ETAS

LABCAR-OPERATOR V5.4.1

Benutzerhandbuch

Copyright

V5.4.1 R01 DE - 08.2016

Inhalt

1 Einführung

Dieses Handbuch richtet sich an Fachpersonal in den Bereichen Entwicklung und Test von Kfz-Steuergeräten. Fachwissen im Bereich Mess- und Steuergerätetechnik wird vorausgesetzt.

1.1 Über dieses Handbuch

Dieses Handbuch enthält Informationen zum Arbeiten mit LABCAR-OPERATOR V5.4.1.

Das Handbuch besteht aus folgenden Kapiteln:

- **„Die Bedienoberfläche von LABCAR-IP“ auf Seite 13**

  In diesem Kapitel finden Sie eine Beschreibung der Bedienoberfläche von LABCAR-IP.

  – „Funktionsübersicht“ auf Seite 13
  – „Das Hauptmenü von LABCAR-IP“ auf Seite 14
  – „Die Werkzeugleiste“ auf Seite 17
  – „Das Hauptfenster“ auf Seite 18
  – „Der Project Explorer“ auf Seite 19
  – „Das Log-Fenster“ auf Seite 19
  – „Projekteinstellungen“ auf Seite 20

- **„Arbeiten mit LABCAR-IP“ auf Seite 23**

  Dieses Kapitel enthält die Beschreibung der Integration verschiedener Module in ein LABCAR-OPERATOR-Projekt und der anschließenden Generierung von Code für ein in ETAS Experiment Environment ausführbares Experiment.

  – „Modularer Simulationscode“ auf Seite 25
  – „MATLAB/Simulink-Modelle“ auf Seite 38
  – „ASCET-Module“ auf Seite 52
  – „C-Code-Module“ auf Seite 60
  – „CAN-Module (Network Integration CAN)“ auf Seite 91
  – „LIN-Module (Network Integration LIN)“ auf Seite 143
  – „NIF-Module (Network Integration FlexRay)“ auf Seite 179
  – „FiL-Module“ auf Seite 206
  – „Real-Time Plugins“ auf Seite 222
  – „Module zur Signalkonvertierung“ auf Seite 237
  – „Hardware konfigurieren mit dem RTIO-Editor“ auf Seite 242
  – „Der Connection Manager“ auf Seite 243
  – „Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)“ auf Seite 256

- **„ETAS Experiment Environment - Eine Übersicht“ auf Seite 279**

  ETAS Experiment Environment (ETAS EE) dient zur Ausführung eines LABCAR-OPERATOR-Experimentes. In diesem Kapitel finden Sie eine Übersicht über die Bedienoberfläche und die Funktionen von ETAS EE.

  – „Die Bestandteile der Bedienoberfläche“ auf Seite 280
  – „Der Experiment Explorer“ auf Seite 282
  – „Das Fenster „Workspace Elements““ auf Seite 284
  – „Der Hauptarbeitsbereich“ auf Seite 290
  – „Mit Parametern arbeiten“ auf Seite 321
  – „Parameterdateien bearbeiten mit LABCAR-PA V1.0“ auf Seite 341
  – „LABCAR-CCI V5.4.1 (Calibration Connector for INCA)“ auf Seite 378

1.2 Umgang mit dem Handbuch

Darstellung von Information

Alle vom Anwender auszuführenden Tätigkeiten werden in einem sogenannten „Use-Case“-Format dargestellt. D.h., dass das zu erreichende Ziel zuerst in der Titelzeile kurz definiert wird, und die jeweiligen Schritte, die notwendig sind, um dieses Ziel zu erreichen, dann in einer Liste aufgeführt werden. Die Darstellung sieht wie folgt aus:

Zieldefinition

eventuelle Vorabinformation...

- Schritt 1

  eventuelle Erläuterung zu Schritt 1...
- Schritt 2

  eventuelle Erläuterung zu Schritt 2...
- Schritt 3

  eventuelle Erläuterung zu Schritt 3...

eventuelle abschließende Bemerkungen...

konkretes Beispiel:

Erstellen einer neuen Datei

Vor dem Erstellen einer neuen Datei darf keine andere geöffnet sein.

- Wählen Sie **Datei** → **Neu**.

  Die Dialogbox „Datei Erstellen“ erscheint.
- Geben Sie den Namen für die Datei im Feld „Dateiname“ ein.

  Der Dateiname darf nicht mehr als 8 Zeichen lang sein.
- Klicken Sie **OK**.

Die neue Datei wird erstellt und unter dem von ihnen angegebenen Namen abgelegt. Sie können nun mit der Datei arbeiten.

Typografische Konventionen

Folgende typografischen Konventionen werden verwendet:

Wählen Sie **Datei** → **Öffnen**.	Menübefehle werden fett/blau dargestellt.
Klicken Sie **OK**.	Schaltflächen werden fett/blau dargestellt.
Drücken Sie <EINGABE>.	Tastaturbefehle werden in spitzen Klammern in Kapitälchen dargestellt.
Das Dialogfenster „Datei öffnen“ erscheint.	Namen von Programmfenstern, Dialogfenstern, Feldern u.ä. werden in Anführungszeichen gesetzt.
Wählen Sie die Datei `setup.exe` aus.	Text in Auswahllisten, Programmcode, sowie Pfad- und Dateinamen werden in der Schriftart `Courier` dargestellt.
Eine Konvertierung zwischen den Datentypen logisch und arithmetisch ist *nicht* möglich.	Inhaltliche Hervorhebungen und neu eingeführte Begriffe werden *kursiv* gesetzt

Wichtige Hinweise für den Anwender werden so dargestellt:

Hinweis

Wichtiger Hinweis für den Anwender.

2 Die Bedienoberfläche von LABCAR-IP

In diesem Kapitel finden Sie eine Beschreibung der Bedienoberfläche von LABCAR-IP.

Die einzelnen Abschnitte enthalten Informationen über:

2.1 Funktionsübersicht

Wenn Sie LABCAR-IP gestartet und ein Projekt geladen haben, hat die Bedienoberfläche etwa folgendes Aussehen.

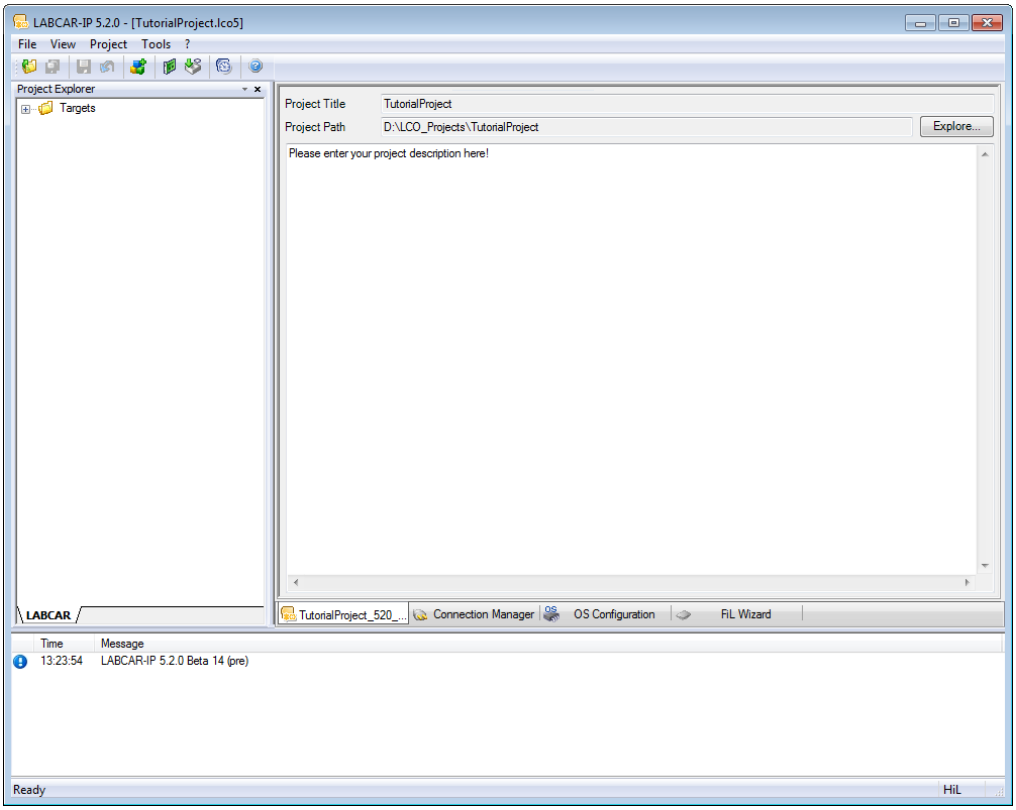


Abb. 2-1 Die Bedienoberfläche von LABCAR-IP mit geöffnetem Projekt

Die Oberfläche besitzt folgende Bedienelemente:

- Die Menüleiste

  Die Beschreibung der Menüs von LABCAR-IP finden Sie im Abschnitt „Das Hauptmenü von LABCAR-IP“ auf Seite 14.

- Die Werkzeugleiste

  Die Beschreibung der Elemente in der Werkzeugleiste finden Sie im Abschnitt „Die Werkzeugleiste“ auf Seite 17.
- Das Hauptfenster

  Die verschiedenen Register enthalten Projektinformationen, den Connection Manager, den CAN Editor und die OS Configuration (siehe „Das Hauptfenster“ auf Seite 18).
- Der Project Explorer

  Siehe „Der Project Explorer“ auf Seite 19
- Das Log-Fenster

  Siehe „Das Log-Fenster“ auf Seite 19

2.2 Das Hauptmenü von LABCAR-IP

Das Hauptmenüs enthält die folgenden Einträge:

- „Das Menü „File““ auf Seite 14

  In diesem Menü sind alle dateibezogenen Aktionen gruppiert.
- „Das Menü „View““ auf Seite 15

  In diesem Menü wird die Sichtbarkeit der einzelnen Fenster gesteuert.
- „Das Menü „Project““ auf Seite 15

  In diesem Menü finden sich alle Funktionen, die zur Projektbearbeitung, Projektverwaltung und Experimentsteuerung benötigt werden.
- „Das Menü „Tools““ auf Seite 16

  Über dieses Menü können Sie verschiedene Werkzeuge aufrufen.
- „Das Menü „?““ auf Seite 16

  Dieses Menü enthält Informationen zu LABCAR-IP.

Hinweis

Die folgende Beschreibung der Menüstruktur von LABCAR-IP soll nur einen Überblick vermitteln – ausführliche Informationen zu den einzelnen Funktionen erhalten Sie in den entsprechenden Kapiteln.

2.2.1 Das Menü „File“

In diesem Menü sind alle dateibezogenen Aktionen gruppiert.

- **File → New Project**

  Zum Erstellen neuer Projekte
- **File → Open**

  Öffnet ein vorhandenes Projekt
- **File → Close**

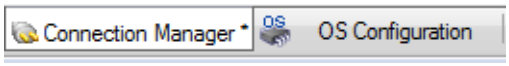
  Schließt das geöffnete Projekt

- **File → Save All**

  Speichert das gesamte Projekt inklusive aller Änderungen in CAN Editor, Connection Manager etc.
- **File → Save As**

  Speichert das gesamte Projekt unter einem anderen Namen
- **File → Save**

  Speichert den Inhalt des momentan aktiven Dokuments (CAN Editor, Connection Manager etc.). Register mit unspeicherten Änderungen sind an dem Stern hinter dem Namen des Registers zu erkennen.

- **File → Discard**

  Nimmt alle ungespeicherten Änderungen im momentan aktiven Dokument zurück (siehe **File → Save**)
- **File → Recent Projects →**

  Zeigt die Liste der vier zuletzt geöffneten Projektdateien
- **File → Exit**

  Beendet LABCAR-IP

2.2.2 Das Menü „View“

In diesem Menü wird die Sichtbarkeit der einzelnen Fenster gesteuert.

- **View → Status Bar**

  Aktiviert/deaktiviert die Statusanzeige im unteren Rand des Hauptfensters
- **View → Tool Bar**

  Aktiviert/deaktiviert die Anzeige der Werkzeugleiste
- **View → Project Explorer**

  Öffnet/schließt den Project Explorer
- **View → Log Window**

  Öffnet/schließt das Fenster „Log Window“

2.2.3 Das Menü „Project“

In diesem Menü finden sich alle Funktionen, die zur Projektbearbeitung, Projektverwaltung und Experimentsteuerung benötigt werden.

- **Project → Add Module**

  Öffnet den „Add Module Wizard“.
- **Project → RTIO Editor**

  Öffnet den RTIO-Editor zur Konfiguration der angeschlossenen Hardware
- **Project → Build**

  Öffnet das Dialogfenster für die Codegenerierung
- **Project → Options**

  Öffnet das Dialogfenster, in dem Projekteinstellungen vorgenommen werden (siehe auch „Projekteinstellungen“ auf Seite 20)

- **Project → ECU Pin List**

  Öffnet das Fenster „ECU Pin & HW Pin Properties“
- **Project → OS Configuration**

  Öffnet das Register „OS Configuration“

2.2.4 Das Menü „Tools“

Über dieses Menü können Sie verschiedene Werkzeuge aufrufen.

- **Tools → HSP Update Tool**

  Startet die aktuelle Version des HSP Update Tools
- **Tools → Open LABCAR-IP Log File**

  Öffnet die Protokolldatei, die ausführliche Information zur Verfügung stellt
- **Tools → Zip Log Files**

  Öffnet ein Dialogfenster zum Packen der vorhandenen Log-Dateien
- **Tools → Zip And Go**

  Ermöglicht das Packen des aktuellen Projekts in eine Datei
- **Tools → Open Experiment Environment**

  Öffnet die Experimentierumgebung ETAS EE
- **Tools → View Version Info**

  Zeigt eine Liste mit den Versionen aller Softwarekomponenten
- **Tools → Options**

  Öffnet ein Dialogfenster zum Editieren der LABCAR ID

2.2.5 Das Menü „?“

Dieses Menü enthält Informationen zu LABCAR-IP.

- **? → Help**

  Öffnet eine HTML-Datei mit Links zur Dokumentation und weiteren Informationen
- **? → About**

  Informationen über installierte LABCAR Softwareprodukte und deren Versionen
- **? → Contact**

  Zeigt ETAS Kontaktinformationen
- **? → License**

  Öffnet den ETAS Lizenzmanager

2.3 Die Werkzeugleiste

Die Werkzeugleiste enthält folgende Funktionen:

1 Open

Öffnet ein Dokument (identisch mit **File → Open**)

2 Save All

Speichert das gesamte Projekt inklusive aller Änderungen in CAN Editor, Connection Manager etc. (identisch mit **File → Save All**)

3 Save

Speichert den Inhalt des momentan aktiven Dokuments (CAN Editor, Connection Manager etc.). (identisch mit **File → Save**)

4 Discard

Nimmt alle ungespeicherten Änderungen im momentan aktiven Dokument zurück (identisch mit **File → Discard**)

5 Add Module

Öffnet den Add Module Wizard, mit dem Module hinzugefügt werden können.

6 Edit Hardware Configuration

Startet den RTIO-Editor (identisch mit **Project → RTIO Editor**)

7 Build LABCAR Project

Startet Codegenerierung (identisch mit **Project → Build**)

8 Open Experiment Environment

Öffnet die Experimentierumgebung ETAS EE (identisch mit **Tools → Open Experiment Environment**)

9 Help

Informationen zu LABCAR-OPERATOR V5.4.1 (identisch mit **Help → Help**)

2.4 Das Hauptfenster

Im Hauptfenster sind alle wichtigen Funktionen von LABCAR-IP vereinigt – der Übersichtlichkeit halber sind diese Funktionen über verschiedene Register verteilt.

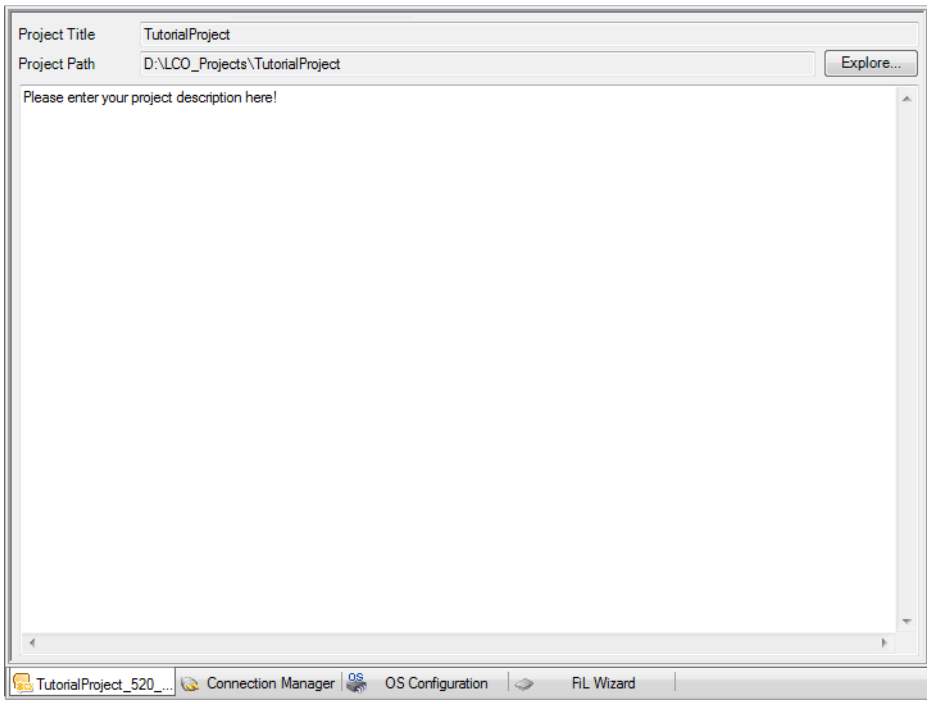


Abb. 2-2 Das Hauptfenster von LABCAR-IP

Das Register „Project Info“

In diesem Register wird der Projektname und der Speicherort der Projektdateien angezeigt. Zudem können hier in einem Textfeld Informationen zum Projekt eingegeben werden.

Das Register „Connection Manager“

Dieses Register umfasst alle Funktionen des Connection Managers (siehe auch „Der Connection Manager“ auf Seite 243).

Das Register „OS Configuration“

In diesem Register können alle Einstellungen für das Echtzeit-Betriebssystem vorgenommen werden (siehe auch „Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)“ auf Seite 256).

Das Register „FiL Wizard“

In diesem Register findet die Erstellung und Integration von FiL-Modulen statt („FiL-Module“ auf Seite 206).

2.5 Der Project Explorer

Im Dockingfenster des Project Explorers werden alle projektrelevanten Objekte und Dokumente verwaltet.

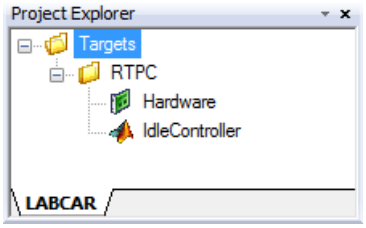


Abb. 2-3 Project Explorer

Targets

Der Ordner „Targets“ enthält einen Ordner für das bei der Projekterstellung spezifizierte Experimentaltarget. Für jedes Experimentaltarget wird ein Unterordner mit dessen Namen angelegt.

- ***Experimentalsystemname* („RTPC“)**

  Über das Kontextmenü des Ordners können Sie folgende Funktionen ausführen:

  – **Add Module**

  – **Edit OS Configuration**

  Jeder dieser Ordner enthält:

  – Alle Module (Simulinkmodelle, C-Code, CAN, Signal Conversion Blocks etc.)

  – Die HW-Konfiguration

  Über das Kontextmenü einer Datei dieses Ordners können Sie folgende Funktionen ausführen:

  – **Edit**

    Öffnet den entsprechenden Editor

  – **Update**

    Das Modul wird im entsprechenden Editor geöffnet und eventuelle Änderungen aktualisiert.

  – **Remove** (nicht bei Hardwarekonfiguration)

2.6 Das Log-Fenster

Register „Messages“

In diesem Fenster werden Informationen und Fehlermeldungen von LABCAR-OPERATOR V5.4.1 ausgegeben.

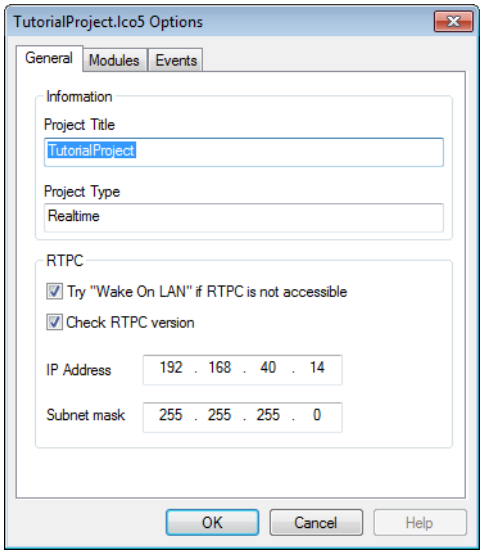
Um die Ausgaben in diesem Fenster zu löschen, rechtsklicken Sie in diesem Fenster und wählen Sie **Clear Log**.

2.7 Projekteinstellungen

Bei einem LABCAR-OPERATOR-Projekt gibt es eine Reihe von Einstellungen, die sich auf die verschiedenen Komponenten von LABCAR-OPERATOR beziehen. Diese Einstellungen sind über **Project → Options** zugänglich.

2.7.1 Registerkarte „General“

Diese Registerkarte enthält zum Einen Informationen über den Projektnamen („Project Title“) und den Projekttyp („Projekt Type“).



Außerdem können Einstellungen für den Real-Time PCvorgenommen werden:

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

  Ist diese Option gewählt und der Real-Time PC antwortet nicht, wird versucht, diesen über Wake-On-LAN zu starten.

- Check RTPC version

  Mit dieser Option wird die Version des erkannten Real-Time PC ermittelt. Beim Build wird dann geprüft, ob diese Version mit dem aktuellen LABCAR-OPERATOR zusammenarbeitet.

- IP Address

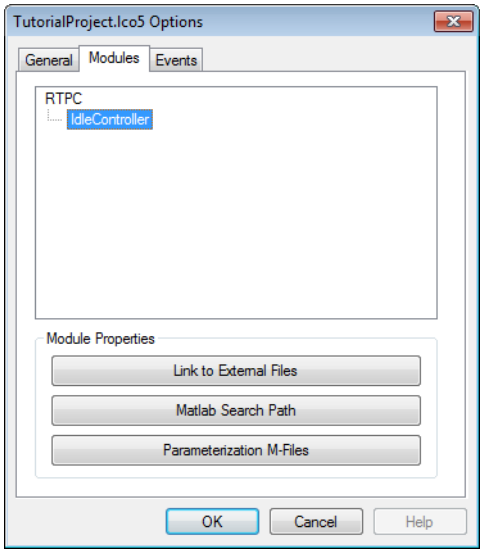
  Die IP-Adresse des Real-Time PC, der diesem Projekt zugeordnet ist (für Multi-RTPC-Projekte).

- Subnet Mask

  Subnetzmaske zur Festlegung des Netzpräfix

2.7.2 Registerkarte „Modules“

Wenn Sie mit dem Modeling Connector for Simulink arbeiten, werden hier die Module des Projekts wie z.B. das Simulink-Modell und externer C-Code aufgeführt und verwaltet.



Eine genaue Beschreibung der Funktion dieser Einstellungen finden Sie im Abschnitt „Weitere Einstellungen für das Simulink-Modell“ auf Seite 45.

2.7.3 Registerkarte „Events“

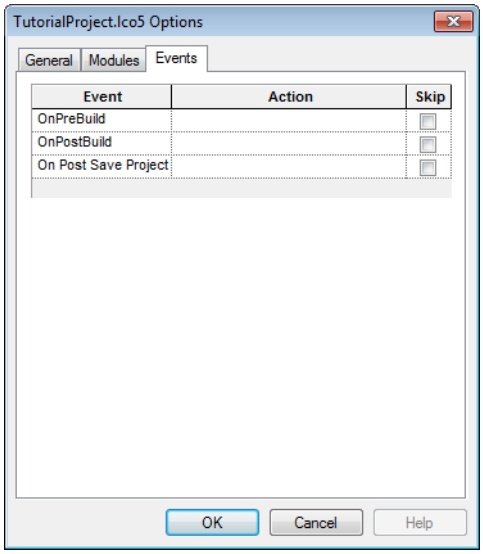
Während des Ablaufs eines Projektes können Skripts eingebunden werden, die vor oder nach dem Buildprozess ausgeführt werden. Als Skript gilt hier jede unter Windows ausführbare Datei (*.exe, *.bat, *.cmd, etc.).

Skript mit Ereignis verbinden

- Wählen Sie **Project** → **Options**.

  Das Fenster „*Projektname* properties“ wird geöffnet.
- Wählen Sie das Register „Events“.

  Dieses enthält eine Liste von Ereignissen (Events), mit denen die Ausführung von Skripts (Action) verknüpft werden kann.



- Um ein solches Skript zu spezifizieren, klicken Sie die Zelle neben dem jeweiligen Event.
- Klicken Sie auf den Pfeil am Rand der Zelle und wählen Sie **<browse >**.

  Ein Dateiauswahlfenster wird geöffnet, in dem Sie die auszuführende Skript-Datei wählen können.
- Markieren Sie **Skip**, wenn Sie ein spezifiziertes Skript (temporär) nicht ausführen wollen.

3 Arbeiten mit LABCAR-IP

Dieses Kapitel enthält die Beschreibung der Integration verschiedener Module in ein LABCAR-OPERATOR-Projekt und der anschließenden Generierung von Code für ein in ETAS Experiment Environment ausführbares Experiment.

Die einzelnen Abschnitte enthalten Informationen über:

- „Modularer Simulationscode“ auf Seite 25

  Dieser Abschnitt gibt einen kurzen Überblick über die Integration von Modulen verschiedenster Herkunft zu einem LABCAR-OPERATOR-Gesamtprojekt.

- „MATLAB/Simulink-Modelle“ auf Seite 38

  In diesem Abschnitt erhalten Sie Informationen zur Integration von MATLAB/Simulink-Modellen:

- „ASCET-Module“ auf Seite 52

  In diesem Abschnitt finden Sie Informationen zur Integration von ASCET-Modulen in ein LABCAR-OPERATOR-Projekt.

- „C-Code-Module“ auf Seite 60

  Neben MATLAB/Simulink-Modellen und ASCET-Modulen ist es in LABCAR-IP auch möglich, vom Anwender definierten C-Code in das Projekt zu integrieren.

- „CAN-Module (Network Integration CAN)“ auf Seite 91

  LABCAR-NIC V5.4.1 (Network Integration CAN) ist ein Add-On zu LABCAR-OPERATOR V5.4.1. Es ermöglicht ein einfaches Testen von Steuergerätefunktionen, die CAN-Kommunikation beinhalten.

- „Instrumentierung für die CAN-Simulation“ auf Seite 135

  Wenn mit LABCAR-NIC V5.4.1 CAN-Restbussimulation durchgeführt wird, gibt es in ETAS EE spezielle Instrumente für CAN-Messages und einen sogenannten „CAN Bus Monitor“.

- „LIN-Module (Network Integration LIN)“ auf Seite 143

  LABCAR-NIL (Network Integration LIN) ist ein Add-On zu LABCAR-OPERATOR. Es ermöglicht das Testen von Steuergerätefunktionen, die LIN-Kommunikation nach LIN 2.0 und LIN 2.1/LIN 2.2 beinhalten.

- „NIF-Module (Network Integration FlexRay)“ auf Seite 179

  LABCAR-NIF V5.4.1 (Network Integration FlexRay) ist ein Modultyp in LABCAR-OPERATOR V5.4.1. Es ermöglicht ein einfaches Testen von Steuergerätefunktionen, die FlexRay-Kommunikation beinhalten.

- „FiL-Module“ auf Seite 206

  In diesem Kapitel wird die Erstellung von FiL-Modulen beschrieben.

- „Real-Time Plugins“ auf Seite 222

  In diesem Kapitel finden Sie Informationen zur Erstellung von Real-Time Plugins, deren Integration in ein LABCAR-OPERATOR-Projekt und das Arbeiten mit diesen in der Experimentierumgebung ETAS EE.

- „Module zur Signalkonvertierung“ auf Seite 237

  Mit LABCAR-OPERATOR V5.4.1 wird ein Standard-Signalkonvertierungsmodul mitgeliefert, mit dem die generische Open-Loop-Konfiguration realisiert werden kann.

- „Hardware konfigurieren mit dem RTIO-Editor“ auf Seite 242

  Der grundsätzliche Umgang mit dem RTIO-Editor und die Konfiguration der eingesetzten Hardware ist im LABCAR-RTC V5.4.1 - Benutzerhandbuch ausführlich beschrieben.

- „Der Connection Manager“ auf Seite 243

  Der Connection Manager ermöglicht den Closed-Loop-Betrieb, indem er die Ein- und Ausgänge der vorhandenen Module entsprechend verbindet.

- „Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)“ auf Seite 256

  Bei der Erstellung eines LABCAR-OPERATOR Projektes wird automatisch eine Default-OS-Konfiguration erstellt und zum Projekt zugeordnet.

- „Erstellen von Multi-RTPC-Netzwerken“ auf Seite 269

  In diesem Kapitel wird beschrieben, wie man Multi-RTPC-Netzwerke erstellt.

3.1 Modularer Simulationscode

Dieser Abschnitt gibt einen kurzen Überblick über die Integration von Modulen verschiedenster Herkunft zu einem LABCAR-OPERATOR-Gesamtprojekt.

Im Einzelnen finden Sie Informationen zu folgenden Themen:

3.1.1 LABCAR-Module

In LABCAR-OPERATOR besteht der Code des Simulationsmodells aus einzelnen „LABCAR-Modulen“, die jeweils eine Komponente des Gesamtprojektes beschreiben.

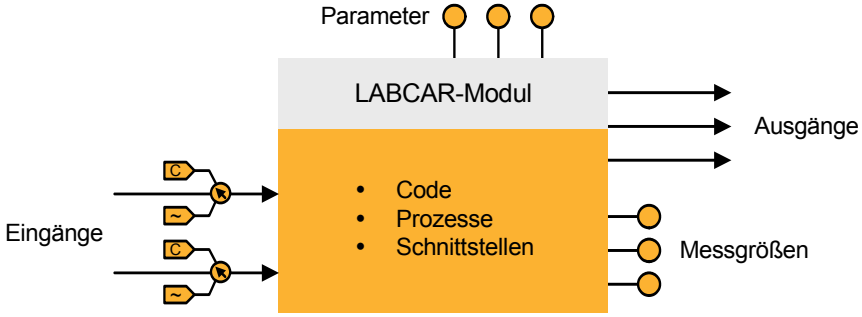


Abb. 3-1 Ein LABCAR-Modul

Ein einzelnes LABCAR-Modul besteht aus

- der Schnittstelleninformation

  und

- der Funktionalität selbst (in Form von Standard-C-Code).

Die Schnittstelleninformation beinhaltet die Definition der Ein- und Ausgänge des Moduls, Informationen darüber, auf welche Mess- und Verstellgrößen zugegriffen werden kann und welche Prozesse das Modul steuern.

Abb. 3-2 auf Seite 26 zeigt den Aufbau eines Projektes aus mehreren Modulen. Jedes dieser Module kann auf Signalebene (mit Hilfe des Connection Managers) mit anderen Modulen verbunden werden. Auf diese Weise werden Signalpfade definiert und Regelkreise geschlossen.

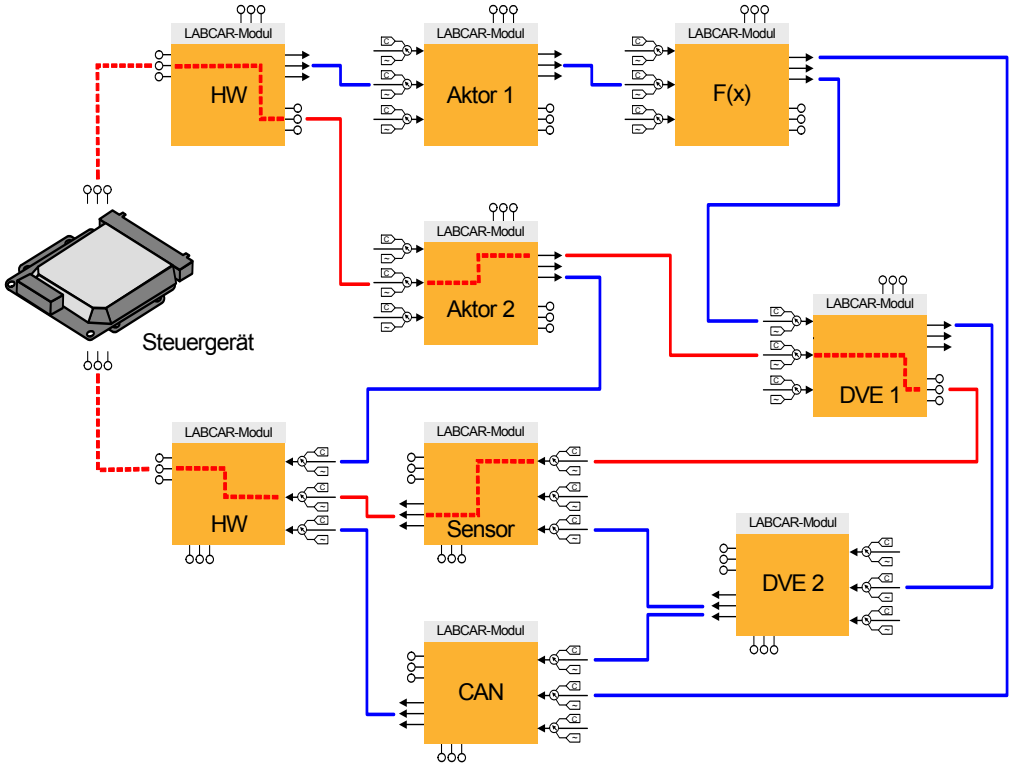


Abb. 3-2 Module eines LABCAR-OPERATOR-Projektes

Jeder der Eingänge kann zur Laufzeit aufgebrochen werden, um einen Konstantwert, eine synthetische Signalform oder einen aufgezeichneten Signalverlauf (z.B. von einer Testfahrt) einzuspeisen (siehe „Module zur Signalkonvertierung“ auf Seite 237).

Aus jedem der Module wird bei der Codegenerierung zunächst C-Code generiert, sofern diese nicht bereits in Form von C-Code vorliegen. Die einzelnen C-Code-Teile werden danach zum Gesamt-Simulationscode integriert, der später auf dem Simulationstarget (RTPC) ausgeführt wird.

Dieser Ansatz bietet maximale Flexibilität, da jedes verhaltensbeschreibende Werkzeug eingesetzt werden kann, das in der Lage ist, C-Code zu erzeugen. Auf diese Weise können die domänenspezifischen Vorteile diverser Werkzeuge gezielt zum Einsatz kommen, wie zum Beispiel die Verwendung verschiedener Modellierungssprachen, Restbussimulatoren oder C-Code-Bibliotheken.

Berechnung des Simulationscodes

Die Echtzeit-Berechnung des Simulationscodes erfolgt auf einem Real-Time PC, einem auf Intel®-Prozessorarchitektur basierenden Simulationstarget, das generische Unterstützung für Multicore-Prozessoren bietet. Dadurch ist es möglich, die einzelnen Module auf mehrere CPU-Kerne zu verteilen.

3.1.2 Welche Typen von Modulen können integriert werden

Im Folgenden finden Sie eine Liste der verschiedenen Module, die in ein LABCAR-OPERATOR-Projekt eingebunden werden können und Hinweise darauf, wo in diesem Handbuch die Integration dieser Module beschrieben wird.

Die einzelnen Module werden im Project Explorer durch die gezeigten Symbole gekennzeichnet.


- **ASCET- und MATLAB/Simulink-Module**

  Näheres zur Integration von diesen Modulen finden Sie in den Kapiteln „MATLAB/Simulink-Modelle“ auf Seite 38 und „ASCET-Module“ auf Seite 52.

- **C-Code-Module**

  Näheres zur Integration von C-Code-Modulen finden Sie im Kapitel „C-Code-Module“ auf Seite 60.



- **CAN-, LIN- und FlexRay-Module**

  Näheres zur Integration von diesen Modulen finden Sie in den Kapiteln „CAN-Module (Network Integration CAN)“ auf Seite 91, „LIN-Module (Network Integration LIN)“ auf Seite 143 und „NIF-Module (Network Integration FlexRay)“ auf Seite 179.

- **FiL-Module**

  Näheres zur Integration von FiL-Modulen finden Sie im Kapitel „FiL-Module“ auf Seite 206.

- **Module für Open-Loop-Zugriff und Signalumwandlung**

  Näheres zur Erstellung und Konfiguration dieser Modulen finden Sie im Kapitel „Module zur Signalkonvertierung“ auf Seite 237

- **I/O-Hardwaremodule**

  Diese Module sind nur insofern Teil des Simulationscodes, als dass sie die I/O-Hardware zwischen Modell und Steuergerät beschreiben. Der Umgang mit dem RTIO-Editor und die Konfiguration der eingesetzten Hardware ist im LABCAR-RTC V5.4.1 - Benutzerhandbuch ausführlich beschrieben.

Die Vorgehensweise bei der Integration hängt von der Art des Moduls ab und wird im Folgenden beschrieben.

3.1.3 LABCAR-OPERATOR-Projekt erstellen und Module integrieren

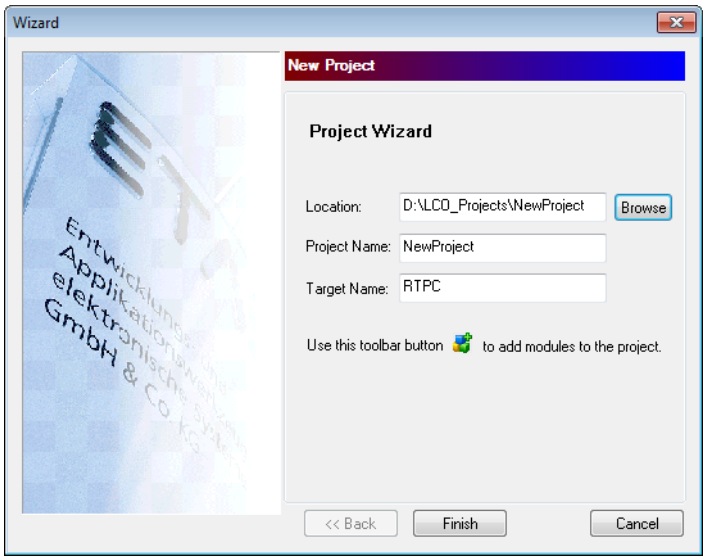
Hier erstellen Sie ein Projekt (mit oder ohne Simulink-Modell) und integrieren Sie ggf. weitere C-Code-basierte Module.

Projekt erstellen

- Wählen Sie **File → New Project.**

  Der Project Wizard wird geöffnet.
- Wählen Sie unter „Location“ ein Verzeichnis, in dem das neue Projekt angelegt werden soll.

- Geben Sie unter „Project Name“ einen Projektnamen an.

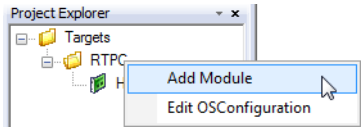


- Weisen Sie diesem Target unter „Target Name“ ggf. einen anderen Namen zu.
- Klicken Sie **Finish**.

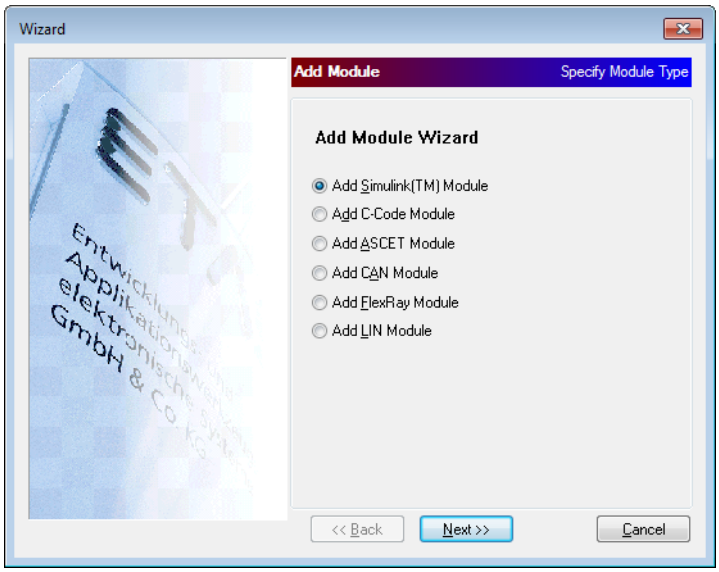
  Das LABCAR-OPERATOR Projekt wird erstellt. Sie können jetzt Module hinzufügen.

Hinzufügen von Modulen

- Wählen Sie im Projekt Explorer den Ordner mit dem Namen des Experimentaltargets.
- Rechtsklicken Sie und wählen Sie **Add Module**.



- Im „Add Module Wizard“, wählen Sie das gewünschte Modul.



Zur Integration weiterer (nicht im Module Wizard wählbarer) Module sind unterschiedliche Verfahrensweisen erforderlich:

- Hardware-Modul

  Jedes LABCAR-OPERATOR-Projekt besitzt ein Hardware-Modul, das mit dem Projekt erstellt wird und das zu diesem Zeitpunkt außer dem Simulationstarget „RTPC“ selbst keine weitere Hardware enthält. Näheres zur Konfiguration des Moduls (= Integration von I/0-Hardware) finden Sie im LABCAR-RTC V5.4.1 - Benutzerhandbuch.

- CAN-Module

  CAN-Module werden erstellt und verwaltet im Register „CAN Editor“ — ausführliche Informationen dazu finden Sie im Kapitel „CAN-Module (Network Integration CAN)“ auf Seite 91.

- FiL-Module

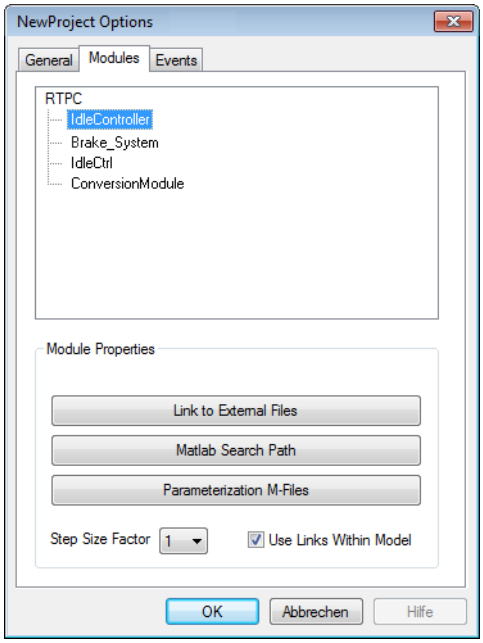
  FiL-Module werden im Register „FiL Wizard“ integriert und bearbeitet– ausführliche Informationen dazu finden Sie im Kapitel „FiL-Module“ auf Seite 206.

3.1.4 Externen Code integrieren

Über die Integration einzelner Module hinaus können Sie bei einigen Modultypen (Simulink, CAN- und Signalkonvertierungsmodule) noch dafür sorgen, dass zusätzliche Header-, Code-, Objekt- und Bibliotheksdateien bei der Codegenerierung des Moduls mitkompiliert und gelinkt werden. Damit können häufig verwendete Codedateien oder Bibliotheken zwischen Projekten ausgetauscht und beliebig wiederverwendet werden.

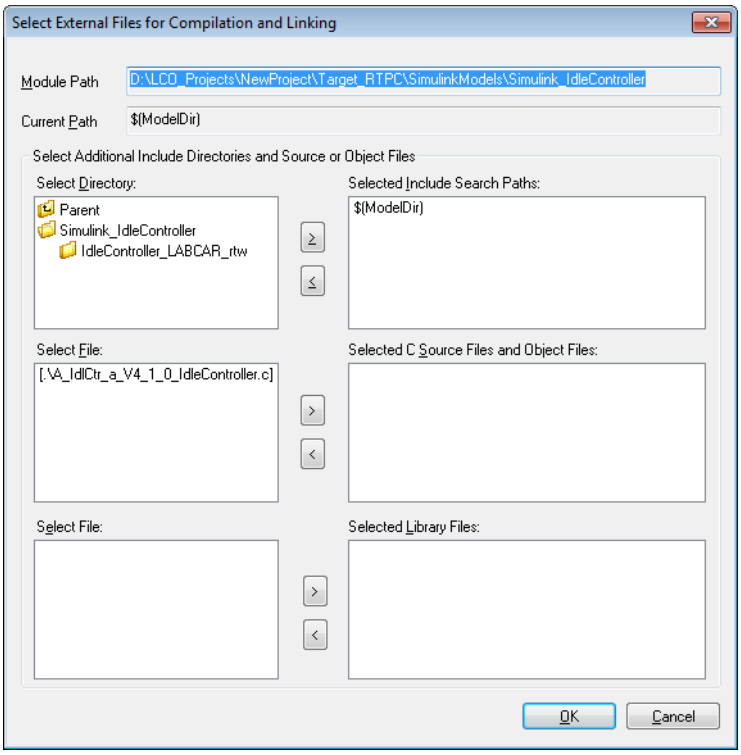
Externe Datenquellen auswählen

- Wählen Sie im Hauptmenü **Project** → **Options**.
- Das Fenster „*<Projektname>* Options“ wird geöffnet.
- Wählen Sie das Register „Modules“.
- Wählen Sie das Modul, für das weitere Dateien spezifiziert werden sollen.



- Klicken Sie **Link to External Files**.

  Das Fenster „Select External Files for Compilation and Linking“ wird geöffnet.



Hier können weitere Verzeichnisse und Dateien angegeben werden, die beim Kompilieren und Linken berücksichtigt werden sollen.

Im Einzelnen sind dies:

- Verzeichnisse, in denen Header-Dateien (*.h) gesucht werden
- Dateien mit Quell- und Objektcode (*.c, *.o)
- Bibliotheksdateien (*.a)

Um die Portierbarkeit des Projekts und die Stabilität der Codegenerierung auf dem Real-Time PC zu garantieren, können hier nur Verzeichnisse und Dateien referenziert werden, die sich innerhalb des Modulpfades (z.B $(ModelDir)) befinden.

Das Verzeichnis „user“

Im Projektverzeichnis <*Projektname*>`\Target_RTPC\` wird ein weiterer Ordner `\user` erstellt, in dem zusätzliche Dateien abgelegt werden können, die auf dem Real-Time PC während des Kompilier-/Link-Vorganges benötigt werden.

Dazu wird der Inhalt des Ordners `\user` während dem Buildprozess auf den Real-Time PC übertragen und zwar in das Verzeichnis

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

wobei „labcar“ der Default-User auf dem ETAS RTPC-Betriebssystem ist.

Der Ordner enthält zwei weitere Unterordner:

- `include`

  Hier können Header-Dateien abgelegt werden, die die Deklarationen für die Routinen der zu verwendenden Libraries enthalten.

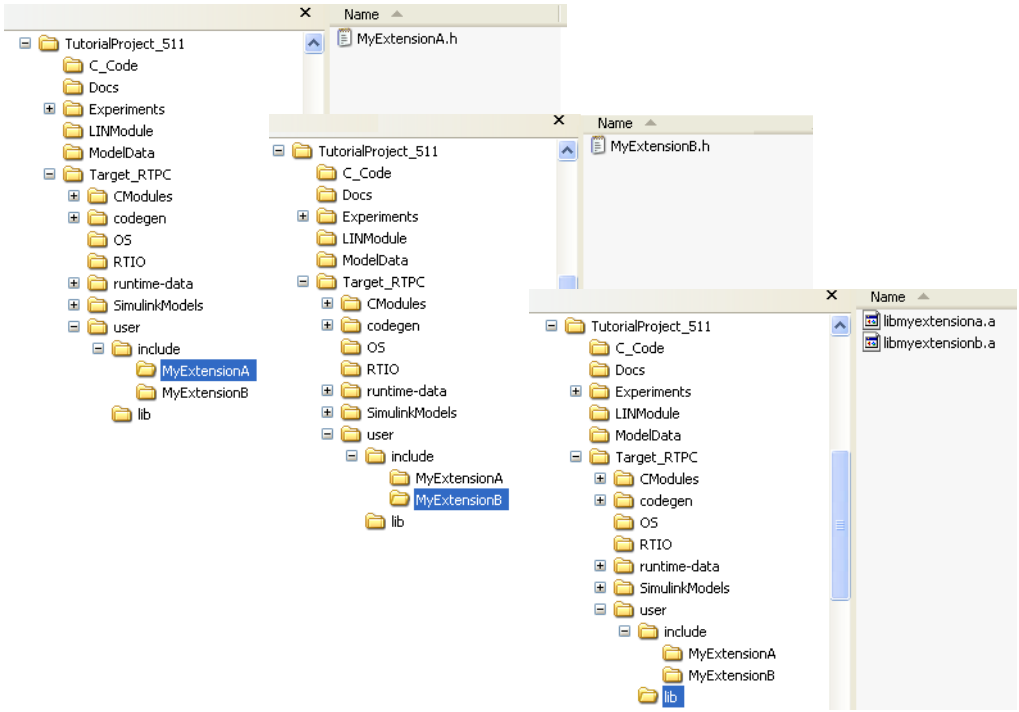
  Unterordner in diesem Ordner werden erkannt.

- `lib`

  Hier können Libraries abgelegt werden, deren Code während des Linkens des Projektes verwendet werden soll/kann.

  Unterordner in diesem Ordner werden **nicht** erkannt.

Beispiel:



Um die Bibliotheksfunktionen für ein LABCAR-OPERATOR-Modul (z.B. ein C-Code-Modul) sichtbar zu machen, müssen die jeweiligen Headerdateien im Quellcode der entsprechenden Module referenziert werden.

Beispiel:

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

Im Gegensatz zu der herkömmlichen Methode, zu einem speziellen Modul externen Code hinzuzufügen (**Project → Options**, Register „Modules“: **Link to External Files**), kann auf die im Verzeichnis `user` abgelegten Bibliotheken und Headerdateien von allen Modulen zugegriffen werden.

3.1.5 Connection Management

In Abb. 3-2 auf Seite 26 sind zusätzlich zu den Modulen des Projekts auch Verbindungen zwischen deren Ein- und Ausgängen gezeigt. Diese Verbindungen werden im Connection Manager erstellt und ermöglichen den Closed-Loop-Betrieb und Signalverfolgung.

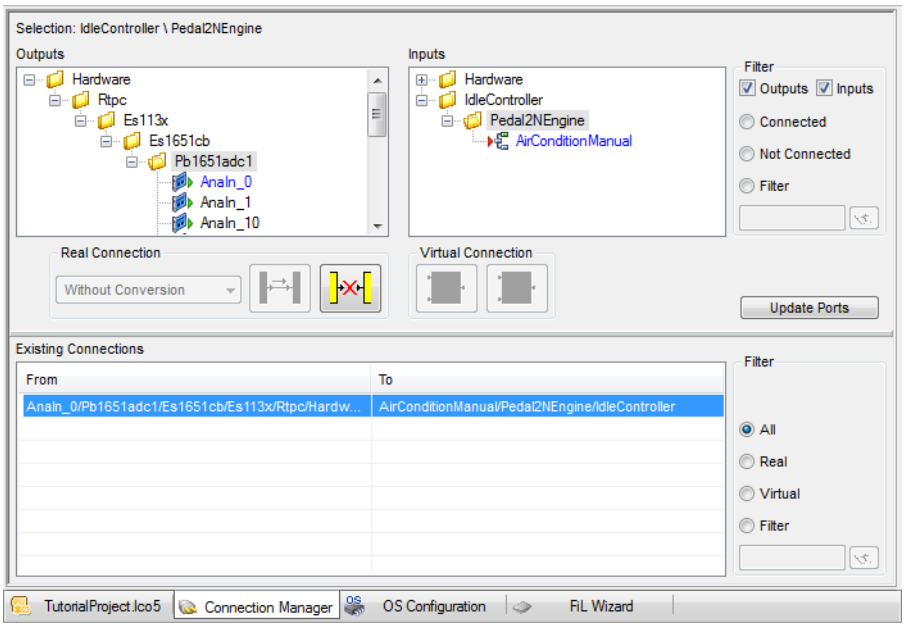


Abb. 3-3 Der Connection Manager

Ausführliche Information zum Connection Management finden Sie im Abschnitt „Der Connection Manager“ auf Seite 243.

Zudem kann bei der Erstellung einer Verbindung entschieden werden, ob am Eingang des einen Moduls ein Modul zur Signalkonvertierung eingefügt werden soll. Mit einem solchen Modul haben Sie die Wahl zwischen verschiedenen Arten von Signalkonvertierung – darüber hinaus können Sie die Verbindung zum anderen Modul auch aufbrechen und stattdessen manuell spezifizierte oder von einem Signalgenerator stammende Signale einspeisen (siehe auch „Signalkonvertierung“ auf Seite 253)

Die Konfiguration dieser Module erfolgt in speziellen GUIs in der Experimentierumgebung (siehe „Module zur Signalkonvertierung“ auf Seite 237).

3.1.6 Konfiguration des Betriebssystems („OS Settings“)

Wenn ein Modul in das Projekt eingebunden wird, so werden die jeweiligen Prozesse automatisch in die OS-Konfiguration übernommen und geeigneten Tasks zugewiesen. Sind alle Module des Projekt integriert, so muss eventuell noch die Konfiguration des Echtzeitbetriebssystems angepasst werden.

Dies betrifft das Hinzufügen und Entfernen von Tasks, die Task-Eigenschaften selbst und das Zuweisen und Entfernen von Prozessen zu/von Tasks.

Eine ausführliche Beschreibung dieser Aufgaben finden Sie in Abschnitt „Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)“ auf Seite 256.

3.1.7 Codegenerierung

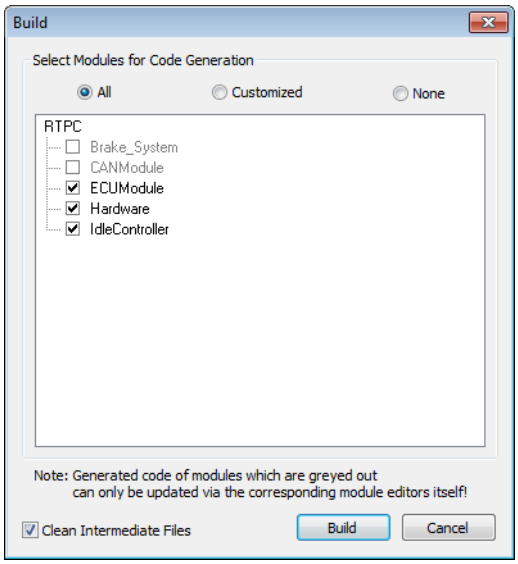
Wenn alle Module in das Projekt integriert sind, die Signale verbunden sind und die Berechnung der Tasks konfiguriert sind, kann der Simulationscode für das Experimentaltarget generiert werden.

Code generieren

- Wählen Sie **Project → Build...**

oder

- Klicken Sie das Symbol **Build LABCAR Project**.
- Wählen Sie im folgenden Fenster die Module aus, für die Code erzeugt werden soll.



Wenn Module in dieser Liste ausgegraut (d.h. nicht wählbar) sind, kann das zwei Gründe haben:

– Es handelt sich um C-Code-Module, für die automatisch Code generiert wird.

– Es handelt sich um Simulink- bzw. ASCET-Module, wobei das entsprechende Add-On LABCAR-MCS (Modeling Connector for Simulink) bzw. LABCAR-MCA (Modeling Connector for ASCET) nicht installiert worden ist.

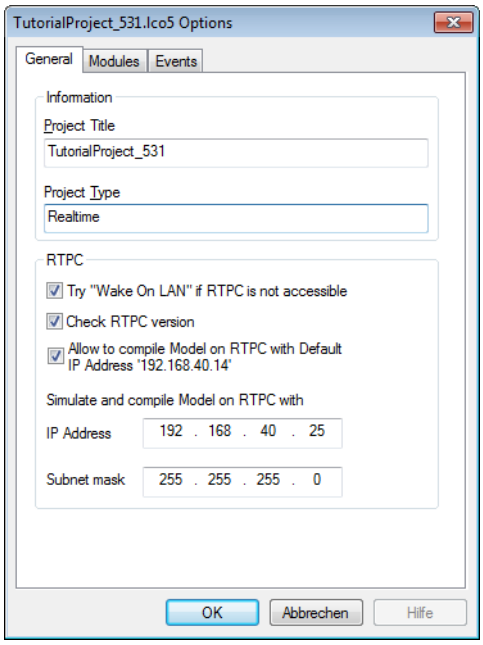
Wenn Sie die Option „Clean Intermediate Files“ aktiveren, wird der kompilierte Objektcode nach dem Build-Vorgang gelöscht. Dies ist dann notwendig, wenn vorkompilierter Code nicht wieder verwendet werden soll.

- Klicken Sie **Build**.

  Der Codegenerierungsprozess wird gestartet und der Fortgang der Ereignisse wird im Fenster „Messages“ ausgegeben.

3.1.8 Allgemeine Projektoptionen

Einige wichtige allgemeine Einstellungen für das Projekt finden Sie unter **Project → Options** in der Registerkarte „General“.



Information:

- **Project Title**

  Der Name des Projekts, definiert bei dessen Erstellung.

- **Project Type**

  Immer „Realtime“

RTPC:

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

  Damit wird versucht, den Real-Time PC über das Netzwerk zu starten („Wake-on-LAN“).

  Dafür benötigen Sie auf dem jeweiligen Netzwerk-PC ein entsprechendes Tool, das die sogenannten „Magic Packets“ sendet (z.B. das freie Tool „WOL - Magic Packet Sender 2007“). Die notwendigen Daten für die Konfiguration dieses Tool finden Sie im Web-Interface von ETAS RTPC unter „Wake On LAN Settings“ (**Main Page >> Power Control**).

- **Check RTPC version**

  Grundsätzlich unterstützt eine bestimmte Version von LABCAR-OPERATOR nur bestimmte Versionen von ETAS RTPC (siehe Release Notes). Um sicherzustellen, dass das Simulationstarget die richtige Version hat, kann diese Option aktiviert werden.

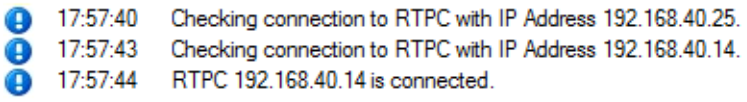
  Dann wird beim Vorliegen einer nicht unterstützten Version der Build-Vorgang mit einem Fehler abgebrochen.

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

  Wählen Sie diese Option, wenn Sie ein Projekt – unabhängig von der IP-Adresse des Real-Time PCs für dieses Projekt – immer auf einem Rechner mit der mit der Standard-IP-Adresse bauen wollen.

  Beispiel: Das Projekt ist für einen Real-Time PC mit der IP-Adresse 192.168.40.25 konfiguriert (siehe „Simulate and compile model on RTPC with“ auf Seite 37), der aber beim Build auf dem Rechner mit der Standard-Adresse nicht verfügbar ist.

  Dann wird von der nicht gefundenen IP-Adresse zur Standard-Adresse gewechselt:

  Der grundsätzliche Ablauf ist wie folgt:

  – Prüfe die konfigurierte IP-Adresse

    Wird der Real-Time PC unter der konfigurierten Adresse gefunden, wird er für den Build verwendet.

    Falls nicht:

  – Prüfe Fallback-IP-Adresse 192.168.40.14.

    Wird der Real-Time PC unter der Fallback-Adresse gefunden, wird er für den Build verwendet.

    Falls nicht:

  – Wenn „Wake On LAN“ aktiviert ist, wird versucht, den Real-Time PC unter der konfigurierten Adresse aufzuwecken.

    Kann der Real-Time PC unter der konfigurierten Adresse aufgeweckt werden, wird er für den Build verwendet.

Falls nicht:

– Versuch, den Real-Time PC unter der Fallback-IP-Adresse aufzuwecken.

  Kann der Real-Time PC unter der Fallback-IP-Adresse aufgeweckt werden, wird er für den Build verwendet.

  Falls nicht:

– Build nicht möglich

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

  Hier können Sie eine IP-Adresse (bevorzugter Bereich: 192.168.40.10 ... 192.168.40.40) und die Subnetz-Maske des Real-Time PCangeben, auf dem das Projekt kompiliert und ausgeführt werden soll.

3.2 MATLAB/Simulink-Modelle

In diesem Abschnitt erhalten Sie Informationen zur Integration von MATLAB/Simulink-Modellen:

Softwarevoraussetzungen

Wenn Ihr LABCAR-Projekt MATLAB/Simulink-Modelle verwenden soll, brauchen Sie

- eine Installation von MATLAB.

  Eine Liste der unterstützten Releases finden Sie in den Release Notes (über das Menü **? → Help**).
- eine Lizenz für LABCAR-MCS V5.4.1 (Modeling Connector for Simulink)

3.2.1 Simulink-Modelle

Der Real-Time Workshop erzeugt aus Simulink-Modellen C-Code - dabei gibt es einige Punkte, die in diesem Abschnitt beschrieben werden und die unbedingt beachtet werden sollten.

Hinweis

*Beachten Sie, dass bei der Codegenerierung nur **eine** Version des Real-Time Workshop verwendet wird! Dies kann zu Problemen führen, wenn Sie in Ihr Projekt mehrere Modelle integrieren wollen, die mit **verschiedenen** Versionen von MATLAB/Simulink erstellt wurden!*

Benutzerdefinierter C-Code: Pfadnamen

Der Real-Time Workshop ermöglicht die Integration von vom Anwender definiertem Code. Dies kann eine S-Function oder anderer, von einer S-Function aufgerufener Code sein.

Wird das Projekt für ein bereits vorhandenes Modell erstellt, so wird das Original und dessen vollständiges Verzeichnis (mit allen Unterverzeichnissen) entweder in das Projektverzeichnis kopiert oder einfach referenziert, was eine Versionskontrolle des Modells ermöglicht.

Zum Kompilieren und Linken werden die entsprechenden Dateien an den Ort kopiert, an dem der Build-Vorgang stattfindet. Daraus ergeben sich einige Einschränkungen in Bezug auf benutzerdefiniertem Code, der zu einem Simulink-Modell gehört:

- In C kann der #include-Befehl eine Datei mit beliebiger Erweiterung referenzieren, wobei als gängiger Standard *.h verwendet wird.
  LABCAR-MCS V5.4.1 unterstützt nur die Dateierweiterungen *.h und *.c - bei Verwendung anderer Erweiterungen kann es vorkommen, dass der Compiler die entsprechende Datei nicht findet.
- In C kann der #include-Befehl eine Datei über einen Pfad referenzieren, wobei als gängiger Standard allein ein Dateiname verwendet wird und die Pfadinformation über entsprechende Compilerschalter hinzu kommt.

  Zum Beispiel:

  ```
  #include Dateiname.h
  #include Unterverzeichnis\Dateiname.h
  #include ..\Parallelverzeichnis\Dateiname.h
  ```

  LABCAR-MCS V5.4.1 unterstützt nur die ersten beiden Variante - andere Formen der Referenzierung können dazu führen, dass der Compiler die entsprechende Datei nicht findet.
- Der Inhalt der verschiedenen Include-Suchpfade wird zum Kompilieren in ein bestimmtes Verzeichnis kopiert, und zwar in das gleiche Verzeichnis, auf das auch die Dateien kopiert werden, die beim Linken externer Dateien (siehe „Weitere Dateien hinzulinken („Link to External Files“)“ auf Seite 46) spezifiziert werden.

  Achten Sie darauf, dass es in diesen Verzeichnissen nicht mehrere Dateien gleichen Namens und unterschiedlichen Inhalts gibt. Ansonsten kann es je nach der Reihenfolge, in der das Kopieren stattfindet, zu unerwünschtem Überschreiben kommen.

Function-Call Subsystems

Es kann vorkommen, dass das vom Anwender verwendete Modell Function-Call Subsystems enthält oder dass der Anwender Modellteile verwendet, die nicht im kleinen Rechenraster ausgeführt werden sollen und er deshalb Function-Call Subsystems anlegt. In diesen Fällen beachten Sie bitte folgende Hinweise.

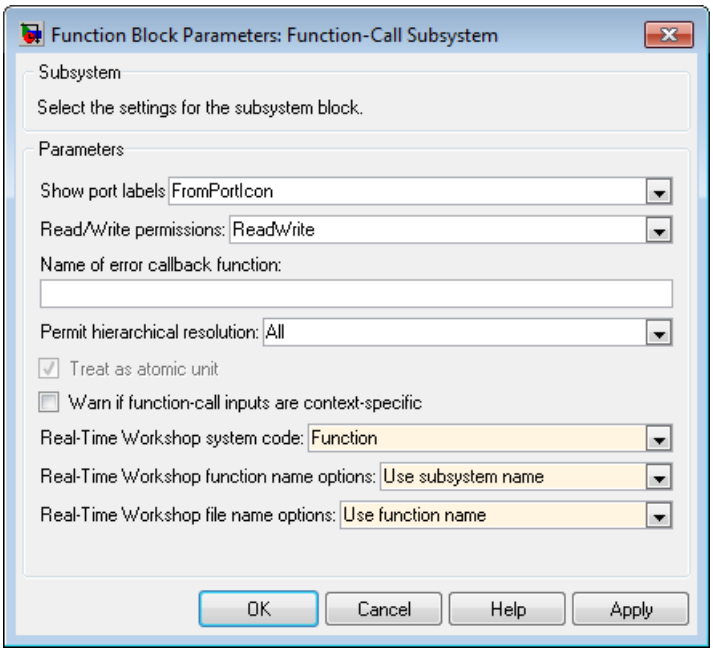
Bei geeigneter Parametrierung dieser Blöcke erzeugt der Real-Time Workshop für den Modellteil, der darin enthalten ist, einen eigenen Prozess. Dieser Modellteil wird auf dem Target nicht mit dem Simulationsrechenraster gerechnet, präziser: er wird garnicht gerechnet, solange er vom Anwender nicht in eine Task (die zeit- oder eventgetriggert sein kann) eingehängt wird.

Dabei hängt es von den Parametern eines solchen Function-Call Subsystems ab, ob ein separater Prozess erzeugt wird oder nicht. Soll ein solcher Prozess erzeugt werden, gehen Sie wie folgt vor:

Einstellungen wählen

- Klicken Sie den „Function-Call Subsystem“-Block im Simulink-Modell mit der rechten Maustaste.
- Wählen Sie **Subsystem parameters**.

  Das Fenster „Block Parameters: Function-Call Subsystems“ wird geöffnet.
- Wählen Sie die in der folgenden Abbildung (R2007b) gezeigten Einstellungen:



Der mit diesen Einstellungen erzeugte Prozess erhält den Namen `<Modellname>_<Subsystemblockname>`.

Hinweis

*Dieser Prozess wird erst dann in der OS-Konfiguration (siehe „Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)“ auf Seite 256) angezeigt, nachdem das Projekt in LABCAR-IP neu erstellt worden ist (**Project → Build**).*

3.2.2 Einschränkungen bezüglich der verwendbaren Simulink-Modelle

Der Real-Time Workshop besitzt bezüglich der verwenbaren Simulink-Modelle einige Einschränkungen, die im Folgenden beschrieben werden.

S-Functions

Der Real-Time Workshop kann nur S-Functions verarbeiten, die Parameter vom Datentyp „double“ enthalten. Wird ein anderer Datentyp gefunden, wird die Codegenerierung abgebrochen.

Desweiteren kann der Real-Time Workshop keine S-Functions verarbeiten, die Outputs mit selbstdefiniertem Datentyp enthalten (vereinbart mit „ssRegisterDataType“). In diesem Fall wird zwar erfolgreich C-Code erzeugt, beim Kompilieren des Codes wird aber eine Fehlermeldung ausgegeben, da ein Makro den Aufruf von „ssRegisterDataType“ durch „ssRegisterDataType_cannot_be_used_in_RTW“ ersetzt.

Blöcke mit absoluter Zeit

Der Real-Time Workshop kann keine Blöcke verarbeiten, die die absolute Zeit verwenden (z.B. „Sine Wave“, „Clock“, „Pulse Generator“, „Signal Generator“, „Transport Delay“).

Zwar wird für ein Modell, das solche Blöcke enthält, erfolgreich C-Code erzeugt, der auch kompiliert wird - bei der Ausführung liefern diese Blöcke aber falsche Ergebnisse.

Blöcke für die keine Parameter und Outputs erzeugt werden

Für eine Reihe von Modellblöcken werden beim Import keine Parameter und keine Outputs erzeugt - dies trifft zu auf die Blöcke:

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

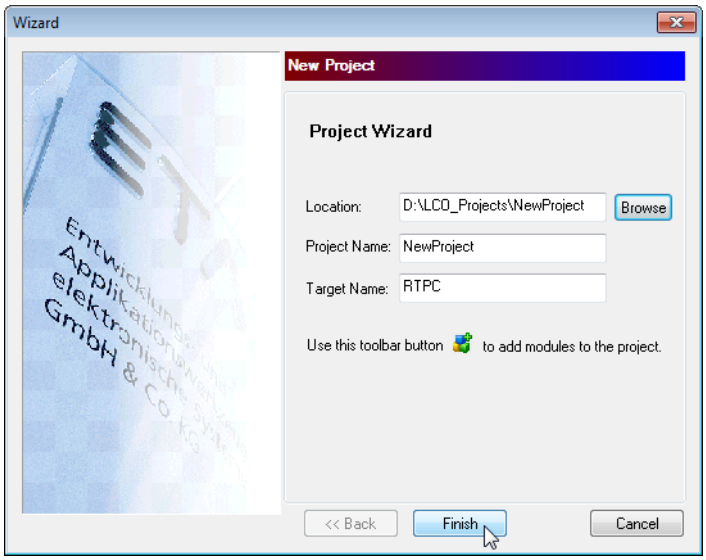
3.2.3 Neue Projekte erstellen

In diesem Abschnitt erhalten Sie Informationen zum Erstellen von LABCAR-OPERATOR-Projekten.

Ein LABCAR-OPERATOR-Projekt erstellen

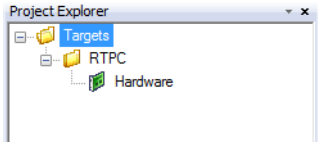
- Wählen Sie **File → New Project.**

  Der Project Wizard wird geöffnet.
- Wählen Sie unter „Location“ ein Verzeichnis, in dem das neue Projekt angelegt werden soll.
- Geben Sie unter „Project Name“ einen Projektnamen an.



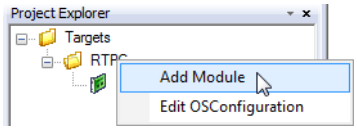
- Weisen Sie dem Target unter „Target Name“ ggf. einen anderen Namen zu.
- Klicken Sie **Finish**.

  Das Projekt wird mit einem Default-Hardwaremodul erstellt.

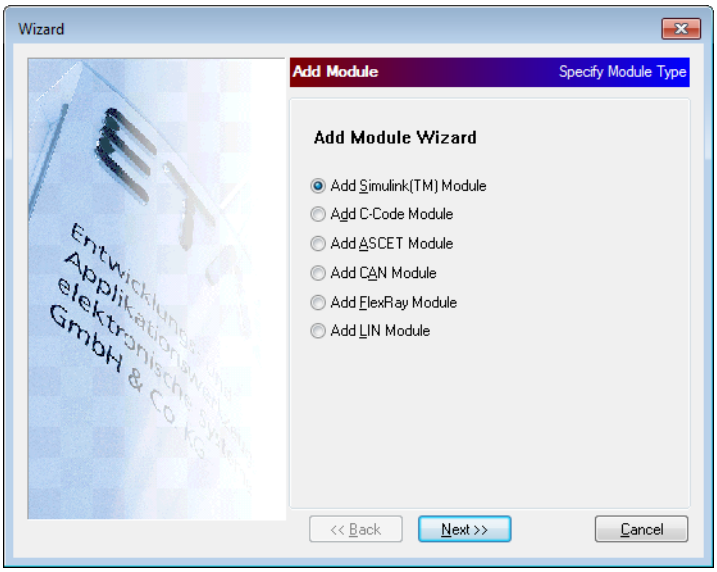


Ein Simulink-Modell hinzufügen

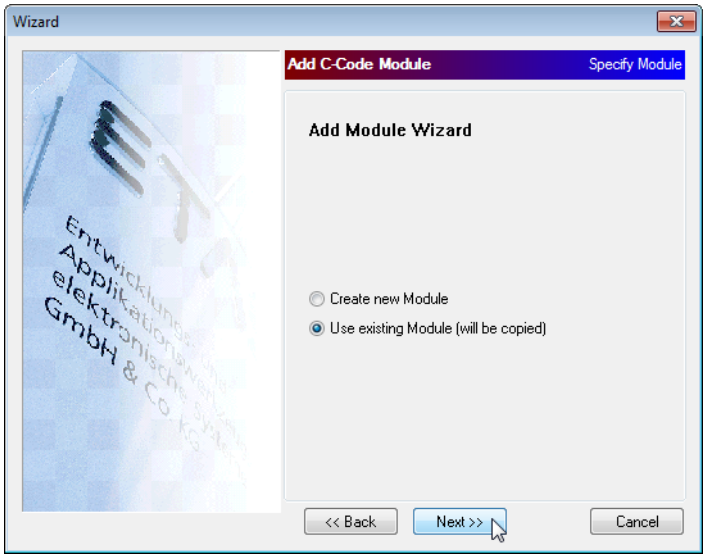
- Wählen Sie im Projekt Explorer den Ordner mit dem Namen des Experimentaltargets.
- Rechtsklicken Sie und wählen Sie **Add Module**.



- Im „Add Module Wizard“, wählen Sie „Add Simulink Module“.

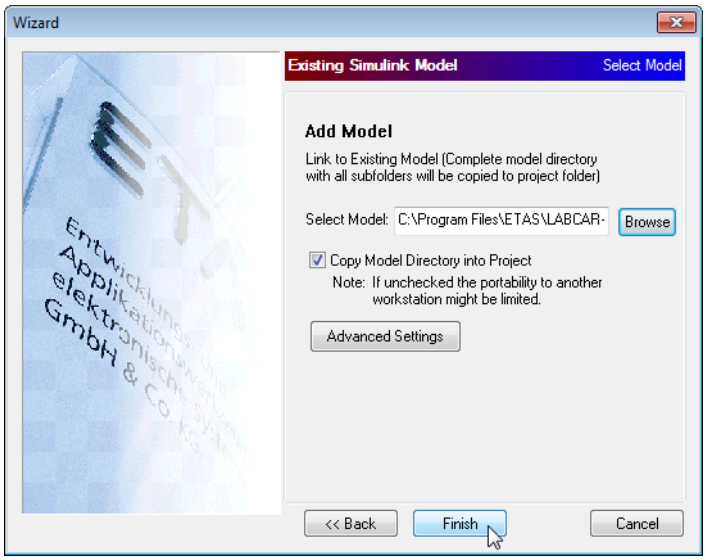


- Klicken Sie **Next**.
- Wählen Sie „Use existing Simulink model“.



- Klicken Sie **Next**.

- Wählen Sie mit **Browse** das Simulink-Modell, das Sie verwenden möchten.



- Ist die Option „Copy Model Directory into Project“ gewählt, wird das Simulink-Modell und die dazu gehörenden Dateien in das Projektverzeichnis kopiert – ansonsten wird das angegebene Modell lediglich aus dem Projekt heraus referenziert.

Hinweis

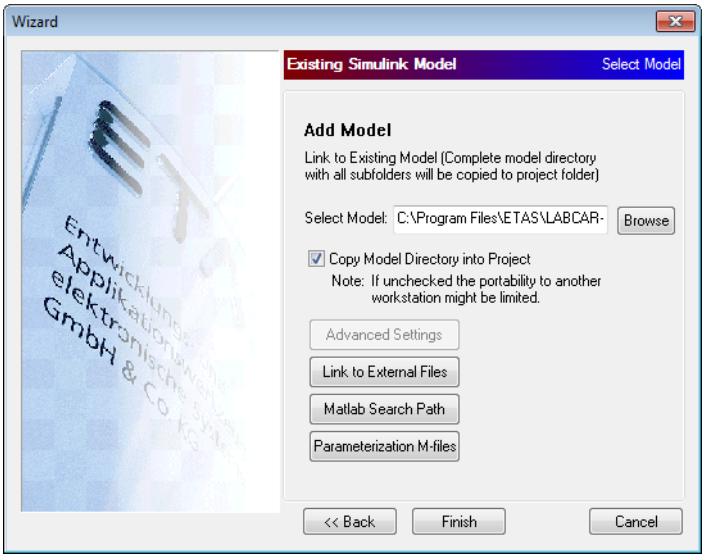
Wenn das Modell nur referenziert wird, muss diese Referenz bei der Migration des LABCAR-OPERATOR-Projekts angepasst werden. Der (absolute) Pfad zum Modellverzeichnis steht in der Textdatei `<Modellname>.conf` *(im Verzeichnis <Projektname>\Target_RTPC\SimulinkModels\Simulink_<Modelname>) und kann als relative oder absolute Pfadangabe (entsprechend der neuen Position des LABCAR-OPERATOR-Projektes) angepasst werden.*

- Klicken Sie **Finish**.

  Das Modell wird zum Projekt hinzugefügt.

3.2.4 Weitere Einstellungen für das Simulink-Modell

Wenn Sie bei der Erstellung des Projektes das zu verwendende Simulink-Modell angeben, finden Sie im selben Fenster die Schaltfläche **Advanced Settings**, über die Sie weitere Einstellungen für das Simulink-Modell erreichen können.

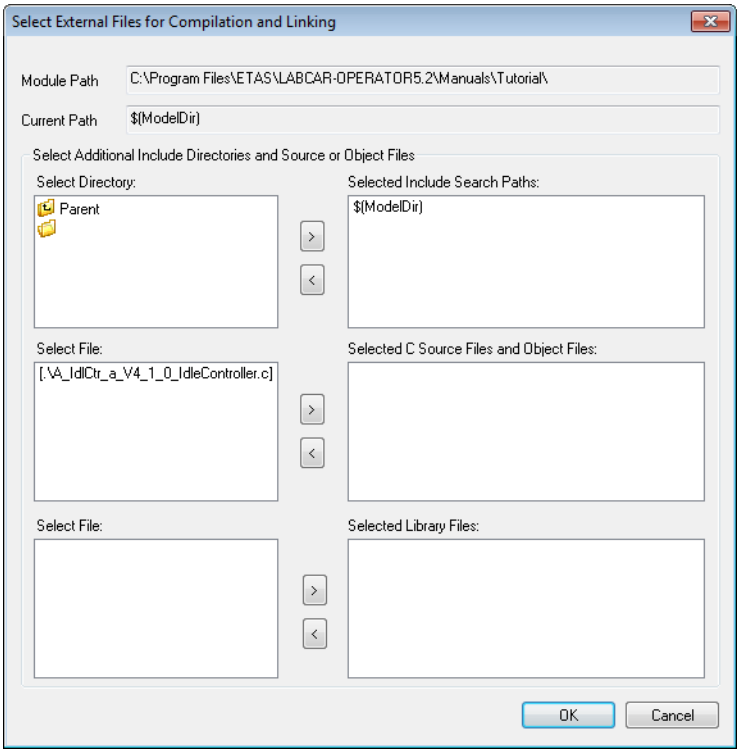


Wenn zur Kompilierung des Modells weitere Include-Verzeichnisse und Quell- oder Objektdateien benötigt werden, können Sie diese hier spezifizieren. Zusätzlich können Objekt-Bibliotheken gewählt werden, die beim Build-Prozess hinzugelinkt werden.

Weitere Dateien hinzulinken („Link to External Files“)

- Klicken Sie **Link to External Files**.

  Das folgende Fenster wird geöffnet.



Hinweis

Bibliotheksdateien müssen die Erweiterung „.a“ oder „.lib“ haben, damit Sie in der Liste angezeigt werden.

- Wählen Sie die entsprechenden Verzeichnisse und Dateien mit der Maus und fügen Sie diese Dateien mit den **>** Schaltflächen zu den Listen auf der rechten Seite des Fensters hinzu.

Hinweis

Es können nur Dateien und Verzeichnisse ausgewählt werden, die sich in Unterverzeichnissen des Verzeichnisses befinden, in dem sich das Modell befindet.

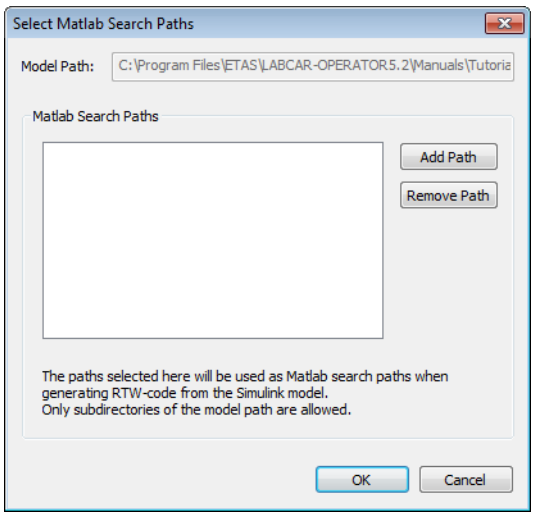
- Klicken Sie **OK**.

Um zum Projekt MATLAB-Suchpfade hinzuzufügen, gehen Sie wie folgt vor:

MATLAB Suchpfade hinzufügen („Matlab Search Paths“)

- Klicken Sie **Matlab Search Paths**.

  Das folgende Fenster wird geöffnet.



- Klicken Sie **Add Path**.

  Es wird ein Verzeichnisbrowser geöffnet, aus dem Sie ein Verzeichnis auswählen können.

Hinweis

Es können nur Verzeichnisse unterhalb des Modell-pfades gewählt werden (Feld „Model Path“)

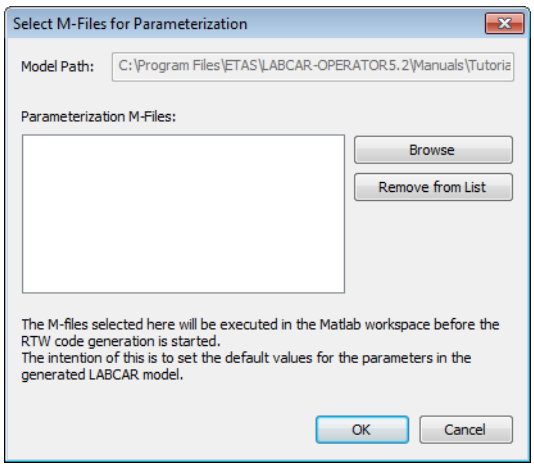
  Die ausgewählten Pfade werden in der Liste angezeigt.

- Wenn Sie einen Suchpfad aus der Liste entfernen wollen, wählen Sie diesen in der Liste und klicken Sie **Remove Path**.
- Klicken Sie **OK**.

Um Parametrierungsdateien auszuwählen, die beim Öffnen des Simulink-Modells ausgewertet werden, gehen Sie wie folgt vor:

Parametrierungsdateien auswählen („Parameterization M-files“)

- Klicken Sie **Parameterization M-files**.
- Das folgende Fenster wird geöffnet.



- Klicken Sie **Browse**.

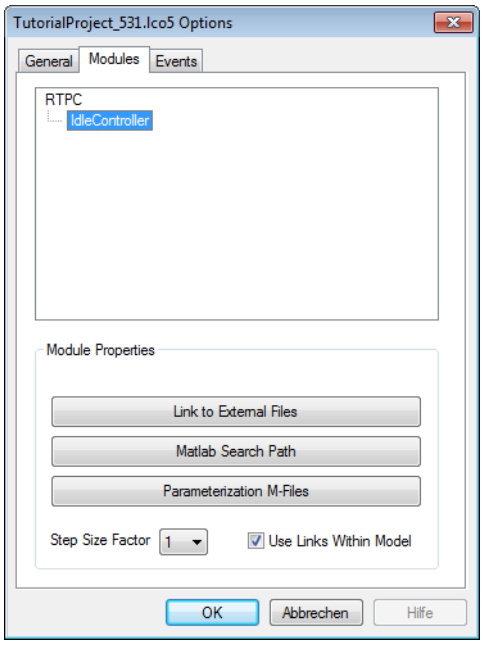
  Es wird ein Dateiauswahlfenster geöffnet, in dem Sie eine Datei auswählen können.

  Die ausgewählte Datei wird in der Liste angezeigt.
- Wenn Sie eine Datei aus der Liste entfernen wollen, wählen Sie diese und klicken Sie **Remove from List**.
- Klicken Sie **OK**.

Damit die oben beschriebenen Einstellungen nicht nur während der Projekterstellung, sondern auch im weiteren Verlauf der Arbeit mit dem Projekt verändert werden können, sind die beschriebenen Dialogfenster immer zugänglich:

Einstellungen ändern

- Wählen Sie **Project** → **Options**.
  Das Fenster „*Projektname* Options“ wird geöffnet.
- Wählen Sie das Register „Modules“.



Hier können Sie alle weiter oben beschriebenen Einstellungen für das jeweilige Modell (im Fenster links) bearbeiten.

Integrationsschrittweite (Simulink-Modell) und Task Period (ETAS EE)

Da es in vielen Fällen sinnvoll ist, die Berechnung des Modells (definiert durch die fixed-step-size des Simulink-Modells) wesentlich höherfrequenter auszuführen als die Verarbeitung/Darstellung in der Experimentierumgebung (definiert durch die Task Period in der OS Configuration), können für die Task Period dT_{EE} größere Werte (Faktor n = 1..10) gegenüber der Integrationsschrittweite fss_{SL} konfiguriert werden:

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

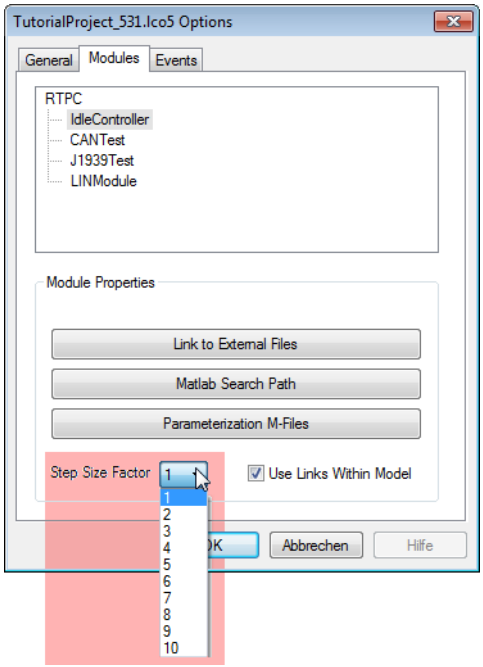
Hinweis

Dieser Faktor n kann für jedes Simulink-Modell des LABCAR-OPERATOR-Projektes anders sein!

Gehen Sie wie folgt vor:

- Wählen Sie **Project** → **Options**.
  Das Fenster „*Projektname* Options“ wird geöffnet.
- Wählen Sie das Register „Modules“.
- Wählen Sie das Simulink-Modell, für das Sie diese Einstellung vornehmen wollen.

- Stellen Sie den gewünschten Faktor ein.



- Bestätigen Sie mit **OK**.
- Speichern Sie Änderung mit **File** → **Save**.

  Damit wird die Rechenschrittweite fss des Simulink-modells als ein Bruchteil (1/n) der Task Period gesetzt, die in der OS Configurationdefiniert wird.

Aktivieren/Deaktivieren der Unterstützung von Matlab Links

Seit Version 5.3.1 werden defaultmässig Matlab Links für Outports und Inports unterstützt. Um Code wie in früheren Versionen zu generieren, können Sie diese Unterstützung deaktivieren.

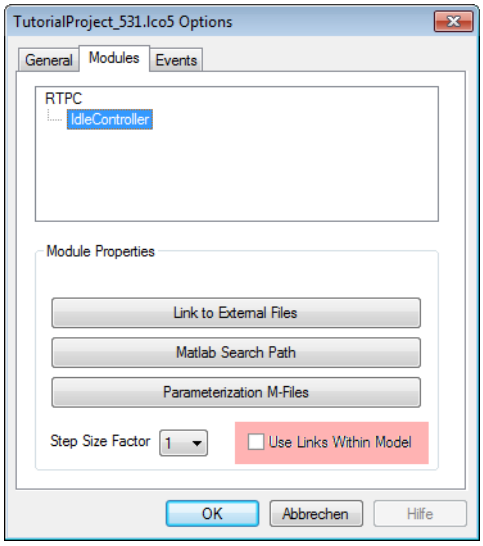
- Wählen Sie **Project** → **Options**.

  Das Fenster „*Projektname* Options“ wird geöffnet.
- Wählen Sie das Register „Modules“.
- Wählen Sie das Simulink-Modell, für das Sie diese Einstellung vornehmen wollen.

Hinweis

Diese Option kann für jedes Simulink-Modell des LABCAR-OPERATOR-Projektes anders gewählt werden!

- Deaktivieren sie die Option **Use Links Within Model**.



- Bestätigen Sie mit **OK**.
- Speichern Sie Änderung mit **File** → **Save**.

3.3 ASCET-Module

In diesem Abschnitt finden Sie Informationen zur Integration von ASCET-Modulen in ein LABCAR-OPERATOR-Projekt.

3.3.1 Allgemeines zum Thema ASCET-Module

Das ASCET-Projekt stellt nach Codegenerierung für das Target „RTPC“ und Transfer neben einer Reihe von Code- und Headerdateien insbesondere folgende Dateien zur Verfügung:

- `<Projektname>.six`
- `<Projektname>.oil`

Die Datei `<Projektname>.six` ist eine sog. SCOOP-IX-Datei[1], die eine vollständige Schnittstellenbeschreibung des ASCET-Moduls enthält. Mit den Informationen in dieser Datei kann das ASCET-Projekt als Modul in das LABCAR-OPERATOR-Projekt integriert werden.

Die Datei `<Projektname>.oil` enthält die Konfiguration des Betriebssystems (Tasks, Prozesse etc), die bei der Integration des ASCET-Projekts wahlweise mitgenommen werden kann.

3.3.2 Vorbereitungen

ASCET installieren

Damit Sie ASCET-Projekte als Module in LABCAR-OPERATOR integrieren können, benötigen Sie eine aktuellen ASCET-Installation (Version 6.0.1 oder höher) inklusive der Komponenten ASCET-SE, ASCET-MD und ASCET-RP.

ASCET vorbereiten

Jede ASCET-Version muss durch eine spezielle Zusatzinstallation in die Lage versetzt werden, Projekte für das Target „RTPC“ exportieren zu können.

Hinweis

Bei späteren Versionen von ASCET (V6.1.0 und höher) muss dieses Feature nicht mehr von Hand nachinstalliert werden.

Die benötigte Datei finden Sie im Startbildschirm der Installations-CD unter **Installation**.

Klicken Sie den Eintrag **Install RTPC export for ASCET 6.x.y (LABCAR-ASC)** und folgen Sie den Anweisungen des Installationsprogrammes.

