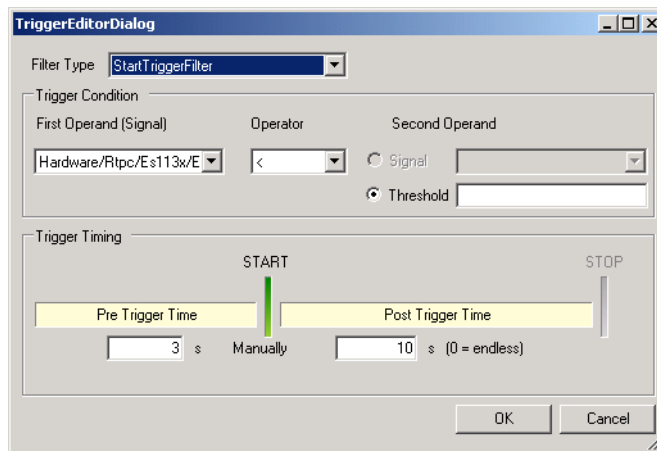
The "Trigger Settings" Field

This is where the conditions (trigger) for the automatic start of the Dataloggers is determined.



To specify the trigger condition desired for starting and stopping the Datalogger, click the relevant ... buttons at the bottom right.

In the following dialog box, make the settings for, for example, the start trigger.



- **Filter Type**
Manual or trigger condition
- **Trigger Condition**
 - **First Operand**
The signal whose value initiates the trigger
 - **Operator**
The operator for the reference value
 - **Second Operand**
The signal or value with which the trigger signal is compared
- **Trigger Timing**
 - **Pre-Trigger Time**
The time before the trigger condition occurs in which the signal is also recorded
 - **Post-Trigger Time**
The time in which the signal is recorded after the occurrence of the trigger condition (0 means endless, i.e. until a stop trigger occurs or until a manual stop)

4.4.3 The "Signal Generator" Tab

In this tab, signal generators can be created, managed and operated.

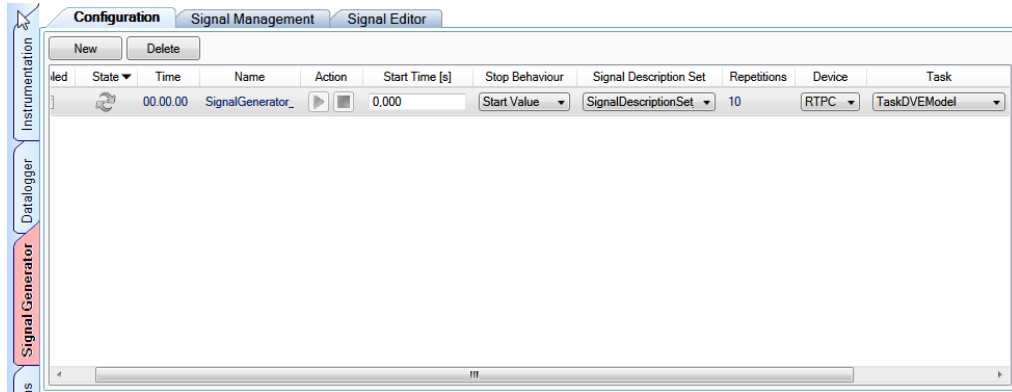


Fig. 4-5 The "Signal Generator" Tab

In turn, the signal generator user interface itself has three tabs:

- **Configuration**
In this tab, signal generators can be generated, configured and controlled (see "The "Configuration" Tab" on page 282).
- **Signal Management**
In the "Signal Management" tab, the signal description sets for the signal generators are generated and configured (see "The "Signal Management" Tab" on page 285).
- **Signal Editor**
This is where segments of the signal descriptions can be edited¹ (see "The "Signal Editor" Tab" on page 295).

The "Configuration" Tab

In this tab, signal generators can be generated, configured and controlled.

To create a signal generator

- To create a signal generator, click **New**.
A signal generator is added to the list.
A signal generator with the name "SignalGenerator" is generated (or "SignalGenerator_1", "SignalGenerator_2" etc.)

¹. Only visible if the editing function for a signal description was selected.

To rename a signal generator

- Select the signal generator from the list and right-click.
- Select **Rename** from the context menu.

or

- Press <F2>.
The name can now be edited.

To delete a signal generator

- To delete a signal generator from the list, click **Delete**.
A signal generator is deleted.

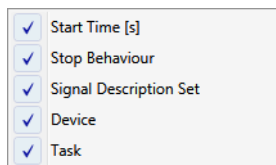
Properties and control possibilities:

Every signal generator in the list has properties and control possibilities which are described below.

Note

For information on how to operate the signal generators, refer to the section "To operate a signal generator" on page 299.

Right-clicking the column headings opens a context menu in which individual columns can be hidden.



The order of the columns can be modified using Drag-and-Drop.

- **Enabled**
Enables or disables the relevant signal generator.
- **State**
Shows the state of the signal generator with an icon.
 - **Invalid**
The icon and the icon **Start** ("Action" column) are grayed out. This is because the signal generator configuration is invalid (unsigned signals etc.)
 - **Stopped**
The icon is displayed and is static. The relevant signal generator can be started via the **Start** icon ("Action" column).
 - **Running**
The rotating icon shows a signal generator is in operation.

- **Paused**

The paused and flashing icon shows that the signal generator is paused – the last value of the signal generator continues to be output.
- **Name**

The name of the relevant signal generator.
- **Action**

In this column, you can control how the signal generator is run (see "To operate a signal generator" on page 299).

 - **Start**

Starts the signal generator – if a start trigger has been defined, the signal generator does not start until the trigger condition occurs.
 - **Stop**

Stops the signal generator.
 - **Pause**

Pauses the signal generator. The model is now stimulated with the last value. When the generator is restarted with **Play**, signal generation starts at the point at which it was paused.
- **Start Time**

The time after the start of the signal generator at which signal output begins (default: 0.000).
- **Stop Behaviour**

Defines the behavior of the relevant signal if the signal generator was paused.

The following settings are possible:

 - **Start Value** (Default)

Once paused, the start value of the signal description is output.
 - **0 (zero)**

Once paused, the value "0" is output.
- **Signal Description Set**

This is where the signal description set is selected. The set contains the description of the signals the signal generator is to output. All signal description sets in the "Signal Management" tab are available.
- **Repetitions**

The number of repetitions for the relevant signal description set.
- **Device**

The simulation target (RTPC)
- **Task**

The task in whose time pattern the calculation of the relevant signal generator is to take place.

The "Signal Management" Tab

In the "Signal Management" tab, the signal description sets for the signal generators are generated and configured.

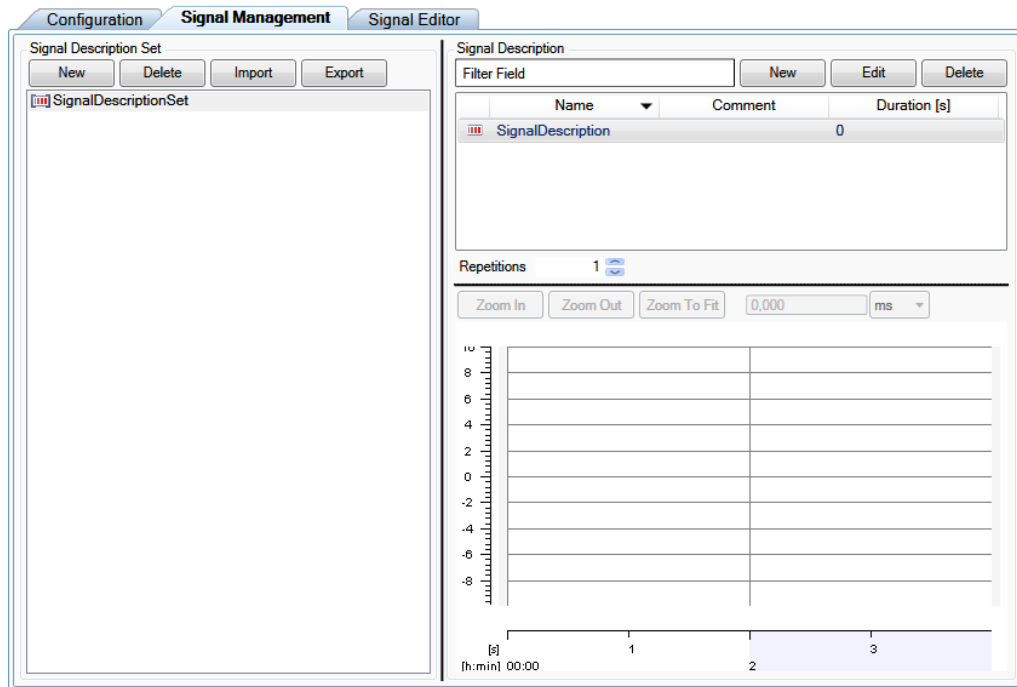


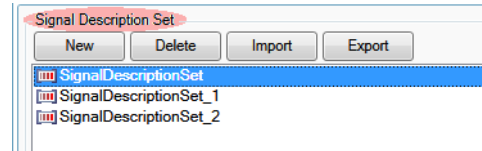
Fig. 4-6 The "Signal Management" Tab

It consists of two areas:

- The "Signal Description Set" area in which the signal description sets of the experiment are managed (see "**The "Signal Description Set" area:**" on page 286).
- The "Signal Description" area in which the signal descriptions of the set are managed (see "**The "Signal Description" area:**" on page 292).

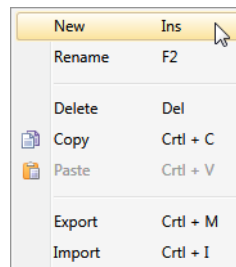
The "Signal Description Set" area:

In this area, the signal description sets of the current experiment are created and managed.



To create a signal description set

- To create a signal description set, click **New**.
- or
- Right-click the area and select **New** from the short-cut menu.



or

- Press <INS>.
The signal description set is generated.

To copy a signal description set

- To copy a signal description set, select it from the list.
- Right-click and select **Copy** from the shortcut menu.

or

- Press <CTRL+C>.
The signal description set is copied and can be inserted into the list of signal description sets using **Paste** (in the context menu) or <CTRL-V>.

To rename a signal description set

- To rename a signal description set, select it from the list.
- Right-click and select **Rename** from the shortcut menu.

or

- Press <F2>.

The name of the signal description set can now be changed

To delete a signal description set

- To delete a signal description set, click **Delete**.

or

- Right-click the area and select **Delete** from the shortcut menu.

or

- Press .

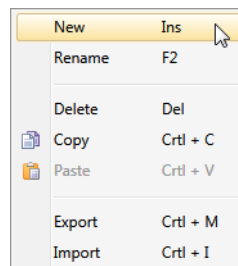
The signal description set is deleted.

To import a signal description set

- To import a signal description set, click **Import**.

or

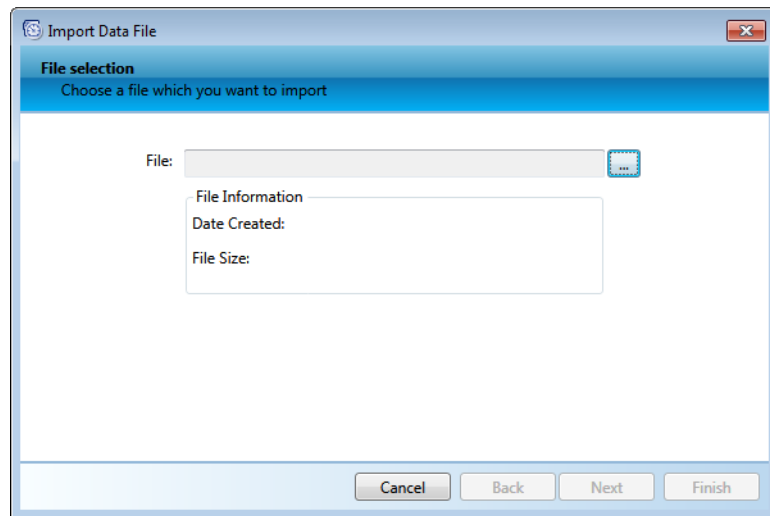
- Right-click the area and select **Import** from the shortcut menu.



or

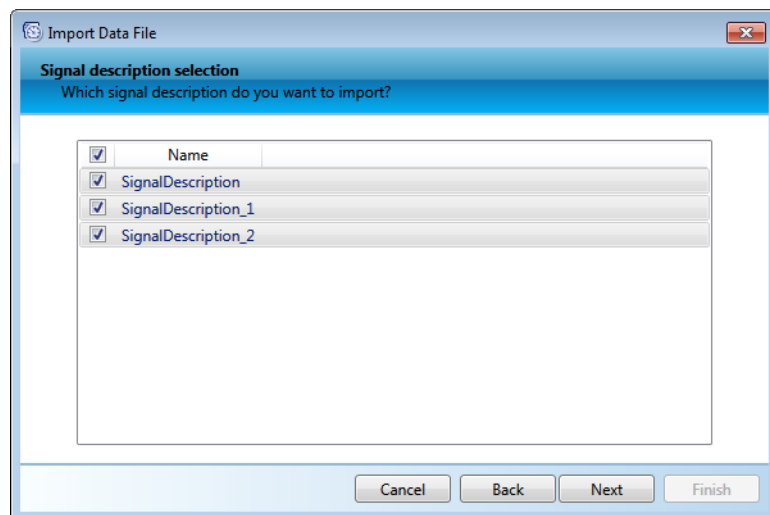
- Press <CTRL+I>.

The "Import Data File" window opens.



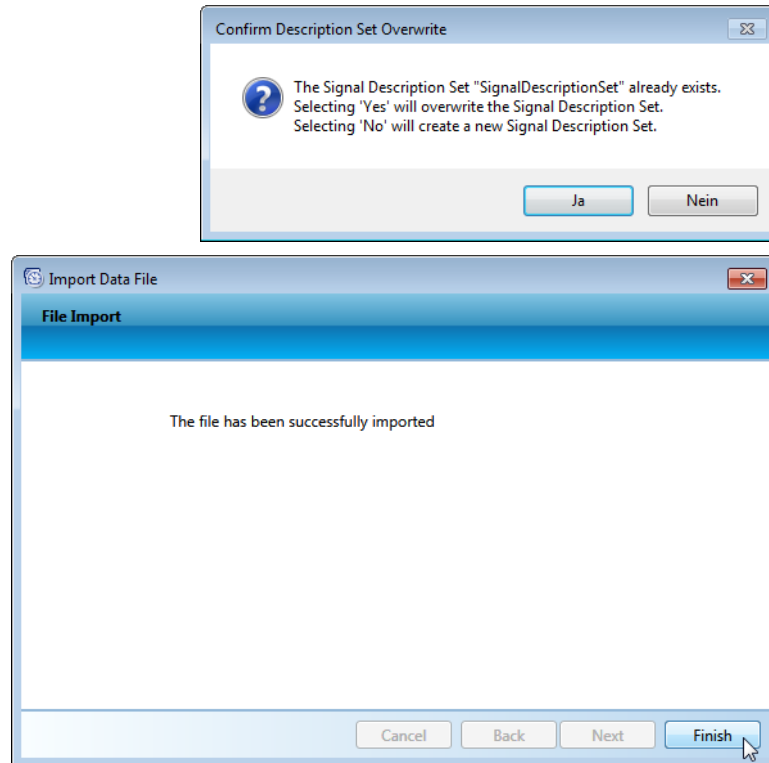
- Click ... to open a file selection window.
- Select a file of the following format and click **Next**.
 - ETAS Stimulus File (*.esti)
 - HIL API Stimulus File (*.sti)
 - Measurement Data File (*.dat)
 - LABCAR Stimulus File (*.lcs)
 - MAT-File (*.mat)

The "Import Data File" window opens.



Select the required signal descriptions.

If there is already a signal description set with the name "signal description set", you can choose whether it should be overwritten (**Yes**) or whether a set with a new name "signal description set_n" should be created (**No**).



- Click **Finish**.
The signal description set is created (or overwritten).

Importing a MATLAB file (*.mat)

When importing MATLAB *.mat files, two formats are supported:

- **Matrix-based**

Used for signals with different time scales. Each signal is a matrix – a MATLAB file can contain any number of $n \times 2$ matrices (n can be different for every matrix).

- To generate a signal generator set in MATLAB, enter, for example, the following code in the MATLAB operating window:

```
t1=0:0.1:10;
t2=0:0.5:30;
s1=sin(0.5*pi*t1)+1;
s2=3*cos(0.2*pi*t2);
X=[t1' s1'];
Y=[t2' s2'];
save C:\MatrixBasedFormat.mat X Y;
```

This example generates the MATLAB file `MatrixBasedFormat.mat` with the two signals (matrices) X and Y.

Signal 1:

0	2
0.5	3
1	4
1.5	5
2	8

Signal 2:

0	1
0.5	4
1	5
1.5	8
2	9

- **Interval-based (sequence format)**

Used for signals that have the same time scale.

Interval vectors contain the duration of a single interval, data vectors contain the signal values for these intervals. Data and interval vectors must have the same length - the name of the interval vector must be "Intervals".

- Enter, e.g., the following code in the MATLAB operating window:

```
Intervals = ones(size(-pi:0.01:pi));
X = sin(-pi:0.01:pi) + 1;
Y = 3 * cos(-pi:0.01:pi);
save C:\IntervalsBasedFormat.mat X Y
Intervals;
```

This example generates the MATLAB file `IntervalsBasedFormat.mat` with the two signal vectors "Signal1_Value" and "Signal2_Value".

Signal1_Value	2	3	4	5	8
Signal2_Value	1	4	5	8	9
Intervals	0.5	0.5	0.5	0.5	0.5

To export a signal description set

- To export a signal description set, select it from the list of signal description sets.

- Click **Export**.

or

- Right-click the area and select **Export** from the shortcut menu.

or

- Press <CTRL+M>.

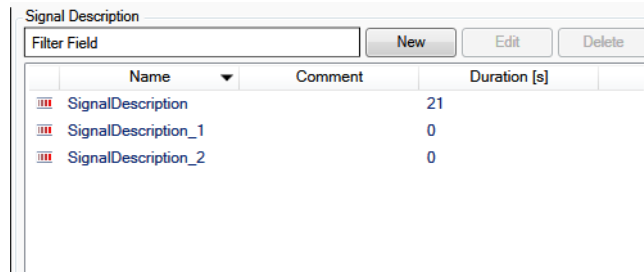
A file selection window opens.

- Select a file format (ETAS Stimulus File (*.esti) or HIL API Stimulus File (*.sti)), select a directory and assign a file name.
- Click **Save**.

The signal description set is saved in the selected format.

The "Signal Description" area:

This area lists the signal descriptions that are part of a signal description set.



- **Name**
Name of the signal description (see "**Name**" on page 295)
- **Comment**
Comment (see "**Comment**" on page 295)
- **Duration**
Total duration of the signal description (calculated from the length of the individual sequences).

The **Filter Field** makes it possible to search for signal descriptions using a filter expression. If this field is empty, all available signal descriptions are listed.

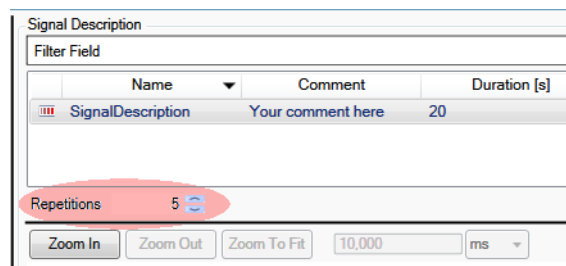
Every character (string) entered in this field is used for an incremental search in the names of the existing signal descriptions. If, for example, "abc" is entered, all signal descriptions starting with "abc" are listed.

To simplify the search, regular expressions with wildcards can also be used:

- The asterisk ("*") finds any character string (also including 0 characters).
"*_TRK" e.g. finds all names that end with "_TRK".
- The question mark ("?") finds exactly one character.
- "?TRK" e.g. finds all names which start with any letter followed by "TRK".

These match patterns are not case-sensitive (in other words do not make a distinction between upper and lower case).

With **Repetitions**, you can specify how often the sequence is run.

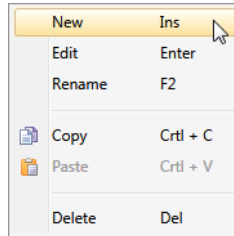


To create a signal description

- To create a signal description, click **New**.

or

- Right-click the area and select **New** from the short-cut menu.



or

- Press <INS>.
The signal description is generated.

To edit a signal description

- To edit a signal description, click **Edit**.

or

- Right-click the area and select **Edit** from the short-cut menu.

or

- Press <ENTER>.
The view toggles to the "Signal Editor" tab (see "The "Signal Editor" Tab" on page 295), in which the signals of the signal description can be edited.