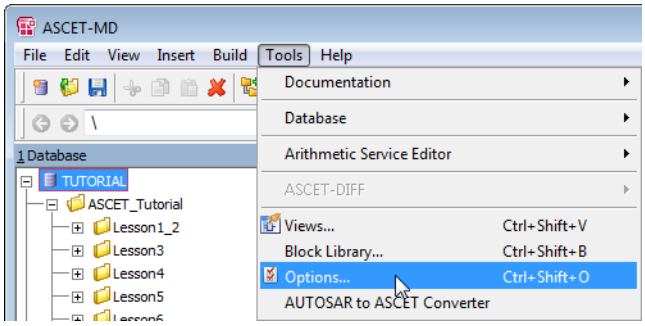
1. Ausführliche Informationen zum Thema „SCOOP-IX“ finden Sie im INTECRIO V3.0 Benutzerhandbuch

Einstellungen in ASCET

Der Transfer des Projekts aus ASCET setzt eine bestimmte Einstellung für eine ASCET-Option voraus.

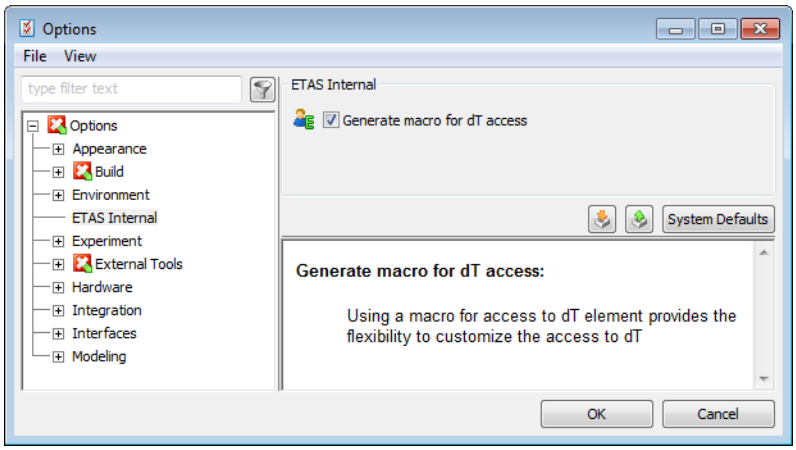
Optionen einstellen

- Starten Sie ASCET.
- Wählen Sie im Hauptmenü **Tools** → **Options**.



Das Fenster „Options“ wird geöffnet.

- Wählen die links den Eintrag „ETAS Internal“ und aktivieren Sie die Option „Generate macro for dT access“.



- Klicken Sie **OK**, um das Fenster zu schließen.

3.3.3 Integration des ASCET-Moduls

In diesem Abschnitt wird beschrieben, wie Sie ein ASCET-Projekt in LABCAR-OPERATOR integrieren.

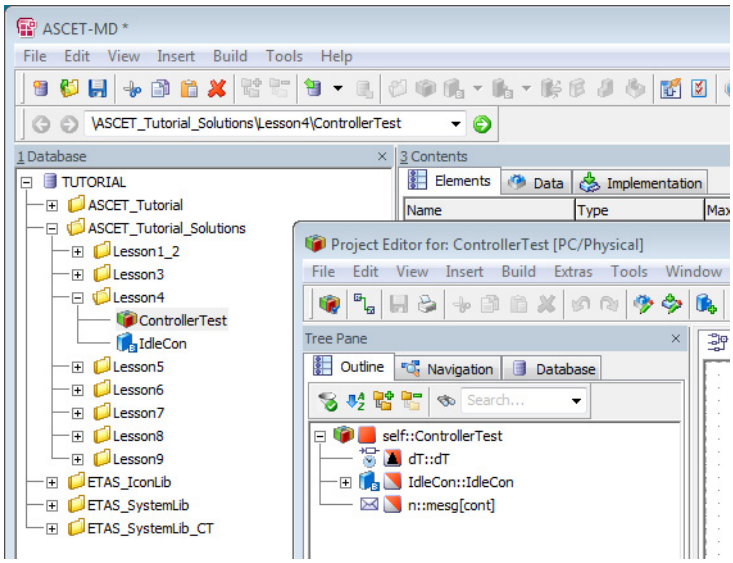
Hinweis

Wenn das Projekt in einer ASCET-Version < V6.0.1 vorliegt, müssen Sie das Projekt (oder die gesamte Datenbank) aus der alten Version exportieren und in die aktuellen Version importieren.

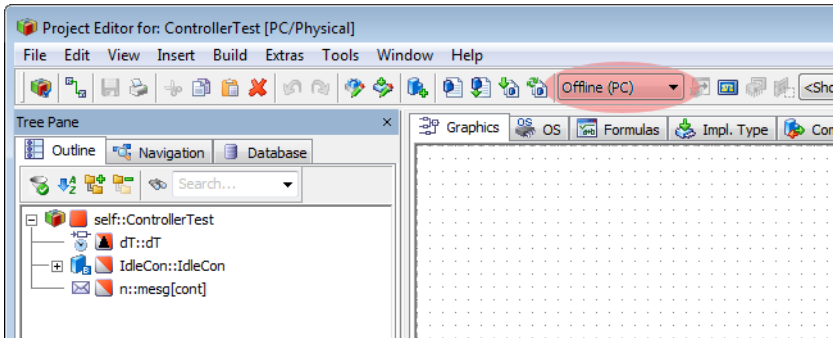
Das ASCET-Projekt transferieren

- Öffnen Sie die jeweilige Datebank in ASCET.
- Doppelklicken Sie das gewünschte Projekt (in diesem Beispiel ist dies das Projekt „ControllerTest“ im Ordner „Lesson4“ der Datenbank „TUTORIAL“).

  Das Projekt wird im Project Editor geöffnet.



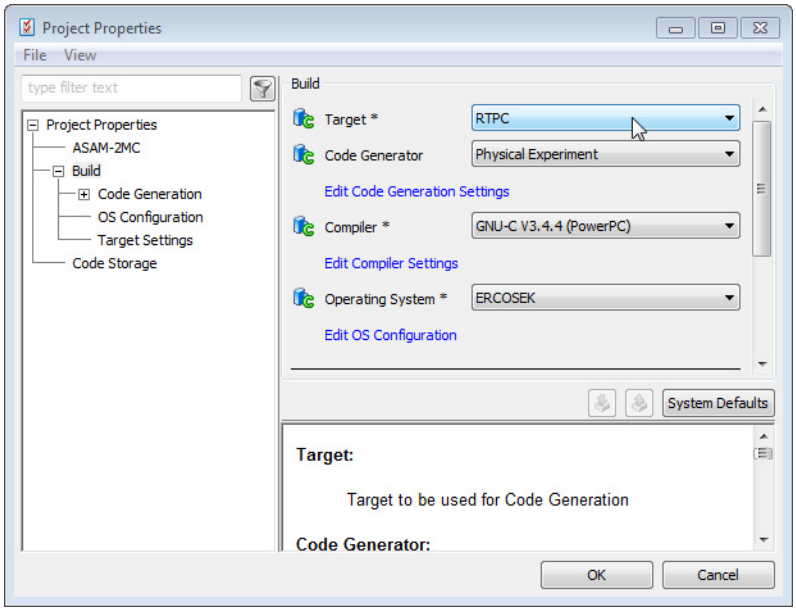
Aktuell ist das Target für die Offline-Simulation auf dem PC gewählt.



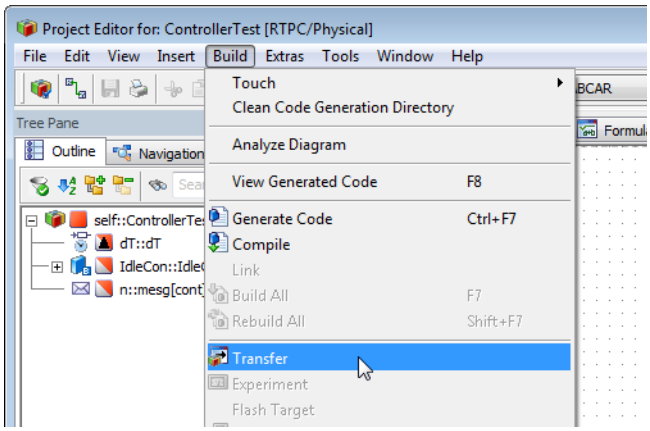
- Wählen Sie im Project Editor **File** → **Properties**.

  Das Fenster „Project Properties“ wird geöffnet.

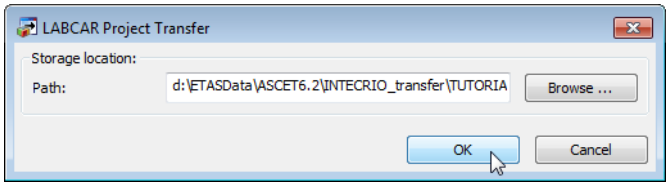
- Wählen Sie links „Build“ und bei Target „RTPC“.



- Klicken Sie **OK**.
- Wählen Sie im ASCET Hauptfenster **File** → **Save**.
- Wählen Sie im Hauptmenü des ASCET Project Editors **Build** → **Touch** → **Recursive**.
- Wählen Sie **Build** → **Transfer**.



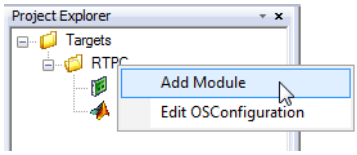
- Wählen Sie ein Verzeichnis, in das die zu erzeugenden Dateien abgelegt werden sollen.



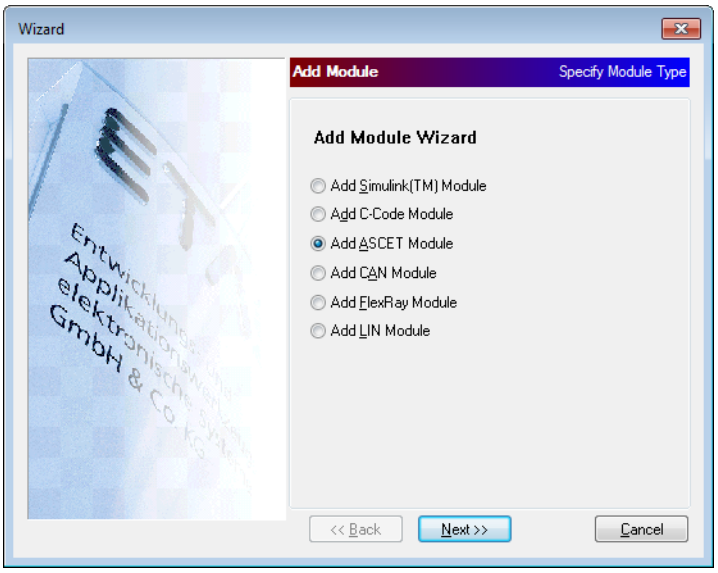
Die Dateien *c, *.h, *<Projektname>*.six und *<Projektname>*.oil werden in diesem Verzeichnis gespeichert.

ASCET-Modul integrieren

- Starten Sie LABCAR-IP und öffnen Sie das LABCAR-OPERATOR-Projekt, zu dem das ASCET-Modul hinzugefügt werden soll.
- Wählen Sie im Project Explorer den Ordner mit dem Namen des Experimentaltargets.
- Rechtsklicken Sie und wählen Sie **Add Module**.

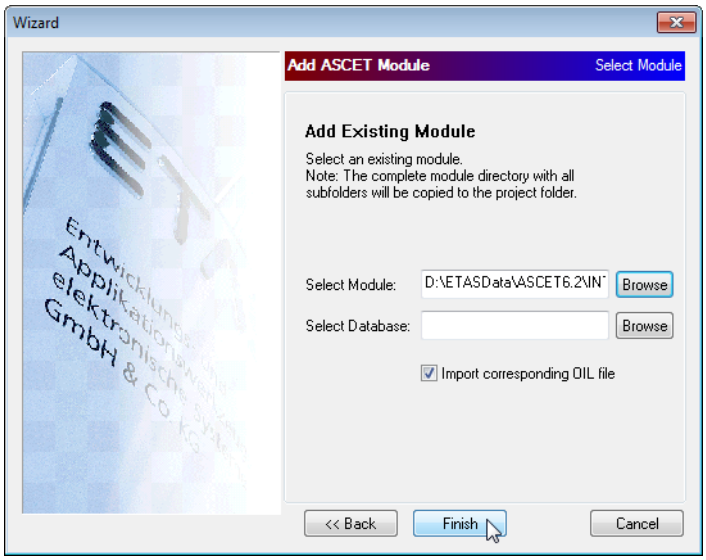


- Im „Add Module Wizard“, wählen Sie „Add ASCET Module“.



- Klicken Sie **Next**.
- Wählen Sie **Use existing ASCET model**.
- Klicken Sie **Next**.

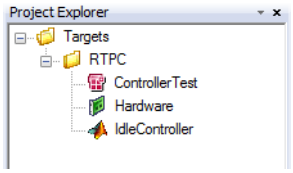
- Wählen Sie die Beschreibungsdatei `ControllerTest.six` des einzufügenden Moduls.



- Wenn Sie (unter „Select Database“) die ASCET-Datenbank angeben, aus der das exportierte Modell stammt, wird diese beim Doppelklick (im Project Explorer) auf das ASCET-Modul geöffnet.

  Ist hier keine Datenbank spezifiziert, muss nach einem Doppelklick der Pfad zu dieser manuell eingegeben werden.
- Klicken Sie **Finish**.

  Das Modul wird integriert und im Project Explorer angezeigt.



- Speichern Sie das Projekt.

  Für die Receive-Messages des ASCET-Projektes, die nicht mit Send-Messages des ASCET-Projektes verbunden sind, werden beim Import Inports erzeugt, die im Connection Manager zur Verbindung mit anderen Signalen des Projekts verbunden werden können. Dasselbe gilt analog für Send-Messages des ASCET-Projekts und Outports im Connection Manager.
- Stellen Sie im Connection Manager die Verbindungen zu anderen Modulen her.
- Generieren Sie den Code für das Experiment (**Project** → **Build**).

- Öffnen Sie das Experiment in ETAS EE (**Tools** → **Open Experiment Environment**).

3.3.4 Messgrössen und Parameter in ETAS Experiment Environment

Das ASCET-Projekt und ETAS EE

Die Komponenten des ASCET-Projekts (im Ordner „Lesson4“ der Datenbank „TUTORIAL“), das im obigen Beispiel verwendet wurde, sind in Abb. 3-4 gezeigt.

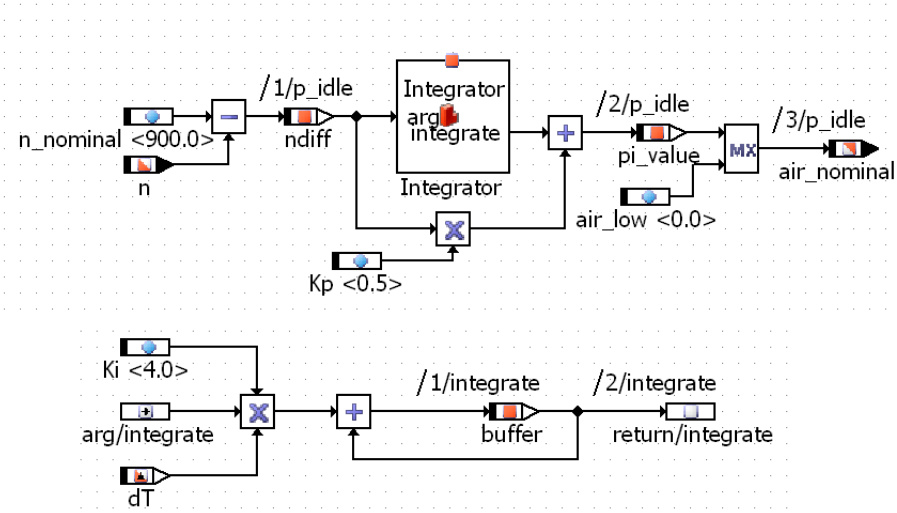


Abb. 3-4 Die Komponenten „IdleCon“ (oben) und „Integrator“ (unten) des Projekts „ControllerTest“

Der Leerlaufregler vergleicht die Ist-Drehzahl „n“ mit der Soll-Drehzahl „n_nominal“ und regelt mit diesen Werten die Soll-Luftzufuhr „air_nominal“.

Die Komponenten, Messgrößen und Parameter des ASCET-Projekts sind in Abb. 3-5 gezeigt.

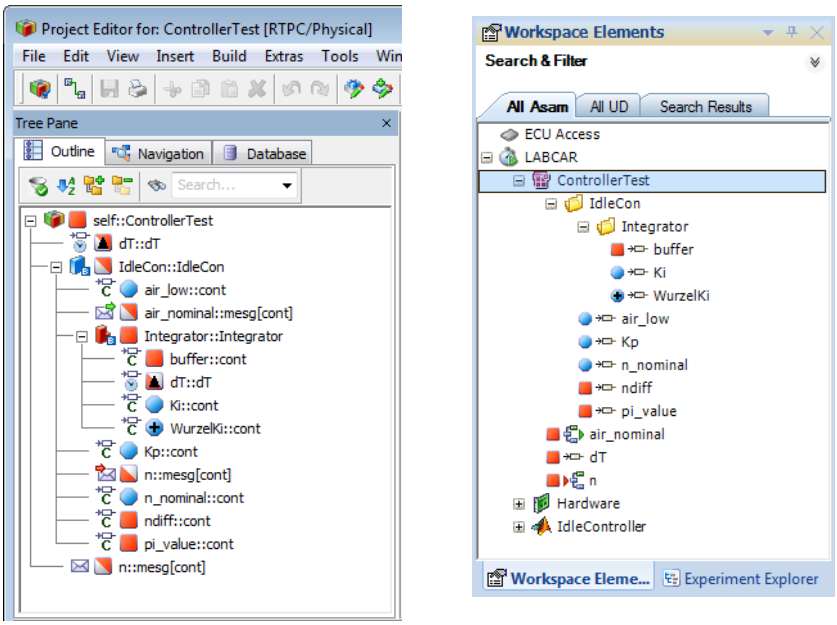


Abb. 3-5 Das ASCET-Projekt „ControllerTest“ (links) und das integrierte Modul im Fenster „Workspace Elements“ von ETAS Experiment Environment (rechts).

3.4 C-Code-Module

Neben MATLAB/Simulink-Modellen und ASCET-Modulen ist es in LABCAR-IP auch möglich, vom Anwender definierten C-Code in das Projekt zu integrieren.

Diese ermöglicht insbesondere:

- Die einfache Integration beliebiger Funktionalität für preiswerte, aber flexible anwenderspezifische Lösungen
- Die Erstellung von LABCAR-konformen Simulationsmodellen mit Third-Party-Tools
- Die Wiederverwendung von älterem LABCAR-Code
- Die Integration von Functional Mock-Up Units (FMU)

Dieses Kapitel enthält ein Tutorial, das die verschiedenen Arten der Erstellung von C-Code-Modulen illustriert. Im Einzelnen finden Sie hier Informationen zu folgenden Themen:

- „Manuelles Erstellen eines C-Code-Moduls“ auf Seite 61
- „Code hinzufügen“ auf Seite 69
- „Erstellen eines C-Code-Moduls mit dem Automation Server“ auf Seite 73
- „C-Code-Modul exportieren“ auf Seite 73
- „Bestehendes C-Code-Modul hinzufügen“ auf Seite 73
- „Änderungen eines existierenden C-Code-Moduls“ auf Seite 76
- „Linken von externem C-Code“ auf Seite 76
- „Variablenlabels“ auf Seite 78
- „Functional Mock-up Units (FMU)“ auf Seite 78

3.4.1 Tutorial

In diesem Tutorial wird beschrieben, wie Sie - mit Hilfe von C-Code-Modulen - C-Code in ein LABCAR-Projekt einbinden.

Diese Integration umfasst die folgenden Schritte:

- Spezifikation des Interfaces des C-Code-Moduls
  - entweder manuell (siehe „Manuelles Erstellen eines C-Code-Moduls“ auf Seite 61)

  oder

  - über die Automatisierungsschnittstelle (siehe „Erstellen eines C-Code-Moduls mit dem Automation Server“ auf Seite 73)
- Hinzufügen des Codes zu dem erstellten Interface

Diese Schritte werden im Folgenden unter Verwendung eines einfachen Code-Beispiels durchgeführt.

Das Code-Beispiel

Als Beispiel soll ein sehr einfaches Bremssystem dienen. Der Eingang des Bremssystems wird von der Bremspedalstellung (0...1) bereit gestellt. Der Ausgang des Systems besteht aus vier Werten M_LF, M_RF, M_LR, M_RR, die die jeweiligen Bremsmomente enthalten, die auf die vier Räder wirken.

Die Berechnung der Ausgänge wird folgendermaßen durchgeführt:

Zuerst wird die Pedalstellung (`pedal`) mittels eines Faktors (`pedal_pressure_factor`) in den hydraulischen Bremsdruck umgerechnet. Anschließend erfolgt mittels weiterer Umrechnungsfaktoren (`pressure_torque_factor[]`) die Berechnung der einzelnen Bremsmomente.

Das Modul wird somit durch die folgenden Gleichnungen beschrieben:

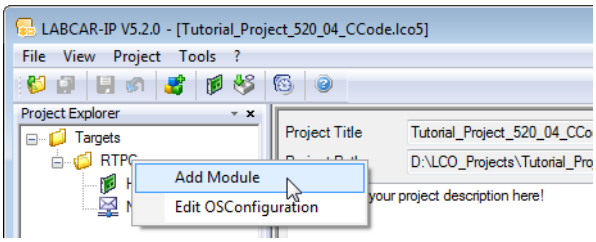
```
M_LF = pressure * pressure_torque_factor[0]
M_RF = pressure * pressure_torque_factor[1]
M_LR = pressure * pressure_torque_factor[2]
M_RR = pressure * pressure_torque_factor[3]
wobei
   pressure = pedal * pedal_pressure_factor
```

3.4.2 Manuelles Erstellen eines C-Code-Moduls

Für die manuelle Erstellung eines C-Code-Moduls steht in LABCAR-IP ein Wizard zur Verfügung.

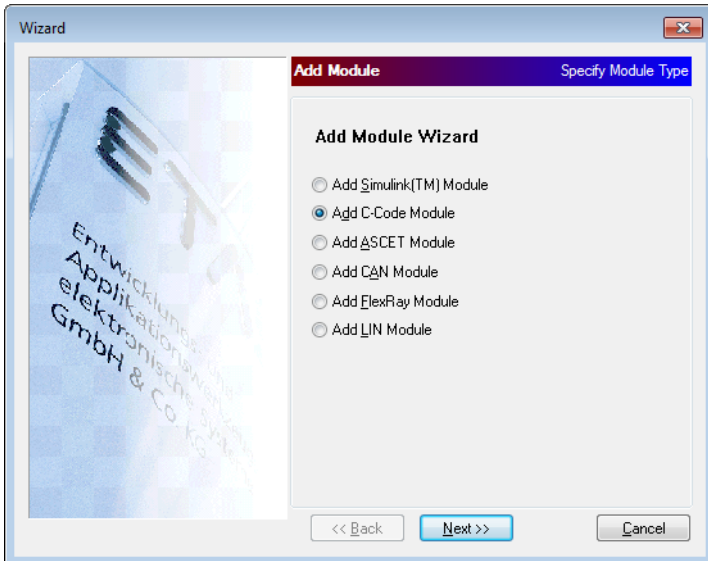
C-Code-Modul erstellen

- Öffnen Sie das LABCAR-Projekt, in dem Sie das C-Code-Modul erstellen wollen.
- Wählen Sie im Project Explorer das Target.
- Im Kontextmenü wählen Sie **Add Module**.

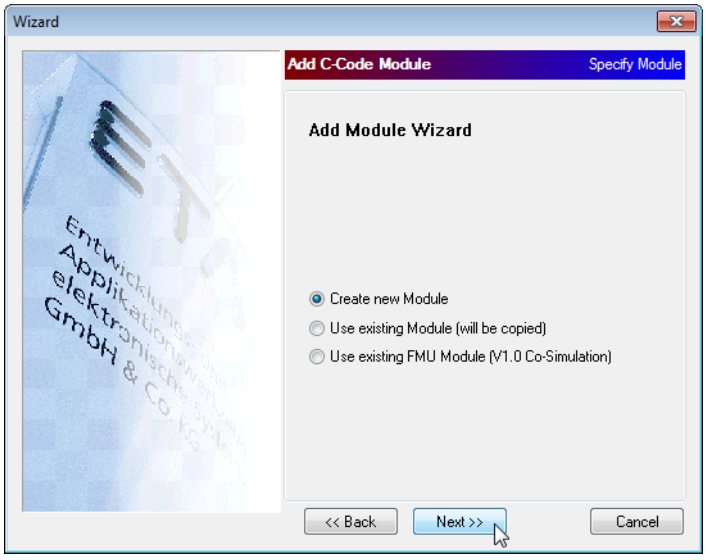


Der Wizard wird geöffnet.

- Wählen Sie die Option „Add C-Code Module“.

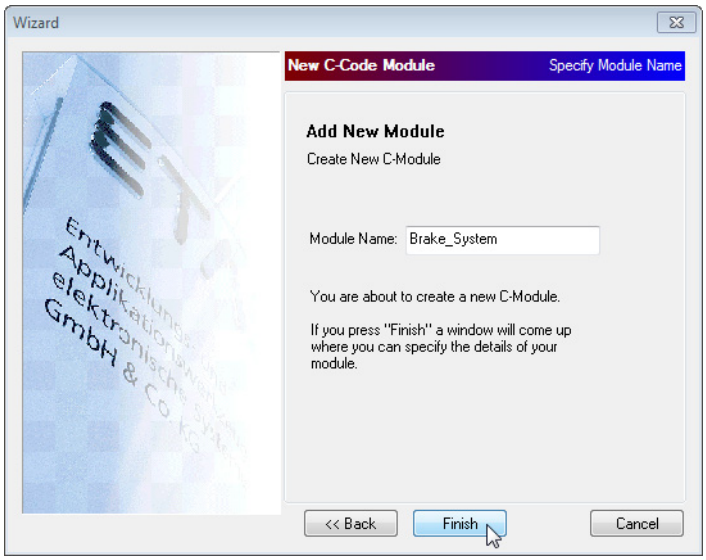


- Klicken Sie **Next**.
- Wählen Sie die Option „Create new Module“.



- Klicken Sie **Next**.

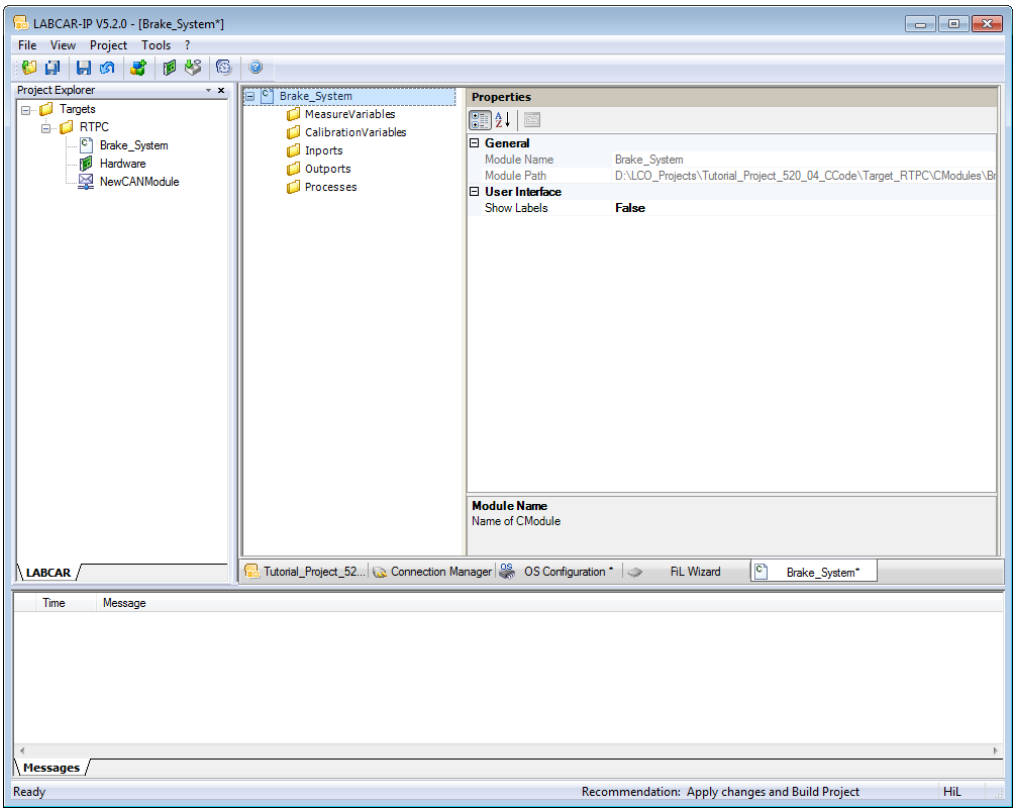
- Geben Sie im nächsten Fenster einen Namen für das zu erstellende Modul ein.



- Klicken Sie **Finish**.

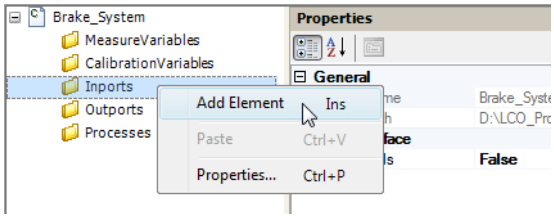
  Das C-Code-Modul wird erstellt und im Project Explorer dargestellt.
- Doppelklicken Sie das Modul.

  Das Register mit dem C-Code Editor wird geöffnet.



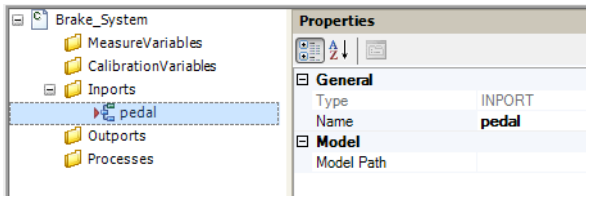
Einen Moduleingang definieren

- Um einen Eingang zu definieren, rechtsklicken Sie den Ordner „Inports“.
- Im Kontextmenü wählen Sie **Add Element**.



Das Element „Inport“ wird erstellt.

- Wählen Sie das Element und geben Sie (im Fenster „Properties“ bei „Name“) einen Namen („pedal“) für den Inport an.



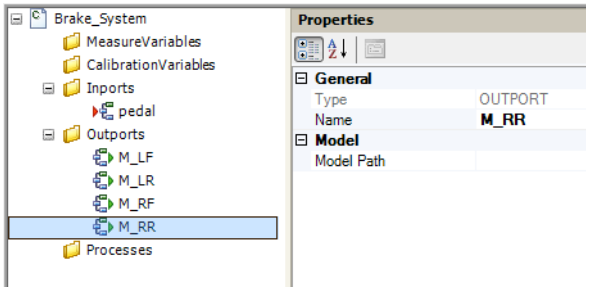
- Klicken Sie **File → Save All**.

Hinweis

Inports und Outports sind immer Skalare vom Datentype „Double“. Daher sind keine weiteren Angaben erforderlich.

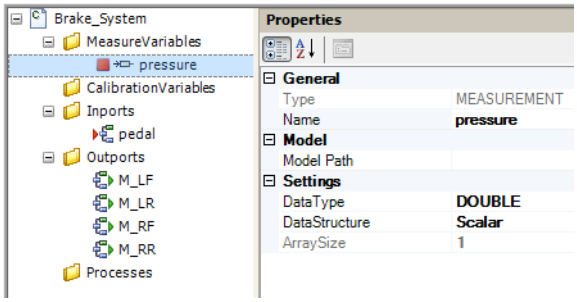
Modulausgänge definieren

- Um einen Modulausgang zu definieren, rechtsklicken Sie den Ordner „Outports“.
- Im Kontextmenü wählen Sie **Add Element**.
- Geben Sie einen Namen („M_LF“) für den Outport an.
- Legen Sie drei weitere Ausgänge mit den Namen „M_RF“, „M_LR“ und „M_RR“ an (wie oben beschrieben oder via **Copy** und **Paste** im Kontextmenü).



Eine Messgröße definieren

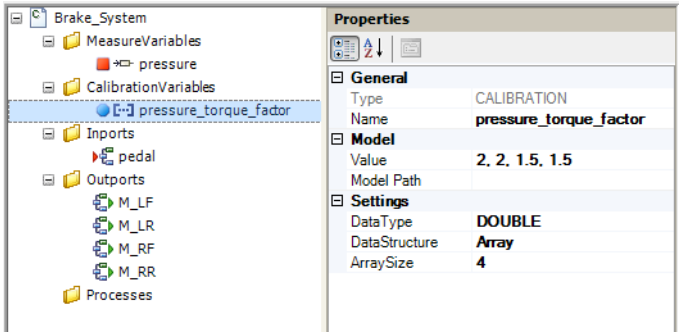
- Um eine Messgröße zu definieren, rechtsklicken Sie den Ordner „MeasureVariables“.
- Im Kontextmenü wählen Sie **Add Element**.
- Geben Sie eine Namen („pressure“) für den Outport an.
- Wählen Sie bei „Data Type“ „DOUBLE“.
- Wählen Sie unter „DataStructure“ „Scalar“.



Kalibriergrößen definieren

- Um eine Kalibriergröße zu definieren, rechtsklicken Sie den Ordner „CalibrationVariables“.
- Im Kontextmenü wählen Sie **Add Element**.

  Die erforderlichen Eingaben hier sind im Prinzip die selben wie bei einer Messgröße bis auf die Tatsache, dass einer Kalibriergröße ein Wert zugewiesen wird.
- Nennen Sie die Variable „pressure_torque_factor“.
- Wählen Sie „DOUBLE“ als Datentyp.
- Wählen Sie „Array“ mit „ArraySize“ = 4.
- Die Wertzuweisung erfolgt unter „Value“, wobei die einzelnen Werte durch Kommata getrennt werden.



- Erstellen Sie eine weitere Kalibriergröße namens „pedal_pressure_factor“ und weisen Sie ihr den Wert 200 zu.

Prozesse

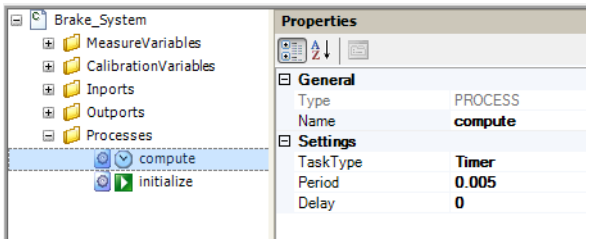
Ein Prozess ist eine Funktion, die vom Betriebssystem aufgerufen wird - die Art des Aufrufs hängt vom Prozesstyp ab:

- Ein Init-Prozess wird einmal beim Start des Experiments aufgerufen.
- Ein Exit-Prozess wird einmal beim Beenden des Experiments aufgerufen.
- Ein Timer-Prozess wird in einem speziellen Raster (definiert durch „Period“ und „Delay“) aufgerufen
- Es ist aber nicht unbedingt nötig, einen Prozess einer Task zuzuweisen - man kann dann von einem „Event“-Prozess sprechen.

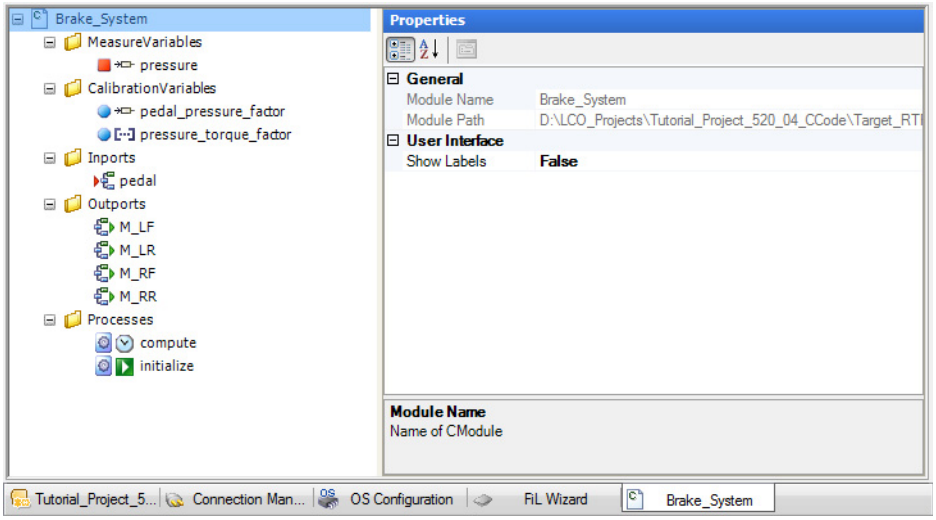
Einen Prozess hinzufügen

- Um einen Prozess zu definieren, rechtsklicken Sie den Ordner „Processes“.
- Im Kontextmenü wählen Sie **Add Element**.
- Geben Sie als Prozessnamen („Name“) „initialize“ ein.
- Wählen Sie bei „TaskType“ „Init“.
- Definieren Sie eine weiteren Prozess namens „compute“.

- Als „TaskType“ wählen Sie „Timer“ mit folgenden Einstellungen:
  - Period = 0.005 s
  - Delay = 0 s

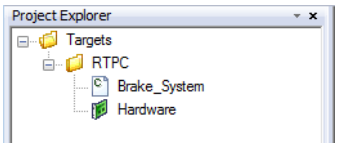


Das Register „Brake_System“ sollte nun aussehen wie folgt:



- Wählen Sie **File** → **Save**.

  Das soeben erstellte Modul wird gespeichert und zum Projekt hinzugefügt.



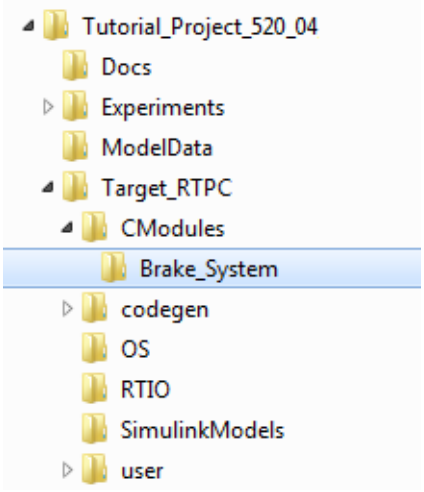
Nach dem Speichern des Projektes werden die folgenden Schritte durchgeführt:

- Überprüfen der Daten auf Konsistenz

  Hier wird z.B. überprüft, ob der Modulname gültig ist oder die Namen der Variablen und Prozesse eindeutig sind.

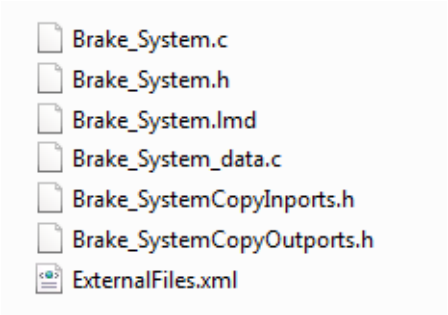
- Erstellen der Verzeichnisse

  Für das Modul namens „Brake_System“ wird im Projektordner ein Verzeichnis angelegt.



- Generieren der Dateien

  In dem o.g. Verzeichnis werden die folgenden Dateien angelegt:



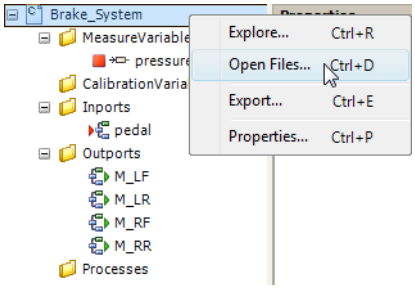
  Neben den Dateien, die den C-Code und weitere Definitionen enthalten (*.c und *.h) wird insbesondere die *.lmd Datei erzeugt.

- Integration in das Datenmodell von LABCAR-OPERATOR.

Quell- und Include-Dateien editieren

Um die Quell- (*.c) und Include-Dateien des Moduls im dafür zugewiesenen Editor (z.B. Visual Studio) zu öffnen, gehen Sie wie folgt vor:

- Wählen Sie im Kontextmenü des C-Code-Moduls **Open Files**.



Die zum Modul gehörenden Dateien werden im gewählten Editor geöffnet.

3.4.3 Code hinzufügen

Der eigentliche Code, wie er aus den bis dato gemachten Angaben erzeugt werden konnte, befindet sich in der Datei `Brake_System.c`. Diese Datei ist eine Vorlage mit „leeren“ Prozessen, zu denen jetzt Code hinzugefügt werden muss.

Der Inhalt der Datei ist im Folgenden gezeigt, der fett gedruckte Teil stellt den hinzugefügten Code für das Bremssystem dar.

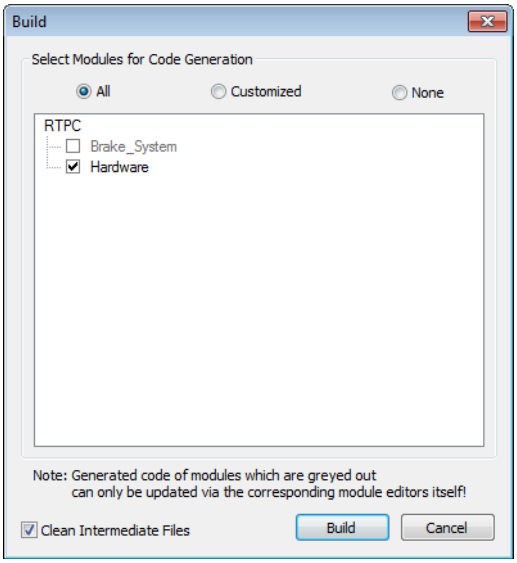
```
// Add application-specific include statements here:

#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"
}
```

In dem hinzugefügten Code können die beim Erstellen des C-Code-Moduls definierten Variablen verwendet werden. Die Deklaration dieser Variablen finden Sie in der Datei „Brake_System.h“. Was die Funktion „initialize“ anbelangt, wird diese im Beispiel nicht benötigt - sie dient hier lediglich zur Demonstrationszwecken.

Damit ist die Erstellung des C-Code-Moduls beendet.

Wählen Sie nun **Project** → **Build**, um das Projekt zu erstellen.

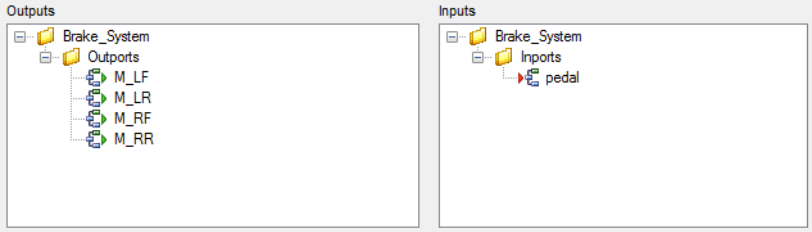


Das C-Code-Modul „Brake_System“ ist im Fenster „Build“ nicht wählbar, da hier kein Code mehr generiert werden muss. Klicken Sie **Build**.

Nach dem erfolgreichen Build-Vorgang finden Sie alle In- und Outports, Mess- und Kalibriergrößen in den entsprechenden Fenstern der Bedienoberflächen von LABCAR-OPERATOR.

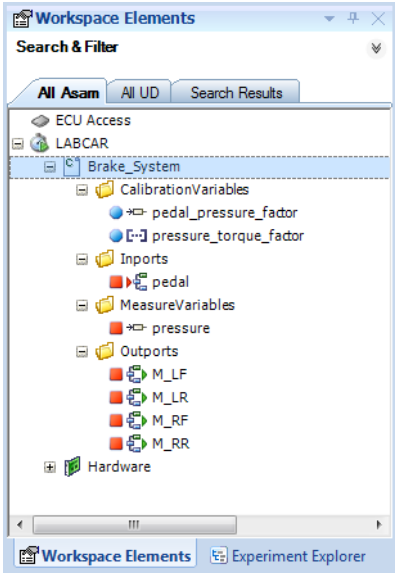
Connection Manager

Die In- und Outports sind im Connection Manager (in LABCAR-IP) zur Verbindung mit anderen Modulen verfügbar. Wählen Sie ggf. **Update Ports**.



Workspace Elements

Beim Experimentieren in ETAS EE stehen die In- und Outports sowie die Mess- und Kalibriervariablen des Moduls im Fenster „Workspace Elements“ zur Verfügung.

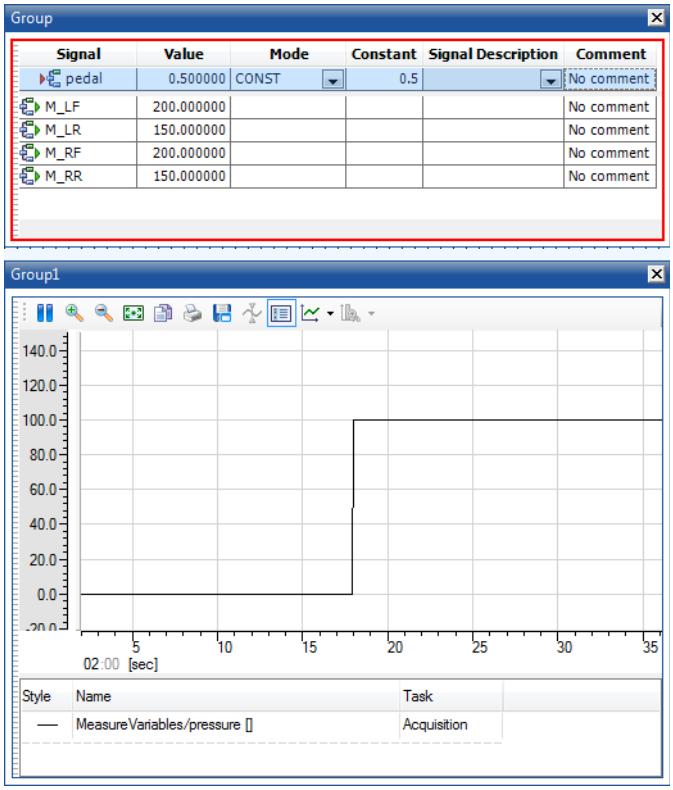


Wenn Sie mit dem Modul experimentieren wollen, führen Sie einen Build durch und öffnen Sie das Experiment in der Experimentierumgebung.

Simulation durchführen

- Erstellen Sie eine Signal List.
- Fügen Sie dieser Signal List den Inport und die vier Outports hinzu.
- Erstellen Sie ein Oszilloskop und fügen Sie die Messgröße „pressure“ hinzu.
- Starten Sie das Experiment.
- Setzen Sie beim Inport „pedal“ „Mode“ auf „CONST“ und weisen Sie dem Inport den Wert 0,5 zu.

- Drücken Sie <RETURN>.



„pressure“ nimmt jetzt den Wert „100“ an - das Bremsmoment der beiden Vorderräder beträgt jetzt 200 und das der beiden Hinterräder 150.

3.4.4 Erstellen eines C-Code-Moduls mit dem Automation Server

Die im vorhergehenden Abschnitt beschriebene Aktionen zur Erstellung eines C-Code-Moduls wie das Hinzufügen von Ein- oder Ausgängen, Variablen etc. sind auch über API-Funktionen verfügbar. Das „ICModuleManager Interface“ stellt dafür die Schnittstellenfunktionen zur Verfügung.

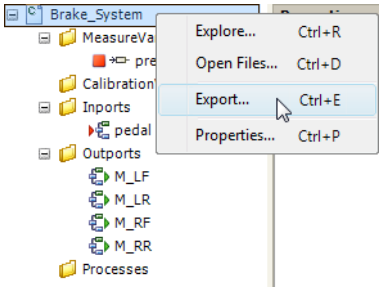
Die Dokumentation (chm-Datei) dazu finden Sie über **? → Help** unter „API Documentation“.

3.4.5 C-Code-Modul exportieren

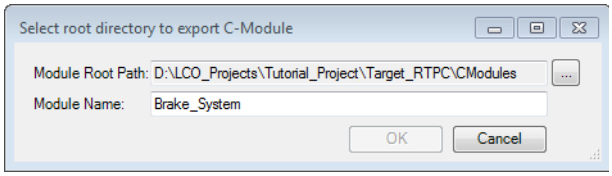
Sie können das C-Code-Modul (d.h. die dazu gehörenden Dateien – siehe „Generieren der Dateien“ auf Seite 68) auch (mit einem frei wählbaren Namen) exportieren, um es ggf. zu anderen Projekten wieder hinzuzufügen.

C-Code-Modul exportieren

- Wählen Sie im Kontextmenü des C-Code-Moduls **Export**.



- Wählen Sie im folgenden Dialog den Pfad und geben Sie ggf. dem Modul einen anderen Namen.



Das Modul wird unter dem angegebenen Namen exportiert.

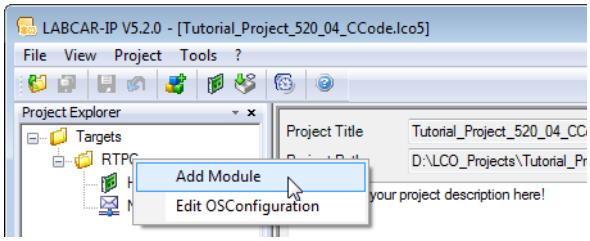
3.4.6 Bestehendes C-Code-Modul hinzufügen

Sie können Ihrem Projekt auch ein bereits vorhandenes C-Code-Modul hinzufügen – gehen Sie dazu wie folgt vor.

Modul hinzufügen

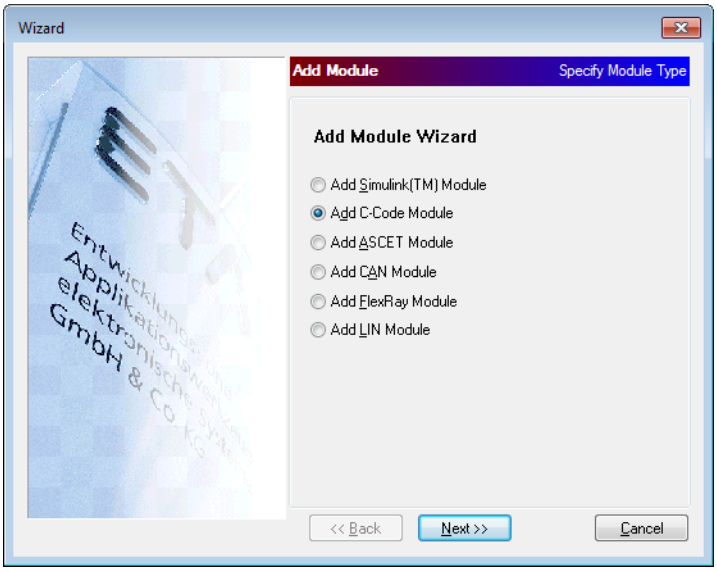
- Öffnen Sie das LABCAR-Projekt, dem Sie das C-Code-Modul hinzufügen wollen.
- Wählen Sie im Project Explorer das Target „RTPC“.

- Im Kontextmenü wählen Sie **Add Module**.

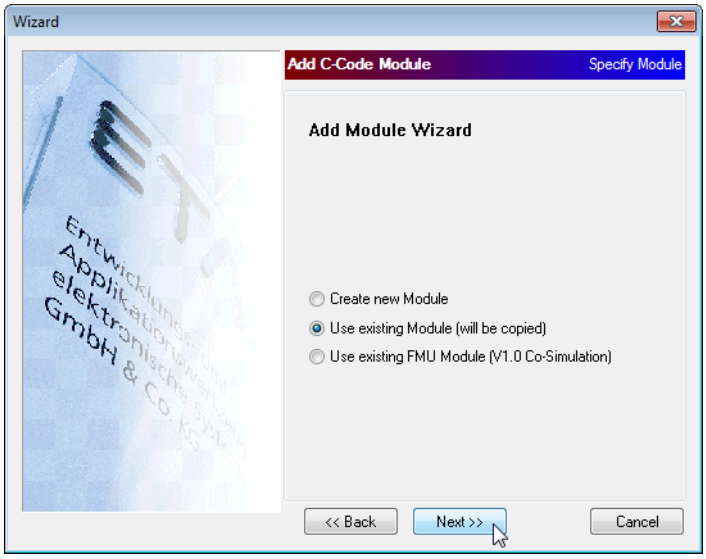


Der Wizard wird geöffnet.

- Wählen Sie „Add C-Code Module“.

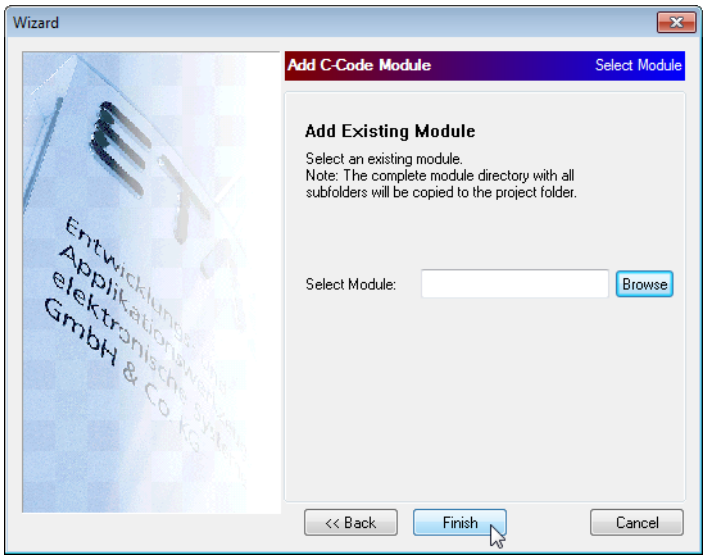


- Klicken Sie **Next**.
- Wählen Sie „Use existing Module“.



- Klicken Sie **Next**.

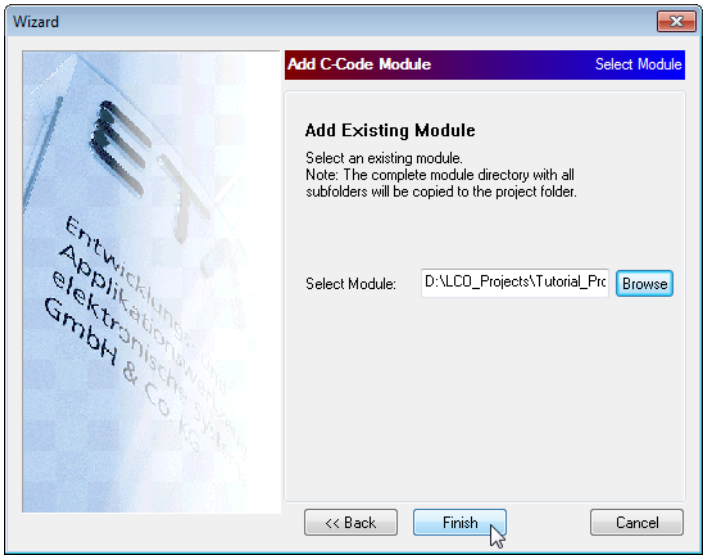
  Das folgende Fenster dient zur Auswahl des Moduls.



- Klicken Sie **Browse**.

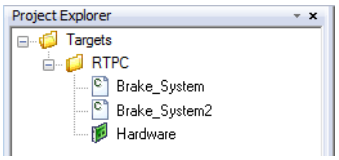
  Ein Dateiauswahlfenster wird geöffnet.

- Wählen Sie die Datei und klicken Sie **Öffnen**.



- Klicken Sie **Finish**.

Das soeben eingefügte Modul „Brake_System2“ wird jetzt im Project Explorer parallel zum bereits vorhandenen Modul „Brake_System“ angezeigt.



Nachdem Sie die bereits im ersten Beispiel hinzugefügten Gleichungen wieder eingefügt haben, führt auch dieses Beispiel zu dem selben Simulationsergebnis. Der einzige Unterschied besteht in den geänderten Pfaden der Messgröße und der Parameter im Fenster „Workspace Elements“ in ETAS EE.

3.4.7 Änderungen eines existierenden C-Code-Moduls

Wenn Sie ein bereits bestehendes C-Code-Modul verändern, müssen Sie u.U. auch das Interface durch Hinzufügen, Bearbeiten oder Löschen von Variablen oder Prozesse ändern.

Wenn Sie danach **File → Save** wählen, werden fünf der weiter oben beschriebenen Dateien neu erstellt, die Datei mit dem eigentlichen Code `<Modul-name>.c`, die auch den hinzugefügten Code enthält, bleibt entweder unverändert oder wird erweitert.

Dabei gelten die folgenden Regeln:

- Wenn lediglich Veränderungen bezüglich der Variablen vorgenommen wurden, betrifft dies die C-Datei nicht.
- Wenn ein neuer Prozess hinzufügt wurde, wird das jeweilige Funktionstemplate zum Code hinzugefügt.
- Wird ein Prozess gelöscht, bleibt die entsprechende Funktion erhalten.
- Wenn ein Prozess umbenannt wird, so ist dies, als ob der Prozess gelöscht und ein neuer hinzugefügt worden wäre. Dies bedeutet, dass ein neues Funktionstemplate in der C-Datei hinzugefügt wird, die Funktion aber leer ist. Der Code der alten Funktion muss also von Anwender per Hand in die neue Funktion übertragen werden.

Durch diese Vorgehensweise wird sichergestellt, dass Code niemals gelöscht wird. Wenn die Funktion eines gelöschten Prozesses in der Datei verbleibt, wird zwar Code generiert, was aber ohne Bedeutung ist, solange die Funktion nicht aufgerufen wird.

Der einzig mögliche Fall, in dem Code per Hand gelöscht werden muss, tritt dann ein, wenn ein Prozess mit einem bestimmten Namen zuerst gelöscht und anschließend mit dem gleichen Namen wieder erstellt wird. Dann sind im C-Code zwei Funktionen mit demselben Namen vorhanden, was nicht zulässig ist.

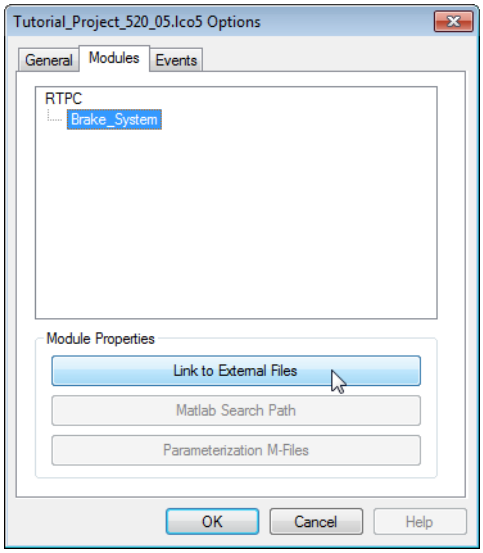
Wenn Code an eine Datei angehängt wird, wird auf jeden Fall vorher eine Sicherungskopie angelegt, die den alten Dateinamen mit einer angehängten, fortlaufenden Nummer trägt.

3.4.8 Linken von externem C-Code

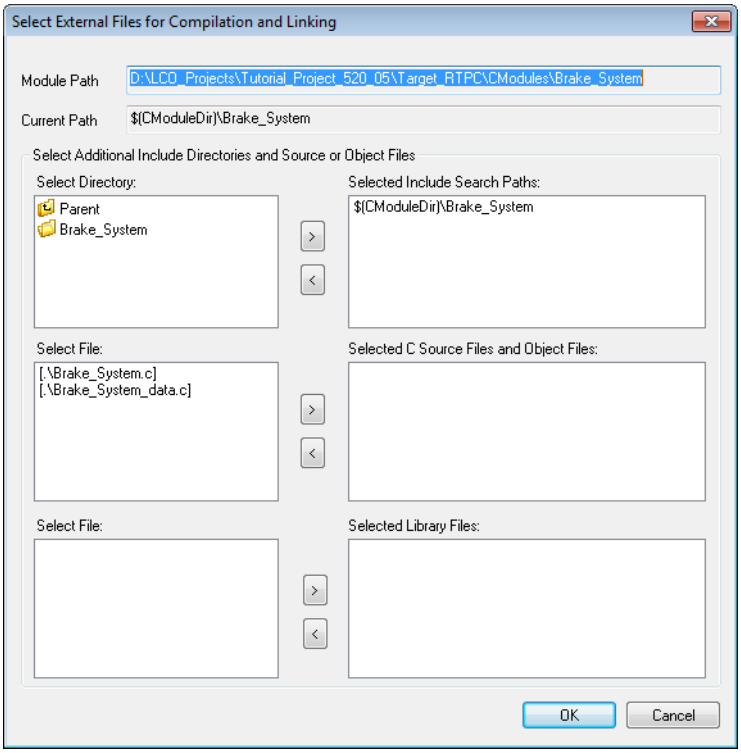
Die vom Editor erstellte C-Code-Datei muss mit vom Anwender definiertem C-Code ergänzt werden. Dies kann durch ein direkt eingefügtes Stück Code erfolgen, es kann aber auch auf externe Quellen verwiesen werden.

- Wählen Sie dazu **Project → Options** im Hauptmenü von LABCAR-OPERATOR.
- Wählen Sie die Registerkarte „Modules“.
- Wählen Sie das Modul, für das die externen Dateien gelinkt werden sollen.

- Klicken Sie **Link to External Files.**



Das Fenster zum Auswählen von Suchpfaden, Quell- und Objektdateien und Bibliotheken wird geöffnet.



- Wählen Sie die entsprechenden Dateien.
- Klicken Sie zweimal **OK**.

3.4.9 Variablenlabels

Neben der Verwendung von Standardlabeln ist es in LABCAR-OPERATOR möglich, vom Anwender definierte Labels zu verwenden. Dies gilt auch sowohl bei der Spezifikation eines C-Code-Moduls von Hand als auch bei der Erstellung über API-Funktionen.

Dieses Label entspricht dem Pfad, unter dem Variablen, Ein- und Ausgänge in den Workspaces Elements (ETAS EE) oder im Connection Manager dargestellt werden.

Werden Labels nicht explizit spezifiziert, werden folgende Defaultlabels verwendet:

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

Wird vom Anwender ein bestimmtes Label spezifiziert, z.B. `User/ Defined/Path` für eine Messvariable, wird die Variable in der jeweiligen Hierarchie so dargestellt:

`<TargetName>/<ModuleName>/User/Defined/Path/ <VariableName>`

3.4.10 Functional Mock-up Units (FMU)

In diesem Abschnitt wird beschrieben, wie Sie Functional Mock-up Units (FMU) in ein LABCAR-OPERATOR-Projekt importieren können – insbesondere wird die Erstellung des Shared Object Files für Linux beschrieben.

Im Einzelnen finden Sie Informationen zu folgenden Themen:

- „Übersicht“ auf Seite 78
- .„Integration einer FMU in LABCAR-IP“ auf Seite 80
- „Generieren des Shared Object File (.so)“ auf Seite 83
- „Hinweise und Einschränkungen“ auf Seite 90

Übersicht

Functional Mock-up Interface (FMI) definiert einen Tool-unabhängigen Schnittstellenstandard für die Co-Simulation und den Austausch von Modellen aus unterschiedlichen Quellen.

Über die Erstellung einer Software-Bibliothek namens FMU (Functional Mock-up Unit) können Modelle eines Modelliertools in einer anderen Toolumgebung ausgeführt werden.

Es gibt verschiedene Typen von FMUs

- Model Exchange

  Hier werden die Solver des Tools verwendet, in das die FMU importiert wurde.

- Co-Simulation

  Hier werden die Solver des Simulationswerkzeuges verwendet, aus dem das Modell stammt.

  Bei Co-Simulation unterscheidet man wiederum zwei Typen:

  – Standalone

    Bei diesem Typ sind die Solver in der FMU „eingebaut“.

  – ToolExchange

    Hier werden Methoden aufgerufen, die die Solver des Simulationstools verwenden.

Hinweis

LABCAR-OPERATOR V5.4.1 unterstützt Co-Simulation (Version 1.0) vom Typ „Standalone“.

Jedes FMU-Modell (FMU = Functional Mock-up Unit) wird als Zip-Datei mit der Erweiterung „fmu“ zur Verfügung gestellt, die Folgendes enthält:

- `sources`

  Dieser Ordner enthält die Quelldateien mit den C-Funktionen der Modellgleichungen.

- `binaries`

  Dieser Ordner enthält Unterordner mit den Binärdateien für die verschiedenen Plattformen:

  – `win32` / `win64`

  – `linux32` / `linux64`

- `modelDescription.xml`

  Diese Datei enthält die Definition aller Variablen des Modells und weitere Modellinformationen.

- Optionale weitere Ordner mit Daten (wie Parametertabellen, die Benutzeroberfläche etc.), die vom Modell benötigt werden.

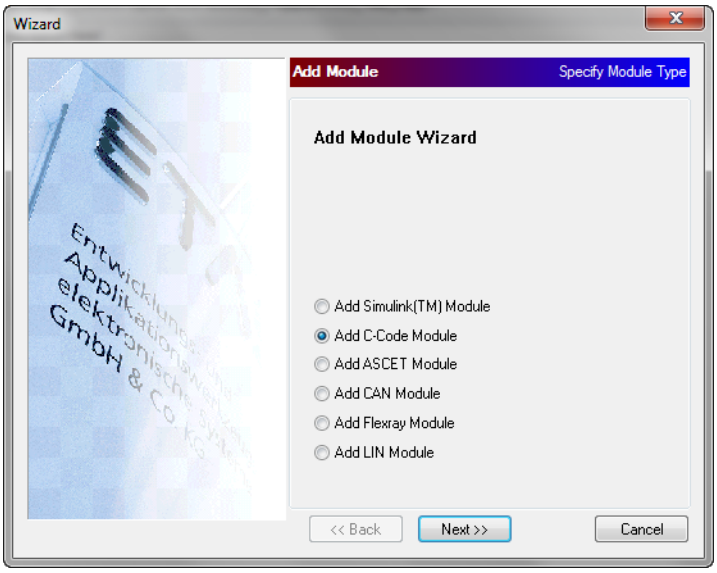
Integration einer FMU in LABCAR-IP

Hinweis

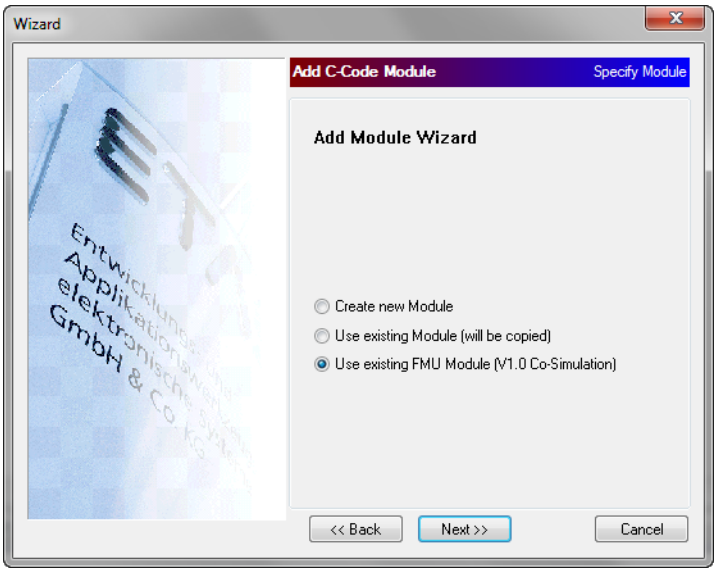
Das FMU-Modul kann nur integriert werden, wenn die Quelldateien der FMU vorhanden sind. Wenn bereits eine Linux-Binärdatei (Shared Object File) vorhanden ist, die auf anderem Wege als im Folgenden beschrieben erzeugt wurde, funktioniert das LABCAR-OPERATOR-Projekt möglicherweise nicht korrekt.
Es wird deshalb dringend empfohlen, die Linux-Binärdatei wie in Abschnitt „Generieren des Shared Object File (.so)“ auf Seite 83 beschrieben zu erstellen!

Zur Integration einer FMU gehen Sie wie folgt vor:

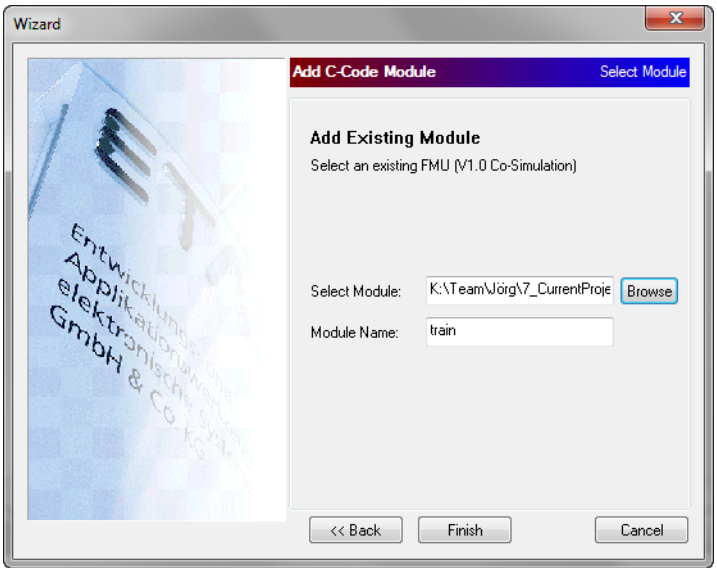
- Wählen Sie wie bei der Integration eines C-Code-Moduls **Add Module**.



- Wählen Sie **Add C-Code Module** und klicken Sie **Next**.



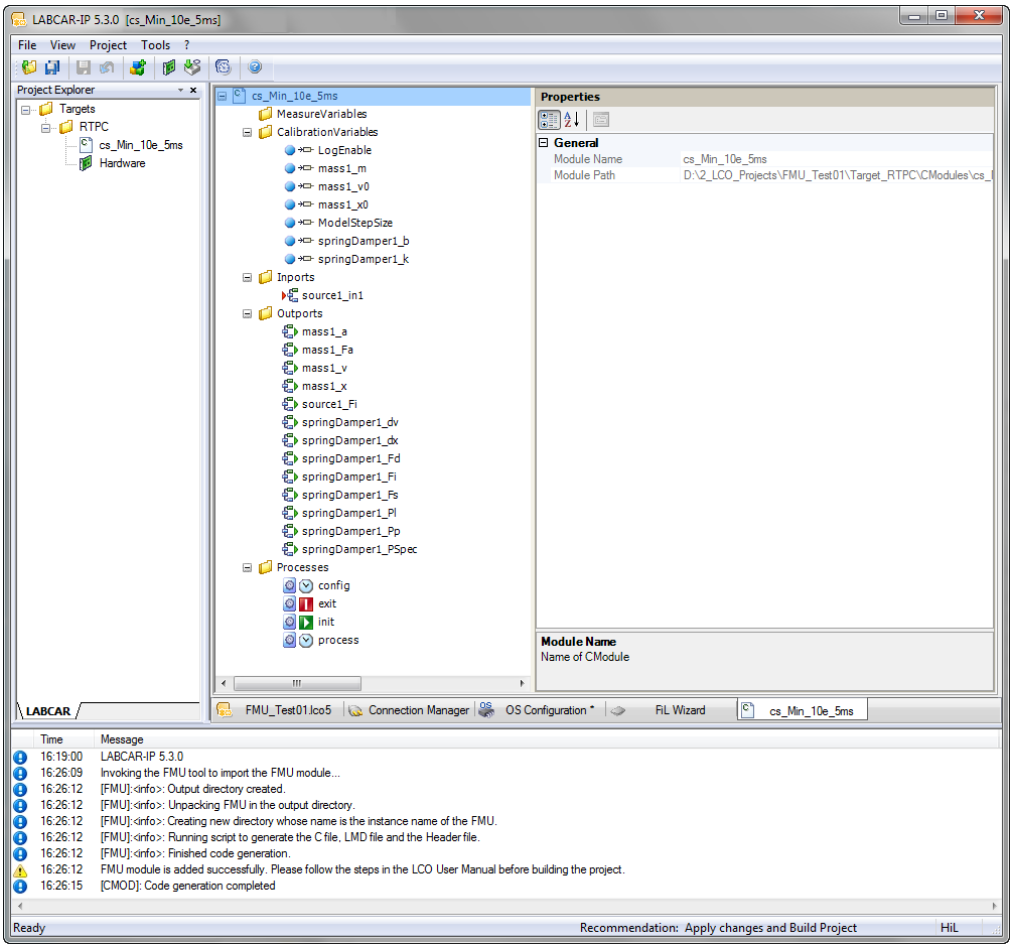
- Wählen Sie **Use existing FMU Module** und klicken Sie **Next**.



- Geben Sie den Pfad zur *.fmu-Datei an und vergeben Sie einen Modulnamen (muss ein gültiger C-Identifier sein).

- Klicken Sie **Finish**.

  Das Modul wird integriert und in LABCAR-IP dargestellt.



Vor dem Build des Projektes muss das noch zu generierende Shared Object File (.so) in das Projekt kopiert werden (siehe „Generieren des Shared Object File (.so)“ auf Seite 83).

- Kopieren Sie die „.so“ Datei in folgende Verzeichnisse:

  `Target_<TargetName>\User\lib`

  `Target_<TargetName>\runtime-data\lib`

- Jetzt kann der Build des Projektes durchgeführt werden.

Generieren des Shared Object File (.so)

Ein Shared Object File ist eine Binärdatei, die alle Funktionalität der FMU enthält. Diese Datei muss (wie im Folgenden beschrieben) aus den Quelldateien erzeugt werden.

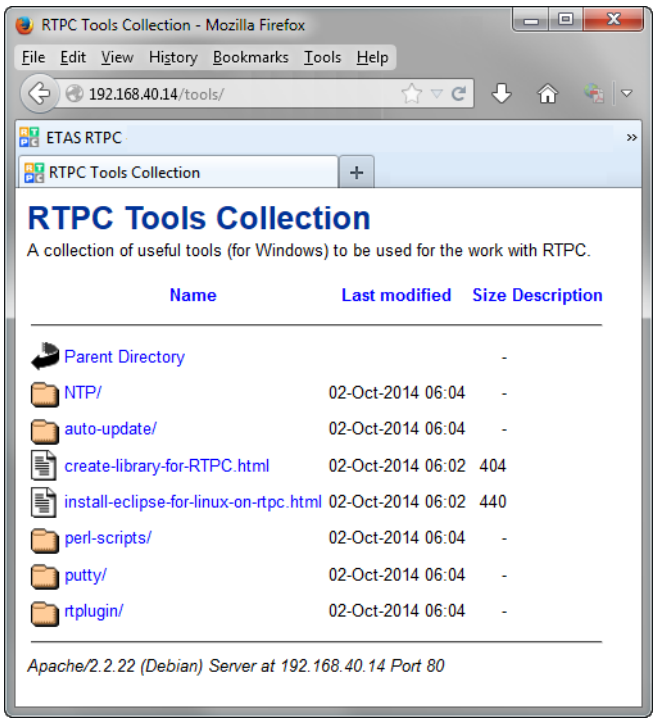
Generieren des Shared Object file

Bei der Integration einer FMU in LABCAR-IP, werden die C-Quelldateien in das Projektverzeichnis in den Ordner des FMU-Moduls kopiert. Wurde das LABCAR-OPERATOR-Projekt im Verzeichnis `D:\Temp` erstellt und der Name des FMU-Moduls lautet „Engine“, so befinden sich die Quelldateien der FMU im Verzeichnis

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

- Zippen Sie alle Dateien **im Ordner** (**nicht den Ordner!**) `FMU_Sources` in eine Datei.

  Damit ist sichergestellt, dass sich die Make-Datei `makefile_labcar` im Hauptverzeichnis befindet.
- Öffnen Sie das Web-Interface von ETAS RTPC.
- Stoppen Sie den Simulationscontroller.
- Wechseln Sie zur Hauptseite und klicken Sie **Tools**.



- Klicken Sie den Link **create-library-for-RTPC.html**.

- Klicken Sie **Delete All Files**.
- Klicken Sie **Choose File**.
- Wählen Sie die zuvor erstellte Zip-Datei `FMU_Sources.zip`.
- Klicken Sie **Upload**, um die Datei auf den Real-Time PC hochzuladen.
- Klicken Sie im Abschnitt „Build User Library“ **Build**.

  Wenn der Build-Prozess erfolgreich war, wird das erstellte Source Object File im Abschnitt „Download Library Files“ angezeigt.
  Sind degegen Fehler aufgetreten, werden diese im Log-Fenster ausgegeben. Lösungen für mögliche Fehler finden Sie im Abschnitt „Fehlerbehebung“ auf Seite 85.

- Um das Source Object File auf Ihren PC herunterzuladen, klicken Sie den entsprechenden Link im Abschnitt „Download Library File.

Hinweis

Die mitgelieferte Make-Datei `makefile_labcar` ist lediglich ein Template, in dem alle im Ordner `FMU_Sources` vorhandenen C-Dateien berücksichtigt werden. Außerdem sind bestimmte Flags für die Integration von mehreren FMUs enthalten.
Darüberhinaus kann es für den Buildvorgang erforderlich sein, weitere Compilerflags, Definitionen und Include-Verzeichnisse hinzuzufügen. Auch kann es erforderlich sein, bestimmte C-Dateien von der Kompilierung auszuschliessen.

Fehlerbehebung

Question: Die Quelldateien enthalten weder die Datei `fmiFunctions.h` and noch die Datei `fmiPlatformTypes.h`. Wo kann ich diese bekommen?

Response: Diese Dateien können Sie von der Hompage von JModelica herunter laden:

`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: Wie ändere ich den Namen einer Variablen?

Response: Zum Ändern eines Variablennamens gehen Sie wie folgt vor:

- Ändern Sie den Variablennamen im „Properties“ Fenster.
- Öffnen Sie die Datei `<FMU_InstanceName>_FMU.c` (im Ordner `Target_<TargetName>/CModules/instanceName`”).
- Suchen Sie in der Methode `cmod_process_<InstanceName>FMU()` den alten Variablennamen und ersetzen Sie Ihn durch den neuen.

Beispiel: Wenn der Variablenname von „value“ nach „valueone“ geändert werden soll und der Code folgende Zeile enthält:

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

muss diese ersetzt werden durch:

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

Hinweis

Stellen Sie vor dem Build des Projektes sicher, dass der Ordner `FMU_Sources` (im LABCAR-OPERATOR Projektverzeichnis) gelöscht wurde!

Frage: Nach dem Hochladen der Datei `FMU_Sources.zip` wird die Meldung

```
The makefile "makefile_labcar" will be used to build
the library
```

nicht ausgegeben und zudem wird während des Build-Vorgangs die Meldung

```
Missing Arguments
```

ausgegeben.

Antwort: Die Ursache dafür kann eine nicht korrekte Zip-Datei der Quelldateien im Ordner `\FMU_Sources` sein. Wenn die gepackte Datei funktioniert, wird ihr Inhalt (nach dem Hochladen der Zip-Datei) im Web-interface im Abschnitt „Build User" Library" (**Main Page → User Library**) dargestellt.

Frage: Der Build-Vorgang wird mit Fehlermeldungen des Compilers abgebrochen.

Antwort: Ein Abbruch des Build-Vorganges kann eine Reihe von Ursachen haben. Abhängig vom Tool, mit dem die FMU erstellt worden ist, sollten Sie eine der folgenden Lösungen ausprobieren.

Hinweis

Grundsätzlich ist zu empfehlen, die Make-Datei des Erstellers der FMU (sofern diese zur Verfügung steht) als Ausgangspunkt zu verwenden und die Make-Datei `makefile_labcar` entsprechend anzupassen.

Lösung 1: Wenn die FMU mit MapleSim® erstellt wurde und der Build-Vorgang aufgrund eines Compilerfehlers abgebrochen wurde, kann es sein, dass sich unter den Quelldateien eine Datei `fmuTemplate.c` befindet. Wenn diese Datei von anderen Dateien durch Include-Anweisungen verwendet wird, kann dies zu Fehlern führen.

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:
```

- Ändern Sie die Make-Datei durch Entfernen der Zeile:

```
./fmuTemplate.o: fmuTemplate.c
        $(CC) $(CFLAGS) -c ./fmuTemplate.c
```

und

- entfernen Sie

```
./ fmuTemplate.o \
```

in der OBJECTS Variablen in der Make-Datei.

Wenn die Make-Datei so aussieht:

```
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)
```

- ändern sie diese zu:

```
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)
```

Wenn der Fehler weiterhin besteht, könnte dies an fehlenden Headerdateien liegen, die Sie vom Ersteller der FMU erhalten können.

Lösung 2: Wenn die FMU mit SimulationX® erstellt wurde und Sie erhalten Fehlermeldungen wie unten gezeigt, kann das daran liegen, dass in der Make-Datei Definitionen fehlen.

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'
```

Für das Tool SimulationX® kann das an der fehlenden Definition

```
-DITI_CVODE_EXT
```

in der CFLAGS Variablen liegen.

- Fügen Sie diese Variable hinzu:

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

Frage: Beim Build der .so-Datei erscheint folgende Fehlermeldung:

```
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'
```

Antwort: Der Grund für diese Fehlermeldung ist das Fehlen des Makros `#define FMI_COSIMULATION` im Quellcode.

Öffnen Sie die Hauptquelldatei (üblicherweise die C-Datei mit dem Namen der FMU und fügen Sie das Makro am Anfang der Dateiwie unten gezeigt ein:

```
/* 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
```

Frage: Der Build der Shared Object Datei war erfolgreich, aber beim Hinzufügen mehrerer FMUs zum LABCAR-OPERATOR-Projekt schlägt der Buildvorgang fehl.

Antwort: Grund dafür können Konflikte zwischen den Methoden verschiedener FMUs sein. Um dieses Problem zu beheben, müssen die Quelldateien entsprechend angepasst werden.

Diese Anpassungen werden von LABCAR-IP automatisch durchgeführt, wenn das Verzeichnis `\FMU_Source` eine Datei namens `fmiFunctions.h` enthält.

Ist diese Datei nicht vorhanden, sind manuelle Anpassungen erforderlich. Führen Sie dazu folgende Schritte durch:

- Suchen Sie die Headerdatei, die den String „fmiInitializeSlave“ enthält.
- Öffnen Sie diese Datei in einem Texteditor und ersetzen Sie die Zeile

  `#define DllExport`

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

durch die folgende Zeile:

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

Hinweise und Einschränkungen

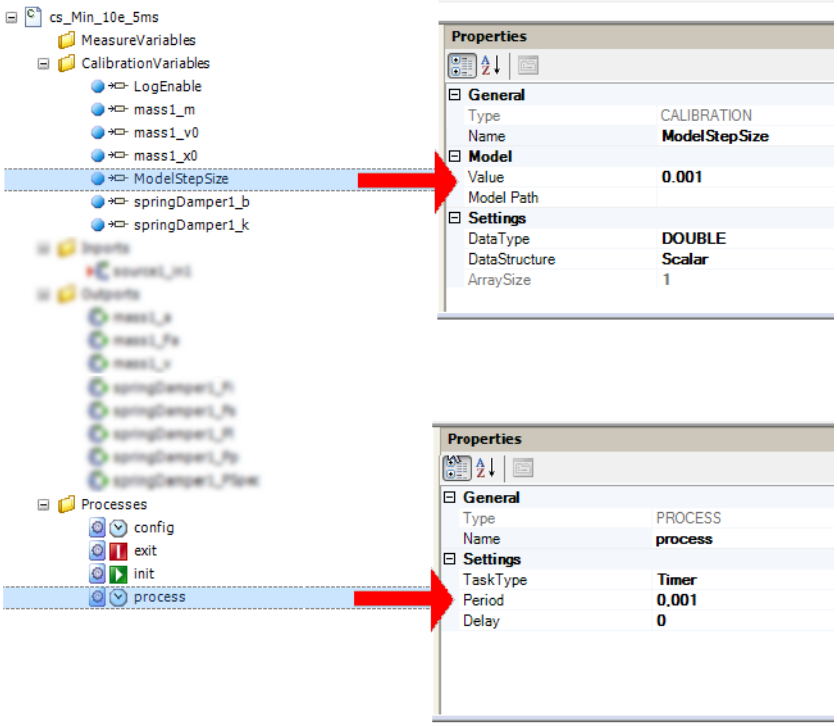
- **Namenskonventionen für das Shared Object File**

  Der Name des Shared Object File darf nicht mit der Buchstabenfolge „lib“ gefolgt vom Modulnamen beginnen. Zudem darf der Name nur aus Kleinbuchstaben bestehen.

  Beispiel: Besteht das Projekt aus zwei Modulen „gear“ und „Engine“, so sind Shared Object Files mit Namen wie `libgear.so` und `libEngine.so` nicht zulässig.

- **Taskdauer und Schrittweite des Modells**

  Die Periode des Prozesses „process“ des Moduls muss identisch sein mit der Schrittweite des Models (Typ „CALIBRATION“).



- **Kalibriervariablen**

  Stellen Sie sicher, das die Werte der Kalibriervariablen vor der Ausführung des Experiments in der Experimentierumgebung definiert werden. Diese können während der Asuführung des Experiments nicht mehr geändert werden.

- **Inports und Outports**

  LABCAR-IP ignoriert Inports und Outports des Typs „String“, Boolean“ und „Enum“, da diese nicht unterstützt werden - Arrays werden ebenfalls ignoriert. Für FMUs in LABCAR-IP werden lediglich Skalare unterstützt.

- **Kalibrier- und Messvariablen**

  LABCAR-IP unterstützt keine Kalibrier- und Messvariablen des Typs „String“ und „Enum“.

3.5 CAN-Module (Network Integration CAN)

LABCAR-NIC V5.4.1 (Network Integration CAN) ist ein Add-On zu LABCAR-OPERATOR V5.4.1. Es ermöglicht ein einfaches Testen von Steuergerätefunktionen, die CAN-Kommunikation beinhalten.

Die gesamte CAN-Bus-Kommunikation wird aus einer oder mehreren CANdb-Dateien eingelesen, der Anwender wählt die physikalisch vorhandenen CAN-Knoten (die UuT) und LABCAR-NIC V5.4.1 erstellt automatisch Code für den zu simulierenden Restbus. Zu diesem kann benutzerdefinierter Code (z.B. Zähler oder Berechnung von Checksummen) hinzugefügt werden.

Die Signale der ausgewählten Messages sind im Connection Manager verfügbar und können dort mit Modelleingängen (Receive-Messages) und Modellausgängen (Send-Messages) verbunden werden.

ETAS EE bietet zur Darstellung von Messages spezielle GUIs und einen CAN-Monitor.

Seit Version 5.0 von LABCAR-NIC wird auch das J1939 Netzwerkprotokoll unterstützt (siehe „Die J1939-Erweiterung“ auf Seite 93).

Hardwareanforderungen

Arbeiten mit Network Integration CAN setzt das Vorhandensein eines Real-Time PC mit ETAS RTPC V6.2.0 und (mindestens) eines CAN-Boards des Typs „iPC-I XC16/PCI“ oder „CAN-IB200/PCIe“ der Firma IXXAT voraus.

Eine Einbindung des CAN-Boards in die RTIO (LABCAR-RTC V5.4.1) ist nicht erforderlich - die Konfiguration der CAN-Boards wird im Abschnitt „Bus“ auf Seite 108 beschrieben.

3.5.1 Einführung

Als Beispiel sei ein CAN-Netzwerk mit vier Knoten (bezeichnet als „Node A, B, C und D“) und den CAN-Messages 1 - 5 betrachtet. Die in der CANdb-Datei beschriebene Kommunikation ist in Abb. 3-6 gezeigt.

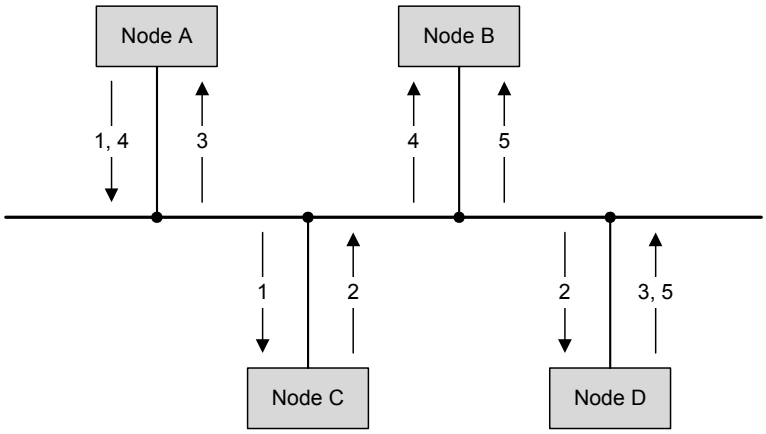


Abb. 3-6 CAN-Netzwerk mit vier Knoten und fünf Messages

Von den vier Knoten seien zwei physikalisch vorhanden, nämlich C und D, die die UuT bilden. Die „Gegenstellen“ der Messages 1, 3 und 5 müssen simuliert werden.

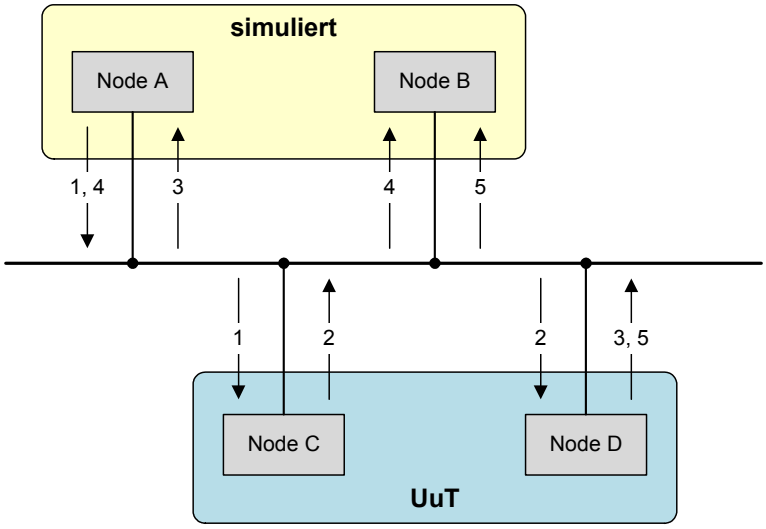


Abb. 3-7 Physikalisch vorhandene (UuT) und simulierte Knoten

Die Message 4 wird innerhalb der simulierten Umgebung ausgetauscht und spielt daher bei der Kommunikation mit der UuT keine Rolle. Die verbleibende zu simulierende Kommunikation bildet den sog. „Restbus“, für den Code generiert werden muss.

Der simulierte Restbus besitzt die Send-Message 1 und die Receive-Messages 3 und 5.