To rename a signal description

- To rename a signal description, select it from the list.
- Right-click and select **Rename** from the shortcut menu.

or

- Press <F2>.
The name of the signal description can now be changed.

To copy a signal description

- To copy a signal description set, select it from the list.
- Right-click and select **Copy** from the shortcut menu.

or

- Press <CTRL-C>.

The signal description is copied and can be inserted into the list of signal descriptions with **Paste** (in the context menu) or <CTRL-V>.

To delete a signal description

- To delete a signal description, click **Delete**.

or

- Right-click the area and select **Delete** from the shortcut menu.

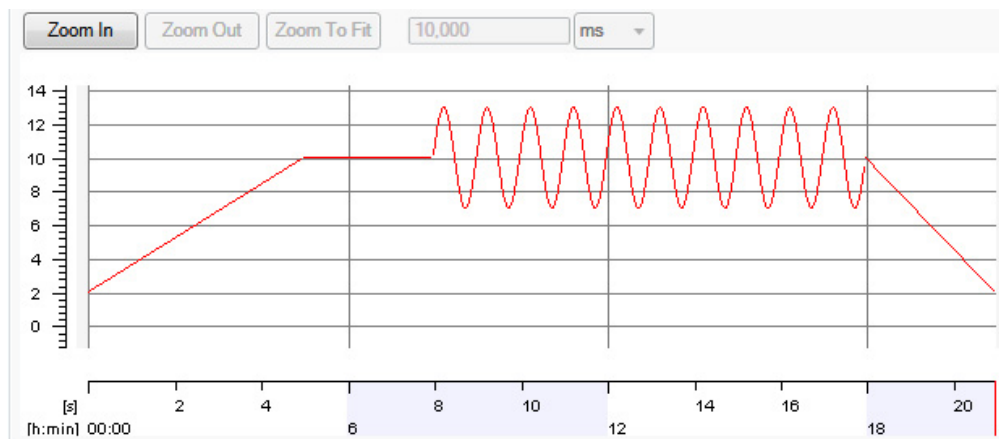
or

- Press .

The signal description is deleted.

The signal view:

The signal view provides an overview of the signal segments defined in this tab.



The axes can be moved by dragging them and keeping the mouse button pressed down and be enlarged or reduced by keeping the <CTRL> key pressed down.

The signal description selected in the list is displayed. Use the <CTRL> key for a multiple selection – the signal traces are then shown in different colors.

Axis scaling can also be modified using **Zoom In/Zoom Out – Zoom To Fit** adapts the coordinate system to the signal trace.

The "Signal Editor" Tab

The "Signal Editor" tab is displayed if you select a signal description from the list of signal descriptions (in the "Signal Management" tab) and either click **Edit** or double-click it.

The "Signal Editor" tab then opens. This is where the signal trace is created and can be edited.

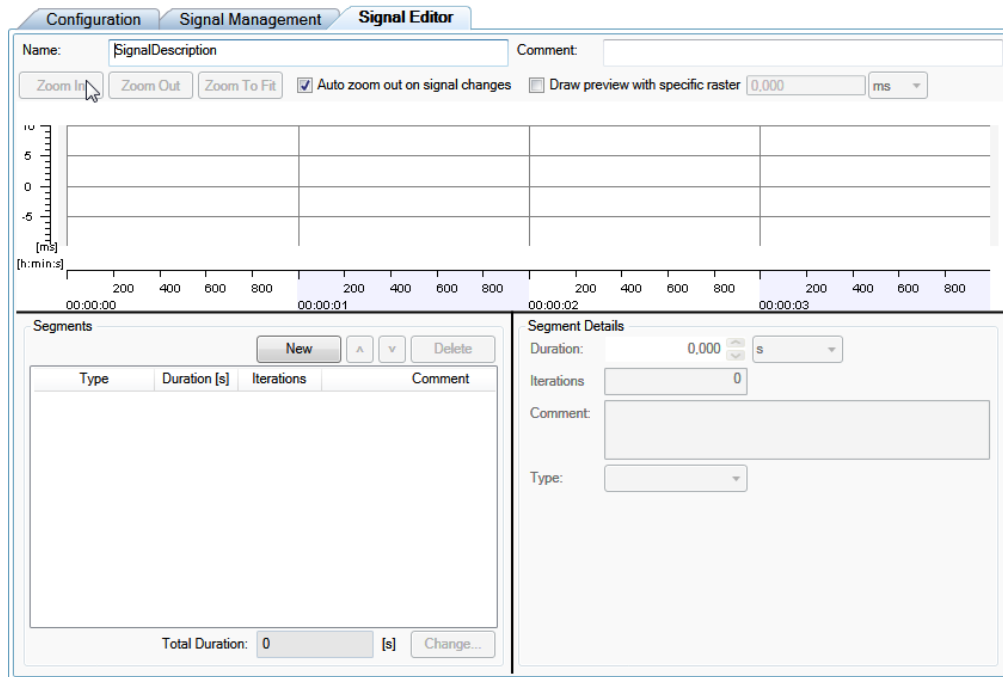


Fig. 4-7 The "Signal Editor" Tab

- **Name**
This is where a name can be assigned to the signal description.
- **Comment**
This is where a comment can be assigned to the signal description.

The signal view: T

The signal view provides an overview of the signal segments that can be edited in this tab.

The signal description selected in the list is displayed. Use the <CTRL> key for a multiple selection – the signal traces are then shown in different colors.

Axis scaling can also be modified using **Zoom In/Zoom Out – Zoom To Fit** adapts the display to the signal trace. The axes can be moved by dragging them and keeping the mouse button pressed down and be enlarged or reduced by keeping the <CTRL> key pressed down.

- **Auto Zoom out on Signal Changes**
Ensures the coordinate system is adapted automatically when segments are modified.
- **Draw Preview with Specific Raster**
This is where you can specify a particular division of the time axis.

The "Segments" area:

In this area, the properties and sequence of the individual signal segments are shown:

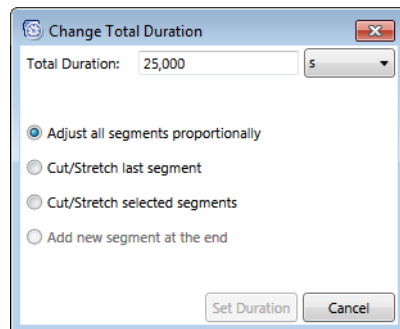
- **Type**
This is where the different segment types are managed. If a segment is selected, it can be edited in the "Segment Types" area.
- **Duration**
The duration of the segment (is specified in the "Segment Types" area).
- **Iterations**
The number of iterations of the segment (is specified in the "Segment Types" area).
- **Comment**
A comment (is specified in the "Segment Types" area).

New is used to create a new segment; **Delete** to delete one previously selected.

Use the arrow keys to move one or more selected segments up or down.

In the "Total Duration" field, the total duration (sum of all segments) of the signal is shown. This can be modified with **Change**.

There are four different options:



- **Adjust all signals proportionally**
All segments are changed proportionally.
- **Cut/Stretch last segment**
Only the last segment of the signal is changed accordingly (i.e. cut or stretched).
- **Cut/Stretch selected segments**
Only selected segments are stretched or cut.
- **Add new segment at the end**
If the time duration selected is longer than the current duration, a new segment is added at the end.

The segments are edited in the "Segment Details" area.

The "Segment Details" area: T

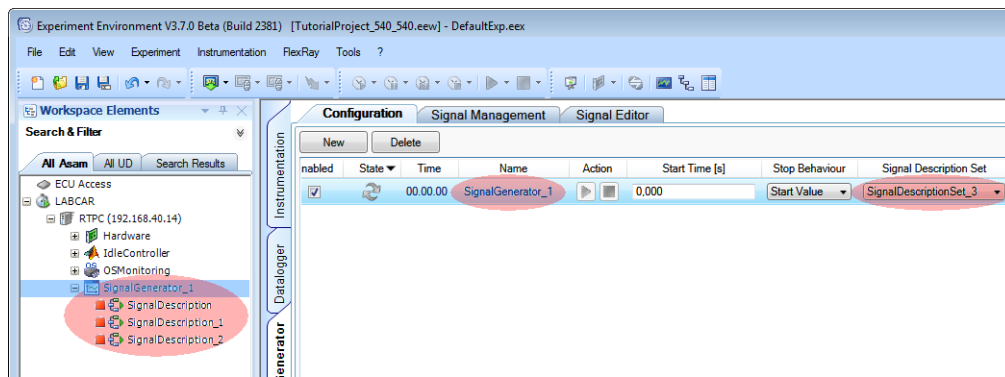
This is where the properties of a segment (selected in the "Segments" area) can be edited.

- **Duration**
This is where the length of the segment is specified.
- **Iteration**
This is where the number of iterations of the segment is defined.
- **Comment**
This is where a comment for the signal segment can be made.
- **Type**
The segment type:
 - **Constant**
Signal has a constant value ("Value").
 - **Data**
The segment is read in from a file (Measurement Data File (*.dat) or MAT-File (*.mat)).
 - **Pulse**
The segment consists of one PWM signal, defined by period duration ("Period"), amplitude ("Amplitude"), duty cycle ("Duty Cycle") and offset ("Offset").
 - **Ramp**
Rising or falling ramp, defined by start value ("Start Value"), duration ("Duration") and stop value ("Stop Value") or start value, duration and slope ("Slope").
 - **Random**
The segment consists of random values with defined amplitude ("Amplitude") and offset ("Offset").
 - **Saw**
Sawtooth, defined by amplitude ("Amplitude"), period duration ("Period") and offset ("Offset").
 - **Sine**
Sine signal, defined by amplitude ("Amplitude"), period duration ("Period"), phase shift ("Phase") and offset ("Offset").

Working with the Signal Generator

Signal description sets are assigned to the signal generators during configuration. These, in turn, contain signal descriptions.

These signal descriptions are shown as module outputs in the Workspace Elements (under the relevant signal generator) and can be connected with other inputs in a signal list.

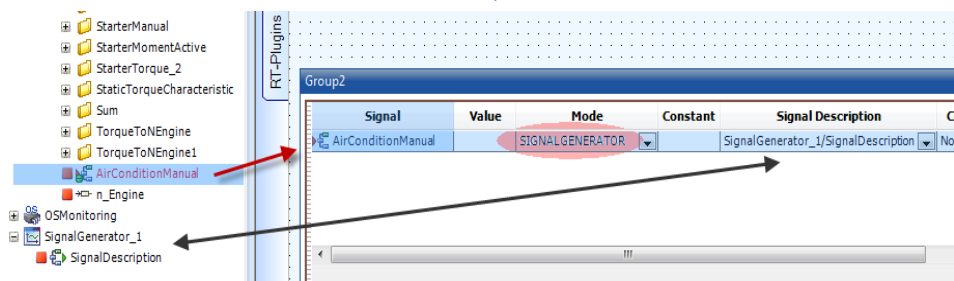


Note

For the signal generator and its signal descriptions to be shown in the element list, it has to be completely configured, i.e. on the one hand a signal description set must be assigned to the signal generator, and the signal description set must in turn have been assigned signal descriptions and, on the other hand, a task must have been selected.

To connect a signal generator

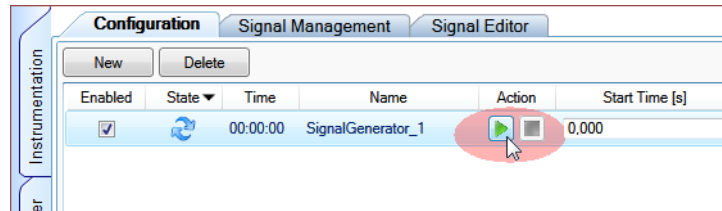
- Create a signal list in a layer (in the "Instrumentation" tab).
- Select a module input in Workspace Elements that you want to stimulate with a signal generator and drag it to the signal list.
- In the "Mode" column, select the option "Signal Generator".
- In the "Signal Description" column, select the signal generator and the signal description with which the module input is to be stimulated.



- Save the experiment.

To operate a signal generator

- Download the experiment for the Real-Time PC.
- Start the experiment.
- Toggle to the "Signal Generator" tab.
- Select the required signal generator and click the **Start** icon.



The running signal generator is shown by the **Running** icon.

The "Time" column shows the time which has elapsed since the last start of the signal generator.



- **Pause** is used to pause the signal generator – the last value of the signal generator continues to be output.



- **Stop and reset signal generator** stops the signal generator and resets it to the value selected in the column "Stop Behaviour".

4.4.4 The "RT-Plugins" Tab

In this tab RT-Plugins are managed. A description of this tab can be found in section "The "RT Plugins" Tab" on page 222.

4.4.5 The "Instruments" Window

The "Instruments" window contains a large number of instruments for interaction with the experiment.

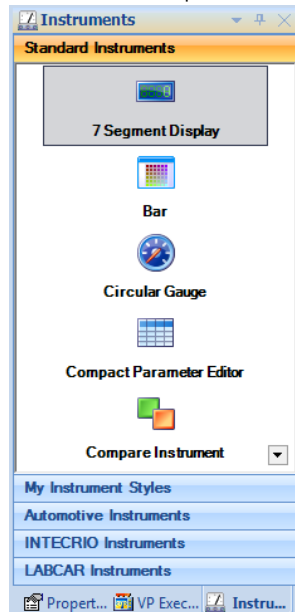


Fig. 4-8 The "Instruments" Window

The available instruments are divided into several groups:

- Standard Instruments
- Automotive Instruments
- LABCAR Instruments

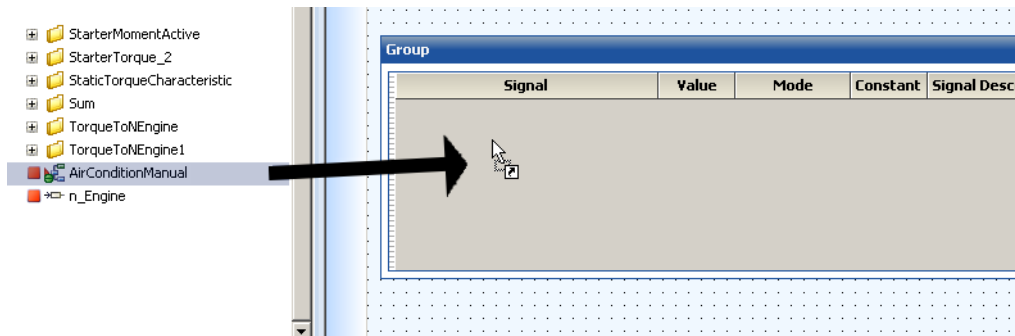
All of these instruments can be dragged into a layer of the "Instrumentation" tab using drag-and-drop – the assignment of measure variables and parameters also usually takes place via the **Measure/Calibrate** function of the shortcut menu or by drag-and-drop from the "Workspace Elements" window.

4.4.6 Adding Signals to the Signal List

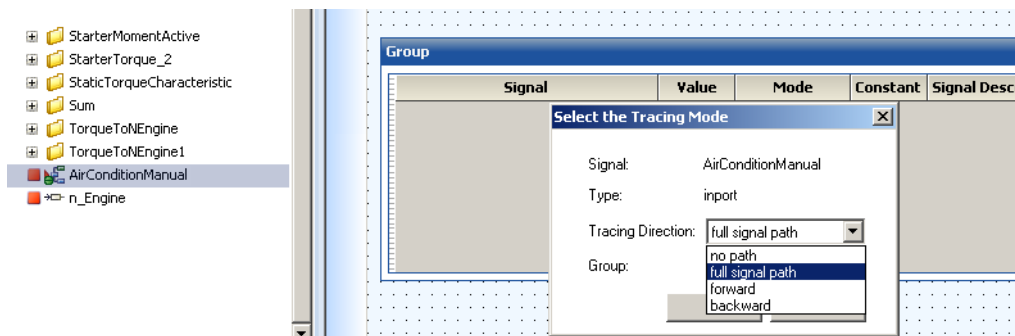
After creating a "Signal List" instrument, there are two different ways of adding signals to it.

To add a signal with signal tracing

- Drag the required signal to the signal list.



- In the "Select the Tracing Mode" dialog box, select "full signal path", "forward" or "backward".



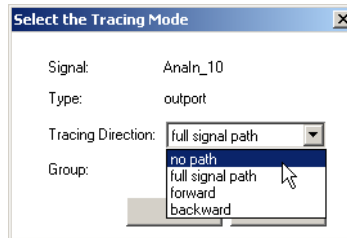
The selected signal is displayed in the signal list with all the signals to which it is connected (in the selected direction).

Signal	Value	Mode	Constant	Signal Description	Comment
AnaIn_0	0.000000				
AirConditionManual	0.000000	MODEL	1		No comment

In this display, an input can also be disconnected from the model and assigned a constant value ("Mode" column).

To add a signal without signal tracing

- If you want to display a signal in the signal list without its other connections, select the option "no path" in the "Select the Tracing Mode" dialog box.



The signal is shown in the selected signal list in a type of sublist.

Signal	Value	Mode	Constant	Signal Description	Comment
AnaIn_0	0.000000				
AirConditionManual	0.000000	MODEL	1		No comment
AnaIn_10	0.000000				

To add further signals to an existing signal list

- Drag another signal to the part of the list that contains the signals with tracing.

The "Select the Tracing Mode" dialog box opens. This is where you can specify the type of signal tracing.

- Drag another signal to the part of the list that contains the signals without tracing.

The signal is added to this part of the list.

Signal	Value	Mode	Constant	Signal Description	Comment
AnaIn_0	0.000000				
AirConditionManual	0.000000	MODEL	1		No comment
AnaIn_1	0.000000				
AnaIn_10	0.000000				
AnaIn_11	0.000000				
AnaIn_12	0.000000				
AnaIn_13	0.000000				

The following also applies to the signals in the list:

- If several signals (selected in Workspace Elements in a multiple selection) are moved to the list, they are always added to the part without signal tracing.
- Signals from one part of the list can be copied to the other part (or to a new list) by selecting the properties accordingly in the "Select the Tracing Mode" dialog box.

- Signals from the part of the list without signal tracing can be moved to other signal lists by drag & drop - the "Tracing Mode" can be specified in the target list.

4.5 The Script Recorder

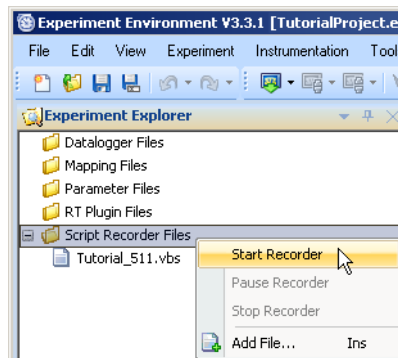
When running experiments, various actions such as operating GUIs (e.g. opening GUIs and modifying parameters, loading parameter files etc.) are often executed repeatedly. LABCAR-OPERATOR V5.4.4 thus provides a recorder so that these repeated procedures can be automated.

These macros are stored as Visual Basic Scripts (*.vbs). Any number of scripts can be assigned to any one project - a script can also be used in other projects.

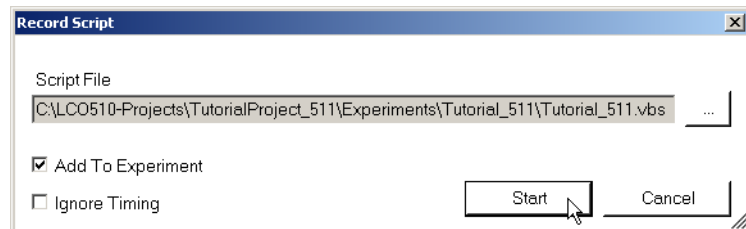
The LABCAR-OPERATOR V5.4.4 Script Recorder can be used to record all user actions supported by the ETAS EE Scripting API.

To record a script

- Right-click the "Script Recorder Files" folder in the Experiment Explorer.
- Select **Start Recorder** from the shortcut menu.



- Select a file in the following dialog box - regardless of whether a script file has already been assigned to the project or not.



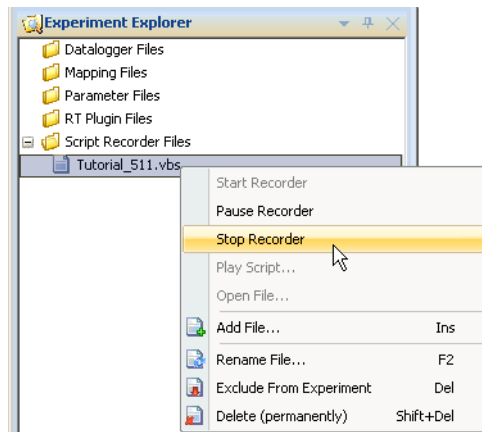
- Select "Add to Experiment" if the file is to be added to the experiment (in the directory "Script Recorder Files").
- The "Ignore Timing" option ensures that the times between the individual actions of the user are not recorded.

- Click **Start**.

Recording 00:00:29

Recording starts. The status of the Script Recorder and the duration of the recording so far are shown at the bottom edge of the main window of the experiment environment.

- To stop the recording, select **Stop Recorder** from the shortcut menu.

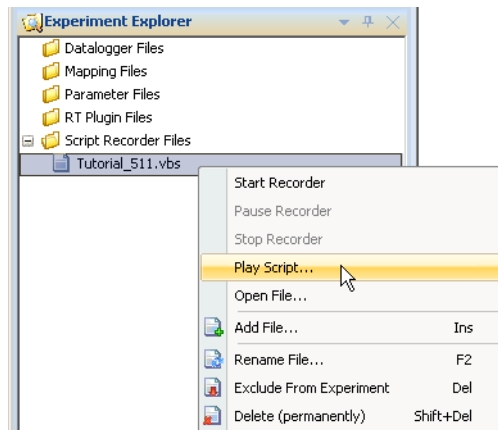


Script recorder idle

Recording stops and the script is stored.

To play a script

- Select the script to be played from the Experiment Explorer.
- Select "Play Script" from the shortcut menu.



Playback 00:00:30

The script is played.

To manage scripts

- Select **Add File** to add a script to the project.
- Select **Rename File** to rename a script.
- Select **Exclude From Experiment** to cancel the assignment of the file to the experiment.
- Select **Delete (permanently)** to delete the file.

4.6 Working with Parameters

This section contains information on how to work with the parameters of your LABCAR-OPERATOR project in the experiment environment ETAS EE.

This includes information on:

- "Parameters in ETAS EE" on page 307
- "The Shortcut Menu" on page 308
- "Modifying Individual Parameters During the Experiment" on page 309
- "Saving Complete or Partial Parameterizations" on page 313
- "Managing Parameter Files in the Experiment Explorer" on page 314
- "Variants and Parameter Combinations" on page 317
- "Mapping Files" on page 321
- "Creating and Managing Mapping Files" on page 322

4.6.1 Parameters in ETAS EE

Every module that is added to the overall project in LABCAR-IP includes a new set of parameters. These parameters and their assigned values (referred to as parameterization) can be activated, modified, saved or exported.

In ETAS EE, a project's parameters and measure values are collected in the "Workspace Elements" window – for a detailed description of this window, refer to the section "The "Workspace Elements" Window" on page 270.

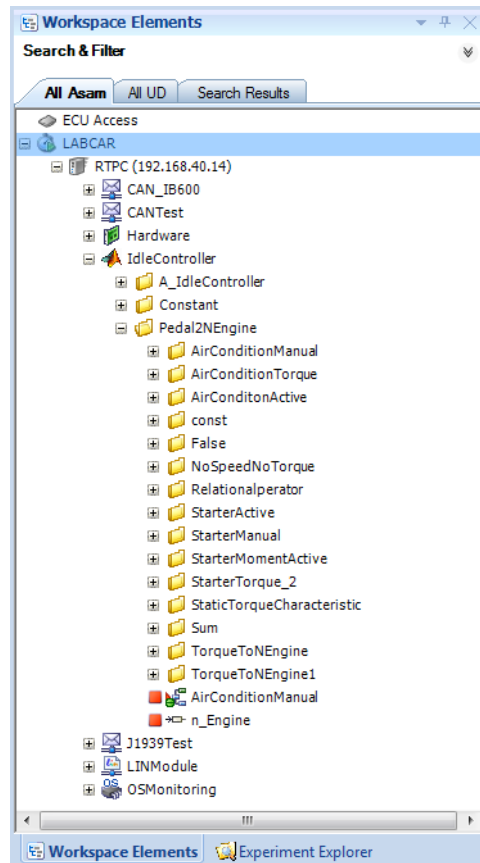


Fig. 4-9 The "Workspace Elements" Window

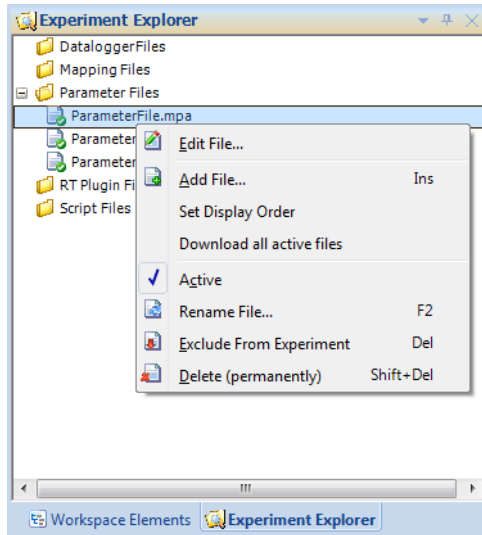
The parameters named are in the "All Asam" tab (the two other tabs are described in section "The Tabs of the "Workspace Elements" Window" on page 271).

In this tab, all modules and their parameters and measure values are grouped in the "LABCAR" folder – the "ECU Access" folder contains the definitions of the ECU pins as well as any parameters and measure variables of a parallel INCA experiment (with Calibration Connector for INCA).

The individual modules have special icons that denote their type. These are used to run operations on the module folder or its contents using the corresponding shortcut menus (see "Working with the Workspace Elements" on page 272).

Managing Parameter Files

Files that contain parameters and assigned values are managed in the "Experiment Explorer" window – the shortcut menu contains the corresponding functions for this purpose.



You will find the following file types:

- *.mpa and *.dcm
Files that contain parameters and value assignments
- *.olc4
File with a specific configuration of signal paths and curves
- *.pac
Parameter variant (or parameter combination): Compilation of several *.mpa, *.dcm and *.olc files

For more details, refer to the section "Managing Parameter Files in the Experiment Explorer" on page 314.

4.6.2 *The Shortcut Menu*

The shortcut menu in the "All Asam" tab has already been described in "Working with the Workspace Elements" on page 272 – the following section focuses on working with parameters.

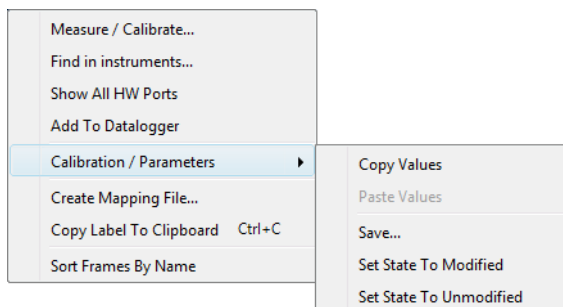


Fig. 4-10 Shortcut Menu for Parameters (with Active Experiment)

The shortcut menu provides a range of functions:

- Creating an instrumentation for modifying individual parameters (**Measure / Calibrate**) (see "Modifying Individual Parameters During the Experiment" on page 309)
- Saving a parameter set (**Calibration / Parameters → Save**) (see: "Saving Complete or Partial Parameterizations" on page 313).
- Editing entire parameter sets (*.mpa) (see "To edit a parameter file" on page 314)

In addition to the modification of individual parameters or entire parameter sets, this menu also enables other actions:

- Copying and pasting individual parameter values (**Calibration / Parameters → Copy/Paste**)
- Editing the modification status of a parameter/measure value (**Calibration / Parameters → Set State To Modified/Unmodified**)
- Creating mapping files (**Create Mapping File...**) (see "Mapping Files" on page 321)
- Copying labels to the clipboard (**Copy Label To Clipboard**)
- Complete display of all hardware parameters (**Show all HW Ports**)
- Search for GUIs in which a specific parameter/measure value is used (**Find in instruments...**)

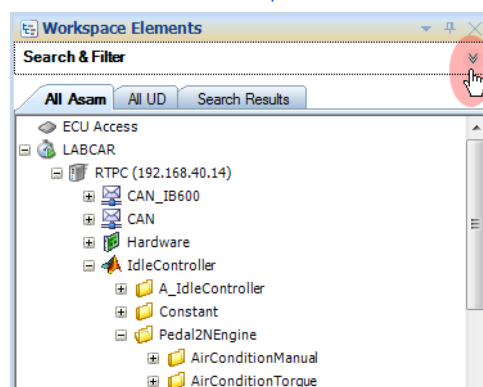
4.6.3 Modifying Individual Parameters During the Experiment

The information on the values of parameters and the modification thereof takes place in a ETAS EE display instrument ("Instrumentation" window). Normally, an "edit box" (see below) is used for this purpose.

A search function is available as an experiment can have a large number of parameters/measure variables.

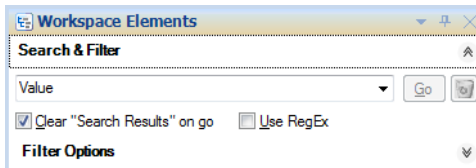
To search for parameters

- Open up the edit line for the search key by clicking the icon in the "Workspace Elements" window.

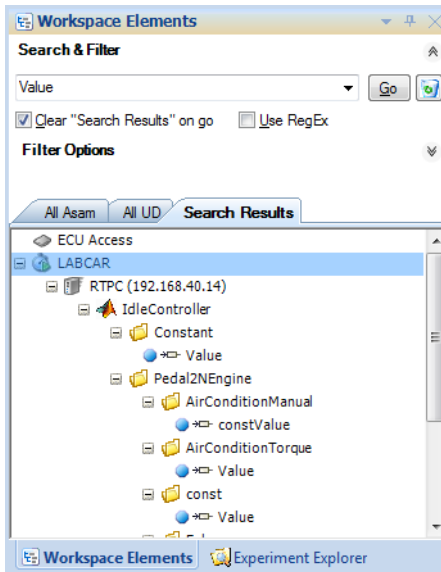


The edit line appears.

- Enter a search string and click **Go**.



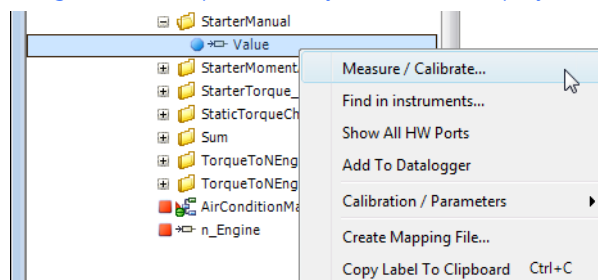
The "Search Results" tab that shows the results of the search is activated.



By selecting suitable filter criteria ("Filter Options"), you can limit the results list further if required.

To create a GUI

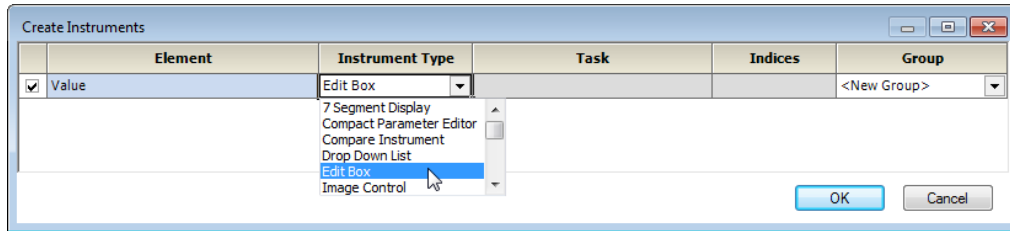
- Right-click the parameter you want to display/edit.



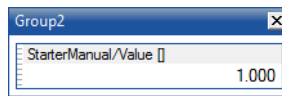
Note

To select several parameters use <SHIFT> or <CTRL>. You can also select entire folders.

- Select **Measure/Calibrate**.
- The "Create Instruments" window opens.

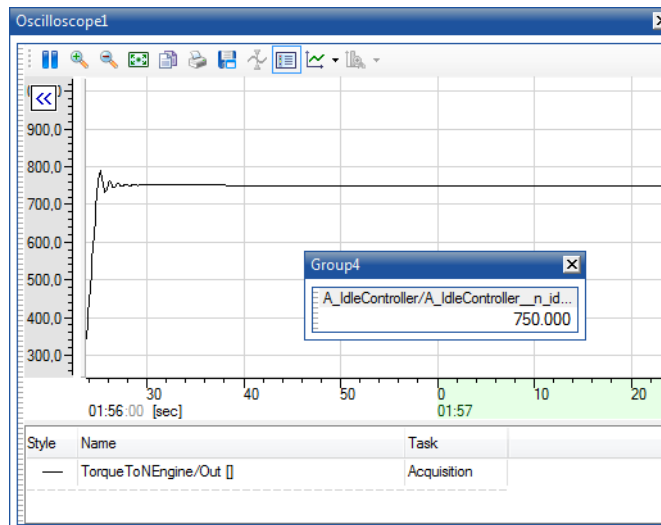


- Select "Edit Box" under "Instrument Type".
- The GUI is created and the current value of the parameter shown.



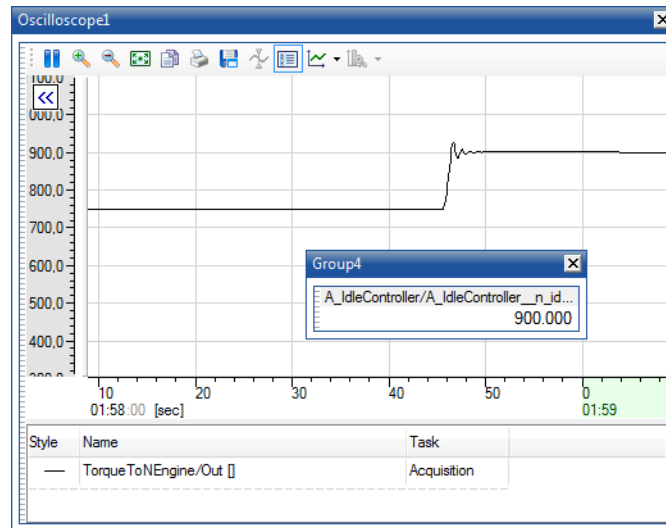
To modify parameters in the current experiment

- Start the experiment.
- Click the GUI with the parameter to be modified.
- The parameter can now be edited.



- Enter the new value.
- Press <RETURN>
- or
- Click outside the GUI.

- The parameter value is modified during the current simulation.



Displaying and Resetting the Modification Status

If the value of a parameter in a current experiment has been changed in comparison to the value when downloading the code with the initial parameterization, the value is shown in red.

This coloring can be set using the shortcut menu with **Calibration / Parameters** → **Set State To Modified** and reset with **Calibration / Parameters** → **Set State To Unmodified**.

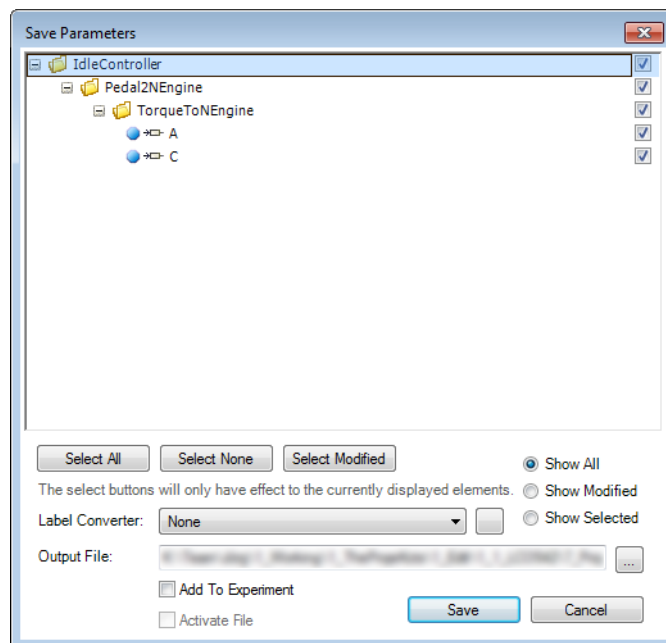
4.6.4 Saving Complete or Partial Parameterizations

To save parameterizations in the current experiment, the user can select one or more parameters in a folder, one or more folders or the entire list in the "Workspace Elements" window and then save as a file.

To save a parameter set

- Select the parameters/folders/modules whose values/parameterization you want to save in a file.
- Right-click and select **Calibration / Parameters** → **Save**.

The "Save Parameters" window opens.



All parameters of the hierarchy/folders selected previously are shown and selected. You can change the "global" selection by using **Select All**, **Select Modified** and **Select None** – the list of selectable elements can also be filtered by using **Show All**, **Show Modified**, **Show Selected**.

The parameter file is selected using "Output File" – clicking ... opens a file selection window in which the name and the format (*.mpa, *.dcm or *.cdfx) of the file can be selected.

Two further options can also be activated in this file selection window:

- Add To Experiment
The file is automatically added to the experiment, i.e. it is available and accessible in the Experiment Explorer in the "Parameter Files" folder.
- Activate File
The file is activated (see "To set a parameter file as active" on page 315)

4.6.5 Managing Parameter Files in the Experiment Explorer

The "Parameter Files" folder of the Experiment Explorer lists all parameter files that belong to the experiment.

You can run the following actions using the shortcut menu:

- "To edit a parameter file" on page 314
- "To add a parameter file" on page 314
- "To specify the order of parameter files" on page 315
- "To load parameters into the experiment" on page 315
- "To set a parameter file as active" on page 315
- "To rename a parameter file" on page 316
- "To exclude a parameter file from the experiment" on page 316
- "To delete a parameter file" on page 316

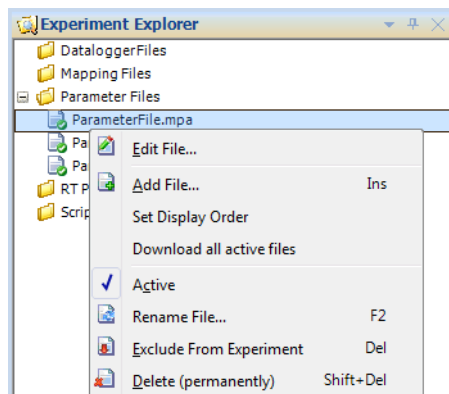


Fig. 4-11 The "Parameter Files" Folder Shortcut Menu

To edit a parameter file

Parameters that have previously been saved in a file (see "Saving Complete or Partial Parameterizations" on page 313) can be opened here for editing purposes.

- Select **Edit File** or double-click the file.
In the case of an *.mpa file, the LABCAR-PA Parameterization Assistant opens and loads the parameter file.

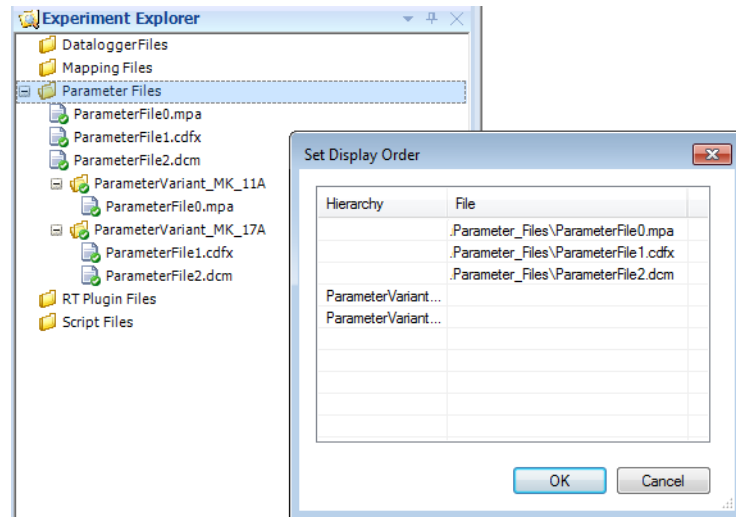
To add a parameter file

- To add further parameter files to the experiment, select **Add File**.
- A file selection window opens.
- Select the file(s) to be added (multiple selection using <SHIFT> or <CTRL>) and click **Open**.
The parameter files are added to the "Parameter Files" folder.

To specify the order of parameter files

When simulation code is loaded, parameter files are loaded onto the experimental target in the same order as they are displayed in the "Parameter" folder.

- Select **Set Display Order** to change this order.
The "Set Display Order" window opens.



In this window, the folders with variants (also referred to as parameter combinations (*.pac) – see "Variants and Parameter Combinations" on page 317) are listed in the "Hierarchy" column and the top-level parameter files in the "File" column.

- Just use Drag & Drop to move a folder or a file up or down the list.
- Click **OK**.

The order of folders/files in the Experiment Explorer is changed accordingly.

To load parameters into the experiment

- To download all activated parameter files to an experiment, select **Download all active files**.

To set a parameter file as active

- To activate a parameter file, select the option **Active** in the shortcut menu.
The parameterization in this file is then automatically loaded to the target.
- You can cancel this by clicking again.