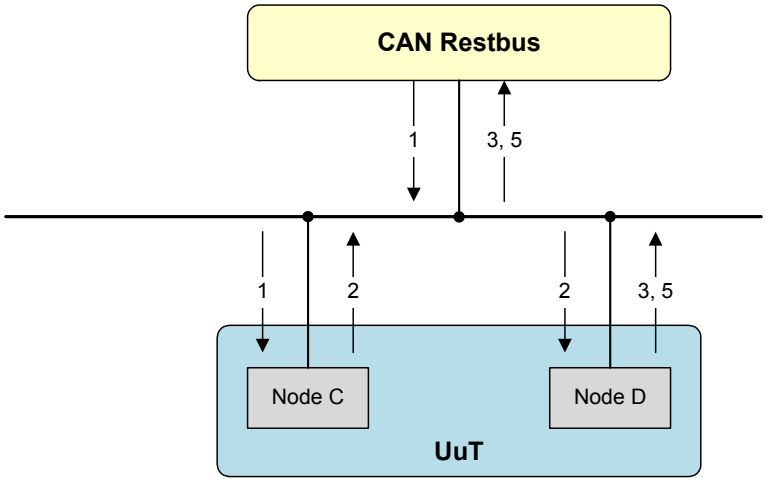


Abb. 3-8 Messages vom und zum Restbus

3.5.2 Die J1939-Erweiterung

Das Netzwerkprotokoll J1939 basiert auf einer Reihe von Standards der Society of Automotive Engineers (SAE). J1939 beschreibt die Kommunikation auf einem CAN-Bus in Nutzfahrzeugen (z.B. Antriebsstrang und Chassis) und arbeitet auf der Bitübertragungsschicht mit CAN-Highspeed nach ISO11898.

Die speziellen Eigenschaften von J1939 sind:

- 29-Bit Identifier nach CAN 2.0B (Extended Format)
- Kommunikation sowohl Peer-to-peer als auch als Broadcast
- Transportprotokolle für bis zu 1785 Byte Daten (BAM (Broadcast Announce Message) und CMDT (Connection Mode Data Transfer))
- Dezentrales Netzwerk-Management
- Informationen werden in Form von Parametern (Signalen) übermittelt, die wiederum in Parameter Groups zusammengefasst und durch eine eindeutige Zahl, die Parameter Group Number (PGN), identifiziert werden.

Eigenschaften der J1939-Erweiterung in LABCAR-NIC V5.4.1

LABCAR-NIC V5.4.1 verarbeitet jetzt auch Busbeschreibungen nach dem J1939-Standard einschließlich:

- Handhabung von 29-Bit Message-Identifiern einschließlich deren Interpretationsregeln für J1939
- Netzwerkmanagement und den entsprechenden Transportprotokollen
- Import von J1939-Busbeschreibungen

Hinweis

Die dbc-Dateien müssen im Format „J1939 PG (ext. ID)" vorliegen, das von CANalyzer V6.0 und höher unterstützt wird – nur bei diesem Format wird der vollständige 29 Bit-Identifier verwendet.

- Konfiguration von Parametern für die Bussimulation

und

- Codegenerierung für die IXXAT CAN Boards im Real-Time PC

Lange CAN-Messages

Inbesondere wird die Übertragung von langen CAN-Messages mit einer Nutzlast von maximal 1785 Byte unterstützt. Auch diese Messages können im CAN-Editor von LABCAR-IP erstellt und bearbeitet werden – deren Verhalten im Connection Manager ist dasselbe wie bei gewöhnlichen CAN-Botschaften.

Hinweis

Um die Dekodierung von J1939-Messages auf einem CAN-Controller zu aktivieren, muss für diesen Controller mindestens eine lange (> 8 Byte) J1939-Message als Send-, Receive- oder Gateway-Message im CAN-Editor konfiguriert sein.

Derzeit nicht unterstützt wird die Übertragung von CAN-Messages variabler Länge (DLC).

Multiple Session Support (J1939-21))

Zum Empfang von langen Messages ist „Multiple Session Support“ implementiert – für die Anzahl gleichzeitig (verschachtelt) empfangbarer Messages gibt es keine Obergrenze.

Was das Senden von langen Messages anbelangt, so unterstützt jedes CAN-Interface die Übertragung jeweils einer langen Message. Sind zwei oder mehr Messages so geplant, dass sich deren Übertragung überlappen würde, so werden diese nacheinander übertragen. Die gleichzeitige Übertragung von langen Messages über verschiedene CAN-Interfaces ist erlaubt.

Zykluszeit

Bei der Koordination zyklischer langer Messages wird die Zykluszeit definiert als die Zeit, die zwischen den jeweils ersten Paketen zweier Messages vergeht.

Beispiel:

- Das Paket 0 der Message 0 wird gesendet zum Zeitpunkt: t_{00}
- Das Paket 1 der Message 0 wird gesendet zum Zeitpunkt: t_{01}
- Das Paket 2 der Message 0 wird gesendet zum Zeitpunkt: t_{02}

  .....
- Das Paket 0 der Message 1 wird gesendet zum Zeitpunkt: t_{10}

Die Zykluszeit beträgt dann $t_{10} – t_{00}$ (Zeit zwischen den jeweils ersten Paketen).

3.5.3 Der CAN Editor

In diesem Abschnitt finden Sie Informationen zum Arbeiten mit dem CAN-Editor.

CAN-Editor öffnen

- Doppelklicken Sie im Project Explorer das CAN-Modul, das Sie im Editor anzeigen oder bearbeiten möchten.

  Das CAN-Modul wird im Register „CAN Editor“ des Hauptfensters dargestellt.

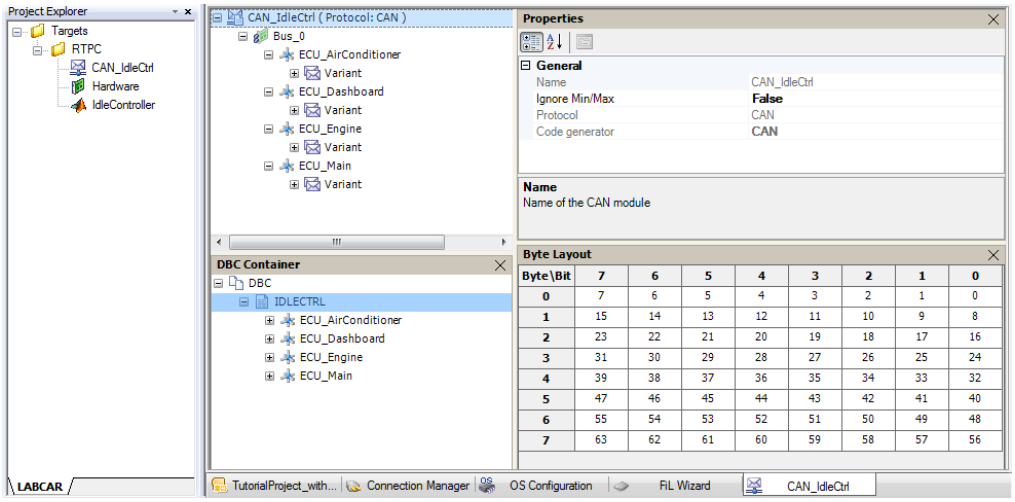


Abb. 3-9 Der CAN-Editor

Der CAN-Editor besteht aus

- der Darstellung des CAN-Netzwerkes,
- dem DBC Container
- dem Fenster „Properties“

und

- einer Anzeige für das Byte-Layout von Frames.

Darstellung des CAN-Netzwerkes

In dieser Ansicht wird das aus tatsächlich vorhandenen Boards und Controllern aufgebaute Netzwerk mit den untergeordneten Elementen hierarchisch angezeigt.

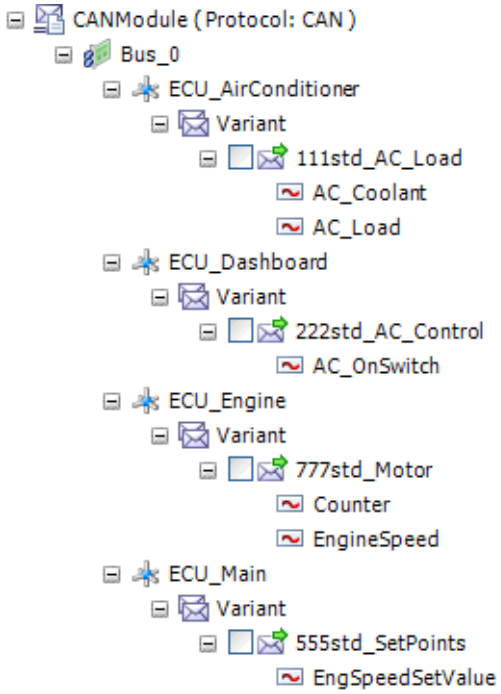


Abb. 3-10 Das CAN-Netzwerk

Informationen zum Aufbau des Netzwerkes finden Sie im Abschnitt „Bearbeiten des CAN-Netzwerks“ auf Seite 99, zu dessen einzelnen Bestandteilen in „Die Bestandteile eines CAN-Netzwerks“ auf Seite 107.

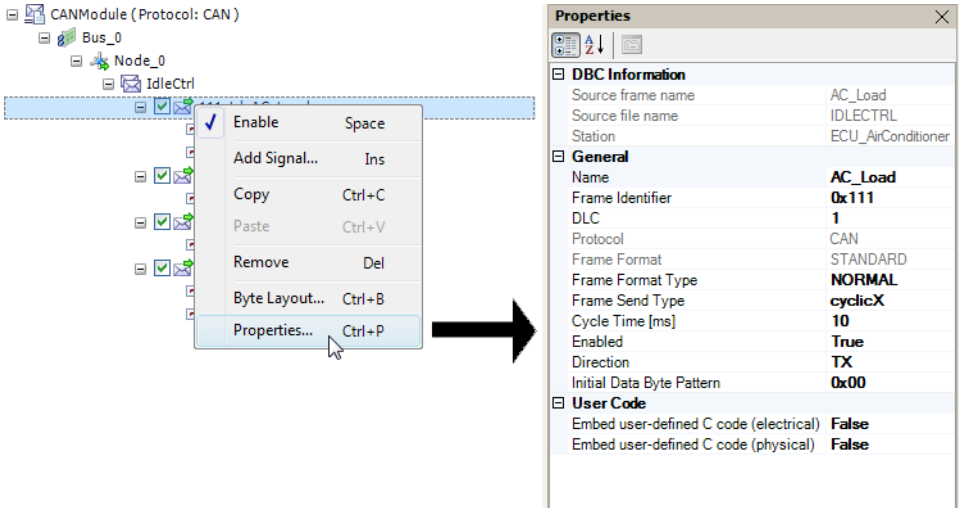
Darstellung von Eigenschaften („Properties“)

Durch Markieren der Elemente in den Baumansichten werden die Eigenschaften des markierten Elements im Fenster „Properties“ sichtbar gemacht. Diejenigen Eigenschaften, die editierbar sind, können dann direkt im Eigenschaftenfenster bearbeitet werden.

Properties anzeigen

Wenn die Eigenschaften nicht automatisch angezeigt werden, gehen Sie wie folgt vor:

- Rechtsklicken Sie das Element, dessen Eigenschaften Sie anzeigen wollen.
- Wählen Sie aus dem Kontextmenü **Properties**.



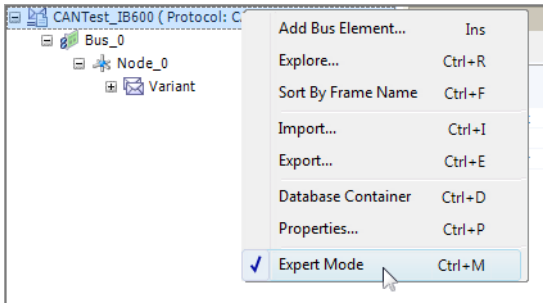
Die Eigenschaften des ausgewählten Elementes werden im Fenster „Properties“ angezeigt.

Nähere Informationen zum Bearbeiten von Eigenschaften finden Sie im Abschnitt „Bearbeiten des CAN-Netzwerks“ auf Seite 99.

Die Option „Expert Mode“

Wenn die Option „Expert Mode“ gewählt ist, werden selten verwendete Eigenschaften angezeigt, die ansonsten verborgen werden.

- Um diese Option zu aktivieren/deaktiviren, rechtsklicken Sie auf das Root-Element des CAN-Netzwerkes und klicken Sie **Expert Mode**.



Diese Eigenschaften sind bei den jeweiligen Elementen des CAN-Netzwerkes beschrieben (siehe „Die Bestandteile eines CAN-Netzwerks“ auf Seite 107).

Darstellung des Byte-Layouts von Frames

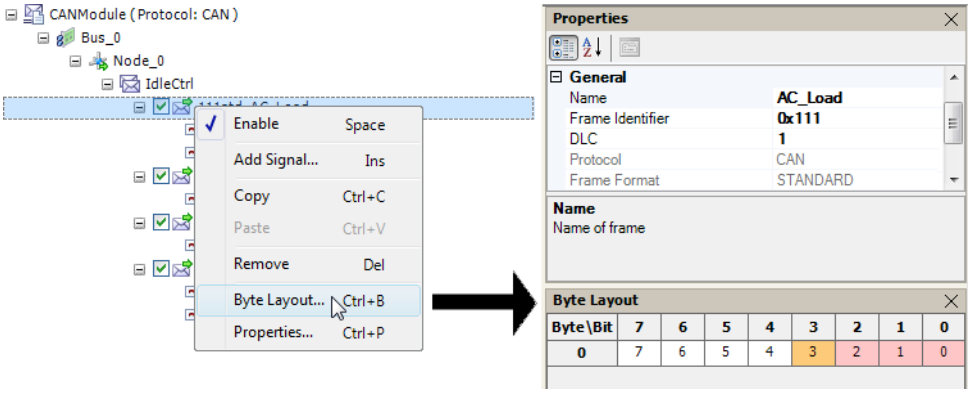
In diesem Bereich wird der Inhalt eines in der Netzwerkansicht ausgewählten Frames dargestellt.

Byte-Layout anzeigen

Um das Byte-Layout eines Frames anzuzeigen, gehen Sie wie folgt vor:

- Rechtsklicken Sie den entsprechenden Frame, dessen Layout Sie anzeigen wollen.
- Wählen Sie aus dem Kontextmenü **Byte Layout**.

  Das Byte-Layout des Frames wird angezeigt.



3.5.4 Bearbeiten des CAN-Netzwerks

Die Bearbeitung des CAN-Netzwerkes erfolgt ausschließlich über das Kontextmenu des jeweiligen Elementes. Welche Elemente ein solches Netzwerk enthält, ist im Abschnitt „Die Bestandteile eines CAN-Netzwerks“ auf Seite 107 beschrieben.

Zum Bearbeiten des Netzwerkes gibt es eine Reihe von Einträgen im Kontextmenü der jeweiligen Elemente, von denen einige auch auf eine Mehrfachauswahl (von gleichartigen Elementen) angewandt werden können.

Elemente hinzufügen

- Markieren Sie das übergeordnete Element und wählen Sie im Kontextmenü **Add (Element)**.

Hinweis

Namen von Parts, Frames und Signalen dürfen nur Zeichen enthalten, die für ANSI-C Identifier erlaubt sind.

Elemente entfernen

- Markieren Sie das Element und wählen Sie im Kontextmenü **Remove**.

Elemente kopieren

- Markieren Sie das Quellelement und wählen Sie im Kontextmenü **Copy**.
- Markieren Sie das übergeordnete Zielelement und wählen Sie im Kontextmenü **Paste**.

Eigenschaften von Elementen kopieren

- Markieren Sie das Quellelement und wählen Sie im Kontextmenü **Copy**.
- Markieren Sie das gewünschte Zielelement und wählen Sie im Kontextmenü **Paste**.

Elemente verschieben

- Markieren Sie das Quellelement.
- Ziehen Sie das markierte Quellelement mit dem Mauszeiger in das übergeordnete Zielelement und lassen Sie den Mauszeiger los.

Hinweis

Nicht immer sind alle oben beschriebenen Aktionen erlaubt – in diesen Fällen ist der entsprechende Eintrag des Kontextmenüs entweder deaktiviert oder nicht vorhanden.

Bearbeiten von Eigenschaften

Die Eigenschaften eines Elementes des CAN-Netzwerkes können im Feld „Properties“ bearbeitet werden. Einige Eigenschaften einiger Elemente können nicht bearbeitet werden, da sie automatisch berechnet werden. Diese Eigenschaften werden im Fenster „Properties“ ausgegraut dargestellt.

Eigenschaftenfenster anzeigen

- Markieren Sie ein Element und wählen Sie im Kontextmenü **Properties**.

  Das Feld „Properties“ wird im CAN-Editor angezeigt.

Eigenschaften mehrerer Elemente bearbeiten

Werden mehrere Elementen des gleichen Typs im Netzwerk markiert, werden diejenigen Eigenschaften angezeigt, die allen gewählten Elementen gemeinsam sind.

- Öffnen Sie die Anzeige von „Properties“.
- Wählen Sie die gewünschten Elemente mit der Maus bei gleichzeitig gedrücker STRG-Taste.

Hinweis

Wird beim Bearbeiten mehrerer Elemente eine Eigenschaft geändert, die das Element innerhalb seines übergeordneten Elements identifiziert (z.B. ein Signalname innerhalb eines Frames), kann dies zu Fehlermeldungen führen, falls die Eigenschaft eines weiteren Elementes des gleichen übergeordneten Elements ebenfalls diesen Wert erhält.

Validierung

Vor dem Speichern des CAN-Moduls wird eine Validierung des CAN-Netzwerks durchgeführt. Dabei werden u.a. die für die Codegenerierung relevanten Aspekte der CAN-Spezifikation geprüft. Fehlermeldungen werden im Logfenster ausgegeben.

Hinweis

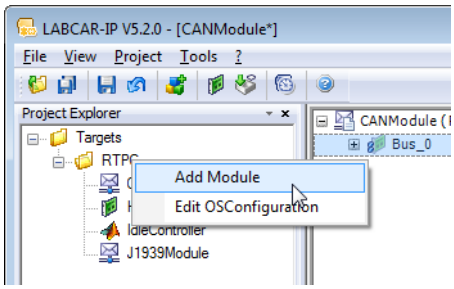
Werden die Fehlermeldungen nicht beachtet, kann dies während des Build-Vorgangs des Projektes zu Compilerfehlern oder zu unerwünschtem Laufzeitverhalten führen.

3.5.5 Erstellen eines CAN-Netzwerkes

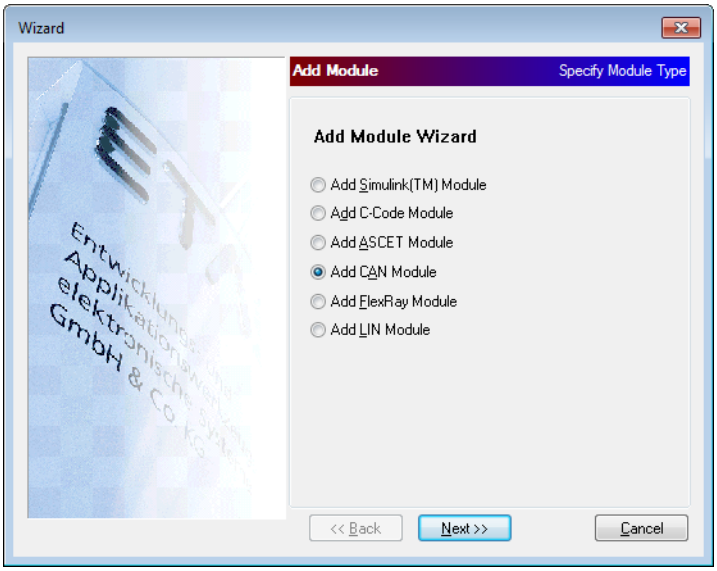
In diesem Abschnitt wird beschrieben, wie Sie ein CAN-Netzwerk durch Import einer CANdb-Datei erstellen können.

CAN-Modul hinzufügen

- Wählen Sie im Project Explorer das Target „RTPC“ und drücken Sie die rechte Maustaste.
- Wählen Sie im Kontextmenü **Add Module**.

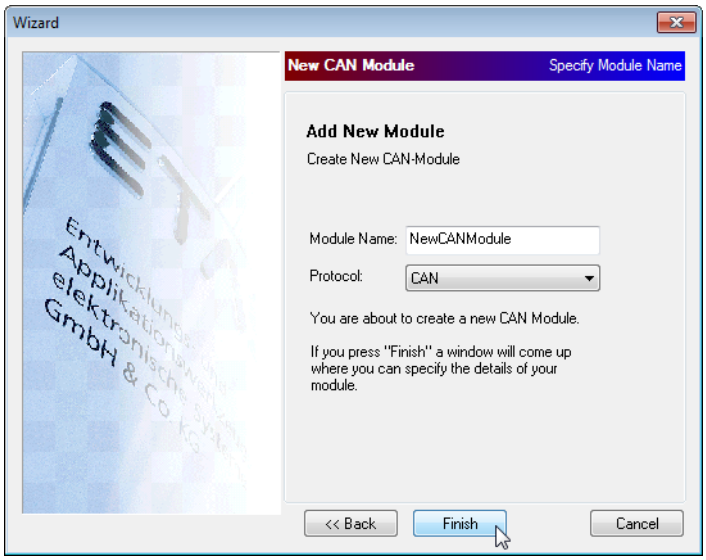


Der „Add Module“ Wizard wird geöffnet.



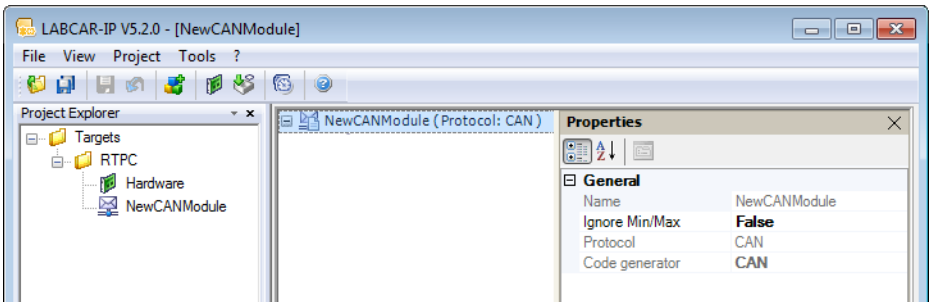
- Wählen Sie **Add CAN Module** und klicken Sie **Next**.

- Geben Sie einen Namen für das Modul ein und wählen Sie das Protokoll („CAN“ oder „J1939“).



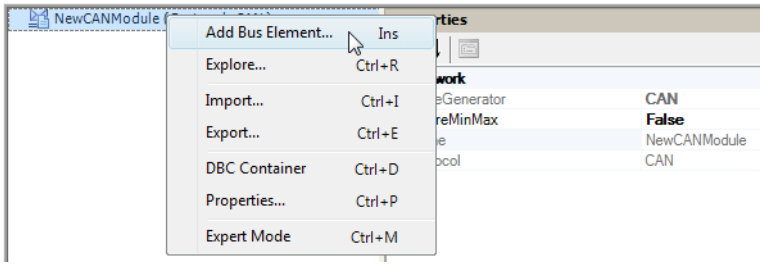
- Klicken Sie **Finish**.

  Das neue CAN Modul wird erstellt und im Editor dargestellt.

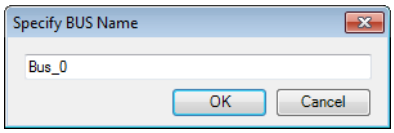


Bus hinzufügen

- Wählen Sie das CAN.Modul und rechtsklicken Sie.
- Wählen Sie **Add Bus Element**.



- Vergeben Sie einen Namen für das Bus-Element und klicken Sie **OK**.



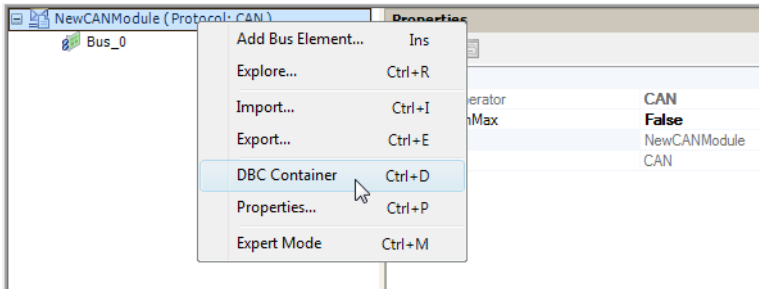
Das Bus-Element wird erstellt.

- Wählen Sie **File** → **Save All**.

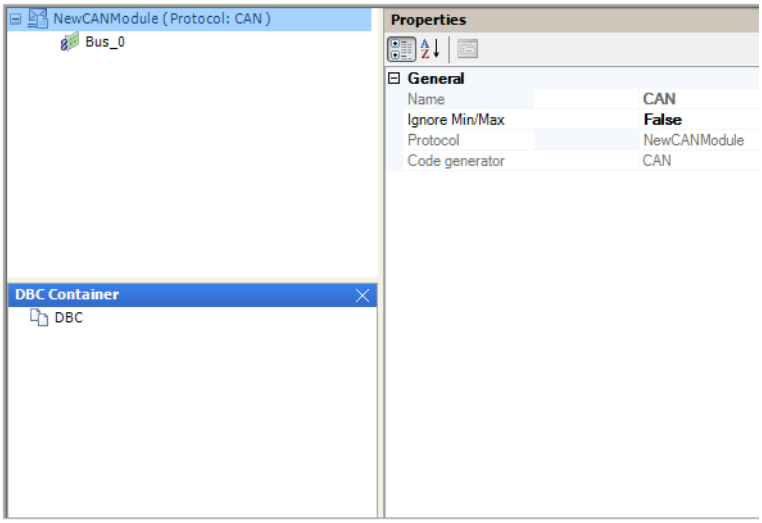
Sie können jetzt ihr CAN-Modul manuell aufbauen (siehe „Bearbeiten des CAN-Netzwerks“ auf Seite 99 und „Die Bestandteile eines CAN-Netzwerks“ auf Seite 107) oder eine DBC-Datei importieren.

DBC Container öffnen

- Wählen Sie das CAN Modul.
- Wählen Sie im Kontextmenü **DBC Container**.

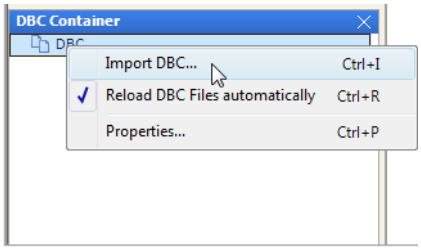


Der Bereich „DBC Container“ wird geöffnet.



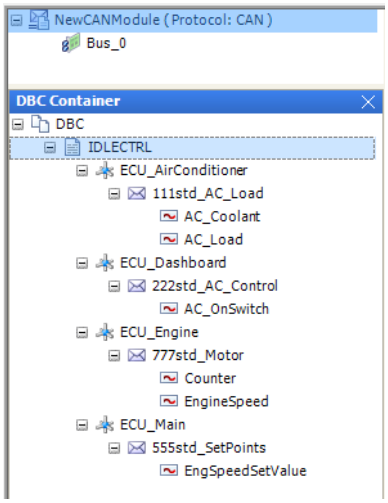
DBC-Datei laden

- Wählen Sie im DBC Container das Element „DBC“.
- Wählen Sie im Kontextmenü **Import DBC**.



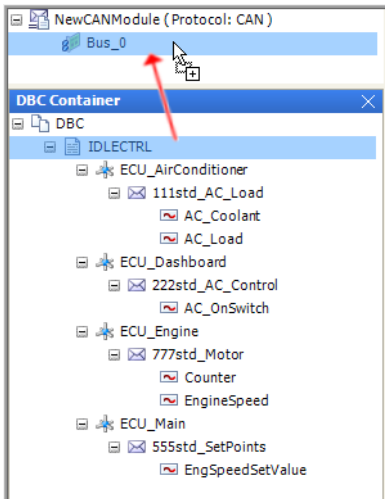
- Wählen Sie im Dateiauswahlfenster die DBC-Datei und klicken Sie **OK**.

  Die in der Datei enthaltenen Informationen werden in den DBC Container importiert.



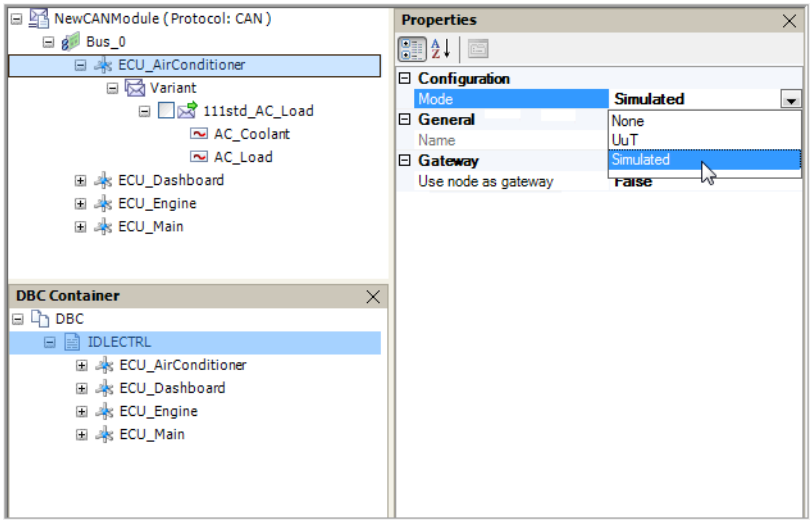
DBC zu Bus hinzufügen

- Wählen Sie die importierte DBC und verschieben Sie diese mit gedrückter linker Maustaste auf das Bus-Element.



Die Knoten der DBC-Datei werden zum Bus hinzugefügt. Zusätzlich wird jeweils zwischen Knoten und Frame ein Part („Variant“) eingefügt.

- Wählen Sie die einzelnen Knoten und definieren Sie im Fenster „Properties“ (Configuration/Mode), ob der Knoten real vorhanden ist („UuT“) oder simuliert werden soll („Simulated“).



Drag & Drop in CAN-Konfiguration

Wenn Elemente (Knoten, Frames oder Signale) in einer CAN-Konfiguration bereits vorhanden sind, gibt es zwei Möglichkeiten:

1. Versionierung

   Bereits vorhanden Elemente werden mit einer angehängten Versionsnummer in die Konfiguration übernommen.

   – **DBC File → Bus**

     Wenn gleichnamiger Knoten bereits vorhanden ist

   – **Knoten → Bus**

     Wenn gleichnamiger Knoten bereits vorhanden ist

   – **Frame → Part**

     Wenn gleichnamiger Frame bereits vorhanden ist

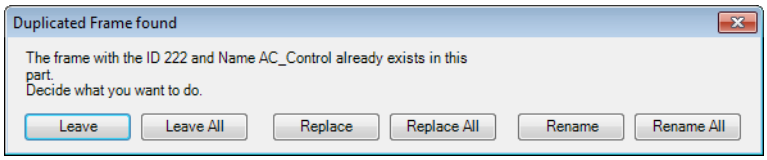
   – **Signals → Frames**

     Wenn gleichnamiges Signal bereits vorhanden ist

2. Abfrage

   – **Knoten → Knoten/Part**

     Frames des Ursprungsknotens, die im Zielknoten/Zielpart nicht vorhanden sind, werden dorthin kopiert – sind identische[1] Frames vorhanden, erfolgt für jeden Frame eine Abfrage:



Folgende Vorgehensweisen stehen zur Auswahl:

- **Leave**

  Der Import des jeweiligen Frames wird übersprungen (mit **Leave All** werden alle weiteren bereits vorhandenen Frames übersprungen).

- **Replace**

  Der bereits vorhandene Frame wird durch den zu importierenden ersetzt (mit **Replace All** werden alle weiteren bereits vorhandenen Frames ersetzt).

- **Rename**

  Die bereits vorhandene Message wird mit einer Versionsnummer versehen und importiert (mit **Rename All** alle weiteren bereits vorhandenen Frames mit Versionsnummmern versehen und importiert).

[1] Frames sind identisch, wenn ID/PGN **und** Namen übereinstimmen

3.5.6 Die Bestandteile eines CAN-Netzwerks

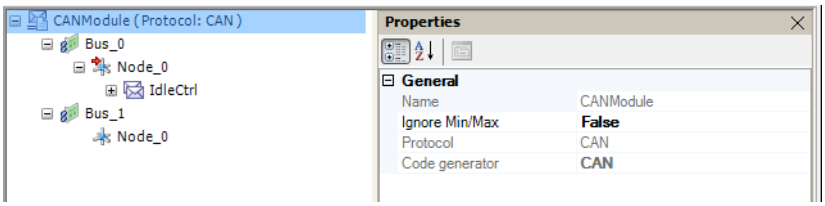
Im Folgenden finden Sie eine Beschreibung der Bestandteile des CAN-Netzwerkes in LABCAR-IP.

Im Einzelnen sind dies:

- „CAN-Netzwerk“ auf Seite 107
- „Bus“ auf Seite 108
- „Nodes“ auf Seite 113
- „Parts“ auf Seite 117
- „Frames“ auf Seite 119
- „Signale“ auf Seite 125

CAN-Netzwerk

Die globalen Eigenschaften des CAN-Netzwerkes werden bei dessen Auswahl im Fenster „Properties“ angezeigt.



- **Name**

  Name des CAN-Moduls

- **Ignore Min/Max**

  Damit können vorgegebene Beschränkungen von Signalwerten (siehe „Min, Max“ auf Seite 127) ignoriert werden.

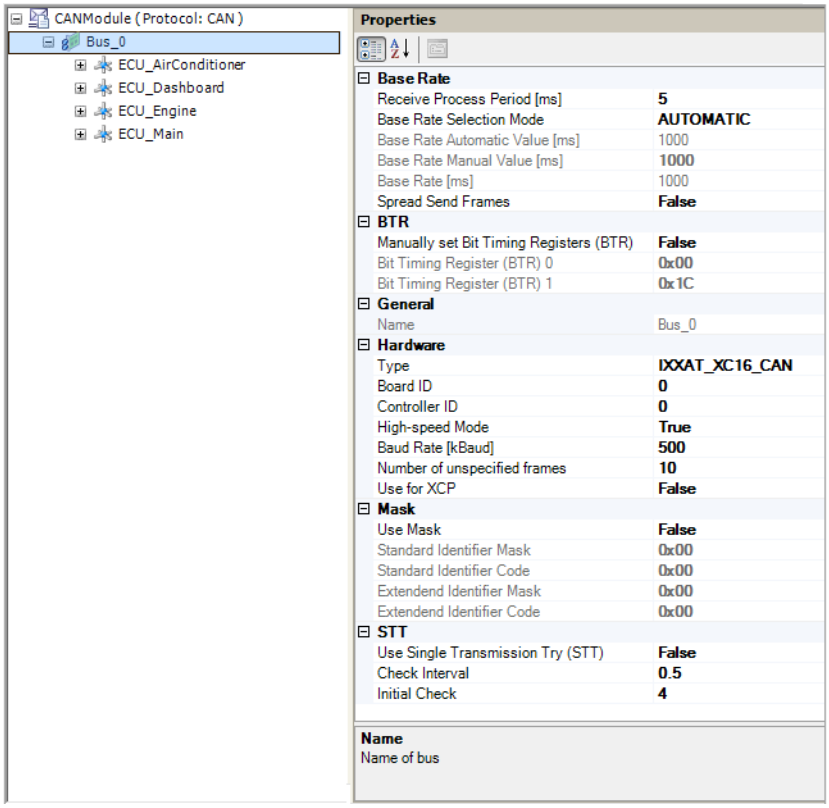
- **Protocol**

  CAN oder J1939 (mit Network Management)

- **Code generator**

  Hier kann der Code-Generator gewählt werden

Bus



Base Rate:

Die Eigenschaft „Base Rate“ eines Controllers definiert die Rate, mit der die Kommunikation zwischen dem LABCAR-OPERATOR-Projekt und dem ausgewählten Controller des CAN-Moduls erfolgt.

Alle CAN Send-Frames werden in der schnellsten Task t gerechnet (z.B. 1 ms). Die aktuelle Übertragungstask eines Frames (cycle time) wird bei ganzzahligen Vielfachen dieser „Grundtask“ (n*t) getriggert.

Gegeben seien z.B. drei Send-Frames mit jeweils 5 ms, 10 ms und 2 ms Zykluszeit. Die Grundtask muss daher ein Zeitraster (= Base Rate) haben, in dem sich alle diese Frames darstellen lassen, d.h. der größte gemeinsame Teiler dieser Zykluszeiten (hier: 1 ms).

Wenn die Zykluszeit eines Frames beispielsweise während eines laufenden Experimentes dahingehend geändert werden soll, dass sie nicht zur automatisch berechneten Base Rate kompatibel ist, kann diese auch manuell angepasst werden.

Bei der automatischen Berechnung („AUTOMATIC“) wird der kleinste gemeinsame Teiler aller Zykluszeiten als Periode angenommen - die Base Rate. Mit den Optionen „AUTOMATIC2x“/„AUTOMATIC4x“ wird diese automatisch ermittelte Rate noch einmal durch 2 bzw. 4. geteilt. Dies hilft insbesondere bei der Entzerrung der Sendezeitpunkte mit der Option „Spread Send Frames“.

- **Receive Process Period [ms]**

  Die Rate, mit der der Receive-Prozess des des Busses zyklisch aufgerufen wird.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

- **Base Rate Selection Mode**

  „Base Rate Selection Mode“ gibt an, wie „Base Rate“ bestimmt werden soll. „Base Rate“ kann

  – manuell („MANUAL“)

  oder

  – automatisch („AUTOMATIC“)

  berechnet werden. Bei automatischer Berechnung kann auch eine zwei- bzw. vierfache Übertaktung gewählt werden.

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

  „Base Rate Automatic Value“ wird aus dem größten gemeinsamen Teiler der Zykluszeiten aller aktivierten Frames unterhalb des gewählten Controllers berechnet.

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

  „Base Rate Manual Value“ ist die durch den Anwender manuell eingegebene gewünschte Rate.

- **Base Rate [ms]**

  Die aktuelle Base Rate

- **Spread Send Frames**

  Da der Sendepuffer des CAN-Boards nur eine begrenzte Anzahl von Frames speichern kann, kann es unter Umständen zum Verlust von Send-Frames kommen, wenn Zykluszeiten so gewählt wurden, dass es zu bestimmten Zeiten zu Häufungen von zu sendenden Frames kommt.

  Wenn z.B. 5 Frames im 10 ms-Raster; 5 im 20 ms-Raster und 5 im 100 ms-Raster zu senden sind, kann das zu Häufungen bei n*20 ms (n = 0,1,2,...) und insbesondere bei n*100 ms führen (siehe Abb. 3-11). Die Option

„Spread Send Frames“ verhindert solche Häufungen (und damit eventuelle Datenverluste), indem einzelne Frames (beim erstmaligen Senden) mit einer geringen Verzögerung gegenüber anderen gesendet werden.

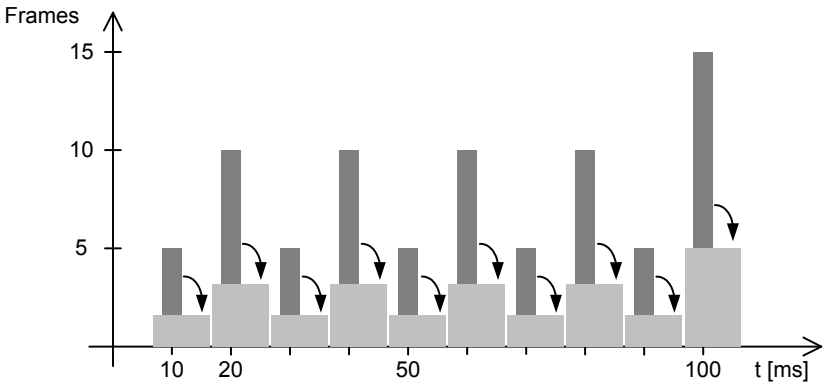


Abb. 3-11 Wirkungsweise der Option „Spread Send Frames“ (siehe Text)

BTR:

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

  Bei Auswahl dieser Option kann direkt auf die Bit Timing Registers (BTRs) des Controllers zugegriffen werden. Die Auswahl „Baud Rate [kBaud]“ wird dann deaktiviert.
- **Bit Timing Register (BTR) 0**

  Bit Timing Register 0
- **Bit Timing Register (BTR) 1**

  Bit Timing Register 1

Hinweis

Detaillierte Informationen zum Inhalt dieser Register finden Sie im Datenblatt des CAN-Controllers.

CAN-FD - Arbitration Bit Rate:

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

  Die Arbitrierungs-Bitrate des Controllers in kBaud (Standard: 500)
- **Arbitration Timing Mode**

  Der Timing-Modus der Arbitrierungs-Bitrate
- **Time Segment 1**

  Dauer des Zeitsegments 1 (in TIME QUANTA)
- **Time Segment 2**

  Dauer des Zeitsegments 2 (in TIME QUANTA)
- **Re-synchronisation Jump Width**

  Resynchronisation Jump Width (in TIME QUANTA)
- **Transceiver Delay Compensation**

  Transceiver Delay Compensation Offset (in TIME QUANTA, 0 = deaktiviert)

CAN-FD - Fast Data Bit Rate:

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

  Die Fast Data Rate des Controllers in kBaud (Standard: 2000)
- **Arbitration Timing Mode**

  Der Timing-Modus der Arbitrierungs-Bitrate
- **Time Segment 1**

  Dauer des Zeitsegments 1 (in TIME QUANTA)
- **Time Segment 2**

  Dauer des Zeitsegments 2 (in TIME QUANTA)
- **Re-synchronisation Jump Width**

  Resynchronisation Jump Width (in TIME QUANTA)
- **Transceiver Delay Compensation**

  Transceiver Delay Compensation Offset (in TIME QUANTA, 0 = deaktiviert)

General:

- **Name**

  Busname

Hardware:

- **Type**

  Typ des verwendeten CAN-Boards („IXXAT_XC16_CAN“, „IXXAT_IB200_CAN“ oder „IXXAT_IB600_CANFD“)
- **Board ID / Controller ID**

  Eindeutige Identifier des Boards und der Controller auf diesem.
- **High-speed Mode**

  Diese Option definiert den CAN-Controller als High-Speed-Schnittstelle.
- **Baud Rate [kBaud]**

  Hier kann die Übertragungsrate des CAN-Controllers festgelegt werden.
- **Number of unspecified frames**

  Zahl der nicht bereits im Experiment spezifizierten Frames, die empfangen werden können. Für diese werden dann in LABCAR-EE im Register „Workspace Elements“ die entsprechenden Messgrößen angelegt.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.
- **Use for XCP**

  Wenn Sie einen Controller von der CAN-Restbussimulation ausschließen wollen (z.B. zur Verwendung im XCP-Kontext), wählen Sie für diese Option „TRUE“.
- **ISO Mode**

  Wenn die verwendete Hardware CAN FD unterstützt („IXXAT_IB600_CANFD“), kann hier gewählt werden, ob das ISO-zertifizierte Protokoll („true“) oder das Original-Bosch-Protokoll („false“) zur Anwendung kommen soll.

Mask:

- **Use Mask**

  Hier kann ausgewählt werden, ob Frames maskiert werden sollen. Ist diese Option aktiviert, können sowohl für „Standard"- als auch für „Extended"-Identifier Masken und Code angegeben werden.

Hinweis

Detaillierte Informationen zu Inhalt und zur Verarbeitung der Acceptance Code Registers und der Acceptance Mask Registers finden Sie im Datenblatt des CAN-Controllers.

SST:

Hier können verschiedene Parameter zur Fehlerbehandlung konfiguriert werden.

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

  Aktiviert/deaktivert die Betriebsart „Single Transmission Mode" des Controllers. In dieser Betriebsart wird ein Send-Frame nach einem Übertragungsfehler (fehlende Quittierung) nicht mehr in die Warteschlange gestellt.

  Diese Betriebsart ist dann von Nutzen, wenn der Controller nicht mit einem vollständigen CAN-Netzwerk verbunden ist (wenn z.B. ein angeschlossenes Steuergerät zeitweise ohne Stromversorgung ist).

  Üblicherweise stellt der Controller alle Frames solange in eine Warteschlange, bis diese korrekt versendet wurden (einschließlich Quittierung). Wird ein zuvor spannungsloses Steuergerät wieder mit Strom versorgt, so führt dies zu einer plötzlichen Häufung von CAN-Frames, da alle 50 Frames im Sendepuffer des Controllers sehr schnell nacheinander gesendet werden, was zu einem Verlust des Timings zwischen diesen Frames führen kann.

- **Check Interval**

  Damit wird ein Zeitintervall (als Fließkommazahl in s) festgelegt, nach dessen Verstreichen der Status des CAN-Busses periodisch überprüft und – falls der Bus in einem regelwidrigen Zustand ist – neu gestartet wird.

  Wird dieses Intervall zu 0 gesetzt, ist die Überprüfung deaktiviert.

- **Initial Check**

  Legt eine Anzahl von Intervallen fest, die zur Feststellung einer gescheiterten Initialisierung verwendet werden – das Minimum beträgt 2.

  Die Defaulteinstellung beträgt 4, wobei die Überprüfung mit der durch „Check Interval" definierten Frequenz durchgeführt wird. In diesem Fall wird ein Fehler dann signalisiert, wenn der Zustand „Initialisierung" auch nach 4 Überprüfungen noch vorliegt.

Weitere Properties für das J1939-Protokoll

Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

DTC:

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

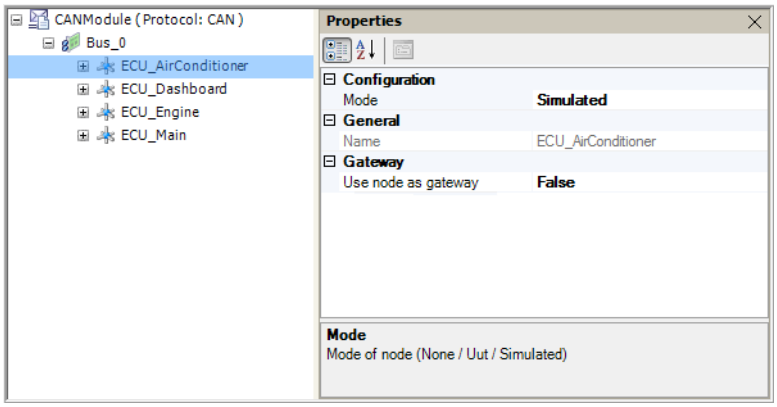
  Ein J1939-Knoten kann auch DTCs behandeln - hiermit wird die Anzahl der pufferbaren DTCs spezifiziert. Diese Puffer gibt es nur für dynamisch hinzugefügte Knoten (siehe „Number of unspecified nodes“ auf Seite 113):

  Da dafür viel Speicher benötigt wird, beträgt die maximale Anzahl 50.

- **Number of unspecified nodes**

  Die maximale Anzahl an dynmisch hinzugefügten Knoten, die sich per Address Claiming im CAN-Netzwerk bekannt machen können.

Nodes



Configuration:

- **Mode**

  Konfiguration des Knotens (siehe Abb. 3-7 auf Seite 92):

  – **Simulated**

    Der Knoten wird simuliert

  – **UuT**

    Der Knoten ist real

  – **None**

    None bedeutet, dass der Knoten neutral agiert: Er wird entweder vollständig benutzerdefiniert gesteuert oder ist von der Konfiguration der anderen Knoten fremdbestimmt.

General:

- **Name**

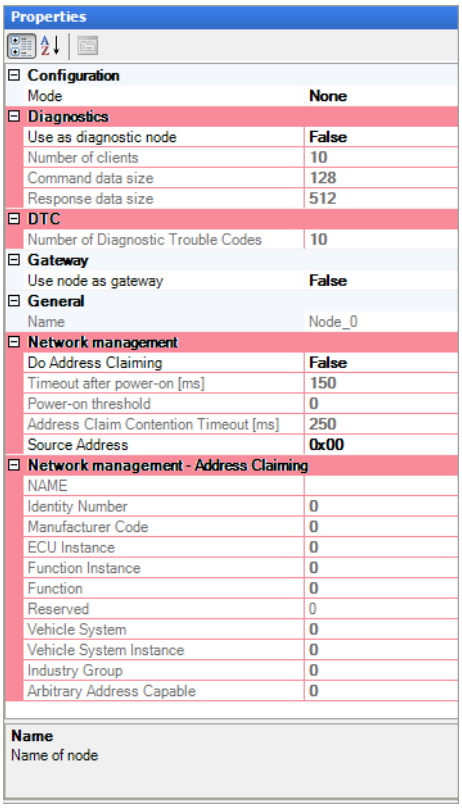
  Name des Knotens

Gateway:

- **Use node as gateway**

  Damit kann der Knoten als Gateway konfiguriert werden.

Weitere Properties für das J1939-Protokoll



Diagnostics:

- **Use as diagnostic node**

  Konfiguration als „Diagnostic Node“

> **Hinweis**
>
> *Pro Bus kann es nur **einen** „Diagnostic Node“ geben!*

Die folgenden drei Optionen sind nur aktiv, wenn „Use as diagnostic node“ gewählt wurde.

- **Number of clients**

  Die Anzahl der Clients in ETAS EE und LABCAR-AUTOMATION mit diagnostischer Funktionalität. Dabei sind immer zwei reserviert für LABCAR-AUTOMATION.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

- **Command data size**

  Diagnostische Steuerbefehle werden immer als Array übertragen. Dieser Wert legt die Größe des Arrays und damit die Anzahl der Befehle fest. Da Befehle unterschiedliche Datenmengen enthalten können, variert damit auch die Anzahl der möglichen Befehle.

  Die Größe des Arrays sollte nur dann erhöht werden, wenn wirklich mehr Befehle zur gleichen Zeit benötigt werden.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

- **Response data size**

  Diagnostische Antworten werden immer als Array übertragen. Dieser Wert legt die Größe des Arrays und damit den Umfang der speicherbaren Antwortdaten fest.

  Da der Umfang der benötigten Antwortdaten von den jeweiligen Befehlen abhängt, sollte die Größe des Arrays sollte nur dann erhöht werden, wenn nicht alle Antwortdaten zu den jeweiligen Befehlen empfangen werden können.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

DTC:

- **Number of Diagnostic Trouble Codes**

  Hiermit wird die Anzahl der pufferbaren DTCs für diesen Knoten spezifiziert.

  Nur sichtbar, wenn die Option **Expert Mode** (siehe „Die Option „Expert Mode“ “ auf Seite 97) aktiviert wurde.

Network management:

- **Do Address Claiming**

  Aktivierung/Deaktivierung des „Address Claiming“ [1]

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

  Die Zeit (in ms), die der simulierte Knoten vor dem Senden des Address Claims warten muss. Dient zur Simulation des Power On Self Tests (POST).

- **Power On Threshold**

  Simulierte Knoten haben einen Eingang(Inport, der mit diesem Schwellwert verglichen wird. Sobald der Schwellwert überschritten wird, wird das Address Claiming durchgeführt.

1. Zur Zeit wird nur der im Dokument „SAE J1939-81, Appendix D, Figure D2 - State Transition Diagram for Initialization of Single Address CAs“ beschriebene Address-Claiming-Prozess unterstützt.

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

  Die Zeit (in ms), die der Knoten nach dem Abschicken seiner Address-Claim-Botschaft auf „Widerspruch“ anderer Knoten wartet. Wenn in dieser Zeit kein anderer Knoten dieselbe Adresse beansprucht, dann nimmt der simulierte Knoten an, dass er seine gewünschte Adresse bekommen hat und verwendet diese dann.

- **Source Address**

  Die Source Address

Network management - Address Claiming:

- **NAME**

  Der Identifier des Knotens (Steuergeräts).

  „NAME“ setzt sich zusammen aus den folgenden zehn Feldern:

- **Identity Number**

  21 Bit (vom Steuergerätehersteller definert)

- **Manufacturer Code**

  11 Bit zur Identifikation des Steuergeräteherstellers

- **ECU Instance**

  3 Bit zu Identifikation einer Steuergeräteinstanz (wenn z.B. mehrere ABS-Steuergeräte vorhanden sind)

- **Function Instance**

  5 Bit zur Identifikation der Funktionsinstanz (z.B. ABS #1)

- **Function**

  8 Bit zur Identifikation der Funktion (z.B. ABS)

- **Reserved**

  Reserviert für zukünftige Definition durch die SAE

- **Vehicle System**

  7 Bit zur Identifizierung des Fahrzeugsystems (z.B. Anhänger)

- **Vehicle System Instance**

  4 Bit zur Identifizierung der Instanz eines Fahrzeugsystems

- **Industry Group**

  3 Bit zur Definition des Industriezweiges (z.B. „On-Highway“, „Agricultural“)

- **Arbitrary Address Capable**

  1 Bit zur Festlegung, ob das Steuergerät – nachdem es bei einer Address-Claim-Prozedur unterlegen war – eine andere Adresse wählen kann.

Parts

Ein Part wird einem übergeordneten Knoten zugeordnet, unter einem Part werden Frames angeordnet. Parts können exportiert und importiert werden. Die Inhalte eines Parts werden innerhalb des Moduls in Dateien mit der Dateierweiterung `.llp` abgelegt.

Parts besitzen innerhalb des gesamten Netzwerkes einen eindeutigen Namen. Existiert bereits ein Part mit einem gegebenen Namen, wird der Name des hinzuzufügenden Parts mit einer fortlaufenden Zahl erweitert.

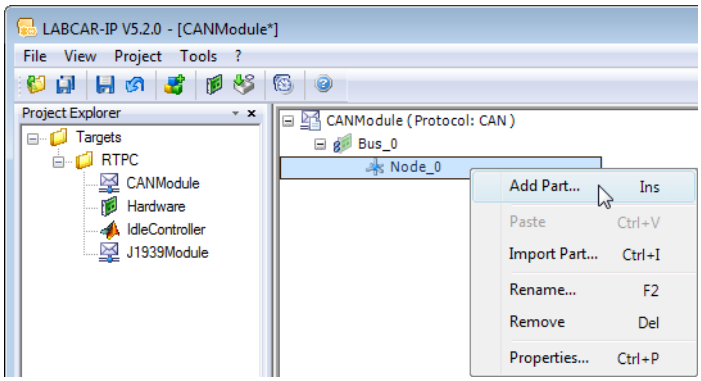
Priorisierung, Aktivierung und Deaktivierung von Parts

Parts dienen der Modellierung von Varianten eines Netzwerkknotens. Dazu können von einem Part Kopien angelegt werden einschließlich aller unter dem Part definierten Frames und Signale.

Über die Eigenschaft „Enabled“ können Parts aktiviert oder deaktiviert werden. Mit der Eigenschaft „Priority“ kann die Abarbeitung der Frames und Signale dieses Parts gesteuert werden: Parts mit höherer Priorität überschreiben in jedem Zyklus der Simulation die Ein- und Ausgaben der Parts mit niederer Priorität.

Part erstellen

- Rechtsklicken Sie den Knoten („Node“), dem der neue Part zugeordnet werden soll.
- Wählen Sie **Add Part**.

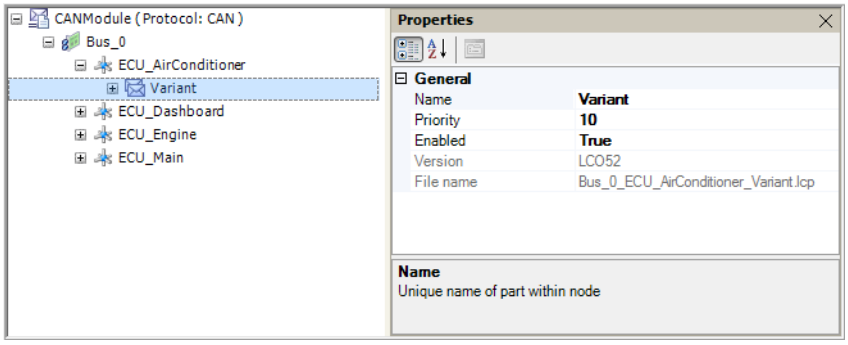


Ein Part mit dem Namen „Variant“ wird erstellt.

Hinweis

Namen von Parts, Frames und Signalen dürfen nur Zeichen enthalten, die für ANSI-C Identifier erlaubt sind.

Zuvor exportierte Parts können auch wieder importiert werden (siehe „Part importieren“ auf Seite 119).



General:

- **Name**

  Der Name des Parts

- **Priority**

  Mit dieser Eigenschaft kann die Priorität des Parts angegeben werden. Die Priorität muss einen eindeutigen Wert haben – die Ausführung der Parts erfolgt entsprechend ihrer Priorität.

- **Enabled**

  Jeder Part kann (unabhängig von den anderen Parts) zur Laufzeit des Experiments aktiviert und deaktiviert werden. Ist ein Part deaktiviert, wird die komplette Kommunikation aller Frames unterhalb des gewählten Parts unterbunden.

- **Version**

  Version des Parts

- **File name**

  Name der Datei (`Busnamen_Knotennamen_Partnamen.lcp`), die den Part beschreibt

Hinweis

Mit den Eigenschaften „Enabled“ und „Priority“ ist ein Variantenhandling während der Laufzeit des Experiments möglich.

Part exportieren

Parts können mit allen Frames und deren Eigenschaften in Form von XML-Dateien exportiert werden.

- Wählen Sie den zu exportierenden Part.
- Rechtsklicken Sie und wählen Sie **Export Part**.
- Wählen Sie im Dateiauswahlfenster ein Verzeichnis und geben Sie einen Dateinamen ein.
- Klicken Sie **OK**.

  Die Informationen des Part werden in die Datei (`*.lcp`) exportiert.

Parts löschen

- Um einen Part zu löschen, rechtsklicken Sie den Part und wählen Sie **Remove**.

Part importieren

Zuvor (s.o.) exportierte Parts können wieder in die CAN-Konfiguration importiert werden.

- Wählen Sie den Knoten, dem der Part zugeordnet werden soll.
- Wählen Sie im Kontextmenü **Import Part**.
- Wählen Sie den zu importierenden Part (`*.lcp`) im Dateiauswahlfenster und klicken Sie **OK**.

  Der Part wird mit dem Informationen aus der angegebenen Datei erstellt.

Frames

Ein Frame wird einem übergeordneten Part zugeordnet, unter einem Frame werden Signale angeordnet.

Für Frames kann neben dem Eigenschaftsfenster auch das Byte-Layout-Fenster angezeigt werden. Dieses zeigt die Belegung der Bytes in den Nutzdaten des Frames mit den Signalen.

Je nach Richtung werden die Frames mit unterschiedlichen Symbolen dargestellt:


– Send-Frame (TX)


– Receive-Frame (RX)


– Gateway-Frame (GW)

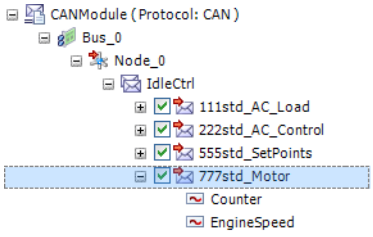
(siehe „Use node as gateway“ auf Seite 114

Hinweis

Beachten Sie bitte, dass die Bezeichnung der Richtung aus Sicht des simulierten Restbusses (siehe Abb. 3-8 auf Seite 92) erfolgt!

Byte-Layout anzeigen

- Markieren Sie den Frame und wählen Sie im Kontextmenü **Byte Layout**.



Byte Layout

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

Im Beispiel besteht der Frame aus dem 16-Bit Signal „EngineSpeed“ (hellrot) und dem 5-Bit Signal „Counter“ (ocker).

- Wenn Sie eines der „Bit-Felder“ doppelklicken, wird das Signal in der Frameliste ausgewählt und dessen Eigenschaften im Feld „Properties of Signal“ dargestellt.

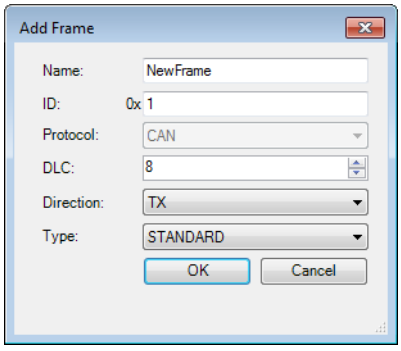
Hinweis

Treten Fehler im Byte-Layout auf, z.B. durch Überlappung zweier Signale oder fehlerhafter Angabe der Länge eines Signals oder des Frames, werden die fehlerhaften Bits in roter Farbe dargestellt.

Benutzerdefinierten Frame anlegen

Neben der Möglichkeit, einen Frame aus einer CANdb-Datei, einem CAN-Modul oder einem Part zu importieren, können auch benutzerdefinierte Frames angelegt werden.

- Markieren Sie den übergeordneten Part und wählen Sie im Kontextmenü **Add Frame**.



Hinweis

Innerhalb eines Parts muss die Kombination aus der ID des Frames und dessen Name eindeutig sein!

- Geben Sie die folgenden Daten ein (siehe „General“ auf Seite 122).

  - Name des Frames (Name)
  - Identifier (ID)
  - Protokoll (Protocol)

    Das Protokoll wurde schon bei der Erstellung des Moduls definiert (siehe „CAN-Modul hinzufügen“ auf Seite 101).
  - Größe des Frames (DLC)
  - Richtung (Direction)
  - Frametyp (Type)
- Klicken Sie **OK**.

  Die neue Frame wird hinzugefügt.

Wenn Sie bei der Konfiguration Ihres CAN-Netzwerkes mehrere CAN-Boards verwenden, beachten Sie bitte folgenden Hinweis:

Hinweis

Entfernen Sie ein Board, dem Frames zugewiesen worden sind, so erfolgt keine automatische Änderung dieser Zuweisung – diese muss per Hand durchgeführt werden!

Aktivierung und Deaktivierung von Frames

Innerhalb eines Parts dürfen mehrere Frames mit gleicher ID definiert sein, sofern diese jeweils einen unterschiedlichen Namen besitzen. Zur Codegenerierung darf jedoch in einem Part nur jeweils ein Frame mit einer gegebenen ID aktiviert sein. Dazu müssen die jeweils anderen Frames aus der Codegenerierung ausgeschlossen werden.

- **Über Baumansicht des Netzwerks**: Aktivieren bzw. deaktivieren Sie das Kästchen neben dem Framesymbol in der Baumansicht des Netzwerkes.
- **Über Kontextmenü**: Markieren Sie den Frame und aktivieren bzw. deaktivieren Sie im Kontextmenü **Enable**.
- **Über Eigenschaftsfenster**: Öffnen Sie das Eigenschaftsfenster und Bearbeiten Sie die Eigenschaft „Enabled“ („True“ oder „False“).

Einen Frame entfernen

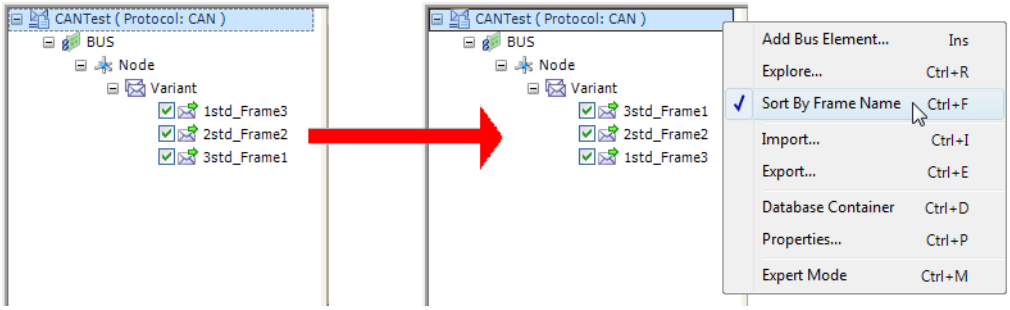
- Wählen Sie im Kontextmenü der Frameliste **Remove**.

  Der Frame wird entfernt.

Frames nach Namen ordnen

Die Frames sind normalerweise nach ihren Frame Identifiern geordnet.

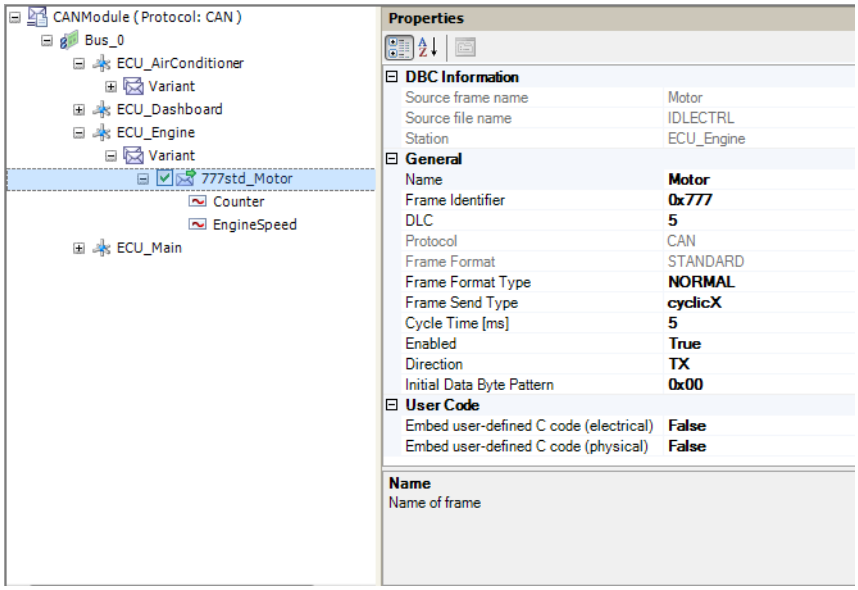
- Um die Frames nach ihrem Namen zu ordnen, rechtsklicken Sie den Modulnamen und wählen Sie die Option **Sort by Frame Name**.



Eigenschaften des Frames darstellen

- Um die Eigenschaften eines Frames darzustellen, rechtsklicken Sie diesen.
- Wählen Sie im Kontextmenü **Properties**.

  Die Eigenschaften des Frames werden im Fenster „Properties“ dargestellt.



DBC Information:

Wenn der Frame aus einer importierten CANdb-Datei stammt, werden hier die entsprechenden Informationen dargestellt.

General:

- **Name**

  Name des Frames

- **Frame Identifier**

  Identifier (hex) des Frames
- **DLC**

  Data Length Code des Frames
- **Protocol**

  Das Protokoll („CAN“ oder „J1939“)
- **Frame Format**

  Das Format des Frames („STANDARD“, „STANDARD_REMOTE“, „EXTENDED“ oder „EXTENDED_REMOTE“)
- **Frame Format Type**

  „NORMAL“ oder „REMOTE“
- **Frame Send Type**

  Den Sendemodus
  - **cyclicX**

    Der Frame wird im Raster der Cycle Time gesendet
  - **spontanX**

    Der Frame wird gesendet, wenn sich eines seiner Signale geändert hat.

Hinweis

Ein Frame mit dem Sendetyp „spontanX“ wird nur dann gesendet, wenn mindestens eines der Signale den Send Mode „On Change“ (siehe „Send Type“ auf Seite 127) besitzt!

  - **cyclicAndSpontanX**

    Der Frame wird in beiden oben beschriebenen Fällen gesendet
- **Cycle Time [ms]**

  Die Zykluszeit des Frames
- **Enabled**

  Aktivierung des Frames (siehe auch „Aktivierung und Deaktivierung von Frames“ auf Seite 121)
- **Direction**

  Die Richtung aus Sicht des Modells („Send (TX)“ oder „Receive (RX)“)
- **CAN-FD**

  Frame wird als CAN-FD-Frame („True“) oder Standard-Frame („False“) interpretiert.
- **Bit Rate Switch (BRS)**

  Legt fest, ob (während der Übertragung des CAN-FD-Frames) die erhöhte Daten-Bitrate verwendet werden soll.
- **Initial Data Byte Pattern**

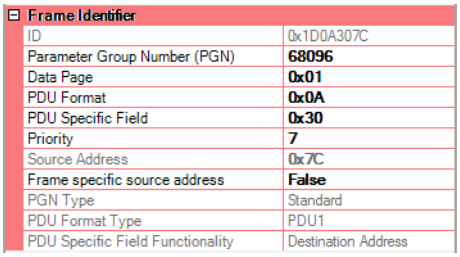
  Das Bytemuster bei Initialisierung des Frames

User Code:

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

  Hier können Sie festlegen, ob zum Frame benutzerdefinierter C-Code hinzugefügt werden soll (siehe „Benutzerdefinierter C-Code“ auf Seite 130).

Weitere Properties für das J1939-Protokoll



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**

  Die vollständige CAN ID der Message (29 Bit), bestehend aus
  - Priority (3 Bit)
  - Parameter Group Number (18 Bit)
  - Source Address (3 Bit
- **Parameter Group Number**

  Die Parameter Group Number, bestehend aus
  - Data Page (1 Bit)
  - PDU Format (8 Bit)
  - PDU Specific (8 Bit)
- **Data Page**

  Data Page Bit
- **PDU Format**

  8 Bit zur Festlegung des PDU-Format (Teil der PGN):
  - < 240: PDU Specific Field ist „Destination Address“ („PDU1“)
  - >= 240: PDU Specific Field ist „Group Extension“ („PDU2“)
- **PDU Specific Field**

  8 Bit (siehe PDU Format)
- **Priority**

  3 Bit zur Optimierung der Latenz der Übertragung auf den Bus (wird von LABCAR-OPERATOR ignoriert)
- **Source Address**

  Source Address des Knoten, zu dem der Frame gehört (8 Bit).
- **Frame specific source address**

  Wenn „TRUE“, kann die Source Address des Frames auf einen Wert gesetzt werden, der von der Adresse des Knotens abweicht.

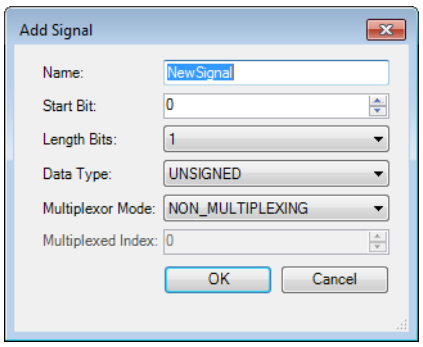
- **PGN Type**
  PGN-Typ
- **PDU Format Type**
  „PDU1“ oder „PDU2“ (siehe PDU Format)
- **PDU Specific Field Functionality**
  Funktion des „PDU Specific“ Feldes (siehe PDU Format)

Signale

Ein Signal wird einem übergeordneten Frame zugeordnet, der Name eines Signals unterhalb dieses Frames muss dabei eindeutig sein.

Benutzerdefiniertes Signal anlegen

- Markieren Sie einen Frame und wählen Sie im Kontextmenü **Add Signal**.



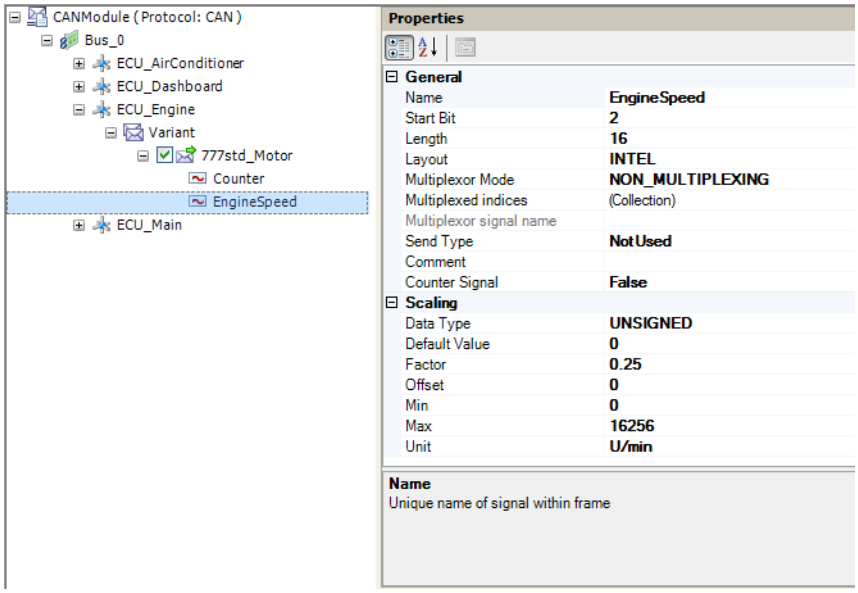
- Geben Sie die folgenden Daten ein (siehe „Scaling“ auf Seite 127 und „General“ auf Seite 126).
  - Name des Signals (Name)
  - Start Bit
  - Länge des Signals (Length Bits)
  - Datentyp (Data Type)
  - Multiplexor Mode
  - Multiplexed Index
- Klicken Sie **OK**.
  Das neue Signal wird zum Frame hinzugefügt.

Ein Signal entfernen

- Wählen Sie im Kontextmenü des Signals **Remove**.
  Das Signal wird entfernt.

Eigenschaften des Signals darstellen

- Um die Eigenschaften eines Signals darzustellen, rechtsklicken Sie dieses.
- Wählen Sie im Kontextmenü **Properties**.



Die Eigenschaften des Signals werden im Fenster „Properties“ dargestellt. Hier werden die Eigenschaften eines gewählten Signals dargestellt und können hier – sofern zulässig – auch editiert werden.

General:

- **Name**

  Name des Signals
- **Start Bit**

  Erstes Bit des Signals
- **Length**

  Signallänge in Bit
- **Layout**

  Byte-Reihenfolge: „INTEL“ (Little-Endian-Format) oder „MOTOROLA“ (Big-Endian-Format)
- **Multiplexor Mode**

  Charakterisierung des Frames
  - NON_MULTIPLEXING

    Charakterisiert einen gewöhnlichen (= nicht-multiplex) Frame
  - MULTIPLEXOR

    Das Signal ist das Steuersignal eines Multiplex-Frames

– MULTIPLEXED_SIGNAL

Die Bedeutung der in diesem Signal übertragenen Daten wird vom aktuellen Wert des Steuersignals bestimmt.

- **Multiplexed indices**

  Kodierung für die Zuordnung bei Multiplex-Botschaften („MULTIPLEXED_SIGNAL“)

  Klicken von  öffnet den Collection Editor, mit dem mehrere Wertzuweisungen vorgenommen werden können.

- **Multiplexor signal name**

  Name des Multiplexersignals, das entscheidet, ob das Signal verwendet wird oder nicht.

- **Send Type**

  Legt fest, wodurch das Senden eines Frames ausgelöst wird:

  – NotUsed

  Signal wird bei der Berechnung des Sendetriggers nicht berücksichtigt

  – OnChange

  Signal wird bei der Berechnung des Sendetriggers berücksichtigt

- **Comment**

  Ein Kommentar zur Beschreibung des Signals

- **Counter Signal**

  Siehe „Ein Signal als Zähler verwenden“ auf Seite 129.

Scaling:

- **Data Type**

  Typ: „UNSIGNED“, „SIGNED“, „IEEE32“ oder „IEEE64“

- **Default Value**

  Ein Signal eines Send-Frames kann mit einem Defaultwert versehen werden – wenn das Signal im Connection Manager mit einem anderen Signal verbunden ist, wird dieser Wert ignoriert.

  Der Nutzen des Defaultwertes liegt beim Import/Export von CAN-Modulen, da der Defaultwert dann auch bei der Wiedernutzung des CAN-Moduls in einem anderen LABCAR-OPERATOR-Projekt zur Verfügung steht.

- **Factor, Offset**

  Ein Faktor a und ein Offset b zur Skalierung des Signals

- **Min, Max**

  Hier können Grenzen für eine Plausibilitätsprüfung angegeben werden (siehe „Min/Max-Werte“ auf Seite 128)

  Wenn Sie das Signal rechtsklicken und **Limit Min/Max to bit length** wählen, wird „Min/Max“ automatisch aus „Data Type“ und „Length“ berechnet.

- **Unit**

  Die physikalische Einheit des Signals

Min/Max-Werte

Beim Speichern einer CAN-Testmatrix wird immer überprüft, ob Min < Max gilt. Falls nicht, kommt es zu einer Warnung, es sei denn, die Einstellung „Ignore Min/Max“ (siehe „Ignore Min/Max“ auf Seite 107) ist aktiviert.

Hinweis

Beachten Sie bitte, dass es zwei Überprüfungen des physikalischen Wertes gibt und der schließlich per CAN übertragene Wert nicht dem ursprünglichen Wert entspricht, wenn dieser den Prüfbedingungen nicht standhält.

Diese Überprüfungen sind:

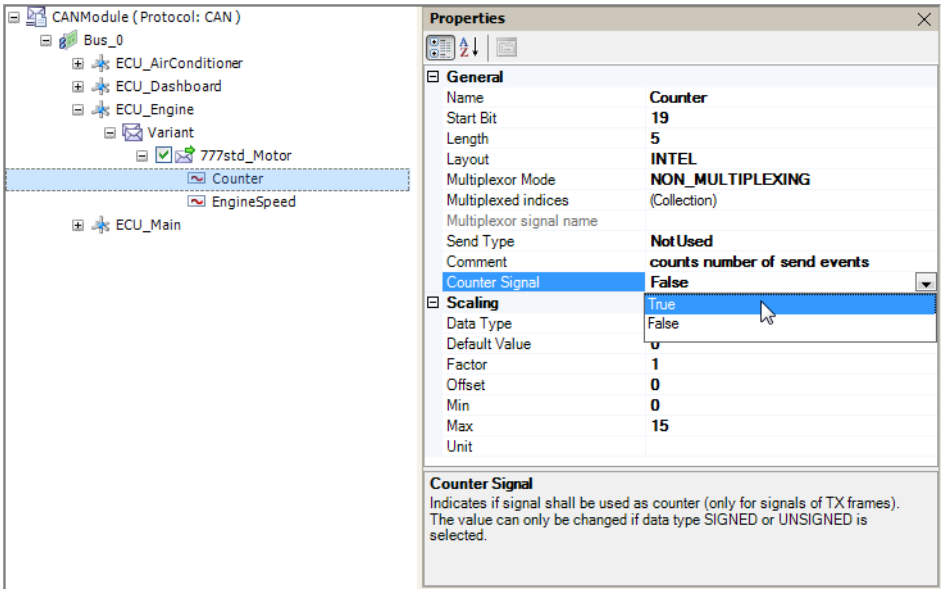
1. Gilt min ≤ phys. Wert ≤ max?
   Falls der Wert diese Bedingung nicht erfüllt, wird der Wert auf min bzw. max gesetzt.
2. Nach der Umrechnung des physikalischen Wertes in den Hex-Wert erfolgt eine Überprüfung dahingehend, ob der Wert mit der Größe des Datenbereichs (unter Berücksichtigung des Datentyps) verträglich ist. Sind beispielsweise 6 Bit für ein „signed integer“ allokiert, so muss der umgerechnete Wert folgende Bedingung erfüllen: -32 ≤ Wert ≤ +31.

   Bei Nichterfüllung der Bedingung erfolgt auch hier eine Änderung des Wertes auf -32 oder +31.

Ein Signal als Zähler verwenden

Sie können jeweils ein Signal eines Send-Frames auf einfache Weise zu einem Zähler machen:

- Wählen Sie das Signal und rechtsklicken Sie.
- Ändern Sie die Eigenschaft „Counter Signal“ zu „True“.



Dieses Signal wird jetzt als Zähler verwendet, was auch an dem „123“ Symbol vor dem Symbol für das Signal erkennbar ist.

Hinweis

Die Option „Use as Counter“ ist nur für ganzzahlige Datentypen verfügbar.

- Diese Eigenschaft können Sie rückgängig machen, indem Sie diese Option wieder abwählen.

Hinweis

Die Min/Max-Werte in den Signaleigenschaften gelten auch für das Counter-Signal!

3.5.7 Benutzerdefinierter C-Code

LABCAR-NIC ermöglicht dem Anwender, zum automatisch erzeugten Code einer Message weiteren benutzerdefinierten Code hinzuzufügen, um den Inhalt der Message zu erweitern oder zu manipulieren. Dies gilt sowohl für den „physikalischen“ Teil (die Signale) wie auch für das schließlich generierte Byte-Array.

Zur Verdeutlichung der Zugriffsmöglichkeiten mittels benutzerdefiniertem Code ist die Situation im Folgenden noch einmal illustriert.

Signalfluss einer Send-Message und Eingriffsmöglichkeiten

In Abb. 3-12 ist zur Übersicht der gesamte Signalfluss von Send-Messages von Modell bis zum CANbus dargestellt.

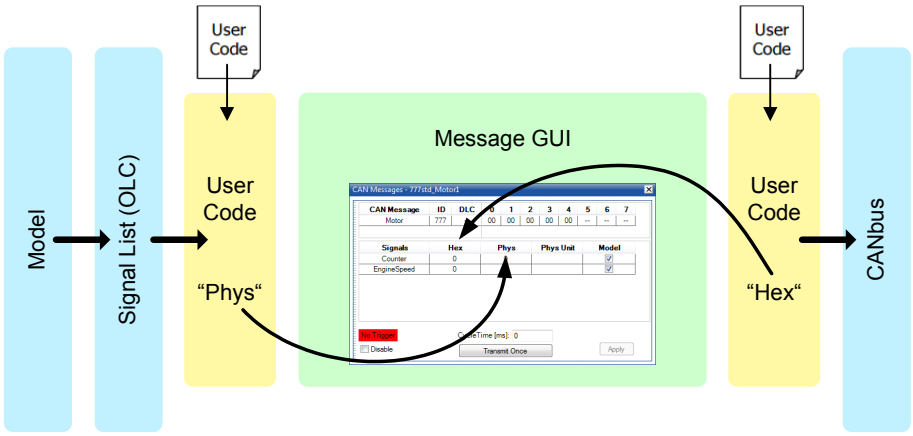


Abb. 3-12 Signalfluss bei Send-Messages (siehe Text)

Neben den Manipulationsmöglichkeiten in der Signal List besteht sowohl die Möglichkeit, über die beschriebenen Message-GUIs (siehe „Instrumentierung für die CAN-Simulation“ auf Seite 135) als auch durch benutzerdefinierten C-Code Einfluss (siehe folgenden Abschnitt) auf den Inhalt der Messages zu nehmen.

Hinweis

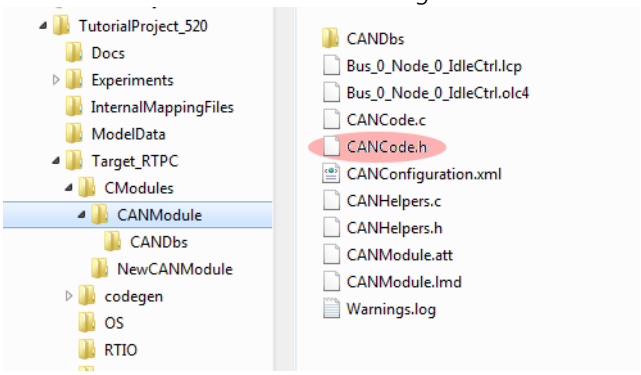
Beachten Sie, dass durch weiter rechts befindliche Eingriffe der ggf. zuvor geänderte Code wieder überschrieben wird!

Entsprechend sind die Verhältnisse bei Receive-Messages: Hier werden durch weiter links getätigte Eingriffe vorherige Änderungen überschrieben.

Im Folgenden finden Sie Code-Beispiele und die Beschreibung, wie Sie diesen Code zum Message-Code hinzufügen können.

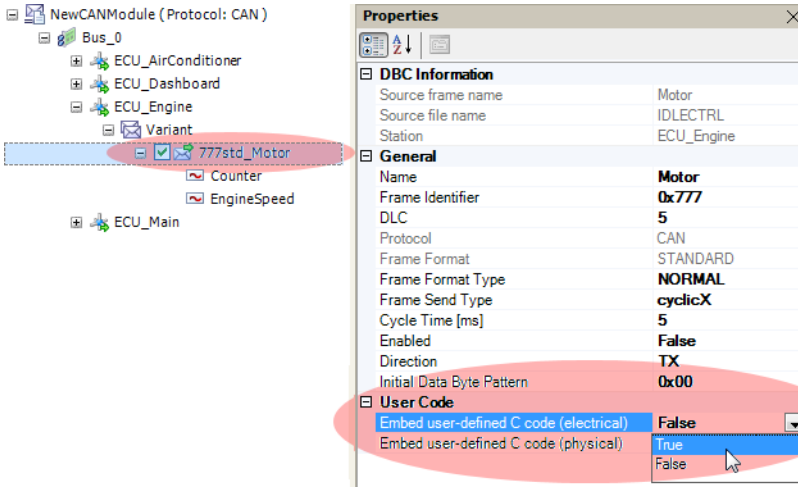
Vorgehensweise

Nach dem Speichern der CAN-Konfiguration befindet sich im unten gezeigten Verzeichnis die Header-Datei `CANCode.h`, die Funktionsdeklarationen und die Deklaration der Strukturen für die Signale enthält.



Benutzerdefinierten Code hinzufügen

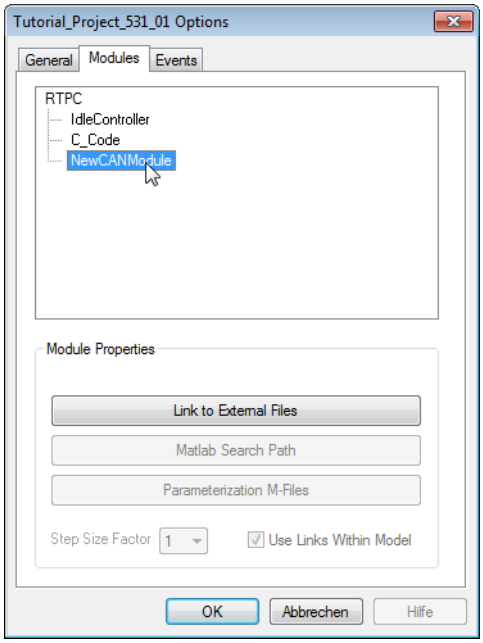
- Wenn Sie eigenen Code hinzufügen wollen, setzen Sie „Embed user-defined C code (electrical)“ oder „Embed user-defined C code (physical)“ auf „True“.



- Damit die Änderungen wirksam werden, klicken Sie **Save**.
- Öffnen Sie die Datei `CANCode.h` mit einem geeigneten Editor.

  In der Datei befinden finden sich jetzt die Funktionsdeklarationen für die oben bearbeiteten Frames. Mit Hilfe dieser Funktionen können Sie auf die physikalischen Signale als auch auf das Byte-Array der Message zugreifen. Desweiteren enthält `CANCode.h` auch die Strukturen für die jeweiligen Signale.

- Fügen Sie Ihren C-Dateien ein „`#include CANCode.h`“ hinzu, damit die Funktions- und Strukturdeklarationen bekannt sind.
- Wählen Sie im Hauptmenü **Project** → **Options** und die Registerkarte „Modules“.



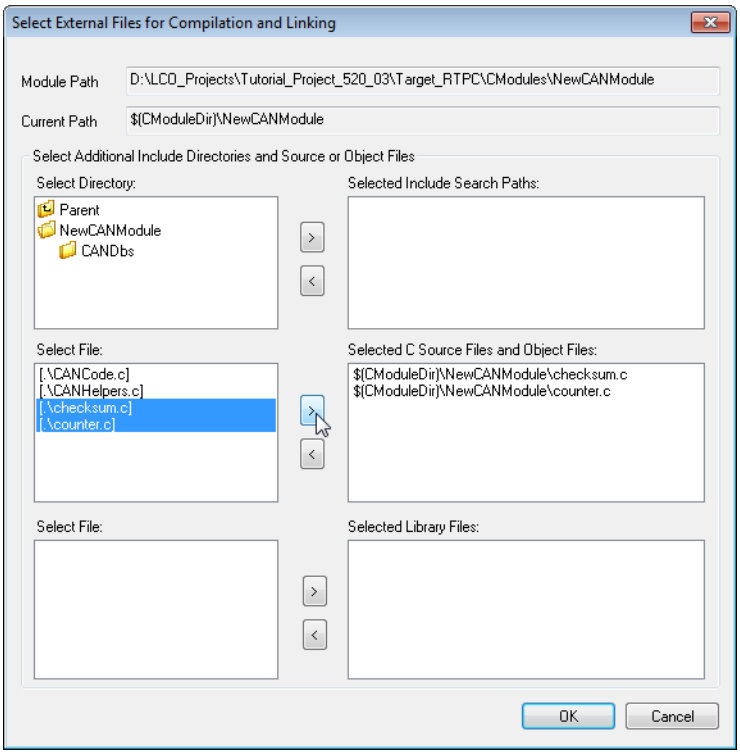
- Wählen Sie das CAN-Modul und klicken Sie die Schaltfläche **Link to External Files**.

  Das Fenster „Select External Files for Compilation and Linking“ wird geöffnet.
- Wählen Sie die einzubinde(n) Quelldatei(en) (siehe „Beispiele“ auf Seite 134) und klicken Sie die Schaltfläche **>** (Mehrfachauswahl ist möglich).

Hinweis

Sie können nur C-Dateien auswählen, die im Projektverzeichnis unter `\Target_RTPC\CModules\CAN_NamederCANdb-Datei` *oder in einem Unterverzeichnis abgelegt wurden.*

- Die gewählte(n) Datei(en) werden im rechten Fenster angezeigt.



- Klicken Sie **OK**.

  Diese Dateien werden bei zukünftigen Codegenerierungen immer mitkompiliert und mitgelinkt.
- Verlassen Sie das Register „Modules“ mit **OK**.