To rename a parameter file

- To rename a parameter file, select **Rename File**.
A dialog box opens in which you can enter a new file name.

To exclude a parameter file from the experiment

- To exclude a parameter file from the experiment, select **Exclude From Experiment**.
The assignment is canceled and the file is excluded from the Experiment Explorer view.

Note

The file is not deleted - its assignment to the project is simply canceled.

To delete a parameter file

- To remove a parameter file both from the experiment and from your file system, select **Delete (permanently)**.

The shortcut menu of the "Parameter Files" folder also allows you to create parameter variants which are described below.

4.6.6 Variants and Parameter Combinations

In an ECU test, you often have several variants of an ECU - in addition, there are various software versions available for each of these ECUs.

For a test project, this means that a test project variant must be available for every combination of ECU variant and software variant (see Fig. 4-12).

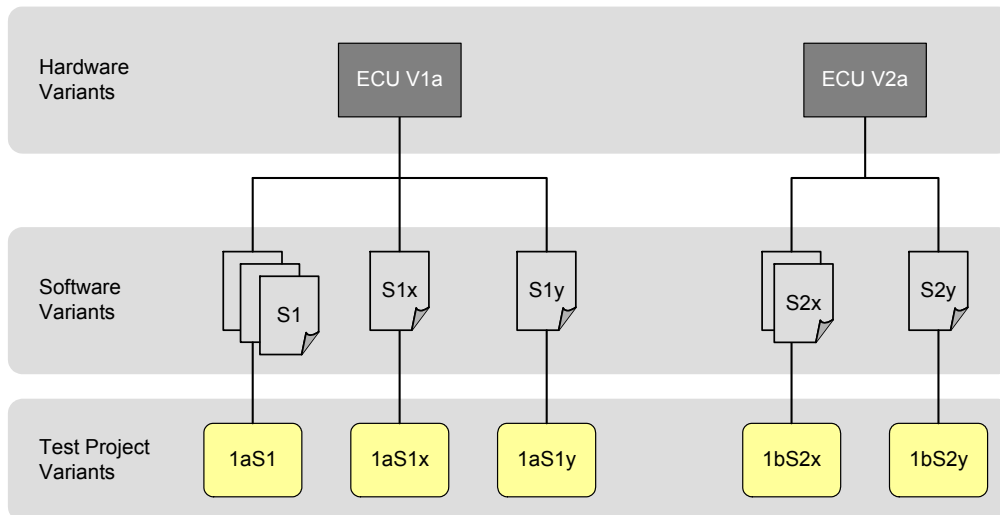


Fig. 4-12 Variants of ECU Hardware, ECU Software and the Corresponding Test Projects

This means on the one hand considerable redundancy and on the other a considerable amount of time when it comes to global changes, in other words changes that affect all variants.

In the test project, differences between the variants can have an effect on the following project data:

- Model parameters
- Open-loop configuration (sensor curves etc.)
- Configuration of the I/O hardware (measure modes, resolution, etc.)
- Connection data of ECU to I/O hardware

The problem can be solved by keeping global data and specific data separate (see Fig. 4-13):

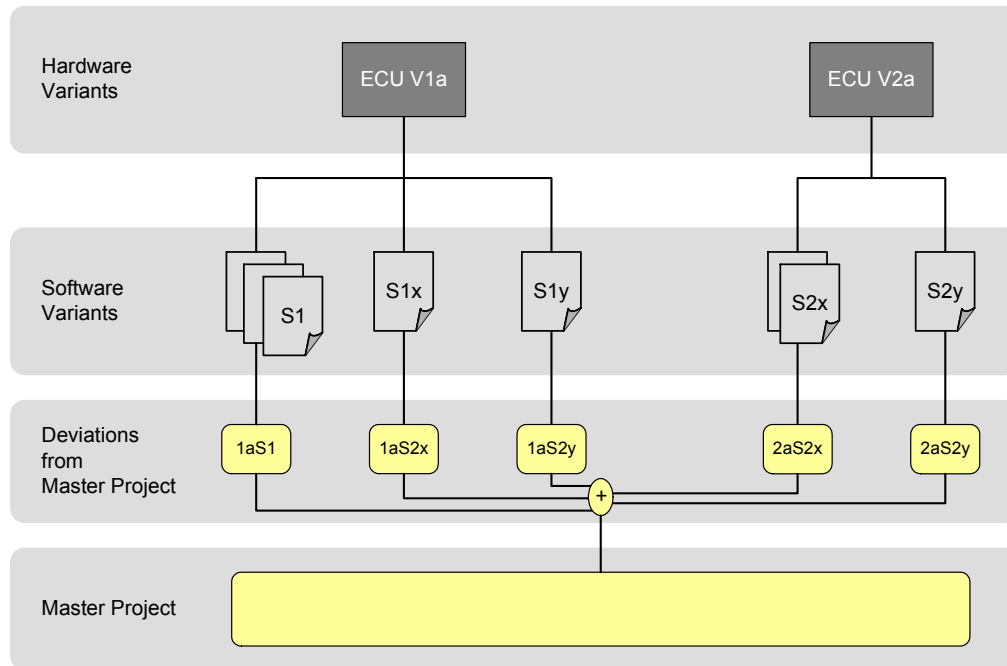


Fig. 4-13 Master Project and Files for Deviations (Variants) from the Master Project

Master Project

The master project is a complete LABCAR-OPERATOR project that contains a default parameterization. It is single-source and free of redundancy.

Deviations

The deviation file contains the special project parameters that make the project usable for a specific UuT variant with a specific software variant. These files are called variants.

What Does a Variant Contain?

Variants can contain the following kinds of data:

- Model parameters
e.g. for different engine components
(saved in an *.mpa, *.dcm or *.cdfx file)
- OLC parameters
e.g. sensor curves of various lambda sensors
(saved in an *.olc4 file)
- Hardware configuration parameters
e.g. different settings of the I/O hardware
(saved in an *.mpa or *.dcm file)
- Connection data: ECU - I/O hardware
(saved in the "properties.ecu" file)

The first three types can be combined and loaded to an experiment by activating them.

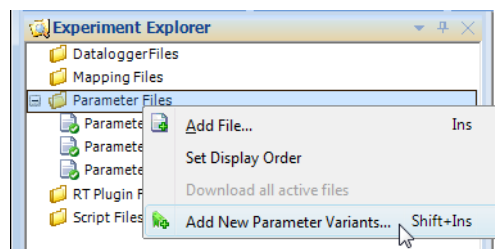
Parameter Combinations

The loading of model parameters to an online experiment usually refers to an ordered sequence of parameter files loaded to the target in the order they appear in the "Parameter Files" folder.

These kinds of sequences can be exported as parameter combinations (*.pac). Any number of these combinations can be assigned to a project.

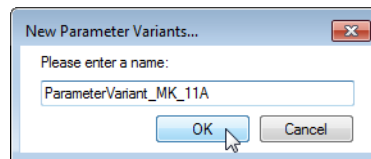
To create a new variant

- Right-click the "Parameter Files" folder and select **Add new Parameter Variants**.



The "New Parameter Variants" window opens.

- Enter a name for this variant and click **OK**.



A file selection window opens from which you can select the files you want to belong to the new variant.

- Select the files and click **Open**.

The variant is created and added to the experiment.

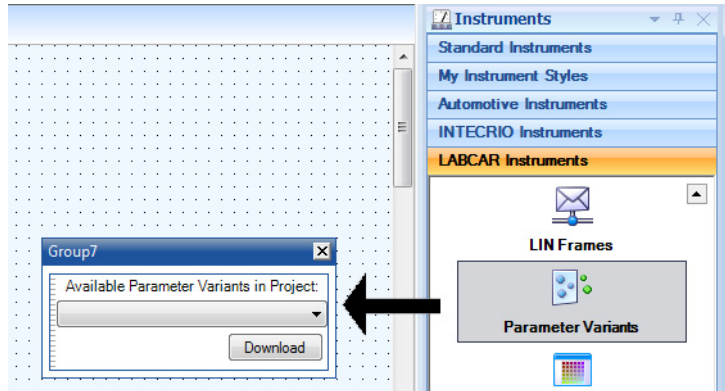
The functionality described in "Managing Parameter Files in the Experiment Explorer" on page 314 is available for managing parameter variants:

- Adding further files to a variant (**Add File**)
- Modifying the order (**Set Display Order**)
- Downloading all active files (**Download all Active Files**)
- Activating and deactivating (**Active**)
- Renaming a file (**Rename File**)
- Excluding a file from a project (**Exclude From Experiment**)
- Deleting file (**Delete (permanently)**)

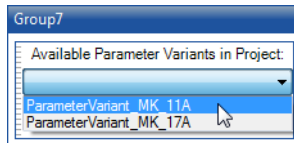
To load select variants to the experiment

In addition to the possibility described above of downloading all parameter variants of the experiment from the shortcut menu (**Download all active files**), there is also a GUI with which you can select a single variant and load it into the experiment at any time.

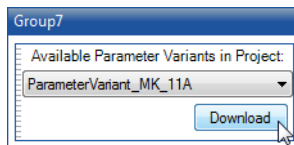
- In the "Instruments" window ("LABCAR Instruments" tab), select the "Parameter Variants" instrument and drag it into a layer.



- From the list, select one of the variants that is part of the project.



- Click **Download**.



The selected variant is loaded into the experiment.

To export a variant as a *.pac

Variants can also be exported as files (*.pac) and then added to an experiment.

- To export a parameter variant, select the folder.
- Select **Export to PAC File** from the shortcut menu. A file selection window opens from which you can select the file name of the *.pac file.
- Enter the file name and select **Save**.

To reimport a variant

- To add a parameter variant to an experiment, select **Add File**.
A file selection window opens.
- Select the required file (with the extension ".pac") and click **Open**.
The variant is added to the experiment.

4.6.7 Mapping Files

ASAM labels can be mapped to user-defined names in a mapping file making them easier to work with.

A mapping file is supplied with, for example, ETAS LABCAR models to support the GUIs of the models, but can also be created by the user.

Generally, a project can be assigned any number of mapping files which can be managed in the ETAS EE Experiment Explorer. Mapping files are edited in a special editor, the SUT Mapping File Editor.

Display of Mappings in the "All UD" Tab

For example, the user can map the label

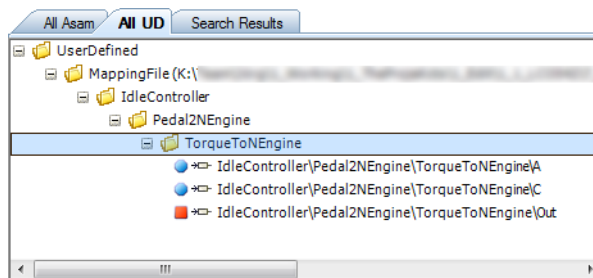
IdleController/Pedal2NEngine/TorqueToNEngine/Out

(the tool label) of a measure variable in a module to the label

EngineSpeed

(the test label).

These user-defined labels are shown in the "Workspace Elements" window in the "All UD" tab – with their hierarchy and the file in which they were defined.



4.6.8 Creating and Managing Mapping Files

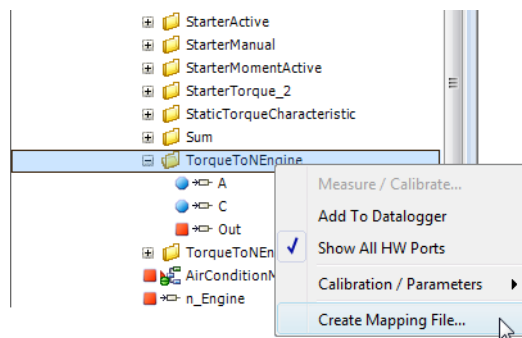
This section describes how to create mapping files and manage them using the shortcut menu in the Experiment Explorer.

To create a mapping file

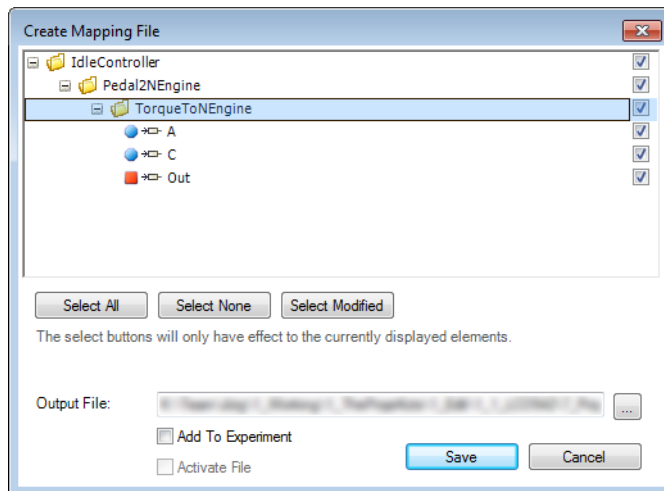
- Select the parameter or the measure variable for which you want to create a mapping file in the "Workspace Elements" tab.

You can also select parts of the hierarchy (= folder) or the main "LABCAR" node.

- Select **Create Mapping File** from the shortcut menu.



The "Create Mapping File" window opens.



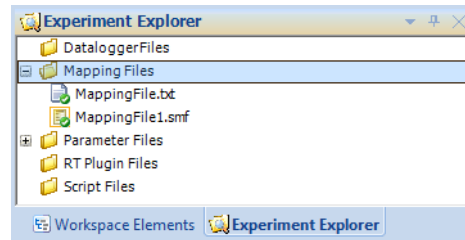
- Select the parameters/measure variables to be present in the mapping file.
- Click ... to specify the file in which the mapping should be saved.

A file selection window opens in which you can enter a name for the file.

- Select "Add to Experiment" if the file is to be added to the experiment (in the directory "Mapping Files" in the "Experiment Explorer" tab).

- Click **Save** to save the file name.
- Click **Save** in the "Create Mapping File" window to save the mapping file.
- Change to the "Experiment Explorer" tab and double-click the "Mapping Files" folder.

The created mapping file has been added to your experiment and activated (shown by the green tick beside the folder icon).



The "Mapping Files" folder of the Experiment Explorer contains all mapping files that belong to the experiment.

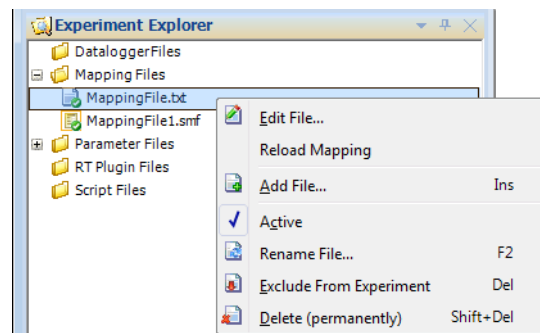


Fig. 4-14 The "Mapping Files" Folder Shortcut Menu

You can use this folder's shortcut menu for editing and managing purposes. The following actions are described in detail below.

- "To edit a mapping file" on page 323
- "To reload a mapping" on page 324
- "To add a mapping file to the project" on page 324
- "To activate a mapping file" on page 324
- "To rename a mapping file" on page 324
- "To exclude a mapping file from the project" on page 325
- "To delete the mapping file permanently" on page 325

To edit a mapping file

- Double-click the file to be edited in the Experiment Explorer.

or

- Right-click the file and select **Edit File**.
The SUT Mapping File Editor opens.

To add a mapping file to the project

- Right-click "Mapping Files" in the Experiment Explorer.
- Select **Add File**.
A file selection window opens.
- Select the required mapping file (*.txt, *.smf) and click **OK**.
The mapping file is added to the project.

To reload a mapping

- Right-click the file to be loaded in the Experiment Explorer.
- Select **Reload Mapping**.
The selected mapping file is reloaded in the experiment.

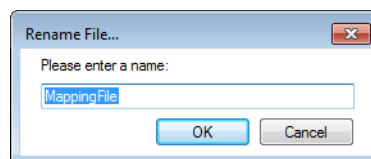
To activate a mapping file

- Right-click the file to be activated in the Experiment Explorer.
- Select **Active**.
The selected mapping file is activated (indicated by the tick).



To rename a mapping file

- Right-click the file to be renamed in the Experiment Explorer.
- Select **Rename File**.
The "Rename File" dialog box opens.



- Enter the new name and click **OK**.

To exclude a mapping file from the project

- Right-click the file to be excluded in the Experiment Explorer.
- Select **Exclude From Experiment**.
The mapping file is excluded from the project.

Note

The file is not deleted - its assignment to the project is simply canceled.

To delete the mapping file permanently

- Right-click the file to be deleted in the Experiment Explorer.
- Select **Delete (permanently)**.
The mapping file is deleted.

4.7 Editing Parameter Files with LABCAR-PA

LABCAR-PA (Parameterization Assistant) is used for the clear display and for editing parameter files.

LABCAR-PA provides the following advantages:

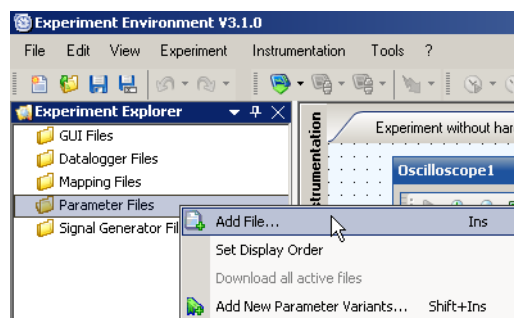
- Structured overview of all model parameters and their attributes
 - Display acc. to model hierarchy or functional units
 - Filter and sort possibilities
- Support of the parameterization process with simple access to parameters:
 - Online and offline operation
 - Visualization of the parameterization status with attributes
 - Visualization of changes
 - Comparison possible with reference parameterization
- Single-source management of model parameterizations in XML files
- Import and export of parameters from and to DCM and M files
- Simple update of the model

4.7.1 Managing Parameter Files

The ETAS EE Experiment Explorer contains the "Parameter Files" folder in which you can manage the experiment's parameter files.

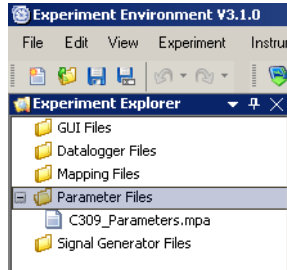
To add a parameter file to the experiment

- In the Experiment Explorer select the "Parameter Files" folder and right-click.
- In the shortcut menu select **Add File**.



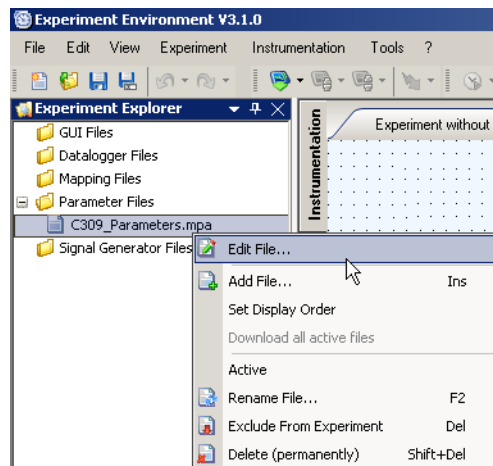
- In the file selector window select the file to be added.

The file is added.

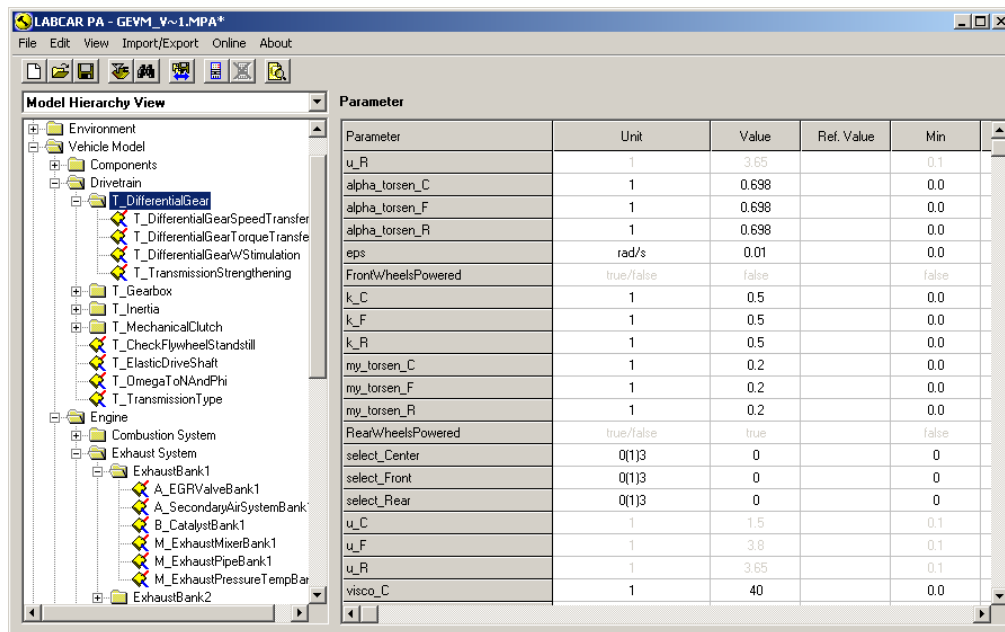


To edit a parameter file

- To edit a parameter file, double-click it
or
- In the shortcut menu select **Edit File**.