Beispiele

Im Folgenden finden Sie zwei Beispiele für benutzerdefinierten 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,
   NewCANModule_Bus_0_ECU_Engine_777std_Motor_PhysValues_t* signal-
   Structure )
   {
      static int counter = 0;
      signalStructure -> Signal_Counter = counter;
      (counter <= 14) ? (counter++): (counter = 0);
   }
```

Hinweis

Das Beispiel „counter.c" funktioniert nur bei zyklisch gesendeten Frames (Frame Send Type = cyclicX, siehe „Frame Send Type" auf Seite 123).

Berechnung einer Checksumme (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_
   777std_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 Instrumentierung für die CAN-Simulation

Wenn mit LABCAR-NIC V5.4.1 CAN-Restbussimulation durchgeführt wird, gibt es in ETAS EE spezielle Instrumente für CAN-Messages und einen sogenannten „CAN Bus Monitor“.

3.6.1 CAN-Module im Fenster „Workspace Elements“

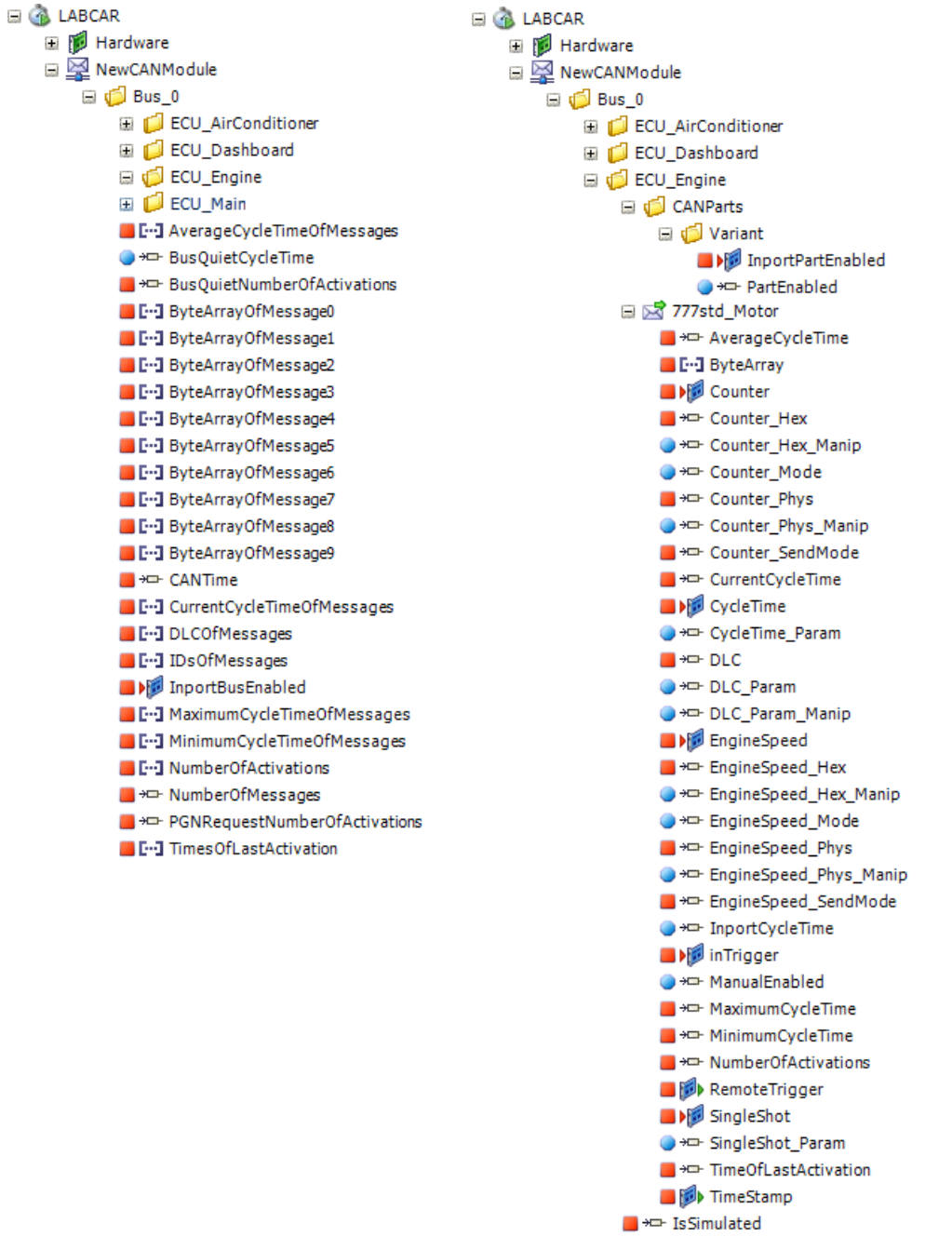


Abb. 3-13 Das CAN-Modul in den Workspace Elements

Das CAN-Modul wird in den Workspace Elements mit der folgenden Hierarchie dargestellt.

Bus

In diesem Ordner befinden sich Ordner für die Knoten und folgende Messgrößen und Parameter (siehe Abb. 3-13 auf Seite 135, linker Teil):

- **AverageCycleTimeOfMessages**

  Array mit n Elementen, die die durchschnittliche Zykluszeiten der n unspezifizierten Messages enthalten.

- **BusQuietCycleTime**

  Die Übertragungsrate der BusQuiet (DM13) Message

- **BusQuietNumberOfActivations**

  Anzahl der Übertragungen der DM13 Message

- **ByteArrayOfMessage*n***

  Arrays, über die die unspezifizierten Messages zugänglich sind. Die Anzahl n dieser Messages wird im CAN-Editor von LABCAR-IP (**Options**) unter „No. of Unspecified Messages“ definiert.

- **CANTime**

  Die Zeitskala für die CAN-Restbussimulation – deren Nullpunkt wird durch die erste Ausführung des CAN-Sendeprozesses festgelegt. Standardmässig wird die CAN-Restbussimulation in ETAS EE zusammen mit dem Experiment selbst gestartet.

- **CurrentCycleTimeOfMessages**

  Array mit n Elementen, die die aktuellen Zykluszeiten der n unspezifizierten Messages enthalten.

- **DLCOfMessages**

  Array mit n Elementen, die die DLCs der n unspezifizierten Messages enthalten.

- **IDsOfMessages**

  Array mit n Elementen, die die IDs der n unspezifizierten Messages enthalten.

- **InportBusEnabled**

  Mit diesem Inport kann der Bus vom Modell aus aktiviert/deaktiviert werden.

  – - 0.5 < Wert des Inports < + 0.5 = FALSE: Bus deaktiviert

  – sonst TRUE (Default = 1.0): Bus aktiviert

- **MaximumCycleTimeOfMessages**

  Array mit n Elementen, die die maximalen Zykluszeiten der n unspezifizierten Messages enthalten.

- **MinimumCycleTimeOfMessages**

  Array mit n Elementen, die die minimalen Zykluszeiten der n unspezifizierten Messages enthalten.

- **NumberOfActivations**

  Array mit n Elementen, die die Anzahl der Aktivierungen der n unspezifizierten Messages enthalten.
- **NumberOfMessages**

  Anzahl n der unspezifizierten Messages
- **PGNRequestNumberOfActivations**

  Anzahl der Übertragungen einer PGN Request Message
- **TimesOfLastActivations**

  Array mit n Elementen, die die Zeitpunkte der letzten Aktivierung der n unspezifizierten Messages enthalten.

Knoten

Die Ordner für die Knoten enthalten den Ordner „CANParts“ mit den Varianten und die Messages dieses Knotens.

- **InportPartEnabled**

  Mit diesem Inport kann der Part vom Modell aus aktiviert/deaktiviert werden.
  - - 0.5 < Wert des Inports < + 0.5 = FALSE (Default = 0): Part deaktiviert
  - sonst TRUE: Part aktiviert
- **PartEnabled**

  Mit diesem Parameter kann dieser Part zur Laufzeit aktiviert und deaktiviert werden.

Messages

Die einzelnen Messages besitzen folgende Messgrößen und Parameter (* nur bei Send-Messages):

- ***Signalname***

  Das Hardwaresignal
- ***Signalname*_Hex**

  Der Hex-Wert eines Signals
- ***Signalname*_Hex_Manip***

  Parameter für den geänderten Hex-Wert eines Signals
- ***Signalname*_Mode**

  Parameter für den Modus eines Signals. Folgende Werte stehen zur Verfügung:
  - Der Wert kommt aus dem Modell (0)
  - Der physikalische Wert ist geändert (1)
  - Der Hex-Wert ist geändert (2)
- ***Signalname*_Phys**

  Der physikalische Wert eines Signals

- ***Signalname*_Phys_Manip***

  Parameter für den geänderten physikalischen Wert eines Signals

- ***Signalname*_SendMode**

  Messwert, der den gewählten „Send Type“ enthält (siehe „Send Type“ auf Seite 127): 0 = NotUsed, 1 = OnChange.

- **AverageCycleTime**

  Die durchschnittliche Zykluszeit

- **ByteArray**

  Das Byte-Array der Message

- **CurrentCycleTime**

  Die aktuell gemessene Cycle Time

- **CycleTime***

  Inport zum Einspeisen einer Zykluszeit (siehe „InportCycleTime“ auf Seite 138)

- **CycleTime_Param***

  Parameter zur Festlegung der Cycle Time

- **DLC**

  Der Data Length Code der Message

- **DLC_param**

  Parameter für die Spezifikation eines vom Anwender definierten DLC für diese Message

- **DLC_param_manip**

  Boolscher Parameter, mit dem der vom Anwender vorgegebene DLC aktiviert wird (1) oder der im CAN-Editor von LABCAR-IP definierte Wert (0).

- **InportCycleTime**

  Bestimmt, ob die Cycle Time vom Parameter „CycleTime_Param“ definiert ist (= False) oder vom Inport „CycleTime“ stimuliert werden kann (= True).

- **InTrigger***

  Aktiviert oder deaktiviert das Senden der Message (siehe „Aktivierung und Deaktivierung von Send-Messages“ auf Seite 140). Ein Wert = 0 bedeutet „FALSE“, ansonsten ist InputTrigger = „TRUE“.

- **ManualEnabled***

  Parameter zur manuellen Aktivierung von Send-Messages. Entspricht der Option „disabled“ Im Message-GUI (siehe „Aktivierung und Deaktivierung von Send-Messages“ auf Seite 140).

- **MaximumCycleTime**

  Die größte gemessene Zykluszeit

- **MinimumCycleTime**

  Die kleinste gemessene Zykluszeit

- **NumberOfActivations**

  Zahl der Aktivierungen der Message

- **RemoteTrigger***

  Ein Outport, der zählt, wie oft die Message per Remote getriggered wurde.
- **InputTrigger**

  Messgröße für das Signal „InTrigger“
- **SingleShot***

  Ermöglicht das einmalige Senden einer Message. Der Parameter ist vom Typ „Boolean“ und wird, nachdem er zum Senden auf „True“ gesetzt wurde, wieder automatisch auf „False“ zurückgesetzt.
- **SingleShot_Param***

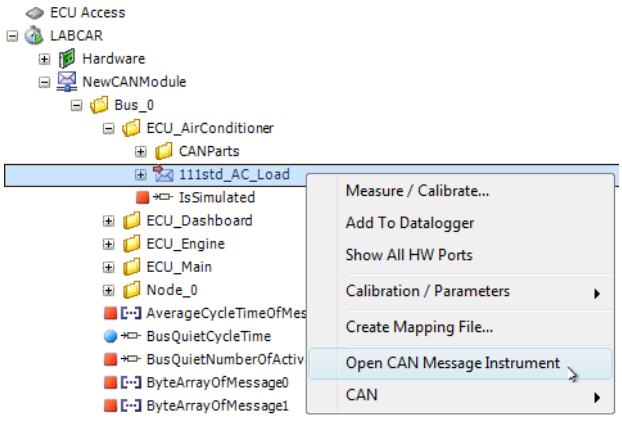
  Parameter zum Auslösen eines „Single Shot“. Entspricht der Schaltfläche **Transmit Once** in einem Message-GUI.
- **TimeOfLastActivation**

  Zeit der letzten Aktivierung der Message
- **TimeStamp**

  Der Zeitstempel, der beim Senden oder Empfang der Message aus der aktuellen CANTime (siehe „CANTime“ auf Seite 136) ermittelt wird.

3.6.2 Instrumente für CAN-Messages

Um ein Instrument zur Darstellung einer Message zu erstellen, rechtsklicken Sie die Message und wählen Sie **Open CAN Message Instrument**.



Je nach Messagetyp wird ein entsprechendes Instrument erstellt, das den Inhalt der gewählten Message darstellt.

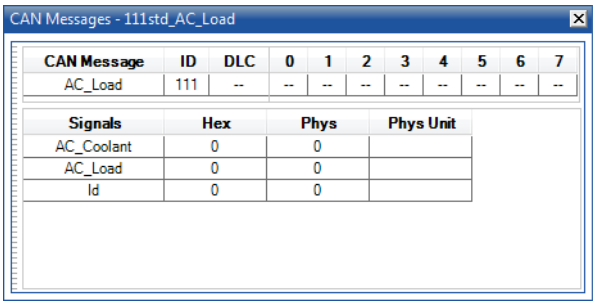


Abb. 3-14 Message-GUI für eine Receive-Message

Bei einer Receive-Message haben Sie keine Möglichkeit, Signale zu editieren - die Message und deren Signale werden lediglich dargestellt.

Bei einer Send-Message können entweder der Hex-Wert oder der physikalische Wert eines Signals (der jeweils andere Wert ist dann nicht mehr editierbar) im Bedienfenster geändert werden. Dazu muss lediglich die Option „Model“ deaktiviert werden. Mit **Apply** wird diese Änderung gültig.

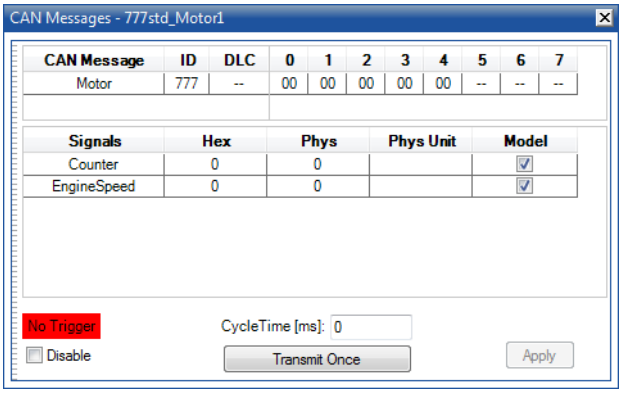


Abb. 3-15 Message-GUI für eine Send-Message

Transmit Once bewirkt ein einmaliges Senden der Message auf dem CAN-Bus (siehe „Single Shot“ auf Seite 141).

Unter „Cycle Time [ms]“ können Sie die gewünschte Zykluszeit für die Send-Message angeben.

Hinweis

Wenn Sie das Signal „CycleTime“ aus dem Modell verwenden, beachten Sie bitte die Einheit dieses Parameters (Sekunden).

3.6.3 Aktivierung und Deaktivierung von Send-Messages

Wenn die Fläche „Trigger“ grün hinterlegt ist, so bedeutet dies, dass das Signal „inTrigger“, das bei jeder Send-Message zu finden ist, den logischen Wert „true“ besitzt. Dieses Signal stammt nicht aus der CANdb-Datei, sondern wird von LABCAR-OPERATOR hinzugefügt - es dient ganz allgemein der Steuerung von Send-Messages aus dem Modell heraus.

Im Message-GUI selbst kann darüber hinaus noch insofern auf die Send-Message Einfluss genommen werden, als dass selbst eine durch das Signal „inTrigger“ freigegebene Send-Message durch Auswahl der Option „disable“ deaktivert werden kann. Genauer gesprochen wird auf das Signal „inTrigger“ und auf die Option „disable“ eine logische Und-Operation angewandt: Nur wenn „Disable = falsch“ (entspricht „Enable = wahr“) und „Trigger = wahr“ sind, wird die Message gesendet.

Single Shot

Wenn das Ergebnis dieser Und-Operation „falsch“ lautet, wird diese Message nicht Timer-gesteuert gesendet. Dann wird die Schaltfläche **Transmit Once** aktiv, die ein einmaliges Senden der Message bewirkt.

Alternativ steht für jede Send-Message (wie „inTrigger“) auch ein Signal „SingleShot“ zur Verfügung - ein Übergang 0 → 1 bei diesem Signal löst ein einmaliges Senden einer Message aus.

3.6.4 Der CAN-Monitor

Zur Beobachtung des Verkehrs auf dem CAN-Bus steht in LABCAR-OPERATOR ein Instrument namens „CAN Bus Monitor“ zur Verfügung.

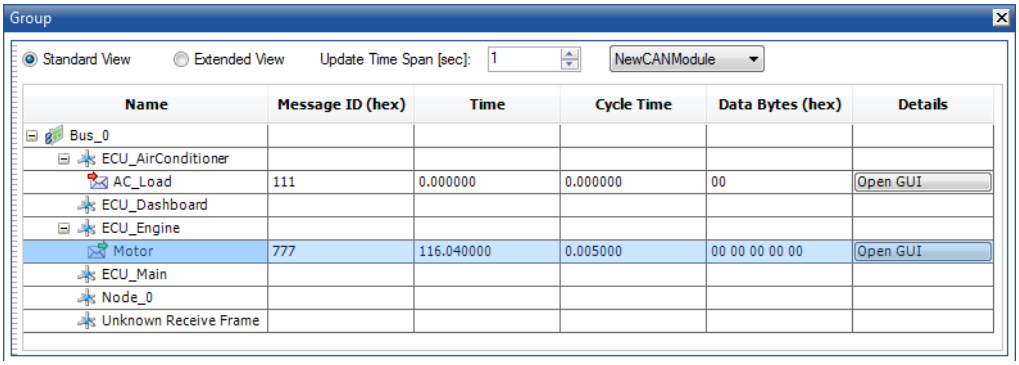


Abb. 3-16 Das Instrument „CAN Bus Monitor“

Neben dem Namen der CAN-Message, der in einer Hierarchie „Bus/Node“ dargestellt wird, enthält das Fenster folgen Spalten:

- **Message ID (hex)**

  Die ID der Message
- **Time**

  Der Zeitstempel der Message (Zeit seit Start der Simulation)
- **Cycle Time**

  Die Zykluszeit der Message
- **Data Bytes (hex)**

  Die Daten-Bytes der Message
- **Details**

  Wenn die Message innerhalb des Projektes bekannt ist, können Sie in dieser Spalte mit **Open GUI** ein Message GUI (siehe Abb. 3-14 auf Seite 140 und Abb. 3-15 auf Seite 140) öffnen.

Weitere Informationen werden angezeigt, wenn Sie statt „Standard View“ die Option „Extended View“ (oben im Fenster) wählen:

- **Message Count**

  Zahl der übertragenen Messages
- **Mean Cycle Time**

  Die durchschnittliche Zykluszeit
- **Min. Cycle Time**

  Die minimale Zykluszeit
- **Max. Cycle Time**

  Die maximale Zykluszeit

Die aufgeführten Send-Messages sind alle die, die im CAN-Editor vereinbart wurden. In den beiden Spalten rechts davon taucht ein Eintrag erst dann auf, wenn die Message erstmals gesendet wurde.

Als Receive-Messages werden alle aufgeführt, die auf dem entsprechenden Controller tatsächlich empfangen wurden. Falls so eine Message auch im CAN-Editor als Receive-Message vereinbart wurde, ist ihr Name bekannt – ansonsten wird „n/a“ angegeben.

Die Zeitspanne, nach der die dargestellte Information jeweils aktualisiert wird, lässt sich unter „Update Time Span [sec]“ festlegen.

3.7 LIN-Module (Network Integration LIN)

LABCAR-NIL (Network Integration LIN) ist ein Add-On zu LABCAR-OPERATOR. Es ermöglicht das Testen von Steuergerätefunktionen, die LIN[1]-Kommunikation nach LIN 2.0 und LIN 2.1/LIN 2.2 beinhalten.

Die Spezifikation der LIN-Bus-Kommunikation kann aus einer oder mehreren LDF[2]-Dateien eingelesen werden und zusätzlich manuell angepasst und erweitert werden. Der Anwender definiert die zu lesenden und schreibenden LIN-Frames und LABCAR-NIL erstellt automatisch Code. Zu diesem kann benutzerdefinierter Code hinzugefügt werden.

Die Signale der ausgewählten Frames sind im Connection Manager verfügbar und können dort mit Modelleingängen (Receive Frames) und Modellausgängen (Send Frames) verbunden werden.

Hardwareanforderungen

Arbeiten mit LABCAR-NIL setzt das Vorhandensein eines Real-Time PC mit einer aktuellen Version von ETAS RTPC und (mindestens) einem LIN-Board des Typs „iPC-I XC16/PCI“ (nur LIN 2.0), „CAN-IB200/PCIe“ oder „CAN-IB600/PCIe“ (LIN 2.0 und LIN 2.1/LIN 2.2) der Firma IXXAT voraus. Eine Einbindung des LIN-Boards in LABCAR-RTC (Real-Time Execution Connector) ist nicht erforderlich.

In LABCAR-IP konfigurieren Sie einen LIN-Bus mit einem LIN-Modul, das die Infrastruktur für die Simulation des kompletten LIN-Busses enthält.

Jedes LIN-Modul unterstützt bis zu 16 LIN-Boards mit jeweils einem (iPC-I XC16/PCI) LIN-Controller oder acht LIN-Boards mit jeweils zwei (CAN-IB600/PCIe) bzw. vier (CAN-IB200/PCIe) LIN-Controllern. Dabei werden in der Codegenerierung sowohl LIN-Master als auch LIN-Slave-Knoten unterstützt.

Alternativ können Sie aber auch nur einen Controller pro LIN-Modul zuordnen und den LIN-Bus zusammen mit den real vorhandenen LIN-Controllern aus mehreren verschiedenen LIN-Modulen aufbauen.

Die LIN-Boards benötigen eine externe Spannungsversorgung über den Anschluss des LIN-Controllers.

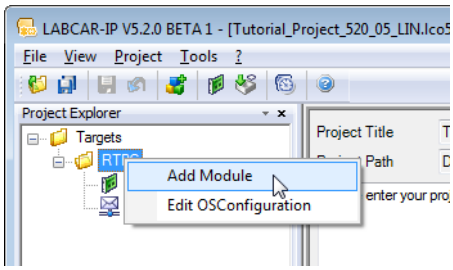
1. **L**ocal **I**nterconnect **N**etwork
2. **L**IN **D**escription **F**ile

3.7.1 Erstellen eines LIN-Moduls

Ein neues LIN-Modul wird über den bekannten Modul-Wizard erstellt.

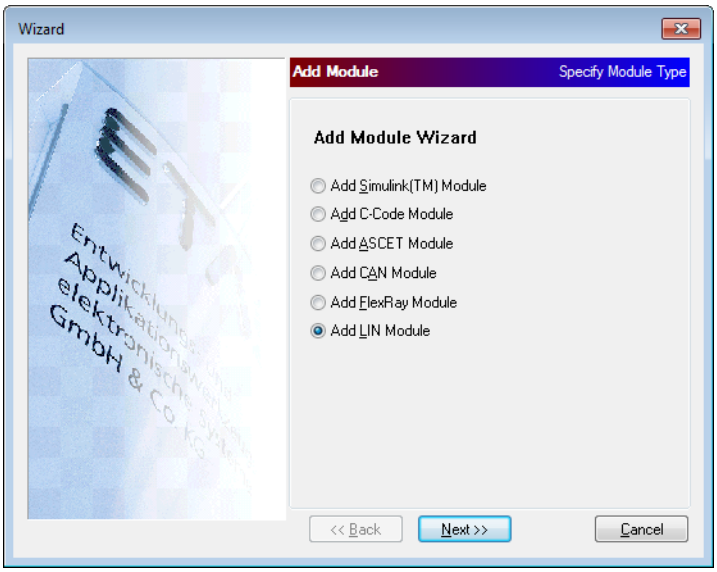
LIN-Modul erstellen

- Öffnen Sie das LABCAR-Projekt, in dem das LIN-Modul erstellt werden soll.
- Wählen Sie im Project Explorer das Target „RTPC“.
- Wählen Sie im Kontextmenü **Add Module**.



Der Modul-Wizard wird geöffnet.

- Wählen Sie die Option „Add LIN Module“.



- Klicken Sie **Next**.
- Geben Sie einen Namen für das Modul ein.
- Wählen Sie die Protokollversion
- Klicken Sie **Finish**.

Das LIN-Modul wird erstellt und im Project Explorer dargestellt.

3.7.2 Der LIN-Editor

Der LIN-Editor wird durch Doppelklick auf das LIN-Modul im Project Explorer geöffnet.

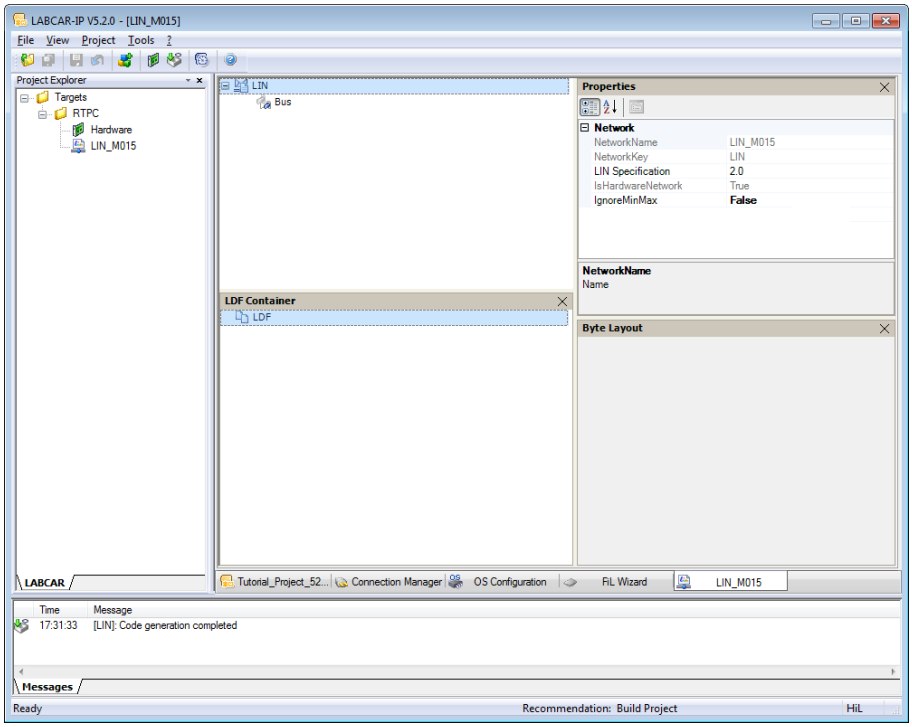


Abb. 3-17 Der LIN-Editor

Der LIN-Editor besteht aus

- der Darstellung des LIN-Netzwerkes (siehe Seite 145),
- der Darstellung der LDF-Container (siehe Seite 146),
- dem Fenster „Properties“ (siehe Seite 147)

und

- einer Anzeige für das Byte-Layout von Frames (siehe Seite 148).

Darstellung des LIN-Netzwerkes

In dieser Ansicht wird das Netzwerk mit den untergeordneten Elementen hierarchisch angezeigt.

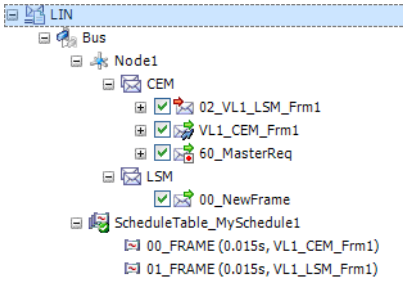


Abb. 3-18 Das LIN-Netzwerk

Informationen zum Aufbau des Netzwerkes finden Sie im Abschnitt „Bearbeiten des LIN-Netzwerks“ auf Seite 148, zu dessen einzelnen Bestandteilen in „Die Bestandteile eines LIN-Netzwerkes“ auf Seite 152.

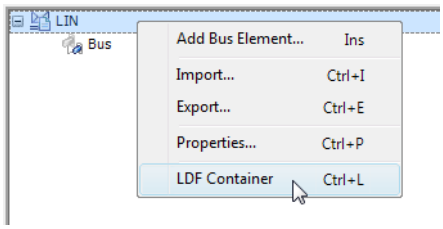
Darstellung der LDF-Container

In dieser werden die Inhalte eines oder mehrerer importierter LDF-Dateien angezeigt.

LDF-Container anzeigen

Wenn im LIN-Editor die LDF-Container nicht automatisch angezeigt werden, gehen Sie wie folgt vor:

- Rechtsklicken Sie den LIN-Netzwerkknoten.
- Wählen Sie aus dem Kontextmenü **LDF Container**.

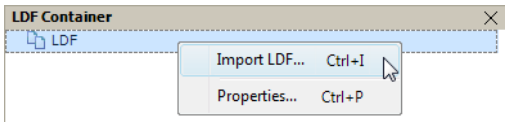


Der Bereich für die LDF-Container wird angezeigt.

Eine LDF-Datei importieren

LDF-Dateien können über das Kontextmenü des LDF-Containers importiert werden – die Elemente innerhalb des LDF-Containers sind jedoch nur lesbar. Achten Sie darauf, dass Sie die zu einem LIN-Modul passende LDF-Datei in der richtigen Protokollversion importieren.

- Rechtsklicken Sie den Container und wählen Sie **Import LDF**.



Ein Dateiauswahlfenster wird geöffnet.

- Wählen Sie die zu importierende Datei.

  Die Datei wird importiert und deren Inhalte unterhalb des LDF-Containers angezeigt.

  Dabei wird ein LDF-Container angelegt, der nach der importierten LDF-Datei benannt wird.
- Wiederholen Sie den letzten Schritt beliebig oft.
- Bereits importierte LDF-Dateien können über das Kontextmenü **Remove** wieder entfernt werden.

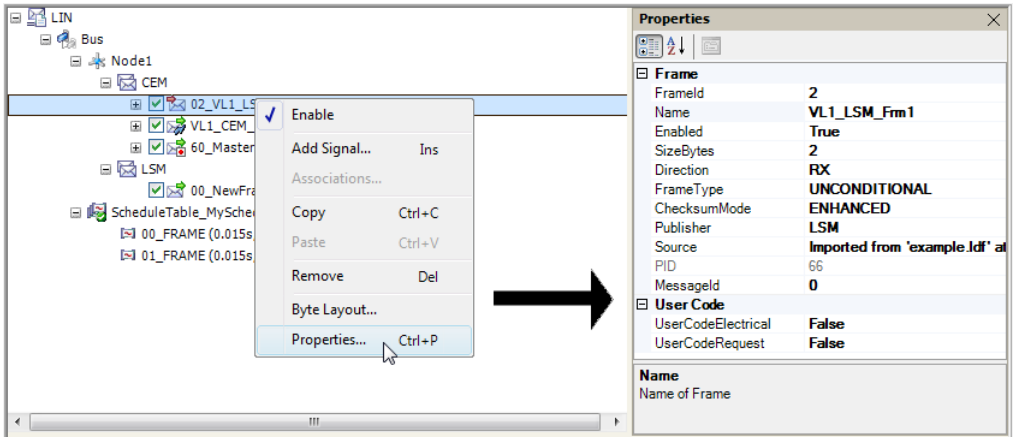
Darstellung von Eigenschaften („Properties“)

Durch Markieren der Elemente in den Baumansichten werden die Eigenschaften des markierten Elements im Fenster „Properties“ sichtbar gemacht. Diejenigen Eigenschaften, die editierbar sind, können dann direkt im Eigenschaftenfenster bearbeitet werden.

Properties anzeigen

Wenn die Eigenschaften nicht automatisch angezeigt werden, gehen Sie wie folgt vor:

- Rechtsklicken Sie das Element, dessen Eigenschaften Sie anzeigen wollen.
- Wählen Sie aus dem Kontextmenü **Properties**.



Die Eigenschaften des ausgewählten Elementes werden im Fenster „Properties“ angezeigt.

Nähere Informationen zum Bearbeiten von Eigenschaften finden Sie im Abschnitt „Bearbeiten von Eigenschaften“ auf Seite 149.

Darstellung des Byte-Layouts von Frames

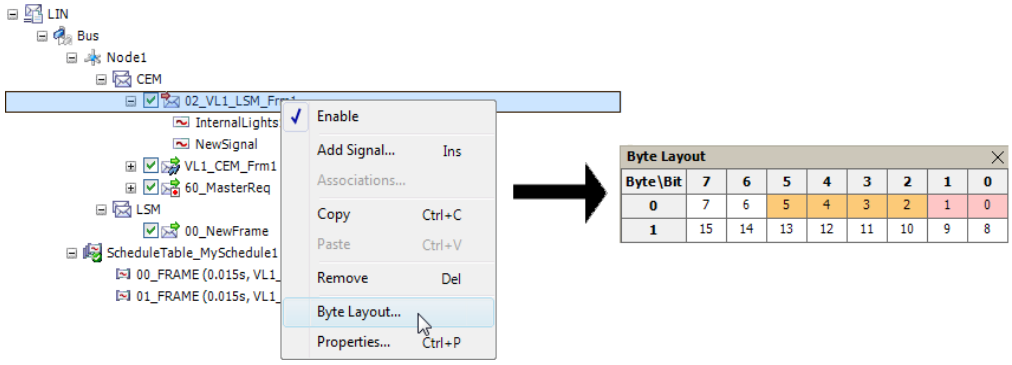
In diesem Bereich wird der Inhalt eines in der Netzwerkansicht ausgewählten Frames dargestellt.

Byte-Layout anzeigen

Um das Byte-Layout eines Frames anzuzeigen, gehen Sie wie folgt vor:

- Rechtsklicken Sie den entsprechenden Frame, dessen Layout Sie anzeigen wollen.
- Wählen Sie aus dem Kontextmenü **Byte Layout**.

  Das Byte-Layout des Frames wird angezeigt.



3.7.3 Bearbeiten des LIN-Netzwerks

Die Bearbeitung des LIN-Netzwerkes erfolgt ausschließlich über das Kontextmenü des jeweiligen Elementes. Welche Elemente ein solches Netzwerk enthält, ist im Abschnitt „Die Bestandteile eines LIN-Netzwerkes“ auf Seite 152 beschrieben.

Hinweis

*Die Eigenschaften von Elementen des LDF-Container können **nicht** bearbeitet werden, jedoch kann ein Element des LDF-Container dazu verwendet werden, durch Kopieren oder Verschieben ein entsprechendes Element neu im LIN-Netzwerk anzulegen.*

Zum Bearbeiten des Netzwerkes gibt es eine Reihe von Einträgen im Kontextmenü der jeweiligen Elemente, von denen einige auch auf eine Mehrfachauswahl (von gleichartigen Elementen) angewandt werden können.

Elemente hinzufügen

- Markieren Sie das übergeordnete Element und wählen Sie im Kontextmenü **Add (Element)**.

Hinweis

Namen von Parts, Frames und Signalen dürfen nur Zeichen enthalten, die für ANSI-C Identifier erlaubt sind.

Elemente entfernen

- Markieren Sie das Element und wählen Sie im Kontextmenü **Remove**.

Elemente kopieren

- Markieren Sie das Quellelement und wählen Sie im Kontextmenü **Copy**.
- Markieren Sie das übergeordnete Zielelement und wählen Sie im Kontextmenü **Paste**.

Eigenschaften von Elementen kopieren

- Markieren Sie das Quellelement und wählen Sie im Kontextmenü **Copy**.
- Markieren Sie das gewünschte Zielelement und wählen Sie im Kontextmenü **Paste**.

Elemente verschieben

- Markieren Sie das Quellelement.
- Ziehen Sie das markierte Quellelement mit dem Mauszeiger in das übergeordnete Zielelement und lassen Sie den Mauszeiger los.

Hinweis

Nicht immer sind alle oben beschriebenen Aktionen erlaubt – in diesen Fällen ist der entsprechende Eintrag des Kontextmenüs entweder deaktiviert oder nicht vorhanden.

Bearbeiten von Eigenschaften

Die Eigenschaften eines Elementes des LIN-Netzwerkes können im Feld „Properties“ bearbeitet werden. Einige Eigenschaften einiger Elemente (wie z.B. die PID eines LIN-Frames) können nicht bearbeitet werden, da sie automatisch berechnet werden. Diese Eigenschaften werden im Fenster „Properties“ ausgegraut dargestellt. Die Eigenschaften der Elemente des LDF-Container können ebenfalls nicht bearbeitet werden.

Eigenschaftenfenster anzeigen

- Markieren Sie ein Element und wählen Sie im Kontextmenü **Properties**.

  Das Feld „Properties“ wird im LIN-Editor angezeigt.

Eigenschaften mehrerer Elemente bearbeiten

Werden mehrere Elementen des gleichen Typs im Netzwerk markiert, werden diejenigen Eigenschaften angezeigt, die allen gewählten Elementen gemeinsam sind.

- Öffnen Sie die Anzeige von „Properties“.
- Wählen Sie die gewünschten Elemente mit der Maus bei gleichzeitig gedrückter STRG-Taste.

Hinweis

Wird beim Bearbeiten mehrerer Elemente eine Eigenschaft geändert, die das Element innerhalb seines übergeordneten Elements identifiziert (z.B. ein Signalname innerhalb eines Frames), kann dies zu Fehlermeldungen führen, falls die Eigenschaft eines weiteren Elementes des gleichen übergeordneten Elements ebenfalls diesen gleichen Wert erhält.

Validierung

Vor dem Speichern des LIN-Moduls wird eine Validierung des LIN-Netzwerks durchgeführt. Dabei werden u.a. die für die Codegenerierung relevanten Aspekte der LIN-Spezifikation geprüft. Fehlermeldungen werden im Logfenster ausgegeben.

Hinweis

Werden die Fehlermeldungen nicht beachtet, kann dies während des Build-Vorgangs des Projektes zu Kompilerfehlern oder zu unerwünschtem Laufzeitverhalten führen.

Module

Innerhalb eines LABCAR-OPERATOR-Projektes können mehrere LIN-Module angelegt werden. Diese Module können unabhängig voneinander konfiguriert werden, wobei jedoch ein LIN-Board nur in jeweils einem Modul verwendet werden kann.

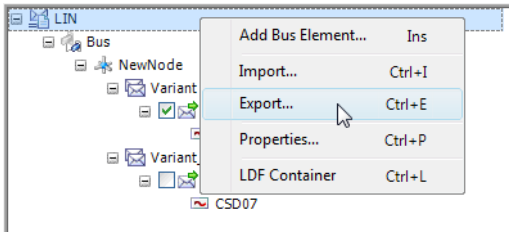
Netzwerke

Innerhalb eines LABCAR-OPERATOR LIN-Moduls entspricht der Wurzelknoten in der Baumansicht einem LIN-Netzwerk – unterhalb eines Netzwerkes werden Bus-Elemente angeordnet. LIN-Netzwerke können exportiert und importiert werden.

Die Hardwaretopologie eines Moduls wird in der Datei `LINConfiguration.xml` abgelegt. In dieser Datei wird auf die Konfigurationsdateien der in der Netzwerktopologie enthaltenen Parts (`*.llp`) und Schedule Tables (`*.lsc`) verwiesen. Sämtliche Dateien werden innerhalb des Verzeichnisses des Moduls abgelegt.

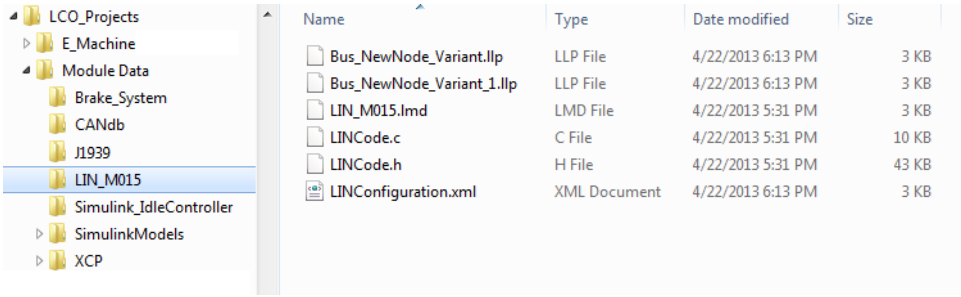
Netzwerk exportieren

- Rechtsklicken Sie das LIN-Netzwerk und wählen Sie **Export**.



- Wählen Sie einen vorhandenen Ordner oder erstellen Sie einen neuen und klicken Sie **OK**.

  Die Dateien werden in einem Verzeichnis mit dem Namen des LIN-Moduls erstellt.



Netzwerk importieren

- Rechtsklicken Sie das LIN-Netzwerk und wählen Sie **Import**.

  Ein Dateiauswahlfenster wird geöffnet.

- Wählen Sie die Datei `LINConfiguration.xml`.

Hinweis

Dateien, die benutzerdefinierten C-Code (.c und *.h) enthalten (siehe auch „Benutzerdefinierter C-Code" auf Seite 167), werden nicht exportiert und importiert. Diese müssen bei Bedarf manuell übertragen werden.*

3.7.4 Die Bestandteile eines LIN-Netzwerkes

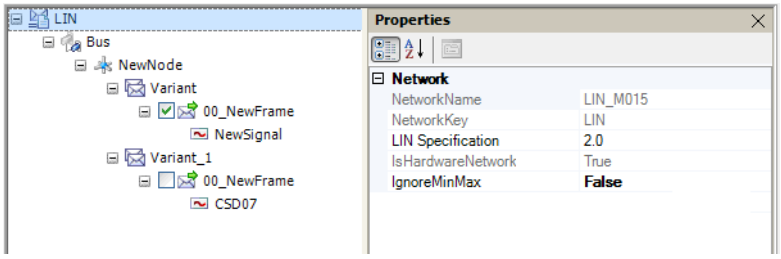
Im Folgenden finden Sie eine Beschreibung der Bestandteile des LIN-Netzwerkes in LABCAR-IP.

Im Einzelnen sind dies:

- „LIN-Netzwerk“ auf Seite 152
- „Bus“ auf Seite 153
- „Nodes“ auf Seite 155
- „Schedule Tables“ auf Seite 155
- „Commands“ auf Seite 156
- „Parts“ auf Seite 157
- „Frames“ auf Seite 159
- „Signale“ auf Seite 163

LIN-Netzwerk

Die globalen Eigenschaften des LIN-Netzwerks werden bei dessen Auswahl im Fenster „Properties“ angezeigt.



Network:

- **NetworkName**

  Editierbarer Name des LIN-Netzwerks
- **NetworkKey**

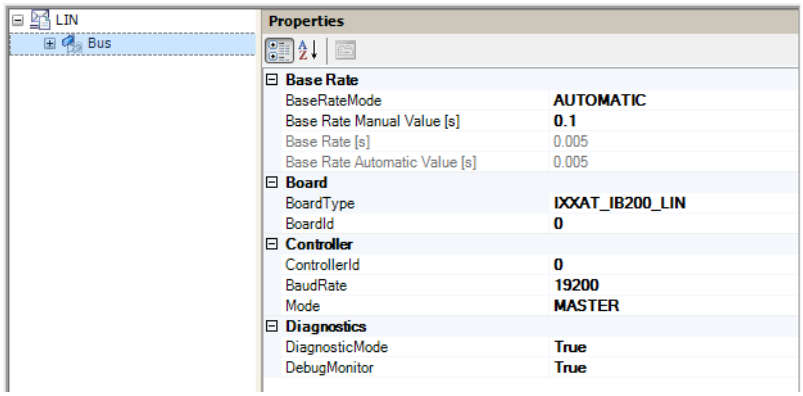
  Eindeutige Kennung des Netzwerkes (wird automatisch generiert)
- **LIN Specification**

  LIN 2.0 oder LIN 2.1/2.2
- **IsHardwareNetwork**

  Gibt an, ob es sich um ein reales Netzwerk mit konfigurierter Hardware handelt oder um ein Netzwerk in einer LDF-Datei.
- **Ignore Min/Max**

  Ignoriert die Min/Max-Beschränkungen von Signalwerten. Ist diese Option gesetzt, werden die physikalischen Signalwerte nicht durch die Min/Max-Einschränkung in den Signaleigenschaften limitiert (bevor diese in die elektrischen Signalwerte umgewandelt werden).

Bus



Base Rate:

Die Eigenschaft „BaseRate“ definiert die Rate, mit der die Kommunikation zwischen dem LABCAR-OPERATOR-Projekt und dem ausgewählten Controller des LIN-Moduls erfolgt.

Der aktuelle Wert von BaseRate wird im Fenster „Properties“ des Controllers angezeigt – er wird aus folgenden Eigenschaften des Controllers bestimmt:

- **BaseRateMode**

  „BaseRateMode“ gibt an, wie „Base Rate“ bestimmt werden soll. „Base Rate“ kann

  – manuell („MANUAL“)

  oder

  – automatisch („AUTOMATIC“)

  berechnet werden. Bei automatischer Berechnung kann auch eine zwei- bzw. vierfache Übertaktung („AUTOMATIC2x“ / „AUTOMATIC4x“) gewählt werden.

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

  „Base Rate Manual Value“ ist die durch den Anwender manuell eingegebene gewünschte Rate.

- **Base Rate [s]**

  Die Base Rate

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

  „Base Rate Automatic Value“ wird aus dem größten gemeinsamen Teiler aller Werte der Eigenschaft „Delay“ aller Commands sowie aller Werte der Eigenschaft „Base Rate“ aller Parts unterhalb des gewählten Controllers berechnet. Die Eigenschaft „Base Rate“ eines Parts wird aus dem Parameter „Master Time Base“ aus den importierten LDF-Dateien definiert. Diese Eigenschaft eines Parts kann auch nachträglich geändert werden.

Board:

- **BoardType**

  Typ des verwendeten Boards („iPC-I XC16/PCI“, „CAN-IB200/PCIe“ oder „CAN-IB600/PCIe“).

- **BoardID**

  Eindeutiger Identifier des Boards innerhalb des Netzwerks

Controller:

- **ControllerID**

  Eindeutiger Identifier des Controllers auf einem Board

Hinweis

Beim CAN-IB200 entspricht Modul-Steckplatz 1 (Kanal 1) dem Controller 1 und Modul-Steckplatz 2 (Kanal 2) dem Controller 0!

- **BaudRate**

  Die Übertragungsrate des Controllers

- **Mode**

  „MASTER“ oder „SLAVE“.

Hinweis

Innerhalb eines LIN-Moduls darf es nur einen Controller im Modus „MASTER“ geben!

Diagnostics:

Zur Unterstützung der Ausführung von Diagnoseaufgaben stehen folgende Eigenschaften zur Verfügung:

- **DiagnosticMode**

  Ist diese Eigenschaft aktiviert (Eigenschaft „DiagnosticMode“ = TRUE), werden boolesche Kalibriervariablen erzeugt, die es zur Laufzeit des Experimentes erlauben, die Kommunikation auf Nicht-Diagnostic Frames zu unterbinden (Kalibriervariable = TRUE) oder zuzulassen (Kalibriervariable = FALSE).

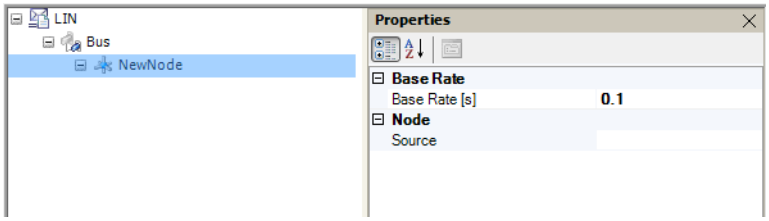
- **DebugMonitor**

  Ist diese Eigenschaft aktiviert, werden Messvariablen erzeugt, welche es zur Laufzeit des Experimentes erlauben, die Kommunikation zwischen dem Modul und den vorhandenen Controllern zu beobachten.
  Für gesendete oder empfangene Frames (Master und Slave) sowie für gesendete Frame-Header von angeforderten Frames (nur Master) werden entsprechende Messvariablen angelegt. Diese zeigen den vollständigen Inhalt aller zuletzt übertragenen Bytes.

Hinweis

Da die Kommunikation zwischen Modul und Controller mit einer höheren Rate erfolgt als mit der eingestellten Base Rate (pro Zyklus werden die Daten mehrerer Frames ausgetauscht), sind diese Messparameter nicht dazu geeignet, verlustfrei die Kommunikation aufzuzeichnen.

Nodes



Base Rate:

Für einen Part steht folgende Eigenschaft zur Konfiguration des Zeitverhaltens zur Verfügung:

- **Base Rate []**

  Die Base Rate eines Parts wird über den Parameter „Master Time Base“ aus den importierten LDF-Dateien festgelegt.

  Diese Eigenschaft eines Parts kann auch nachträglich geändert werden. Aus der Base Rate wird die Eigenschaft „BaseRateAutomaticValue“ (siehe „Base Rate Automatic Value [s]“ auf Seite 153) eines Controllers berechnet.

Node:

- **Source**

  Quelle des Knotens (nach Import)

Schedule Tables

Eine Schedule Table wird einem übergeordneten Controller (im Master-Modus) zugeordnet. Unter einer Schedule Table werden Commands (siehe „Commands“ auf Seite 156) angeordnet. Die Inhalte einer Schedule Table werden innerhalb des Moduls in Dateien mit der Dateierweiterung `.lsc` abgelegt.

Innerhalb des Netzwerkes besitzen Schedule Tables einen eindeutigen Namen. Ist bereits eine Schedule Table mit einem gegebenen Namen vorhanden, wird der Name der hinzuzufügenden Schedule Table mit einem durchgehenden Zähler als Suffix erweitert.

Startup-Verhalten ändern

- Markieren Sie die Schedule Table und aktivieren oder deaktivieren Sie im Kontextmenü **Use as Startup**.

Hinweis

Innerhalb eines Controllers kann nur eine Schedule Table als „Startup“ markiert werden. Wird eine andere Schedule Table als „Startup“ markiert, wird die vorherige Markierung entfernt.

Commands

Ein Command wird einer übergeordneten Schedule Table zugeordnet. Commands werden innerhalb einer Schedule Table anhand einer eindeutigen Position identifiziert. Die Commands werden in der Reihenfolge ihrer Positionen abgearbeitet.

Position ändern

- Markieren Sie das gewünschte Command und wählen Sie im Kontextmenü **Move Up** oder **Move Down**.

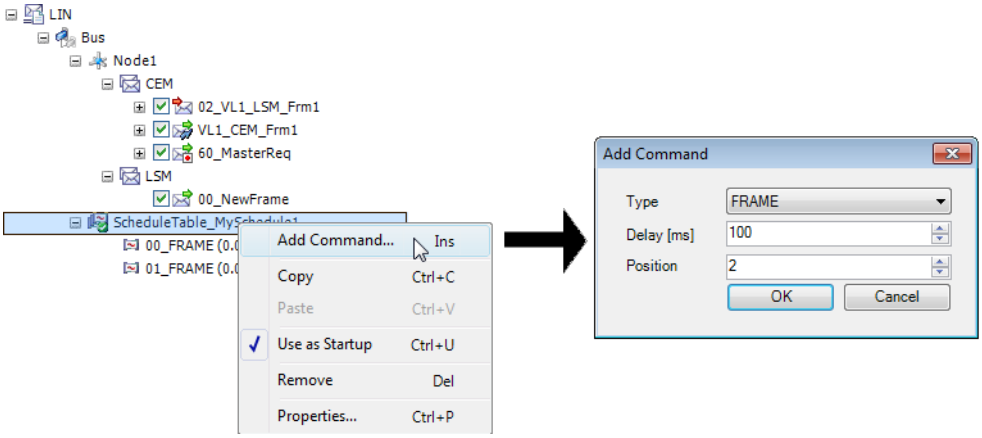
Hinweis

Die Position eines Commands kann auch über das Eigenschaftsfenster angepasst werden. Ändert sich die Position eines Commands, werden u.U. die Positionen anderer Commands automatisch angepasst. Die Eigenschaft „Position" eines Commands ist für jedes Command einer Schedule Table eindeutig.

Benutzerdefiniertes Command anlegen

Neben der Möglichkeit, ein Command aus einer LDF-Datei zu importieren, können auch benutzerdefinierte Commands angelegt werden.

- Markieren Sie die übergeordnete Schedule Table und wählen Sie im Kontextmenü **Add Command**.



Hinweis

Beim Anlegen eines benutzerdefinierten Commands müssen zunächst nur die allgemein notwendigen Parameter „Delay" und „Position" angegeben werden. Im Anschluss können die für den Typ des Commands spezifischen Parameter Im Feld „Properties" angepasst werden.

Parts

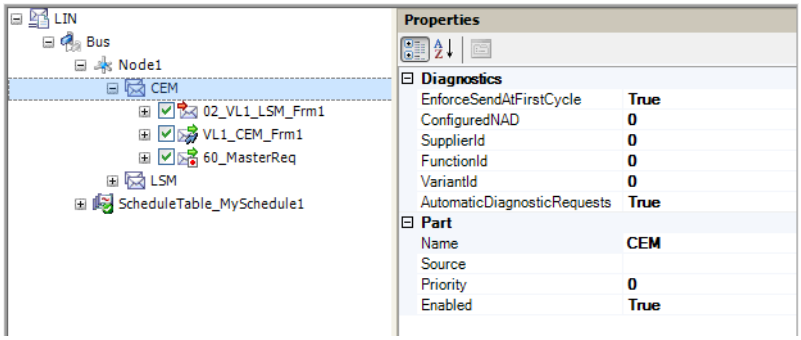
Ein Part wird einem Knoten zugeordnet, unter einem Part werden Frames angeordnet. Parts können exportiert und importiert werden. Die Inhalte eines Parts werden innerhalb des Moduls in Dateien mit der Dateierweiterung `.llp` abgelegt.

Parts besitzen innerhalb des gesamten Netzwerkes einen eindeutigen Namen. Existiert bereits ein Part mit einem gegebenen Namen, wird der Name des hinzuzufügenden Parts mit einer fortlaufenden Zahl erweitert.

Priorisierung, Aktivierung und Deaktivierung von Parts

Parts dienen der Modellierung von Varianten eines Netzwerkknotens. Dazu können von einem Part Kopien angelegt werden einschließlich aller unter dem Part definierten Frames und Signale.

Über die Eigenschaft „Enabled“ können Parts aktiviert oder deaktiviert werden. Mit der Eigenschaft „Priority“ kann die Abarbeitung der Frames und Signale dieses Parts gesteuert werden: Parts mit höherer Priorität überschreiben in jedem Zyklus der Simulation die Ein- und Ausgaben der Parts mit niederer Priorität.



Part:

- **Name**

  Der Name des Parts

- **Source**

  Quelle des Parts (nach Import)

- **Priority**

  Mit dieser Eigenschaft kann die Priorität des Parts angegeben werden. Die Priorität muss einen eindeutigen Wert haben – die Ausführung der Parts erfolgt entsprechend ihrer Priorität.

- **Enabled**

  Jeder Part kann (unabhängig von den anderen Parts) zur Laufzeit des Experiments aktiviert und deaktiviert werden. Ist ein Part deaktiviert, wird die komplette Kommunikation aller Frames unterhalb des gewählten Parts unterbunden.

Hinweis

Mit den Eigenschaften „Enabled“ und „Priority“ ist ein Variantenhandling während der Laufzeit des Experiments möglich.

Diagnostics:

Zur Unterstützung der Ausführung von Diagnoseaufgaben stehen folgende Eigenschaften zur Verfügung:

- **EnforceSendAtFirstCycle**

  Zur Optimierung der Performanz wird der Code so generiert, dass die Daten eines zu sendenden Frames nur dann in den Übertragungspuffer des Controllers geschrieben werden, wenn sich seit der letzten Übertragung mindestens ein Signal des Frames geändert hat.

  Im ersten Zyklus der Simulation wird der Frame mit den konfigurierten Startwerten der Signale in den Übertragungspuffer geschrieben. Ist die Eigenschaft „EnforceSendAtFirstCycle" deaktiviert („FALSE"), wird diese Initialisierung des Übertragungspuffers unterbunden. Damit kann simuliert werden, dass ein Part zu Beginn der Simulation noch nicht an der Netzwerkkommunikation teilnimmt.
- **ConfiguredNAD**
- **SupplierId**
- **FunctionId**
- **VariantId**

  Diese Eigenschaften identifizieren den durch den Part formulierten Knoten im Netzwerk. Dies ist zur automatisierten Behandlung der Commands `ASSIGN_NAD` und `LIN PRODUCT ACTIVATION` notwendig.
- **AutomaticDiagnosticRequest**

  Mit dieser Eigenschaft kann die Codegenerierung zur automatischen Behandlung der Commands `ASSIGN_NAD` und `LIN PRODUCT ACTIVATION` gemäß LIN-Spezifikation für diesen Part aktiviert werden. Die automatische Behandlung dieser Commands erfolgt nur auf Controller im Slave-Modus.

Hinweis

Ist diese Eigenschaft aktiviert, wird für diesen Part eine Messvariable angelegt, die den aktuellen Wert der Netzwerkadresse „Current_NAD" anzeigt.

Part exportieren

- Markieren Sie den zu exportierenden Part und wählen Sie im Kontextmenü **Export Part**.

  Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie ein Verzeichnis und einen Dateinamen und klicken Sie **OK**.

  Der Part wird exportiert.

Part importieren

- Markieren Sie den übergeordneten Controller und wählen Sie im Kontextmenü **Import Part**.

  Ein Dateiauswahlfenster wird geöffnet.

- Wählen Sie eine Datei und klicken Sie **OK**.
  Der Part wird importiert.

Frames

Ein Frame wird einem übergeordneten Part zugeordnet, unter einem Frame werden (abhängig vom Typ des Frames) Signale angeordnet.

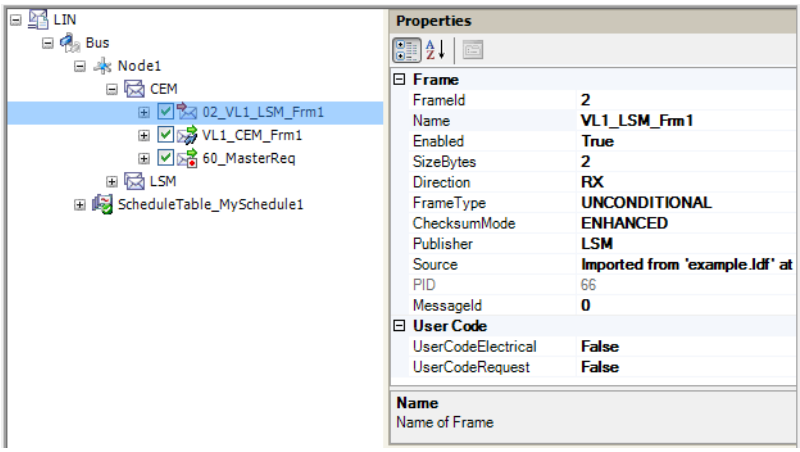
Für Frames kann neben dem Eigenschaftenfenster auch das Byte-Layout-Fenster angezeigt werden. Dieses zeigt die Belegung der Bytes in der Payload des Frames mit den Signalen.

Byte-Layout anzeigen

- Markieren Sie den Frame und wählen Sie im Kontextmenü **Byte Layout**.

Hinweis

Treten Fehler im Byte-Layout auf, z.B. durch Überlappung zweier Signale oder fehlerhafter Angabe der Länge eines Signals oder des Frames, werden die fehlerhaften Bits in roter Farbe dargestellt.



Frame:

- **FrameId**
  ID des Frames
- **Name**
  Name des Frames
- **Enabled**
  Frame in Simulation aktiviert („TRUE“) oder („FALSE“)
- **SizeBytes**
  Größe des Frames (in Bytes)
- **Direction**
  Transportrichtung (TX = senden, RX = empfangen)

- **FrameType**

  Typ des Frames:

  – UNCONDITIONAL
  – SPORADIC
  – EVENTTRIGGERED
  – DIAGNOSTIC
  – USERDEFINED

- **ChecksumMode**

  Modus für Berechnung der Checksumme

  – CLASSIC
  – ENHANCED

- **Publisher**

  Editierbarer Name des sendenden Knotens (wird aus LDF entnommen)

- **Source**

  Quelle des Elements

- **PID**

  Protected Identifier des Frames (berechneter Wert)

- **MessageId**

  Optionale Message-ID des Frames zur Verwendung des Commands ASSIGN_FRAME_ID

User Code:

Hier können Sie festlegen, ob zu dem Frame benutzerdefinierter C-Code hinzugefügt werden soll (siehe „Benutzerdefinierter C-Code“ auf Seite 167).

- **UserCodeElectrical**

  Dieser Code wird vor dem Senden bzw. nach dem Empfangen eines Frames aufgerufen und ermöglicht Zugriff auf die Payload des Frames.

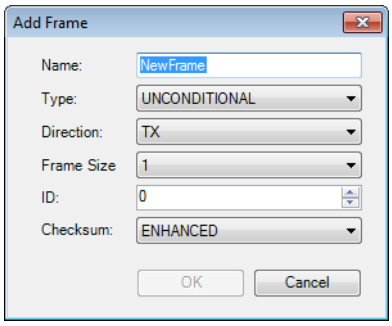
  UserCodeRequest

  Dieser Code wird nach dem Empfangen eines Frame Request aufgerufen.

Benutzerdefinierten Frame anlegen

Neben der Möglichkeit, einen Frame aus einer LDF-Datei, einem Modul oder einem Part zu importieren, können auch benutzerdefinierte Frames angelegt werden.

- Markieren Sie den übergeordneten Part und wählen Sie im Kontextmenü **Add Frame**.



Hinweis

Innerhalb eines Parts muss die Kombination aus der ID des Frames und dessen Name eindeutig sein!

Hinweis

Nicht alle Kombinationen der Parameter eines Frames sind erlaubt!
Beispielsweise ergibt sich aus der Kombination der Eigenschaften
- Type = DIAGNOSTIC,
- der Senderichtung
und
- des Modus des Controllers
zwangsläufig die ID des Frames.
*Sind bei bestimmten Kombinationen einzelne Parameter nicht mehr frei wählbar oder zumindest nur mit eingeschränktem Wertebereich, werden die entsprechenden Eingabefelder deaktiviert oder deren Wertebereich entsprechend eingeschränkt. Würde die Eingabe zu schweren Fehlern in der Konfiguration führen, kann die Eingabemaske nicht mit **OK** geschlossen werden.*

Aktivierung und Deaktivierung von Frames

Innerhalb eines Parts dürfen mehrere Frames mit gleicher ID definiert sein, sofern diese jeweils einen unterschiedlichen Namen besitzen. Zur Codegenerierung darf jedoch in einem Part nur jeweils ein Frame mit einer gegebenen ID aktiviert sein. Dazu müssen die jeweils anderen Frames aus der Codegenerierung ausgeschlossen werden.

- **Über Baumansicht des Netzwerks**: Aktivieren bzw. Deaktivieren Sie das Kästchen neben dem Framesymbol in der Baumansicht des Netzwerkes.
- **Über Kontextmenü**: Markieren Sie den Frame und aktivieren bzw. deaktivieren Sie im Kontextmenü **Enable**.

- **Über Eigenschaftsfenster**: Öffnen Sie das Eigenschaftsfenster und Bearbeiten Sie die Eigenschaft „Enabled“ („True“ oder „False“).

Einen Frame entfernen

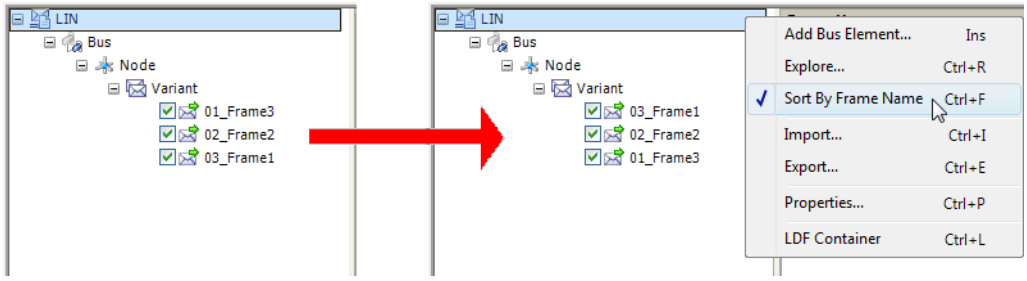
- Wählen Sie im Kontextmenü der Frameliste **Remove**.

  Der Frame wird entfernt.

Frames nach Namen ordnen

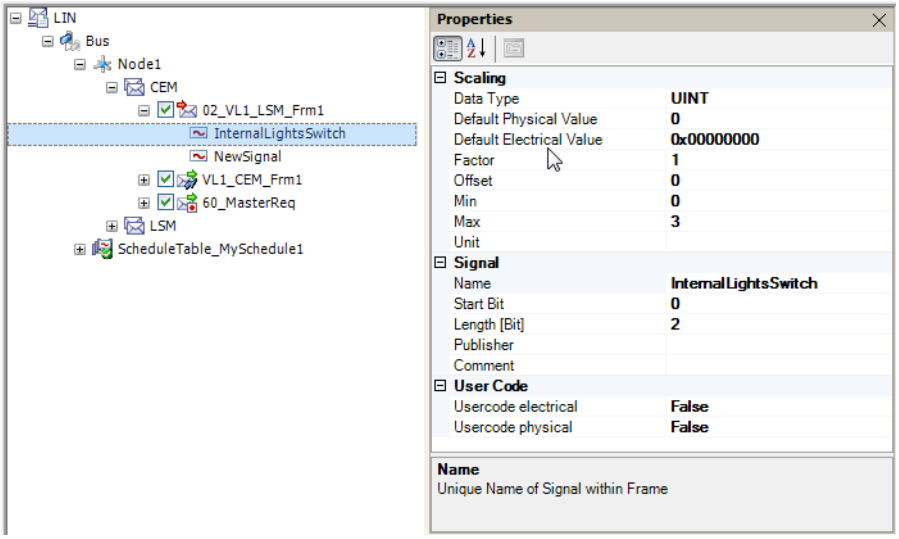
Die Frames sind normalerweise nach ihren Frame Identifiern geordnet.

- Um die Frames nach ihrem Namen zu ordnen, rechtsklicken Sie den Modulnamen und wählen Sie die Option **Sort by Frame Name**.



Signale

Ein Signal wird einem übergeordneten Frame zugeordnet, der Name eines Signals unterhalb dieses Frames muss dabei eindeutig sein.



Scaling:

- **Data Type**

  „UINT“, „BOOLEAN“ oder „BYTEARRAY“
- **Default Physical Value**
- **Default Electrical Value**

  Ein Signal eines Frames kann mit einem Defaultwert versehen werden – wenn das Signal im Connection Manager mit einem anderen Signal verbunden ist, wird dieser Wert ignoriert.

  Der Nutzen des Defaultwertes liegt beim Import/Export von LIN-Modulen, da der Defaultwert dann auch bei der Wiedernutzung des LIN-Moduls in einem anderen LABCAR-OPERATOR-Projekt zur Verfügung steht.
- **Factor, Offset**

  Ein Faktor a und ein Offset b zur Skalierung des Signals
- **Min, Max**

  Hier können Grenzen für eine Plausibilitätsprüfung angegeben werden (siehe „Min/Max-Werte“ auf Seite 128)
- **Unit**

  Die physikalische Einheit des Signals

Signal:

- **Name**

  Name des Signals
- **Start Bit**

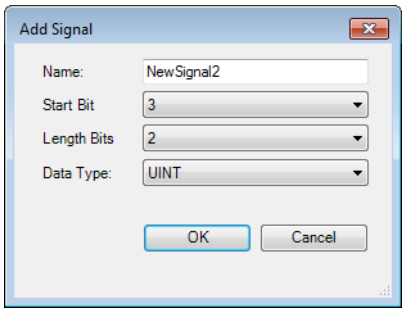
  Erstes Bit des Signals
- **Length [Bit]**

  Signallänge in Bit

- **Publisher**
  Editierbarer Name des sendenden Knotens (wird aus LDF entnommen)
- **Comment**
  Ein (optionaler) Kommentar

Benutzerdefiniertes Signal anlegen

- Markieren Sie einen Frame und wählen Sie im Kontextmenü **Add Signal**.



Hinweis

Nicht alle Kombinationen der Parameter eines Signals sind erlaubt! Beispielsweise ergibt sich aus der Eigenschaft „Signal Type = BOOLEAN" zwangsläufig die Länge des Signals.
*Sind bei bestimmten Kombinationen einzelne Parameter nicht mehr frei wählbar oder zumindest nur mit eingeschränktem Wertebereich, werden die entsprechenden Eingabefelder deaktiviert oder deren Wertebereich entsprechend eingeschränkt. Würde die Eingabe zu schweren Fehlern in der Konfiguration führen kann die Eingabemaske nicht mit **OK** geschlossen werden.*

Associations

Eine „Association" ist die Verknüpfung eines Unconditional Frame mit einem Event Triggered Frame oder Sporadic Frame verstanden. Eine Association wird einem übergeordneten Frame zugeordnet.

Die Association wird in der Baumansicht des Netzwerkes als Verknüpfung des Unconditional Frames unterhalb des zugeordneten Event Triggered Frame oder Sporadic Frame dargestellt.

Das obige Beispiel zeigt ein Event Triggered Frame (ID 13) mit zwei Associations zu den beiden Unconditional Frames ID 11 und ID 12.

Ein Event Triggered Frame oder Sporadic Frame darf Associations zu mehreren Unconditional Frames besitzen, jedoch darf nur je ein Unconditional Frame im gleichen Part wie der Event Triggered Frame oder Sporadic Frame sein. Associations zu Unconditional Frames, welche in einem anderen Part definiert sind, werden daher ausgegraut dargestellt.

Eine Association erzeugen

- Ziehen Sie den Unconditional Frame mit dem Mauszeiger auf einen Event Triggered Frame oder einen Sporadic Frame und lassen Sie dort den Unconditional Frame los.

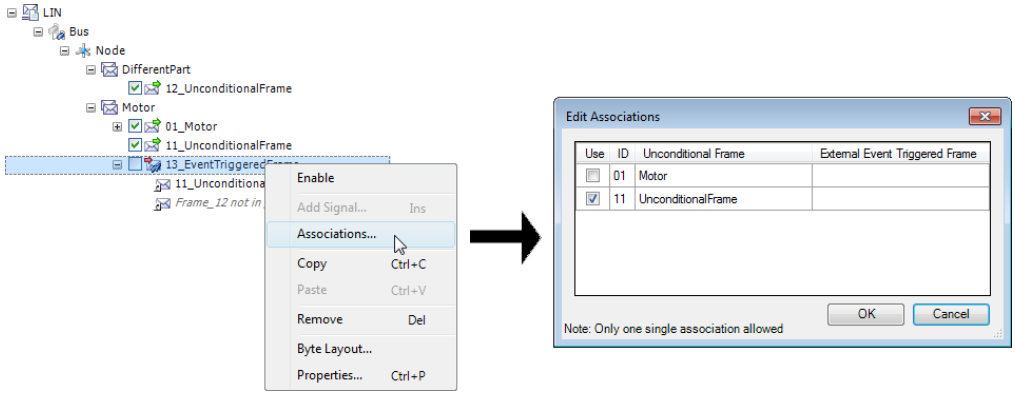
Eine Association entfernen

- Markieren Sie die zu entfernende Association unter dem übergeordneten Event Triggered Frame und wählen Sie im Kontextmenü **Remove**.

Association bearbeiten

- Markieren Sie den übergeordneten Event Triggered Frame und wählen Sie im Kontextmenü **Associations**.

  Das Fenster „Edit Associations“ wird geöffnet.



- Wählen Sie den zum Event Triggered Frame oder Sporadic Frame zu assoziierenden Unconditional Frame (innerhalb des gleichen Parts). Dabei kann nur ein einzelner Unconditional Frame ausgewählt werden.

Hinweis

Associations von Unconditional Frames des übergeordneten Parts zu weiteren Event Triggered Frames oder Sporadic Frames werden ausgegraut dargestellt. Diese Associations können nur durch Bearbeiten des jeweils anderen Event Triggered Frames geändert werden.

Event Triggered Frames

- Im Falle einer Association mit einem Event Triggered Frame müssen sowohl der Master als auch die betroffenen Slaves über diese Beziehung Kenntnis besitzen. Daher muss die Association zwischen Event Triggered Frame und Unconditional Frame in jedem betroffenen Part formuliert werden.
- Für Parts unter Controllern im Master-Modus müssen die Richtung des Event Triggered Frames und des Unconditional Frames RX (lesend), für Controller im Slave-Modus jeweils TX (sendend) sein.

Sporadic Frames

- Im Falle einer Association mit einem Sporadic Frame müssen nur der sendende Part über diese Beziehung Kenntnis besitzen.
- Für alle Parts, in denen eine solche Beziehung vorkommt, muss die Richtung des Sporadic Frames und des Unconditional Frames TX (sendend) sein.

3.7.5 Benutzerdefinierter C-Code

In LABCAR-NIL ist es möglich, zum automatisch erzeugten Code weiteren benutzerdefinierten Code hinzuzufügen, um die Inhalte von Frames oder Signalen zu zu manipulieren oder zu erweitern. Bei Signalen gilt dies sowohl für die physikalische Darstellung (skaliert und mit physikalischen Einheiten) als auch für die elektrische Darstellung (unskaliert und ohne physikalische Einheit).

- **Send Frames**

  Die Payload des Frames kann unmittelbar vor dem Versenden manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodeElectrical“ des Frames.

- **Receive Frames**

  Die Payload des Frames kann unmittelbar nach dem Empfang manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodeElectrical“ des Frames.

- **Frame Header**

  Der angeforderte Frame-Header kann unmittelbar vor dem Versenden manipuliert werden (nur für Controller im Master-Modus) – verwenden Sie dazu die Eigenschaft „UserCodeRequest“ des Frames.

- **Send Signals**

  Das Signal kann unmittelbar nach dem Auslesen des Inports als Wert vom Typ „Double“ manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodePhysical“ des Signals.

  Das Signal kann unmittelbar vor dem Einfügen in die Payload des Frames als elektrische Größe (unskaliert und ohne physikalischer Einheit) als Wert vom Typ „uint64“ manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodeElectrical“ des Signals.

- **Receive Signals**

  Das Signal kann unmittelbar nach der Extraktion aus der Payload des Frames als elektrische Größe (unskaliert und ohne physikalischer Einheit) als Wert vom Typ „uint64“ manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodeElectrical“ des Signals.

  Das Signal kann unmittelbar vor dem Schreiben des Outports des LIN-Moduls als physikalische Größe (skaliert und mit physikalischer Einheit) als Wert vom Typ „Double“ manipuliert werden – verwenden Sie dazu die Eigenschaft „UserCodePhysical“ des Signals.

Vorgehensweise

Nach dem Speichern der LIN-Konfiguration befindet sich im Modulverzeichnis die Header-Datei `LINCode.h`, welche die Funktionsdeklarationen zum Einbinden des Benutzercodes enthält.

Am Ende der Datei `LINCode.h` befindet eine Kommentarzeile

```
// Declarations for user code insertion:
```

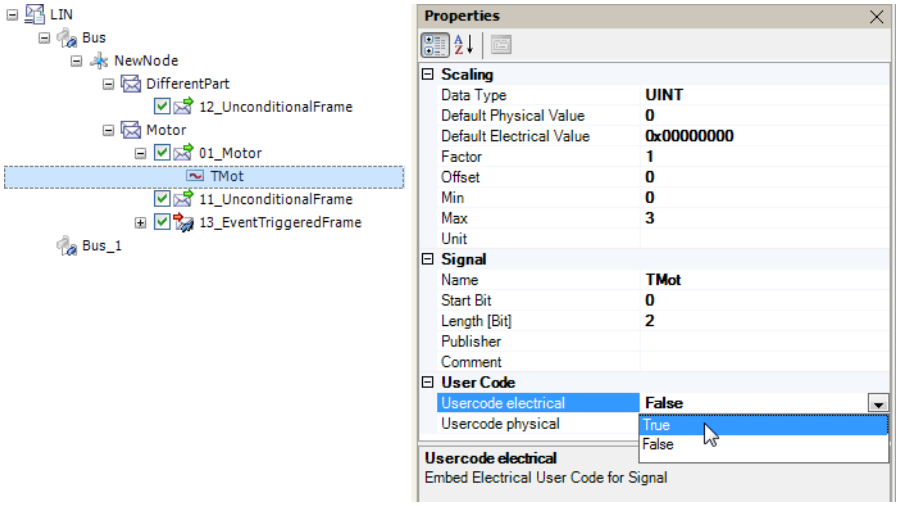
Unterhalb dieser Kommentarzeile werden die Funktionsaufrufe des benutzerdefinierten Codes deklariert – diese Funktionen müssen im benutzerdefinierten Codes implementiert werden.

Benutzerdefinierten Code deklarieren

- Bearbeiten Sie die Eigenschaften
  - „UserCodePhysical“
  - „UserCodeElectrical“

  und
  - „UserCodeRequest“

  der Frames und Signale, zu denen Sie benutzerdefinierten Code hinzufügen wollen.

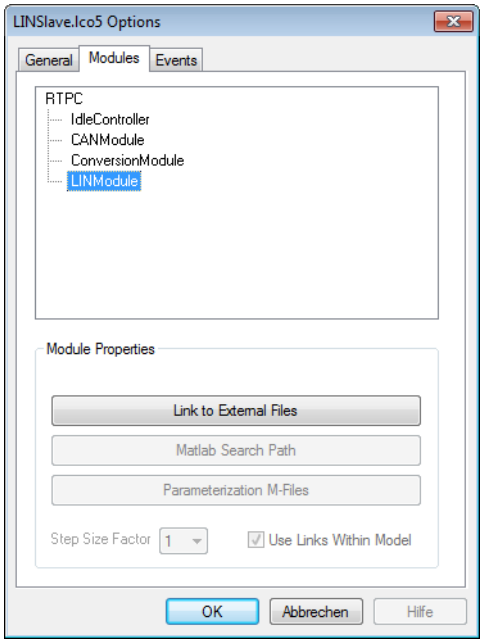


- Wählen Sie **File** → **Save**, um das Modul zu speichern.
- Öffnen Sie mit einem geeigneten Editor die Datei `LINCode.h` im Modulverzeichnis.
- Kopieren Sie die Funktionsdeklarationen des benutzerdefinierten Codes in eine Textdatei.
- Nennen Sie diese Textdatei z.B. `UserCode.c` und legen Sie diese innerhalb des LIN-Modulverzeichnisses ab.

- Fügen Sie Ihrer C-Datei `UserCode.c` die Zeile

  `#include "LINCode.h"`

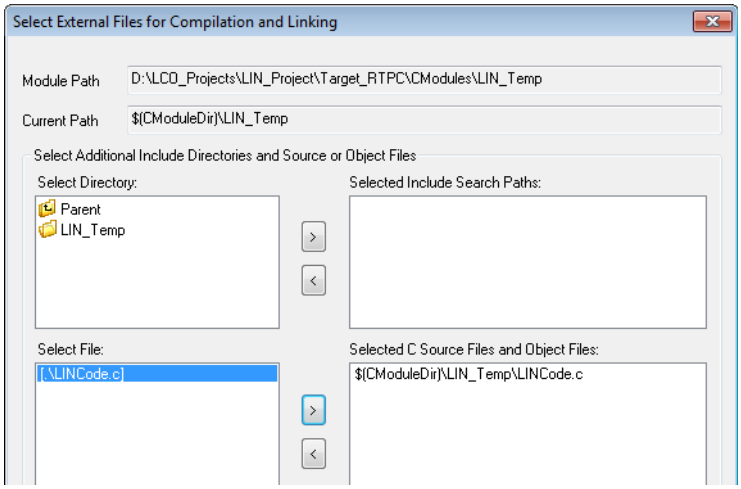
  hinzu, damit die Funktions- und Struktur-deklarationen bekannt sind.
- Implementieren Sie den benutzerdefinierten Code in `UserCode.c`.
- Wählen Sie im Hauptmenü **Project → Options**.
- Wählen Sie im Optionsfenster die Registerkarte „Modules“.



- Wählen Sie „LINModules“ und klicken Sie **Link to External Files**.

  Das Fenster „Select External Files for Compilation and Linking“ wird geöffnet.

- Wählen Sie die einzubindende(n) Quelldatei(en).



Hinweis

Es können nur Dateien ausgewählt werden, die sich im Modulverzeichnis befinden.

Die gewählten Dateien werden im Feld „Selected C Source Files and Object Files“ angezeigt.

- Klicken Sie **OK**.

  Diese Dateien werden bei zukünftigen Codegenerierungen immer mitkompiliert und mitgelinkt.
- Verlassen Sie das Register „Options“ mit **OK**.

Datentypen und Strukturen

In den Funktionsdeklarationen des benutzerdefinierten Codes zur Manipulation eines Frames werden folgende Datentypen und Strukturen verwendet:

Konstanten

```
/* The following constants are used for the "mtype" field in
ixxat_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
ixxat_lin_data_t:
*/
#define IXXAT_LIN_CRCMODEL_CLASSIC        0
#define IXXAT_LIN_CRCMODEL_ENHANCED       1
```

Datenübertragung Senden und Empfangen

```
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;
```

Datenübertragung Empfangen: Fehlerrückgabe

```
typedef struct {
   unsigned short errorcode;     // LIN error code (receive only)
} ixxat_lin_load_error_t;
```

Datenübertragung Empfangen: Statusrückgabe

```
typedef struct {
   unsigned short status;        // LIN status (receive only)
} ixxat_lin_load_status_t;
```

Datenübertragung Empfangen: 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;
```

Datenübertragung Senden und Empfangen: 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 mechanismns:
              unsigned short uwdata[4];
              unsigned int udwdata[2];
              unsigned long long uqwdata;
              signed char sdata[8];
              signed short swdata[4];
              signed int sdwdata[2];
              signed long  sqwdata;
        };
  } ixxat_lin_load_data_t;
```

3.7.6 Das LIN-Modul in ETAS EE

In diesem Abschnitt finden Sie Informationen zum Arbeiten mit einem LIN-Modul in der Experimentierumgebung ETAS EE.

Nachdem das Projekt erstellt und auf dem Target kompiliert wurde, stehen für jedes LIN-Modul im Projekt eine Anzahl Ein- und Ausgänge zur Steuerung und Kommunikation zur Verfügung. Die Ein- und Ausgänge eines LIN-Moduls werden im Experiment Explorer unterhalb des Moduls angezeigt.

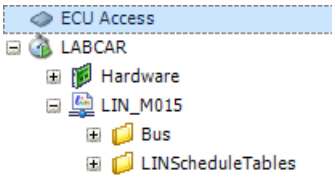
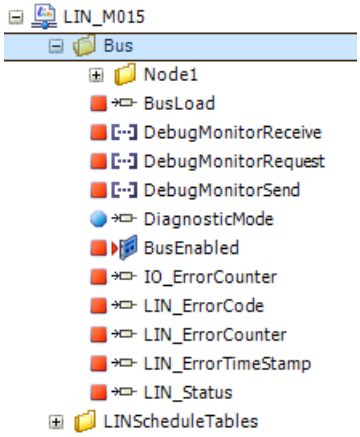


Abb. 3-19 Das LIN-Modul in den Workspace Elements

In den Workspace Elements wird jedes LIN-Modul in einem eigenen Knoten („Name des Moduls“) angezeigt. Unter diesem Modulknoten wird für jedes konfigurierte Buselement ein jeweiliges Buselement-Knoten („Bus“) angezeigt. Sämtliche Ein- und Ausgänge zur Steuerung des Buselements oder der darin enthaltenen Elemente werden unter diesem Buselement-Knoten dargestellt.

Bus

Unter „Bus“ werden Ein- und Ausgänge zur Steuerung und Überwachung der Kommunikation mit dem konfigurierten LIN-Controller angezeigt.



- **BusLoad**

  Zeigt die momentane Auslastung des LIN-Busses in Prozent an.

Hinweis

Wenn diese Anzeige vom Board nicht unterstützt wird, wird der Wert „-1“ angezeigt“!

- **DebugMonitorReceive**
- **DebugMonitorRequest**
- **DebugMonitorSend**

  Messvariablen zur Anzeige der zuletzt übertragenen Bytes zwischen LIN-Modul und Controller

Hinweis

Da die Kommunikation zwischen Modul und Controller mit einer höheren Rate als der eingestellten Base Rate erfolgt (pro Zyklus werden die Daten mehrerer Frames ausgetauscht), sind diese Messvariablen nicht dazu geeignet, die Kommunikation verlustfrei aufzuzeichnen.

- **DiagnosticMode**

  Kalibriervariable zur Aktivierung und Deaktivierung des „Diagnostic Modes“ des Controllers

- **BusEnabled**

  Inport zur Aktivierung und Deaktivierung eines Busses.

  – - 0.5 < Wert des Inports < + 0.5 = FALSE: Bus deaktiviert

– sonst TRUE (Default = 1.0): Bus aktiviert

Hinweis

Die Aktivierung und Deaktivierung eines Busses geschieht durch das verwendete Board. Dabei kann es während eines Zustandswechsels der Kommunikation für einen einzelnen Controller zur Störungen auf Bussen auch an anderen Controllern kommen. Diese Störungen dauern etwa so lange wie der Zustandswechsel.

- **IO_ErrorCounter**

  Fehlerzähler für die Kommunikation mit dem gewählten Controller.

  Häufige Fehlerursache sind falsche Angaben der Eigenschaften „BoardId“ und „ControllerId“ oder Fehler in der Firmware der LIN-Boards.

- **LIN_ErrorCounter**

  Fehlerzähler für Fehler bei der LIN-Kommunikation mit dem IXXAT-Treiber.

- **LIN_ErrorTimeStamp**

  Zeitstempel des letzten Auftretens von „LIN_ErrorCode“. Der Zeitstempel wird in Mikrosekunden seit Start der LIN-Kommunikation angegeben und vom LIN-Board bestimmt.

- **LIN_ErrorCode**

  Fehlercode für die Kommunikation über das LIN-Protokoll.

  Häufige Fehlerursache sind falsche Angaben des Zeitverhaltens oder sonstiger Eigenschaften der Elemente des LIN-Netzwerkes.

```
#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**

  Fehlercode für den Status der Kommunikation mit dem LIN-Boards

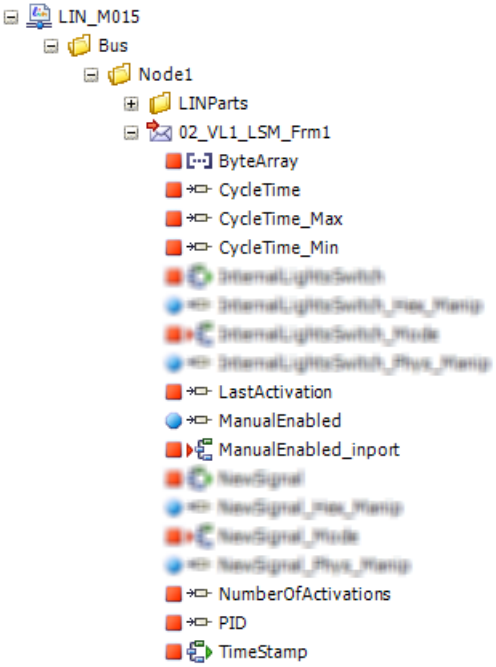
```
// status 0x00 means communication is running
#define BCI_LIN_STATUS_OVRRUN    0x01 /* data overrun occurred */
#define BCI_LIN_STATUS_ININIT    0x10 /* init mode active */
```

Nodes

Unter dem Bus-Knoten wird für jeden Knoten des LIN-Netzwerks ein eigener Node angezeigt. Sämtliche Ein- und Ausgänge zur Steuerung des LIN-Knotens oder der darin enthaltenen Elemente werden hier dargestellt.

Frames

Unter einem Node werden dessen Frames angezeigt. Die dargestellten Frames ist die Vereinigungsmenge aller Frames in allen Parts des LIN Nodes.



- **ByteArray**

  Die Payload des Frames als Byte-Array
- **CycleTime**
- **CycleTime_Max**
- **CycleTime_Min**

  Die durchschnittliche/maximale/minimale Zykluszeit des Frames in µs. Die Zykluszeit ist die Zeit zwischen zwei Aktivierungen des Frames.
- **ManualEnabled**
- **ManualEnabled_inport**

  Eine Kalibriervariable und ein Inport, mit dem der Frame während der Laufzeit des Experiments aktiviert und deaktiviert werden kann.
- **LastActivation**
- **NumberOfActivations**

  Zeitstempel der letzten Aktivierung des Frames in s und Anzahl der Aktivierungen.

  Ein Frame gilt als aktiviert, wenn der Frame vom LIN-Bus gelesen bzw. auf den LIN-Bus geschrieben wird. Zur Bestimmung von "LastActivation" werden die Prozessortakte des Realtime PC verwendet.

- **PID**

  Der Protected Identifier des Frames (wird aus der "FrameId" berechnet).

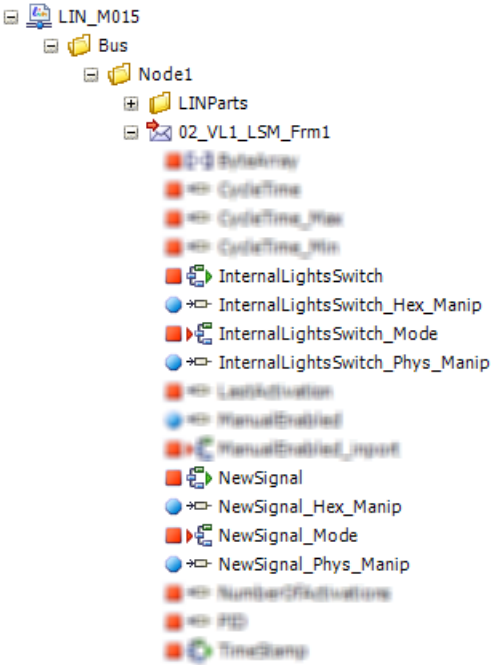
Für Receive Frames wird zusätzlich noch das Signal "TimeStamp" angelegt.

- **TimeStamp**

  Zeitstempel des letzten Empfangs des Frames in µs. Die Bestimmung von TimeStamp erfolgt durch das LIN-Board.

Signale

Unter dem Knoten eines Frames werden die Ein- und Ausgänge zur Steuerung der Signale angelegt.



- ***<SignalName>***

  Gibt den physikalischen Wert des Signals an und kann entweder den Wert des Signals auf dem LIN-Bus repräsentieren oder einen manuell angegebenen Ersatzwert.

- ***<SignalName>*_Mode**

  Auswahlmodus für die Bestimmung des Wertes von <*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**

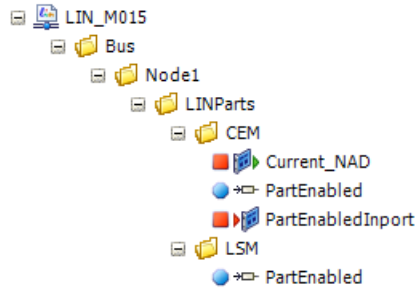
  Der elektrische manuelle Ersatzwert für <*SignalName*>

- ***<SignalName>*_Phys_Manip**

  Der physikalische manuelle Ersatzwert für *<SignalName>*

LINParts

Hier wird für jeden LIN Part ein eigener Ordner mit dem Namen des Parts angezeigt.



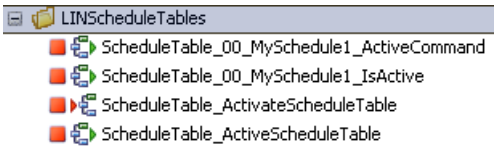
- **Current_NAD**

  Aktueller Wert der Eigenschaft „NAD“ für Parts mit aktivierter Eigenschaft „AutomaticDiagnosticRequest“.
- **PartEnabled**

  Kalibriervariable zur Aktivierung und Deaktivierung eines Parts
- **PartEnabledInport**

  Inport zur Aktivierung und Deaktivierung eines Parts

LINScheduleTables (nur für Controller im Master-Mode)



- ***<ScheduleTableIndex>*_*<ScheduleTableName>*_ActiveCommand**

  Zeigt die Position des aktuell ausgeführten Commands der Schedule Table an.
- ***<ScheduleTableIndex>*_*<ScheduleTableName>*_IsActive**

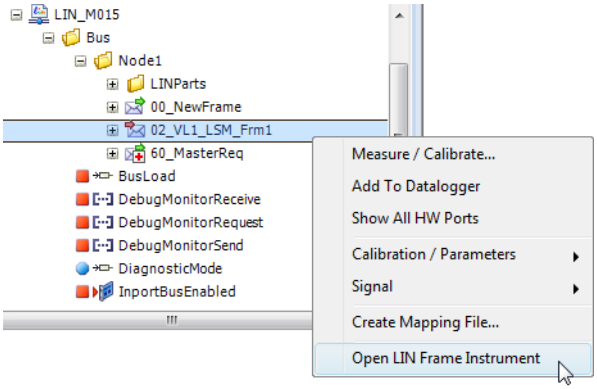
  Zeigt den aktuellen Ausführungsstatus der gewählten Schedule Table an (0 = inaktiv, 1 = aktiv).
- **ActivateScheduleTable**

  Eingabe des „Schedule Table Index“ der zu aktivierenden Schedule Table.
- **ActiveScheduleTable**

  Ausgabe des „Schedule Table Index“ der aktuell aktivierten Schedule Table.

3.7.7 Instrumente für LIN-Frames

Um ein Instrument zur Darstellung einer Message zu erstellen, rechtsklicken Sie die Message und wählen Sie **Open LIN Frame Instrument**.



Es wird ein Instrument erstellt, das den Inhalt der gewählten Message darstellt.

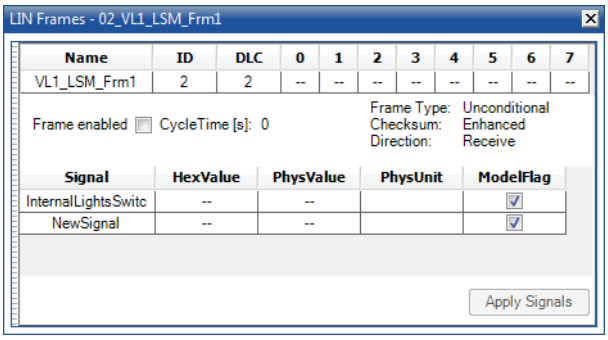


Abb. 3-20 Instrument für einen LIN-Frame

Bei einem Frame können entweder der Hex-Wert oder der physikalische Wert eines Signals (der jeweils andere Wert ist dann nicht mehr editierbar) im Bedienfenster geändert werden.

Dazu muss lediglich die Option „ModelFlag“ deaktiviert werden. Mit **Apply Signals** wird diese Änderung gültig.

3.8 NIF-Module (Network Integration FlexRay)

LABCAR-NIF V5.4.1 (Network Integration FlexRay) ist ein Modultyp in LABCAR-OPERATOR V5.4.1. Es ermöglicht ein einfaches Testen von Steuergerätefunktionen, die FlexRay-Kommunikation beinhalten.

Die gesamte FlexRay-Buskonfiguration wird aus einem Datenmodell eingelesen, das zuvor mittels einer Softwarelösung der Firma Elektrobit (EB tresos Busmirror) erstellt wurde. LABCAR-NIF V5.4.1 erstellt automatisiert den Quellcode für den zu simulierenden Restbus in Form eines NIF-Moduls. Zu diesem kann auch benutzerdefinierter Code hinzugefügt werden.

Die Signale des NIF-Moduls, welche den Signalen auf dem FlexRay-Bus entsprechen, sind im Connection Manager verfügbar und können dort mit Signalen anderer Module verbunden werden.

Hinweis

LABCAR-NIF V5.4.1 verwendet die „StringTemplate.NET“ und „ANTLR“ Bibliotheken – bitte beachten Sie die entsprechenden Lizenzbedingungen, die Sie in „Appendix“ auf Seite 383 finden.

Hardwareanforderungen

Damit Sie FlexRay-Bussimulationen durchführen können, benötigen Sie auf dem Simulationstarget Real-Time PC ein oder mehrere PCI-Boards vom Typ EB 5100 oder EB 5200 der Firma Elektrobit Automotive. Dabei ist auch ein Mischbetrieb möglich – pro NIF-Modul kann jeweils ein Board angesprochen werden.

Sie können dieses FlexRay Interface auch bei ETAS bestellen – die Bestelldaten sind wie folgt:

Bestellname	Bestellnummer
EB 5100 Elektrobit FlexRay Interface Solution	F-00K-106-407
EB 5200 Elektrobit FlexRay Interface Solution	F-00K-108-467

Softwareanforderungen

Für die Erzeugung der zu importierenden Dateien zur Restbussimulation wird eine Lizenz der Software „EB tresos Busmirror“ (4.6.x und höher) von Elektrobit benötigt.

Hinweis

Aufgrund der Umstellung des ETAS RTPC Betriebssystems auf 64 Bit werden Versionen < 4.6 nicht mehr unterstützt. Bei der Migration älterer LABCAR-OPERATOR-Projekte in die aktuelle Version werden auf nicht mehr unterstützten EB tresos Busmirror-Versionen basierende NIF-Module nicht geändert. Diese Module können dann nicht mehr editiert oder auf dem Real-Time PC ausgeführt werden und müssen durch neue NIF-Module ersetzt werden.

3.8.1 Erstellen der benötigten Dateien mit EB tresos Busmirror

Für die Integration der FlexRay-Buskonfiguration in LABCAR-OPERATOR muss das Datenmodel sowie die auf den Boards EB5100 oder EB5200 ausführbaren Binärdateien aus einem EB tresos Busmirror-Projekt importiert werden.

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

Einzelheiten dazu, wie diese Dateien erstellt werden, entnehmen Sie bitte der Dokumentation zu EB tresos Busmirror.

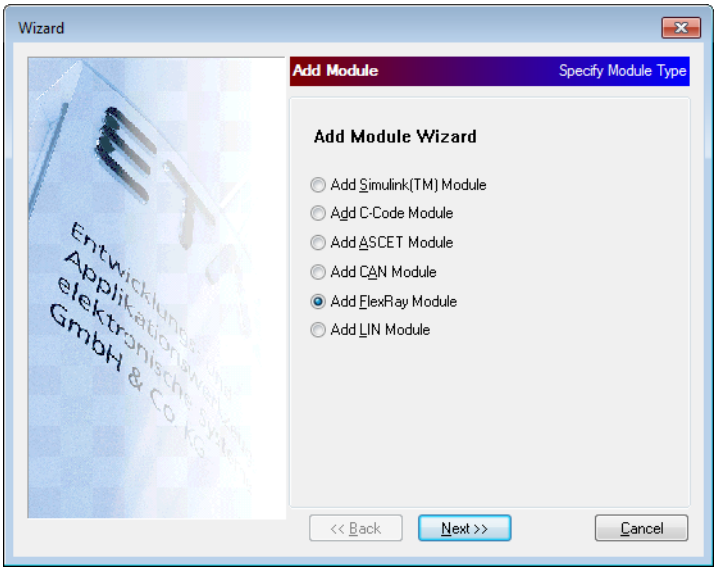
3.8.2 Integration des NIF-Moduls in das LABCAR-OPERATOR-Projekt

Nachdem mit EB tresos Busmirror das Datenmodell und die ausführbaren Binärdateien erzeugt wurden, können diese in das LABCAR-OPERATOR-Projekt integriert werden. Gehen Sie dazu wie folgt vor:

NIF-Modul erstellen

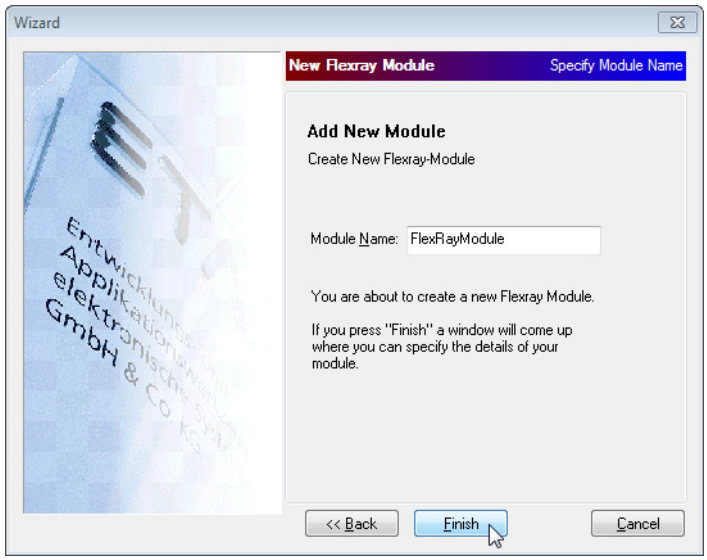
- Öffnen Sie das LABCAR-Projekt, in dem Sie das NIF-Modul erstellen wollen.
- Wählen Sie im Project Explorer das Target.
- Wählen Sie Im Kontextmenü **Add Module**.

  Der Wizard wird geöffnet.



- Wählen Sie **Add Flexray Module**.
- Klicken Sie **Next**.

- Geben Sie im nächsten Fenster einen Namen für das zu erstellende Modul ein.

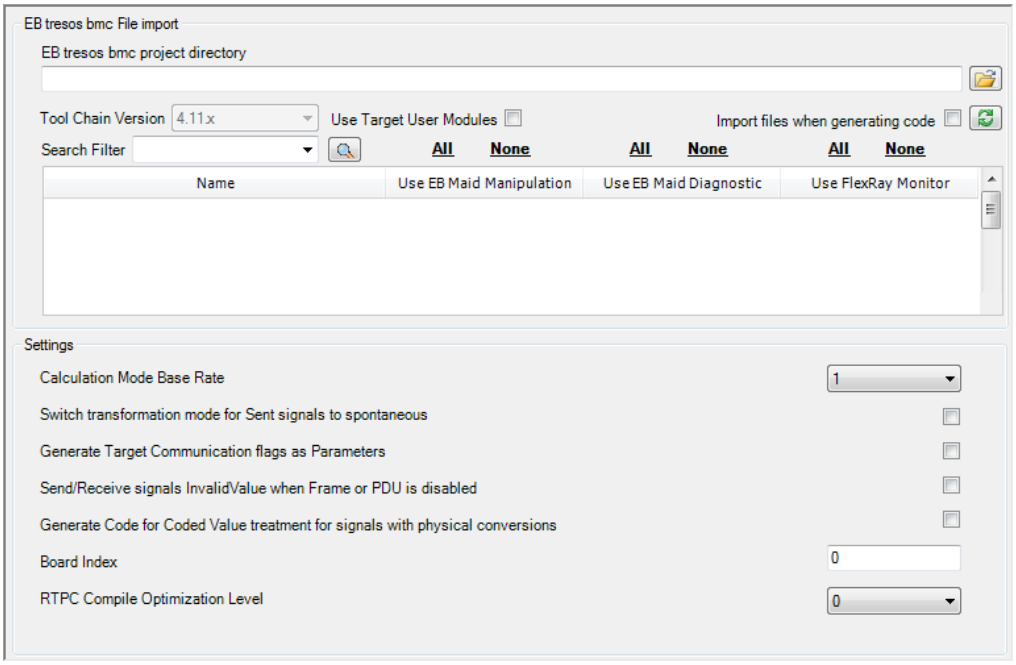


- Klicken Sie **Finish**.

  Im Hauptfenster von LABCAR-IP wird ein neues Register erstellt, in dem Sie den Import der nötigen Dateien durchführen können.

3.8.3 Anbindung an die EB tresos bmc Toolkette

Das Fenster eines NIF-Moduls unterteilt sich in die zwei Bereiche „EB tresos bmc File import“ und „Settings“.



- Wählen Sie im Feld „EB tresos bmc project directory“ das EB tresos Projekt aus, dessen Datenmodel und ausführbaren Binärdateien Sie in das NIF-Modul importieren wollen.
- Speichern Sie das NIF-Modul.

Synchronisieren der EB tresos bmc Dateien und Generierung des NIF-Modul Quellcodes

Die Generierung des Quellcodes wird immer dann ausgeführt, wenn das Modul gespeichert wird oder wenn die externen Dateien synchronisiert werden. Die Generierung des Quellcodes für das NIF-Modul greift grundsätzlich auf Arbeitskopien der importierten EB tresos bmc-Dateien zu. Mittels eines Synchronisationsmechanismus können diese externen Dateien erneut in das NIF-Modul importiert werden.

Dieser Vorgang kann automatisiert werden:

- Die automatische Synchronisation wird durch die Option „Import files when generating code“ aktiviert.
  - Aktiviert: Die Dateien werden immer vor einer erneuten Codegenerierung synchronisiert. Die Codegenerierung erfolgt dann basierend auf den synchronisierten Dateien.

– Deaktiviert: Die Dateien werden nur manuell synchronisiert. Die manuelle Synchronisierung und die anschließende Codegenerierung werden manuell über die Schaltfläche ausgeführt.

Hinweis

Wird in dem NIF-Modul manuell erstellter User Code verwendet, wird dieser erst bei Ausführung der Codegenerierung in den NIF-Modul-Quellcode integriert.

Integration von Target User Modules (EB MAID)

EB tresos bmc ermöglicht es, über sog. Target User Modules (TUM) zusätzliche Funktionalität in die ausführbare Binärdatei zu integrieren. Die Schnittstellen dieser TUM sind in sog. TUM-Descriptions hinterlegt. Mittels dieser TUM-Descriptions kann zusätzlicher Quellcode für das NIF-Modul generiert werden, welcher eine Kommunikation mit den TUM zur Laufzeit ermöglicht.

- Aktivieren Sie die Unterstützung von TUMs durch die Option „Use Target User Modules“.

Hinweis

Ist dieser Schalter deaktiviert, wird die TUM-Description ignoriert und es erfolgt auch keine Codegenerierung für die EB MAID Unterstützung.

3.8.4 Modulkonfiguration

Das Fenster eines NIF-Moduls ist in die zwei Bereiche „EB tresos bmc File import“ und „Settings“ unterteilt. Bearbeiten Sie im Feld „Settings“ die jeweiligen Optionen und speichern Sie das NIF-Modul.

Anpassen der Taktzeit des NIF-Moduls auf dem Realtime PC

Die Taktzeit der SendReceive-Task beträgt grundsätzlich 1 ms. Die zum vollständigen Signalaustausch zwischen NIF-Modul und Applikation auf dem Board notwendigen Aufrufe werden dabei gleichmäßig auf fünf sog. Calculation Slots verteilt.

In jeder Ausführung der SendReceive-Task wird dabei ein einzelner Calculation Slot ausgeführt, wobei dessen Anteile am Gesamtaustausch zwischen NIF-Modul und Board synchronisiert werden. Zusätzlich werden die Aufrufe zum Austausch der Signale einer einzelnen PDU noch mit der FlexRay Cycle Repetition der zugehörigen FrameTriggerings moduliert, wodurch eine möglichst gleichmäßige Auslastung des Real-Time PC und des Boards erreicht wird.

Dies bedeutet, dass der vollständige Signalaustausch nach insgesamt 5 ms vollendet wird. Durch Änderungen der Taktzeit kann entweder eine Performanzoptimierung erreicht werden (Taktzeit > 1 ms) oder eine Steigerung der Synchronisierung (Taktzeit < 1 ms).

- Wählen Sie im Feld „Settings“ unter „Calculation Mode Base Rate“ den Teiler für die Periode.

  Mögliche Werte sind: 1/8, 1/4, 1/2. 1, 2, 4 und 8.

  1/8 = Steigerung der Performanz (effektiv 8 ms)

  8 = Steigerung der Synchronisierung (effektiv 125 µs)

Hinweis

Die tatsächliche Ausführung der Lese.- und Sendekommunikation für einen Frame oder eine PDU erfolgt in ganzzahligen Vielfachen der Taktzeit des NIF-Moduls. Die jeweilige Taktzeit eines Frames oder einer PDU wird dabei dem zugrundeliegenden Datenmodell entnommen.

Optimierung der Performanz und Vermeidung konkurrierender Signalwerte

Um die Aufrufe der Funktionen zum Signalaustausch von Sendesignalen zu minieren, werden diese bei Bedarf nur dann ausgeführt, wenn sich der Wert des zu sendenden Signals seit dem letzten Aufruf geändert hat. Dadurch kann eine Optimierung der Performanz erreicht werden.

- Wählen Sie diese Option durch Aktivieren von „Switch transformation mode for Sent signals to spontaneous“.

Generierung der Target Communication Flags als Parameter

Zur Steuerung der Kommunikation für Frames bzw. PDUs oder der Buskommunikation werden pro Frame bzw. PDU ein Steuersignal „Enable“ und „Idle“ sowie für die Buskommunikation „CommEnable“ und „HardBoundaries“ generiert. Mit dieser Option kann gewählt werden, ob diese Signale als Inports oder als Parameter generiert werden.

Die Signale für die Ausgabe des Zeitstempels der letzten Änderung einer Receive-PDU werden grundsätzlich als Outport angelegt.

Signal InvalidValue bei deaktivierter Payload

Ist ein Frame oder eine PDU über das Signal „Enable“ zur Laufzeit deaktiviert, kann über diese Option gesteuert werden, ob in diesem Fall für jedes Signal der „InvalidValue“ aus dem FlexRay-Datenmodell als Ersatzwert gesendet oder empfangen werden soll.

Unterstützung symbolischer Werte bei Signalen mit Skalierungen

Verfügt ein FlexRay-Signal über eine Skalierung zur Umrechnung des übertragenen Rohwertes in physikalische Größen, werden häufig auch sogenannte „symbolische Werte“ verwendet (z.B. um einen Fehlerzustand zu übertragen). Diese symbolische Werte dürfen dabei nicht der Skalierung unterworfen werden.

Um nun einen skalierten physikalischen Wert von einem unskalierten symbolischen Wert unterscheiden zu können, können zusätzliche Inports und Outports generiert werden, mit denen dieses Verhalten zur Ausführungszeit gesteuert werden kann.

- Um diese Codegenerierung auszuführen, aktivieren Sie die Option „Generate Code for Coded Value treatment for signals with physical conversions“.

Unterstützung mehrerer EB5100 oder EB5200 in einem Real-Time PC

Es können auch mehrere unabhängige LABCAR-NIF-Module erzeugt werden – die Identifikation der jeweiligen EB5100 oder EB5200 (innerhalb der Verzeichnisstruktur des LABCAR-OPERATOR-Projektes) erfolgt über den Index des Boards.

- Geben Sie den Index unter „Board Index“ ein.

Compile Optimization Level

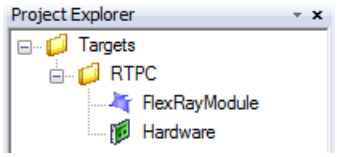
Mit dem Schalter „RTPC Compile Optimization Level“ kann der Optimierungsgrad der Build-Umgebung auf dem Real-Time PC bestimmen. Diese Option entspricht dem Optimierungsgrad welcher über das Webinterface des Real-Time PC eingestellt werden kann, betrifft jedoch nur den generierten Quellcode des NIF-Moduls.

3.8.5 Anbindung des NIF-Moduls in das LABCAR-OPERATOR Projekt

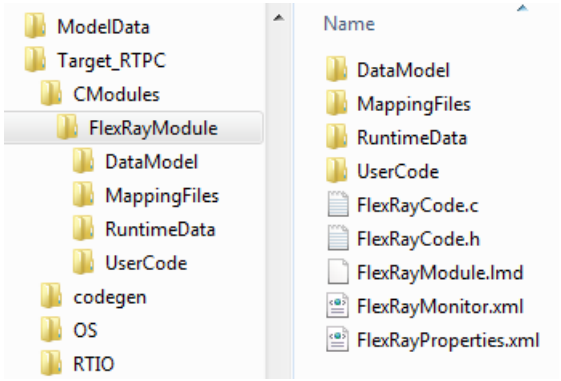
Modulkonfiguration speichern

- Wählen Sie **File** → **Save**.

  Die Modulkonfiguration wird abgespeichert und der Quellcode des Moduls generiert. Das Modul wird dann in das LABCAR-OPERATOR Projekt integriert und im Project Explorer dargestellt.



  Nach erfolgreicher Codegenerierung wird für das NIF-Modul ein Verzeichnis mit folgenden Dateien und Verzeichnissen angelegt.



OS Configuration und Connection Manager aktualisieren

- Wechseln Sie in das Register „OS Configuration“.
- Klicken Sie **Update Processes**.

  Die für das NIF-Modul erstellten Tasks und Prozesse werden angezeigt.
- Wechseln Sie in das Register „Connection Manager“.
- Klicken Sie **Update Ports**.

  Alle Inputs (Send-Payloads) und Outputs (Receive-Payloads) des Datenmodells werden angezeigt.

Modul kompilieren

- Wählen Sie **Project** → **Build**

oder

- Klicken Sie das Symbol **Build LABCAR Project**.

3.8.6 Benutzerdefinierter Code und Anpassungen

Als Erweiterung zu dem aus dem Datenmodell automatisiert generierten NIF-Modul Quellcode kann manuell erstellter Quellcode integriert werden. Die einzufügenden Code-Fragmente werden per Platzhalter an den entsprechenden Stellen des generierten Codes injiziert.

Dateien für Codefragmente

Das NIF-Modulverzeichnis enthält einen Unterverzeichnis zur Integration von manuell erstelltem Quellcode:

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

Jede Datei in diesem Verzeichnis mit Dateiendung `.fragment.c` wird als Codefragment ausgewertet. Das Format dieser Dateien ist wie folgt:

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

Hinweis

Wird in dem NIF-Modul manuell ersteller User Code verwendet, wird dieser erst bei Ausführung der Codegenerierung in den NIF-Modul Quellcode integriert.

Code Location Indicator

Der Beginn eines Codefragments wird durch die Angabe des Code Location Indicator angegeben. Der Code Location Indicator wird in eckigen Klammern eingeschlossen und als Kommentarzeile im Quellcode angegeben.

Nach dem Code Location Indicator folgt der zu integrierende Quellcode, der vom Codegenerator in das NIF-Modul eingefügt wird. Der Code endet vor dem nächsten Code Location Indicator – das Format des Code Location Indicators folgt einer der folgenden Varianten:

Spezifische Code Location Indicators mit globalem Kontext

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

Dabei ist `CodePoint` eine Textmarke im generierten Quellcode, an der der User Code eingefügt wird. Es gibt globale Textmarken, die nur einmal im generierten Quellcode vorkommen. Für diese müssen dann keine weiteren Informationen zur Angabe des Ortes angegeben werden.

Gültige Textmarken für `CodePoint` in dieser Variante sind:

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

Spezifische Code Location Indicators mit PDU- und Signal-Kontext

Für andere Textmarken gibt es mehrere Instanzen, z.B. pro Payload oder pro Signal. Für diese Textmarken müssen weitere Identifikatoren angeben werden, um den Kontext sowie das Objekt anzugeben, auf den sich der Code Location Indicator bezieht:

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

Gültige Textmarken für `CodePoint` in dieser Variante sind:

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

Der `context` der Textmarke kann sich auf folgende Vorgänge im Programmablauf beziehen:

```
SendSignal
ReceiveSignal
SendPdu
ReceivePdu
```

Abschließend muss noch das `object` angegeben werden, auf das sich die Textmarke bezieht. Als Angabe des `object` können Namen von Signalen bzw. PDUs verwendet werden.

Hinweis

Eine Auflistung aller Namen wird in den Dateien `SignalList.txt` und `PduList.txt` im `UserCode` Verzeichnis generiert. Sämtliche gültigen Code Location Indicators werden bei der Codegenerierung als kommentierte Textmarken in den NIF-Modul Quellcode eingefügt.

Spezifische Code Location Indicators mit TUM-Kontext

Für Objekte, die zu unterschiedlichen Zeiten unterschiedliche Funktionen ausführen, kann eine weitere Stufe der Verfeinerung verwendet werden.

Derzeit wird dies nur für EB Maid Target User Modules mit Senderichtung „ToHost" unterstützt.

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

Gültige Textmarken für CodePoint in dieser Variante sind:

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

Der `context` der Textmarke kann sich derzeit nur auf folgende Vorgänge im Programmablauf beziehen:

```
ReceiveTum
```

Danach muss das `object` angegeben werden, auf das sich die Textmarke bezieht. Als Angabe des `object` muss der Name des TUM verwendet werden. Abschließend muss noch die `function` angegeben werden, die das TUM gerade ausführt.

Die folgende Tabelle zeigt die gültigen Kombinationen von `codepoint`, `context` und `function`.

		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 Gültige Kombinationen von `codepoint`, `context` und `function`

Beispiel:

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

Der angegebene Code Location Indicator weist auf Quellcode, der nach der Deklaration der globalen Variablen des NIF-Moduls eingefügt wird. Im obigen Beispiel wird eine Variable `counter` deklariert.

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

Der angegebene Code Location Indicator weist auf Quellcode, der vor dem Senden der angegebenen PDU ausgeführt wird. Der User Code inkrementiert dabei dir zuvor deklarierte Variable `counter`.

Code Location Indicator und Programmfluss

In diesem Abschnitt wird beschrieben, an welcher Stelle des Programmflusses Code Location Indicators eingefügt werden. Namen in spitzen Klammern `<text>` weisen auf entsprechende Platzhalter hin.

Globaler Kontext:

```
<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 Kontext:

```
//[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 Kontext:

```
//[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 Kontext:

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

ReceivePDU Kontext:

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

Benutzerdefinierte Parameter, Messgrößen, Inports und Outports

Um auf in benutzerdefiniertem Code berechnete Größen zugreifen zu können, besteht die Möglichkeit, das NIF-Modul um zusätzliche Parameter und Ports zu erweitern. Diese werden dann über die Experimentierumgebung (EE) zugänglich gemacht.

Parameter:

Parameter werden in folgender Datei definiert:

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

Ein Parameter wird durch eine Zeile definiert:

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

Gültige Werte für <type> sind

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

Es ist auch möglich, hierarchische Variablen zu deklarieren, indem zwei Unterstriche als Separatoren verwendet werden. So erzeugt beispielsweise die Definition

```
real64 setting__var = 42.0;
```

den Parameter

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

und initialisiert diesen mit dem angegebenen Wert.

Hinweis

Werden Namen von Signalen oder PDUs aus dem Datenmodell des EB tresos BMC verwendet, werden darin enthaltene doppelte Unterstriche nicht als Hierarchieebenen interpretiert!

Der Zugriff auf den Parameter im Quellcode geschieht über die Variable `<name>`.

Messgrößen:

Messgrößen werden in folgender Datei definiert:

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

Eine Messgröße wird durch eine Zeile definiert:

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

Inports:

Inports werden in folgender Datei definiert:

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

Ein Inport wird durch eine Zeile definiert:

```
TPortObj <Modulname>_UserCode_inport_<name>;
```

Der Zugriff auf den Inport im Quellcode geschieht über die Variable <name>. Inports können wie Variablen auch hierarchisch deklariert werden.

Outports:

Outports werden in folgender Datei definiert:

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

Ein Outport wird durch eine Zeile definiert:

```
TPortObj <Modulname>_UserCode_outport_<name>;
```

Der Zugriff auf den Outport im Quellcode geschieht über die Variable <name>. Outports können wie Parameter und Messgrößen hierarchisch deklariert werden.

Beispiel:

Das folgende Beispiel soll zeigen, wie benutzerspezifischer Code eingesetzt werden kann, um das NIF-Modul um Funktionalität zu erweitern.

Es soll der Wert eines bestimmten Signals

```
sint32 water_temperature
```

vom Bus gelesen werden.

Anlegen einer Messgröße:

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

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

Anlegen des Codefragments:

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

Dieser Aufruf ruft den zuletzt empfangenen Wert des Signals ab und weist ihn der von uns deklarierten Variablen zu.

Da wir den Wert unmittelbar nach dem Empfang des Signals verwenden wollen, wird als CodePoint `afterReceive` gewählt. Da das Signal vom Bus gelesen wird, lautet der korrekte Context `ReceiveSignal`. Der Name des Signals, für das der Code eingefügt wurde, lautet `water_temperature`, und stammt aus der Datei `Signals.txt`.

Nach der erneuten Codegenerierung in LABCAR-IP steht die neue Messgröße in der Experimentierumgebung LABCAR-EE zur Verfügung.

Hinweis

Wird in dem NIF-Modul manuell erstellter User Code verwendet, wird dieser erst bei Ausführung der Codegenerierung in den NIF-Modul Quellcode integriert.

3.8.7 Anbindung an ETAS Bus Communication Monitor (BCM)

Der ETAS Bus Communication Monitor ermöglicht Manipulationen am Signalfluss innerhalb des NIF-Moduls, um z.B. die Werte eines Signals vor dessen Senden durch einen manuell vorgegebenen Wert zu überschrieben.

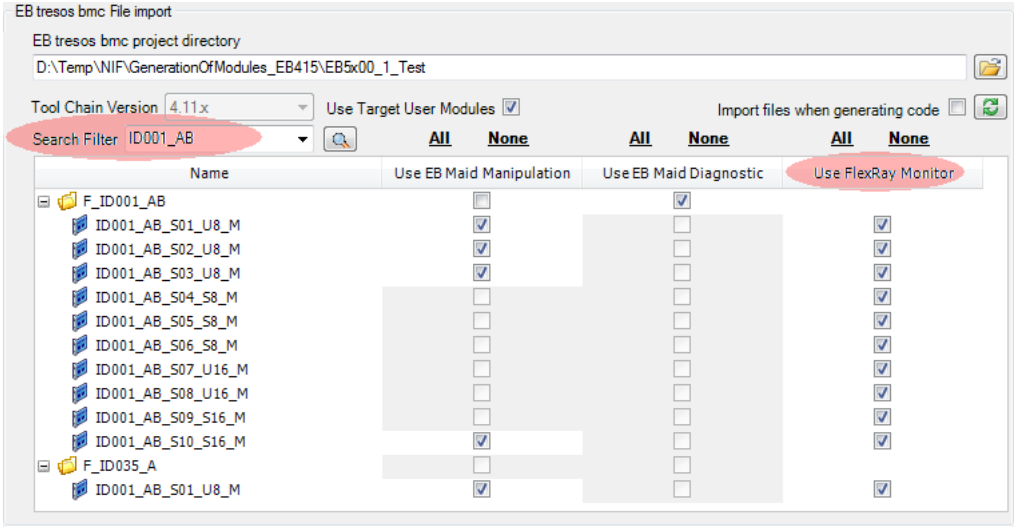
Zur Optimierung der Performanz kann die Auswahl der für die Manipulation durch den ETAS Bus Communication Monitor zu berücksichtigenden Signale für die Codegenerierung über auf der Benutzeroberfläche eingeschränkt werden.

Die Konfiguration des ETAS Bus Communication Monitor wird in einer Konfigurationsdatei gespeichert:

```
..\<Projektverzeichnis>\Target_RTPC\
        CModules\<Modulname>\FlexRayViewModel.xml
```

Signale im Bus Communication Monitor anzeigen

- Wenn Sie einzelne Signale im Bus Communication Monitor zugänglich machen wollen, müssen diese in der Benutzeroberfläche (in der Auswahl der vorhandenen Signale des Datenmodells) aktiviert werden:



Hinweis

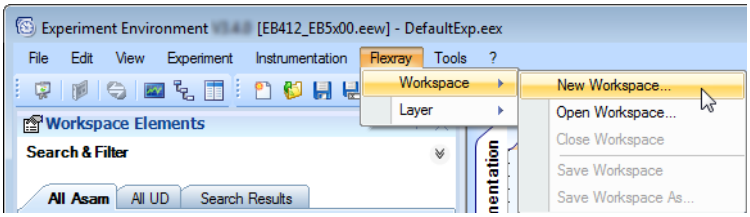
Der Bus Communication Monitor unterstützt nur skalare Signale und Signale mit einer Länge von maximal 32 Bit. ByteArray-Signale oder Signale, die länger als 32 Bit sind, werden nicht unterstützt!

- Um die Unterstützung von Bus Communication Monitor für einzelne Signale zur ermöglichen, aktivieren Sie für diese die Option „Use for FlexRay Monitor“ in der Benutzeroberfläche.
  - Durch Angabe eines regulären Ausdrucks als Suchfilter können Sie die Auswahl der angezeigten Signale einschränken.

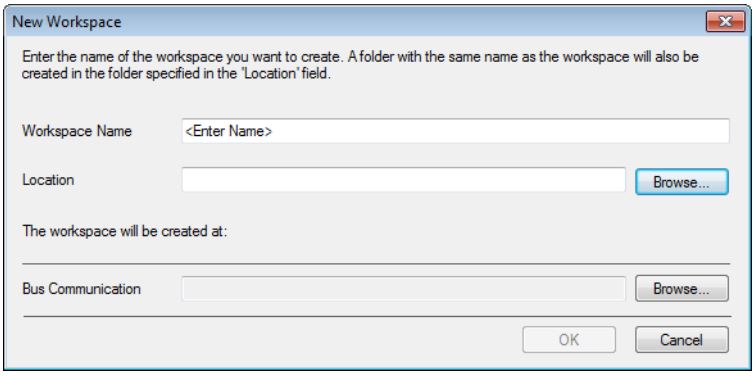
– Über die Optionen „All“ und „None“ können alle Signale, welche dem Suchkriterium genügen, aktiviert bzw. deaktiviert werden.

  Eine Aktivierung oder Deaktivierung wirkt sich auf alle Instanzen des gewählten Signals aus.

- Speichern Sie das Modul.
- Generieren Sie mit **Project** → **Build** den Code neu.
- Um die FlexRay-Elemente anzuzeigen, wählen Sie **View** → **Flexray Elements**.
- Um den Bus Communication Monitor zu öffnen, wählen Sie **View** → **Bus Communication Monitor**.
- Erstellen Sie mit **Flexray** → **Workspace** → **New Workspace** einen speziellen FlexRay Workspace.



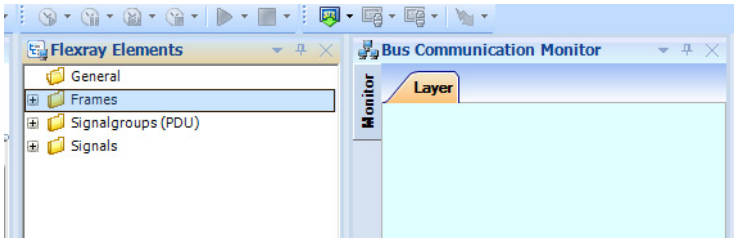
- Wählen Sie einen Namen und den Speicherort für den neuen Workspace.
- Wählen Sie unter „Bus Communication“ die o.g. Datei `FlexRayMonitor.xml`.



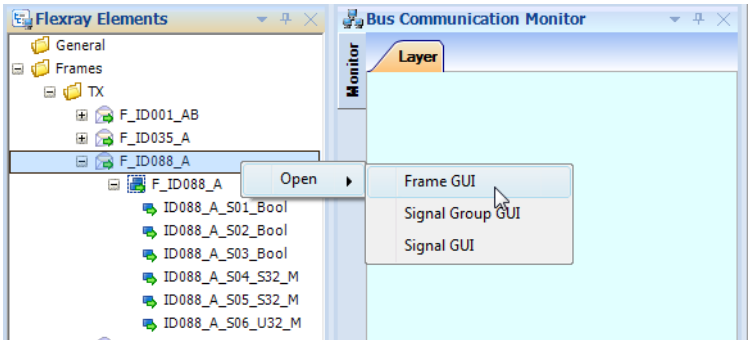
oder

- Öffnen Sie mit **Flexray → Workspace → Open Workspace** einen bereits vorhandenen Workspace (`*.bcw`).

  In den Flexray Elements werden die Frames, Signalgruppen und Signale angezeigt, die aktiviert wurden.

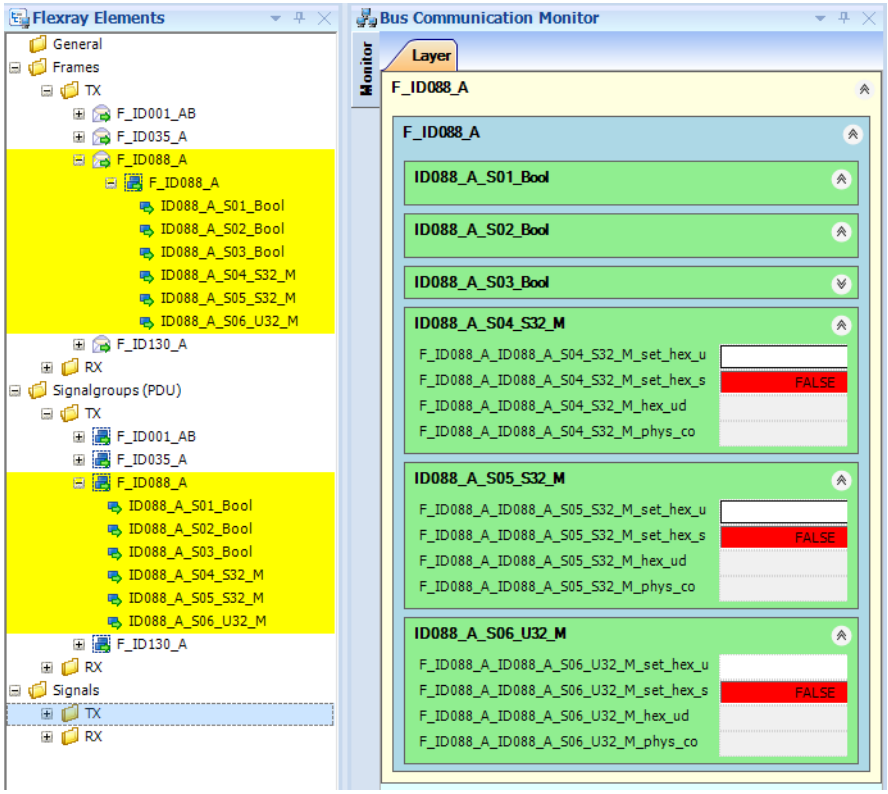


- Wählen Sie in den Flexray Elements den Frame, zu dem die im Bus Communication Monitor darzustellenden Signale gehören.
- Rechtsklicken Sie und wählen Sie **Open → Frame GUI**.



  Der Frame wird im aktuellen Layer des Bus Communication Monitors dargestellt.

- Klappen Sie die einzelnen Bereiche auf. Da nur drei Signale des Frames „freigeschaltet“ wurden, werden auch nur diese dargestellt.



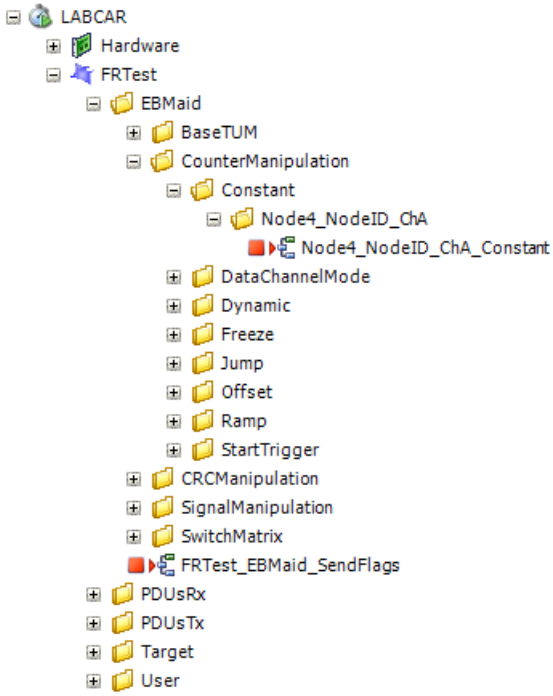
Die zur Anzeige im Bus Communication Monitor ausgewählten Signale werden in den Workspace Elements gelb hervorgehoben.

3.8.8 Target User Modules

Da bestimmte Operationen Detailinformationen über die FlexRay-Kommunikation benötigen, muss die jeweilige Funktionalität direkt auf dem Board implementiert werden – Beispiele sind Alive-Counter und Checksummenprüfungen. Die Daten beispielsweise zur Berechnung von CRCs stehen auf dem Real-Time PC nicht zur Verfügung – diese muss daher auf dem Board selbst durchgeführt werden.

Benutzerdefinierte Funktionalität auf dem Board wird in Form von sogenannten Target User Modules (TUM) implementiert. Das TUM-API bietet Funktionen zur Registrierung von Callbacks für bestimmte Ereignisse wie das Senden und Empfangen von Frames. Zudem eröffnet es Wege, Nachrichten mit dem NIF-Modul auszutauschen und dadurch zum Beispiel mit der Experimentierumgebung zu kommunizieren. Nähere Informationen zu den Target User Modules entnehmen Sie bitte der entsprechenden Elektrobit Busmirror TUM Dokumentation.

Bei aktiviertem TUM-Support werden für die in importierten TUM-Descriptions definierten Signale zur Steuerung der TUMs entsprechende Ports angelegt. Dabei werden in den Workspace Elements der EE unter jedem NIF-Modul folgende Signale angelegt:



Der relative Pfad unter diesem Knoten spiegelt die in den TUM abgebildete Funktionalität wider.

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

Die erste Ebene gibt den Namen des TUM an, wobei folgende TUM-Typen unterstützt werden:

```
SYSTEM
PROTOCOL
PROTOCOL-SAFEGUARDING
MANIPULATION
SIGNAL
```

Die zweite Ebene gibt die Funktionalität an, die in dem TUM abgebildet ist, z.B. `CounterManipulation`. Die dritte Ebene gibt den Namen des Signals an, für welches das TUM die Funktionalität anbietet, und darunter schließlich die benötigten Parameter zur Steuerung dieser Funktionalität.

TUM-Unterstützung für Signale und PDUs bearbeiten

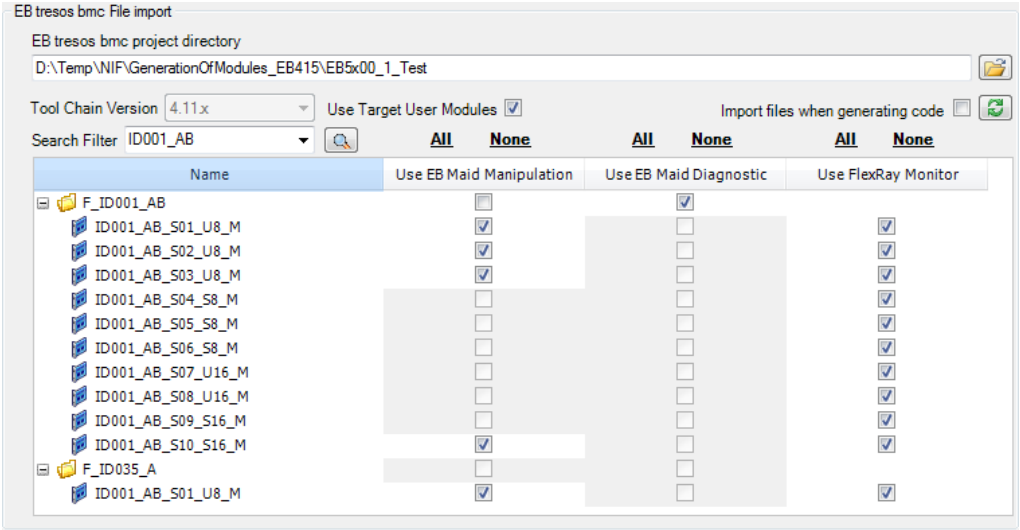
Die TUM-Unterstützung kann für einzelne Signale und PDUs des EB tresos BMC-Datenmodells aktiviert und deaktiviert werden. Dies dient zur Steigerung der Performanz und Übersichtlichkeit.

Alle TUM-Signale, die unabhängig vom Datenmodell des EB tresos BMC sind (z.B. die Signale von BaseTum oder SwitchMatrix), werden immer unterstützt, sofern die entsprechenden TUMs Teil des EB tresos BMC-Projektes sind.

- Bearbeiten Sie das EB tresos BMC-Projekt, um die anwendungsspezifische TUM-Funktionalität zum Projekt hinzuzufügen.
- Um das Datenmodell zu aktualisieren, synchronisieren Sie gegebenenfalls das NIF-Modul.
- Aktivieren Sie in der Benutzeroberfläche die Option „Use Target User Modules“.
- Um für einzelne Signale die TUM-Unterstützung zugänglich zu machen, aktivieren Sie für diese in der Benutzeroberfläche die Option „Use EB Maid Manipulation“ (Senden von Real-Time PC zu Board) oder „Use EB Maid Diagnostic“ (Empfangen von Board zu Real-Time PC).
  - Durch Angabe eines regulären Ausdrucks als Suchfilter können Sie die Auswahl der angezeigten Signale einschränken.
  - Über die Optionen „All“ und „None“ können alle Signale, die dem Suchkriterium genügen, aktiviert bzw. deaktiviert werden.

    Eine Aktivierung oder Deaktivierung wirkt sich auf alle Instanzen des gewählten Signals aus.

    Signale mit mehr als 4 Byte werden nicht als Outports weitergegeben. Für diese Signale stellt der generierte Code ein Byte Array zur Verfügung, welches durch User Code weiter verarbeitet werden kann.



Hinweis

Die TUM-Unterstützung wird nur für Signale und PDUs angeboten, die in mindestens einem TUM des EB tresos BMC-Projektes enthalten sind.

- Speichern Sie das Modul.

Wahl der TUM-Funktionalität für ein Signal

Für Signale in TUMs mit mehreren Funktionen, wie z.B. Dynamic Value, Hold, Ramp etc. kann zu jedem Zeitpunkt nur eine einzelne Funktion ausgeführt werden. Die Wahl dieser Funktion geschieht über den entsprechenden Inport unter dem Knoten

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

Der Wert dieses Inports gibt die auszuführende Funktion an. Details über gültige Werte und deren Funktionalität können der TUM-Dokumentation entnommen werden.

Der Wert 0 gibt immer den Defaultzustand der Funktion an. Diese ist `OFF`, d.h. die Funktion innerhalb des TUM wird nicht ausgeführt.

Starten und Beenden der TUM-Funktionalität für ein Signal

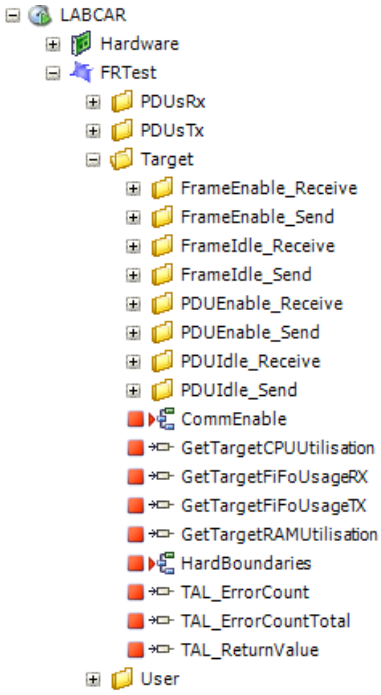
Ein TUM wird nicht zyklisch ausgeführt. Alle Parameter, die das TUM zur Ausführung benötigt, werden durch einen einzelnen Funktionsaufruf übergeben. Im Anschluss beginnt die Ausführung des TUMs, bis es durch einen weiteren Aufruf beendet wird. Um die Ausführung einer Funktion zu starten, kann folgender Inport verwendet werden:

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

Bei einem Wechsel des Wertes von 0 auf 1 wird die Ausführung der Funktion gestartet. Um eine Funktion zu beenden, muss zunächst der `DataChannel-Mode` = 0 gesetzt werden. Anschließend muss wieder ein Wechsel von 0 nach 1 im StartTrigger eingegeben werden.

3.8.9 Experimentierumgebung

In der Experimentierumgebung werden im Fenster „Workspace Elements“ (in Abhängigkeit der verwendeten EB tresos bmc Version) eine Reihe von Messgrößen, Parametern und Inports zur Kommunikation mit dem NIF-Modul angeboten.



Target

Zur Steuerung der Kommunikation stehen unter diesem Knoten diverse Signale zur Verfügung. Diese werden in Abhängigkeit der Option „Generate Target Communication flags as Variable“ als Parameter oder als Inport generiert.

- <*Modulname*>/Target/CommEnable

  Aktiviert oder deaktiviert die Buskommunikation. Wenn dieser Parameter auf 0 gesetzt wird, dann findet keine Kommunikation auf dem Bus statt.

- <*Modulname*>/Target/HardBoundaries

  Aktiviert oder deaktiviert „harte“ oder „weiche“ Grenzen. Wird dieser Parameter auf 0 gesetzt, werden die im Datenmodell beschriebenen MinVal- und MaxVal-Werte als „weiche“ Grenzen des Signals verwendet.Wird dieser Parameter auf 1 gesetzt, werden die (mittels der im Datenmodell angegebenen Bitlänge) berechneten Werte als „harte“ Grenzen des Signals verwendet. Der Signalwert wird in jedem Zyklus auf die jeweiligen Grenzwerte limitiert.

- <*Modulname*>/Target/GetTargetCPUUtilisation

  Die aktuelle Auslastung der CPU des Boards in Prozent

- <*Modulname*>/Target/GetTargetRAMUtilisation

  Die aktuelle Verwendung des RAM-Speichers des Boards in Prozent. Dabei wird nur der zur Laufzeit dynamisch allokierte Speicher berücksichtigt - der im ausführbaren Code programmierte Anteil („static memory") bleibt unberücksichtigt, so dass der Wert von „GetTargetRAMUtilisation" im Extremfall sogar 0% sein kann.

- <*Modulname*>/Target/TAL_ErrorCount

  Dieser Zähler gibt während der Simulation die Anzahl der TAL-Fehler im Laufe eines Kommunikationszyklus wieder.

- <*Modulname*>/Target/TAL_ErrorCountTotal

  Die Gesamtzahl aller Fehler aus TAL-Aufrufen seit Start der Simulation. Dieser Zähler wird zurückgesetzt durch erneutes Starten der Kommunikation.

- <*Modulname*>/Target/TAL_ReturnValue

  Der erste Fehlercode aus einem TAL-Aufruf im letzten Kommunikationszyklus. Fehlercodes zu den TAL-Funktionen können aus der TAL-Benutzerdokumentation entnommen werden.

- <*Modulname*>/Target/GetTargetFiFoUsageRX

  Die aktuelle Verwendung des Empfangs-FiFo zwischen Real-Time PC und RAM-Speicher des Boards in Prozent.

- <*Modulname*>/Target/GetTargetFiFoUsageTX

  Die aktuelle Verwendung des Sende-FiFo zwischen Real-Time PC und RAM-Speicher des Boards in Prozent.

Target/FrameEnable_Receive/Send

Zur Aktivierung und Deaktivierung der Übertragung von Frames zwischen NIF-Modul und FlexRay-Bus stehen unter diesem Knoten diverse Signale zur Verfügung.

- <*Modulname*>/Target/FrameEnable_Send/
  FrameEnable_Send_<*Framename*>

  Aktiviert oder deaktiviert das Senden des angegebenen Frames auf dem Bus. Diese Funktion wird mittels TAL-Aufrufen ausgeführt:

  – Wert = 1: Der Frame wird gesendet.

  – Wert = 0: Der Frame wird nicht mehr gesendet. Der zuletzt gesendete Wert bleibt auf dem Bus stehen oder die Signalwerte werden bei aktivierter Option „Signal InvalidValue" bei deaktivierter Payload durch den InvalidValue des Signals ersetzt.

  – Wert = -1: Ein NULL-Frame wird gesendet.

- <*Modulname*>/Target/FrameEnable_Receive/ FrameEnable_Receive_<*Framename*>

  Aktiviert oder deaktiviert den Empfang des angegebenen Frames vom Bus. Diese Funktion wird mittels TAL-Aufrufen ausgeführt:

  – Wert = 1: Der Frame wird empfangen.

  – Wert = 0: Der Frame wird nicht mehr empfangen oder die Signalwerte werden bei aktivierter Option „Signal InvalidValue“ bei deaktivierter Payload durch den InvalidValue des Signals ersetzt.

Hinweis

Sind einem Frame mehr als ein FrameTrigger zugewiesen, wird für jeden FrameTrigger ein eigenes Signal zur Steuerung generiert. Der Name des Signals wird durch das Suffix `ft_[SlotID]_[BaseCycle]_[CycleRepetion]` *des FrameTriggers erweitert.*
Zusätzlich wird ein Signal zur gemeinsamen Steuerung aller FrameTrigger des Frames mit dem ursprünglichen Namen generiert. Hat dieses gemeinsame Signal einen gültigen Wert [-1, 0, 1], so wird dessen Wert übernommen. In allen anderen Fällen werden die Werte der Steuersignale der einzelnen FrameTrigger verwendet.

Target/PDUEnable_Receive/Send

Zur Aktivierung und Deaktivierung der Übertragung von Signalen in PDUs zwischen NIF-Modul und FlexRay-Bus stehen unter diesem Knoten diverse Signale zur Verfügung.

- <*Modulname*>/Target/PDUEnable_Send/PDUEnable_Send_<*PDUname*>

  Aktiviert oder deaktiviert das Senden der angegebenen PDU vom Bus. Diese Funktion wird mittels TAL-Aufrufen ausgeführt:

  – Wert = 1: Die PDU wird gesendet.

  – Wert = 0: Die PDU wird nicht mehr gesendet. Der zuletzt gesendete Wert bleibt auf dem Bus stehen oder die Signalwerte werden bei aktivierter Option „Signal InvalidValue“ bei deaktivierter Payload durch den InvalidValue des Signals ersetzt.

- <*Modulname*>/Target/PDUEnable_Receive/ PDUEnable_Receive_<*PDUname*>

  Aktiviert oder deaktiviert den Empfang der angegebenen PDU vom Bus. Diese Funktion wird mittels TAL-Aufrufen ausgeführt:

  – Wert = 1: Die PDU wird empfangen.

  – Wert = 0: Die PDU wird nicht mehr empfangen oder die Signalwerte werden bei aktivierter Option „Signal InvalidValue“ bei deaktivierter Payload durch den InvalidValue des Signals ersetzt.

Target/FrameIdle_Receive/Send

Zur Aktivierung und Deaktivierung der Übertragung von Signalen zwischen LABCAR-OPERATOR-Projekt und NIF-Modul stehen unter diesem Knoten diverse Signale zur Verfügung.

- <*Modulname*>/Target/FrameIdle_Send/FrameIdle_Send_<*Framename*>

  Aktiviert oder deaktiviert das Übernehmen der zu sendenden Signalwerte aus dem LABCAR-OPERATOR-Projekt.

  – Wert = 0: Signalwerte des Frames werden vom Projekt übernommen.

  – Wert = 1: Signalwerte des Frames werden nicht vom Projekt übernommen. Der zuletzt übernommene Wert wird beibehalten.

- <*Modulname*>/Target/FrameIdle_Receive/
  FrameIdle_Receive_<*Framename*>

  Aktiviert oder deaktiviert das Übergeben der empfangenen Signalwerte an das LABCAR-OPERATOR-Projekt.

  – Wert = 0: Signalwerte des Frames werden an das Projekt übergeben.

  – Wert = 1: Signalwerte des Frames werden nicht an das Projekt übergeben. Die zuletzt übergebenen Signalwerte werden beibehalten.

Target/PDUIdle_Receive/Send

Zur Aktivierung und Deaktivierung der Übertragung von Signalen zwischen LABCAR-OPERATOR-Projekt und NIF-Modul stehen unter diesem Knoten diverse Signale zur Verfügung.

- <*Modulname*>/Target/PDUIdle_Send/PDUIdle_Send_<*PDUname*>

  Aktiviert oder deaktiviert das Übernehmen der zu sendenden Signalwerte aus dem LABCAR-OPERATOR-Projekt.

  – Wert = 0: Signalwerte der PDU werden vom Projekt übernommen.

  – Wert = 1: Signalwerte der PDU werden nicht vom Projekt übernommen. Der zuletzt übernommene Wert wird beibehalten.

- <*Modulname*>/Target/PDUIdle_Receive/PDUIdle_Receive_<*PDUname*>

  Aktiviert oder deaktiviert das Übergeben der empfangenen Signalwerte an das LABCAR-OPERATOR-Projekt.

  – Wert = 0: Signalwerte der PDU werden an das Projekt übergeben.

  – Wert = 1: Signalwerte der PDU werden nicht an das Projekt übergeben. Die zuletzt übergebenen Signalwerte werden beibehalten.

Target/PDUTiming_Receive

Zur Ausgabe des Zeitstempels, zu dem ein Signal der angegebenen PDU zuletzt empfangen wurde, stehen unter diesem Knoten für jede PDU je ein Signal zur Verfügung. Der Zeitstempel wird in Millisekunden ausgegeben.

Diese Signale werden unabhängig von der Einstellung „Generate Target communication Flags as Parameters“ immer als Outport angelegt.

Send/ReceiveFrames

Unter diesem Knoten stehen Inports für die zu sendenden sowie Outports für die empfangenen Signalwerte zur Verfügung. In dieser Version sind die Signale unter den jeweiligen PDUs angeordnet.

User

Unter diesem Knoten stehen in Abhängigkeit des benutzerdefinierten Codes des Moduls die benutzerdefinierten Messgrößen, Parameter, Inports und Outports zur Verfügung.

3.8.10 Erstellen eines NIF-Moduls mit dem Automation Server

Ein NIF-Modul kann auch über API-Funktionen des LABCAR-OPERATOR erstellt werden. Die API bildet alle Funktionen ab, die der Anwender über die GUI durchführen kann. Das automatisierte Einfügen von benutzerdefiniertem Code oder Anpassungen wird nicht unterstützt, sondern muss im Falle einer Automatisierung außerhalb von LABCAR-OPERATOR implementiert werden.

Es gibt eine Reihe von Set/Get-Methoden mit denen die Konfiguration eines NIF-Moduls festgelegt bzw. abgefragt werden kann. Im Fehlerfall geben diese Set Methoden „False“ und im Erfolgsfall „True“ zurück.

Die Dokumentation (chm-Datei) dazu finden Sie über **?** → **Help** unter „API Documentation“.

3.9 FiL-Module

In diesem Kapitel wird die Erstellung von FiL-Modulen beschrieben.

Function-in-the-Loop (FiL)

Ein traditionelles HiL-System (Abb. 3-21) besteht aus einem Steuergerät und einem Simulationsmodell, miteinander verbunden in einer geschlossenen Regelschleife. Dabei werden die Steuergeräteausgänge mit einer Karte zur Signalerfassung für den Simulationsrechner verbunden. Auf diesem läuft ein echtzeitfähiges Simulationsmodell, das – basierend auf den gemessenen Eingangsgrößen – die entsprechenden Ausgangsgrößen berechnet. Diese wiederum werden in elektrische Signal umgewandelt und mit den Steuergeräteeingängen verbunden.

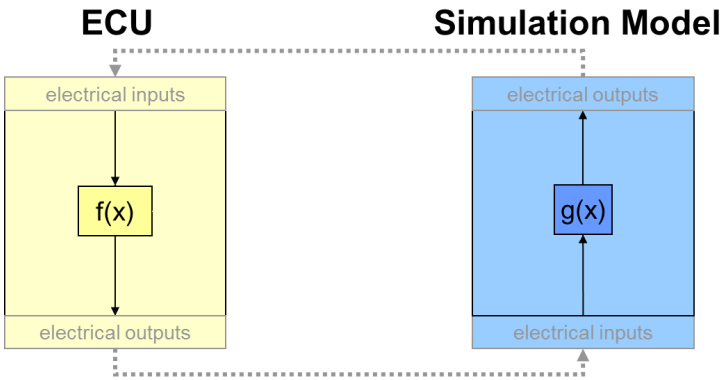


Abb. 3-21 Aufbau eines HiL-Systems

Bei einem FiL-System (Abb. 3-22) werden einige (oder alle) dieser elektrischen Verbindungen umgangen und auf den Speicher des Steuergerätes direkt zugegriffen – dazu muss das Steuergerät mit einem ETK oder XETK ausgerüstet sein[1].

Dadurch werden die Eingangs- und Ausgangsstufen als auch die Hardwareverbindungen vom und zum Simulationsmodell umgangen.

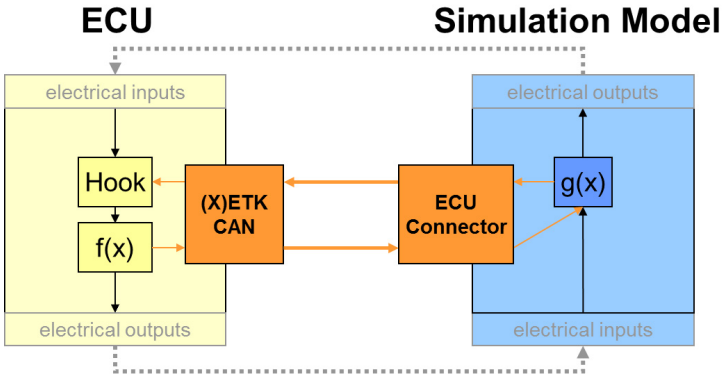


Abb. 3-22 FiL-System

1. Ein Zugriff über XCPonCAN ist ebenfalls möglich – dessen Geschwindigkeit ist für ein echtes FiL-System aber zu gering.

Zum Aufbau eines FiL-Systems wird Folgendes benötigt:

- Das Steuergerät muss mit einem ETK oder XETK ausgerüstet sein.
- Steuergeräteausgänge, die vom ETK/XETK vermessen werden, müssen zum Simulationsmodell verbunden werden.
- Eingänge für die Steuergerätefunktionen, die vom Modell stimuliert werden sollen, müssen über einen Softwarehook verfügen. Ein Hook ist ein Schalter, mit dem der Eingang einer Steuergerätefunktion vom elektrischen Eingang oder von der vorhergehenden Steuergerätefunktion getrennt werden kann und über ETK/XETK mit dem Modell verbunden werden kann.
- Detailliertes Wissen über die internen Funktionsblöcke des Steuergerätes
- Um eine ordnungsgemäße Stimulation der Funktionseingänge zu ermöglichen, muss ggf. die diagnostische Funktionalität des Steuergerätes deaktiviert werden.

Die Konfiguration des FiL-Systems erfolgt in LABCAR-IP, die Kommunikation mit dem FiL-Modul erfolgt in der Experimentierumgebung ETAS EE.

3.9.1 Konfiguration in LABCAR-IP

Die folgende Abbildung zeigt die Startseite des FiL Wizards.

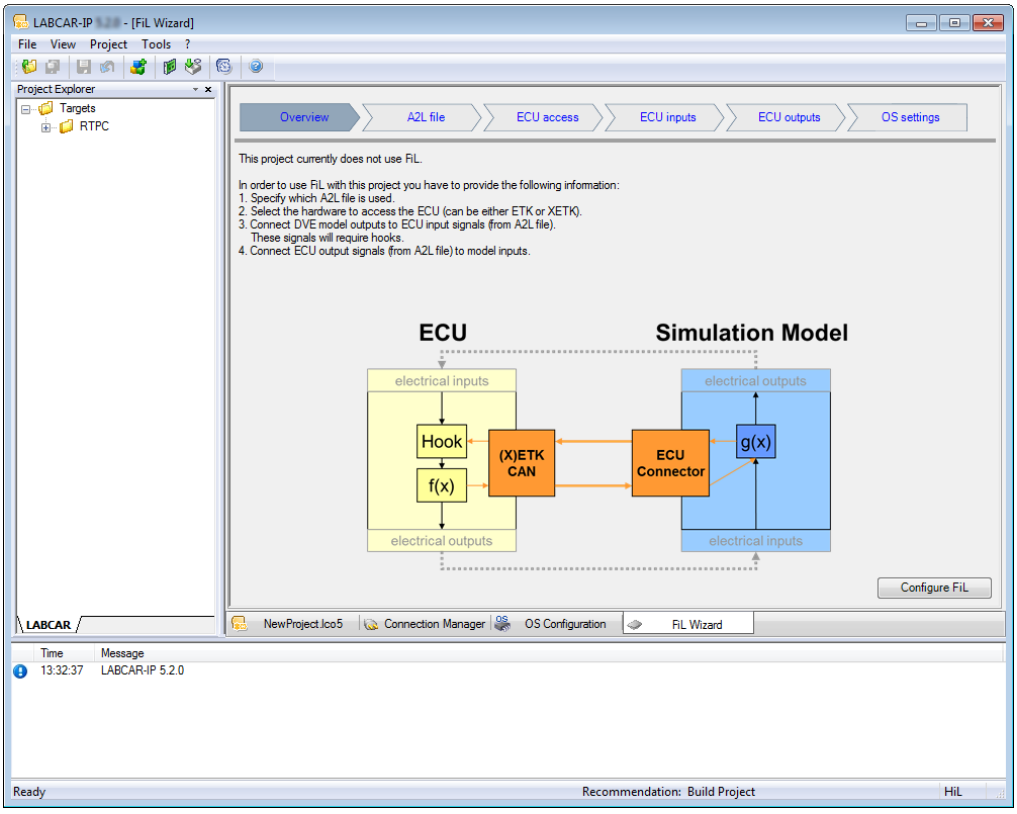


Abb. 3-23 Die Startseite des FiL Wizards

Auf dieser Seite finden Sie eine kurze Übersicht über die durchzuführenden Schritte.

1. Definition der Steuergerätedatei (siehe „A2L-Datei wählen“ auf Seite 209)
2. Definition der Verbindung vom Real-Time PC zum Steuergerät (siehe „Hardware für Steuergerätezugriff definieren“ auf Seite 210)
3. Definition der Steuergerätegrößen, die vom Modell stimuliert werden sollen (siehe „Steuergerätevariablen für die Stimulation wählen“ auf Seite 213).
4. Definition der Steuergerätegrößen, die mit Eingängen des Simulationsmodells verbunden werden sollen (siehe „Steuergeräteausgänge für die Verbindung zum Modell wählen“ auf Seite 215).
5. Zuordnung der Steuergeräteraster zu Tasks der OS Configuration (siehe „OS-Einstellungen“ auf Seite 216)

Diese Reihenfolge spiegelt sich auch in der Abfolge der Register wider:

Zustands- und Fehleranzeige

Zudem zeigt jedes Register (Symbol) auch den Zustand des jeweiligen Schrittes an (OK, Warnung oder Fehler).

Detaillierte Warn- und Fehlermeldungen während des Konfigurationsprozesses werden im Fenster „Messages“ ausgegeben und in die Log-Datei geschrieben.

3.9.2 A2L-Datei wählen

Zu Beginn der Konfiguration muss die Steuergerätebeschreibungsdatei definiert werden.

Steuergerätebeschreibungsdatei (a2l-Datei) auswählen

- Wählen Sie das Register „A2L file“.
- Klicken Sie **Browse** und wählen Sie die A2L-Datei.

  Die Datei wird geladen und ihr Inhalt (Parameter, Messgrößen und verfügbare Hooks) dargestellt.

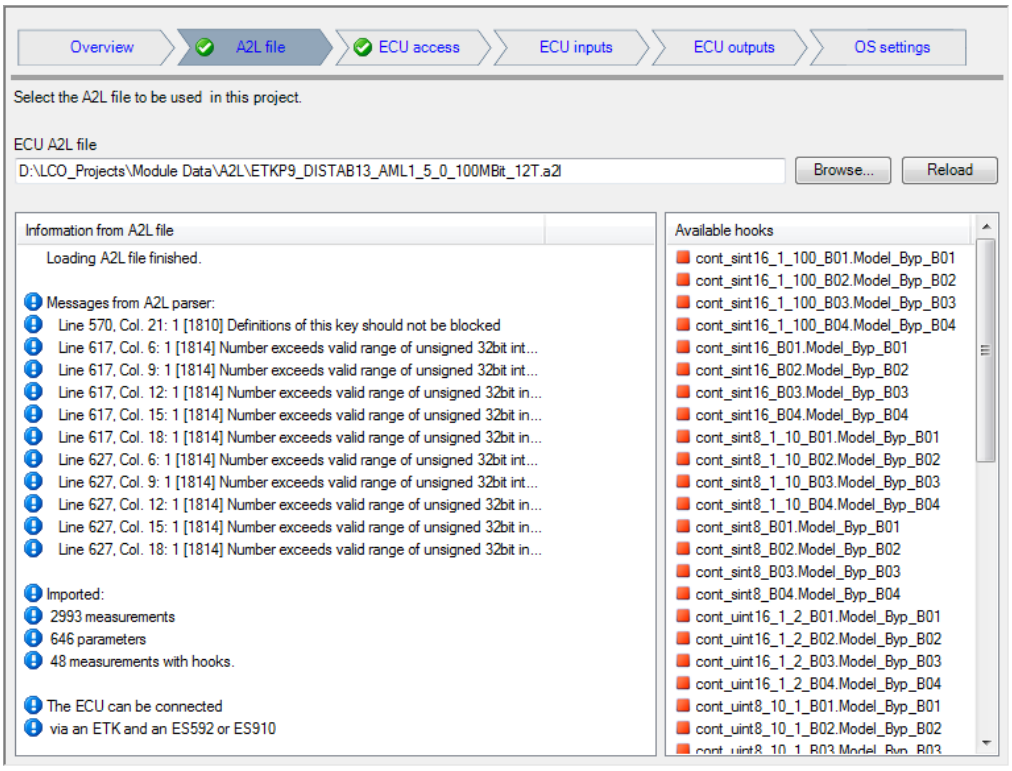


Abb. 3-24 Auswahl der A2L-Datei

Nach dem Einlesen der Datei werden folgende Informationen dargestellt:

- **Information from A2L file**

  Hier wird eine Reihe von Parser-Messages ausgegeben.

  Wenn die Datei nicht geladen werden konnte, wird im Fenster „Messages“ eine Liste aller Fehler angezeigt.

- **Available hooks**

  Hier werden alle Signale aufgeführt, für die (in der A2L-Datei) Hooks definiert sind.

Wenn die A2L-Datei verändert wurde (aber Name und Verzeichnis gleich geblieben sind), kann die Datei mit **Reload** erneut eingelesen werden.

3.9.3 Hardware für Steuergerätezugriff definieren

Im nächsten Schritt wird die Hardware definiert, über die der Steuergerätezugriff erfolgen soll:

- **XETK**

  Für ein Steuergerät mit XETK – der XETK wird per Ethernet direkt an den Real-Time PC angeschlossen.

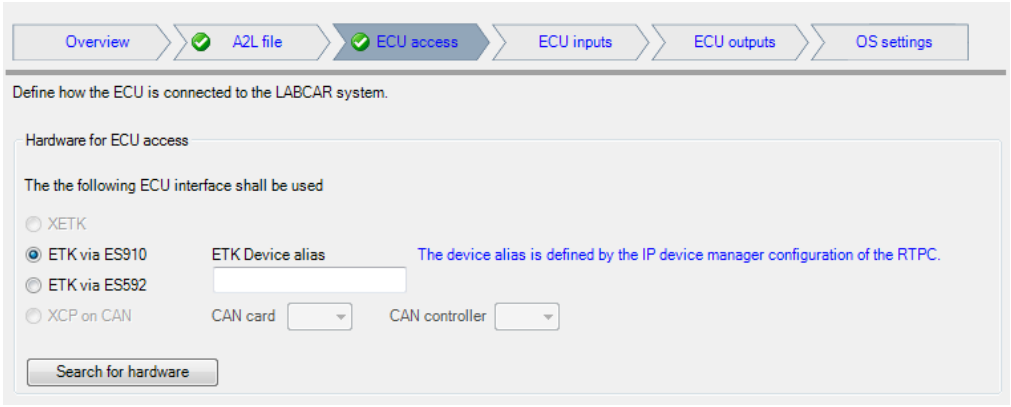
- **ETK**

  Für ein Steuergerät mit ETK – der ETK wird an eine ES592 oder ES910 angeschlossen und diese werden per Ethernet mit dem Real-Time PC verbunden.

- **XCP on CAN**

Hardware wählen

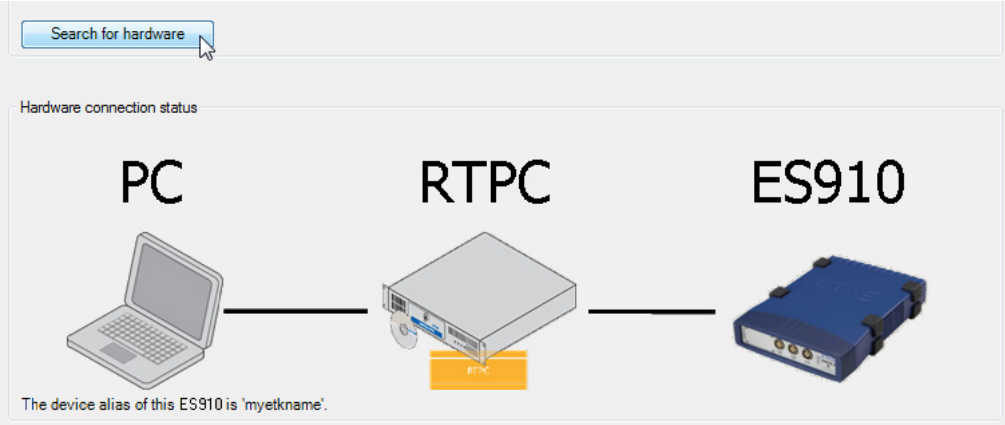
- Gehen Sie zum Register „ECU access“.
- Wählen Sie die Hardware für den Steuergerätezugriff.



- Geben Sie bei Zugriff über ES910 oder ES592 den „ETK Device Alias“ an (siehe „ETK Device Alias“ auf Seite 211).

- Um zu prüfen, ob die gewählte Hardware auch angeschlossen ist, klicken Sie **Search for hardware**.

  Wenn die gewählte Hardware vorhanden ist, wird Sie mit einem Symbol dargestellt (und ggf. den ausgelesenen Device Alias).



Hinweis

Die Hardwaresuche ist jeweils beschränkt auf die im Feld „Hardware for ECU access" getroffene Auswahl. Andere angeschlossene Geräte werden nicht angezeigt!

ETK Device Alias

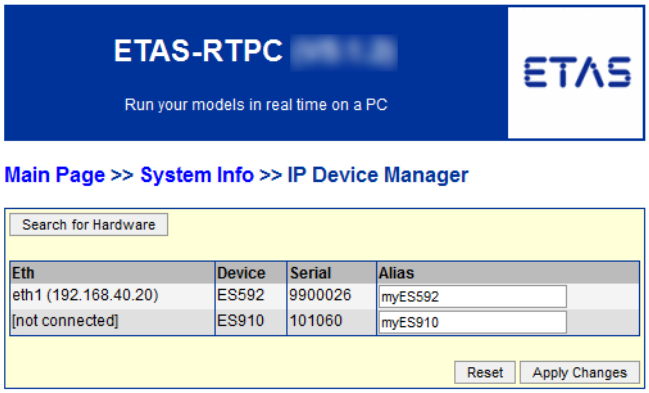
Der „ETK Device Alias" ersetzt die „harte Verdrahtung" mit der IP-Adresse der angeschlossenen ES592 oder ES910 und erleichtert damit die Portierung von Projekten.

Hinweis

Beim Zugriff über ETK muss ein „ETK Device Alias" hier angegeben werden (und vorher in ETAS RTPC definiert werden)!

Device Alias in ETAS RTPC vergeben

- Klicken auf den Link öffnet eine Browser-Instanz und in dieser die Seite „IP Device Manager“ des Web-Interfaces von ETAS RTPC.



- Klicken Sie **Search for Hardware**, um nach Ethernetverbindungen zu suchen, die für die Echtzeitkommunikation („rtudp_*n*“) konfiguriert wurden.

  Nach erfolgreichem Scan werden die folgenden Informationen angezeigt:

  – **Eth**

  Der Ethernetadapter des Real-Time PC, an dem die Hardware angeschlossen ist.

  – **Device**

  Der Gerätetyp

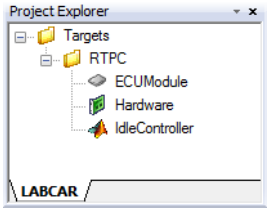
  – **Serial**

  Die Seriennummer des Gerätes

- Geben Sie unter **Alias** einen Identifier ein (C-konform, max. 32 Zeichen).
- Wählen Sie **Reset**, um gemachte Änderungen zurückzusetzen.
- Um die Eingabe zu übernehmen, wählen Sie **Apply Changes**.
- Verlassen Sie das Web-Interface und kehren Sie zu LABCAR-IP zurück.

Projekt speichern

- Speichern Sie die Konfiguration mit **File** → **Save**.
- Das FiL-Modul wird gespeichert und im Project Explorer dargestellt („ECUModule“).



3.9.4 Steuergerätevariablen für die Stimulation wählen

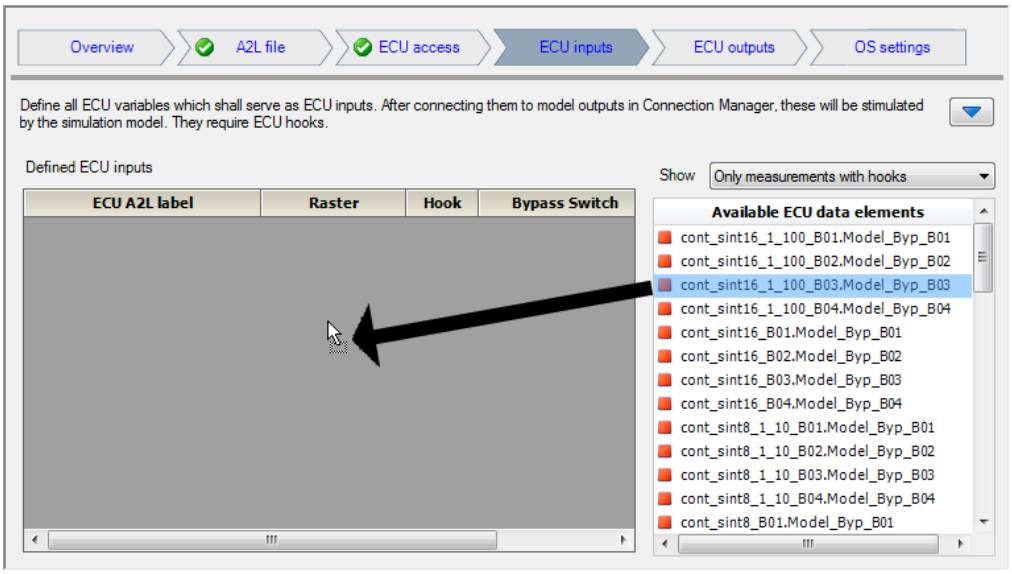
In diesem Schritt definieren Sie die Steuergerätevariablen, die als Eingänge fungieren sollen. Diese werden von Ausgängen des Simulationsmodell stimuliert und benötigen hierfür Steuergeräte-Hooks.

Diese Hooks trennen den jeweiligen Funktionsblock von den Ausgängen des Blocks davor (oder sogar vom Steuergeräteeingang) und verbinden diesen mit den vom angeschlossenen ETK/XETK zur Verfügung gestellten Eingangsgrößen. Die Hooks sind damit die Schalter, mit den die externe Bypassfunktion aktiviert bzw. deaktiviert wird.

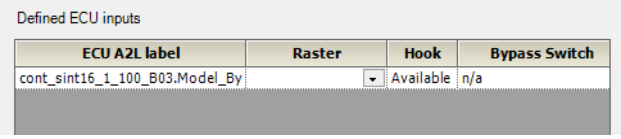
Eingänge auswählen

- Wählen Sie das Register „ECU inputs“.
- Wählen Sie als Filter („Show“) **Only measurements with hooks**.
- Wählen Sie in der Liste „Available ECU data elements“ eine Messgröße

- Verschieben Sie diese per Drag & Drop in die Liste links davon.



Die Messgröße wird in die Liste übernommen.



Messgröße aus Liste entfernen

- Um eine Messgröße wieder aus der Liste zu entfernen, rechtsklicken Sie diese und wählen Sie **Delete**.

Bedeutung/Funktion der einzelnen Spalten

- **ECU A2L label**

  Das Label des Datenelements
- **Raster**

  Hier muss das Steuergeräteraster gewählt werden, in dem das Datenelement erfasst wird.
- **Hook**

  Hier wird angezeigt, ob für dieses Datenelement ein Hook verfügbar ist.
- **Bypass Switch**

  Ein Parameter, mit dem der Bypass aktiviert/deaktiviert werden kann. Dieser kann per Drag & Drop aus der Liste „Available ECU data elements“ (Filter: „Characteristics“) gewählt werden.

- **Initial Value**

  Startwert für den Parameter in der Spalte „Bypass Switch“.
  In der Experimentierumgebung gibt es ein Instrument zur Bedienung der Kommunikation, das eine Funktion (siehe „Apply bypass switch settings“ auf Seite 220) besitzt, mit der alle Bypasses mit ihrem jeweiligen Startwert aktiviert werden können.

Die hier gewählten Datenelemente sind nach dem Speichern im Connection Manager als Inputs verfügbar (falls nicht, klicken Sie im Connection Manager **Update Ports**).

3.9.5 Steuergeräteausgänge für die Verbindung zum Modell wählen

In diesem Schritt wählen Sie Datenelemente des Steuergerätes aus, die später zur Messung mit Modelleingängen verbunden werden können.

Die folgende Abbildung zeigt das Register „ECU outputs“.

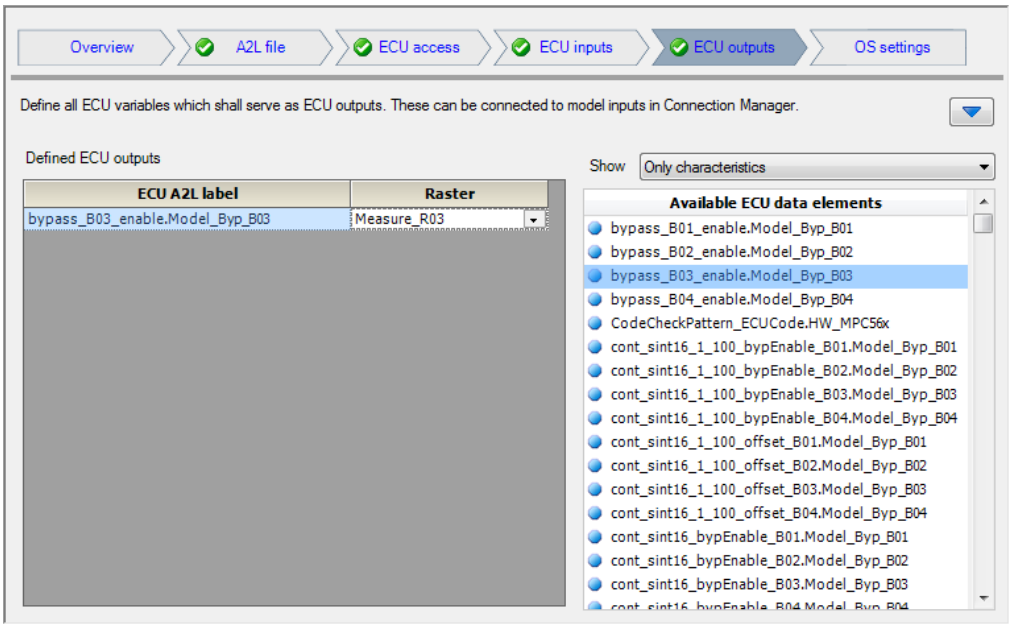


Abb. 3-25 Steuergeräteausgänge zur Verbindung mit Modelleingängen bereitstellen

Datenelement auswählen

- Wählen Sie ein Datenelement in der Liste rechts und ziehen Sie es in Liste links davon.

  Das gewählte Element wird in der Spalte „ECU A2L label“ eingefügt.
- Wählen Sie in der Spalte „Raster“ aus der Liste ein Raster aus.

Die hier gewählten Datenelemente sind nach dem Speichern im Connection Manger als Outputs verfügbar (falls nicht, klicken Sie im Connection Manager **Update Ports**).

3.9.6 OS-Einstellungen

In diesem Schritt können Sie Steuergeräteraster mit Tasks des Real-Time PCs verknüpfen. Dies ermöglicht eine enge Kopplung der Simulation an ein Steuergeräte-Raster.

Die folgende Abbildung zeigt das Register „OS settings“.

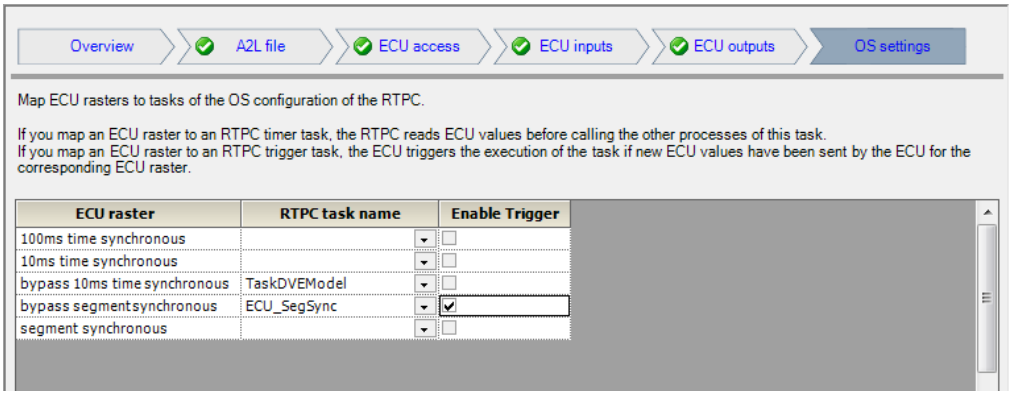
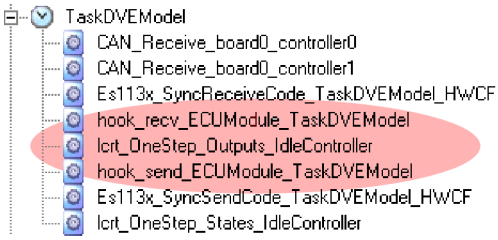


Abb. 3-26 Zuordnung von Tasks zu Steuergeräterastern

Dies ist sowohl für Timer Tasks als auch Trigger Tasks von Bedeutung.

Timer Task

Wird ein Steuergeräteraster auf die Timer Task des Real-Time PCs gemappt, in der das Modell gerechnet wird, dann – wird vor dem Modellberechnungsschritt dt – vom Steuergerät gelesen (hook_recv_ECUModule_[*Taskname*]), dann gerechnet (lcrt_OneStep_Outputs_...) und schließlich zum Steuergerät geschrieben (hook_send_ECUModule_[*Taskname*]).

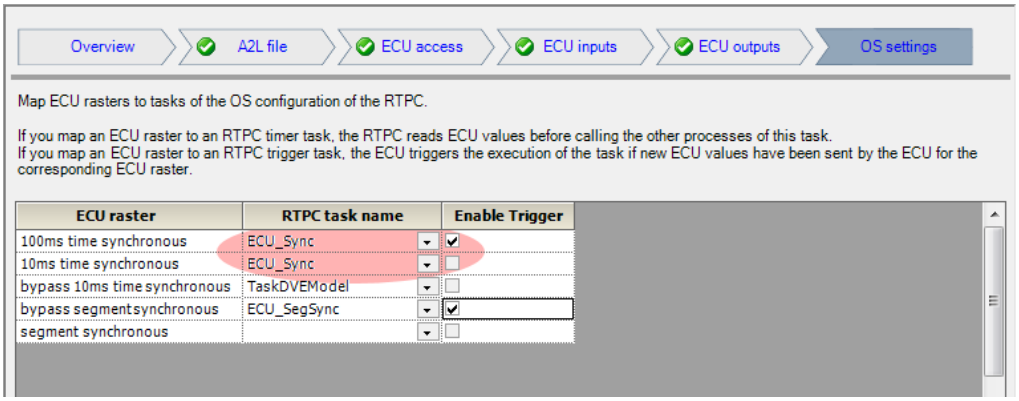


Trigger Task

Ein Mapping eines Steuergeräterasters auf eine Trigger Task ermöglicht eine enge Kopplung: Das Steuergerät kann zum Taktgeber der Simulation gemacht werden – der Real-Time PC startet die Modellberechnung erst, nachdem das Steuergerät Daten gesendet hat.

Wenn das Steuergerät zum Taktgeber werden soll, muss man eine Trigger-Task zuweisen und die Option „Enable Trigger“ aktivieren (Abb. 3-26 auf Seite 216). Ist diese Option nicht gewählt, wird die Real-Time PC-Task nicht getriggert, obwohl das Steuergerät Daten sendet.

Deshalb wird die Option „Enable Trigger“ in den allermeisten Fällen aktiviert sein – sie nicht zu aktivieren, ist dennoch in folgendem Fall sinnvoll: Wenn mehrere Steuergeräte-Raster derselben Trigger-Task zugeordnet wurden, dann sollte die Option „Enable Trigger“ nur für ein Raster aktiviert sein (siehe Abb.).

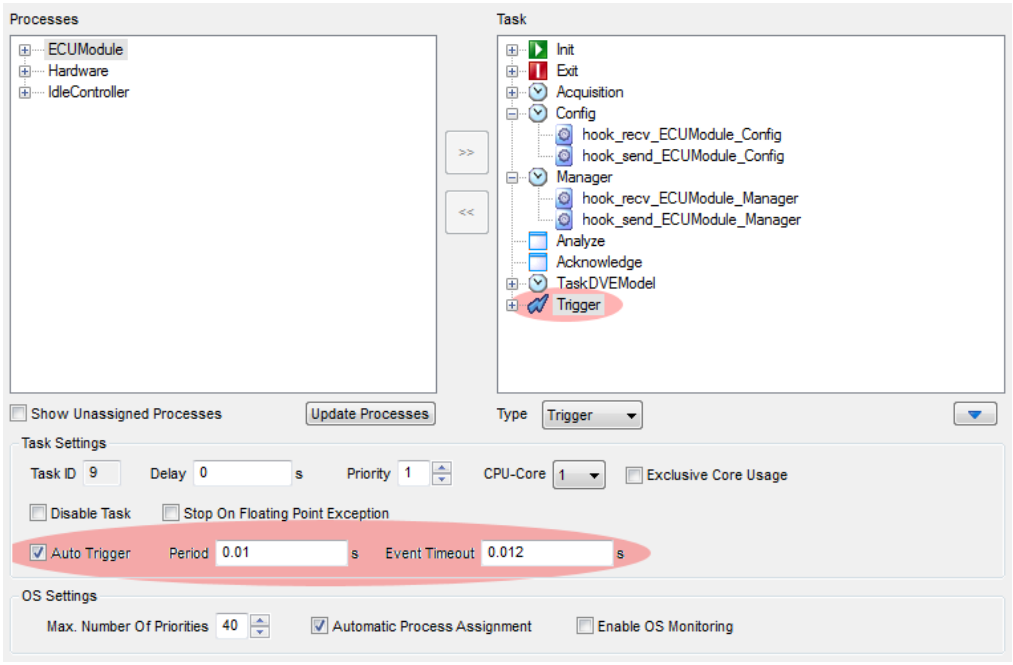


Ansonsten würde das Steuergerät die Trigger-Task (in der Abb. „ECU_Sync“) mehrfach während einer Periode dT triggern, was zu einem schwer nachvollziehbarem Zeitverhalten führt (mehrere, nicht-äquidistante Lese-Zyklen vom Steuergerät).

Die Option „Auto Trigger“

Bleiben Daten vom Steuergerät aus, führt dies zum Stillstand der Simulation. Um dies zu verhindern, gibt es für Tasks vom Typ „Trigger“ die Option „Auto Trigger“.

Wenn Sie im Register „OS Configuration“ die entsprechende Trigger Task wählen, können Sie die Option „Auto Trigger“ unten im Fenster aktivieren.



Beispiel: Wenn nach 12 ms („Event Timeout“) noch keine Daten vom Steuergerät angekommen sind, wird mit einer Schrittweite von 10 ms („Period“) weitergerechnet, und zwar so lange, bis wieder Daten vom Steuergerät kommen.

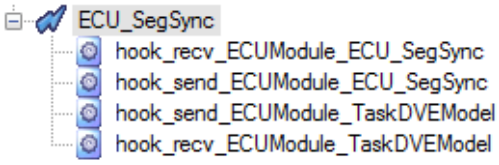
Die Zuordnung einer Real-Time PC-Task in der Spalte „RTPC Task Name“ (siehe Abb. 3-26 auf Seite 216) fügt die Prozesse

„hook_recv_ECUModule_[*Taskname*]“

und

„hook_send_ECUModule_[*Taskname*]“

zu der jeweiligen Task hinzu (Register „OS Configuration“ im Hauptfenster von LABCAR-IP).



Hinweis

Überprüfen Sie am Ende der Konfiguration, ob die Reihenfolge der Prozesse in den für die Steuergerätekommunikation verwendeten Timer/Trigger Tasks korrekt ist.

3.9.7 Verbindungen im Connection Manager erstellen

Wenn Sie die Datenelemente zur Stimulation und Erfassung gewählt haben und Steuergeräteraster auf Tasks gemappt haben, ist die Konfiguration des FiL-Moduls vollständig.

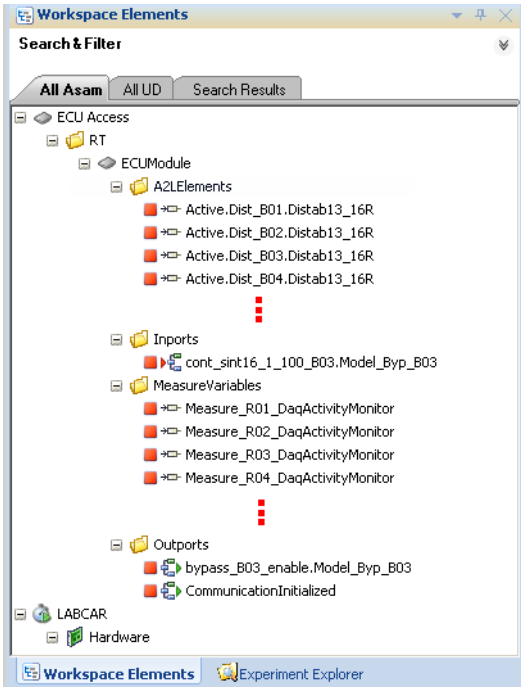
Um mit dem Modul arbeiten zu können, müssen dessen Ein- und Ausgänge anschließend im Connection Manager mit den Ein- und Ausgängen des DVE-Modells verbunden werden.

Der Ausgang „CommunicationInitialized“

Zusätzlich zu den von Ihnen konfigurierten Ausgängen steht im Connection Manager der Ausgang „CommunicationInitialized“ zur Verfügung. Es handelt sich um einen Wert vom Typ „Boolean“, dessen Übergang von 0 → 1 signalisiert, dass das Steuergerät kommunikationsbereit ist.

3.9.8 Arbeiten in ETAS EE

Nach einem erneuten Build des Projektes kann es in der Experimentierumgebung geöffnet werden. Die folgende Abbildung zeigt die Darstellung des FiL-Moduls in den Workspace Elements.



Das Modul besitzt folgende Unterordner

- **A2LElements**

  Dieser Ordner enthält alle A2L-Label (Messgrößen und Parameter) der Steuergerätebeschreibungsdatei.

- **Inports**

  Die bei der Konfiguration definierten Eingänge des Moduls

- **MeasureVariables**

  Je eine Variable pro Steuergeräteraster, die hochgezählt wird, wenn dieses Raster vom Steuergerät getriggert wird.

- **Outports**

  Die bei der Konfiguration definierten Ausgänge des Moduls. Dieser Ordner enthält auch den Ausgang „CommunicationInitialized“ (siehe „Der Ausgang „CommunicationInitialized““ auf Seite 218).

3.9.9 Das Instrument „RT ECU Access“

Messgrößen und Parameter können mit den üblichen Instrumenten angezeigt und geändert werden. Zur Steuerung der FiL-Funktion gibt es ein spezielles Instrument „RT ECU Access“.

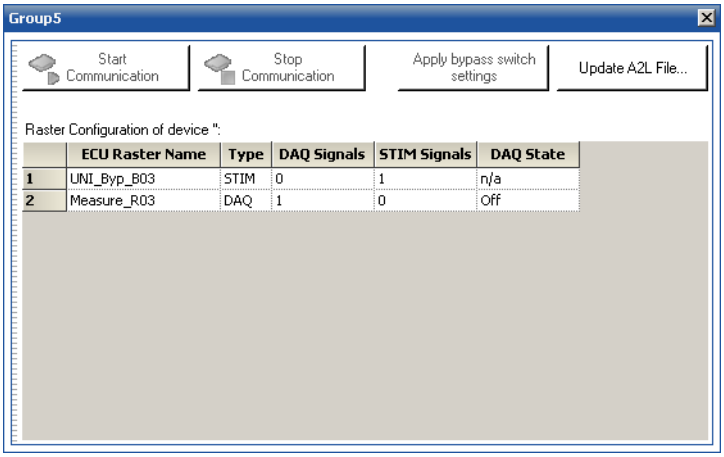


Abb. 3-27 Instrument zur Steuerung der FiL-Funktion

Im oberen Teil des Instrument befinden sich die Schaltflächen zur Steuerung der Kommunikation:

- **Start Communication / Stop Communication**

  Damit können Sie die Kommunikation des FiL-Moduls mit dem Steuergerät starten und stoppen.

- **Apply bypass switch settings**

  Damit können Sie die Default-Werte für die Bypass-Schalter (siehe „Initial Value“ auf Seite 215) setzen lassen.

- **Update A2L File**

  Wenn Sie aufgrund eines neuen Softwarestandes die A2L-Datei austauschen, können Sie mit dieser Funktion eine Aktualisierung durchführen, ohne die Konfiguration in LABCAR-IP zu öffnen.

Hinweis

Die Aktualisierung funktioniert nur dann, wenn die neue Datei alle Raster und Steuergerätelabels des Moduls enthält!

Im unteren Teil des Instruments wird die Konfiguration der Steuergeräteraster dargestellt. Die Bedeutung der einzelnen Spalten ist wie folgt:

- **ECU Raster name**

  Der Name des Rasters

- **Type**

  Typ: „DAQ“ oder „STIM“

- **DAQ Signals / STIM Signals**

  Zahl der in der jeweiligen Task gemessenen/stimulierten Signale

- **DAQ State**

  Zeigt an, ob für das DAQ-Raster gerade Werte vom ECU geliefert werden. Mögliche Werte sind „n/a“, „Off“ und„Running“.

3.10 Real-Time Plugins

In diesem Kapitel finden Sie Informationen zur Erstellung von Real-Time Plugins, deren Integration in ein LABCAR-OPERATOR-Projekt und das Arbeiten mit diesen in der Experimentierumgebung ETAS EE.

Real-Time Plugins

Ein Real-Time Plugin ist eine dynamisch nachladbare Programmbibliothek für den Real-Time PC, dessen Funktionen zur Laufzeit in das Echtzeitbetriebssystem eingebunden und in diesem ausgeführt werden können.

Mit dem RT-Plugin Builder wird ein Werkzeug zur Erstellung eines Real-Time Plugin Projektes und zur Verwaltung des zugehörigen Quellcodes zur Verfügung gestellt. Der RT-Plugin Builder steuert auch das Kompilieren und Linken des Plugins auf dem Real-Time PC und erzeugt eine Paketdatei für das Real-Time Plugin.

ETAS RTPC

Das Kompilieren und Linken des Plugin-Quellcodes zu einer dynamisch nachladbaren Programmbibliothek (Shared Object Library) erfolgt auf dem Real-Time PC. Dieser erstellt die notwendige Funktionalität und Infrastruktur bereit, die für Real-Time Plugins benötigt wird.

Hierzu gehört insbesondere

- das Laden und Entladen von Real-Time Plugins
- das Erzeugen von Hook-Prozessen
- das Binden von Funktionen (des Plugins) an Hook-Prozesse
- das Auflösen von Datenelementbezeichnern in Speicheradressen
- das Aufrufen der Hook-Prozesse im Echtzeitkontext

Konfiguration des Betriebssystems

Das Einfügen von Hooks in Tasks des Echtzeitbetriebssystems erfolgt in LABCAR-IP im Register „OS Configuration“.

Experimentierumgebung (ETAS EE)

Real-Time Plugin Paketdateien können in der Experimentierumgebung hinzugefügt, aktiviert, deaktiviert und wieder entfernt werden.

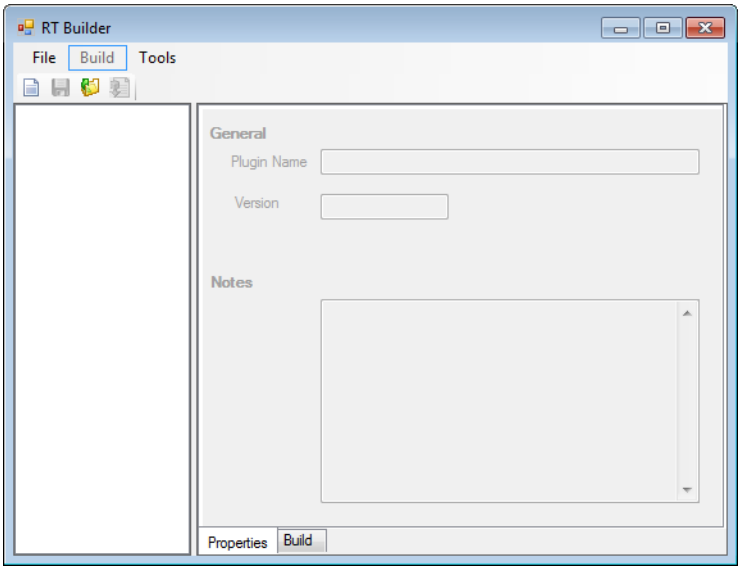
3.10.1 Der RT-Plugin Builder

Der RT-Plugin Builder ist eine Anwendung zum Erstellen von Real-Time Plugins. Der RT-Plugin Builder erzeugt eine Verzeichnisstruktur und Dateien mit Funktionsrümpfen für die erforderlichen Callback-Funktionen des Plugins.

Real-Time Plugin Projekte erstellen

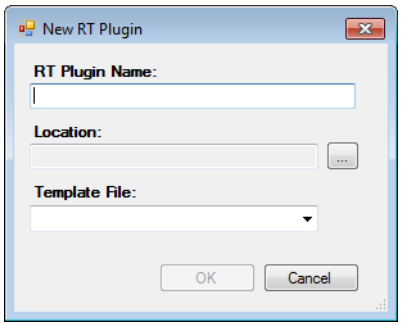
- Wählen Sie im Startmenü **Programme** → **ETAS** → **LABCAR-OPERATOR X.Y** → **RT-Plugin Builder**.

  Der RT-Plugin Builder wird geöffnet.



- Wählen Sie **File** → **New**.

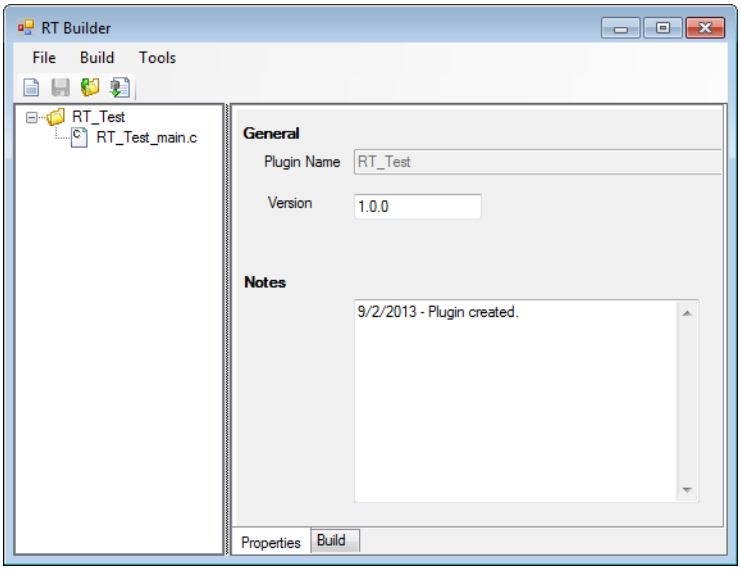
  Das Fenster „New RT Plugin“ wird geöffnet.



- Geben Sie den Namen des RT-Plugins und das Verzeichnis an, in dem die Dateien des Projekts gespeichert werden sollen.
- Wählen Sie eine Dateivorlage (Template File).

- Klicken Sie **OK**.

  Das Projekt wird erstellt.



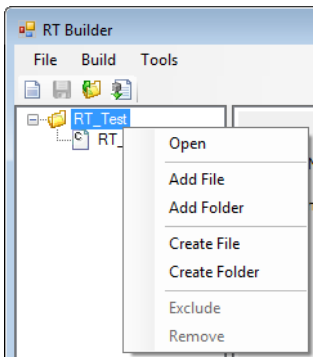
- Verändern Sie ggf. die Versionsnummer („Version“).
- Speichern Sie das Projekt mit **File** → **Save**.

Real-Time Plugin Quellcode verwalten

Alle Dateien, die zu einem Plugin gehören, werden im RT-Plugin Builder in Form einer logischen Baumstruktur verwaltet.

- Rechtklicken Sie auf dem Projektordner.

  Das Kontextmenü wird geöffnet.



- Mit **Create...** können Sie neue Dateien oder Ordner erstellen.
- Mit **Add...** können Sie (im Projektverzeichnis) vorhandene Dateien oder Ordner hinzufügen.

- Um eine Datei oder einen Unterordner aus dem Projekt zu entfernen, wählen Sie **Exclude**.

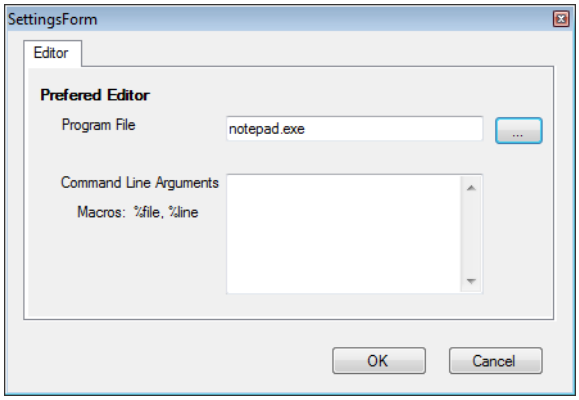
  Die Dateien/Ordner im Projektordner bleiben erhalten und können mit **Add File** / **Add Folder** wieder hinzugefügt werden.
- Um eine Datei oder einen Unterordner zu löschen, wählen Sie **Remove**.

  Die Dateien/Ordner werden physikalisch gelöscht.

Dateien editieren

- Um eine Datei im Baum zu editieren, doppelklicken Sie diese.

  Die Datei wird im voreingestellten Editor geöffnet.
- Wählen Sie **Tools** → **Settings**, um einen anderen Editor zu definieren.



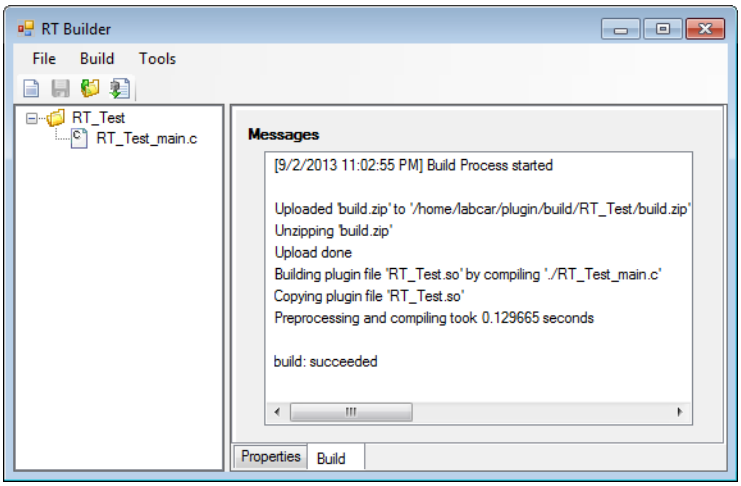
- Wenn Sie beim Aufruf der Editors Argumente übergeben wollen, können Sie diese im Feld „Command Line Arguments“ spezifizieren.

Kompilieren und Linken eines Real-Time Plugins

- Wechseln Sie zum Register „Build“.

- Wählen Sie **Build** → **Build Plugin**.

  Die Dateien werden zum Real-Time PC übertragen und dort verarbeitet.



Die Ergebnisse des Build-Prozesses werden unter „Messages“ angezeigt.

Real-Time Plugin Packages

Beim Build-Vorgang wird aus den Dateien mit dem Quellcode eine Shared-Object-Library erzeugt. Der RT-Plugin Builder packt diese Datei zusammen mit anderen Dateien in eine Archivdatei, die im Folgenden als Real-Time Plugin Package bezeichnet wird.

Ein Real-Time Plugin Package enthält:

- Eine Metadatendatei (xml) mit Informationen zu Tools, Versionen etc.
- Die Shared-Object-Library des Plugins

3.10.2 Beispiel

Das folgende Beispiel zeigt die Implementierung eines einfachen Counters, der mittels der Befehle „enable“, „disable“ und „reset“ gesteuert werden kann.

Das Plugin interagiert mit einem C-Code-Modul des LABCAR-OPERATOR-Projektes, indem es den einen Skalierungsfaktor vom Modul liest und den (skalierten) Zählerwert in einen Modulparameter „In“ schreibt. Das C-Code-Modul gibt „In“ über den Outport „Out“ aus.