LABCAR-PA opens and the selected file is loaded.



The sections below contain a description of the LABCAR-PA GUI and instructions on how to work with LABCAR-PA.

4.7.2 The Parameter Table

This section describes the functions of the parameter table. You will find information on:

- "Scope of the Parameter Table" on page 328
- "The Entries in the Parameter Table" on page 329
- "Which Table Entries can be Edited?" on page 330
- "The Shortcut Menu of the Parameter Table" on page 330
- "Display Options for the Parameter Table" on page 331

Section "Editing Parameters" on page 342 describes how parameters can be edited.

Scope of the Parameter Table

The number of attributes shown in the parameter table (= columns) can be changed (see "The Entries in the Parameter Table" on page 329). This preference must be defined in the **View** menu (see section "View" Menu" on page 334).

The following settings are possible:

- **Basic Attributes**

The columns "Unit", "Value" and "Comment" are shown in addition to the parameter name.

- **All Attributes**

All items listed in section are displayed.

- **Custom Attributes**

All entries selected using [View](#) → [Customize Attribute View](#) are displayed here (see "View Menu" on page 334).

The Entries in the Parameter Table

The meaning of the individual columns in the parameter table is described below.

- **Parameter**
This is the name of the parameter as used in the model.
- **Unit**
The physical unit of the parameter, e.g. km/h or Pa/s.
- **Value**
The value of the scalar parameter or the first value of a non-scalar parameter.
- **Ref. Value**
The value of the parameter in the reference file.
- **Min**
The recommended minimum value of the parameter.
- **Max**
The recommended maximum value of the parameter.
- **Dimension**
The dimension of the parameter
 - scalar (= fixed value)
 - array (= fixed value block)
 - 1D table (= curve)
 - 2D table (= map)
- **Scope**
The validity of the parameter:
 - local
 - exported
 - imported

If a parameter is imported, this column shows the name of the block the parameter was exported from.
- **Proc. State**
Describes the status of the parameter in the parameterization process.
 - coarse
 - fine
 - switch
- **Type**
The variable type:
 - integer (sdisc)
 - floating point number (cont)

- Boolean (log)
- ModelOption
 - Functional context of the parameter (general, WVLT, GDI, etc.)
- UserGroup

"initial" means that this parameter belongs to the group of parameters whose value has to be set before a LABCAR project is run for the first time. These parameters are displayed via **View** → **Initial Project Parameters**. All other values have no significance in this version.
- Mod. Status

Modification status of the parameter:

 - unchanged
 - changed
 - accepted

The colors indicating the modification status of a parameter are described in section "Display Options for the Parameter Table" on page 331.
- Comment

Contains a short comment on the parameter.

Which Table Entries can be Edited?

The values of the following columns can be edited by the user:

- Value
- Min
- Max
- Proc. State
- Mod. Status

The Shortcut Menu of the Parameter Table

Right-clicking one line of the table opens a shortcut menu with which it is possible to modify values, to set the modification status (changed, accepted) and to have a look at the modification history .

Edit	
Copy	Strg+C
Paste	Strg+V
Mark as "accepted"	
Mark as "changed"	
View History	

- Edit

Edits the parameter in the current column.

Note

The value of the parameter ("Value" column) is changed more easily by clicking it with the left-hand mouse button.

- Copy, Paste

Copying and pasting a value (and entire ranges)

- Mark as accepted
This marks the value of a parameter as being accepted.
- Mark as changed
This marks a parameter as being changed.
- View History
Opens the "Parameter History" dialog box in which you can view and edit the history of the parameter (for more details see "Viewing and Editing the Parameter History" on page 346).

Note

If an error message is displayed, the path to the documentation files has been set incorrectly (see "Paths for Working, Model and Model Documentation Directories" on page 339) or the file does not exist.

Display Options for the Parameter Table

Certain parameter properties are indicated by special markings when the parameter table is displayed:

- Parameter has been changed
- Modification of the parameter accepted
- Parameter is imported
- Parameter is imported and has been changed
- Parameter is imported and has been accepted
- Changed parameter with a loaded reference file

Parameter has been changed:

If a parameter has been changed, the following table entries are displayed in red:

- Parameter Name
- Parameter Value
- Modification Status

"Modification Status" is also altered from "unchanged" to "changed".

Aggressiveness	0.1	0.99	1	changed	0: sluggish driver, 1: aggressive d
----------------	-----	------	---	---------	-------------------------------------

Modification of the parameter accepted:

If modifying the parameter in a particular phase of parameterization proves to be sensible, the "Modification Status" can be set to "accepted" (see section "The Shortcut Menu of the Parameter Table" on page 330).

An entry previously shown in red in the table is now shown in green.

Aggressiveness	0.1	0.99	1	accepted	0: sluggish driver, 1: aggressive d
----------------	-----	------	---	----------	-------------------------------------

The same is also true of parameters with "Modification Status" "unchanged".

Parameter is imported:

A parameter cannot be changed in a block if it has been imported. This kind of imported parameter is shown in gray in the parameter table.

ClutchDirect	0.1	0.01		unchanged	Lowpass for clutch
--------------	-----	------	--	-----------	--------------------

An imported parameter can only be changed in the block from which it was exported. How to find out exactly which block this was, is explained in section "Determining the Parameter Scope Binding" on page 358.

Parameter is imported and has been changed:

If an imported parameter has been changed in the block from which it was exported, the parameter name is displayed in red; the other two entries (Value and Modification Status) are shown in orange.

BatteryIsOnManual		false	false	changed	signal for battery status
-------------------	--	-------	-------	---------	---------------------------

Parameter is imported and has been accepted:

If an imported, changed parameter has been accepted in the block from which it was exported, the parameter name is displayed in green; the other two entries (Value and Modification Status) are shown in light green.

BatteryIsOnManual		false	false	accepted	signal for battery status
-------------------	--	-------	-------	----------	---------------------------

Changed parameter with a loaded reference file:

If a parameter has been changed and a reference file is open, the increase or decrease in value in comparison to the value in the reference file is shown to the right of the parameter value by a red arrow pointing either up or down.

F_StandingLimit	N	0.0	↓	3000	changed	maximum adhesion Force
k_FRolling	1	2.0	↑	0.015	changed	Rolling resistance coefficient

If the changed parameter is a table, values may have been changed up and down. This is shown by a double arrow.

Note

The display of the arrow depends on whether the new value deviates from the original one by a specific percentage entered by the user (see "Settings for Reference Files" on page 338).

Note

Changes are not displayed for parameters of the type "Enumeration".

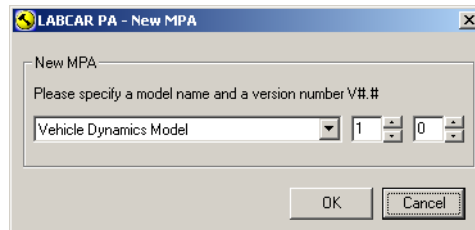
4.7.3 A Description of the Menus

This section contains a general description of the function of each of the available menu items.

"File" Menu

- **File** → **New**

Creates a new (empty) project parameter file (*.mpa).



Selecting a model or a model component and the version helps with the assignment of parameter data in the created file to a model or model component.

The entire hierarchy is created for complete models – the model name then appears as the top element in the structure view.

- **File** → **Open**

Opens a file selector window with which a project parameter file (*.mpa) can be loaded.

A working directory is specified via the **Edit** → **Options** menu (see "Edit" Menu" on page 334).

- **File** → **Close**

Closes the opened project parameter file.

- **File** → **Save**

Saves the project parameter file currently open.

- **File** → **Save As**

Saves the project parameter file currently open under a name to be specified.

- **File** → **Open Reference**

This is used to load a reference file (see section "Reference Files" on page 347).

- **File** → **Set To Reference File**

This is where you can save the parameterization of the current file as a reference file. The default file name is `<file_name>Reference_0.mpa`.

- **File** → **Close Reference**

Closes the reference file currently open.

- **File** → **Exit**

Exits the program.

"Edit" Menu

- **Edit** → **Edit Parameter**
For editing the selected parameter (see "Editing Parameters" on page 342).
- **Edit** → **Filter Settings**
Opens a dialog box in which display filters and sort criteria can be entered (see "The Filter and Sorting Function" on page 353).
- **Edit** → **Search Model Parameters**
Opens a dialog box in which you can search for model parameters (see "The Search Function for Model Parameters" on page 355).
- **Edit** → **Options**
Opens a dialog box in which options can be specified (see "Setting Options" on page 337).

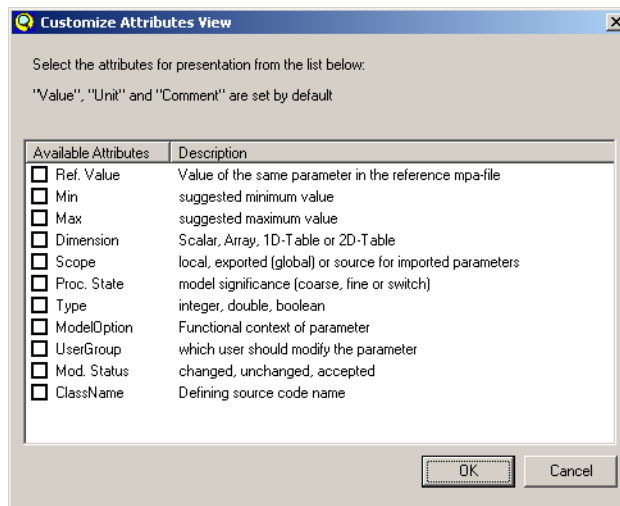
"View" Menu

- **View** → **Initial Project Parameters**
If this item is selected, only those parameters are displayed which are necessary for the first running of a LABCAR project.
- **View** → **Parameter Scope Binding**
Opens a list of parameters which are exported from blocks and imported into others. The names are also specified of the blocks from which or into which they are exported or imported respectively (see "Determining the Parameter Scope Binding" on page 358).
- **View** → **Basic Attributes**
If this item is selected, only the attributes "Unit", "Value" and "Comment" are shown in the parameter table (see "Scope of the Parameter Table" on page 328).
- **View** → **All Attributes**
This results in all attributes being displayed in the parameter table (see "Scope of the Parameter Table" on page 328).
- **View** → **Custom Attributes**
This results in all attributes being displayed in the parameter table which were selected in **View** → **Customize Attribute View** (see "Scope of the Parameter Table" on page 328).

- **View** → **Customize Attribute View**

This is where you can select the attributes which are displayed with **View** → **Custom Attributes** (see "Scope of the Parameter Table" on page 328).

The meaning of these attributes is described in the section "The Entries in the Parameter Table" on page 329.



"Import/Export" Menu

- **Import/Export** → **Import/Export Parameter Data**

Opens the dialog box for importing parameters from or exporting them to parameter files (see "Importing Parameters from a File" on page 350 and "Exporting Parameters to a File" on page 348).

"Online" Menu

- **Online** → **Go Online**

A check takes place during simulation as to whether the parameters of the opened mpa file are also available in the model (criterion: the ASAM labels have to be identical). For more details on this subject see "Online Parameterization" on page 359.

- **Online** → **Synchronize ASAM Labels**

Parameters in LabCar Developer models have different names from those in Simulink models. This function synchronizes parameters in mpa files with parameters imported from Simulink models and, if necessary, adapts their ASAM labels.

- **Online** → **Read Parameter**

- **Read Parameter from Target**

Reads the value of a parameter selected in the parameter table from the running model.

- **Read all Parameters from Target**

Reads all values of the parameters and the selected hierarchy from the running model.

- **Online** → **Write Parameter**
 - **Write Parameter to Target**
Writes the value of a parameter selected in the parameter table to the running model. However, this function is already executed when the value is changed in the parameter table.
 - **Write all Parameters to Target**
Writes all values of the parameters and the selected hierarchy to the running model.
- **Online** → **Go Offline**
Ends online parameterization.

4.7.4 The Icon Bar

Some of the menu functions described above can also be activated via the icon bar:



- **Create new MPA file**
Creates a new (empty) project parameter file (*.mpa). Exactly the same as **File** → **New**.



- **Open MPA file**
Opens a file selector window with which a project parameter file (*.mpa) can be loaded. Exactly the same as **File** → **Open**.



- **Save MPA file**
Saves the project parameter file currently open. Exactly the same as **File** → **Save**.



- **Filter settings**
Opens a dialog box in which display filters and sort criteria can be entered. Exactly the same as **Edit** → **Filter Settings**.



- **Search parameters**
Opens a dialog box in which you can search for model parameters. Exactly the same as **Edit** → **Search Model Parameters**.



- **Import/Export parameters**
Opens the dialog box for importing parameters from or exporting them to parameter files. Exactly the same as **Import/Export** → **Import/Export Parameter Data**.



- **Online parameterization**
In a running simulation, the current parameters are loaded into the model (online parameterization). Exactly the same as **Online** → **Go Online**.



- **Offline parameterization**
Ends online parameterization. Exactly the same as **Online** → **Go Offline**.



- **View parameter scope binding**
Opens a list of parameters which are imported and exported. The names are also specified of the blocks from which they are exported or into which they are imported respectively. Exactly the same as **View** → **Parameter Scope Binding**.

4.7.5 Working with LABCAR-PA

This section contains information on how to work with the LABCAR-PA.

This includes:

- "Setting Options" on page 337
This section describes the setting of different options.
- "Editing Parameters" on page 342
This section contains information on editing parameters.
- "Reference Files" on page 347
This section contains information on reference files.
- "Exporting Parameters to a File" on page 348
This section describes exporting model parameters to m- or DCM files.
- "Importing Parameters from a File" on page 350
This section describes importing model parameters from m- or DCM files.
- "The Filter and Sorting Function" on page 353
This section contains the filter and sort functions for the parameter table.
- "The Search Function for Model Parameters" on page 355
This section describes the search function for model parameters
- "Determining the Parameter Scope Binding" on page 358
This section describes how you can identify parameters which are imported or exported from blocks of the model.
- "Online Parameterization" on page 359
This section describes the parameterization of running models.

4.7.6 Setting Options

This section describes the setting of the following options.

- "General Options" on page 337
- "Settings for Reference Files" on page 338
- "Paths for Working, Model and Model Documentation Directories" on page 339
- "Options for Parameter Import and Export" on page 340
- "History Options" on page 341

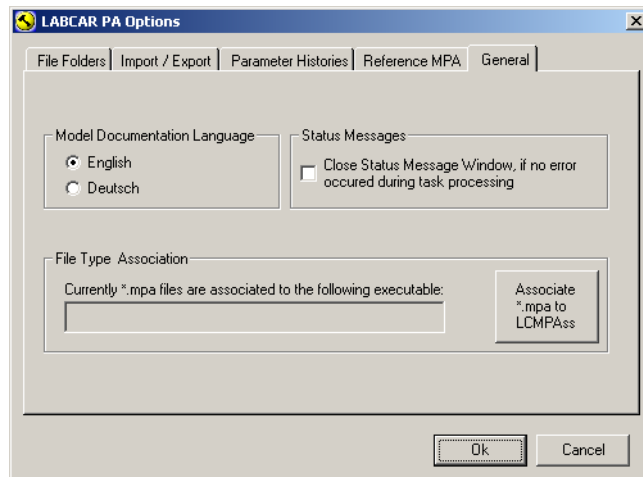
General Options

The following settings can be made in the "General Options" section:

- specify the model block documentation language
- specify the response of the window for status messages
- change the * .mpa file type association

To change one or more of these options, proceed as follows:

- Select **Edit** → **Options**.
The "LABCAR-PA Options" window opens.
- Select the "General" tab.



To specify the model documentation language

- Select either "English" or "Deutsch" under "Model Documentation Language".

Response of windows for status messages

- If you want the "Status Message Window" to close automatically once a task has been processed successfully, activate the option "Close Status Message Window, if no error occurred during task processing".

To change the *.mpa file type association

Normally the *.mpa file type is associated with multimedia data in Microsoft Windows operating systems, i.e. when you double-click a file with this association, a multimedia player is started.

- If you want files with the extension "mpa" to be linked with LABCAR-PA, click the **Associate *.mpa to LCMPAss** button under "File Type Association".

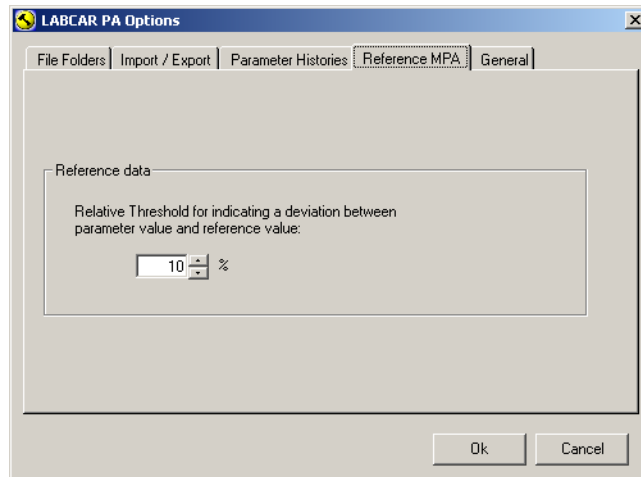
Settings for Reference Files

If a reference file is open and a parameter in the loaded parameter file differs by a certain amount from the amount of the parameter in the reference file, the entry in the parameter table is assigned an arrow (pointing up or down).

The deviation (in %), above which this should be displayed, can be set.

To set the threshold for displaying a deviation

- Select **Edit** → **Options**.
The "LABCAR-PA Options" window opens.
- Select the "Reference MPA" tab.



- To define the threshold, click on the arrows or enter the percentage rate required in the box.

Paths for Working, Model and Model Documentation Directories

To work with LABCAR-PA, the paths for the following directories have to be set correctly:

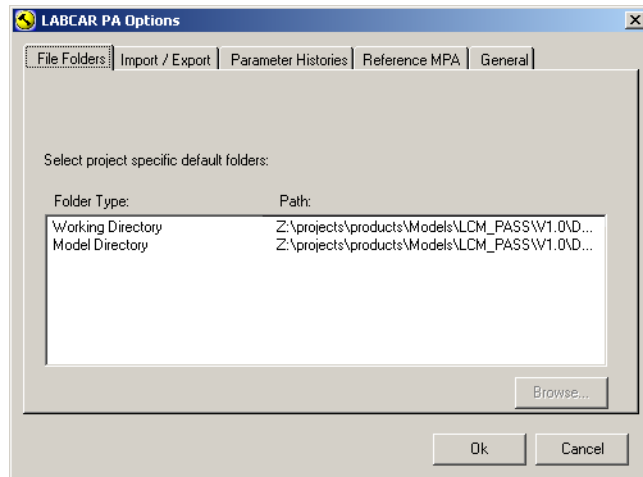
- Working Directory
This is the directory for all files containing parameterization data (*.mpa, *.dcm, *.m)
- Model Directory
This path is not used in this version.
- Model Documentation Directory
Path for block and model documentation (*.pdf)

To change the working directory

To change the working directory, proceed as follows:

- Select **Edit** → **Options**.

The "LABCAR-PA Options" window opens.



- Select the "File Folders" tab.
- Use your mouse to select "Working Directory".
- Click the **Browse** button.
The "Select Folder" window opens.
- Select a drive and a folder.
- Click **OK**.

Use exactly the same procedure to change the model and model documentation directories.

Options for Parameter Import and Export

The following options can be specified for the import and export of parameters from or to files:

- whether ETAS attributes are included when importing and exporting from and to Matlab Simulink m-files
- the method for generating ASAM labels

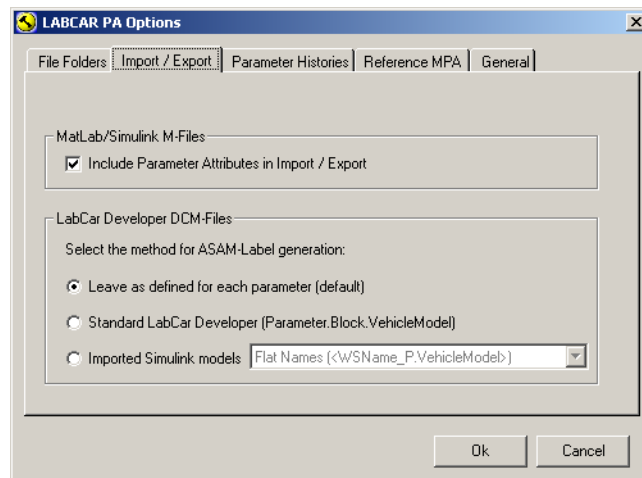
To change import/export options

To change these options, proceed as follows:

- Select **Edit** → **Options**.

The "LABCAR-PA Options" window opens.

- Select the "Import/Export" tab.



In the "Matlab/Simulink M-Files" box, you can activate the "Include Parameter Attributes in Import/Export" option. In this case, the parameter attributes are written into the M-file or left out of it.

In the "LabCar Developer DCM-Files" box, you can specify the method for generating ASAM labels. You can choose from the following options:

- Leave as defined for each parameter (default)
The ASAM label, as used in the mpa file ("ModelPath" attribute).
- Standard LabCar Developer (Parameter.Block.VehicleModel)
LabCar Developer Syntax: Parameter_name.Block_name.Model_name
- Imported Simulink Models
Options used with Simulink model import:
 - Flat Names (<WSName_P.VehicleModel>)
Workspace_name.Model_name
 - Standard SL Import
With complete hierarchy
 - SL Import Reverse Quantity Names
With complete hierarchy in reverse order

History Options

A history entry is added for every parameter changed every time the parameter file is edited. If the parameter file is often edited, it gradually becomes larger – to reduce the size of the file again, the history entries can be deleted.

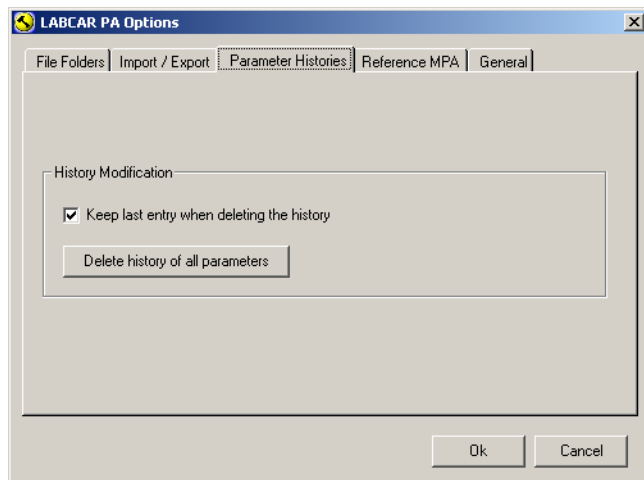
You can set an option specifying that the last entry should be retained on deletion.

To delete the history

To delete the parameter history, proceed as follows:

- Select **Edit** → **Options**.
The "LABCAR-PA Options" window opens.

- Select the "Parameter Histories" tab.



- If you want to keep the last entry in the parameter history each time, activate "Keep last entry when deleting the history".
- To delete the history, click **Delete history of all parameters**.

4.7.7 Editing Parameters

This section contains information on editing parameters. This includes:

- "Editing Parameter Values" on page 342
This section describes how to edit parameters depending on their format.
- "Editing Minimum and Maximum Values" on page 344
Contains tips on editing minimum and maximum values.
- "The Table Editor" on page 344
Describes how to work with the Table Editor.
- "Viewing and Editing the Parameter History" on page 346
Provides information on how to work with the parameter history.

Editing Parameter Values

As parameters can have different formats, how to edit them also differs.

The following cases are described:

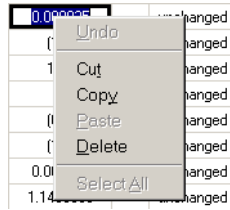
- "The parameter is an integer or a floating point number" on page 342
- "The parameter is a 1D (curve) or 2D table (map)" on page 343
- "The parameter is a logical value" on page 343
- "Parameter is imported" on page 344

The parameter is an integer or a floating point number:

To edit a parameter, click the relevant table cell in the "Value" column with the left-hand mouse button. The table cell is then in edit mode and can be changed.

V_Rail	m ³	0.000035	unchanged	volume of the rail
--------	----------------	----------	-----------	--------------------

When the cell is in edit mode, the following shortcut menu is opened by clicking the right-hand mouse button:



The following operations can be carried out using this menu:

- undoing a change just carried out
- cutting the value
- copying the value or an entire range
- deleting the value
- marking an entire string

Note

Depending on what has happened before, only certain menu items may be active.

The parameter is a 1D (curve) or 2D table (map):

To edit a table, double-click the relevant table cell in the "Value" column with the left-hand mouse button. The Table Editor for this parameter then opens.

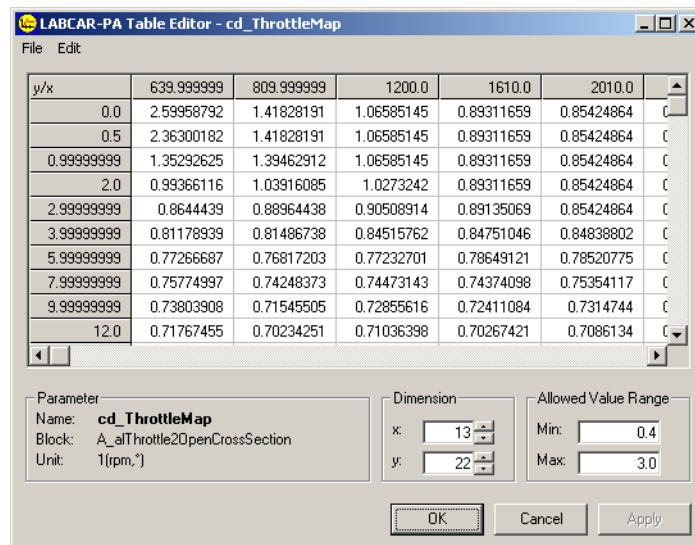
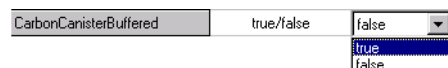


Fig. 4-15 The Table Editor

The section "The Table Editor" on page 344 describes how to operate the Table Editor.

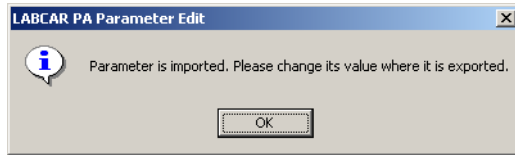
The parameter is a logical value:

If the parameter consists of a true/false value, a dropdown list from which you can select the value you require opens when you click the mouse button.



Parameter is imported:

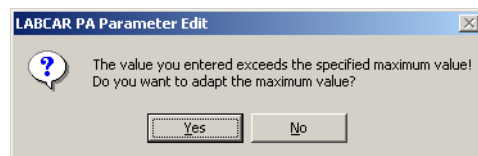
If a parameter is imported, it cannot be edited in this block; this is also indicated by the gray background. If, however, you still try to edit the parameter, the following message is displayed:



The parameter can only be modified in the block from which it was exported. To determine which block the parameter was exported from, select **Select source parameter ("exported")** from the shortcut menu. The relevant block is then activated and its parameters shown in the table – the exported parameter is shown against a blue background.

Editing Minimum and Maximum Values

The minimum and maximum value specifications for the parameter values are limits chosen deliberately for the model to work well. If you specify a value which is either above or below this kind of limit in LABCAR-PA, a warning is issued.



The user then has the chance to cancel this change or execute it anyway.

For "integer or floating point number" parameters, it is also possible to change these values directly.

The Table Editor

If you double-click the value of a parameter that consists of a 1D or 2D table, the Table Editor opens.

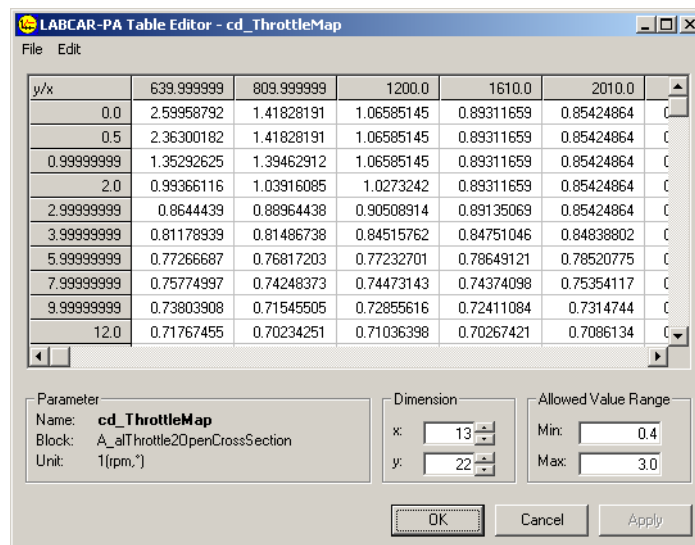


Fig. 4-16 The Table Editor with the "cd_ThrottleMap" Parameter

A single table cell can be edited by clicking the left-hand mouse button.

When the cell is in edit mode, the following shortcut menu is opened by clicking the right-hand mouse button.

734.4	741.2	747.15	753.1
729.3	723.35	718.25	713.15
708.05	702.1	697.0	

Rückgängig			
Ausschneiden			
Kopieren			
Einfügen			
Löschen			
Alles markieren			

The following operations can be carried out using this menu:

- undoing a change just carried out
- cutting the value
- copying the value
- deleting the value
- marking an entire string

Note

Depending on what has happened before, only certain menu items may be active.

The user interface of the Table Editor consists of one menu and several fields which are described below.

Menu:

The Table Editor menu contains the following menu items

- **File** → **Open Reference Table Viewer**
This opens the relevant table of the reference file. This contains one menu item **Data** → **Copy Reference Data to Table**, which copies the entire contents of the reference table to the current table.
- **Edit** → **Edit Axes**
This allows you to change the x-/y-axis values of the table (see "Dimension" Field" on page 346).
- **File** → **Import from File**
This allows you to import data from an m- or DCM file.

"Parameter" Field:

This field contains information on the parameter.

- Name
The name of the parameter
- Block
The name of the block to which the parameter belongs
- Unit
The physical unit of the parameter

"Dimension" Field:

The number of columns and rows in the table is shown in this field – and can also be changed here.

Fig. 4-16 on page 344 shows a fuel density map. If you want to extend this map by one column, increase the x-value by "1" in the "Dimension" field.

A new column is added with an x-value incremented by "1" – the y-values of this new column are acquired by extrapolation.

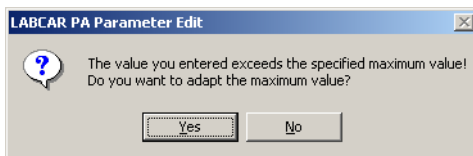
If you need a new x-value "2200", select the **Edit** → **Edit Axes** menu and change "2001.0" to "2200.0". The new y-values for this x-value have to be changed accordingly.

The same is also true of adding columns.

If the number of columns/rows is reduced, the last row/column is deleted each time.

"Allowed Value Range" Field:

This is where the minimum and maximum function values are set. If one of these thresholds is exceeded, the following warning is displayed:



If you select **No**, the change is canceled; if you select **Yes**, the relevant minimum or maximum value of the current change is adjusted.

Viewing and Editing the Parameter History

Right-clicking one of the lines of the parameter table opens a shortcut menu which contains the **View History** menu item. If this is selected, the "Parameter History" dialog box opens.

In addition to the name and the block the parameter belongs to, this window contains the history of the parameter.

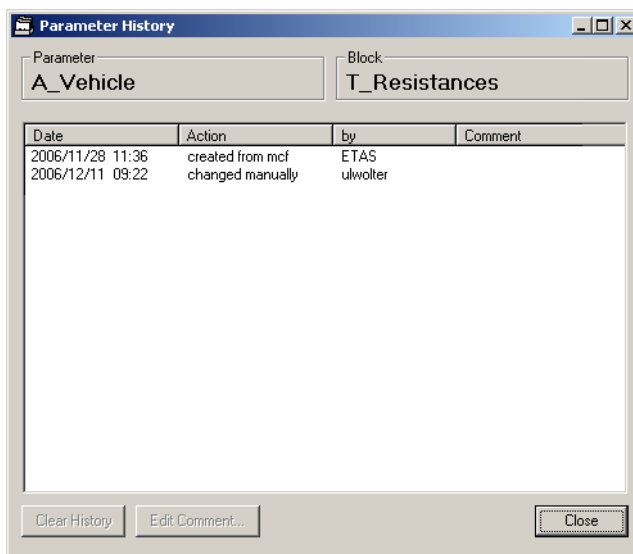


Fig. 4-17 Parameter History

The entries can be deleted by clicking the **Clear History** button. In the options, you can set whether all entries should be deleted or whether the last entry should be retained (see the section "History Options" on page 341).

If you select an entry from the table using the mouse, the **Edit Comment** button becomes active.

If you click this button, a dialog box opens in which you can enter a comment on the particular change.



4.7.8 Reference Files

Reference files help you when you parameterize your model. They can, for example, contain a set of "time-tested" parameters from which you make changes.

If you want to save a parameterization as a reference, select **File** → **Set To Reference File**. The name of the file plus the character string "Reference_N" is then offered to you in the file selector window as the file name.

To load a saved reference file, select **File** → **Open Reference**. You can select the required file from the file selector window.

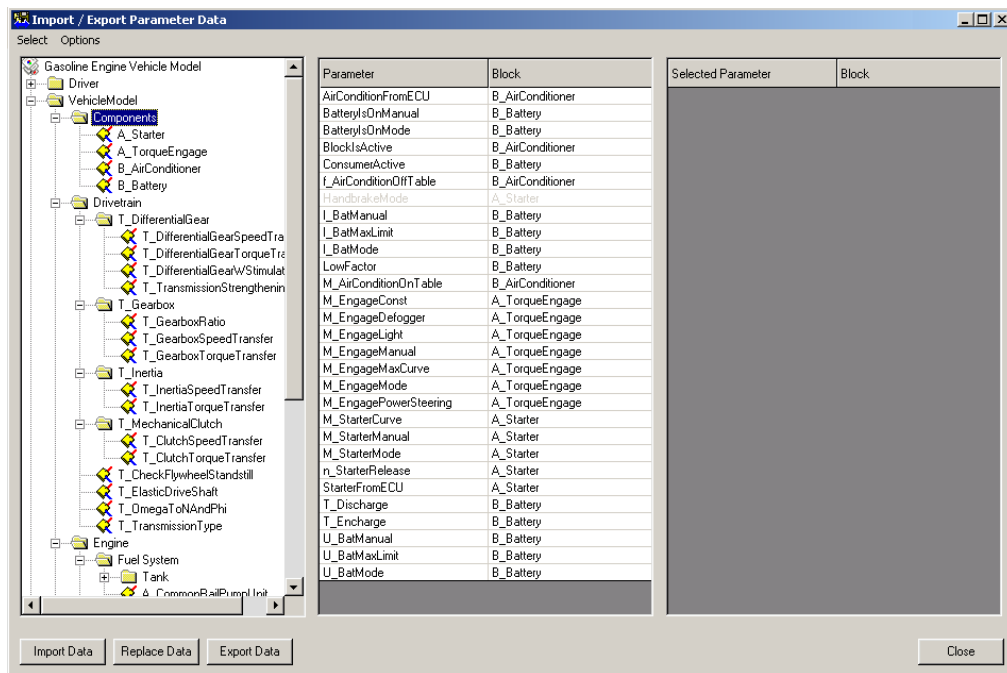
4.7.9 Exporting Parameters to a File

This section describes exporting parameters to m- or DCM files.

To export parameters to a new file ("Export Data")

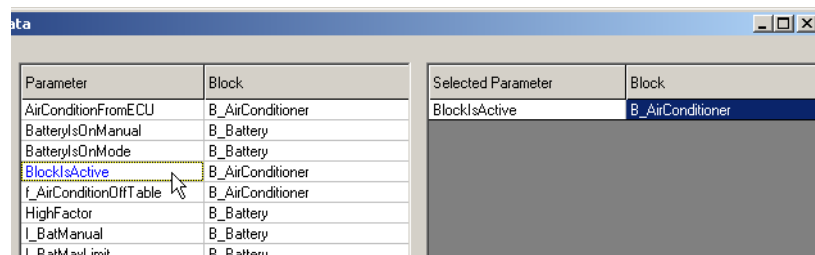
- To export parameters to m-, DCM or mpa files, select **Import/Export** → **Import/Export Parameter Data**.

The "Import/Export Parameter Data" dialog box opens.



The hierarchical model view is displayed on the left-hand side of the window. The middle section lists all parameters from the selected model component (here: GEVM → VehicleModel → Components). The parameters selected for export are shown on the right.

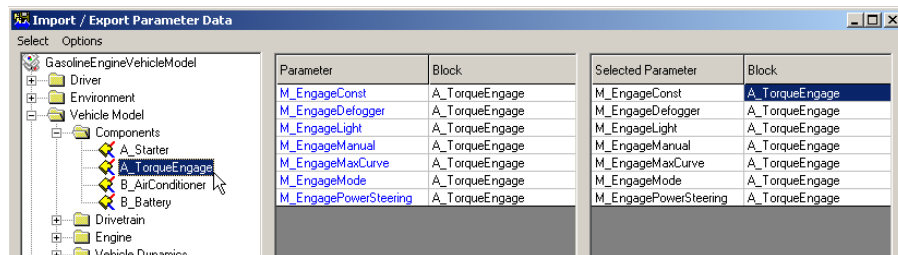
For example, to export the "BlocksActive" parameter of the "B_AirConditioner" block, click this entry. The selected parameter is then entered under "Selected Parameters". Parameters added to this list are shown in blue.



- To undo the selection of a parameter, click the relevant parameter under "Selected Parameters".
- To select all parameters of a selected block or model component (e.g. from the "A_TorqueEngage" block), activate this block or model component.
- Select **Select** → **Select All**.

or

- Press <CTRL+A>.
- All parameters are then added to the "Selected Parameters" list.



- Selecting **Select** → **Deselect All** deletes all parameters selected so far from the "Selected Parameters" column.
- Click the **Export Data** button to open the file selector window.
- Enter a file name or select an existing file.
- Select the type of file.
- Click **Save**.

The new file is created.

If the file already exists, you are asked to confirm that you want to replace the existing file.

To export parameters to an existing file ("Replace Data")

- To export parameters to existing m- or DCM files, proceed as described above.
- To export, click the **Replace Data** button.

If the selected parameters are already in the file, these entries are overwritten; otherwise they are added to the file.

To set options for parameter export

With **Options** → **Im-/Export Attributes in Matlab M-Files**, you can set whether, when exporting parameters, only the parameter itself is to be written to the file (deactivated) or whether further attributes, such as "Min.", "Max." or "Modification State" etc., should also be included (activated).

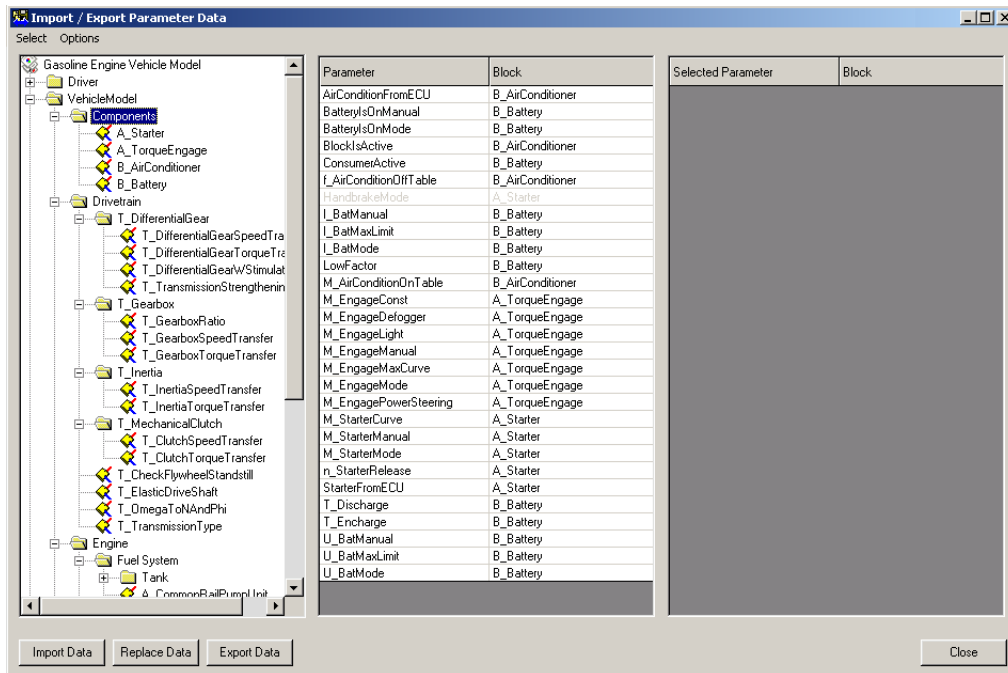
4.7.10 Importing Parameters from a File

Parameters can be imported from a file in exactly the same way. The prerequisite for this is, however, that the selected parameter exists in the file – otherwise an error message is issued.