```
#include <stdio.h>
#include <math.h>
#include "rtos_hook.h"
#include "rtos_rtplugin.h"

// Label definition
//    raw label
#typedlabel double cmodIn CModule/CalibrationVariables/In
```

```
//    mapped (user defined label)
#typedlabel 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)
{
}
```

Erläuterungen zu diesem Beispiel finden Sie in den folgenden Abschnitten.

3.10.3 Definition von Hooks in der OS Configuration von LABCAR-IP

Ein Hook fungiert als Platzhalter für einen oder mehrere Prozesse in der OS Configuration eines LABCAR-Projektes. Ein Hook kann wie ein Prozess an einer beliebige Stelle in einer Task eingefügt, verschoben und auch gelöscht werden. Im Gegensatz zu Prozessen kann der Name eines Hooks vom Benutzer frei vorgegeben (und auch umbenannt) werden.

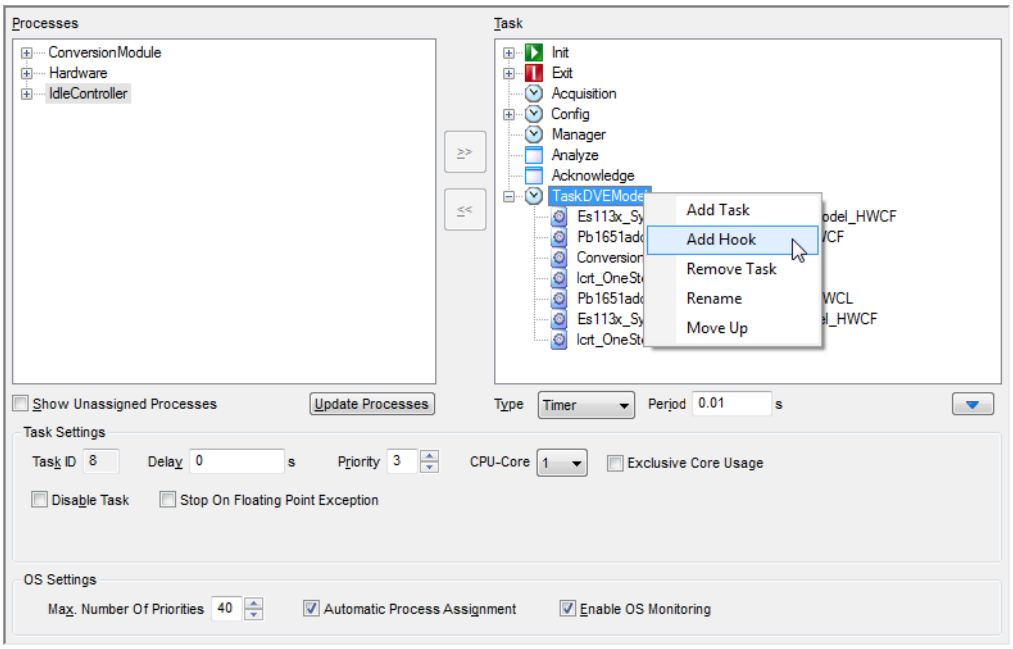


Abb. 3-28 Hinzufügen eines Hooks in der OS Configuration

Ein Hook darf auch mehrfach verwendet werden, d.h. das Einfügen von Hooks mit gleichen Namen an unterschiedlichen Positionen innerhalb einer oder mehrerer Tasks ist zulässig.

3.10.4 Anhängen von Plugin-Prozessen an Hooks

Das Anhängen von Prozessen an einen Hook erfolgt durch das Real-Time Plugin selbst. Beim Laden eines Real-Time Plugins wird die Methode

```
on_load
```

des Plugins aufgerufen.

Innerhalb dieser können dann mittels der Funktion

```
rtos_hook_attach
```

Prozesse an bestimmte Hooks gebunden werden.

Ist ein erforderlicher Hook nicht vorhanden, so hängt es von der Implementierung der `on_load` Methode ab, ob das Laden des Real-Time Plugins abgebrochen werden muss oder fortgesetzt werden kann.

Es ist möglich, mehrere Prozesse an einen Hook zu binden. Neue Prozesse werden grundsätzlich immer am Ende der Prozessliste eines Hooks angehängt. Das Einfügen eines Prozesses an einer beliebigen Position der Hook-Prozessliste ist nicht möglich.

Beim Entladen eines Real-Time Plugins sind alle angehängten Prozesse in der Methode

`on_unload`

mittels der Funktion

`rtos_hook_detach`

aus den Prozesslisten der Hooks zu entfernen.

3.10.5 Label und Mapping

Jedes Datenelement in einem LABCAR-OPERATOR-Projekt besitzt ein Label, mit dessen Hilfe eine eindeutige Adressierung möglich ist.

Adressierung von Datenelementen

Die Adressierung von Datenelementen eines LABCAR-OPERATOR-Projektes wird bei Real-Time Plugins durch eine Compiler-Direktive realisiert.

Die Syntax der Direktive lautet:

```
#typedlabel  <Typ> <Variablenbezeichner> <Label des Datenelements>
```

Der Zugriff auf das Datenelement erfolgt im Quellcode des Plugins über den angegebenen Variablenbezeichner. Die Auflösung eines Labels in die Speicheradresse des Datenelements erfolgt zur Laufzeit beim Laden des Plugins – vor Aufruf der Plugin-Funktion `on_load`.

Ist das Label innerhalb des LABCAR-OPERATOR-Projektes unbekannt oder entspricht der Datentyp des Datenelements nicht dem erwarteten Typ, so wird das Real-Time Plugin nicht geladen und entsprechende Fehlereinträge in der Logdatei von ETAS RTPC erzeugt.

Mapping von Labeln

Benutzerdefinierte Label für Datenelemente werden bei LABCAR-OPERATOR mit Hilfe von Mapping-Dateien und SuT-Mapping-Dateien realisiert. Die Verwendung von benutzerdefinierten Label im Rahmen von Real-Time Plugins ermöglicht das Erstellen von projektunabhängigen Plugins.

Die Auflösung eines benutzerdefinierten Labels in die Speicheradresse des Datenelements erfolgt zur Laufzeit beim Laden des Plugins – vor Aufruf der Plugin-Funktion `on_load`.

Ist das Label innerhalb des LABCAR-OPERATOR-Projektes unbekannt oder kann aufgrund eines fehlendes Mappings nicht aufgelöst werden, oder entspricht der Datentyp des Datenelements nicht dem erwarteten Typ, so wird das Real-Time Plugin nicht geladen und entsprechende Fehlereinträge in der Logdatei von ETAS RTPC erzeugt.

Hintergrunddienste

Funktionen, die nicht im Echtzeitkontext ausgeführt werden sollen (z.B. Zugriffe auf Dateien), können über die Funktion `rtos_hook_attach` der Background-Task des Betriebssystems zugeordnet werden – der zugehörige Hook-Bezeichner lautet `rtos_hook_background`.

3.10.6 Lebenszyklus eines Real-Time Plugins

Das Aktivitätsdiagramm in Abb. 3-29 zeigt die Abfolge der einzelnen Aktivitäten beim Laden bzw. Entladen des Real-Time Plugins.

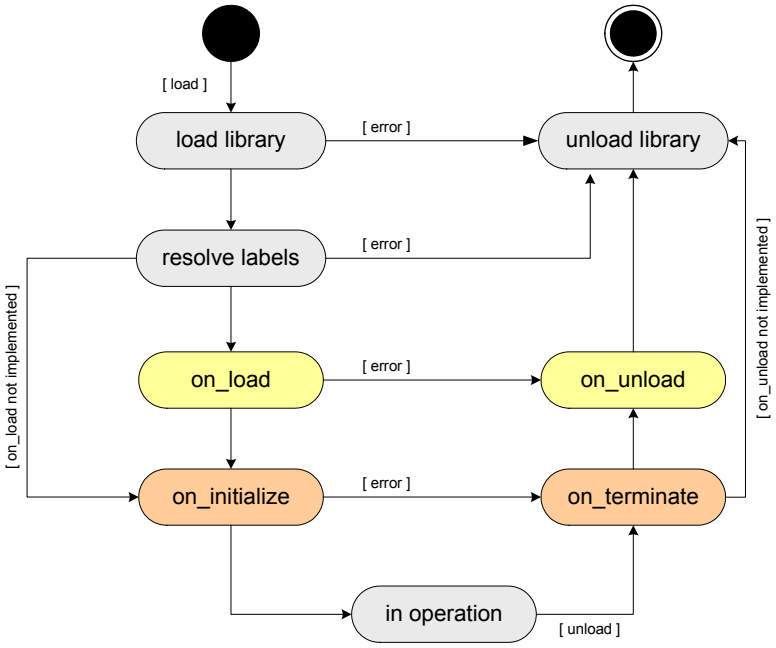


Abb. 3-29 Aktivitätsdiagramm

- **load library**

  Laden der Shared Object Library und Auflösen der Funktionszeiger für die Callback-Funktionen des Plugins

  – `on_load` (optional)
  – `on_initialize` (erforderlich)
  – `on_terminate` (erforderlich)
  – `on_unload` (optional).

  Fehlerfall: Die Funktionszeiger für erforderliche Callback-Funktionen können nicht aufgelöst werden.

- **resolve labels**

  Auflösen der Label für Datenelemente in Speicheradressen.

  Fehlerfall: Label eines Datenelements kann nicht aufgelöst werden oder Typ des Datenelements ist inkompatibel.

- **on_load**

  Ausführen der Callback-Funktion `on_load`, sofern implementiert. Diese Funktion dient zur Behandlung von Abhängigkeiten und zum Reservieren von Systemressourcen.

  Fehlerfall: Rückgabewert der Callback-Funktion ist < 0.

- **on_initialize**

  Ausführen der Callback-Funktion `on_ initialize`. In dieser Funktion werden typischerweise Echtzeitfunktionen bestimmten Hooks zugewiesen, Background-Funktionen und Command-Handler registriert.

  Fehlerfall: Rückgabewert der Callback-Funktion ist < 0.

- **in operation**

  Ausführung der registrierten Echtzeit- und Nichtechtzeitfunktionen durch das Betriebssystem des Real-Time PC.

- **on_terminate**

  Ausführen der Callback-Funktion `on_ terminate`. In dieser Funktion werden die zuvor in `on_initialize` registrierten Funktionen wieder abgemeldet.

- **on_unload**

  Ausführen der Callback-Funktion `on_unload`, sofern implementiert. Systemressourcen, die zuvor in `on_load` reserviert wurden, sind in dieser Funktion wieder freizugeben.

- **unload library**

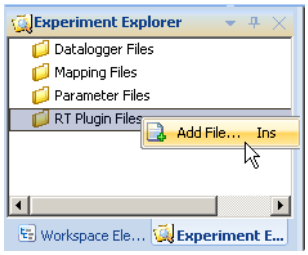
  Die Shared Object Library des Real-Time Plugins wird freigegeben und entladen.

3.10.7 Arbeiten mit Real-Time Plugins in der Experimentierumgebung

Real-Time Plugins werden in der Experimentierumgebung genauso verwaltet wie Parameter- oder Mapping-Dateien, d.h. Hinzufügen, Entfernen, Aktivieren und Deaktivieren erfolgt im Experiment Explorer von ETAS EE.

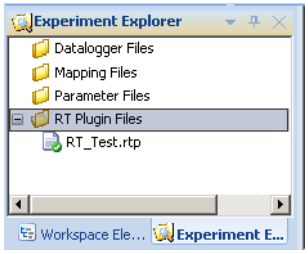
RT-Plugins zum Experiment hinzufügen

- Rechtsklicken Sie im Experiment Explorer auf den Ordner „RT Plugin Files“.
- Wählen Sie **Add File**.

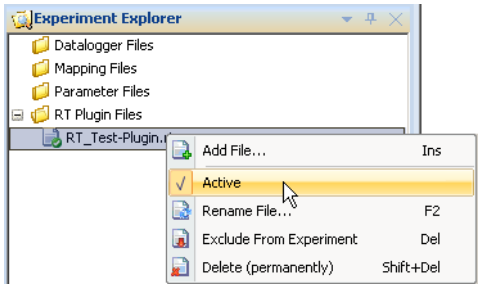


- Wählen Sie im Dateiauswahlfenster die Datei des Plugins (*.rtp).

  Das Real-Time Plugin wird zum Experiment hinzugefügt.



- Falls das Plugin nicht aktiv ist, rechtsklicken Sie die Datei und wählen Sie **Active**.



- Um eine Datei aus dem Ordner „RT Plugin Files“ zu entfernen, wählen Sie **Exclude From Experiment**.
- Um eine Datei zu löschen, wählen Sie **Delete (permanentely)**.

Experiment mit Real-Time Plugin ausführen

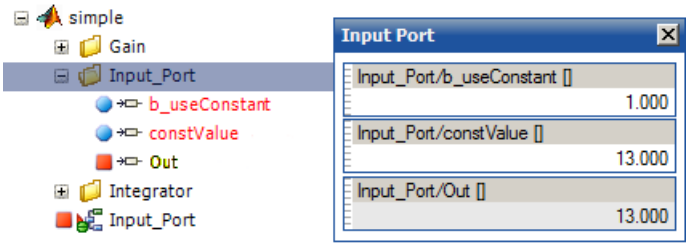
- Wählen Sie **Experiment → Download → LABCAR**, um das Experiment zum Simulationstarget herunterzuladen.

  Der Zustand des Plugins wechselt zu „Initialized“.
- Wählen Sie **Experiment → Start Simulation → LABCAR**, um die Simulation zu starten.

  Das Experiment wird ausgeführt.
- Wechseln Sie in das Register „RT Plugins“.

Hinweis bei Verwendung von LABCAR Ports in Simulink-Modellen

LABCAR Input Ports von Simulink-Modellen sollten von RT-Plugins nicht direkt beschrieben werden. Setzen Sie statt dessen den Parameter „B_useConstant“ auf „1.0“ und beschreiben Sie den Parameter „constValue“ aus dem RT-Plugin.



Dadurch werden der Messwert „Out“ („simple/Input_Port/Out“) und alle nachfolgenden Simulink Blöcke zuverlässig beeinflusst, falls der zugehörige Hook entsprechend in der OS Configuration eingehängt wurde.

LABCAR Output Ports sollten ebenfalls nicht von RT-Plugins beschrieben werden. Das Beschreiben von Eingängen von C-Code-Modulen ist dagegen möglich.

3.10.8 Das Register „RT-Plugins“

Die folgende Abbildung zeigt das Register „RT Plugins“.

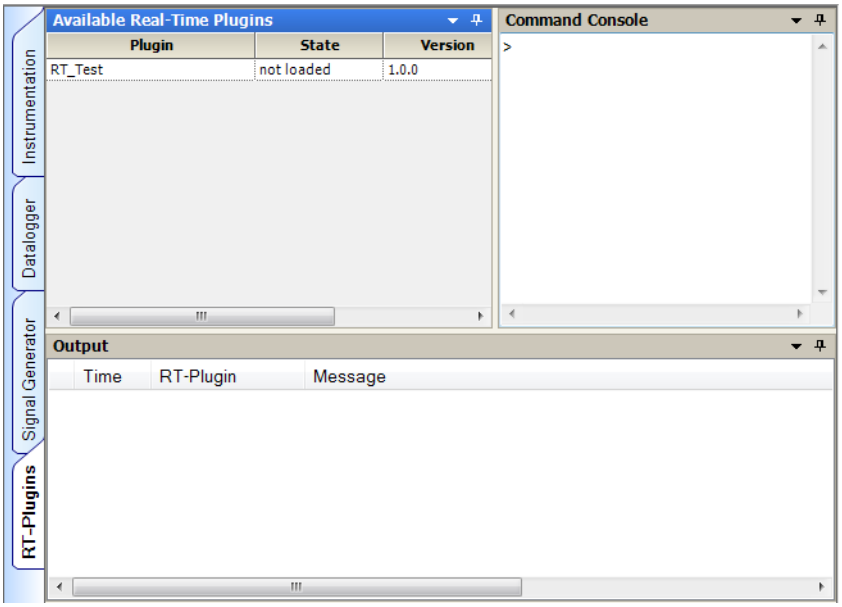


Abb. 3-30 Das Register „RT Plugins“ in ETAS EE

Der Bereich besteht aus drei Teilen:

- Available Real-Time Plugins
- Output (siehe „Output“ auf Seite 235)
- Command Console (siehe „Command Console“ auf Seite 235)

Available Real-Time Plugins

Hier sind alle Real-Time Plugins des Experiments aufgelistet.

- **Plugin**

  Der Name des Plugins

- **State**

  Die Spalte „State“ zeigt den aktuellen Zustand der einzelnen Plugins an. Die Abfrage erfolgt zyklisch im Raster von etwa 500 ms.

  Die Zustände „loaded“ und „not loaded“ sind vordefiniert und zeigen den Ladezustand eines Plugins an. Die Definition benutzerdefinierter Zustände ist möglich und erfolgt im Quellcode des Plugins.

- **Version**

  Die Version des Plugins (wird bei Erstellung im RT-Plugin Builder vergeben)

Output

Im Register „Output“ (in der untere Hälfte) werden die Log-Messages angezeigt, die von den Real-Time Plugins generiert werden.

Command Console

Das Befehlsinterface bietet die Möglichkeit, für jedes Real-Time Plugin ein eigenes Befehlsprotokoll (basierend auf ASCII-Zeichenketten) zu implementieren. Dazu muss das Plugin einen Protokoll-Handler bereitstellen, welche die Interpretation der übergebenen Strings durchführt.

Dieser Handler muss beim Laden des Real-Time Plugins mittels

```
rtos_plugin_command
```

registriert werden.

Folgende Befehle stehen zur Verfügung:

at oder @

Delegiert den nachfolgend angegebene Befehl an das benannte RT-Plugin
Beispiel: `at ExamplePlugin disable`

clear

Löscht die Befehlskonsole

info

Listet alle (auf dem Target) geladenen Plugins auf

help

Hilfestellung

load

Lädt das genannte Plugin (→ `on_load`)
Beispiel: `load ExamplePlugin`

unload

Entlädt das genannte Plugin (→ `on_unload`)
Beispiel: `unload ExamplePlugin`

init

initialisiert ein Plugin (→ `on_initialize`)
Beispiel: `init ExamplePlugin`

terminate

terminiert ein Plugin (→ `on_terminate`)
Beispiel: `terminate ExamplePlugin`

<Tab>

Autovervollständigung (case-sensitive)

<Pfeil hoch>

Befehlshistorie

<Pfeil runter>

Befehlshistorie

3.11 Module zur Signalkonvertierung

Mit LABCAR-OPERATOR V5.4.1 wird ein Standard-Signalkonvertierungsmodul mitgeliefert, mit dem die generische Open-Loop-Konfiguration realisiert werden kann.

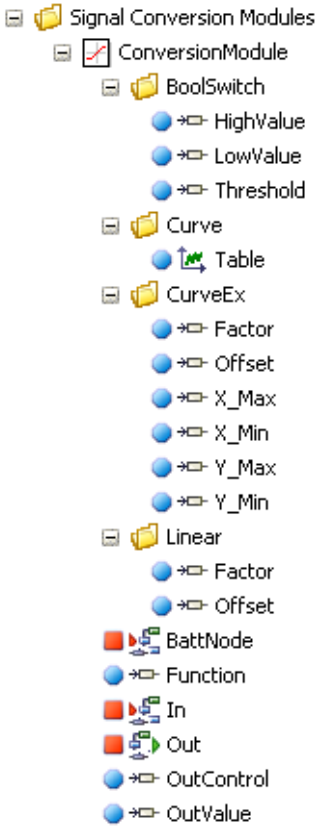
Dieses Modul kann im Connection Manager vor den Eingängen anderer Module eingefügt werden und ermöglicht insbesondere (zwischen I/O-Hardware und DVE-Modell) eine Sensor/Aktor-Modellierung.

3.11.1 Module einfügen

Wie ein solches Modul zur Signalkonvertierung eingefügt wird, ist in Abschnitt „Signalkonvertierung“ auf Seite 253 beschrieben.

3.11.2 Parameter

Wurde ein Modul zur Signalkonvertierung eingefügt, stehen sämtliche Ein- und Ausgänge und Parameter auch in ETAS EE im Fenster „Workspace Elements“ zur Verfügung.

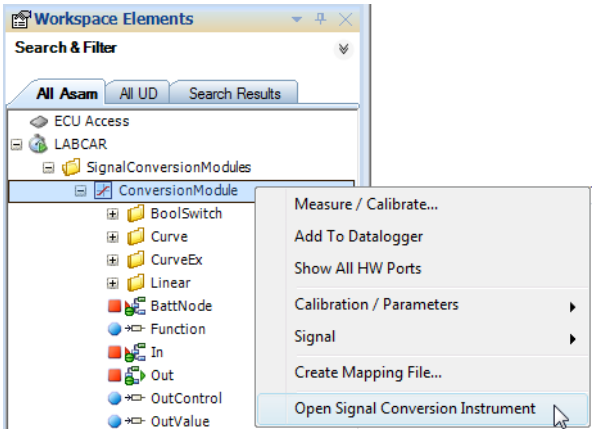


Neben der Parametrierung über Parameterdateien stellt die Instrumentierung von ETAS EE auch ein GUI zur Verfügung, über das die Parameter des Moduls verstellt werden können.

Die Bedeutung der Parameter (und damit die Funktionsweise der Module) wird im Folgenden anhand des GUIs beschrieben.

3.11.3 Das GUI zur Steuerung der Signalkonvertierung

Zum Erstellen eines solchen Bedienfensters rechtsklicken Sie das jeweilige Modul in der Baumansicht des Fensters „Workspace Elements“ und wählen Sie **Open Signal Conversion Instrument**.



Im GUI, das im aktiven Layer (im Register „Instrumentation“) erstellt wird, sind unter „Type“ folgende Konvertierungstypen wählbar:

- Boolean
- Curve
- CurveEx
- Identity
- Linear

Steuerung des Ausgangssignals

Gemeinsam ist allen Typen, dass sich der Ausgang steuern lässt – hierzu dienen die Optionen im Feld „Out Control“.

Folgende Einstellungen sind möglich:

- On

  Der Ausgang wird immer berechnet.
- Off

  Der Ausgang wird nicht berechnet - der Ausgang wird auf den im Feld eingegebenen Wert gesetzt.

- Node

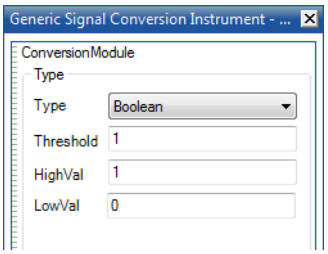
  Die Berechnung des Ausgangs ist abhängig vom Zustand eines Batterieknotens (Eingang „BattNode“ in den Workspace Elements und im Connection Manager). Wenn der gewählte Batterieknoten nicht geschaltet ist, wird die Kennlinie nicht berechnet.

Hinweis

Wenn Sie Zweifel an den Werten irgendeines Signals haben, prüfen Sie immer, ob nicht die Berechnung des Signals vom Status eines Batterieknotens abhängt oder ganz abgeschaltet wurde.

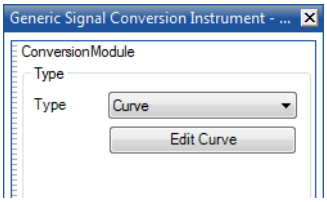
Typ „Boolean“

Hier kann ein Schwellwert „Threshold“ für das Eingangssignal eingegeben werden, unterhalb dessen der Ausgangswert den Wert „LowVal“ und oberhalb dessen er den Wert „HighVal“ annimmt.

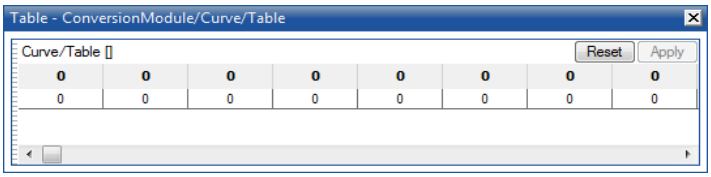


Typ „Curve“

Hier kann über die Schaltfläche **Edit Curve** eine Tabelle für eine Kennlinie bearbeitet werden.

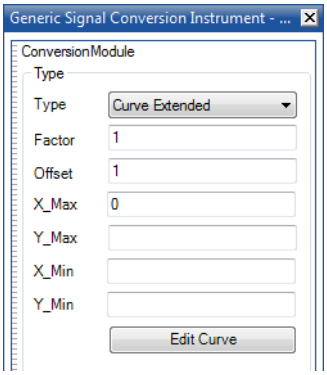


Dabei wird ein GUI geöffnet, in dem die Wertepaare für den Parameter „ConversionModule/Curve/Table“ editiert werden können.



Typ „Curve Extended“

Die erweiterte Form des Typs „Curve“, die eine weitere Veränderung der Kennlinie zulässt.



Hier kann das Signal in mehreren Stufen verändert werden:

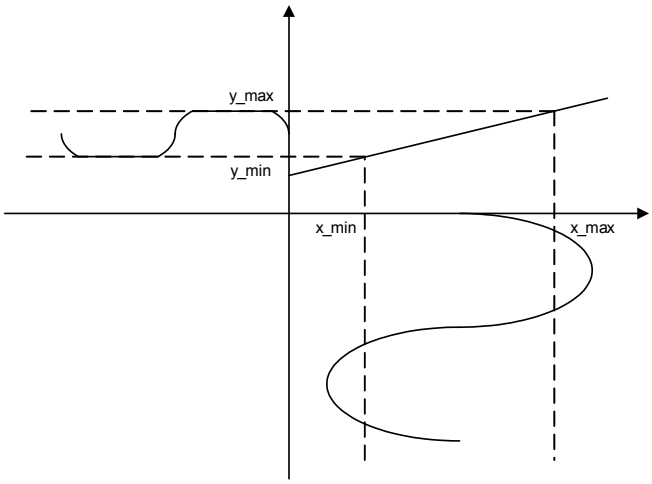
1. Abbildung über eine lineare Funktion v = a*u + b, die eine Verstärkung oder Abschwächung a („Factor“) und eine Verschiebung b („Offset“) des Signals bewirkt. Voreinstellung: Offset = 0 und Factor = 1, also keine Änderung.

Hinweis

*Diese Abbildung wirkt auf die **Eingangsgrößen** der im nächsten Schritt in Form einer Tabelle definierten Kennlinie!*

2. Abbildung durch eine mit **Edit Curve** in Tabellenform definierte Kennline (siehe „Typ „Curve““ auf Seite 239).
3. Eine Signalbegrenzung (spezifiziert über „y_min“ und „y_max“) zuzüglich einer linearen Abbildung (durch Angabe entsprechender Werte „x_min“, „x_max“, „y_min“ und „y_max“).

   Die folgende Abbildung zeigt die Wirkungsweise der Signalbegrenzungsfunktion, die durch die Werte x_min, x_max, y_min und y_max definiert ist.



Das Eingangssignal wird durch x_min und x_max beschränkt - die Angabe von x_min, x_max, y_min und y_max legt eine Geradengleichung fest, die das Signal mit einer Verstärkung und mit einem Offset versieht.

Wenn keine Beschränkung des Signals gewünscht wird, wählen Sie die Minimal- bzw. Maximalwerte entsprechend klein bzw. groß .

Wenn keine weitere Verstärkung und kein Offset gewünscht wird, setzen Sie einfach y_min = x_min und y_max = x_max.

Ansonsten besteht zwischen Verstärkung (Steigung a der Geradengleichung (1)), Offset (Parameter b der Geradengleichung (1)) und den Werten x_max, x_min, y_max und y_min folgender Zusammenhang:

$y = a*x + b \quad (1)$

$y = ((y\_max - y\_min)/(x\_max - x\_min))*x - ((y\_max - y\_min)/(x\_max - x\_min))*x1 + y1$

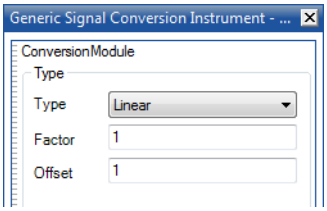
wobei für x1 und y1 eines der beiden Wertepaare (x_max,y_max) oder (x_min,y_min) einzusetzen ist.

Typ „Identity“

Damit wird das Signal 1:1 unverändert weitergegeben.

Typ „Linear“

Damit wird der Eingangswert mittels einer Geradengleichung (ausgedrückt durch „Factor“ und „Offset“) auf den Ausgangswert abgebildet.



3.12 Hardware konfigurieren mit dem RTIO-Editor

Der grundsätzliche Umgang mit dem RTIO-Editor und die Konfiguration der eingesetzten Hardware ist im LABCAR-RTC V5.4.1 - Benutzerhandbuch ausführlich beschrieben.

Ein Beispiel zur Einbindung einer Hardwarekarte finden Sie im Tutorial des Handbuches „LABCAR-OPERATOR V5.4.1 - Schnelleinstieg“.

3.13 Der Connection Manager

Der Connection Manager ermöglicht den Closed-Loop-Betrieb, indem er die Ein- und Ausgänge der vorhandenen Module entsprechend verbindet.

Seine Eigenschaften sind:

- Darstellung aller Ein- und Ausgänge, die verbunden werden können
- Verbinden einzelner Ein- und Ausgänge und Unterbrechen dieser
- Erstellen virtueller Verbindungen zur Signalverfolgung
- Hervorheben bereits existierender Verbindungen
- Möglichkeit zum Export/Import der Verbindungsliste in/aus andere(n) Projekte(n)
- Sortier- und Darstellungsoptionen für Verbindungsliste
- Filter zur Auswahl spezieller Verbindungen

In den folgenden Abschnitten wird beschrieben, wie Sie Modelle mit Ein- und Ausgängen versehen, Verbindungen erstellen und Module zur Signalkonvertierung einfügen.

3.13.1 Reale und virtuelle Verbindungen

Um Signale im Connection Mananger darstellen zu können, müssen diese bekannt sein. Bei den mit LABCAR-IP integrierten Modulen ist dies selbstverständlich, da die Ein- und Ausgänge eines Moduls Teil der Modulbeschreibung sind. Dies betrifft die bereits beschriebenen Module wie Simulink-Modelle, CAN- und FlexRay-Module, vom Anwender bereitgestellte C-Code-Module, Module zum Eingriff in den Signalpfad (Signalkonvertierung und Öffnen der Closed-Loop) und die Hardwarekonfiguration in LABCAR-RTC.

Deren Signale stehen im Connection Manager zur Verfügung und nach der Codegenerierung auch im Experiment in ETAS EE. Die folgende Abbildung zeigt schematisch ein HiL-Experiment mit den typischen Komponenten (Hardware, Signalkonvertierung) und einem Modell mit einer Aktor- und einer Sensorsimulation.

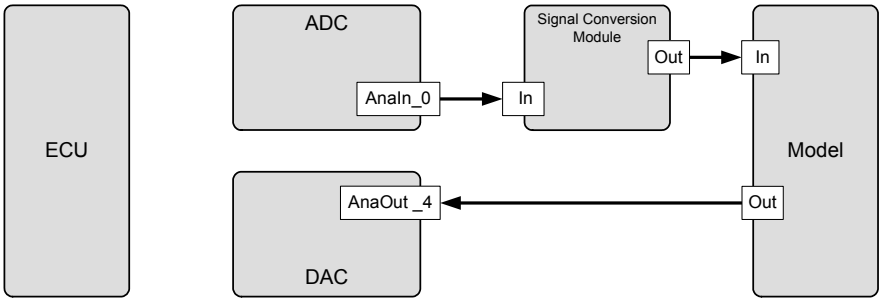


Abb. 3-31 Die Signale des Experiments

Das Aktorsignal vom Steuergerät erscheint hier erstmals als Ausgangssignal eines ADC-Moduls, und wird danach über ein Modul zur Signalkonvertierung an einen Modelleingang geleitet. Beim Sensor beginnt der bekannte Signalpfad an einem Modellausgang und endet am Eingang eines DAC-Moduls.

Bis dato unbekannt sind:

- die Verbindungen zwischen dem Steuergerät und der RTIO-Hardware und
- die Verbindungen innerhalb der Module

Damit fehlen aber wichtige Informationen über den vollständigen Signalpfad bei einem Closed-Loop-Experiment, dessen Verfolgbarkeit eine Anforderung der Experimentierumgebung ist.

Der erste Punkt wird adressiert durch das Bereitstellen von Listen über ECU-Pins und HW-Pins, der zweite durch sogenannte virtuelle Verbindungen.

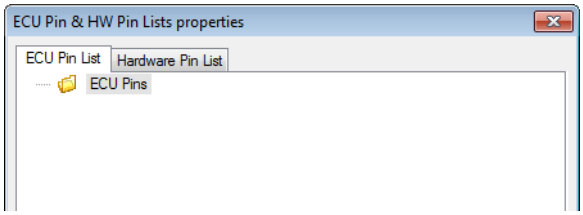
ECU- und HW-Pins

Sie können die ECU Pin List oder die Hardware Pin List erstellen und bearbeiten, wie dies im Folgenden beschrieben wird. Mit der Scripting API von LABCAR-IP lässt sich dies jedoch auch bequem automatisieren. Die Dokumentation dieser API finden Sie in auf der Hilfeseite (**? → Help**).

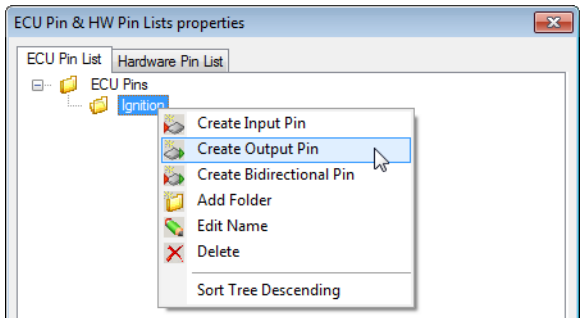
Liste der ECU-Pins erstellen

- Wählen Sie im Hauptmenu von LABCAR-IP **Project → ECU Pin List**.

  Das Fenster „ECU Pin & HW Pin Lists properties“ wird geöffnet.



- Wählen Sie das Register „ECU Pin List“.
- Rechtsklicken Sie auf den Root-Eintrag „ECU Pins“.
- Wählen Sie **Add Folder**.
- Eine neuer Ordner wird erstellt, dessen Namen Sie hier ändern können.
- Rechtsklicken Sie diesen Ordner und wählen Sie z.B. **Create Output Pin**.

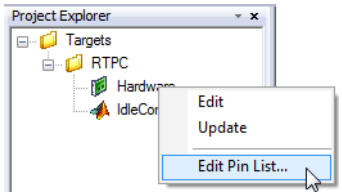


Liste von HW-Pins erstellen

- Wählen Sie **Project** → **ECU Pin List** und anschliessend das Register „Hardware Pin List“.

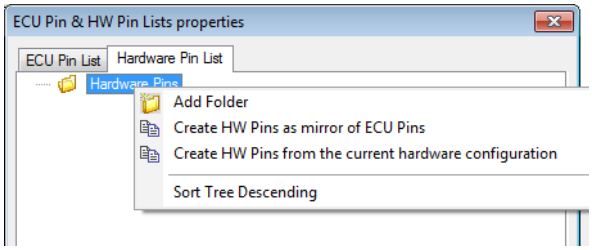
oder

- Wählen Sie im Project Explorer das Hardwaremodul.
- Rechtsklicken Sie und wählen Sie **Edit Pin List**.



Das Fenster „ECU Pin & HW Pin Lists properties“ wird geöffnet.

- Wählen Sie das Register „Hardware Pin List“.
- Im Kontextmenü des Rooteintrages „Hardware“ haben Sie nun folgende Möglichkeiten.



  - Einen Ordner zu erstellen, in dem Sie dann alle Möglichkeiten haben, wie dies bereits bei der ECU Pin List beschrieben wurde.
  - Die Hardware Pin List als Spiegel einer bereits vorhanden ECU Pin List zu erstellen.
  - Die Hardware Pin List aus den Signalen der aktuellen Hardwarekonfiguration zu erstellen. Bei dieser Variante werden auch automatisch virtuelle Verbindungen zwischen Hardwarepins und den entsprechenden Hardwaresignalen erstellt.

Als Ergebnis stehen dann Anschlüsse von Steuergerät und Hardware damit im Connection Manager zur Verfügung und können dort verbunden werden.

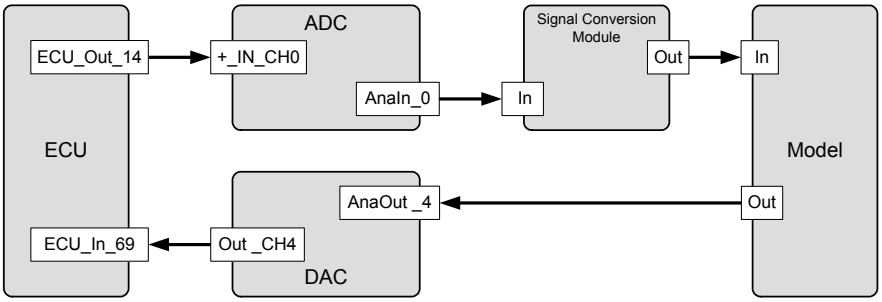


Abb. 3-32 Die Signale und die Pins von Steuergerät und Hardware

Virtuelle Verbindungen

Um eine Signalverfolgung zu ermöglichen, müssen auch die Zusammenhänge innerhalb eines Moduls bekannt sein.

In einfachen Fällen, wie bei einer Hardware-Karte, bei der die Liste „HW Pins“ automatisch aus der vorhandenen Konfiguration erzeugt wurde, kann dies automatisch geschehen, da der Pin und das Signal sozusagen fest verdrahtet sind. Ebenso ist es bei einem Modul zur Signalkonvertierung naheliegend, dass Eingang und Ausgang intern miteinander verbunden sind.

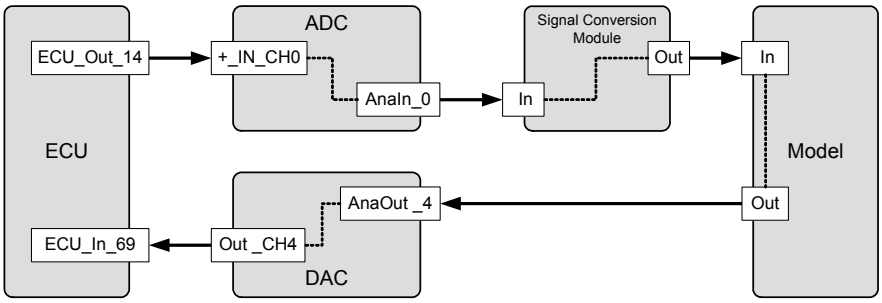


Abb. 3-33 Signale, Pins und virtuelle Verbindungen (gestrichelt)

Hinweis

Die Verbindungen zwischen Steuergerät und LABCAR Hardware sind insofern real, als dass Leitungen zwischen Steuergerät und Hardware angeschlossen werden.
Nichtsdestotrotz müssen diese Verbindungen im Connection Manager auch als virtuelle Verbindungen angelegt werden („ECU_Out_14“ und „+_IN_CH0“ sowie „ECU_In_69“ und „Out_CH4“ in Abb. 3-33). Dies ermöglicht ein vollständiges Tracing der Signale zwischen Steuergerät und LABCAR Hardware.

Wie Sie virtuelle Verbindungen erstellen, ist in „Virtuelle Verbindung erstellen“ auf Seite 250 beschrieben.

3.13.2 Inports und Outports zum Simulink-Modell hinzufügen

Um für HiL-Anwendungen Modellsignale mit Hardwaresignalen verbinden zu können, müssen die Modellsignale zugänglich sein. Dies wird mit den sogenannten LABCAR Inports und Outports bewerkstelligt.

Diese Blöcke können grundsätzlich auf jeder Hierarchieebene des Simulink-Modells platziert werden.

In MiL-Anwendungen kann ein Inport mit einem konstanten Wert belegt werden, so dass er sich wie ein Simulink „constant block“ verhält - LABCAR Outports verhalten sich dann wie Simulink „Terminator Blocks“.

Diese Inports und Outports finden Sie in der Library „LABCAR Port Blocks“ (Abb. 3-34).

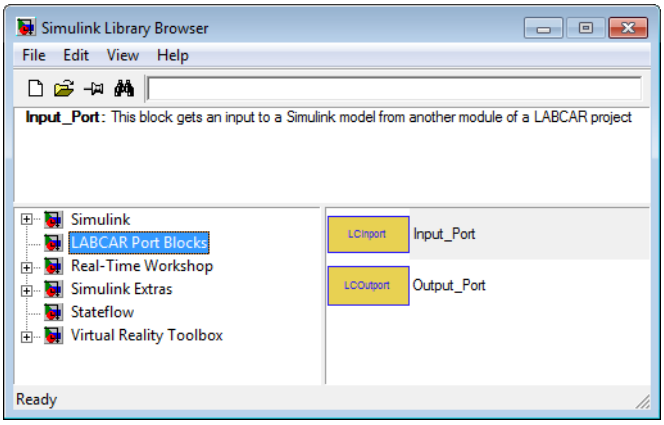


Abb. 3-34 LABCAR Port Blocks

Die Verbindungen werden im Connection Manager erstellt, der dafür eine einfach zu bedienende Bedienoberfläche zur Verfügung stellt (siehe „Signale verbinden im Connection Manager“ auf Seite 248).

Der Vorteil dieser Vorgehensweise ist, dass diese Definition der Schnittstelle unabhängig ist von der jeweils eingesetzten Hardware und somit auch für MiL-Anwendungen gültig ist. Zudem können Hardwareverbindungen auch geändert werden, nachdem der Modellcode bereits generiert wurde - dieser muss damit nicht neu generiert werden.

Hinweis

Sollten im Library Browser diese Port Blocks nicht vorhanden sein, beachten Sie bitte den folgenden Abschnitt.

LABCAR Port Blocks zum Library Browser hinzufügen

MATLAB durchsucht beim Öffnen des Simulink Browsers alle Pfade nach Dateien namens `slblocks.m`. Diese M-Files enthalten eine Beschreibung der Erweiterungen des Simulink Model Browers um entsprechende Elemente.

Im Falle von LABCAR-OPERATOR liegt diese Datei im Verzeichnis `<Laufwerk>:\Programme\ETAS\LABCAR-OPERATORX.Y\SiCo`.

Wenn Sie Ihr Simulink-Modell nicht aus LABCAR-OPERATOR geöffnet haben, sind die LABCAR Port Blocks nicht im Library Browser vorhanden. Fügen diesen Pfad im MATLAB Command Window mit