To select parameters to be imported

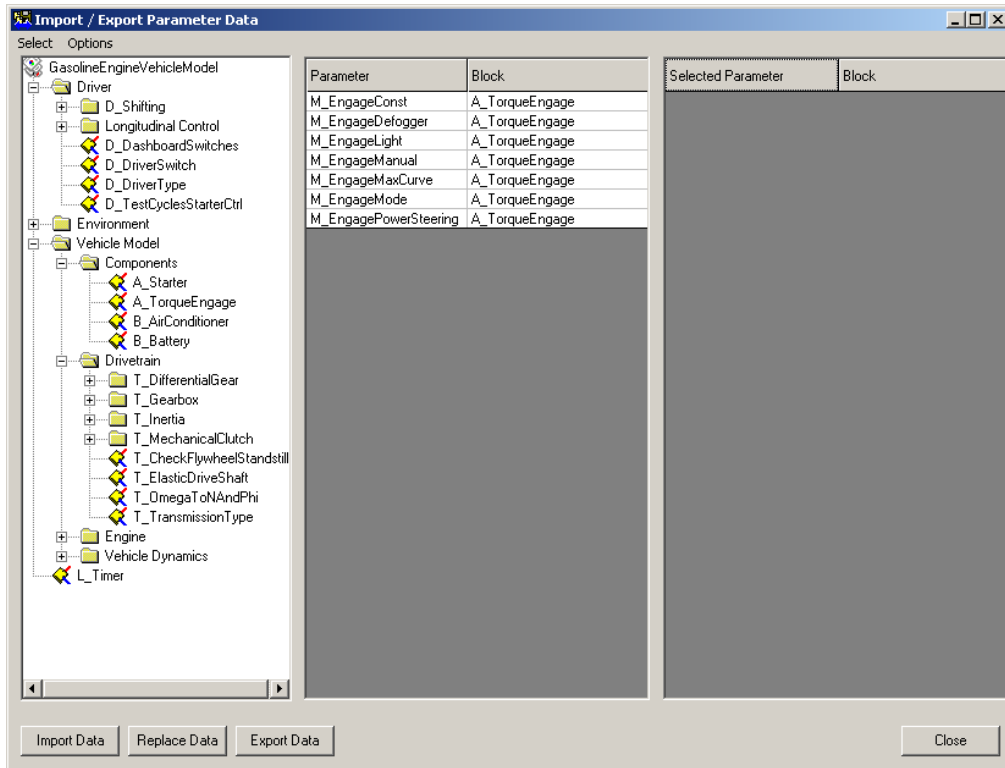
- To export parameters from an m- or DCM file, select **Import/Export** → **Import/Export Parameter Data**.

The "Import/Export Parameter Data" dialog box opens.



- Using **Options** → **Im-/Export Attributes in Matlab M-File** you again have the opportunity to change the import/export options (see "Options for Parameter Import and Export" on page 340).

- For example, to import parameters from the "A_TorqueEngage" block, click this entry.

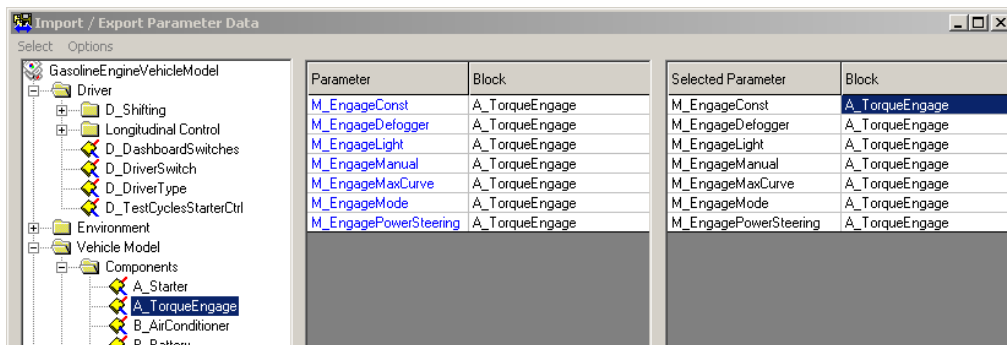


The parameters of the block are listed in the middle column of the window.

Parameters with the scope "imported" are shown gray in the table.

- Select **Select** → **Select All**.

The selected parameters (here: all parameters of the "A_TorqueEngage" block) are entered under "Selected Parameters". Parameters added to this list are shown in blue.



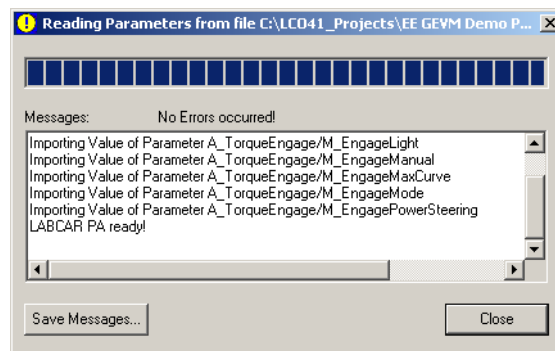
To select a file

- Click the **Import Data** button.
A file selector window opens.
- Select the type of file.
- Enter a file name or select a file.
- Click **Open**.
The parameters are imported.

During import, the values of the selected parameters are replaced by those from the imported file. There are two possible scenarios:

- the new parameter value is exactly the same as the old one.
- the new parameter value is not the same as the old one.

This is shown again in the "Import Parameters" status window.

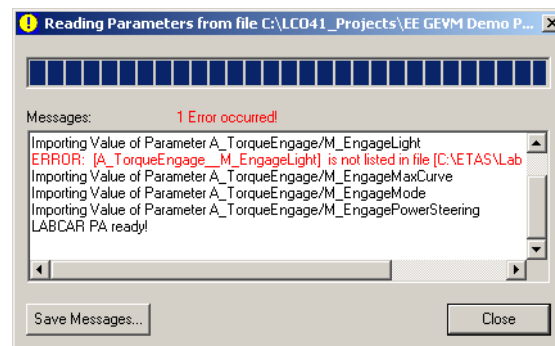


Note

As these are not error messages, the window only stays open once the import is completed, if the relevant option (see "Response of windows for status messages" on page 338) is deactivated.

Error Messages

If the selected parameter is not available in the file from which it should be imported or if other errors occur when it is read, a corresponding error message is issued.



These messages can also be saved in ASCII or RTF files.

To save an error message

- Click the **Save Errors** button.
A file selector window opens.
- Select the type of file (ASCII or Rich Text Format).
- Enter a file name.
- Click **Save**.

Note

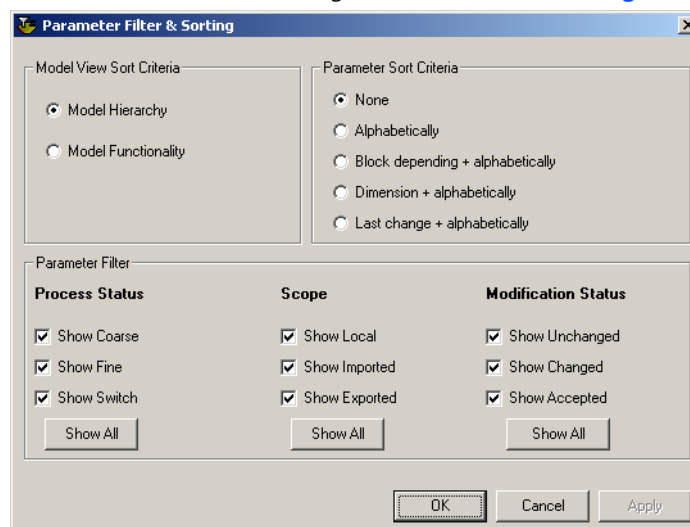
In an RTF file, the relevant error messages are displayed in red; this is not possible in an ASCII file.

4.7.11 The Filter and Sorting Function

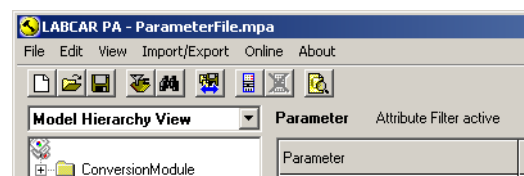
A filter can be used with the parameters displayed in the parameter table; it is used to limit the display of parameters for example to only those with the status "changed" or those which have the scope "exported".

You can also specify the structure view of the model and the order criteria for the parameters in the table.

The function is invoked using **Edit** → **Filter Settings**.



If a filter is set, the user interface indicates this above the parameter table ("Attribute Filter Active").



The meaning of the individual filter criteria is described below.

Model View Sort Criteria

This is where you can specify how the model is to be displayed on the left-hand side of the user interface.

- **Model Hierarchy**

This view shows the model in its hierarchical structure.

- **Model Functionality**

This view shows the model grouped according to its functional units (e.g. Driver, Engine, etc.).

This function is identical to the selection in the dropdown menu at the top left of the user interface.

Parameter Sort Criteria

This is where sort criteria are specified for the model components and the parameters listed on the right-hand side of the user interface.

- **None**

No explicit sorting – the parameters are listed in the order in which they occur in the individual blocks in the model view (corresponds to the order in the mpa file).

- **Alphabetically**

Alphabetical sorting

- **Block depending and alphabetically**

The first sort criterion is the block name – the parameters of a block are ordered alphabetically.

- **Dimension and alphabetically**

The first sort criterion is the dimension of the parameter (2D-table, 1D-table, scalar, array) – parameters with the same dimension are sorted alphabetically.

- **Last change and alphabetically**

First sort criterion is the change date; the second is the parameter name (in alphabetical order).

Parameter Filters

This is where parameters can be displayed in the table according to specific criteria.

Process Status

- Show Coarse

All parameters with the parameterization status "coarse" are displayed.

- Show Fine

All parameters with the parameterization status "fine" are displayed.

- Show Switch

All parameters with the parameterization status "switch" are displayed.

All three filter options are activated simultaneously if you use the **Show All** button; you are effectively deactivating the filter by doing this.

Scope

- Show Local

All parameters with the "local" scope are displayed.

- Show Imported
All parameters with the "imported" scope are displayed.
- Show Exported
All parameters with the "exported" scope are displayed.

All three filter options are activated simultaneously if you use the **Show All** button; you are effectively deactivating the filter by doing this.

Modification Status

- Show Unchanged
All parameters with the "unchanged" status are displayed.
- Show Changed
All parameters with the "changed" status are displayed.
- Show Accepted
All parameters with the "accepted" status are displayed.

All three filter options are activated simultaneously if you use the **Show All** button; you are effectively deactivating the filter by doing this.

Note

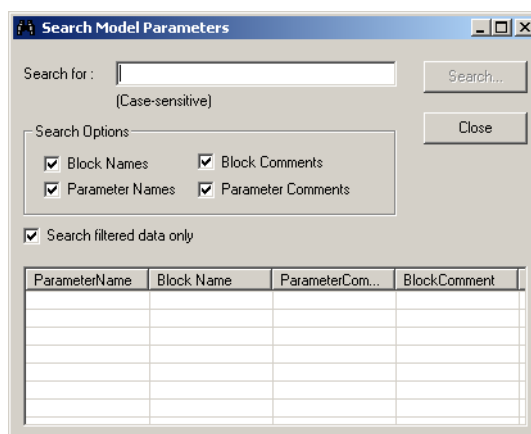
It is pointless to deactivate all three filter options within a group: this would result in an empty table and the criteria of the two other groups would have no effect.

This is why deactivating all three filter options of a criterion has no effect and all parameters are displayed.

4.7.12 The Search Function for Model Parameters

You can use the search function to search for parameters in the model with different options.

The function is invoked using **Edit** → **Search Model Parameters** or with <CTRL+F>.



A string is searched for which occurs in certain attributes of the parameter. These attributes are:

Search Options

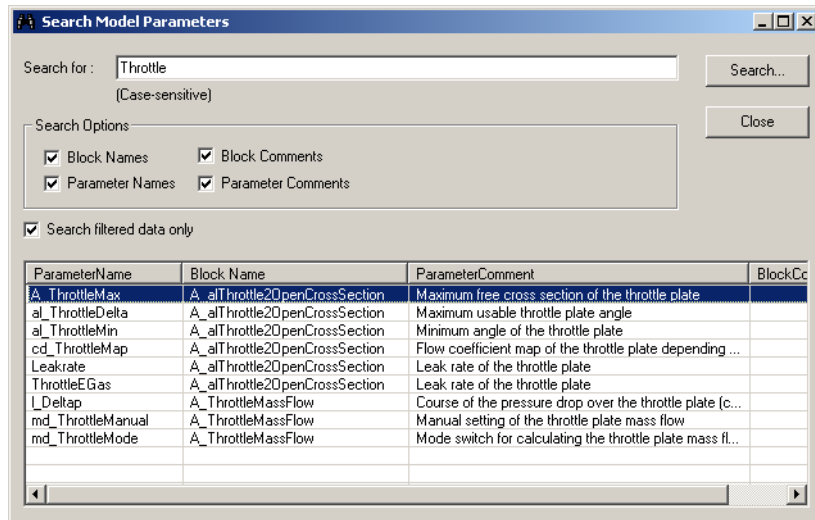
- **Block Names**
The string has to occur in the "Block Name" attribute.
- **Block Comments**
The string has to occur in the "Block Comment" attribute (the same string which is displayed in the tooltip when the mouse pointer points to a block).
- **Parameter Names**
The string has to occur in the parameter name.
- **Parameter Comments**
The string has to occur in the "Comment" attribute.

Search filtered data only

If a filter is used for the parameter table, this option results in the search criteria only being used on the parameters displayed.

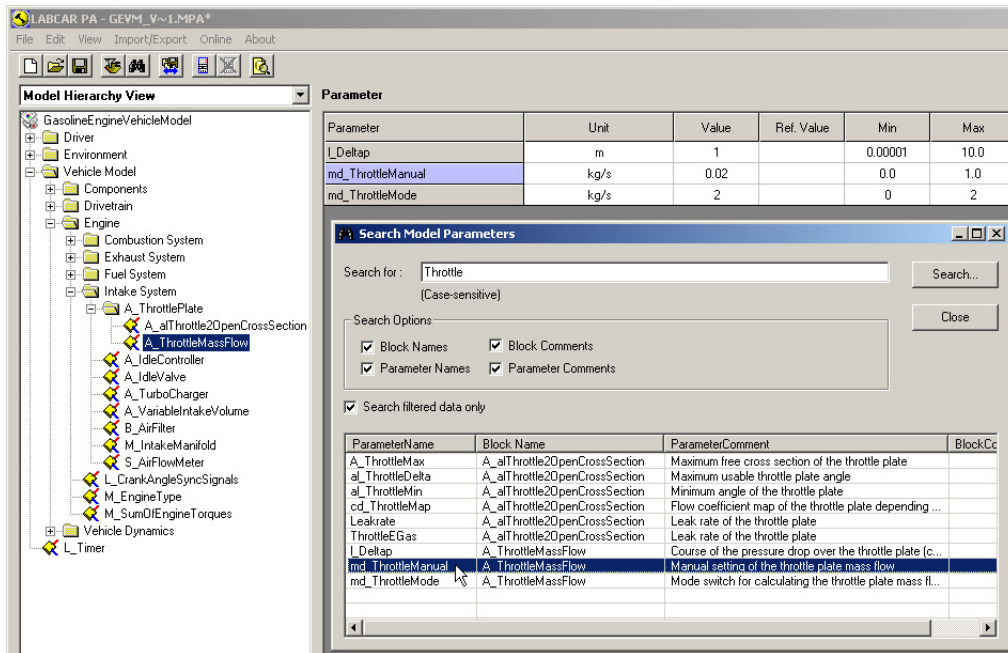
"Throttle" Example

In this example, a search is being executed for the "Throttle" string - all criteria are active.



If the search for the string was restricted to the parameter name, for example, the parameters "Leakrate" and "I_Deltap" would not appear in the results list.

If you click on a parameter in the results list, the block is selected in the structure view. All parameters of this block are then displayed in the parameter table – the one selected in the search list is shown on a blue background.



4.7.13 Determining the Parameter Scope Binding

As the value of imported parameters can only be changed in the block from which they were exported, it is possible to create a table of exported and imported parameters.

The function is invoked using [View](#) → [Parameter Scope Binding](#) .

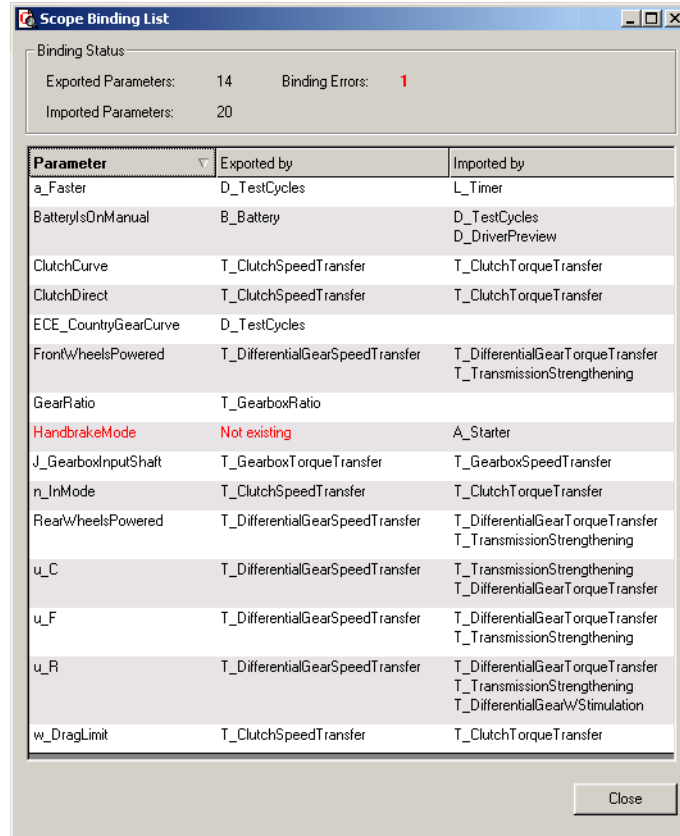


Fig. 4-18 Scope Binding List

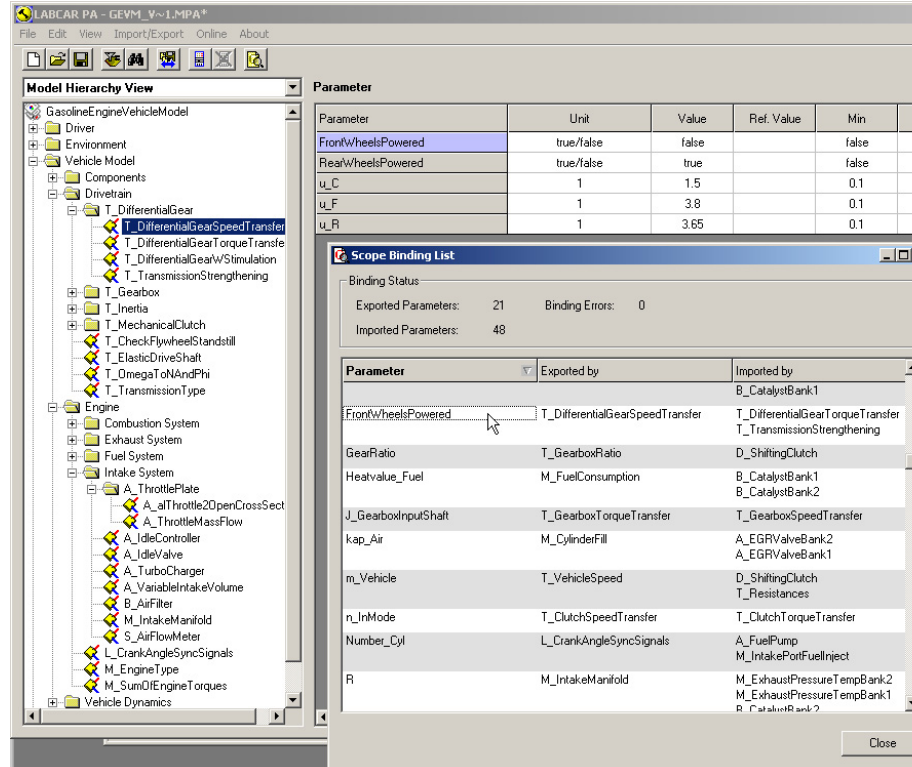
The "Parameter" column lists all parameters which are exported from one block ("Exported by" column) and imported into others ("Imported by" column).

The parameters can be sorted alphabetically either starting with a or starting with z (simple click of the mouse on the column title "Parameter").

The table in Fig. 4-18 on page 358 also shows when there are inconsistencies in "Scope Binding": The "Handbrake Mode" parameter is imported by the block "A_Starter" but not exported by any block. It is then displayed in red.

Inconsistencies of this type are actually checked when the file is opened; the user is informed of any inconsistencies.

By clicking the parameter name or the name of the exporting block, the block is selected in structure view. All parameters of this block are then displayed in the parameter table – the exported parameter is shown on a blue background.



The equivalent is true when you click on the name of the importing block.

4.7.14 Online Parameterization

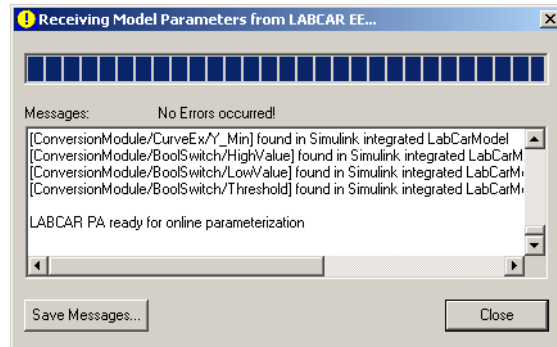
During online parameterization, parameters changed in LABCAR-PA are written directly to an experiment running in ETAS EE.

To start online parameterization

- Launch the experiment in ETAS EE.
- Open the parameter file (*.mpa) of the model in LABCAR-PA.

- Select **Online** → **Go Online**.

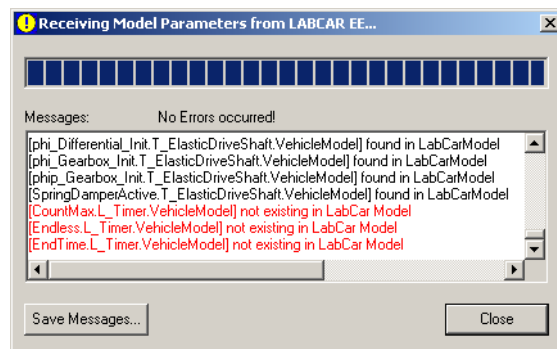
A connection to the experiment environment is established and a search is carried out in the current project for all parameters which occur in the current parameter file.



The parameters found are assigned the "Online" icon in the parameter table.

Parameter	
Parameter	Unit
A_ThrottleMax	m²
A_Vehicle	m ²
Accelerator_Start	0..1
AcceleratorManual	0..1
AcceleratorMode	0(1)3
AcceleratorOffroad	1
AcceleratorReduced	true/false
AddtorqueOfInertia	0/1
AddTorqueOfInertia	true/false
AFR	1
AFR	1

Error messages are also displayed if a parameter was not found.



- Select **Close**.

You can now change parameters of the current model in the LABCAR-PA parameter table.

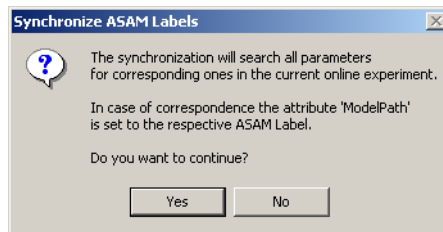
Error: Parameter not found

There are two reasons why a parameter from the parameter file cannot be found in the current model:

- the ASAM labels do not correspond to each other (perhaps because of another name in the Simulink import).
This can be solved by synchronizing the label (see below).
- the parameter does not exist in the current model

To synchronize ASAM labels

- Select **Online** → **Synchronize ASAM Labels**.
The current experiment is searched for ASAM labels which correspond to parameters in the parameter file.



Note

Synchronization can only be successful in those cases where ASAM labels in the current model (for example after a Simulink import) were created in accordance with the rules described in section "Options for Parameter Import and Export" on page 340.

If there is a match, the ASAM label in the parameter file ("ModelPath" attribute) is given the name used in the current model.

If the ASAM labels come from an imported Simulink model, these synchronized ASAM labels are assigned a modified "Online" icon in the parameter table.

Parameter		
Parameter	Unit	Value
Aggressiveness	0..1	0.5
altitude	m	0
AltitudeType	1(1)5	1
b_altitude3	m	10
Body_Car_cmx	[m]	-1.4
Body_Car_cmy	[m]	0
Body_Car_cmz	[m]	0.5

4.8 LABCAR-CCI V5.4.4 (Calibration Connector for INCA)

LABCAR-CCI V5.4.4 is an add-on to LABCAR-OPERATOR V5.4.4 that enables the user to access INCA devices such as ETK. This means parts of the INCA functionality can be controlled from within LABCAR-OPERATOR. The ASAM labels of the INCA experiment are available in the experiment environment (ETAS EE) in the "Workspace Elements" window.

The following figure shows the integration of LABCAR-CCI V5.4.4 in LABCAR-OPERATOR V5.4.4.

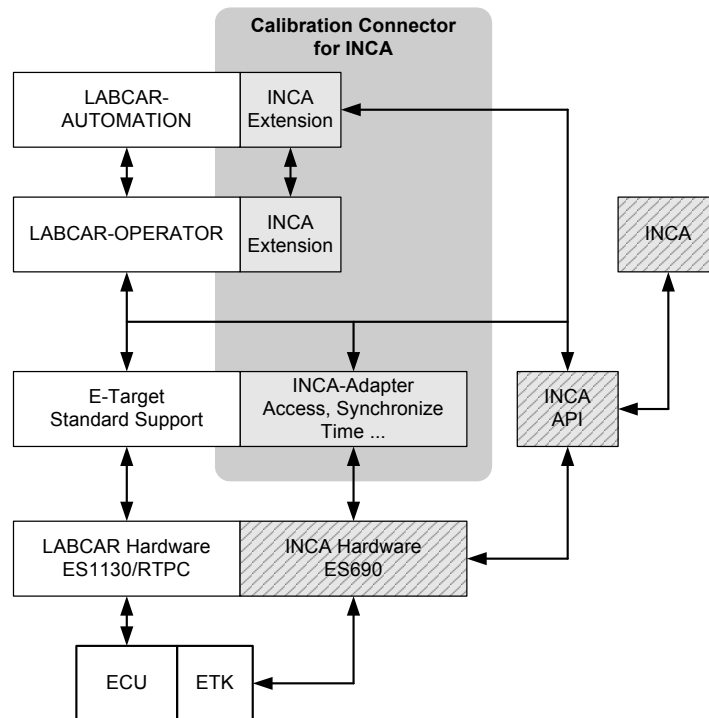


Fig. 4-19 The Integration of LABCAR-CCI V5.4.4 in LABCAR-OPERATOR

LABCAR-CCI V5.4.4 consists of three main components:

- The INCA add-on in LABCAR-OPERATOR V5.4.4 for measuring and calibrating with INCA devices.
This also contains additional functionality such as the opening of a specific INCA database, initializing INCA devices, starting and stopping an INCA measurement.
- The INCA add-on in LABCAR-AUTOMATION is an adaptation to the new architecture, in particular with regard to time synchronization and E-Target.
- The INCA add-on for managing access to INCA devices and for synchronizing the timestamp of E-Target on the one hand and INCA measure values on the other.

This section contains information on the following:

- "System Requirements" on page 363

This section describes the hardware and software requirements for using LABCAR-CCI V5.4.4.

- "Working with LABCAR-CCI V5.4.4" on page 363

This section contains information on working with LABCAR-CCI.

4.8.1 System Requirements

This section describes the necessary hardware and software components for using LABCAR-CCI V5.4.4.

INCA Version

The current list of the INCA versions required for operating LABCAR-CCI V5.4.4 is in the menu **? → Help** in the document "LABCAR-OPERATOR 5.x.y - Software Compatibility List".

Note

INCA itself is not a component part of LABCAR-CCI V5.4.4. You must purchase an INCA license separately. LABCAR-CCI V5.4.4 cannot be used without INCA.

INCA can be installed on the same computer as LABCAR-OPERATOR.

Hardware

Accessing the ECU is specified in the INCA experiment and can take place using, for example, the CAN-, K-Line or ETK interface of an ES690 Compact Module.

4.8.2 Working with LABCAR-CCI V5.4.4

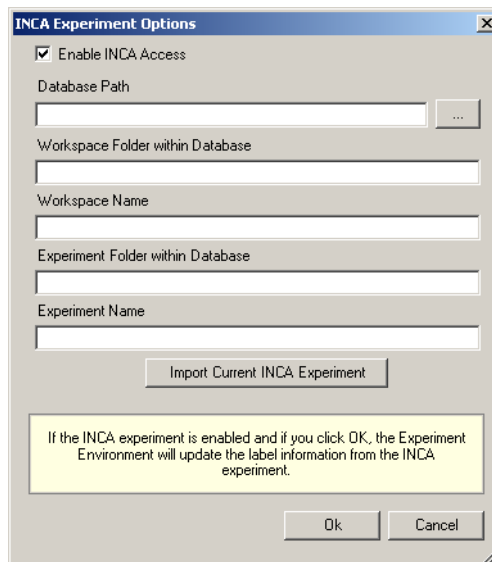
As with LABCAR-OPERATOR experiments, a specific INCA experiment is accessed in the experiment environment ETAS EE.

To do this, proceed as follows:

To specify the INCA experiment

- Launch ETAS EE.
- If necessary, download your LABCAR-OPERATOR experiment to the target and launch the simulation (now or later).
- Select **Experiment → INCA Options**.
The "INCA Experiment Options" window opens.

- Activate the "Enable INCA Access" option.
The various fields for specifying other data are activated.



- You can now either enter the necessary data manually:
 - Database Path: path to the INCA database
 - Workspace Folder within Database
 - Workspace Name
 - Experiment Folder within Database
 - Experiment Name

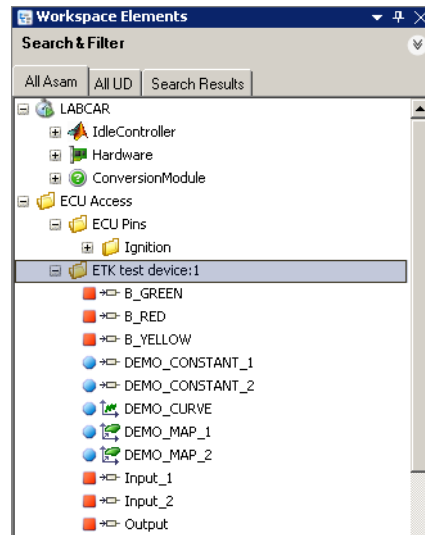
or

- Open the experiment in INCA and click **Import Current INCA Experiment**.

The experiment data is imported into the relevant fields (see above).

- Click **OK**.

To determine the measure variables/parameters of your INCA experiment, a "Connect" to the experiment is now run automatically.



When you then save the experiment in ETAS EE (**File** → **Save Experiment**), the connection to the INCA experiment specified above is also saved.

To connect the INCA experiment

To reconnect the INCA experiment, proceed as follows:

- Start measuring using **Experiment** → **Download** → **INCA**.

or

- Click the relevant icon in the toolbar.



The experiment opens in INCA and the ASAM labels of the INCA experiment can be accessed in ETAS EE.

To initialize INCA hardware

If your INCA hardware is not correctly initialized after a download (e.g. because the hardware or the ECU were powered off), proceed as follows:

- To initialize the INCA hardware select **Experiment** → **Initialize INCA Hardware**.

To start measuring

- Start measuring using **Experiment** → **Start Measurement** → **INCA**.

or

- Click the relevant icon in the toolbar.

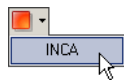


Measuring starts.

To stop measuring

- Stop measuring using **Experiment** → **Stop Measurement** → **INCA**.

or

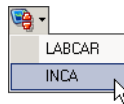


- Click the relevant icon in the toolbar.
Measuring stops.

To interrupt the connection to the INCA experiment

- To interrupt the connection to the experiment, select **Experiment** → **Stop Measurement** → **INCA**.

or



- Click the relevant icon in the toolbar.
The INCA experiment is closed.

5 **Appendix**

LABCAR-NIF V5.4.4 and LABCAR-LCX V5.4.4 use the StringTemplate and ANTLR libraries under the following licenses:

StringTemplate Software License

The BSD License]

Copyright (c) 2008, Terence Parr

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

ANTLR License

[The BSD License]

Copyright (c) 2003–2008, Terence Parr

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 **ETAS Contact Addresses**

ETAS HQ

ETAS GmbH

Borsigstraße 14

70469 Stuttgart

Germany

Phone: +49 711 3423-0

Fax: +49 711 3423-2106

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

Figures

Fig. 2-1	The LABCAR-IP User Interface with an Open Project.....	13
Fig. 2-2	The Main Window of LABCAR-IP	18
Fig. 2-3	Project Explorer.....	19
Fig. 3-1	A LABCAR Module	25
Fig. 3-2	Modules of a LABCAR-OPERATOR Project.....	26
Fig. 3-3	The Connection Manager	33
Fig. 3-4	The Components "IdleCon" (top) and "Integrator" (bottom) of the "ControllerTest" Project	55
Fig. 3-5	The ASCET Project "ControllerTest" (on the left) and the integrated Module in the "Workspace Elements" Window of ETAS Experiment Environment (on the right).	56
Fig. 3-6	CAN Network with Four Nodes and Five Messages	83
Fig. 3-7	Physically present (UuT) and Simulated Nodes	84
Fig. 3-8	Messages to and from the Residual Bus	84
Fig. 3-9	The CAN Editor.....	87
Fig. 3-10	The CAN Network	88
Fig. 3-11	The Effect of the Option "Spread Send Frames" (see Text).....	100
Fig. 3-12	Signal Flow with Send Messages (see Text)	120
Fig. 3-13	The CAN Module in the Workspace Elements	125
Fig. 3-14	Message GUI for a Receive Message	129
Fig. 3-15	Message GUI for a Send Message	130
Fig. 3-16	The "CAN Bus Monitor" Instrument	131
Fig. 3-17	The LIN Editor	134
Fig. 3-18	The LIN Network.....	135
Fig. 3-19	The LIN Module in the Workspace Elements.....	160
Fig. 3-20	Instrument for a LIN Frame	166
Fig. 3-21	PDU Tx Level Statistics.....	171
Fig. 3-22	PDU Rx Level Statistics	172

Fig. 3-23	Structure of an HiL System.....	197
Fig. 3-24	FiL System.....	197
Fig. 3-25	The Start Page of the FiL Wizard	198
Fig. 3-26	Selecting the A2L File.....	199
Fig. 3-27	Making ECU Outputs Available for Connecting with Model Inputs.....	204
Fig. 3-28	Assignment of Tasks to ECU Rasters	205
Fig. 3-29	Instrument for Controlling the FiL Function	209
Fig. 3-30	Adding a Hook to the OS Configuration	217
Fig. 3-31	Activity Diagram	219
Fig. 3-32	The "RT Plugins" Tab in ETAS EE	222
Fig. 3-33	The Experiment Signals	231
Fig. 3-34	The Signals and Pins of ECU and Hardware.....	233
Fig. 3-35	Signals, Pins and Virtual Connections (Dashed Lines).....	234
Fig. 3-36	LABCAR Port Blocks.....	235
Fig. 3-37	The Connection Manager User Interface	236
Fig. 3-38	Global Settings	243
Fig. 3-39	Additional Measure Variables with "Enable OS Monitoring"	244
Fig. 3-40	Calculating "runtime_TurnaroundStatistic"	245
Fig. 3-41	Task Settings	245
Fig. 3-42	The "Processes" Field of the "OS Configuration" Tab.....	247
Fig. 3-43	The "Tasks" Field of the "OS Configuration" Tab.....	248
Fig. 3-44	The "OS Configuration" Tab (extended view)	251
Fig. 3-45	Architecture of a Multi-RTPC System.....	256
Fig. 3-46	Clock Synchronization with PTP	256
Fig. 3-47	Data Exchange between Real-Time PCs.....	257
Fig. 3-48	Settings for a Real-Time PC in the Web Interface	261
Fig. 4-1	The ETAS EE GUI.....	266
Fig. 4-2	The Experiment Explorer	268
Fig. 4-3	The "Workspace Elements" Window.....	271
Fig. 4-4	The "Datalogger" Tab	278
Fig. 4-5	The "Signal Generator" Tab	282
Fig. 4-6	The "Signal Management" Tab	285
Fig. 4-7	The "Signal Editor" Tab	295
Fig. 4-8	The "Instruments" Window.....	300
Fig. 4-9	The "Workspace Elements" Window	307
Fig. 4-10	Shortcut Menu for Parameters (with Active Experiment).....	308
Fig. 4-11	The "Parameter Files" Folder Shortcut Menu	314
Fig. 4-12	Variants of ECU Hardware, ECU Software and the Corresponding Test Projects 317	
Fig. 4-13	Master Project and Files for Deviations (Variants) from the Master Project .	318
Fig. 4-14	The "Mapping Files" Folder Shortcut Menu	323
Fig. 4-15	The Table Editor.....	343
Fig. 4-16	The Table Editor with the "cd_ThrottleMap" Parameter.....	344
Fig. 4-17	Parameter History	346
Fig. 4-18	Scope Binding List.....	358
Fig. 4-19	The Integration of LABCAR-CCI V5.4.4 in LABCAR-OPERATOR	362

Index

A

ASCET module
 integrating 52

B

Bus Communication Monitor 186

C

C code module
 adding a process 61
 API functions 68
 changing 70
 defining a calibration variable 61
 defining a measure variable 60
 defining a model output 60
 defining a module inport 59
 manual creation 58
C code modules
 Tutorial 57
CAN Editor 87
CAN Monitor 131
Check RTPC version 36
Clean Intermediate Files 35
Connection Manager 236
CPU Load 245
Cycle Time 130

D

Device Alias 201

E

ECU pin list 232
ETAS Contact Addresses 369
ETK Device Alias 201
Expert Mode 89

F

FiL modules 197
Filter function 353
Floating Point Exception
 Stop Simulation 247
Functional Mock-up Unit 72
Function-Call subsystems 39
 settings 39
Function-in-the-Loop 197

H

Hardware pin list 232

I

Ignore Min/Max 119
INCA experiment
 specifying 363
INCA hardware
 initializing 365
Inports 234
Instrument "RT ECU Access" 209
inTrigger 130
IP address
 Real-Time PC 37

L

- LABCAR ID 16
- LABCAR-NIF 173
- LABCAR-NIL 133
- LABCAR-OPERATOR
 - creating a project 167
- LDF container 135
- Limit Min/Max to bit length 118
- LIN
 - frames 148
 - parts 145
 - schedule tables 144
 - user-defined C-Code 155
- LIN Editor 134
- LIN module
 - creating 134
 - in LABCAR-EE 160
- LIN network 141

M

- Main menu
 - LABCAR-EE 267
- Mapping
 - reloading 324
- Mapping file 321
 - activating 324
 - adding to project 324
 - creating 322
 - deleting permanently 325
 - editing 323
 - excluding from project 325
 - renaming 324
- MATLAB search paths 45
- Min/Max values 119
- Multi-RTPC project 20, 272

N

- Network Configuration 169
- Network Integration FlexRay 173
- Network Integration LIN 133
- Network Module 167
- Network Module Ports 171
- NIF modules 173

O

- Online parameterization 359
- Operation
 - conventions 11
- OS configuration 243
- Outports 234

P

- Parameter
 - exporting to a file 348
 - importing from a file 350
 - searching 355
- Parameter Combination 317
- Parameter files
 - managing 314
- Parameter scope binding 358
- Parameter table 328
 - entries 329
 - scope 328
 - shortcut menu 330
- Parameterization files 46
- Parameters 306
- PDU Tx Level Statistics 171
- Processes 247
- Project Explorer 19
- Project options
 - general 35

R

- Real-time operating system 243
- Real-time plugins 210
- Reference file 347
- RT-Plugin Builder 211

S

- Script recorder 303
- Search function 355
- Shared Object File 75
- Signal conversion 240
- Signal generators 282
- Signal list
 - adding signals 301
- Sorting function 353
- Spread Send Frames 100

T

- Table Editor 344
- Task settings 245
- Toolbar 17
 - LABCAR-IP 268

V

- Variants 317
- Virtual connections 231

W

- Wake On LAN 20, 36

Z
Zip And Go 16