```
>> addpath('<Laufwerk>:\Programme\ETAS\
      LABCAR-OPERATORX.Y\SiCo');
```

hinzu.

Für den Fall, dass auf dem Rechner, auf dem Sie Ihr Simulink-Modell bearbeiten, LABCAR-OPERATOR noch nie installiert war, kopieren Sie bitte das komplette Verzeichnis `\SiCo` auf diesen und machen den jeweiligen Pfad (wie oben beschrieben) MATLAB bekannt.

Datentypen

Da LABCAR-RTC nur Daten von Typ „double“ verarbeitet, muss ein automatisches Casting von und zu den grundlegenden Simulink-Datentypen stattfinden:

- double (LABCAR Inport) → double, float, boolean, signed/unsigned int8, int16, int32, int64
- double, float, boolean, signed/unsigned int8, int16, int32, int64 → double (LABCAR Outport)

3.13.3 Signale verbinden im Connection Manager

Der Connection Manager bezieht all seine Informationen aus einer XML-Datei, in der die Informationen über alle Ein- und Ausgänge aller Module des Projekts gespeichert sind.

Die Bedienoberfläche des Connection Managers (erreichbar über das entsprechende Register des Hauptfensters) ist in der folgenden Abbildung gezeigt.

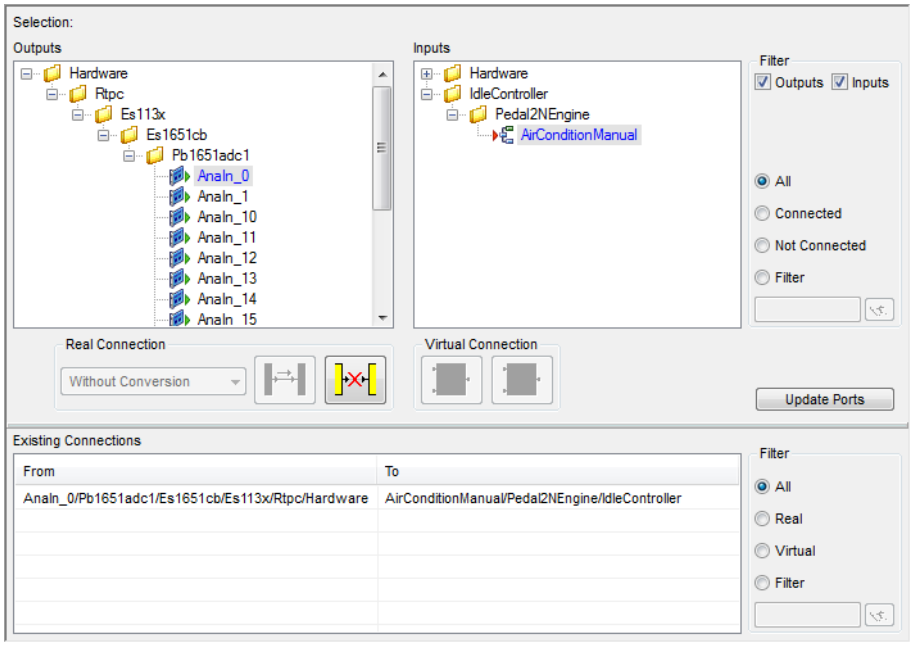


Abb. 3-35 Das Bedienfenster des Connection Manager

Im oberen Teil des Fensters sind alle Ein- und Ausgänge aller Module des Projekts dargestellt.

Signal im jeweiligen Editor anzeigen

- Wählen Sie in der Liste der Out- oder Inports ein Signal.
- Rechtsklicken Sie und wählen Sie **Goto...**.

  Der Editor für Signale des jeweiligen Typs wird geöffnet und das gewählte Signal wird hervorgehoben.

Filter über die Anzeige

Über die Anzeige der Signale lässt sich ein Filter (Feld „Filters“) legen, das die Übersicht über die Signale je nach Art der durchzuführenden Aufgabe erleichtert.

Folgende Filtereinstellungen sind möglich:

- **All**

  Hier werden alle verfügbaren Ein- und Ausgänge angezeigt.
- **Connected**

  Bei dieser Einstellung werden nur die verbundenen Ein- und Ausgänge angezeigt.
- **Not Connected**

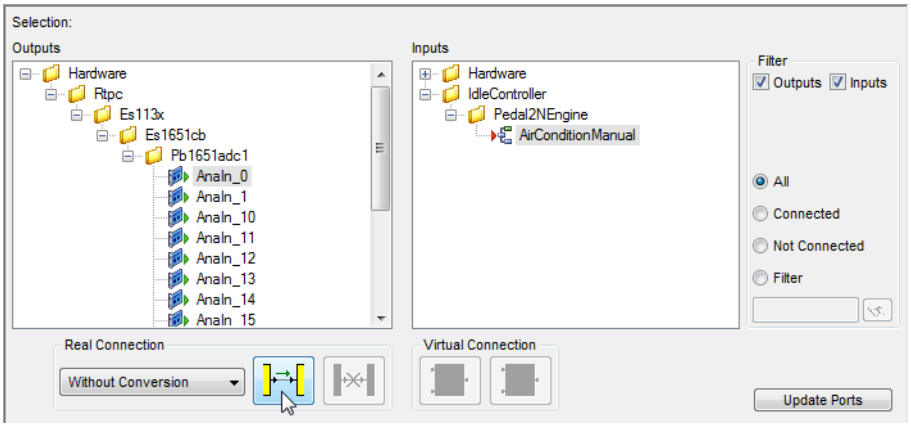
  Bei dieser Einstellung werden nur die nicht verbundenen Ein- und Ausgänge angezeigt.
- **Filter**

  Hier kann eine Zeichenkette eingegeben werden, die die Namen filtert. Zwischen Groß- und Kleinschreibung wird dabei nicht unterschieden.

Zudem kann gewählt werden, ob die Filterbedingung nur für Ausgänge („For Outputs“), nur für Eingänge („For Inputs“) oder für beides gelten soll.

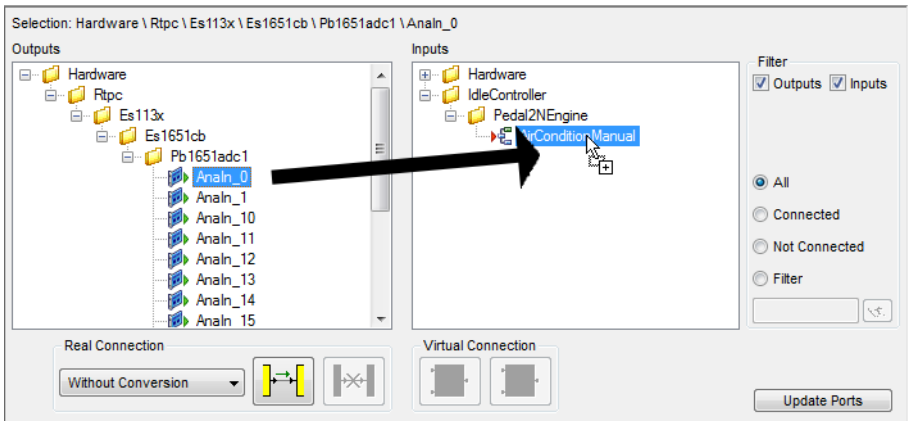
Signale verbinden

- Wählen Sie jeweils einen Eingang und einen Ausgang mit der Maus.
- Klicken Sie das Symbol zum Verbinden der gewählten Signale.

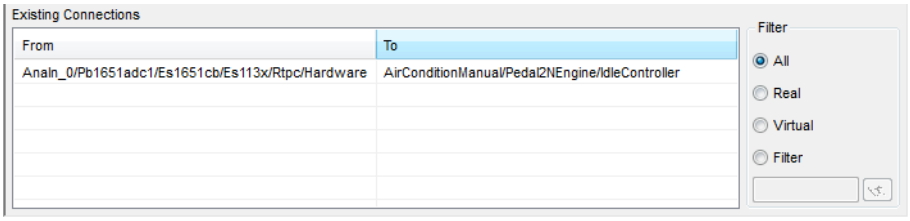


oder

- Wählen Sie einen Ein-/Ausgang mit der Maus und ziehen Sie ihn auf den gewünschten Aus- oder Eingang.

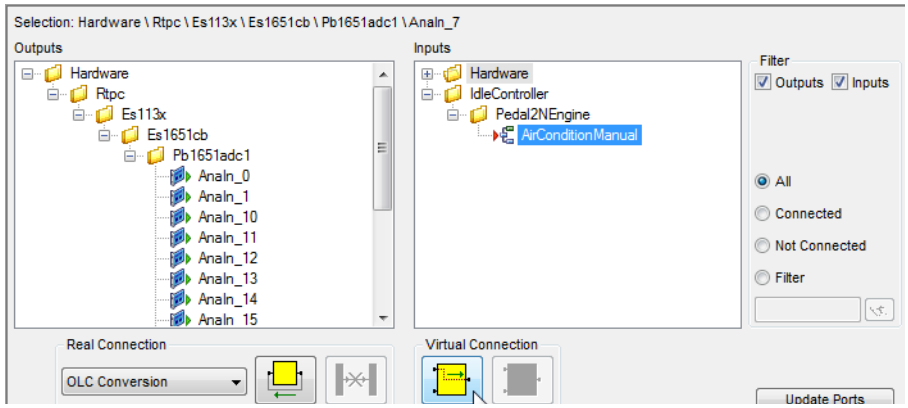


Die Verbindung wird erstellt und im Feld „Existing Connections“ dargestellt (sofern nicht ein Filter aktiv ist, der die neu erstellte Verbindung ausblendet).



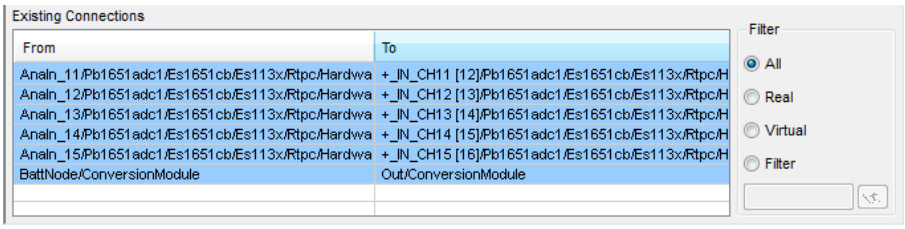
Virtuelle Verbindung erstellen

Um eine virtuelle Verbindung zwischen einem Eingang und einem Ausgang ein und desselben Moduls zu erstellen, gehen Sie wie bei realen Verbindungen vor (s.o.) – das Symbol zum Erstellen der virtuellen Verbindung (innerhalb eines Moduls) sieht nur anders aus als das zum Erstellen von realen Verbindungen (zwischen zwei Modulen).



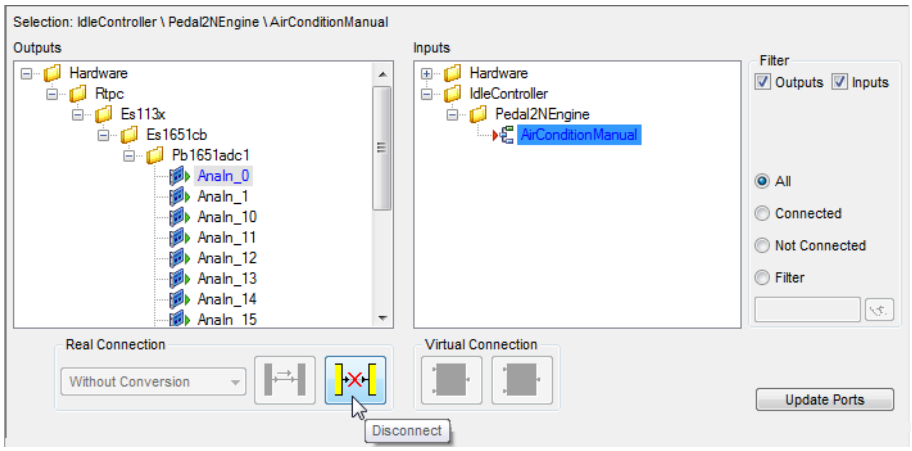
Daneben sehen Sie auch das Symbol zum Erstellen einer realen Verbindung zwischen einem Eingang und einem Ausgang desselben Moduls, was zur Rückkopplung von Signalen dienen kann.

Virtuelle Verbindungen werden im Connection Manager blau hinterlegt dargestellt.



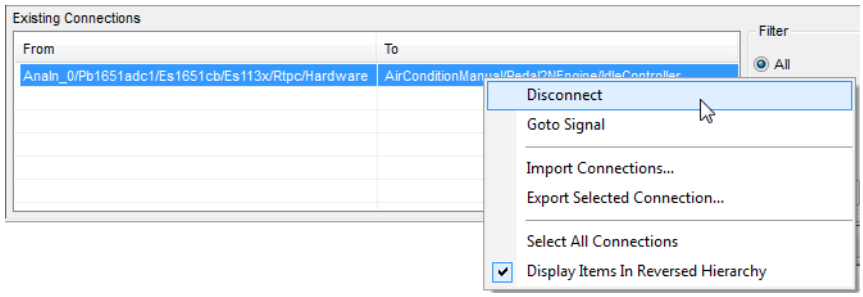
Verbindungen unterbrechen

- Wählen Sie eines der beiden Signale, deren Verbindung unterbrochen werden soll.
- Klicken Sie das Symbol zum Unterbrechen der Verbindung.



oder

- Wählen Sie unter „Existing Connections“ eine bestehende Verbindung.
- Drücken Sie die rechte Maustaste und wählen Sie **Disconnect**.



Die Verbindung wird wieder unterbrochen.

Wurden an einem Modul Änderungen vorgenommen, kann die Ansicht jederzeit aufgefrischt werden.

Ansicht aktualisieren

- Wählen Sie **Update Ports**.

  Die Ansicht wird aktualisiert.

Verbindungsliste sortieren

- Doppelklicken Sie die Spaltenüberschriften („Outputs“ oder „Inputs“) der Verbindungsliste.

  Die jeweilige Spalte wird alphabetisch geordnet (aufsteigend oder absteigend).

Namenshierarchie umkehren

- Wählen Sie im Kontextmenü der Verbindungsliste **Display Items in Reversed Hierarchy**.

  Damit wird die Hierarchie der Signalnamen umgekehrt. Sie beginnt jetzt mit dem Signalnamen und endet mit dem Modulnamen.

Verbindungseinstellungen exportieren

- Wählen Sie per Mausklick eine oder mehrere Verbindungen (Mehrfachauswahl mit gedrückter CTRL- oder SHIFT-Taste ist möglich).
- Wählen Sie im Kontextmenü **Export Selected Connection**.

  Ein Dateiauswahlfenster wird geöffnet.
- Geben Sie einen Dateinamen ein.

  Die gewählten Verbindungen werden in der Datei gespeichert.

Solche gespeicherte Verbindungseinstellungen können Sie wieder importieren.

Exportierte Einstellungen importieren

- Wählen Sie im Kontextmenü der Verbindungsliste **Import Connections**.

  Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie eine Datei mit Verbindungsdaten.

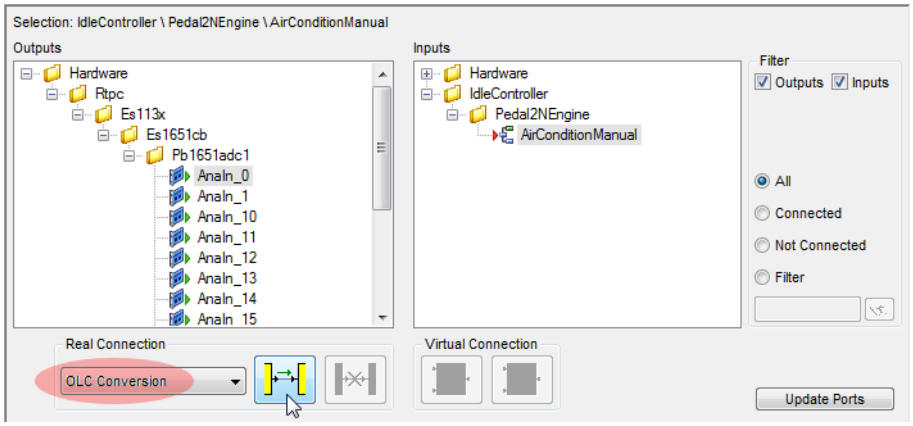
  Die gewählten Verbindungen werden importiert.

3.13.4 Signalkonvertierung

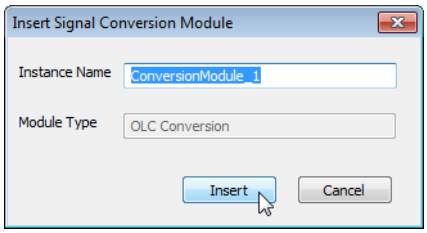
Im Connection Manager können zwischen die zu verbindenden Signale generische Signalkonvertierungsmodule (Aktor/Sensor-Module) eingefügt werden (siehe „Module zur Signalkonvertierung“ auf Seite 237) – die Konfiguration dieser Module erfolgt dann in einem speziellen GUI in ETAS EE.

Verbindung mit Signalkonvertierungsmodul erstellen

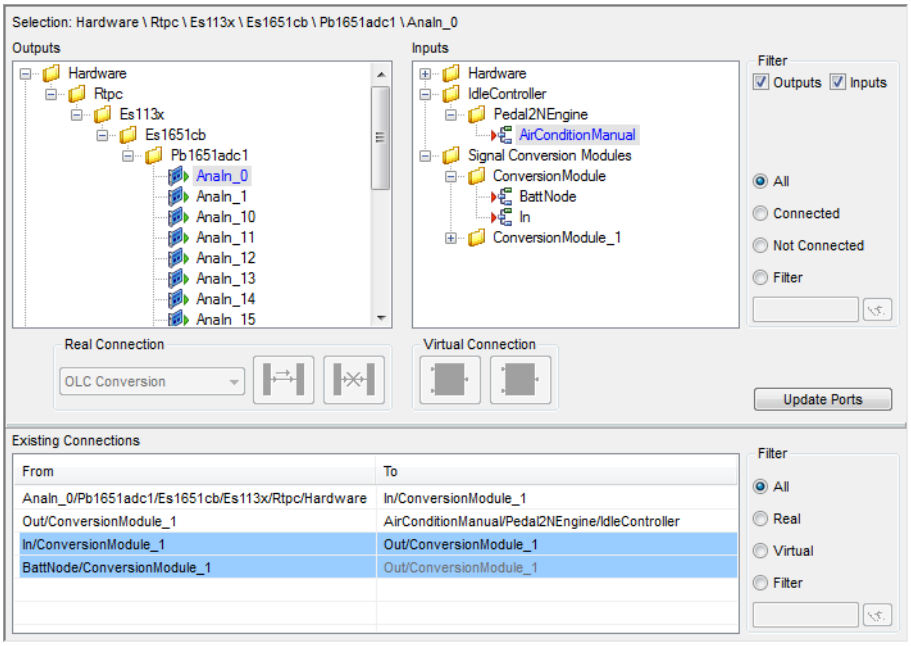
- Wählen Sie die zu verbindenden Signale.
- Wählen Sie **OLC Conversion** und klicken Sie die Schaltfläche daneben.



- Geben Sie im folgenden Fenster einen Namen für das Modul ein und klicken Sie **Insert**.

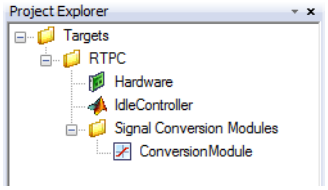


Die beiden Signale werden über ein solches Modul verbunden, indem beispielsweise der Hardware-ausgang mit dem Moduleingang und der Modul-ausgang mit dem Modelleingang verbunden wird.

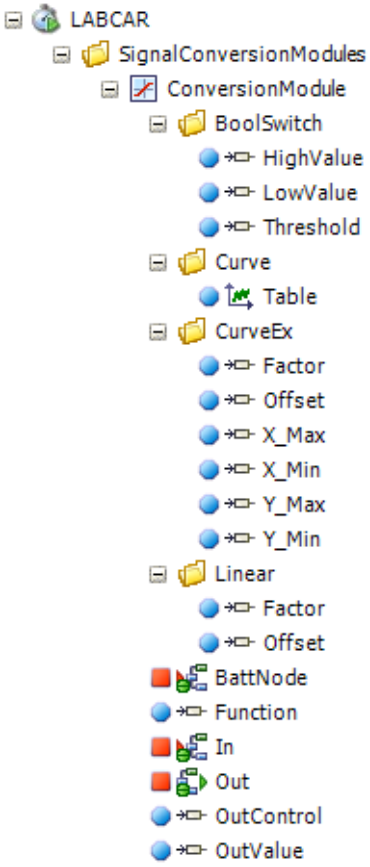


Ebenfalls angezeigt werden die (blau hinterlegten) virtuellen Verbindungen innerhalb des Konvertierungsmoduls.

Das eingefügte Modul wird auch im Project Explorer dargestellt.



Im Fenster „Workspace Elements“ in ETAS EE sind dann sämtliche Ein- und Ausgänge und Parameter des Moduls zugänglich.



3.14 Konfiguration des Echtzeit-Betriebssystems (OS-Konfiguration)

Bei der Erstellung eines LABCAR-OPERATOR Projektes wird automatisch eine Default-OS-Konfiguration erstellt und zum Projekt zugeordnet.

Die Konfiguration des Echtzeit-Betriebssystems erfolgt in LABCAR-IP in einem Bedienfenster, in dem die folgenden Aktionen durchgeführt werden können:

- Hinzufügen und Entfernen von Tasks
- Editieren der Task-Eigenschaften
- Zuweisen und Entfernen von Prozessen zu/von Tasks

Der Umgang mit dem Bedienfester für die OS-Konfiguration wird in Abschnitt „Arbeiten mit OS-Konfigurationen“ auf Seite 263 beschrieben.

Wird ein neues Modul eingebunden, so werden die jeweiligen Prozesse automatisch in die OS-Konfiguration übernommen und geeigneten Tasks zugewiesen.

3.14.1 Elemente einer OS-Konfiguration

Die folgenden Abbildungen zeigen jeweils Ausschnitte des Registers „OS Configuration“, in denen bestimmte Einstellungen vorgenommen werden.

Globale Einstellungen („OS Settings“)

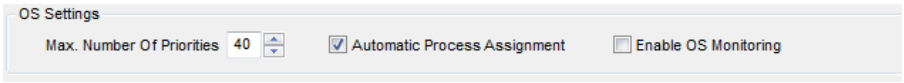


Abb. 3-36 Globale Einstellungen

- **Max. Number of Priorities**

  Maximale Anzahl der Prioritätslevel

- **Automatic Process Assignment**

  Diese Option bewirkt, dass beim Hinzufügen eines neuen Moduls zum LABCAR-OPERATOR-Projekt alle Prozesse des neu hinzugefügten Moduls automatisch den entsprechenden Task zugewiesen werden.

Hinweis

Bereits vorhandene Zuweisungen von Prozessen anderer Module sind davon nicht berührt!
*Sollten es aufgrund manueller Änderungen an den automatischen Zuweisungen zu Fehlern kommen, können diese nur behoben werden, indem **alle Prozesse** aus ihren Tasks entfernt werden, so dass es beim nächsten Build des Projektes zu einer erneuten automatischen Zuweisung kommt.*

- **Enable OS Monitoring**

  Wenn diese Option aktiviert ist, werden nach dem darauffolgenden Buildvorgang eine Reihe zusätzlicher Messgrößen bei den Workspace Elements in ETAS EE hinzugefügt (unter „OSMonitoring“). Diese Größen liefern Informationen über die Laufzeiten der einzelnen Tasks.

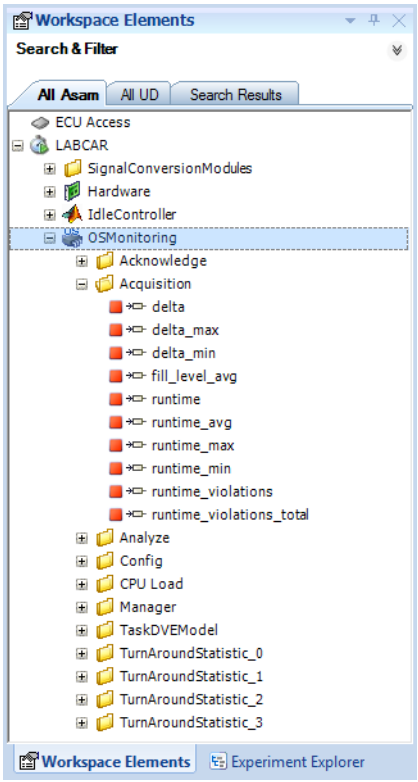


Abb. 3-37 Zusätzliche Messgrößen bei „Enable OS Monitoring“

OS Monitoring von Tasks

Die Bedeutung der Messgrössen ist wie folgt:

- delta (delta_max, delta_min)

  Aktuelle Zeit dt (in ns) zwischen zwei Taskaktivierungen (maximale, minimale Zeit)

- fill_level_avg

  Der Quotient aus durchschnittlicher Ausführungsdauer durch gesamte Periodendauer.

  Beispiel: Wenn ein Task mit einer Periode von 100ms aufgerufen wird und im Durchschnitt (!) 20 ms Ausführungsdauer hat, dann beträgt der Füllstand 20%.

- runtime (runtime_avg, runtime_max, runtime_min)

  Aktuelle Laufzeit (in ns) der Task (durchschnittliche/maximale/minimale Laufzeit)

- runtime_violations

  Integer, die jedesmal um Eins inkrementiert wird, wenn die aktuelle Laufzeit („runtime“) die aktuelle Periode („delta“) überschreitet.
  Diese Größe wird periodisch zurückgesetzt – die Länge einer Periode bis zur Rücksetzung der Messgröße wird im Web-Interface von ETAS RTPC über den Parameter RTPC_TASK_TIMING_STATISTIC (**Main Page → Configuration**) definiert („RTPC_TASK_TIMING_STATISTIC = 0“ deaktiviert die Aufzeichnung).

- runtime_violations_total

  Zahl der „runtime violations“ während der gesamten Ausführungzeit des Experiments.

TurnAroundStatistic_n

In den Ordnern „TurnAroundStatistic_*n* (n ist die Nummer des VME-Chassis und entspricht dem Ethernetadapter „ETHn“, mit dem dieses an den Real-Time PC angeschlossen ist) befinden sich die vier Größen „runtime“, „runtime_avg“, „runtime_min“ und „runtime_max", die auch bei jeder Task zu finden sind.

Die folgende Abbildung zeigt, wie „runtime“ berechnet wird.

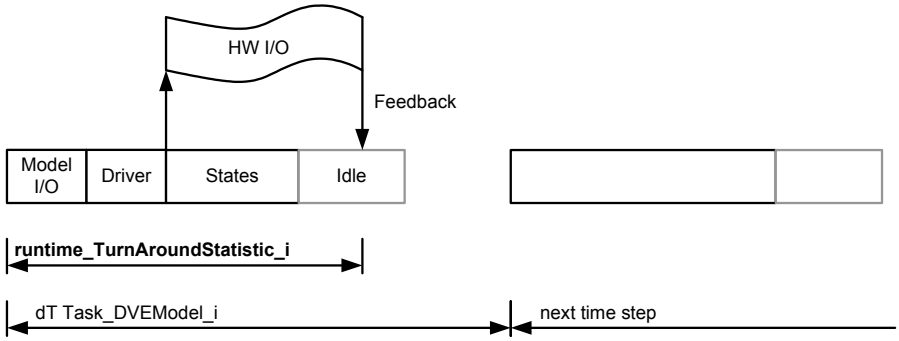


Abb. 3-38 Berechnung der Größe „runtime_TurnaroundStatistic“

CPU Load

Im Ordner „CPU Load“ befinden sich die Messgrößen „Core *n*“, die die Last eines jeden Prozessorkerns enthalten. Dies soll eine optimale Verteilung der Tasks auf die verschiedenen Kerne (siehe „CPU Core“ auf Seite 259) erleichtern.

Task-Einstellungen

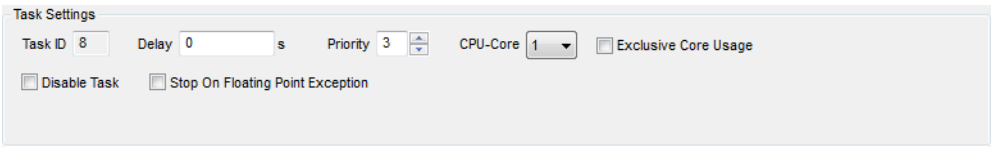


Abb. 3-39 Task-Einstellungen

- **Show Unassigned Processes**

  Wenn diese Option aktivert ist, werden im Feld „Processes“ (siehe „Prozesse“ auf Seite 260) nur diejenigen Prozesse angezeigt, die keiner Task zugewiesen wurden.

- **Update Processes**

  Damit werden alle Module des Projekts erneut auf die vorhandenen Prozesse untersucht.

- **Type**

  Task-Typ (siehe „Tasks und deren Typen“ auf Seite 261)

- **Period** (nur bei Timer Tasks)

  Periode der Timer Task in Sekunden

- **Task ID**

  Die ID einer Task (nicht editierbar) - über diese ID werden beispielsweise Software-Tasks aufgerufen (siehe „Tasks und deren Typen“ auf Seite 261)

- **Delay**

  Verzögerung der Taskaktivierung in Sekunden

- **Priority**

  Hier ist es möglich, einer bestimmten Task eine Priorität zuzuordnen – je größer die Zahl, desto höher die Priorität bei der Ausführung. Die Anzahl der zur Verfügung stehenden Prioritätslevels kann im Feld „OS Settings“ unter „Max. Number of Priorities“ eingestellt werden (siehe „Globale Einstellungen („OS Settings“)“ auf Seite 256).

Hinweis

Das Simulationstarget RTPC arbeitet mit präemptivem Taskscheduling – hierbei wird (innerhalb gewisser Zeitfenster) immer die Task mit der höheren Priorität im jeweiligen Prozessorkern abgearbeitet.

- **CPU Core**

  Der Prozessorkern, der diese Task berechnet.

Hinweis

Zur verteilten Simulation (d.h. zur Zuordnung verschiedener Tasks zu verschieden Kernen) müssen mehr als zwei Kerne zur Verfügung stehen!

- **Exclusive Core Usage**

  Mit dieser Option wird sicher gestellt, dass der zuvor gewählte Prozessorkern für exklusive Nutzung durch diese Task reserviert ist.

Hinweis

Die Einstellung „Exclusive Core Usage“ ist dafür gedacht, Tasks mit sehr kurzen Zykluszeiten und mit möglichst geringer Latenz auszuführen (z.B. Hardware-I/O). Die „Idle“-Zeit einer Task wird bei dieser Betriebsart mit „Busy-Waiting“ ausgefüllt. Folglich ist der jeweilige Prozessorkern ständig zu 100 % ausgelastet.

- **Disable Task**

  Zustand der Task: „aktivert“ oder „deaktiviert“

- **Stop Simulation on Floating Point Exception**

  Diese Option ist nur dann verfügbar, wenn ein Real-Time PC als Simulationstarget verwendet wird.

  Ist diese Option aktiviert, so wird beim Auftreten eines ungewöhnlichen Wertes einer Gleitkommazahl („Not-a-number") eine „Floating-Point Exception" erzeugt und das Target angehalten. Näheres zu den Möglichkeiten, die Ursache des Fehlers zu ermitteln, finden Sie im aktuellen Handbuch von ETAS RTPC.

Bei **Tasks vom Typ „Trigger"** gibt es noch eine weitere Option:

- **Auto Trigger**

  Damit wird erreicht, dass eine Trigger Task trotz Ausbleiben des Triggers gerechnet wird (was sonst zum Stillstand der Simulation führen würde).

- **Period**

  Solange ein Triggerereignis ausbleibt, wird die Task mit dieser Periode berechnet.

- **Event Timeout**

  Wenn nach Ablauf dieser Zeit kein Triggerereignis stattgefunden hat, wird die Auto Trigger Funktion gestartet.

Prozesse

Die folgende Abbildung zeigt eine Liste von Prozessen, wie sie in der Bedienoberfläche dargestellt wird.

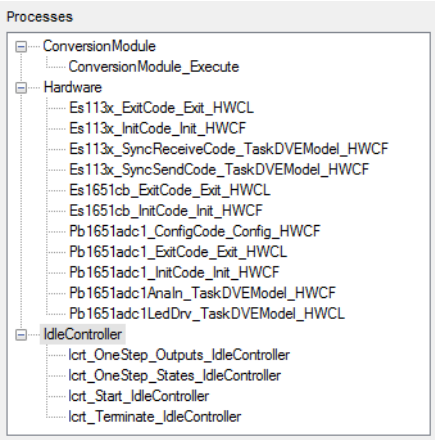


Abb. 3-40 Das Feld „Processes" des Registers „OS Configuration"

Tasks und deren Typen

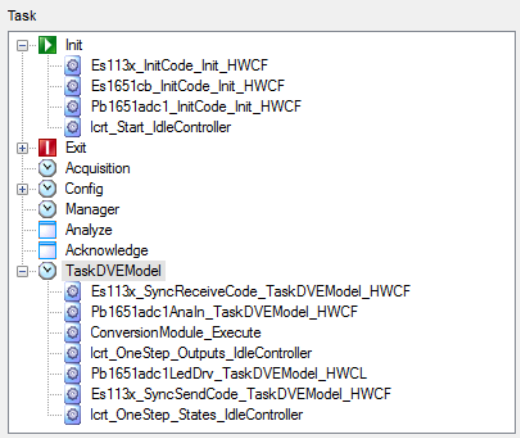


Abb. 3-41 Das Feld „Tasks“ des Registers „OS Configuration“

- **Init**

  Wird immer beim Start der Simulation ausgeführt

- **Exit**

  Wird immer beim Beenden der Simulation ausgeführt

- **Timer**

  Dieser Tasktyp wird in einem bestimmten periodischen Zeitraster (festgelegt durch „Period“) ausgeführt

- **Event**

  Spezielle Task, die von ETAS RTPC zur ereignisgesteuerten Kommunikation verwendet werden kann. Ein Prozess innerhalb dieser Task muss einen „blocking call“ ausführen, der auf Input von einer Hardware oder dem Netzwerk wartet.

  Ein typischer Anwendungsfall ist ein CAN-Zugriff mit Timeout. Die Task wartet (blockiert) bei dem Funktionsaufruf `rtos_comm_read()` so lange, bis eine CAN Message (per Interrupt signalisiert) ankommt. Dies ermöglicht eine sehr schnelle Reaktion auf ankommende CAN Messages.

- **Software**

  Task, die mit `activateTaskWithId(uint32 taskId)` aufgerufen werden kann

- **Trigger**

  Eine Task, die von externen Ereignissen (Hardware, Netzwerk, …) getriggert werden kann.

Die folgende Tabelle enthält die Tasks einer Default OS-Konfiguration, deren Typen und weitere Einstellungen.

Task	Typ	Periode [sec]	Prio	Bemerkung
Acquisition	Timer	0.1	1	Für Messung
Init	Init	-		RTIO
Exit	Exit	-		RTIO
Config	Timer	0.1	1	RTIO
Manager	Timer	0.01	3	RTIO

Tab. 3-2 Tasks einer Default OS-Konfiguration

Abarbeitungsreihenfolge der Prozesse

Grundsätzlich werden die Prozesse in der Reihenfolge abgearbeitet, in der sie unter die jeweiligen Prozesse eingehängt wurden (siehe z.B. Abb. 3-41 auf Seite 261)

Durch Änderungen der Reihenfolge kann ggf. eine Laufzeitoptimierung erzielt werden. z.B. wenn verschiedene Simulink-Modelle gerechnet werden.

Hinweis

*„Outputs“ müssen immer **vor** „States“ gerechnet werden!*

3.14.2 Arbeiten mit OS-Konfigurationen

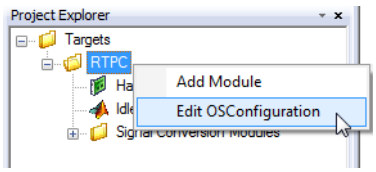
In diesem Abschnitt erhalten Sie eine Anleitung, wie Sie mit OS-Konfigurationen arbeiten können. Im Einzelnen sind dies:

OS Configuration einsehen

- Wählen Sie das Register „OS Configuration“

oder

- Wählen Sie im Project Explorer das Target (z.B. RTPC).
- Rechtsklicken Sie und wählen Sie **Edit OS Configuration**.



Das Register „OS Configuration“ wird geöffnet, das die aktuelle OS-Konfiguration des Projektes enthält.

- Um alle Prozesse und alle Tasks mit den zugewiesenen Prozessen darzustellen, klappen Sie die verschiedenen Einträge auf.

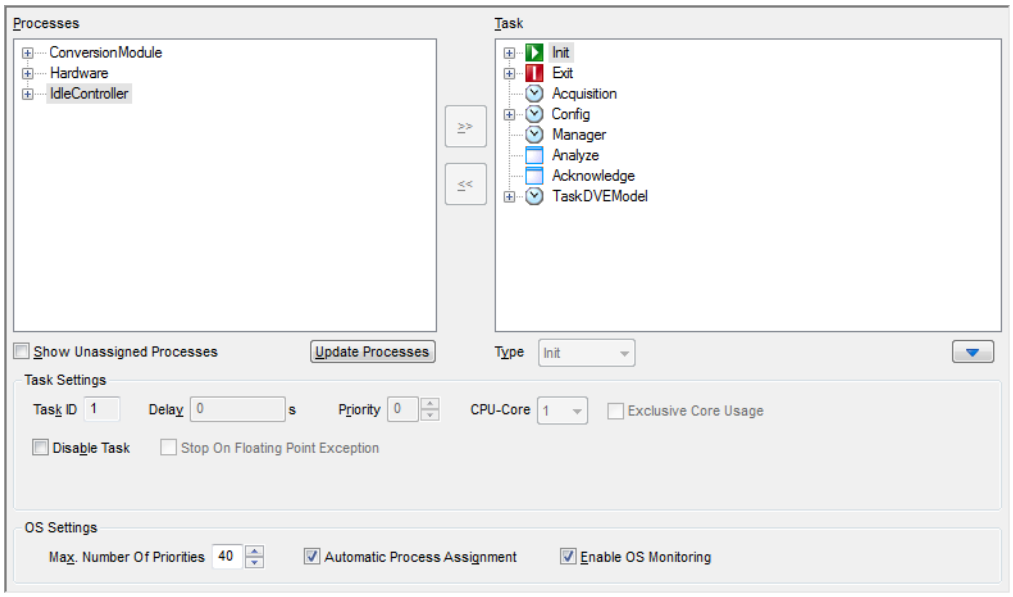


Abb. 3-42 Das Register „OS Configuration“ (erweiterte Ansicht)

Erweiterte Einstellungen anzeigen

- Klicken Sie das Pfeil-Symbol.

  Das Fenster wird erweitert um die Felder „Task Settings“ und „OS Settings“.

Das Feld „Processes“

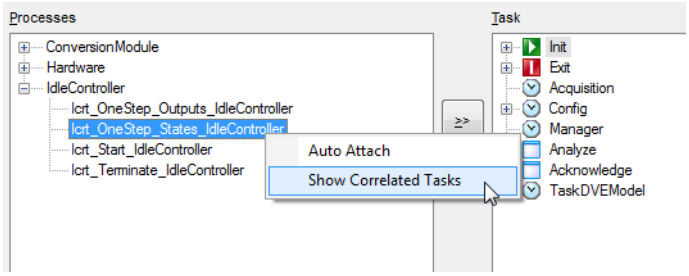
Die Baumansicht in diesem Feld zeigt alle verfügbaren Prozesse der Hardware als auch des Modells. Sollten vorhandene Prozesse nicht zu einer Task (Feld rechts) zugewiesen worden sein, so können diese durch Aktivieren der Option „Show Unassigned Processes“ sichtbar gemacht werden.

Sollten Prozesse nicht zu Tasks zugewiesen worden sein, wählen Sie die Funktion **Auto Attach** aus dem Kontextmenü (s.u.).

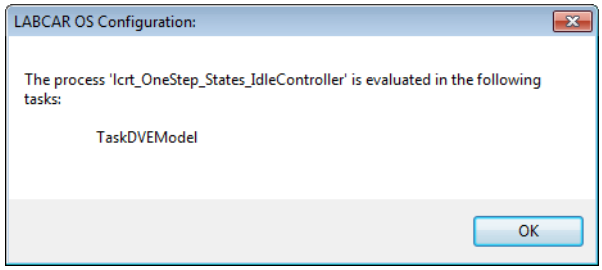
Wenn Sie herausfinden wollen, zu welcher Task ein bestimmter Prozess zugewiesen wurde, gehen Sie wie folgt vor:

Zuordnung zu Task ermitteln

- Wählen Sie im Feld „Processes“ einen Prozess.
- Im Kontextmenü wählen Sie **Show Correlated Tasks**.



Es wird ein Fenster geöffnet, in dem die Task angegeben ist, der der gewählte Prozess zugewiesen wurde.



Das Feld „Tasks“

Dieses Feld zeigt alle verfügbaren Tasks und die diesen zugeordneten Prozesse an.

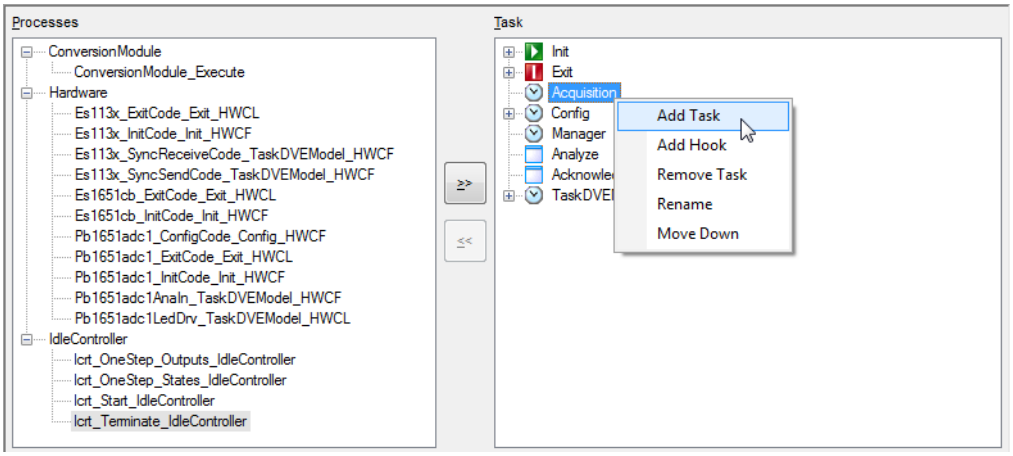
Eine Task hinzufügen

- Wählen Sie die Task, unterhalb der die neue Task eingefügt werden soll.

Hinweis

Sie können die Reihenfolge der Tasks auch später ändern (siehe „Reihenfolge der Tasks ändern“ auf Seite 267).

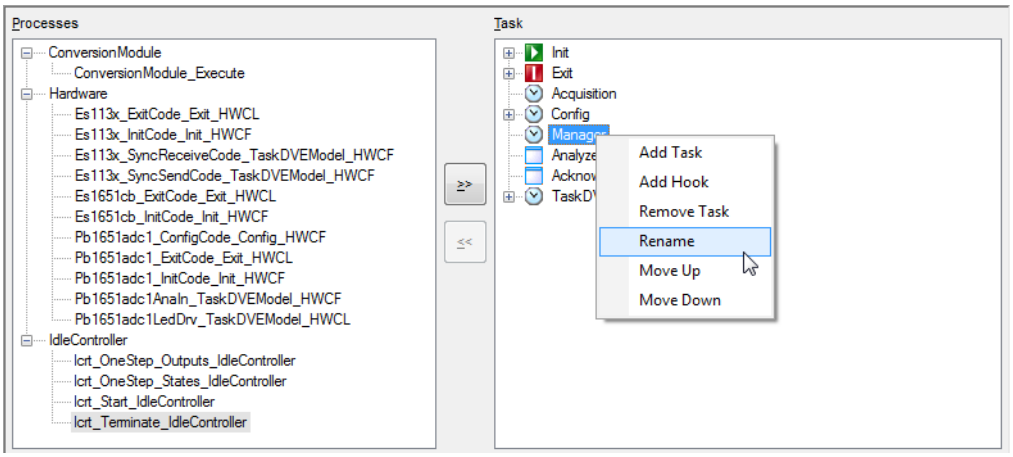
- Im Kontextmenü wählen Sie **Add Task**.



Eine neue Task mit dem Namen „Task_*n*“ (n ist eine laufende Nummer) wird eingefügt.

Eine Task umbenennen

- Wählen Sie die umzubenennende Task.
- Im Kontextmenü wählen Sie **Rename**.



Der Taskname ist jetzt editierbar.

Task-Einstellungen zuweisen

- Weisen Sie der neuen Task unter „Type“ einen Typ zu.
- Legen Sie die „Task Period“ fest.

Eine Task entfernen

- Wählen Sie die zu entfernende Task.
- Im Kontextmenü wählen Sie **Remove Task**.

  Die Task wird entfernt.

Reihenfolge der Tasks ändern

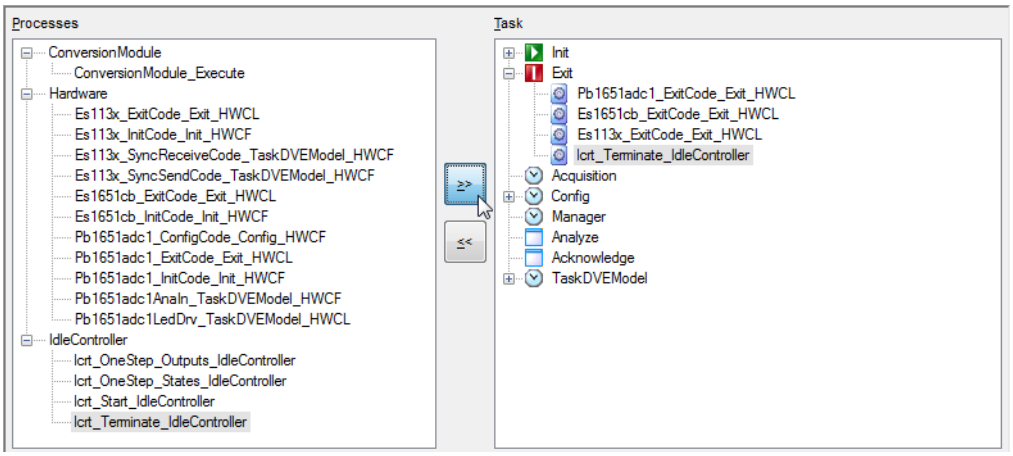
- Wählen Sie die zu verschiebende Task.
- Im Kontextmenü wählen Sie **Move Up** oder **Move Down**.

  Die gewählte Task wird entsprechend verschoben.

Prozess zu einer Task hinzufügen

Um beispielsweise den Prozess „lcrt_Terminate_IdleController“ zur „Exit“ Task zuzuordnen, gehen Sie wie folgt vor:

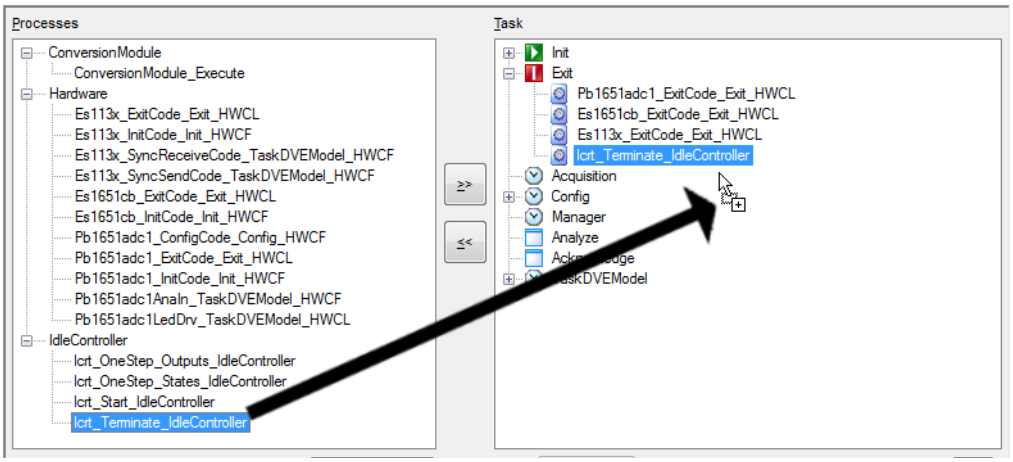
- Wählen Sie den zuzuweisenden Prozess im Feld „Processes“.
- Wählen Sie im Feld „Tasks“ den Prozess einer Task, unterhalb der der Prozess zugewiesen werden soll.
- Klicken Sie die Schaltfläche **>>**.



oder

- Wählen Sie den zuzuweisenden Prozess.

- Ziehen Sie ihn mit gedrückter Maustaste auf den Prozess, unter welchem er zugewiesen werden soll.



Der Prozess wird unterhalb des gewählten Prozesses eingefügt.

Prozess aus einer Task entfernen

- Wählen Sie den zu entfernenden Prozess.
- Im Kontextmenü wählen Sie **Remove Process**.

  Der gewählte Prozess wird aus der Task entfernt.

Einen Hook erstellen

Um innerhalb einer Task einen Software-Hook zu erstellen, gehen Sie wie folgt vor:

- Wählen Sie die Task, innerhalb derer der Hook erstellt werden soll.
- Im Kontextmenü wählen Sie **Add Hook**.

  Der Hook wird erstellt.

Einen Hook umbenennen

- Um den Namen des Hooks zu ändern, wählen Sie **Rename**.

Einen Hook entfernen

- Um den Hook wieder zu entfernen, wählen Sie **Remove**.

3.15 Erstellen von Multi-RTPC-Netzwerken

Abb. 3-43 zeigt den Aufbau und die Bestandteile eines Multi-RTPC-Netzwerkes

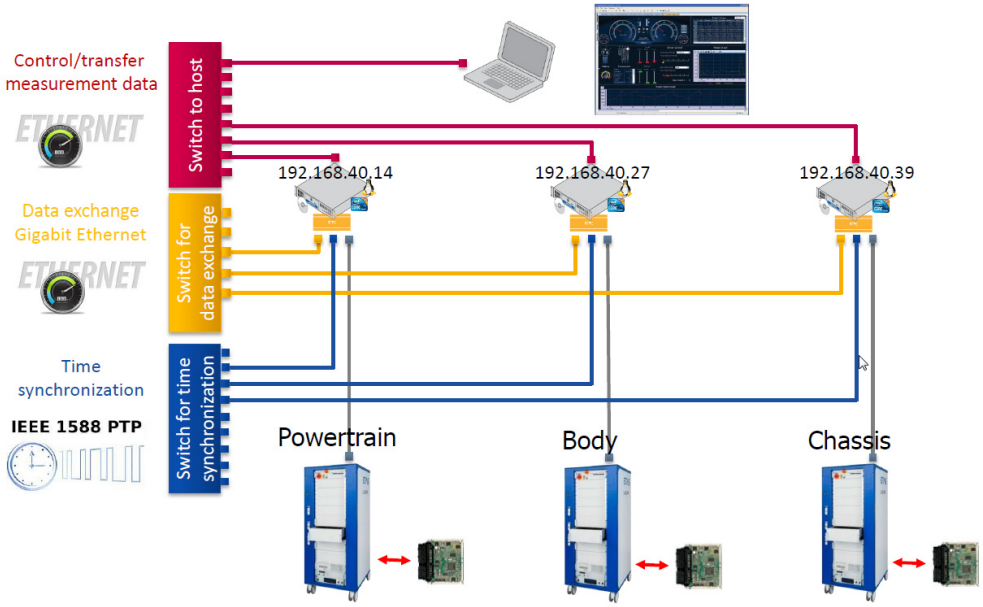


Abb. 3-43 Architektur eines Multi-RTPC-Systems

Folgende Komponenten und Funktionen sind dafür erforderlich:

- Die Steuerung und der Transfer von Messdaten zwischen Bedien-PC und den Real-Time PCs
- Der Datenaustausch zwischen den Real-Time PCs
- Die Zeitsynchronisation zwischen den Real-Time PCs

Hardwareseitig sind die jeweiligen Funktionen/Komponenten über Netzwerk-Switches verbunden.

Synchronisation der Uhren der Real-Time PCs mit PTP 1588

Die Synchronisation der Uhren erfolgt mit dem Precision Time Protokoll (PTP, definiert in IEEE 1588 und in IEC 61588 übernommen), mit dem Geräte in einem Netzwerk mit hoher Genauigkeit synchronisiert werden können. Mit PTP-fähigen Netzwerkkarten liegen die Zeitabweichungen unter 1 µs, bei Softwareimplementierungen liegen die Abweichungen unter 1 ms.

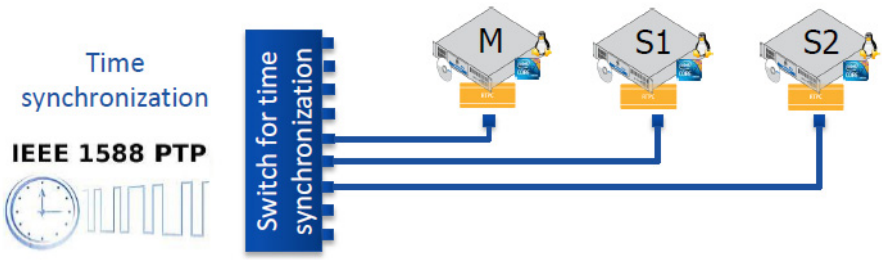


Abb. 3-44 Uhrensynchronisation mit PTP

Echtzeit-Datenaustausch zwischen den Real-Time PCs

Der Datenaustausch zwischen den Modellen/Projekten, die auf den unterschiedlichen Real-Time PCs ausgeführt werden, erfolgt über Echtzeitschnittstellen - eine Bibliothek für den UDP-Datentransfer in Echtzeit wird mit ETAS RTPC mitgeliefert.

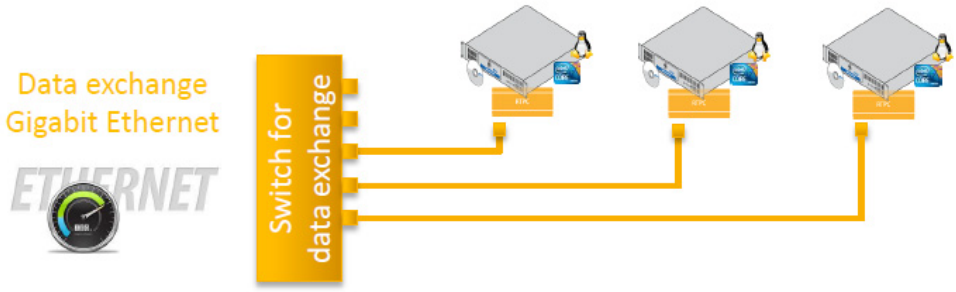


Abb. 3-45 Datenaustausch zwischen den Real-Time PCs

Die möglichen Transferraten variieren entsprechend der Anzahl der PCs und dem Messagetyp (Point-to-Point, Multicast / Broadcast).

Hardwareanforderungen

Hieraus ergeben sich folgende Anforderungen an die Hardware eines Multi-RTPC-Netzwerkes:

- So viele Real-Time PCs wie erforderlich – jeder ausgerüstet mit einer PTP-fähigen Ethernetkarte mit jeweils mindestens drei Ethernet-Ports.
- Drei Ethernet-Switches mit jeweils soviel Ports wie „Anzahl der Real-Time PCs + 1“.
- Ethernetkabel (2 x Anzahl der Real-Time PCs + 1)
- Ein Bedien-PC, auf dem LABCAR-OPERATOR installiert ist

3.15.1 Hardware anschließen

Switch für Verbindung zwischen den Real-Time PCs und dem Bedien-PC

Verbinden Sie den eth0-Anschluss eines jeden Real-Time PC mit dem entsprechenden Switch und verbinden Sie diesen mit dem Bedien-PC.

Switch für die Zeitsynchronisation (über PTP) zwischen den Real-Time PCs

Verbinden Sie jeweils einen Ethernet-Port eines jeden Real-Time PC (z.B. eth1) mit dem dafür vorgesehenen Switch.

Switch für den Datentransfer zwischen den Real-Time PCs

Verbinden Sie jeweils einen Ethernet-Port eines jeden Real-Time PC (z.B. eth2) mit dem dafür vorgesehenen Switch.

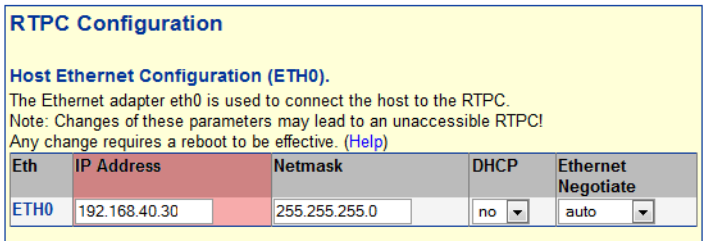
3.15.2 IP-Adressen der Real-Time PCs festlegen

Der Ethernetadapter „eth0“ wird für die Verbindung zum Bedien-PC verwendet.

Hinweis

Führen Sie die folgenden Schritte zur Festlegung der IP-Adressen für alle Real-Time PCs nacheinander durch!

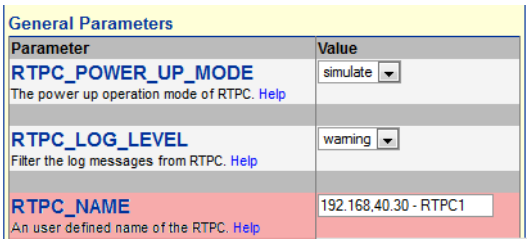
- Schließen Sie den jeweiligen Real-Time PC an den Bedien-PC an.
- Starten Sie den Real-Time PC.
- Öffnen Sie das Web-Interface von ETAS RTPC und navigieren Sie zum Abschnitt „RTPC Configuration“.



Hinweis

192.168.40.14 ist eine „Standardadresse“ für einen Real-Time PC – vergeben Sie Adressen ab 192.168.40.30 aufwärts!

- Für eine bessere Übersicht sollten für die einzelnen Real-Time PCs Namen vergeben werden.



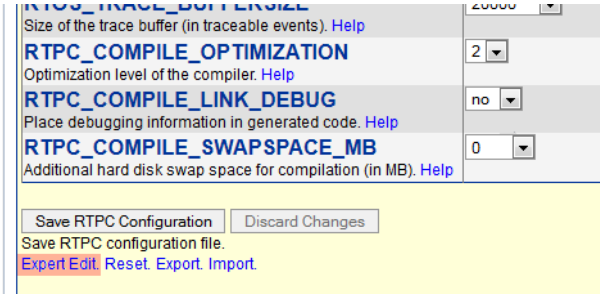
- Führen Sie einen Neustart durch (**Main Page** → **Power Control** → **Reboot RTPC**).

3.15.3 Konfiguration der PTP- und Datenverbindungen im Web-Interface

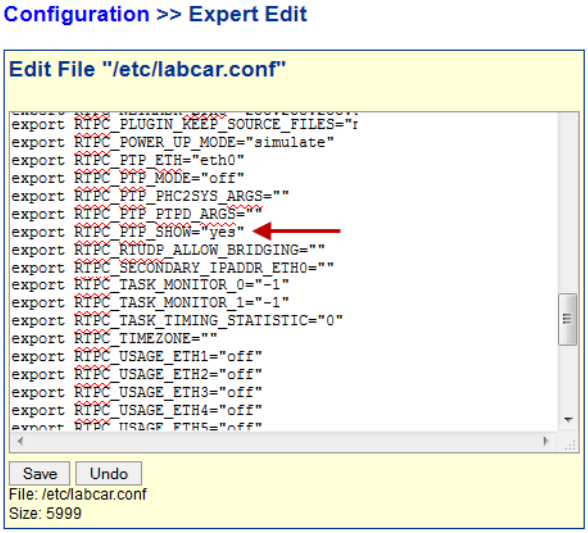
Um PTP in ETAS RTPC zu aktivieren, gehen Sie wie folgt vor:

PTP aktivieren

- Navigieren Sie im Web-Interface von ETAS RTPC zum Abschnitt „RTPC Configuration“.
- Klicken Sie **Expert Edit**.

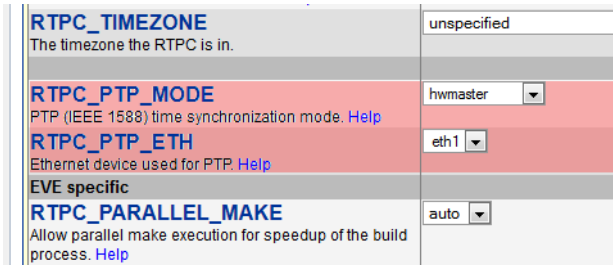


Die Konfigurationsdatei wird im Editor geöffnet.



- Setzen Sie den Wert des Parameters „RTPC_PTP_SHOW“ auf „yes“.
- Klicken Sie **Save** und kehren Sie zurück zu **Configuration**.

  Jetzt sind zwei zusätzliche Parameter sichtbar.



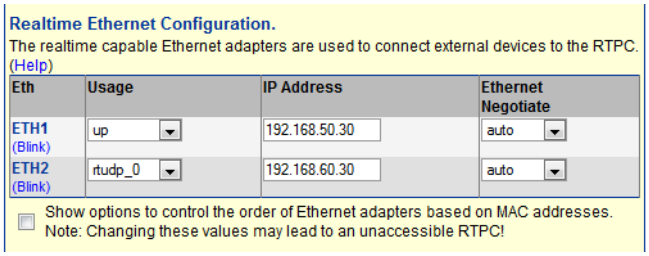
- Für weitere Informationen klicken Sie **Help**.

Um PTP- und Datenverbindungen in ETAS RTPC zu aktivieren, gehen Sie wie folgt vor:

PTP- und Datenverbindung konfigurieren

- Bei „RTPC_PTP_MODE“ wählen Sie für den PTP-Master die Option „hwmaster“, für die anderen Real-Time PCs „hwslave“.
- Weisen Sie (unter „Realtime Ethernet Configuration“) dem Port für den PTP-Switch eine IP-Adresse und die „Usage“ = „up“ zu.

  Im Beispiel unten ist dies „ETH1“ mit der IP-Adresse 192.168.50.30.



- Die „Usage“ für den Ethernet-Port für den Datenaustausch (hier: ETH2, 192.168.60.30) sollte für alle Real-Time PCs zu „rtudp_0“ gesetzt werden.

Hinweis

Da alle PTP- und Datenports jeweils mit dem selben Switch verbunden sind, müssen diese verschiedene IP-Adressen haben. Es ist hilfreich, wenn die Adressen aller Ports an einem Real-Time PC mit den selben Ziffern enden (hier: 30).

- Klicken Sie **Save RTPC Configuration**.
- Führen Sie einen Neustart durch.

Abb. 3-46 auf Seite 274 Abb. 3-46zeigt noch einmal eine Zusammenfassung der Konfigurationsparameter.

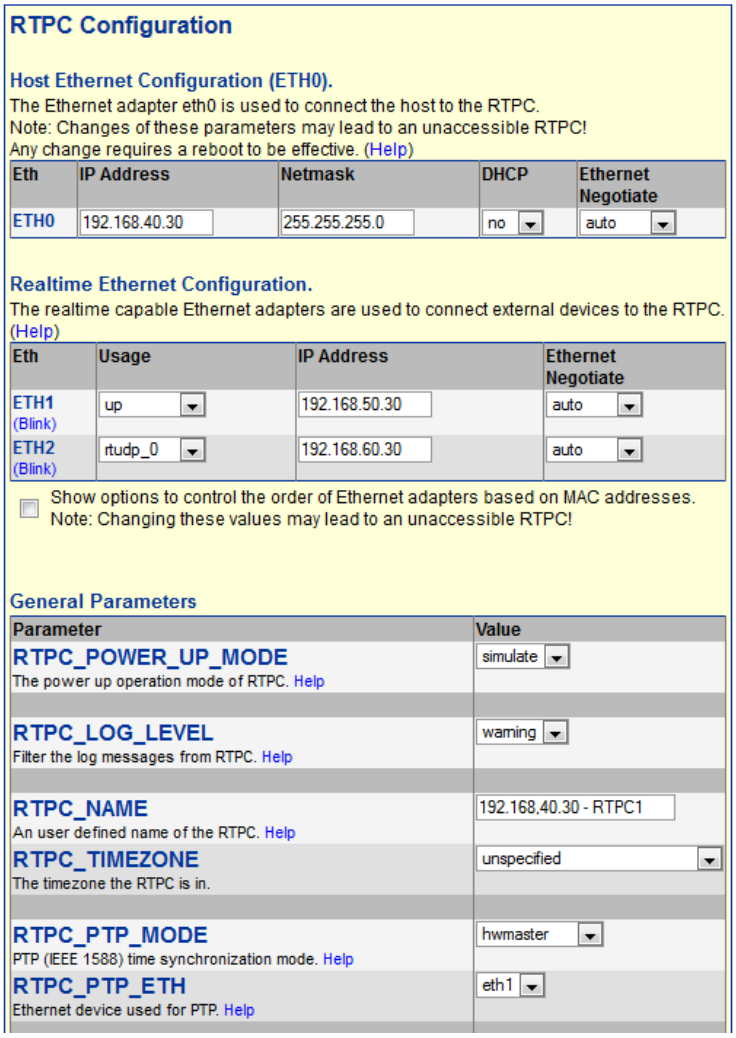
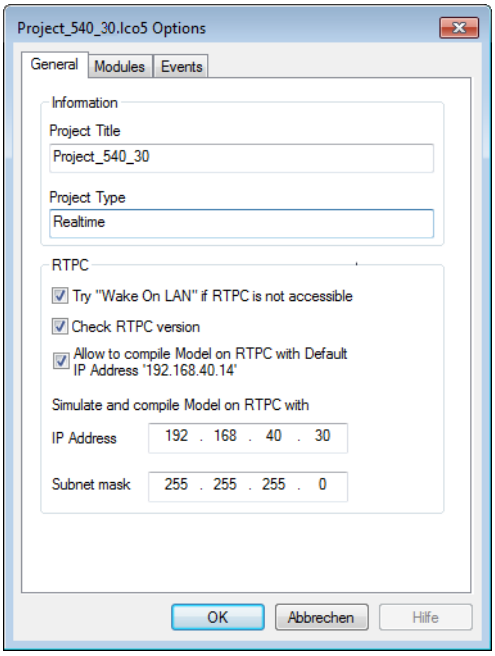


Abb. 3-46 Einstellungen für einen Real-Time PC im Web-Interface

3.15.4 LABCAR-OPERATOR-Projekte erstellen

Für jeden Real-Time PC des Netzwerkes muss ein LABCAR-OPERATOR-Projekt erstellt werden.

Unter **Project → Options** können Sie die IP-Adresse des Real-Time PCs einstellen, auf dem das Projekt erstellt und ausgeführt wird.



Für den Fall, dass Targets des Netzwerkes nicht zur Verfügung stehen, ermöglicht die Option „Allow to compile...“ das Kompilieren des Projekts auf dem „Standardtarget“ mit der IP-Adresse 192.168.40.14.

Hinweis

Beachten Sie bitte, dass die Module aller Projekte (einschließlich der Hardwaremodule) unterschiedliche Namen haben müsse

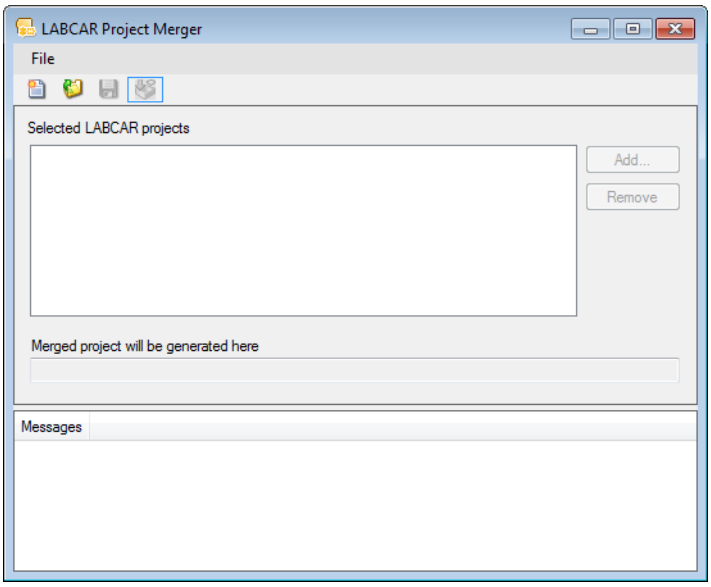
3.15.5 Projekte zusammenführen

Um die einzelnen Projekte zusammenzuführen, gehen Sie wie folgt vor:

Projektkonfiguration erstellen

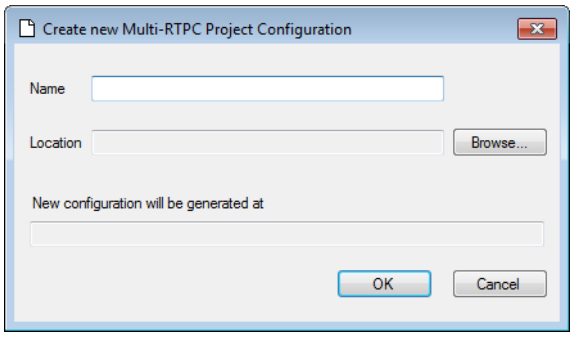
- Starten Sie das Programm *<LABCAR-OPERATOR Installationsverzeichnis>*`/Project-Merger/ProjectMerger.exe`.

  Der LABCAR Project Merger wird gestartet.



- Wählen Sie **File** → **New**.

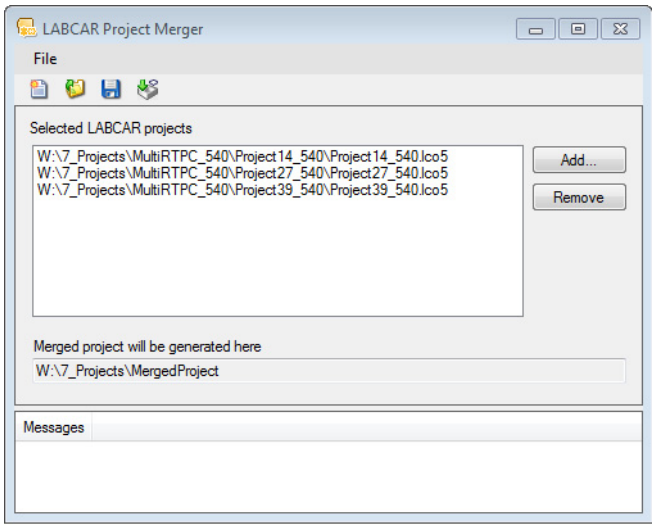
  Das Fenster „Create New Multi-RTPC Project Configuration“ wird geöffnet.



- Legen Sie einen Namen und ein Verzeichnis für die Projektkonfiguration fest.
- Klicken Sie **OK**.

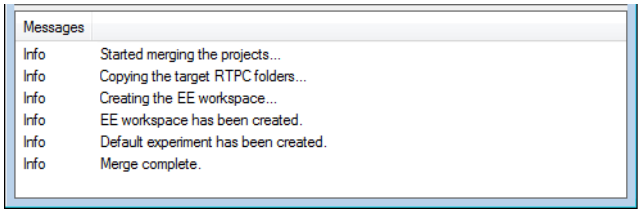
  Im gewählten Verzeichnispfad wird nun ein Verzeichnis mit dem angegebenen Namen angelegt und in diesem die Konfigurationsdatei *<Name>*`.multirtpc` erstellt.

- Wählen Sie mit **Add** die LABCAR-OPERATOR-Projekte, die Sie zu einem Multi-RTPC-Projekt zusammenführen wollen.



- Wählen Sie **File** → **Save**, um die Konfiguration zu speichern.
- Wählen Sie **File** → **Merge**, um das Multi-RTPC-Projekt zu erzeugen.

  Das Projekt wird erstellt und Informationen (insbesondere Fehlermeldungen) werden im Bereich „Messages“ aufgelistet.



4 ETAS Experiment Environment - Eine Übersicht

ETAS Experiment Environment (ETAS EE) dient zur Ausführung eines LABCAR-OPERATOR-Experimentes. In diesem Kapitel finden Sie eine Übersicht über die Bedienoberfläche und die Funktionen von ETAS EE.

Hinweis

Dieses Kapitel soll nur einen Überblick vermitteln – für Details verwenden Sie bitte die Online-Hilfe in ETAS EE.

Die folgenden Abschnitte enthalten Informationen über:

4.1 Die Bestandteile der Bedienoberfläche

Wenn Sie aus LABCAR-IP heraus ein Experiment in der Experimentierumgebung starten, sieht die Bedienoberfläche von ETAS EE ungefähr so aus.

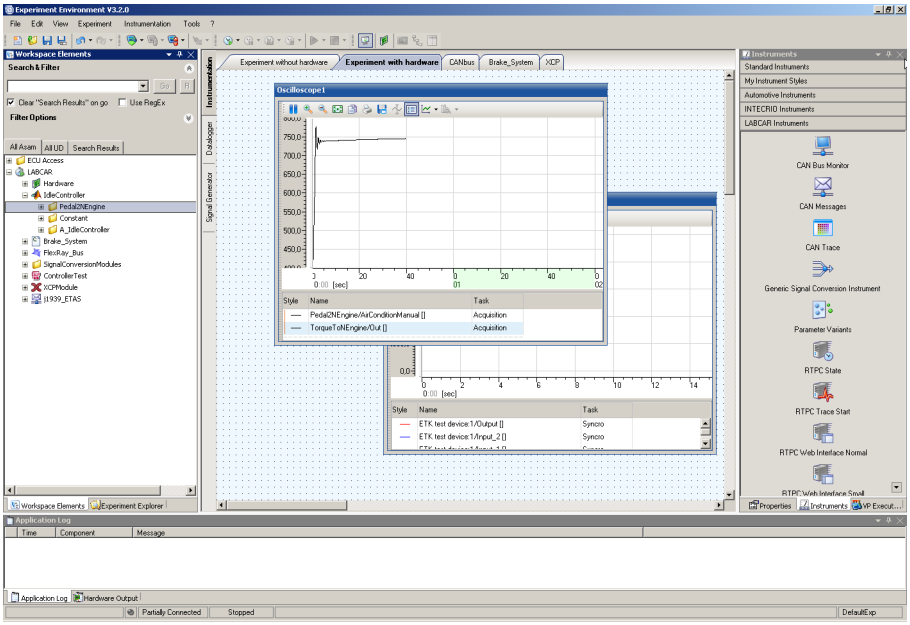


Abb. 4-1 Die Bedienoberfläche von ETAS EE

Im Folgenden werden die einzelnen Bestandteile dieser Bedienoberfläche beschrieben.

4.1.1 Das Fenster „Workspace Elements“

Im Fenster „Workspace Elements“ sind alle Messgrößen, Parameter, Signaleingänge und -ausgänge der Module des Experiments zugänglich gemacht.

Darüber hinaus bietet dieses Fenster Filterfunktionen bezüglich Hierarchien, Element-, Daten- und Objekttyp und Suchfunktionen mit regulären Ausdrücken.

4.1.2 Der Experiment Explorer

Der Experiment Explorer enthält alle Dateien, die zur Konfiguration oder bei der Ausführung eines Experimentes benötigt werden, wie z.B. Parameterdateien, Konfigurationsdateien für den Datalogger, Signal Generator Sets usw..

4.1.3 Der Hauptarbeitsbereich

Im Hauptfenster sind (über mehrere Register) die wichtigen Bedienoberflächen der Experimentierumgebung zugänglich.

Das Register „Instrumentation“

Hier erstellen Sie die Layer, die wiederum die verschiedenen Instrumente (in Gruppen vereinigt) für die Durchführung des Experiments enthalten.

Das Register „Signal Generator“

In diesem Register werden die Signalgeneratoren des Experiments konfiguriert und bedient.

Das Register „Datalogger“

In diesem Register wird der Datalogger konfiguriert und bedient.

4.1.4 Das Fenster „Instruments“

In diesem Fenster finden Sie alle Instrumente, die Sie im Experiment verwenden können.

4.1.5 Das Fenster „Properties“

In diesem Fenster werden die Eigenschaften und Metadaten eines jeden in der Bedienoberfläche gewählten Objektes wie z.B. Parameter oder Instrumente angezeigt.

4.1.6 Die Fenster „Application Log“

Hier werden Informationen, Warnungen und Fehlermeldungen der Anwendung angezeigt.

4.1.7 Das Hauptmenü von ETAS EE

Hinweis

Die folgende Beschreibung der Menüstruktur von ETAS EE soll nur einen Überblick vermitteln – ausführlichere Informationen zu den einzelnen Funktionen erhalten Sie in der Online-Hilfe von ETAS EE.

Das Menü „File“

In diesem Menü sind alle dateibezogenen Aktionen (Workspace und Experiment) zusammengefasst.

Das Menü „Edit“

Bietet „Undo“ und „Redo“ Funktion.

Das Menü „View“

In diesem Menü können die weiter oben beschriebenen Bestandteile der Bedienoberfläche sichtbar gemacht bzw. abgeschaltet werden. Außerdem kann in den Vollbildmodus umgeschaltet werden (<F11>).

Das Menü „Experiment“

In diesem Menü sind alle Funktionen vereinigt, die mit dem Download des Simulationscodes zum Target und der Steuerung der Simulation selbst zu tun haben.

Das Menü „Instrumentation“

Dieses Menü enthält Funktionen zur Verwaltung der Layer im Register „Instrumentation“.

Das Menü „Tools“

Hier sind die verschiedenen Optionen für das Experiment zugänglich.

Das Menü „?“

In diesem Menü finden Sie den Zugang zur Online-Hilfe, der Lizensierung und zu Programm- und Kontaktinformationen.

4.1.8 Die Werkzeugleiste

Die Werkzeugleiste besteht aus vier einzelnen Gruppen.

File

Enthält Funktionen zur Verwaltung von Experimenten (Neu, Öffnen etc.).

Experiment

Enthält Funktionen zum Download und Verbinden zum Target.

Simulation

Enthält Funktionen zur Steuerung von Simulation und Messung.

Instrumentation

Enthält vielgenutzte Funktionen für den Umgang mit Instrumenten.

4.2 Der Experiment Explorer

Der Experiment Explorer enthält alle Dateien, die zur Konfiguration oder bei der Ausführung eines Experimentes benötigt werden, wie z.B. Parameterdateien, Konfigurationsdateien für den Datalogger, Signal Generator Sets usw..

Diese Daten können von hier aus aktiviert, bearbeitet oder wieder aus dem Experiment entfernt werden.

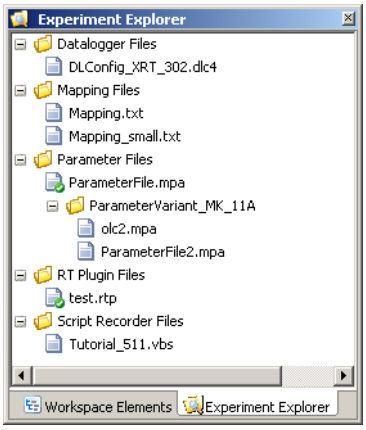


Abb. 4-2 Der Experiment Explorer

4.2.1 Arbeiten mit dem Experiment Explorer

Das Arbeiten mit Ordnern und Dateien des Experiment Explorers erfolgt über die Kontextmenüs (rechter Mausklick) der jeweiligen Objekte.

Funktionen für alle Ordner

- **Add File**

  Öffnet ein Dateiauswahlfenster zur Auswahl einer Datei, die hinzugefügt werden soll

Funktionen für bestimmte Ordner

Ordner „Parameter Files“:

- **Set Display Order**

  Öffnet ein Dialogfenster, in dem die Reihenfolge festgelegt werden kann, in der die vorhandenen Parameterdateien dargestellt werden (d.h. zum Target heruntergeladen werden).

- **Download all active files**

  Damit werden (bei bestehender Verbindung zu einem Experiment) alle aktiv gesetzten Parameterdateien zum Target herunter geladen.

- **Add New Parameter Variants**

  Hier erstellen Sie einen Unterordner und fügen diesem eine Parameterdatei (Variant) hinzu.

Funktionen für alle Dateien

- **Add File**

  Öffnet ein Dateiauswahlfenster zur Auswahl einer Datei, die hinzugefügt werden soll

- **Rename File**

  Benennt die jeweilige Datei um

- **Exclude From Experiment**

  Entfernt die Datei aus dem Experiment (Datei wird nicht gelöscht)

- **Delete (permanently)**

  Entfernt die Datei aus dem Experiment und löscht die Datei

Funktionen für bestimmte Typen von Dateien

- **Edit File**
  - Dateien (*.txt, *.smf) im Ordner „Mapping Files“

    Öffnet eine Mappingdatei im SUT Mapping File Editor
  - Dateien im Ordner „Parameter Files“

    *.mpa: Öffnet die Parameterdatei in LABCAR-PA (Parameterization Assistant) (siehe „Parameterdateien bearbeiten mit LABCAR-PA V1.0“ auf Seite 341)

    *.dcm: Öffnet einen Texteditor zum Bearbeiten der Datei

- **Active**

  Damit markieren Sie eine Datei als „aktiv“. Diese Datei wird bei diesem Vorgang und in Zukunft immer beim Herunterladen des Experiments zum Target zusammen mit allen andere aktiven Dateien herunter geladen.

  – Dateien (*.dlc4) im Ordner „Datalogger Files“

    Aktiviert die ausgewählte Datalogger-Konfiguration

  – Dateien (*.txt) im Ordner „Mapping Files“

    Aktiviert die ausgewählte Mappingdatei

  – Dateien (*.mpa, *.dcm, *.cdfx or *.olc4) im Ordner „Parameter Files“

    Aktiviert die ausgewählte Parameterdatei oder Open-Loop Konfiguration

- **Set Display Order**

  – Dateien im Ordner „Parameter Files“

    Öffnet ein Dialogfenster, in dem die Reihenfolge festgelegt werden kann, in der die vorhandenen Parameterdateien dargestellt werden (d.h. zum Target heruntergeladen werden).

- **Reload Mapping**

  – Dateien im Ordner „Mapping Files“

    Aktualisiert das Mapping entsprechend dem Inhalt der gewählten Datei

4.3 Das Fenster „Workspace Elements“

Das Fenster „Workspace Elements“ ist eines der wichtigsten Fenster der Experimentierumgebung ETAS EE. Die Anordnung der Objekte in diesem Fenster erfolgt in Ordnern für

- Simulink- und ASCET-Modelle
- C-Module (auch Signalkonvertierungsmodule)
- CAN- und FlexRay-Module
- FiL-Module
- Hardware
- ECU Access

In diesen Ordnern haben Sie

- Zugriff auf alle Parameter und Messgrößen
- Zugriff auf alle Ein- und Ausgänge (Signale) der Module
- Zugriff auf Hardware- und Steuergeräteanschlüsse

und können Sie die Ansicht durch Setzen von Filtern und Auswahl von Suchkriterien beeinflussen.

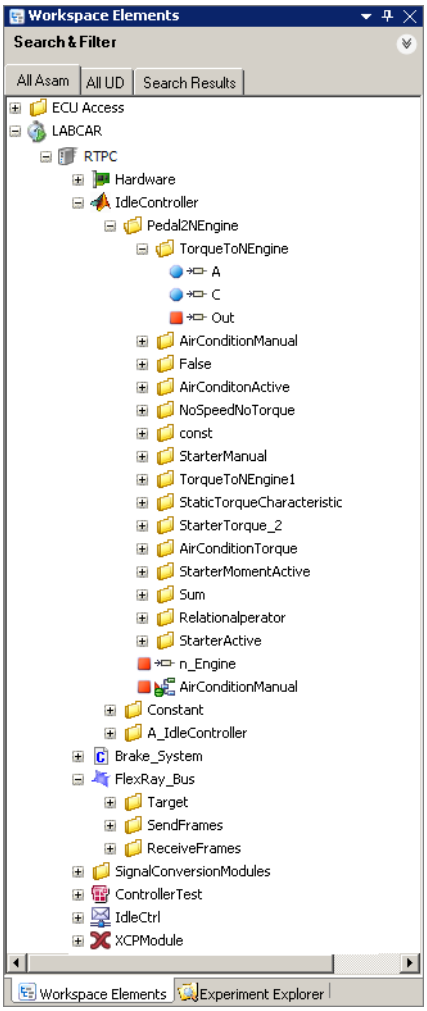


Abb. 4-3 Das Fenster „Workspace Elements“

4.3.1 Die Register des Fensters „Workspace Elements“

Das Fenster „Workspace Elements“ besitzt drei Register, die verschiedene Ansichten auf die Elemente enthalten:

- Register „All ASAM“

  Standard-Baumansicht mit allen Elementen in Hierarchien, die auf dem ASAM-Label oder der Modulspezifikation basieren.

- Register „All UD“

  In diesem Register erfolgt die Darstellung wie oben, nur dass hier die vom Anwenders definierten Namen aus der Mappingdatei verwendet werden.

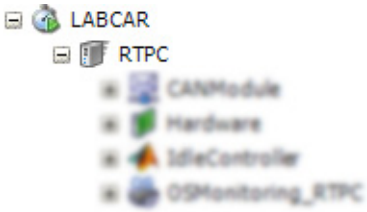
- Register „Search Result“

  In diesem Register werden die Ergebnisse der letzten Suche (siehe „Such- und Filterfunktion“ auf Seite 289) angezeigt.

4.3.2 Arbeiten mit den Workspace Elements

In Abb. 4-3 auf Seite 285 sind verschiedene Elemente durch unterschiedliche Symbole gekennzeichnet.

Das Element „LABCAR“ bildet die oberste Ebene, gefolgt von der Ebene, die den Real-Time PC symbolisiert (hier: „RTPC“), dem das Projekt zugeordnet ist (in einem Multi-RTPC-Projekt können mehrere Real-Time PCs vorhanden sein).



Unter dem jeweiligen Real-Time PC befinden sich die verschiedenen Module des Projektes.

Module

Die unterschiedlichen Typen von Modulen werden mit folgenden Symbolen dargestellt:

- ASCET- und MATLAB/Simulink-Module

- C-Code-Module

- CAN-, LIN- und FlexRay-Module

- FiL-Module

- Module für Open-Loop-Zugriff und Signalumwandlung

- I/O-Hardwaremodule

Messgrößen und Parameter von Modulen

Die Messgrößen und Parameter der einzelnen Module werden mit folgenden Symbolen dargestellt:

- Messgröße

- Parameter

- Modulausgang (Messgröße)

- Moduleingang (Messgröße)

- Hardwareausgang (zu Modul) (Messgröße)

- Hardwareeingang (von Modul) (Messgröße)

- Hardwarepin (Ausgang)

- Hardwarepin (Eingang)

- Steuergerätepin (Ausgang)

- Steuergerätepin (Eingang)

- CAN Send Message

- CAN Receive Message

Hinweis

Das Arbeiten mit Parametern wird in „Mit Parametern arbeiten" auf Seite 321 ausführlich beschrieben.

Kontextmenüs

Zum Arbeiten mit den Elementen des Fensters gibt es für jedes Element ein Kontextmenü, das je nach Typ des Elements mehr oder weniger Funktionen besitzt:

- **Measure / Calibrate**

  Öffnet das Dialogfenster „Create Instruments" zur Zuweisung dieses Elements zu einem vorhandenen oder neu zu erstellenden Instrument

- **Find in Instruments**

  Ermittelt, ob Messgröße/Parameter einem Instrument zugeordnet wurde und gibt das Ergebnis im Fenster „Application Log“ aus. Doppelklicken des jeweiligen Eintrags bringt den Layer, in dem die Messgröße oder der Parameter gefunden wurden, in den Vordergrund.

- **Show All HW Ports**

  Üblicherweise werden im Fenster „Workspace Elements“ nicht alle Hardwaresignale (z.B. Monitoring-Signale oder Signale zum Einstellen von Messmodi) dargestellt. Deren vollständige Darstellung kann mit dieser Option aktiviert werden.

- **Add to Datalogger**

  Fügt das gewählte Element zu einem Datalogger hinzu. Sind mehrere Datalogger konfiguriert, kann ein spezieller gewählt werden.

- **Calibration / Parameters**

  Diese Funktionen sind nur bei Parametern vorhanden und nur aktiv, wenn das Experiment zum Target herunter geladen wurde.

  – **Copy Values**

    Kopiert den Wert des Parameters in die Zwischenablage

  – **Paste Values**

    Weist dem gewählten Parameter den in der Zwischenablage befindlichen Wert zu

  – **Save**

    Öffnet ein Dialogfenster zum Speichern von Parametern in mpa-Dateien. Durch Drücken von <UMSCHALT> können mehrere Parameter ausgewählt werden.

  – **Set State To Modified/Unmodified**

    Ändert den Status eines Parameters von „geändert“ (rot) nach „nicht geändert“ (schwarz)

- **Signal**

  Diese Funktion ist nur bei Eingängen von Modulen und Hardware vorhanden und besitzt folgende Untermenüs:

  – **Trace**

    Öffnet das Dialogfenster zur Signalverfolgung

  – **Save Input Signal Settings**

    Speichert die Einstellungen und Werte für einen Eingang in einer Datei (*.olc4)

  – **Reset Full Path to „Model“**

    Setzt den gesamten Pfad mit dem jeweiligen Signal auf den Wert „MODEL“, um eventuell geöffnete Pfade wieder zu schließen

  – **Reset Input Setting To**

    Zum Ändern der OLC-Einstellungen kann in einem Dialogfenster ein bestimmter Wert („MODEL“, „CONST“, etc.) gewählt werden

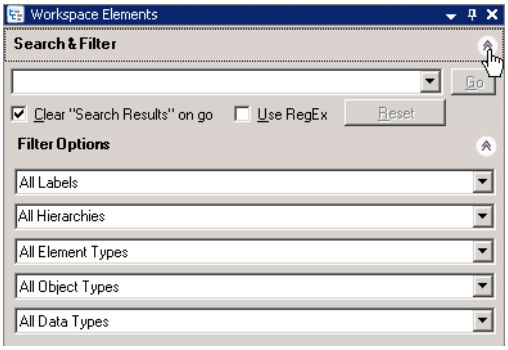
- **CAN**

  Wenn eine Simulation läuft, kann (im Kontext von CAN-Messages) das Senden mit **Disable/Enable all Messages** an- oder abgeschaltet werden
- Reload Mapping (im Register „UD“)

  Lädt die aktive Mapping-Datei erneut

4.3.3 Such- und Filterfunktion

Im Fenster „Workspace Elements“ kann außerdem nach Elementen gesucht oder ein Filter auf die Ansicht angewendet werden



Die Suchfunktion

Um eine Suche durchzuführen, geben Sie den Suchtext ein und klicken Sie **Go**. Im Register „Search Results“ werden dann diejenigen Elemente dargestellt, die den Suchkriterien entsprechen.

Wenn Sie die Option **Clear Search Results on Go** aktivieren, wird die Liste der Suchergebnisse im Register „Search Results“ bei jeder neuen Suche gelöscht, ansonsten werden die Ergebnisse der Liste hinzugefügt. Mit der Option **Use RegEx** können Sie für die Suchmuster reguläre Ausdrücke verwenden.

Die Filterfunktion

Mit der Filterfunktion kann die Liste nach verschiedenen Kriterien und Eigenschaften gefiltert werden:

- Label Type

  Filtert nach Typ der Labels
  - All Labels
  - All ASAM
  - All UD
- Hierarchy

  Stellt nur den gewählten Teil der Hierarchie dar
- Element Type

  Filtert nach Typ der Elemente
  - Inputs
  - Outputs
  - Parameters
  - Measure Elements

  - Pins
- Object Type

  Filtert nach Objekttyp
  - Scalar
  - Array
  - Matrix
  - 1D-Table
  - 2D-Table
- Data Type

  Filtert nach Datentyp
  - Logical
  - Discrete
  - Unsigned Discrete
  - Continuous
  - Enumeration

Ergebnisse in neuem Register speichern

Sie können die im Register „Search Results“ dargestellten Ergebnisse in einem neuen Register ablegen - rechtsklicken Sie dazu auf den Registertitel „Search Results“ und wählen Sie **Export Result to New Tab**. Zudem können Sie den Namen des neuen Registers festlegen.

4.4 Der Hauptarbeitsbereich

Im Hauptarbeitsbereich von ETAS EE sind die Hauptfunktionen auf vier Register verteilt:

- „Das Register „Instrumentation““ auf Seite 291

  Hier wird die Instrumentierung (Anzeige- und Bedienelemente) des Experiments erstellt und bedient.
- „Das Register „Datalogger““ auf Seite 293

  Hier wird der Datalogger konfiguriert und bedient.
- „Das Register „Signal Generator““ auf Seite 297

  In diesem Register können Signalgeneratoren erstellt, verwaltet und bedient werden.
- „Das Register „RT-Plugins““ auf Seite 315

  Hier werden RT-Plugins verwaltet.

4.4.1 Das Register „Instrumentation“

Das Register „Instrumentation“ besteht aus einem oder mehreren Layern, die wiederum Gruppen von Anzeige- und Bedienelementen enthalten.

Layer

Wenn Sie auf den Titel eines Layers rechtsklicken, wird ein Kontextmenü geöffnet, das die folgenden Funktionen enthält:

- **Create Layer**

  Erstellt einen neuen Layer
- **Import Layer**

  Erstellt einen Layer auf Basis eines zuvor exportierten Layers
- **Export Layer**

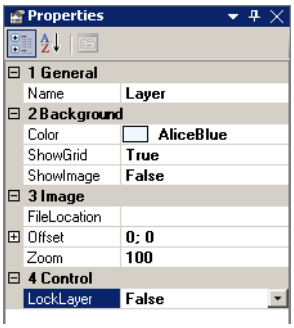
  Speichert eine Layerkonfiguration in einer Datei
- **Rename Layer**

  Benennt einen Layer um
- **Delete Layer**

  Löscht den ausgewählten Layer

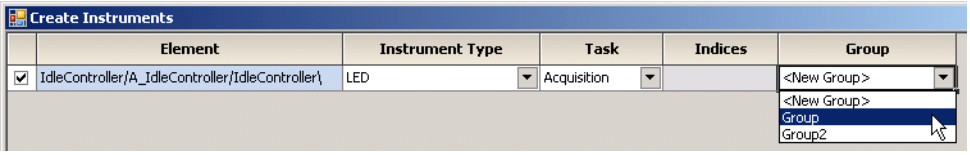
Layereigenschaften

Wenn Sie den Titel eines Layers wählen, werden dessen Eigenschaften im Fenster „Properties“ angezeigt und können dort auch bearbeitet werden.



Gruppen

Wenn Sie über die Funktion **Measure / Calibrate** des Kontextmenüs des Fensters „Workspace Elements“ Messgrößen anzeigen oder Parameter verstellen wollen, so können Sie im Dialogfenster „Create Instruments“ wählen, ob Sie für das Instrument eine neue Gruppe erstellen wollen oder ob Sie es zu einer bereits vorhandenen Gruppe hinzufügen wollen.



Wenn Sie den Gruppentitel wählen und rechtsklicken, wird ein Kontextmenü geöffnet, das folgende Funktionen enthält:

- **Bring to Front**

  Bringt die Gruppe in den Vordergrund
- **Send to Back**

  Bringt die Gruppe in den Hintergrund
- **Cut** / **Copy** / **Paste**

  Ermöglicht das Ausschneiden/Kopieren einer Gruppe eines Layers und das Einfügen in einen anderen Layer
- **Move to Layer**

  Hier kann ein anderer Layer gewählt werden, zu dem die Gruppe verschoben werden soll
- **Hide Group Frame** / **Show Group Frame**

  Entfernt den Rahmen der Gruppe oder fügt einen entfernten Rahmen wieder hinzu
- **Delete**

  Löscht die gewählte Gruppe

Gruppeneigenschaften

Wenn Sie den Titel einer Gruppe wählen, werden im Fenster „Properties“ deren Eigenschaften angezeigt und können dort auch bearbeitet werden.

Instrumente

Das Hinzufügen eines Instrumentes zu einer Gruppe/Layer erfolgt entweder per Drag & Drop aus dem Fenster „Instrumentation“ oder im Kontext des Fensters „Workspace Elements“.

Eigenschaften

Wenn Sie den Titel eines Instruments wählen, werden im Fenster „Properties“ dessen Eigenschaften angezeigt und können auch dort bearbeitet werden.

4.4.2 Das Register „Datalogger“

Das Register „Datalogger“ dient zum Erstellen und Konfigurieren von Dataloggern.

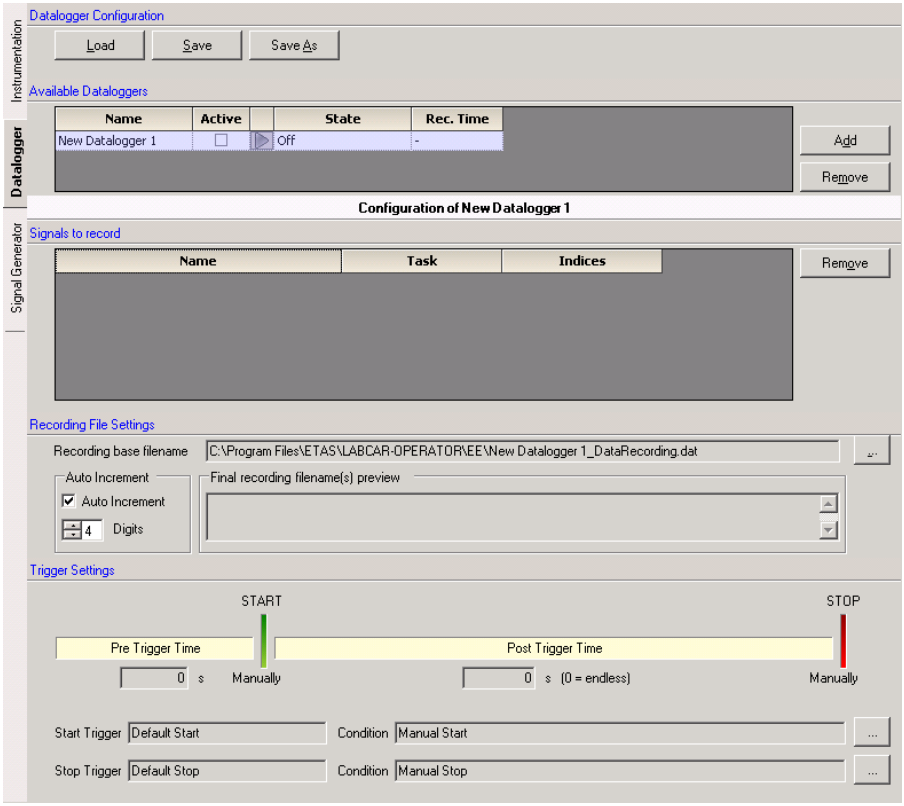


Abb. 4-4 Das Register „Datalogger“

Im Folgenden finden Sie eine Beschreibung der Bedienmöglichkeiten in diesem Register.

Das Feld „Datalogger Configuration“

Hier können Sie die gezeigte Gesamtkonfiguration der Datalogger speichern und zuvor gespeicherte Konfigurationen laden – vorzugsweise werden diese (mit der Erweiterung *.dlc4) im Ordner „Datalogger Files“ im Experiment Explorer verwaltet.

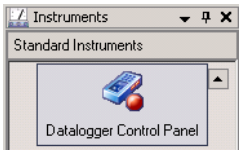
Das Feld „Available Dataloggers“

Hier werden alle verfügbaren Datalogger angezeigt – mit **Add** kann ein neuer erstellt werden, mit **Remove** kann ein Datalogger entfernt werden.

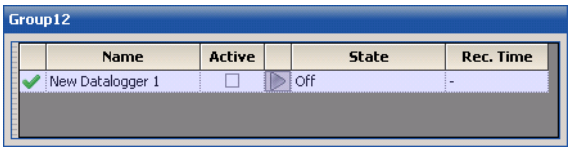
Hinweis

*Es können **maximal zwei** Datenlogger erstellt werden! Einer davon dient zur Aufzeichnung von Größen aus den Modulen des LABCAR-OPERATOR-Projektes – ein zweiter kann zur Fernsteuerung des INCA-Datenloggers verwendet werden (wenn LABCAR-CCI V5.4.1 (Calibration Connector for INCA) verwendet wird).*

Die tatsächliche Bedienung eines Dataloggers wird aber nicht hier, sondern von einem Layer des Registers „Instrumentation“ aus erfolgen. Erstellen Sie dort ein Instrument „Datalogger Control Panel“ (aus der Liste der „Standard Instruments“).

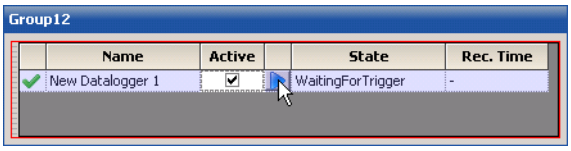


Die Anzeige und die Bedienmöglichkeiten in diesem Instrument sind dann die selben wie im Register „Datalogger“.



In der ersten Spalte wird durch das grüne Häkchen angezeigt, dass der Datalogger „valid“ ist – dieser Zustand wird durch das Hinzufügen eines Signals in der Liste „Signals to record“ (s.u.) herbeigeführt.

Wenn das Experiment läuft, muss der Datalogger im Feld „Active“ aktiviert werden. Er kann dann durch Klicken des blauen Dreiecks manuell gestartet werden oder es wird der Eintritt der Triggerbedingung abgewartet.

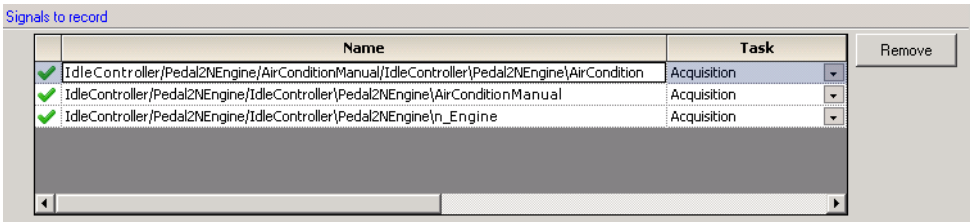


Bei laufenden Datalogger wird das blaue Dreieck zum Rechteck, mit dem der Datalogger auch wieder manuell gestoppt werden kann.

Angezeigt werden außerdem der momentane Zustand des Dataloggers („State“) und (bei laufendem Datalogger) die Aufnahmezeit („Rec. Time“).

Das Feld „Signals to record“

Diese Liste enthält die vom jeweiligen Datalogger aufzuzeichnenden Signale, die aus dem Fenster „Workspace Elements“ per Drag & Drop hinzugefügt werden können. Zudem kann hier jedem Signal eine Task für die Aufzeichnung zugeordnet werden.



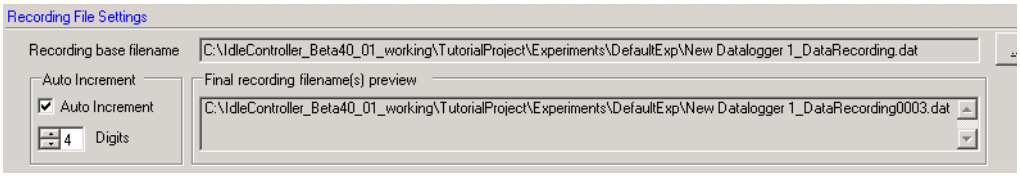
Entfernt werden Signale, indem das Signal gewählt wird und **Remove** geklickt wird.

Das Feld „Recording file settings“

In diesem Feld werden die Einstellungen für die Namen der Log-Dateien festgelegt. Dies geschieht durch

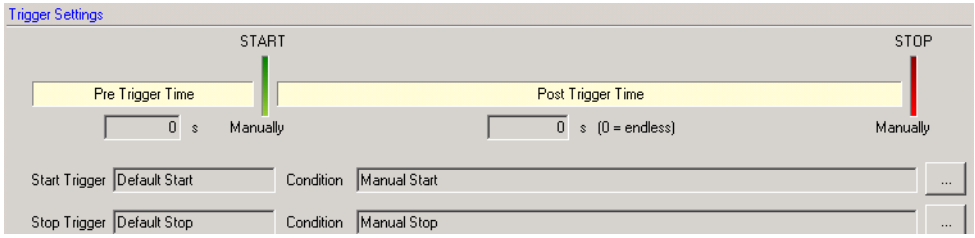
- Festlegen einer „Template“-Datei in einem bestimmten Pfad („Recording base filename“) und
- der Vereinbarung einer Konvention, dass eine an diesen Dateinamen angehängten Zahl hochgezählt wird („Auto Increment“).

Dies führt zu dem Dateinamen, der bei der nächsten Aufnahme verwendet wird („Final recording filename preview“).



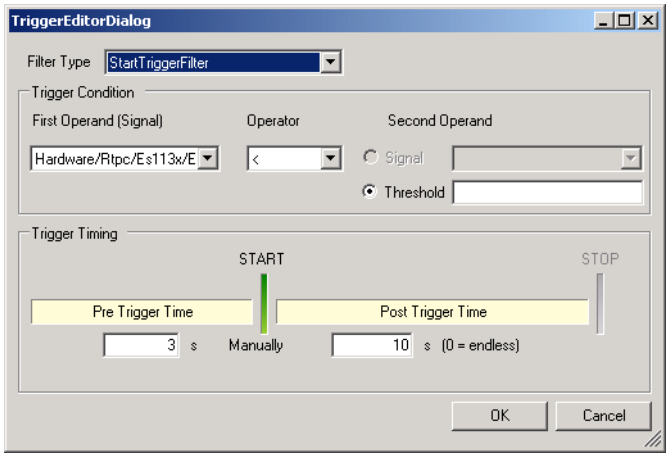
Das Feld „Trigger Settings“

In diesem Feld werden die Bedingungen (Trigger) für den automatischen Start der Datalogger festgelegt.



Um die für Start und Stopp des Dataloggers gewünschten Triggerbedingung festzulegen, klicken Sie auf die entsprechenden Schaltflächen **...** rechts unten.

Im folgenden Dialogfenster können Sie dann die Einstellungen für z.B. den Starttrigger vornehmen.



- **Filter Type**

  Manuell oder Triggerbedingung
- **Trigger Condition**
  - **First Operand**

    Das Signal, dessen Wert den Trigger auslöst
  - **Operator**

    Der Operator für den Vergleichswert

  - **Second Operand**

    Das Signal oder der Wert, mit dem das Triggersignal verglichen wird

- **Trigger Timing**
  - **Pre-Trigger Time**

    Die Zeit vor Eintritt der Triggerbedingung, in der das Signal zusätzlich aufgezeichnet wird
  - **Post-Trigger Time**

    Die Zeit, in der das Signal nach Eintritt der Triggerbedingung aufgezeichnet wird (0 bedeutet endlos, d.h. bis zum Eintreten eines Stopptriggers oder eines manuellen Stopps)

4.4.3 Das Register „Signal Generator“

In diesem Register können Signalgeneratoren erstellt, verwaltet und bedient werden.

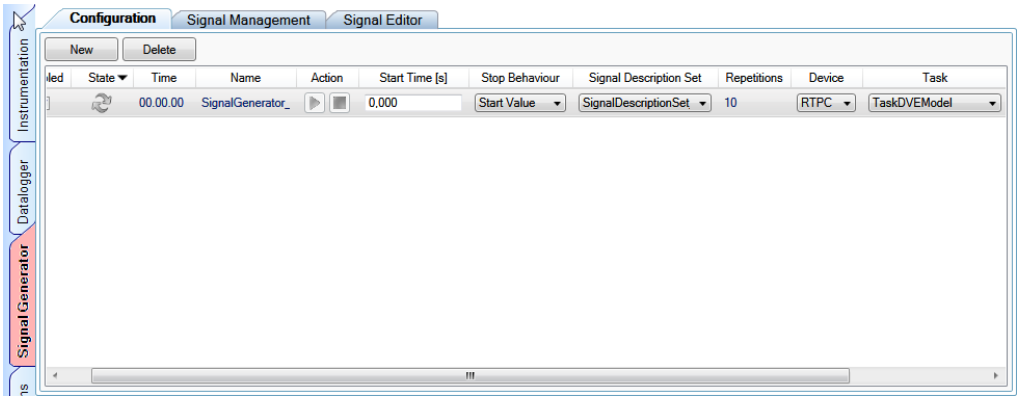


Abb. 4-5 Das Register „Signal Generator“

Die Bedienoberfläche für die Signalgeneratoren selbst besitzt wiederum drei Register:

- **Configuration**

  In diesem Register können Signalgeneratoren erzeugt, konfiguriert und gesteuert werden (siehe „Das Register „Configuration““ auf Seite 297).

- **Signal Management**

  Im Register „Signal Management“ werden die Signal Description Sets für die Signalgeneratoren erzeugt und konfiguriert (siehe „Das Register „Signal Management““ auf Seite 300).

- **Signal Editor**

  Hier können die Segmente der Signal Descriptions bearbeitet werden[1] (siehe „Das Register „Signal Editor““ auf Seite 310).

Das Register „Configuration“

In diesem Register können Signalgeneratoren erzeugt, konfiguriert und gesteuert werden.

Einen Signalgenerator erstellen

- Um einen Signalgenerator zu erstellen, klicken Sie **New**.

  Ein Signalgenerator wird zur Liste hinzugefügt.

  Es wird ein Signalgenerator mit dem Namen „SignalGenerator“ erzeugt (oder „SignalGenerator_1“, „SignalGenerator_2“ usw.)

1. Nur sichtbar, wenn die Editierfunktion für eine Signal Description gewählt wurde.

Einen Signalgenerator umbenennen

- Wählen Sie den Signalgenerator in der Liste und rechtsklicken Sie.
- Wählen Sie im Kontextemenü **Rename**

oder

- Drücken Sie <F2>.
- Der Name (in der Spalte „Name“) kann jetzt editiert werden.

Einen Signalgenerator entfernen

- Um einen Signalgenerator aus der Liste zu entfernen, klicken Sie **Delete**.

  Ein Signalgenerator wird entfernt.

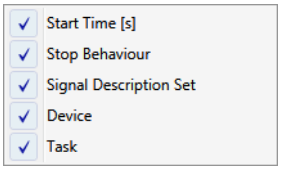
Eigenschaften und Steuerungsmöglichkeiten:

Jeder Signalgenerator in der Liste besitzt Eigenschaften und Steuerungsmöglichkeiten, die im Folgenden beschrieben werden.

Hinweis

Die Bedienung der Signalgeneratoren ist im Abschnitt „Signalgenerator bedienen“ auf Seite 314 beschrieben.

Rechtklicken auf die Spaltenüberschriften öffnet ein Kontextmenü, in dem einzelne Spalten ausgeblendet werden können.



Die Reihenfolge der Spalten kann mit Drag-and-Drop verändert werden.

- **Enabled**

  Aktiviert oder deaktiviert den jeweiligen Signalgenerator.

- **State**

  Hier wird der Zustand des Signalgenerators durch ein Symbol symbolisiert.

  – **Invalid**

    Das Symbol und die Schaltfläche **Start** (Spalte „Action“) sind ausgegraut.

    Der Grund ist eine ungültige Konfiguration des Signalgenerators (nicht zugewiesene Signale etc.)

  – **Stopped**

    Das Symbol wird angezeigt und ist statisch. Der jeweilige Signalgenerator kann über die Schaltfläche **Start** (Spalte „Action“) gestartet werden.

  - **Running**

    Das rotierende Symbol zeigt einen laufenden Signalgenerator.

  - **Paused**

    Das angehaltene und blinkende Symbol zeigt, dass der Signalgenerator pausiert – es wird weiterhin der letzte Wert des Signalgenerators ausgegeben.

- **Name**

  Der Name des jeweiligen Signalgenerators.

- **Action**

  In dieser Spalte können Sie die Ausführung des Signalgenerators steuern.

  - **Start**

    Startet den Signalgenerator – falls ein Starttrigger definiert wurde, wird bis zum Eintritt der Triggerbedingung gewartet.

  - **Stop**

    Stoppt den Signalgenerator.

  - **Pause**

    Pausiert den Signalgenerator. Das Modell wird jetzt mit dem letzten Wert stimuliert. Wenn mit **Play** wieder gestartet wird, wird der Signalgenerierung an dem Punkt wieder aufgenommen, an dem sie zuvor angehalten wurde.

- **Start Time**

  Der Zeitpunkt nach Start des Signalgenerators, zu dem die Signalausgabe beginnt (Default: 0.000).

- **Stop Behaviour**

  Definiert das Verhalten des jeweiligen Signals, wenn der Signalgenerator angehalten wurde.

  Folgende Einstellungen sind möglich:

  - **Start Value** (Default)

    Nach dem Anhalten wird der Startwert der Signal Description ausgegeben.

  - **0 (zero)**

  Nach dem Anhalten wird der Wert „0“ ausgegeben.

- **Signal Description Set**

  Hier wird das Signal Description Set ausgewählt, das die Beschreibung der Signale enthält, die der Signalgenerator ausgeben soll.

  Zur Verfügung stehen alle Signal Description Sets, die im Register „Signal Management“ vorhanden sind.

- **Repetitions**

  Die Anzahl der Durchläufe für das jeweilige Signal Description Set.

- **Device**

  Das Simulationstarget (RTPC)

- **Task**

  Die Task, in deren Zeitraster die Berechnung des jeweiligen Signalgenerators erfolgen soll.

Das Register „Signal Management“

Im Register „Signal Management“ werden die Signal Description Sets für die Signalgeneratoren erzeugt und konfiguriert.

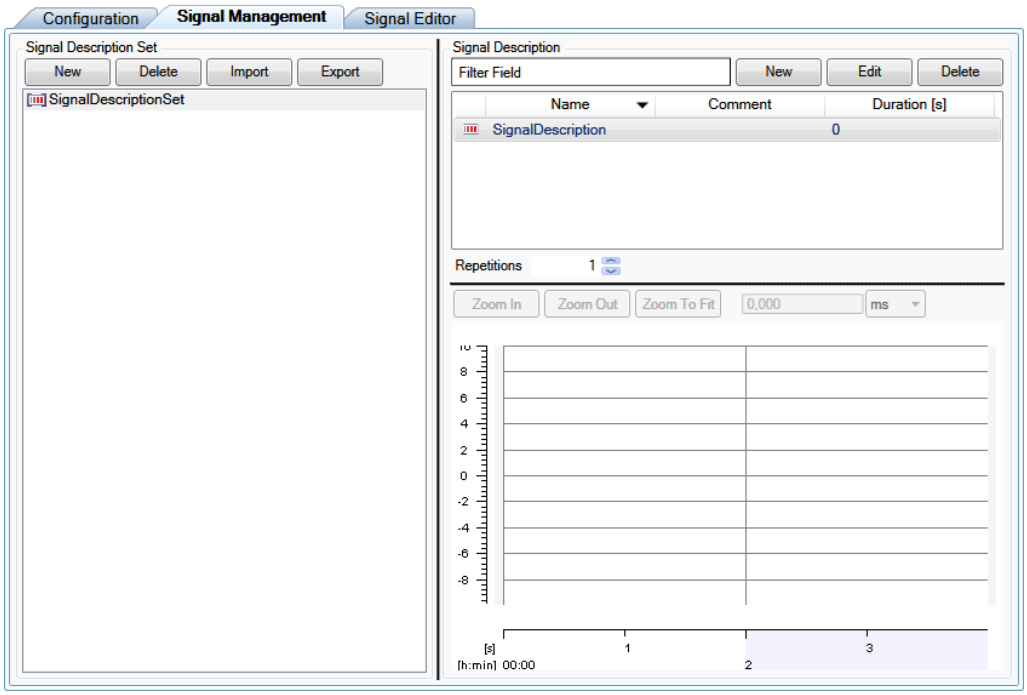


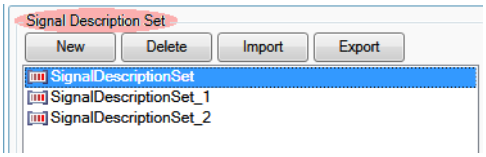
Abb. 4-6 Das Register „Signal Management“

Es besteht auch zwei Bereichen:

- Der Bereich „Signal Description Set“, in dem die Signal Description Sets des Experiments verwaltet werden (siehe „Der Bereich „Signal Description Set“:“ auf Seite 301).
- Der Bereich „Signal Description“, in dem die Signal Descriptions des Sets verwaltet werden (siehe „Der Bereich „Signal Description“:“ auf Seite 307).

Der Bereich „Signal Description Set“:

In diesem Bereich werden die Signal Description Sets des aktuellen Experiments erstellt und verwaltet.

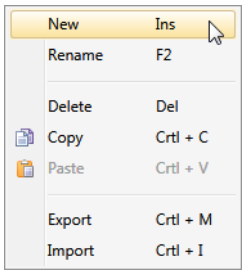


Ein Signal Description Set erstellen

- Um ein Signal Description Set zu erstellen, klicken Sie **New**

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **New** aus dem Kontextmenü.



oder

- Drücken Sie <EINF>.

  Das Signal Description Set wird erzeugt.

Ein Signal Description Set kopieren

- Um ein Signal Description Set zu kopieren, wählen Sie es in der Liste.
- Rechtsklicken Sie und wählen Sie **Copy** aus dem Kontextmenü.

oder

- Drücken Sie <STRG+C>.

  Das Signal Description Set wird kopiert und kann mit **Paste** (im Kontextmenü) oder <STRG-V> in die Liste der Signal Description Sets eingefügt werden.

Ein Signal Description Set umbennen

- Um ein Signal Description Set umzubenennen, wählen Sie es in der Liste.
- Rechtsklicken Sie und wählen Sie **Rename** aus dem Kontextmenü.

oder

- Drücken Sie <F2>.

  Der Name des Signal Description Sets kann jetzt geändert werden.

Ein Signal Description Set entfernen

- Um ein Signal Description Set zu entfernen, klicken Sie **Delete**.

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **Delete** aus dem Kontextmenü.

oder

- Drücken Sie <ENTF>.

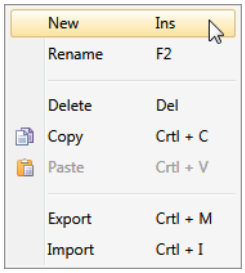
  Das Signal Description Set wird entfernt.

Ein Signal Description Set importieren

- Um ein Signal Description Set zu importieren, klicken Sie **Import**.

oder

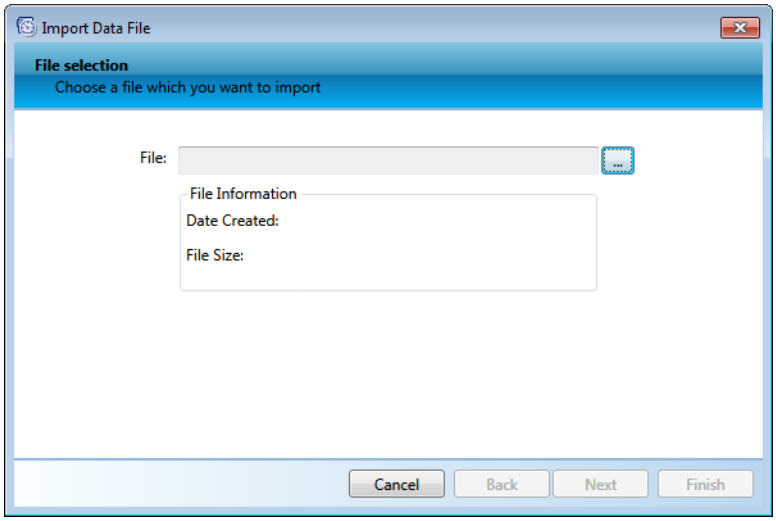
- Rechtsklicken Sie in den Bereich und wählen Sie **Import** aus dem Kontextmenü.



oder

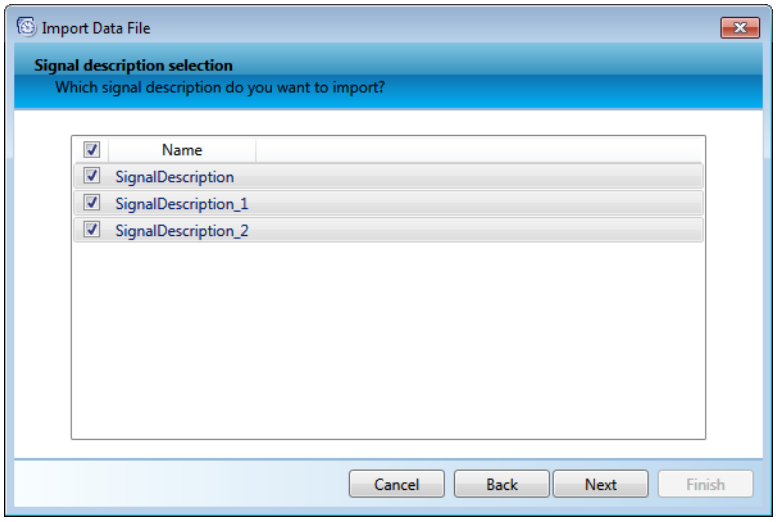
- Drücken Sie <STRG+I>.

  Das Fenster „Import Data File“ wird geöffnet.



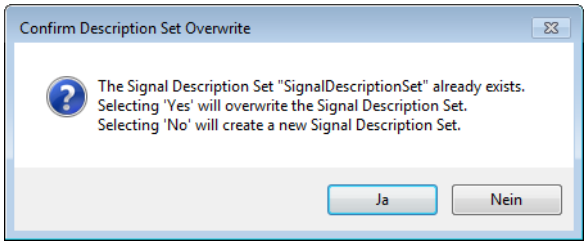
- Klicken Sie **...**, um ein Dateiauswahlfenster zu öffnen.
- Wählen Sie eine Datei folgenden Formates und klicken Sie **Next**.
  - ETAS Stimulus File (*.esti)
  - HIL API Stimulus File (*.sti)
  - Measurement Data File (*.dat)
  - LABCAR Stimulus File (*.lcs)
  - MAT-File (*.mat)

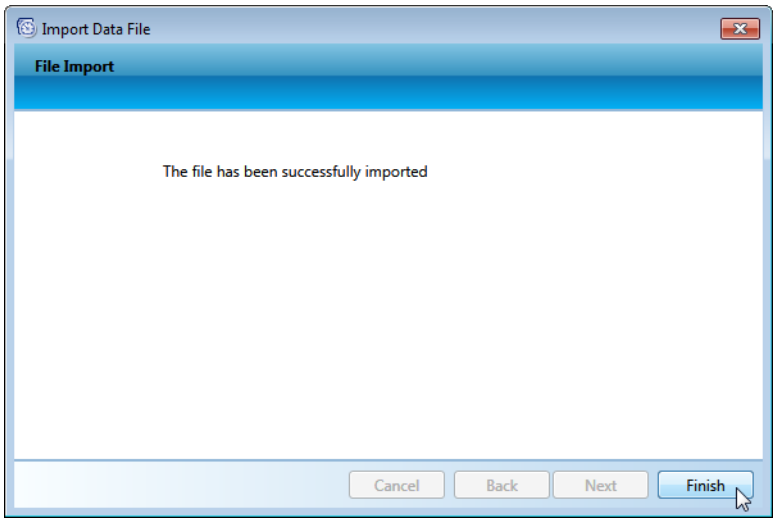
  Das Fenster „Import Data File“ wird geöffnet.



Wählen Sie die gewünschten Signal Descriptions.

Wenn ein Signal Description Set mit dem Namen „Signal Description Set“ schon vorhanden ist, können Sie wählen, ob dieses überschrieben werden soll oder ob ein Set mit neuem Namen „Signal Description Set_*n*“ erstellt werden soll.





- Klicken Sie **Finish**.

  Das Signal Generator Set wird erstellt (oder überschrieben).

MATLAB-Datei (*.mat) importieren

Beim Import von MATLAB *.mat-Dateien werden zwei Formate unterstützt:

- **Matrix-basiert**

  Verwendet für Signale mit unterschiedlichen Zeitskalen. Jedes Signal ist eine Matrix – eine MATLAB-Datei kann eine beliebige Anzahl von n x 2-Matrizen enthalten (n kann bei jeder Matrix verschieden sein).

  - Um ein Signal Generator Set in MATLAB zu erzeugen, geben Sie im MATLAB-Bedienfenster z.B. folgenden Code ein:

    ```
    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;
    ```

    Dieses Beispiel erzeugt die MATLAB-Datei `MatrixBasedFormat.mat` mit den beiden Signalen (Matrizen) X und 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

- **Intervall-basiert (Reihenformat)**

  Verwendet für Signale mit gemeinsamer Zeitskala.

  Intervallvektoren enthalten die Dauer eines einzelnen Intervalls, Datenvektoren enthalten die Signalwerte für diese Intervalle. Dabei müssen Daten- und Intervallvektoren die selbe Länge besitzen - der Name des Intervallvektors muss „Intervals“ lauten.

- Geben Sie im MATLAB-Bedienfenster z.B. folgenden Code ein:

```
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;
```

Dieses Beispiel erzeugt die MATLAB-Datei `IntervalsBasedFormat.mat` mit den beiden Signalvektoren „Signal1_Value“ und „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

Ein Signal Description Set exportieren

- Um ein Signal Description Set zu exportieren, wählen Sie dieses in der Liste der Signal Description Sets.
- Klicken Sie **Export**

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **Export** aus dem Kontextmenü.

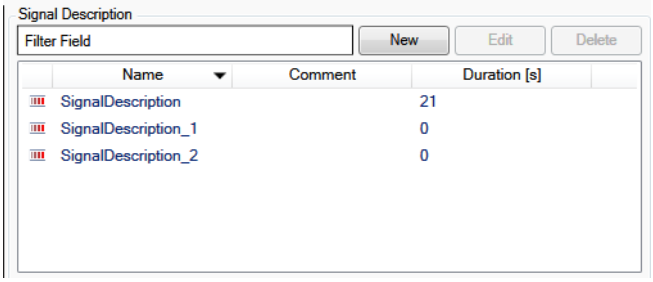
oder

- Drücken Sie <STRG+M>.
- Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie ein Dateiformat (ETAS Stimulus File (*.esti) oder HIL API Stimulus File (*.sti)), wählen Sie ein Verzeichnis und vergeben Sie einen Dateinamen.
- Klicken Sie **Speichern**.

  Das Signal Description Set wird in dem gewählten Format gespeichert.

Der Bereich „Signal Description“:

In diesem Bereich sind die Signal Descriptions aufgeführt, die Bestandteil eines Signal Description Sets sind.



- **Name**

  Name der Signal Description (siehe „Name“ auf Seite 310

- **Comment**

  Kommentar (siehe „Comment“ auf Seite 310

- **Duration**

  Gesamtdauer der Signal Description (errechnet aus der Länge der einzelnen Sequenzen).

Das **Filter Field** ermöglicht die Suche nach Signal Descriptions durch ein Filter. Ist dieses Feld leer, werden alle vorhandenen Signal Descriptions aufgeführt.

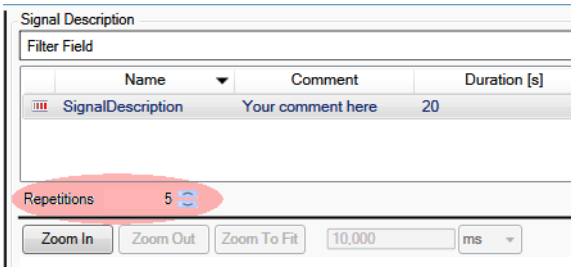
Jede(s) Zeichen(folge), das (die) in dieses Feld eingegeben wird, wird für eine inkrementelle Suche in den Namen der vorhandenen Signal Descriptions verwendet. Wird z.B. „abc“ eingegeben, so werden alle Signal Descriptions aufgeführt, die mit „abc“ beginnen.

Zur Vereinfachung der Suche können auch reguläre Ausdrücke mit Wildcards verwendet werden:

- Der Asterisk („*“) findet jede beliebige Zeichenfolge (auch mit 0 Zeichen). „*_TRK“ z.B. findet alle Namen, die mit „_TRK“ enden.
- Das Fragezeichen („?“) findet genau ein beliebiges Zeichen.
- „?TRK“ z.B. findet alle Namen, die mit einen beliebigen Buchstaben gefolgt von „TRK“ beginnen.

Diese Erkennungsmuster unterscheiden nicht zwischen Groß- und Kleinschreibung.

Mit **Repetitions** legen Sie fest, wie oft die Sequenz durchlaufen wird.

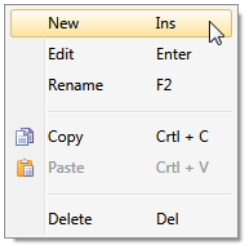


Eine Signal Description erstellen

- Um eine Signal Description zu erstellen, klicken Sie **New**.

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **New** aus dem Kontextmenü.



oder

- Drücken Sie <EINF>.

  Die Signal Description wird erzeugt.

Eine Signal Description bearbeiten

- Um eine Signal Description zu bearbeiten, klicken Sie **Edit**

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **Edit** aus dem Kontextmenü.

oder

- Drücken Sie <EINGABE>.

  Die Ansicht wechselt in das Register „Signal Editor“ (siehe „Das Register „Signal Editor“ “ auf Seite 310), in dem die Signale der Signal Description bearbeitet werden können.

Eine Signal Description umbenennen

- Um eine Signal Description umzubenennen, wählen Sie diese in der Liste.
- Rechtsklicken Sie und wählen Sie **Rename** aus dem Kontextmenü.

oder

- Drücken Sie <F2>.

  Der Name der Signal Description kann jetzt geändert werden.

Eine Signal Description kopieren

- Um eine Signal Description Set zu kopieren, wählen Sie diese in der Liste.
- Rechtsklicken Sie und wählen Sie **Copy** aus dem Kontextmenü.

oder

- Drücken Sie <STRG-C>.

  Die Signal Description wird kopiert und kann mit **Paste** (im Kontextmenü) oder <STRG-V> in die Liste der Signal Descriptions eingefügt werden.

Eine Signal Description entfernen

- Um eine Signal Description zu entfernen, klicken Sie **Delete**

oder

- Rechtsklicken Sie in den Bereich und wählen Sie **Delete** aus dem Kontextmenü.

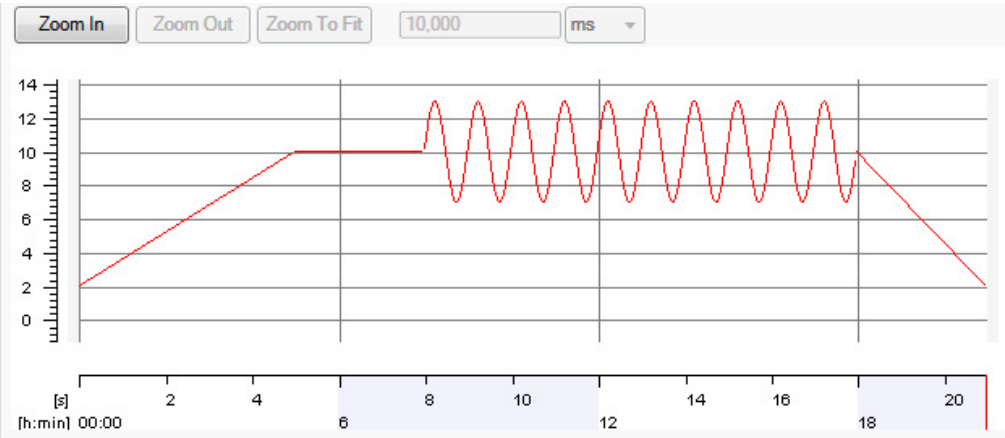
oder

- Drücken Sie <ENTF>.

  Die Signal Description wird entfernt.

Die Signalansicht :

Die Signalansicht vermittelt einen Überblick über die in diesem Register definierten Signalsegmente.



Die Achsen können durch Ziehen mit gedrückter Maustaste verschoben werden und mit gedrückter <STRG>-Taste auch vergrößert bzw. verkleinert werden

Angezeigt wird die jeweils in der Liste gewählte Signal Description. Mit der <STRG>-Taste ist auch eine Mehrfachauswahl möglich – die Signalverläufe werden dann mit unterschiedlichen Farben dargestellt.

Mit **Zoom In**/**Zoom Out** kann die Achsenskalierung ebenfalls verändert werden – **Zoom To Fit** passt die Darstellung an den Signalverlauf an.

Das Register „Signal Editor“

In das Register „Signal Editor“ gelangen Sie, indem Sie in der Liste der Signal Descriptions (im Register „Signal Management“) eine Signal Description auswählen und entweder **Edit** klicken oder diese doppelklicken.

Dann wird das Register „Signal Editor“ geöffnet, in dem der Signalverlauf erstellt und bearbeitet werden kann.

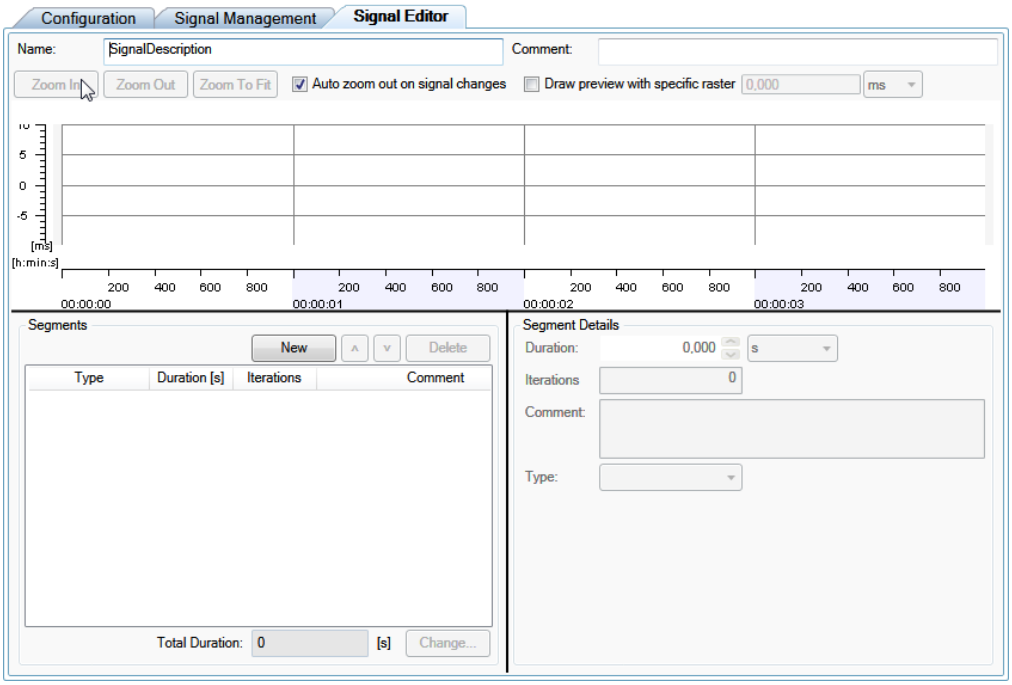


Abb. 4-7 Das Register „Signal Editor“

- **Name**

  Hier kann für die Signal Description ein Name vergeben werden.

- **Comment**

  Hier kann ein Kommentar zur Signal Description vergeben werden.

Die Signalansicht :

Die Signalansicht vermittelt einen Überblick über die in diesem Register bearbeitbaren Signalsegmente.

Angezeigt wird die jeweils in der Liste gewählte Signal Description. Mit der <STRG>-Taste ist auch eine Mehrfachauswahl möglich – die Signalverläufe werden dann mit unterschiedlichen Farben dargestellt.

Mit **Zoom In**/**Zoom Out** kann die Achsenskalierung verändert werden – **Zoom To Fit** passt die Darstellung an den Signalverlauf an. Die Achsen können auch durch Ziehen mit gedrückter Maustaste verschoben werden und mit gedrückter <STRG>-Taste auch vergrößert bzw. verkleinert werden.

- **Auto Zoom out on Signal Changes**

  Bewirkt, dass bei Änderung von Segmenten das Koordinatensystem automatisch angepasst wird („Zoom To Fit“).

- **Draw Preview with Specific Raster**

  Damit kann eine bestimmte Rasterung der Zeitachse vorgeben werden.

Der Bereich „Segments“ :

In diesem Bereich werden die Eigenschaften und die Reihenfolge der einzelnen Signalsegmente dargestellt:

- **Type**

  Hier werden die verschiedenen Segmenttypen verwaltet. Wird ein Segment gewählt, kann dieses im Bereich „Segment Types“ bearbeitet werden.

- **Duration**

  Die Dauer des Segment (wird im Bereich „Segment Types“ festgelegt).

- **Iterations**

  Die Anzahl der Durchläufe des Segments (wird im Bereich „Segment Types“ festgelegt).

- **Comment**

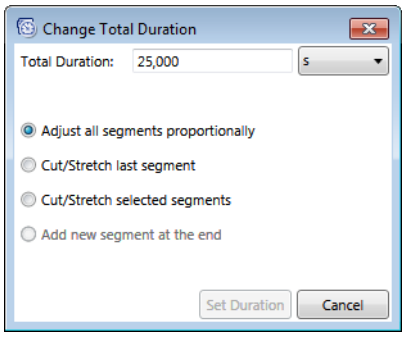
  Ein Kommentar (wird im Bereich „Segment Types“ festgelegt).

Mit **New** kann eine neues Segment erstellt werden, mit **Delete** ein zuvor ausgewähltes entfernt werden.

Mit den Pfeiltasten können ein oder mehrere ausgewählte Segmente nach oben bzw. unten verschoben werden.

Im Feld „Total Duration“ wird die Gesamtdauer (Summe aller Segmente) des Signals dargestellt, die mit **Change** geändert werden kann.

Dabei gibt es vier verschiedene Optionen:



- **Adjust all signals proportionally**

  Alle Segmente werden proportional geändert.

- **Cut/Stretch last segment**

  Nur das letzte Segment des Signal wird entsprechend geändert (d.h. verlängert oder verkürzt).

- **Cut/Stretch selected segments**

  Nur ausgewählte Segmente werden verlängert oder verkürzt.

- **Add new segment at the end**

  Ist die gewählte Zeitdauer länger als die aktuelle, wird am Ende ein neues Segment hinzugefügt.

Die Bearbeitung der Segment erfolgt im Bereich „Segment Details“.

Der Bereich „Segment Details“ :

Hier können die Eigenschaften eines (im Bereich „Segments“ gewählten) Segments bearbeitet werden.

- **Duration**

  Hier wird die Länge des Segments festgelegt.
- **Iteration**

  Hier wird die Anzahl der Durchläufe des Segments definiert.
- **Comment**

  Hier kann ein Kommentar für das Signalsegment vergeben werden.
- **Type**

  Der Segmenttyp:
  - **Constant**

    Signal hat einen konstanten Wert („Value“)
  - **Data**

    Das Segment wird aus einer Datei eingelesen (Measurement Data File (*.dat) oder MAT-File (*.mat))
  - **Pulse**

    Das Segment besteht aus einem PWM-Signal, definiert über Periodendauer („Period“), Amplitude („Amplitude“), Tastverhältnis („Duty Cycle“) und Offset („Offset“).
  - **Ramp**

    Ansteigende oder abfallende Rampe, definiert durch Startwert („Start Value“), Dauer („Duration“) und Endwert („Stop Value“) oder Startwert, Dauer und Steigung („Slope“).
  - **Random**

    Das Segment besteht aus Zufallswerten mit definierter Amplitude („Amplitude“) und Offset („Offset“).
  - **Saw**

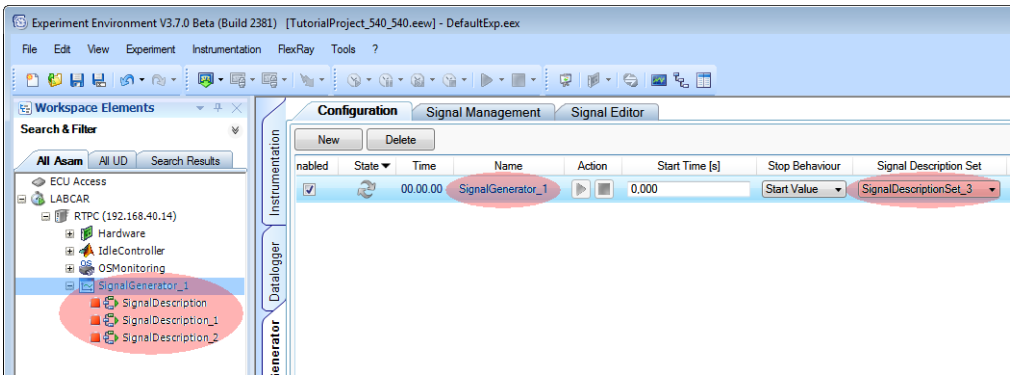
    Sägezahn, definiert durch Amplitude („Amplitude“), Periodendauer („Period“) und Offset („Offset“).
  - **Sine**

    Sinussignal, definiert durch Amplitude („Amplitude“), Periodendauer („Period“), Phasenverschiebung („Phase“) und Offset („Offset“).

Mit dem Signalgenerator arbeiten

Den Signalgeneratoren werden bei der Konfiguration Signal Description Sets zugewiesen, die wiederum Signal Descriptions enthalten.

Diese Signal Descriptions werden in den Workspace Elements als Modulausgänge (unter dem jeweiligen Signalgenerator angezeigt und können in einer Signal List mit anderen Eingängen verbunden werden.

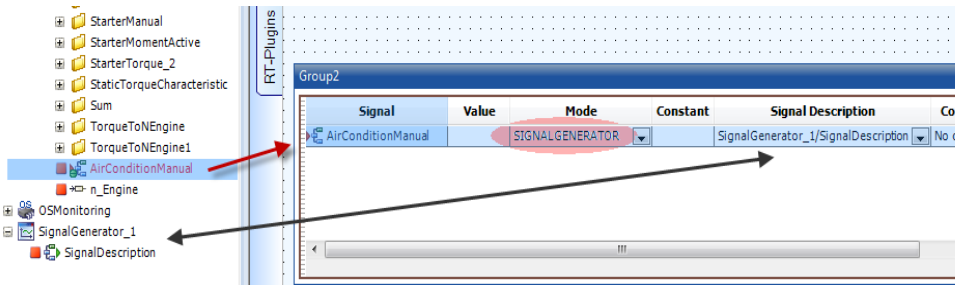


Hinweis

Damit der Signalgenerator und seine Signal Descriptions in der Element List dargestellt wird, muss er vollständig konfiguriert sein, d.h. der Signalgenerator muss einerseits ein Signal Description Set zugeordnet sein, dem wiederum Signal Descriptions zugeordnet wurden und andererseits muss eine Task gewählt worden sein.

Signalgenerator verbinden

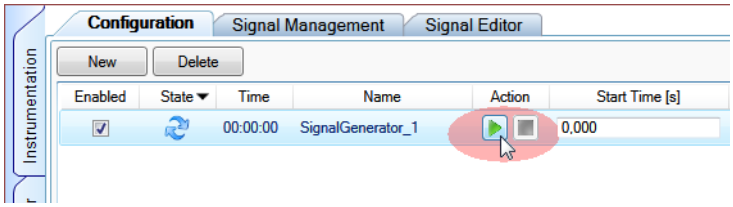
- Erstellen Sie in einem Layer (im Register „Instrumentation“) eine Signal List.
- Wählen Sie in den Workspace Elements einen Moduleingang, den Sie mit einem Signalgenerator stimulieren wollen und ziehen Sie diesen in die Signal List
- Wählen Sie in der Spalte „Mode“ die Option „Signal Generator“.
- Wählen Sie in der Spalte den Signalgenerator und die Signal Description, mit deren Signal der Moduleingang stimuliert werden soll.



- Speichern Sie das Experiment.

Signalgenerator bedienen

- Laden Sie das Experiment zum Real-Time PC herunter.
- Starten Sie das Experiment.
- Wechseln Sie in das Register „Signal Generator“.
- 
Wählen Sie den gewünschten Signalgenerator und klicken Sie das Symbol **Start**.




Der laufende Signalgenerator wird durch das Symbol **Running** angezeigt.

In der Spalte „Zeit“ wird die seit dem letzten Start des Signalgenerators vergangene Zeit angezeigt.

- 
Mit **Pause** kann der Signalgenerator pausiert werden – es wird weiterhin der letzte Wert des Signalgenerators ausgegeben.
- 
Mit **Stop and reset signal generator** wird der Signalgenerator angehalten und auf den in der Spalte „Stop Behaviour“ gewählten Wert zurückgesetzt.

4.4.4 Das Register „RT-Plugins“

Hier werden RT-Plugins verwaltet. Eine Beschreibung dieses Registers finden Sie im Abschnitt „Das Register „RT-Plugins““ auf Seite 234.

4.4.5 Das Fenster „Instruments“

Im Fenster „Instruments“ finden Sie eine Vielzahl von Instrumenten für die Interaktion mit dem Experiment.

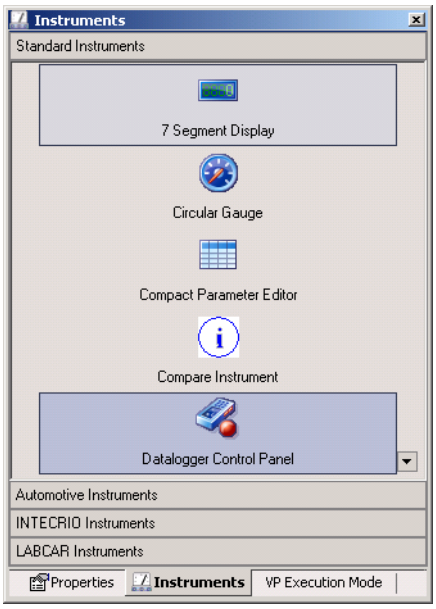


Abb. 4-8 Das Fenster „Instruments“

Die verfügbaren Instrumente sind in mehrere Gruppen unterteilt:

- Standard Instruments
- Automotive Instruments
- LABCAR Instruments

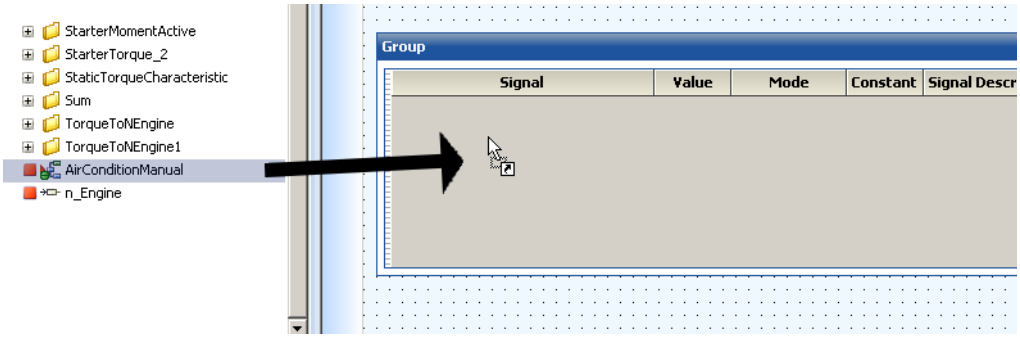
Alle diese Instrumente können per Drag & Drop in einen Layer des Registers „Instrumentation“ gezogen werden – die Zuweisung von Messgrößen und Parametern erfolgt in der Regel ebenfalls über die Funktion **Measure/Calibrate** des Kontextmenüs oder per Drag & Drop aus dem Fenster „Workspace Elements“.

4.4.6 Signale zu Signal List hinzufügen

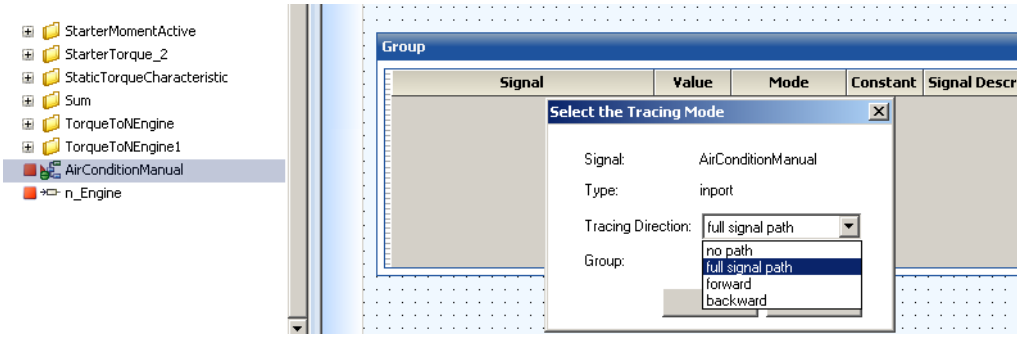
Nach dem Erstellen eines Instruments des Typs „Signal List“ gibt es zwei verschiedene Möglichkeiten, Signale zu dieser Liste hinzuzufügen.

Signal mit Signalverfolgung („Tracing“) hinzufügen

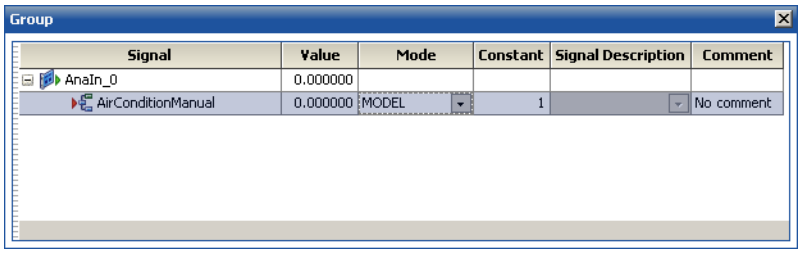
- Ziehen Sie das gewünschte Signal in die Signal List.



- Im Dialogfenster „Select the Tracing Mode“ wählen Sie „full signal path“, „forward“ oder „backward“.



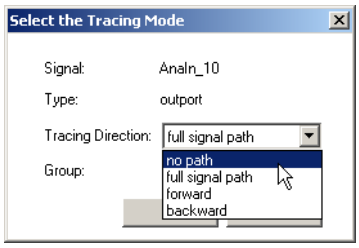
Das gewählte Signal wird in der Signal List mit allen Signalen angezeigt, mit denen es (in der gewählten Richtung) verbunden ist.



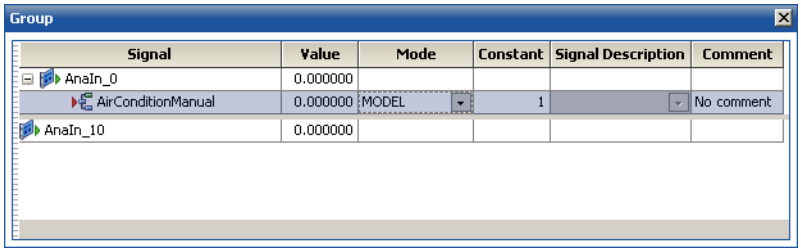
In dieser Darstellung kann ein Eingang auch vom Modell getrennt werden und mit einem konstanten Wert versehen werden (Spalte „Mode“).

Signal ohne Signalverfolgung hinzufügen

- Wenn Sie ein Signal ohne seine weiteren Verbindungen in der Signal Liste darstellen wollen, wählen Sie im Dialog „Select the Tracing Mode“ die Option „no path“.



Das Signal wird in der gewählten Signal List in einer Art Unterliste dargestellt.



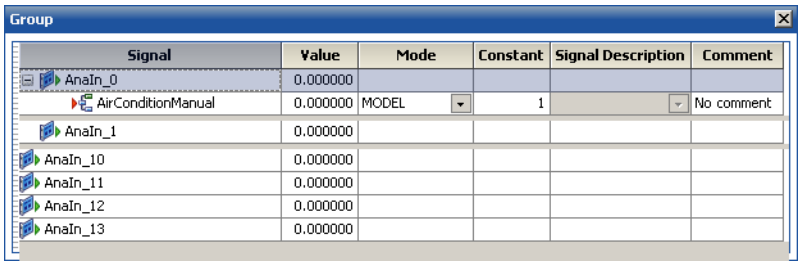
Weitere Signale zu vorhandener Signal List hinzufügen

- Ziehen Sie ein weiteres Signal in den Teil der Liste, die die Signale mit Verfolgung enthält.

  Es wird der Dialog „Select the Tracing Mode“ geöffnet in dem Sie die Art der Signalverfolgung festlegen können.

- Ziehen Sie ein weiteres Signal in den Teil der Liste, die die Signale ohne Verfolgung enthält.

  Das Signal wird zu diesem Teil der Liste hinzugefügt.



Für die Signale in der Liste gilt außerdem:

- Werden mehrere (in den Workspace Elements per Mehrfachauswahl gewählte) Signale in die Liste verschoben, werden diese immer zu dem Teil ohne Signalverfolgung hinzugefügt.
- Signale aus einem Teil der List können in den jeweils anderen Teil (oder in eine neue Liste) kopiert werden, indem die Eigenschaften im Dialog zu „Select the Tracing Mode“ entsprechend gewählt werden.

- Signale aus dem Teil der Liste ohne Signalverfolgung können per Drag & Drop beliebig in andere Signal Lists verschoben werden - dabei kann der „Tracing Mode“ in der Zielliste festgelegt werden.

4.5 Der Skript-Rekorder

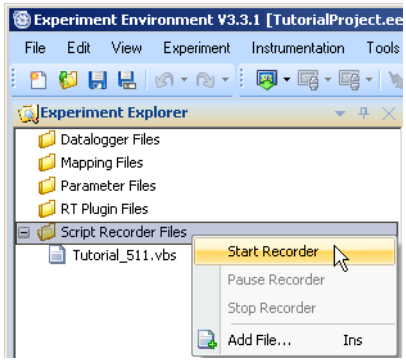
Bei der Durchführung von Experimenten werden häufig Vorgänge wie die Bedienung von GUIs (z.B. Öffnen von GUIs und Änderung von Parametern, Laden von Parameterdateien etc.) wiederholt durchgeführt. LABCAR-OPERATOR V5.4.1 stellt deshalb einen Rekorder zur Verfügung, mit dem solche wiederholten Vorgänge automatisiert werden können.

Diese Makros werden in Form von Visual Basic Scripts (`*.vbs`) gespeichert. Jedem Projekt können beliebig viele Skripte zugeordnet werden - zudem kann ein Skript in anderen Projekten verwendet werden.

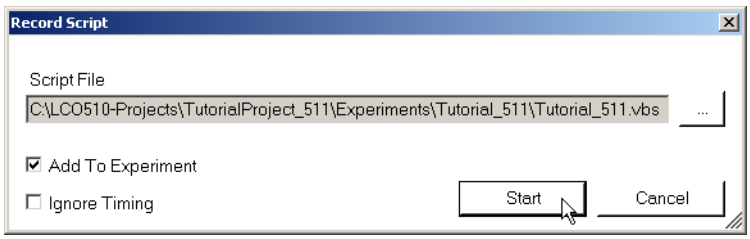
Mit dem Skript-Rekorder von LABCAR-OPERATOR V5.4.1 können alle Benutzeraktionen aufgezeichnet werden, die von der Scripting API von ETAS EE unterstützt werden.

Ein Skript aufzeichnen

- Rechtsklicken Sie im Experiment Explorer den Ordner „Script Recorder Files“.
- Wählen Sie **Start Recorder** im Kontextmenü.



- Wählen Sie - unabhängig davon, ob dem Projekt bereits eine Skriptdatei zugeordnet wurde oder nicht - im folgenden Dialog eine Datei.



  – Wählen Sie die Option „Add to Experiment“, wenn die Datei zum Experiment (im Verzeichnis „Script Recorder Files“) hinzugefügt werden soll.

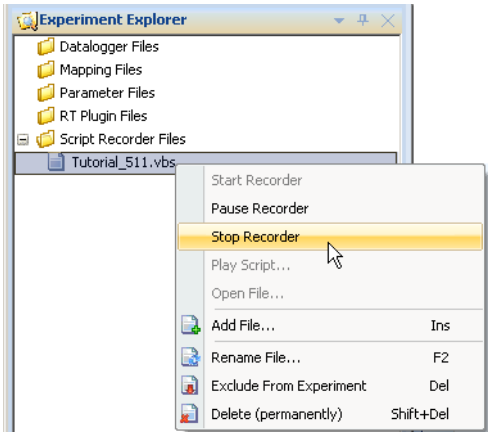
– Die Option „Ignore Timing“ sorgt dafür, dass die Zeiten, die zwischen den einzelnen Aktionen des Anwenders liegen, nicht mitaufgezeichnet werden.

- Klicken Sie **Start**.

Recording 00:00:29

Die Aufzeichnung wird gestartet. Der Zustand des Script-Rekorders und die bisherige Dauer der Aufzeichnung wird am unteren Rand des Hauptfensters der Experimentierumgebung dargestellt.

- Zum Anhalten der Aufzeichnung wählen Sie im Kontextmenü **Stop Recorder**.

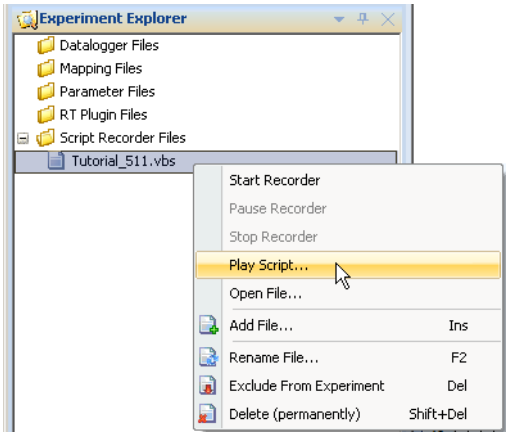


Script recorder idle

Die Aufzeichnung wird angehalten und das Skript gespeichert.

Ein Skript wiedergeben

- Wählen Sie im Experiment Explorer das wiederzugebende Skript.
- Wählen Sie im Kontextmenü „Play Skript“.



Playback 00:00:30

Das Skript wird wiedergegeben.

Skripte verwalten

- Wählen Sie **Add File**, um ein Skript zum Projekt hinzuzufügen.
- Wählen Sie **Rename File**, um ein Skript umzubenennen.
- Wählen Sie **Exclude From Experiment**, um die Zuordnung der Datei zu Experiment aufzuheben.
- Wählen Sie **Delete (permanently)**, um die Datei zu löschen.

4.6 Mit Parametern arbeiten

In diesem Abschnitt finden Sie Informationen zum Umgang mit den Parametern Ihres LABCAR-OPERATOR-Projektes in der Experimentierumgebung ETAS EE.

Im Einzelnen sind dies:

4.6.1 Parameter in ETAS EE

Jedes Modul, das in LABCAR-IP zum Gesamtprojekt hinzugefügt wird, bringt einen neuen Satz Parameter mit. Diese Parameter und deren zugewiesene Werte (die sogenannte Parametrierung) können aktiviert, verändert, gespeichert oder exportiert werden.

Parameter und Messwerte eines Projekts sind in ETAS EE im Fenster „Workspace Elements“ versammelt – eine ausführliche Beschreibung dieses Fensters finden Sie im Abschnitt „Das Fenster „Workspace Elements““ auf Seite 284.

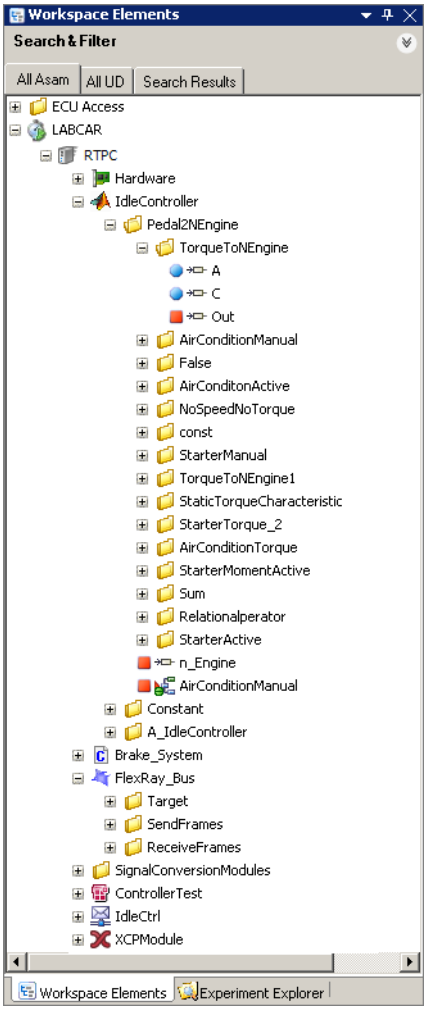


Abb. 4-9 Das Fenster „Workspace Elements“

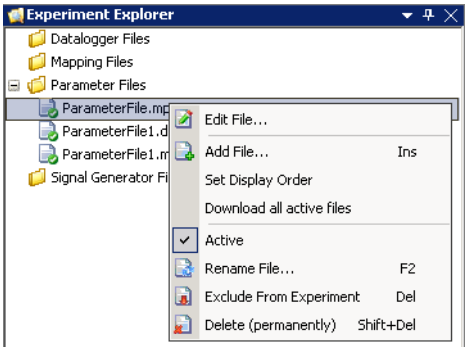
Die genannten Parameter befinden sich im Register „All Asam“ (die beiden anderen Register sind im Abschnitt „Die Register des Fensters „Workspace Elements““ auf Seite 285 beschrieben).

In diesem Register sind alle Module und deren Parameter und Messwerte im Ordner „LABCAR“ gruppiert – der Ordner „ECU Access“ enthält die Definitionen der ECU-Pins sowie ggf. Parameter und Messgrößen eines parallelen INCA-Experiments (mit Calibration Connector for INCA).

Die einzelnen Module sind mit speziellen Symbolen versehen, die ihren Typ kennzeichnen. Über die entsprechenden Kontextmenüs können damit Operationen auf dem Modulordner oder dessen Inhalt ausgeführt werden (siehe „Arbeiten mit den Workspace Elements“ auf Seite 286).

Verwaltung von Parameterdateien

Die Verwaltung von Dateien, die Parameter und zugewiesene Werte enthalten, erfolgt im Fenster „Experiment Explorer“ – das Kontextmenü enthält die entsprechenden Funktionen dafür.



Folgenden Typen von Dateien sind hier zu finden:

- *.mpa und *.dcm

  Dateien, die Parameter und Wertzuweisungen enthalten

- *.olc

  Datei mit einer bestimmten Konfiguration von Signalpfaden und Kennlinien

- *.pac

  Parameter Variant (oder Parameter Combination): Zusammenstellung mehrerer *.mpa-, *.dcm- und *.olc-Dateien

Einzelheiten dazu finden Sie im Abschnitt „Verwaltung von Parameterdateien im Experiment Explorer“ auf Seite 329.

4.6.2 Das Kontextmenü

Das Kontextmenü im Register „All Asam“ wurde bereits in „Arbeiten mit den Workspace Elements“ auf Seite 286 beschrieben – im Folgenden liegt der Fokus auf das Arbeiten mit Parametern.

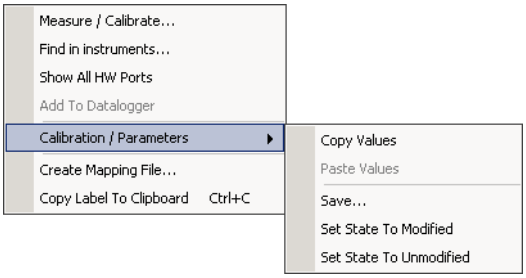


Abb. 4-10 Kontextmenü für Parameter (bei laufendem Experiment)

Das Kontextmenü stellt eine Reihe von Funktionen bereit:

- Erstellung einer Instrumentierung zum Ändern einzelner Parameter (**Measure / Calibrate**) (siehe „Einzelne Parameter während des Experiments ändern“ auf Seite 324)
- Speichern eines Parametersatzes (**Calibration / Parameters → Save**) (siehe: „Komplette oder partielle Parametrierungen speichern“ auf Seite 328).
- Bearbeiten ganzer Parametersätze (*.mpa) (siehe „Parameterdatei bearbeiten“ auf Seite 329)

Neben der Änderung von einzelnen Parametern oder ganzen Parametersätzen können über dieses Menü weitere Aktionen ausgeführt werden:

- Kopieren und Einfügen von einzelnen Parameterwerten (**Calibration / Parameters → Copy/Paste**)
- Bearbeiten des Änderungsstatus eines Parameters/Messwertes (**Calibration / Parameters → Set State To Modified/Unmodified**)
- Erstellen von Mapping-Dateien (**Create Mapping File...**) (siehe „Mappingdateien“ auf Seite 336)
- Kopieren von Labeln in die Zwischenablage (**Copy Label To Clipboard**)
- Vollständige Darstellung aller Hardware-Parameter (**Show all HW Ports**)
- Suchen nach GUIs, in denen ein bestimmter Parameter/Messwert verwendet wird (**Find in instruments...**)

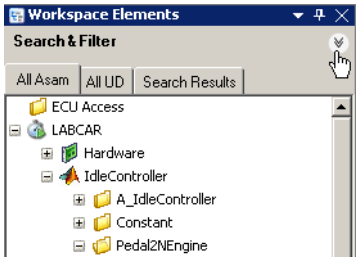
4.6.3 Einzelne Parameter während des Experiments ändern

Die Information über die Werte von Parametern und die Änderung derselben erfolgt in einem Anzeigeinstrument von ETAS EE (Fenster „Instrumentation“). Typischerweise nimmt man dafür eine „Edit Box“ (s.u.).

Da die Zahl der Parameter/Messgrößen eines Experimentes oft sehr umfangreich sein kann, steht eine Suchfunktion zur Verfügung.

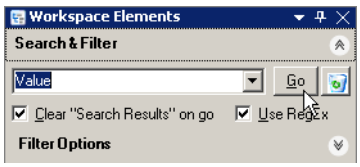
Parameter suchen

- Klappen Sie im Fenster „Workspace Elements“ die Eingabezeile für den Suchbegriff durch Anklicken des Symbols auf.

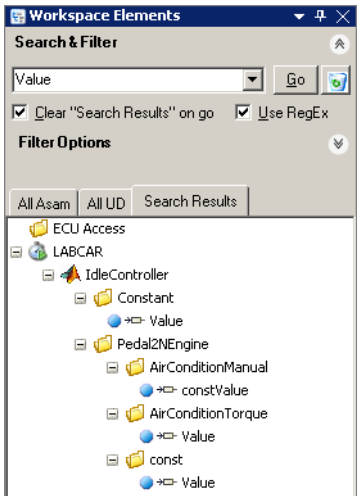


Die Eingabezeile wird angezeigt.

- Geben Sie eine Suchstring ein und klicken Sie **Go**.



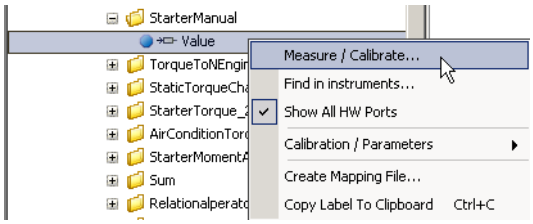
Das aktive Register wechselt zu „Search Results“, in dem die Ergebnisse der Suche angezeigt werden.



Durch Auswahl geeigneter Filterkriterien („Filter Options“) können Sie die Ergebnisliste ggf. weiter einschränken.

GUI erstellen

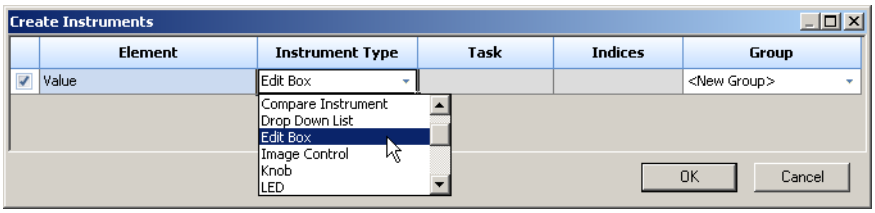
- Rechtsklicken Sie den Parameter, den Sie darstellen/ bearbeiten wollen.



Hinweis

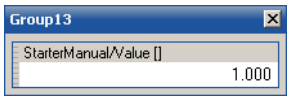
Sie können unter Verwendung von <UMSCHALT> oder <STRG> auch mehrere Parameter auswählen. Zudem können Sie auch ganze Ordner auswählen.

Das Fenster „Create Instruments“ wird geöffnet.



- Wählen Sie unter“ Instrument Type“ „Edit Box.“

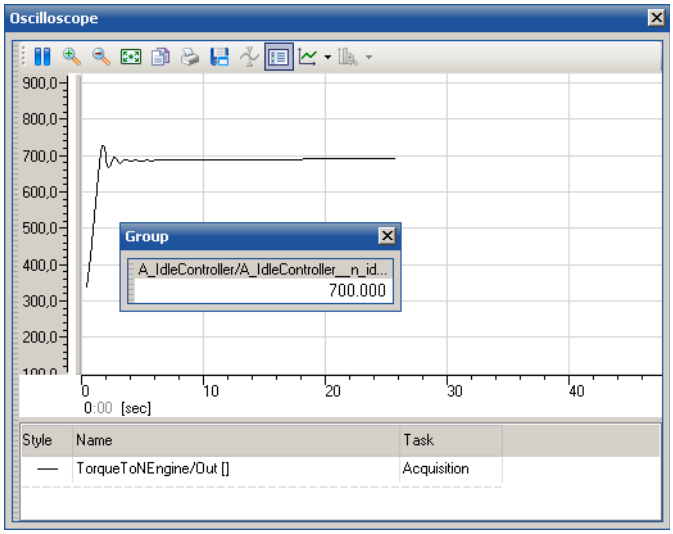
  Das GUI wird erstellt und der aktuellen Wert des Parameters dargestellt.



Parameter im laufenden Experiment ändern

- Starten Sie das Experiment.
- Klicken Sie in das GUI mit dem zu ändernden Parameter.

  Der Parameter kann jetzt editiert werden.

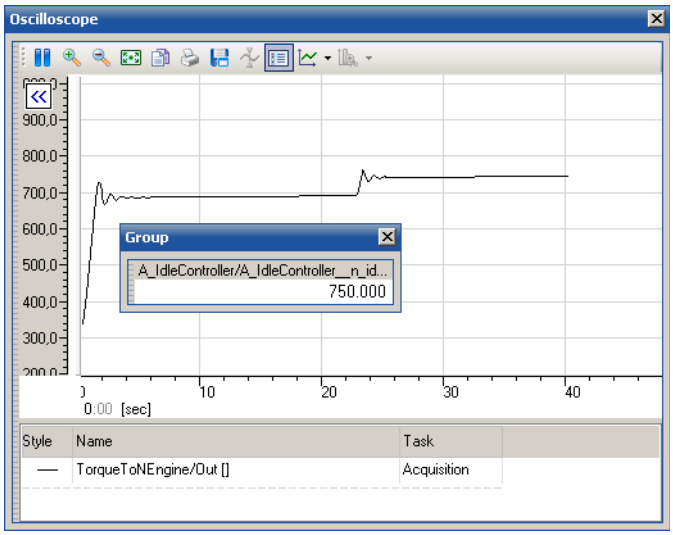


- Geben Sie den neuen Wert ein.
- Drücken Sie <EINGABE>

oder

- Klicken Sie außerhalb des GUIs.

- Der Wert des Parameters wird in der laufenden Simulation verändert.



Anzeige und Zurücksetzen des Änderungsstatus

Wenn der Wert eines Parameters in einem laufenden Experiment gegenüber dem Wert beim Herunterladen des Codes mit der initialen Parametrierung verändert wurde, so wird dieser in roter Farbe hervorgehoben.

Diese Hervorhebung kann auch über das Kontextmenü mit **Calibration / Parameters → Set State To Modified** herbeigeführt und mit **Calibration / Parameters → Set State To Unmodified** wieder zurückgesetzt werden.

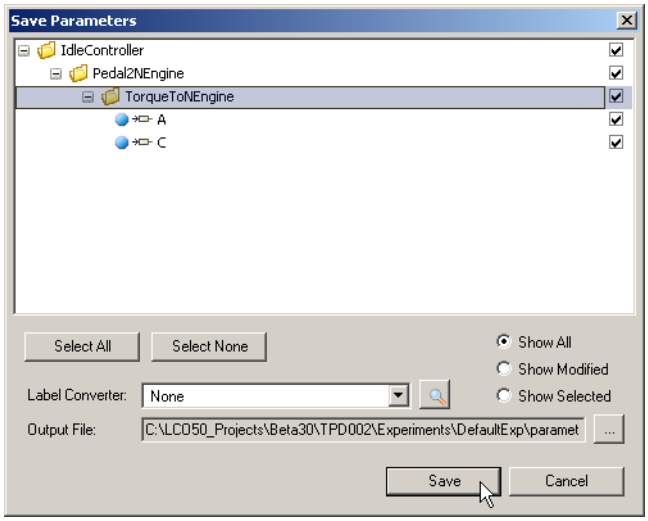
4.6.4 Komplette oder partielle Parametrierungen speichern

Zum Speichern von Parametrierungen im laufenden Experiment kann der Anwender im Fenster „Workspace Elements“ ein oder mehrere Parameter in einem Ordner, einen oder mehrere Ordner oder die ganze Liste auswählen und anschließend als Datei speichern.

Parametersatz speichern

- Wählen Sie die Parameter/Ordner/Module, deren Werte/Parametrierung Sie in einer Datei speichern wollen.
- Rechtsklicken Sie und wählen Sie **Calibration / Parameters → Save**.

  Das Fenster „Save Parameters“ wird geöffnet.



Alle Parameter der zuvor gewählten Hierarchie/Ordner werden angezeigt und sind ausgewählt. Mit **Select All** oder **Select None** kann die Auswahl „global“ verändert werden – darüber hinaus kann die Liste der wählbaren Elemente mit **Show All**, **Show Modified**, **Show Selected** gefiltert werden.

Die Wahl der Parameterdatei erfolgt unter „Output File“ – Klicken von **...** öffnet ein Dateiauswahlfenster, in dem Name und Format der Datei (*.mpa, *.dcm oder *cdfx) gewählt werden können.

Zudem können in diesem Dateiauswahlfenster zwei weitere Optionen aktiviert werden:

- Add To Experiment

  Die Datei wird automatisch zum Experiment hinzugefügt, d.h. sie ist im Experiment Explorer im Ordner „Parameter Files“ vorhanden und zugänglich.
- Activate File

  Die Datei wird aktiv gesetzt (siehe „Parameterdatei aktiv setzen“ auf Seite 330)

4.6.5 Verwaltung von Parameterdateien im Experiment Explorer

Im Ordner „Parameter Files“ des Experiment Explorers sind alle Parameterdateien aufgeführt, die zum Experiment gehören.

Über das Kontextmenü können Sie folgende Aktionen ausführen:

- „Parameterdatei bearbeiten“ auf Seite 329
- „Parameterdatei hinzufügen“ auf Seite 329
- „Reihenfolge der Parameterdateien festlegen“ auf Seite 330
- „Parameter zum Experiment laden“ auf Seite 330
- „Parameterdatei aktiv setzen“ auf Seite 330
- „Parameterdatei umbenennen“ auf Seite 331
- „Parameterdatei aus dem Experiment entfernen“ auf Seite 331
- „Parameterdatei löschen“ auf Seite 331

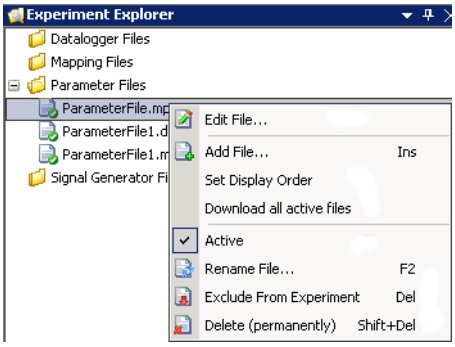


Abb. 4-11 Das Kontextmenü des Ordners „Parameter Files“

Parameterdatei bearbeiten

Parameter, die zuvor in einer Datei gespeichert wurden (siehe „Komplette oder partielle Parametrierungen speichern“ auf Seite 328), können an dieser Stelle zum Bearbeiten geöffnet werden.

- Wählen Sie **Edit File** oder doppelklicken Sie Datei.

  Bei einer *.mpa-Datei wird der LABCAR-PA V1.0 - Parameterization Assistant geöffnet und die Parameterdatei geladen.

Parameterdatei hinzufügen

- Um zum Experiment weitere Parameterdateien hinzuzufügen, wählen Sie **Add File**.
- Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie die hinzuzufügende(n) Datei(en) (Mehrfachauswahl mit <UMSCHALT> oder <STRG> ist möglich) und klicken Sie **Open**.

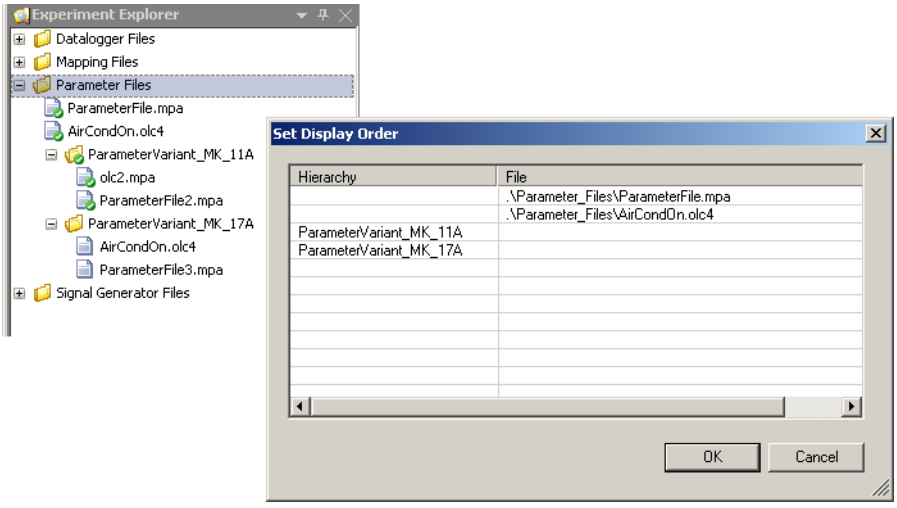
  Die Parameterdateien werden zum Ordner „Parameter Files“ hinzugefügt.

Reihenfolge der Parameterdateien festlegen

Parameterdateien werden beim Laden des Simulationscodes in der Reihenfolge auf das Experimental-Target geladen, in der sie im Ordner „Parameter“ dargestellt sind.

- Wählen Sie **Set Display Order**, um diese Reihenfolge zu ändern.

  Das Fenster „Set Display Order“ wird geöffnet.



  In diesem Fenster sind in der Spalte „Hierarchy“ die Ordner mit Varianten (auch Parameter Combinations (*.pac) genannt – siehe „Varianten und Parameter Combinations“ auf Seite 331) und in der Spalte „File“ die auf oberster Ebene befindlichen Parameterdateien zu sehen.

- Verschieben Sie einen Ordner oder eine Datei in der Liste einfach per Drag & Drop nach oben oder unten
- Klicken Sie **OK**.

  Die Reihenfolge der Ordner/Dateien im Experiment Explorer wird entsprechend geändert.

Parameter zum Experiment laden

- Um alle aktiv gesetzten Parameterdateien zu einem Experiment herunter zu laden, wählen Sie **Download all active files**.

Parameterdatei aktiv setzen

- Um eine Parameterdatei zu aktivieren, wählen Sie im Kontextmenü die Option **Active**.

  Die Parametrierung in dieser Datei wird in diesem Moment automatisch auf das Target geladen.

- Durch erneutes Klicken können Sie diesen Zustand wieder rückgängig machen.

Parameterdatei umbenennen

- Um eine Parameterdatei umzubenennen, wählen Sie **Rename File**.

  Es wird ein Dialogfenster geöffnet, in dem Sie einen neuen Dateinamen eingeben können.

Parameterdatei aus dem Experiment entfernen

- Um eine Parameterdatei aus dem Experiment zu entfernen, wählen Sie **Exclude From Experiment**.

  Die Zuordnung wird aufgehoben und die Datei aus der Ansicht des Experiment Explorers entfernt.

Hinweis

Die Datei wird nicht gelöscht - lediglich ihre Zuordnung zum Projekt wird aufgehoben.

Parameterdatei löschen

- Um eine Parameterdatei sowohl aus dem Experiment zu entfernen als auch in Ihrem Dateisystem zu löschen, wählen Sie **Delete (permanently)**.

Das Kontextmenü des Ordners „Parameter Files“ erlaubt darüber hinaus, sogenannte Parameter Variants erstellen, die im Folgenden beschrieben werden.

4.6.6 Varianten und Parameter Combinations

Beim Steuergerätetest hat man es häufig mit mehreren Varianten eines Steuergerätes zu tun - für jedes dieser Steuergeräte liegen zudem noch unterschiedliche Softwarestände vor.

Für die Testprojekt bedeutet dies, dass für jede Kombination von Steuergerätevariante und Softwarevariante eine Testprojektvariante vorhanden sein muss (siehe Abb. 4-12).

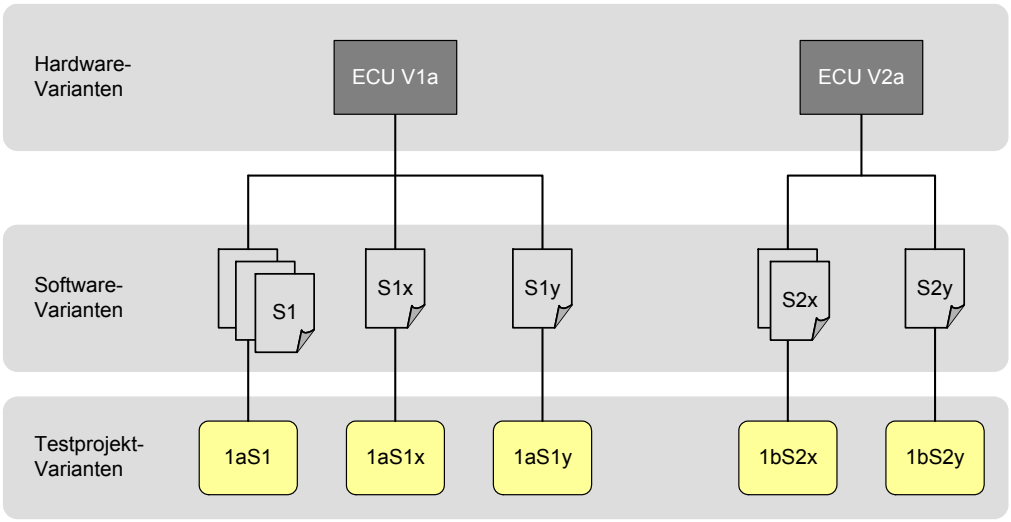


Abb. 4-12 Varianten von Steuergerätehardware, Steuergerätesoftware und die dazugehörigen Testprojekte

Dies bedeutet zum Einen eine hohe Redundanz und zum Anderen einen hohen Aufwand, wenn es zu „globalen", d.h. alle Varianten betreffenden Änderungen kommt.

Im Testprojekt können sich Unterschiede zwischen den Varianten auf folgende Projektdaten auswirken:

- Modellparameter
- Open-Loop Configuration (Sensorkennlinien etc.)
- Konfiguration der I/O-Hardware (Messmodi, Auflösung, etc.)
- Anschlussdaten von Steuergerät an I/O-Hardware

Die Lösung des Problems liegt in der getrennten Verwaltung von „globalen“ Daten und von Differenzdaten (siehe Abb. 4-13):

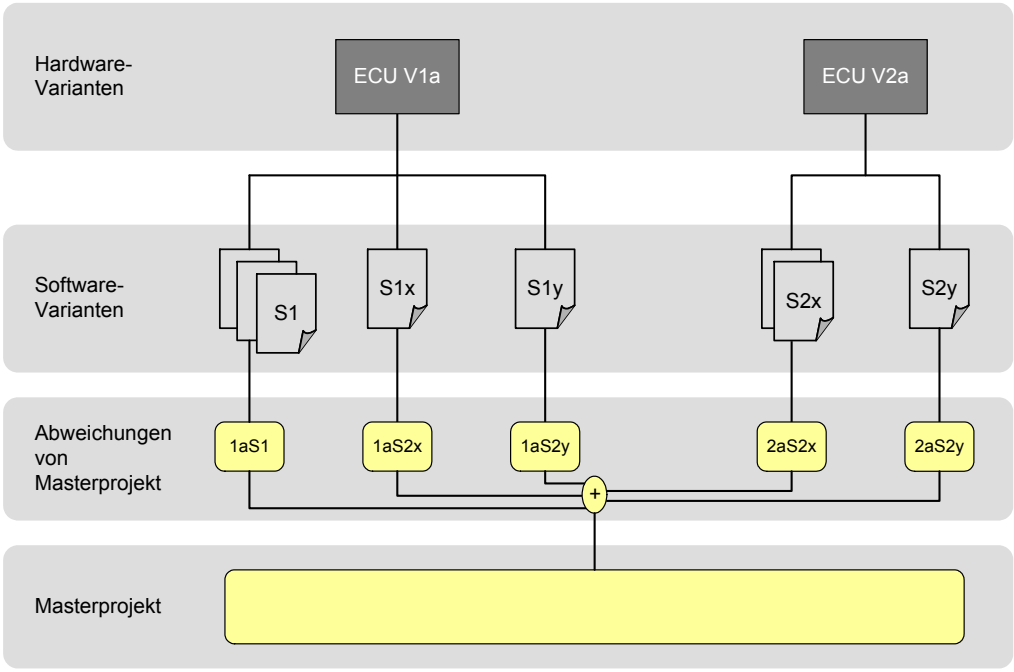


Abb. 4-13 Masterprojekt und Dateien für Abweichungen (Varianten) von diesem

Masterprojekt

Das Masterprojekt ist ein komplettes LABCAR-OPERATOR-Projekt, das eine Default-Parametrierung enthält. Es ist Single-Source und frei von Redundanz.

Abweichungen

Die Differenzdatei enthält die speziellen Projektparameter, die das Projekt für eine bestimmte UuT-Variante mit einer bestimmten Softwarevariante verwendbar macht. Diese Dateien werden Varianten genannt.

Was beeinhaltet eine Variante?

Varianten können folgende Arten von Daten enthalten:

- Modellparameter

  z.B. für verschiedene Motorkomponenten
  (gespeichert in einer *.mpa- , *.dcm- oder *.cdfx-Datei)
- OLC-Parameter

  z.B. Sensorkennlinien verschiedener Lambdasonden
  (gespeichert in *.olc-Datei)
- Parameter der Hardwarekonfiguration

  z.B. unterschiedliche Einstellungen der I/O-Hardware
  (gespeichert in einer *.mpa- oder *.dcm-Datei)
- Anschlussdaten: Steuergerät - I/O-Hardware

  (gespeichert in der Datei „properties.ecu“)

Die ersten drei Arten können zusammengefasst und durch Aktivierung zum Experiment geladen werden.

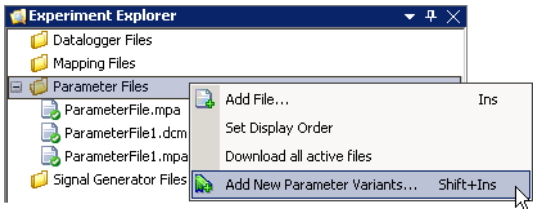
Parameter Combinations

Das Laden von Modellparametern in ein Online-Experiment bezieht sich in der Regel immer auf eine geordnete Sequenz von Parameterdateien, die in der Reihenfolge, wie sie im Ordner „Parameter Files“ dargestellt sind, auch auf das Target geladen werden.

Derartige Sequenzen können als sog. Parameter Combinations (*.pac) exportiert werden und einem Projekt in beliebiger Zahl zugeordnet werden.

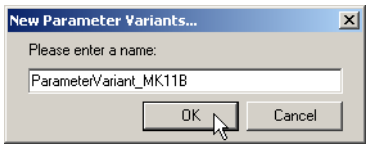
Eine neue Variante erstellen

- Rechtsklicken Sie den Ordner „Parameter Files“ und wählen Sie **Add new Parameter Variants**.



  Das Fenster „New Parameter Variants“ wird geöffnet.

- Geben Sie einen Namen für diese Variante ein und klicken Sie **OK**.



  Ein Dateiauswahlfenster wird geöffnet, in dem Sie diejenigen Dateien wählen können, die zur neuen Variante gehören sollen.

- Wählen Sie die Dateien und klicken Sie **Open**.

  Die Variante wird erstellt und zum Experiment hinzugefügt.

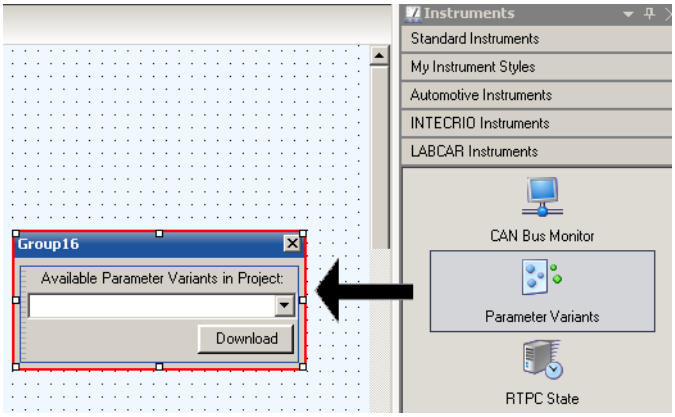
Zur Verwaltung von Parameter Variants steht die bereits in „Verwaltung von Parameterdateien im Experiment Explorer“ auf Seite 329 beschriebene Funktionalität zur Verfügung:

- Weitere Dateien zu Variante hinzufügen (**Add File**)
- Ändern der Reihenfolge (**Set Display Order**)
- Herunterladen aller aktivierten Dateien (**Download all Active Files**)
- Aktiv setzen oder deaktivieren (**Active**)
- Datei umbenennen (**Rename File**)
- Datei aus Projekt entfernen (**Exclude From Experiment**)
- Datei löschen (**Delete (permanently)**)

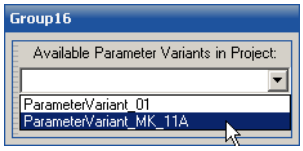
Ausgesuchte Varianten zum Experiment laden

Neben der o.g. Möglichkeit, aus dem Kontextmenü alle Parameter Variants des Experiments herunterzuladen (**Download all Active Files**), gibt es auch ein GUI, in dem Sie eine einzelne Variante wählen und zu einem beliebigen Zeitpunkt zum Experiment laden können.

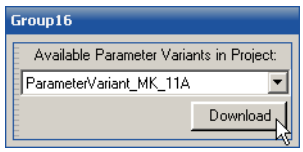
- Wählen Sie im Fenster „Instruments“ (Register „LABCAR Instruments“) das Instrument „Parameter Variants“ und ziehen Sie es in einen Layer.



- Wählen Sie aus der Liste eine der Varianten, die Teil des Projekts sind.



- Klicken Sie **Download**.



Die gewählte Variante wird zum Experiment geladen.

Variante als *.pac exportieren

Varianten können als Dateien (*.pac) exportiert werden und wieder zu einem Experiment hinzugefügt werden.

- Um eine Parameter Variant zu exportieren, wählen Sie den Ordner.
- Wählen Sie im Kontextmenü **Export to PAC File**.

  Ein Dateiauswahlfenster wird geöffnet, in dem Sie den Dateinamen der *.pac-Datei wählen können.
- Geben Sie den Dateinamen ein und wählen Sie **Speichern**.

Variante wieder importieren

- Um eine Parameter Variant zu einem Experiment hinzuzufügen, wählen Sie **Add File**.

  Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie die gewünschte Datei (mit der Erweiterung „pac“) und klicken Sie **Open**.

  Die Variante wird zum Experiment hinzu gefügt.

4.6.7 Mappingdateien

In einer Mappingdatei können ASAM-Labels auf benutzerdefinierte Namen abgebildet werden, was den Umgang mit diesen Labels erheblich vereinfacht.

Eine Mappingdatei wird z.B. mit ETAS LABCAR Modellen ausgeliefert, um die grafischen Bedienoberflächen der Modelle zu unterstützen, kann aber auch vom Anwender erstellt werden.

Grundsätzlich können einem Projekt beliebig viele Mappingdateien zugeordnet und in ETAS EE im Experiment Explorer verwaltet werden. Bearbeitet werden Mappingdateien in einem speziellen Editor, dem SUT Mapping File Editor.

Darstellung der Mappings im Register „All UD“

Zum Beispiel kann das Label

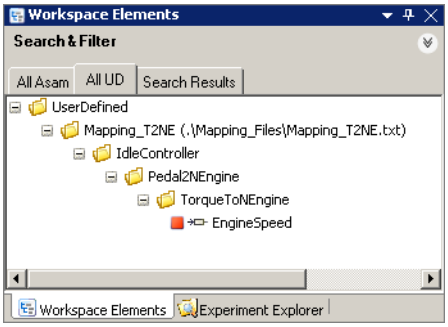
```
IdleController/Pedal2NEngine/TorqueToNEngine/Out
```

(das sog. Tool Label) einer Messgröße in einem Modul vom Anwender auf das Label

```
EngineSpeed
```

(dem sog. Test Label) gemappt werden.

Diese anwenderdefinierten Labels werden in den Workspace Elements im Register „All UD“ dargestellt – mit der Hierarchie, in der sie sich befinden und der Datei, in der sie definiert wurden.



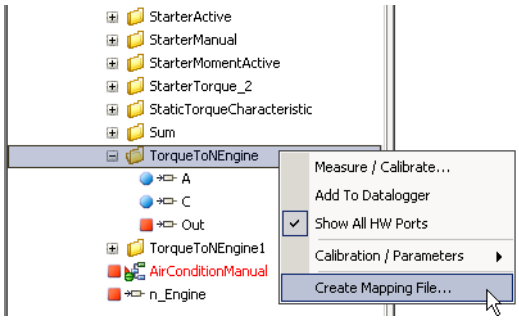
4.6.8 Mappingdateien erstellen und verwalten

In diesem Abschnitt erfahren Sie, wie Sie Mappingdateien erstellen und diese Dateien im Experiment Explorer über das Kontextmenü verwalten.

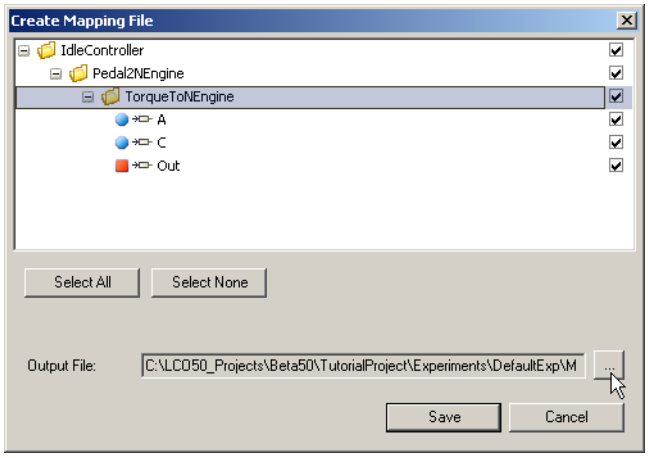
Mappingdatei erstellen

- Wählen Sie im Register „Workspace Elements“ den Parameter oder die Messgröße, für die Sie eine Mappingdatei erstellen wollen.

  Sie können auch Teile der Hierarchie (= Ordner) oder den Hauptknoten „LABCAR“ wählen.
- Wählen Sie im Kontextmenü **Create Mapping File**.



Das Fenster „Create Mapping File“ wird geöffnet.

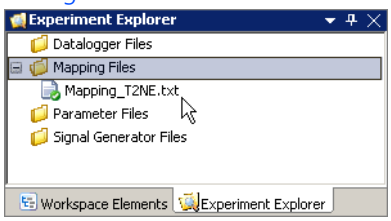


- Wählen Sie hier die Parameter/Messgrößen aus, die in der Mappingdatei vorhanden sein sollen.
- Klicken Sie **...**, um die Datei festzulegen, in der das Mapping gespeichert werden soll.

  Ein Dateiauswahlfenster wird geöffnet, in dem Sie einen Namen für die Datei eingeben können.
- Klicken Sie **Save**, um den Dateinamen zu speichern.
- Klicken Sie im Fenster „Create Mapping File“ **Save**, um die Mappingdatei zu speichern.

- Wechseln Sie in das Register „Experiment Explorer“ und doppelklicken Sie den Ordner „Mapping Files“

  Die erstellte Mappingdatei wurde zu Ihrem Experiment hinzugefügt und aktiv gesetzt (erkennbar an dem grünen Häkchen am Ordnersymbol).



Im Ordner „Mapping Files“ des Experiment Explorers sind alle Mappingdateien aufgeführt, die zum Experiment gehören. Über das Kontextmenü dieses Ordners können Sie Aktionen zum Bearbeiten und Verwalten ausführen, die im Folgenden beschrieben werden.

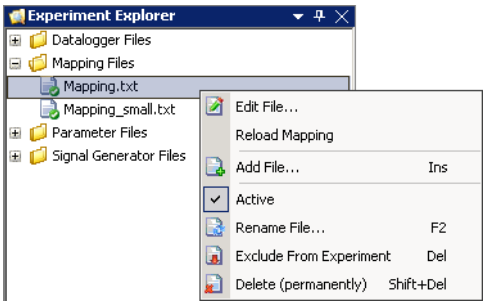


Abb. 4-14 Das Kontextmenü des Ordners „Mapping Files“

Über das Kontextmenü dieses Ordners können Sie Aktionen zum Bearbeiten und Verwalten ausführen, die im Folgenden beschrieben werden.

- „Mappingdatei bearbeiten“ auf Seite 338
- „Mapping erneut laden“ auf Seite 339
- „Mappingdatei zum Projekt hinzufügen“ auf Seite 339
- „Mappingdatei aktiv setzen“ auf Seite 339
- „Mappingdatei umbenennen“ auf Seite 339
- „Mappingdatei aus Projekt entfernen“ auf Seite 340
- „Mappingdatei dauerhaft löschen“ auf Seite 340

Mappingdatei bearbeiten

- Doppelklicken Sie die zu bearbeitende Datei im Experiment Explorer.

oder

- Rechtsklicken Sie die Datei und wählen Sie **Edit File**.

  Der SUT Mapping File Editor wird geöffnet.

Mappingdatei zum Projekt hinzufügen

- Rechtsklicken Sie im Experiment Explorer „Mapping Files“.
- Wählen Sie **Add File**.

  Ein Dateiauswahlfenster wird geöffnet.
- Wählen Sie die gewünschte Mappingdatei (*.txt, *.smf) und klicken Sie **OK**.

  Die Mappingdatei wird zum Projekt hinzugefügt.

Mapping erneut laden

- Rechtsklicken Sie im Experiment Explorer die zu ladende Datei.
- Wählen Sie **Reload Mapping**.

  Die gewählte Mappingdatei wird erneut in das Experiment geladen.

Mappingdatei aktiv setzen

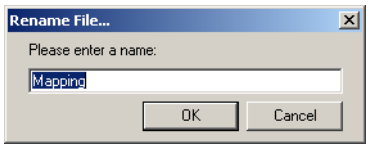
- Rechtsklicken Sie im Experiment Explorer die zu aktivierende Datei.
- Wählen Sie **Active**.

Die gewählte Mappingdatei wird aktiv gesetzt (erkennbar an dem Häkchen).

Mappingdatei umbenennen

- Rechtsklicken Sie im Experiment Explorer die zu umzubenennende Datei.
- Wählen Sie **Rename File**.

  Der „Rename File“ Dialog wird geöffnet.



- Geben Sie den neuen Namen ein und klicken Sie **OK**.

Mappingdatei aus Projekt entfernen

- Rechtsklicken Sie im Experiment Explorer die zu entfernende Datei.
- Wählen Sie **Exclude From Experiment**.

  Die Mappingdatei wird aus dem Projekt entfernt.

Hinweis

Die Datei wird nicht gelöscht - lediglich ihre Zuordnung zum Projekt wird aufgehoben.

Mappingdatei dauerhaft löschen

- Rechtsklicken Sie im Experiment Explorer die zu löschende Datei.
- Wählen Sie **Delete (permanently)**.

  Die Mappingdatei wird gelöscht.

4.7 Parameterdateien bearbeiten mit LABCAR-PA V1.0

LABCAR-PA V1.0 (Parameterization Assistant) dient zur übersichtlichen Darstellung und Bearbeitung von Parameterdateien.

LABCAR-PA bietet folgende Vorteile:

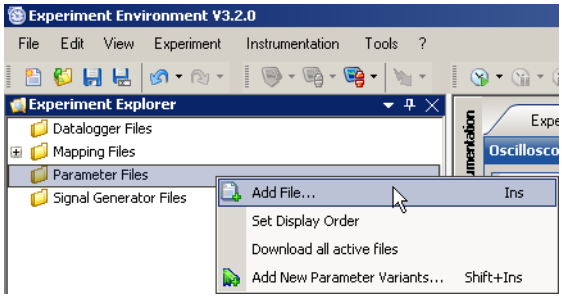
- Strukturierten Überblick über alle Modellparameter und deren Attribute
  - Darstellung nach Modellhierarchie oder funktionalen Einheiten
  - Filter- und Sortiermöglichkeiten
- Unterstützung des Parametrisierprozesses durch einfachen Zugriff auf Parameter:
  - Online- und Offline-Betrieb
  - Visualisierung des Parametrierstatus durch Attribute
  - Visualisierung von Änderungen
  - Vergleich mit einer Referenzparametrierung möglich
- Single-Source-Verwaltung von Modellparametrierungen in XML-Dateien
- Import und Export von Parametern aus und in DCM- und M-Dateien
- Einfacher Update des Modells

4.7.1 Parameterdateien verwalten

Im Experiment Explorer von ETAS EE befindet sich der Ordner „Parameter Files“, in dem Sie die Parameterdateien des Experiments verwalten können.

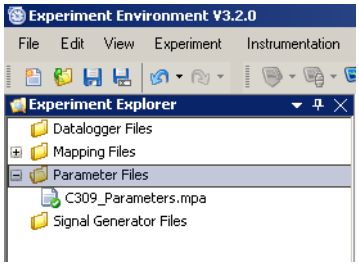
Parameterdatei zum Experiment hinzufügen

- Wählen Sie im Experiment Explorer den Ordner „Parameter Files“ und rechtsklicken Sie.
- Im Kontextmenü wählen Sie **Add File**.



- Wählen Sie im Dateiauswahlfenster die hinzuzufügenden Datei.

  Die Datei wird hinzugefügt.

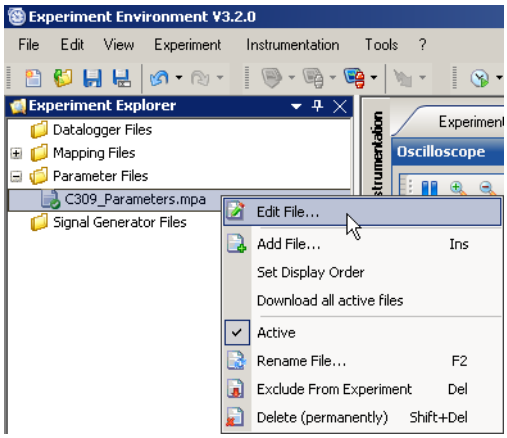


Parameterdatei editieren

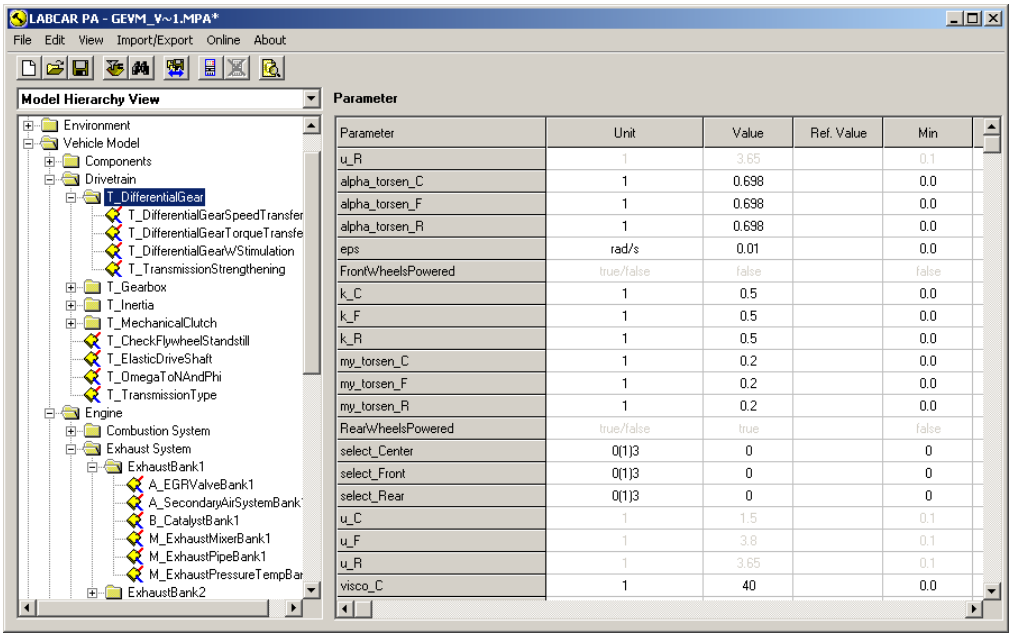
- Um eine Parameterdatei zu bearbeiten. doppelklicken Sie diese

oder

- Wählen Sie im Kontextmenü **Edit File**.



LABCAR-PA wird geöffnet und die gewählte Datei wird geladen.



Im Folgenden finden Sie eine Beschreibung der Bedienoberfläche von LABCAR-PA und eine Anleitung zum Arbeiten mit LABCAR-PA.

4.7.2 Die Parameterliste

In diesem Abschnitt werden die Funktionen der Parameterliste beschrieben. Im Einzelnen finden Sie Informationen über:

- „Umfang der Parameterliste“ auf Seite 343
- „Die Einträge in der Parameterliste“ auf Seite 343
- „Welche Einträge der Liste können editiert werden?“ auf Seite 345
- „Das Kontextmenü der Parameterliste“ auf Seite 345
- „Anzeigeoptionen für die Parameterliste“ auf Seite 346

Wie Parameter editiert werden, ist in Abschnitt „Parameter editieren“ auf Seite 357 beschrieben.

Umfang der Parameterliste

Die Zahl der in der Parameterliste angezeigten Attribute (= Spalten) kann modifiziert werden (siehe „Die Einträge in der Parameterliste“ auf Seite 343). Die Einstellung erfolgt im Menü **View** (siehe Abschnitt „Menü „View““ auf Seite 349).

Folgende Einstellungen sind möglich:

- **Basic Attributes**

  Hier werden außer dem Parameternamen nur die Spalten „Unit“, Value“ und „Comment“ angezeigt.
- **All Attributes**

  Dabei werden alle in Abschnitt aufgeführten Einträge dargestellt.
- **Custom Attributes**

  Hier werden alle über den Menüpunkt **View → Customize Attribute View** ausgewählten Einträge dargestellt (siehe auch „Menü „View““ auf Seite 349).

Die Einträge in der Parameterliste

Im Folgenden wird die Bedeutung der einzelnen Spalten der Parameterliste beschrieben.

- Parameter

  Dies ist der Name des Parameters, wie er im Modell verwendet wird.
- Unit

  Die physikalische Einheit des Parameters, z.B. km/h oder Pa/s.
- Value

  Der Wert des skalaren Parameters oder der erste Wert eines nicht-skalaren.
- Ref. Value

  Der Wert des Parameters in der Referenzdatei.
- Min

  Der empfohlene Minimalwert des Parameters.
- Max

  Der empfohlene Maximalwert des Parameters

- Dimension

  Die Dimension des Parameters

  – Skalar (= Festwert)
  – Array (= Festwerteblock)
  – 1D-Table (= Kennline = Curve)
  – 2D-Table (= Kennfeld = Map)

- Scope

  Der Gültigkeitsbereich des Parameters:

  – lokal (local)
  – exportiert (exported)
  – importiert (imported)

    Wenn ein Parameter importiert ist, wird in dieser Spalte der Name des Blockes angezeigt, aus dem exportiert wurde.

- Proc. State

  Beschreibt den Status des Parameters im Parametrierungsprozess.

  – Grobparametrisierung (coarse)
  – Feinparametrisierung (fine)
  – Schalter (switch)

- Type

  Der Variablentyp:

  – Ganzzahl (sdisc)
  – Fließkommazahl (cont)
  – Boolesch (log)

- ModelOption

  Funktionaler Kontext des Parameters (Allgemein, VVLT, GDI, etc.)

- UserGroup

  Der Wert „initial“ bedeutet, dass dieser Parameter zu denen gehört, deren Wert für einen ersten Lauf eines LABCAR-Projektes eingestellt sein muss. Diese Parameter werden mit **View → Initial Project Parameters** angezeigt.

  Alle anderen Werte sind bei dieser Version ohne Bedeutung.

- Mod. Status

  Änderungsstatus des Parameters:

  – unchanged
  – changed
  – accepted

  Aus dem Änderungsstatus eines Parameters resultierende farbliche Hervorhebungen sind im Abschnitt „Anzeigeoptionen für die Parameterliste“ auf Seite 346 beschrieben.

- Comment

  Enthält einen kurzen Kommentar zum Parameter.

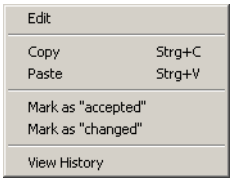
Welche Einträge der Liste können editiert werden?

Die Werte folgender Spalten können vom Anwender bearbeitet werden:

- Value
- Min
- Max
- Proc. State
- Mod. Status

Das Kontextmenü der Parameterliste

Ein rechter Mausklick auf eine Zeile der Tabelle öffnet ein Kontextmenü, mit dem Werte geändert werden können, der Änderungszustand (changed, accepted) gesetzt werden kann und die Änderungsgeschichte eingesehen werden kann.



- Edit

  Editiert den Parameter in der aktuellen Spalte.

  Hinweis

  Der Wert des Parameters (Spalte „Value") wird einfacher dadurch geändert, indem man ihn mit der linken Maustaste anklickt.

- Copy, Paste

  Kopieren und Einfügen eines Wertes (auch ganzer Bereiche)

- Mark as accepted

  Damit wird der Wert eines Parameters als akzeptiert gekennzeichnet.

- Mark as changed

  Damit wird ein Parameter als geändert gekennzeichnet.

- View History

  Öffnet das Dialogfenster „Parameter History", in dem die History des Parameters eingesehen und editiert werden kann (Näheres siehe „Parameter History einsehen und editieren" auf Seite 362).

Hinweis

Sollte es zu einer Fehlermeldung kommen, ist der Pfad zu den Dokumentationsdateien falsch eingestellt (siehe „Pfade für Arbeits-, Modell- und Dokumentationsverzeichnis" auf Seite 354) oder die Datei ist nicht vorhanden.

Anzeigeoptionen für die Parameterliste

Um bestimmte Eigenschaften von Parametern anzuzeigen, gibt es für folgende Fälle besondere Hervorhebungen bei der Darstellung in der Parameterliste:

- Der Parameter wurde geändert
- Änderung des Parameters akzeptiert
- Parameter ist importiert
- Parameter ist importiert und wurde geändert
- Parameter ist importiert und wurde akzeptiert
- Geänderter Parameter mit geladener Referenzdatei

Der Parameter wurde geändert:

Wenn ein Parameter geändert worden ist, werden folgende Listeneinträge in rot dargestellt:

- Parametername
- Parameterwert
- Modification Status

Außerdem wird der „Modification Status“ von „unchanged“ auf „changed“ gesetzt.

Änderung des Parameters akzeptiert :

Wenn Änderung des Parameters sich in einem bestimmte Stadium der Parametrierung als sinnvoll erweist, kann der „Modification Status“ auf „accepted“ gesetzt werden (siehe Abschnitt „Das Kontextmenü der Parameterliste“ auf Seite 345).

Ein vorher roter Listeneintrag wird dann grün dargestellt.

Dasselbe gilt übrigens auch für Parameter mit „Modification Status“ „unchanged“.

Parameter ist importiert :

Ein Parameter kann nicht in einem Block geändert werden, wenn er importiert ist. Ein solcher importierter Parameter wird in der Parameterliste grau dargestellt.

Ein importierter Parameter kann nur in dem Block geändert werden, aus dem er exportiert wurde. Wie dieser Block ermittelt wird, ist in Abschnitt „Parameter Scope Binding ermitteln“ auf Seite 374 beschrieben.

Parameter ist importiert und wurde geändert:

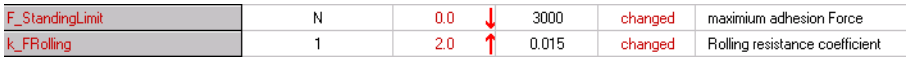
Wenn ein importierter Parameter in dem Block geändert wurde, aus dem er exportiert ist, so wird der Parametername rot dargestellt, die beide anderen Einträge (Parameterwert und Modification Status) orange.

Parameter ist importiert und wurde akzeptiert:

Wenn ein importierter, geänderter Parameter in dem Block akzeptiert wurde, aus dem er exportiert ist, so wird der Parametername grün dargestellt, die beide anderen Einträge (Parameterwert und Modification Status) hellgrün.

Geänderter Parameter mit geladener Referenzdatei:

Wenn ein Parameter geändert worden ist und es ist eine Referenzdatei geöffnet, so wird rechts vom Wert des Parameters durch eine roten Pfeil nach oben bzw. nach unten die Erhöhung bzw. Verringerung des Wertes gegenüber dem Wert in der Refenzdatei angezeigt.



Handelt es sich bei dem geänderten Parameter um eine Tabelle, können darin sowohl Werte nach oben oder nach unten verändert worden sein. In diesem Fall wird dies durch einen Doppelpfeil angezeigt.

Hinweis

Die Anzeige des Pfeiles hängt davon ab, ob sich der neue Wert um einen bestimmten, vom Benutzer vorgegebenen Prozentsatz vom ursprünglichen Wert unterscheidet (siehe „Einstellungen für Referenzdateien“ auf Seite 354).

Hinweis

Bei Parametern des Typs „Enumeration“ werden Änderungen nicht angezeigt.

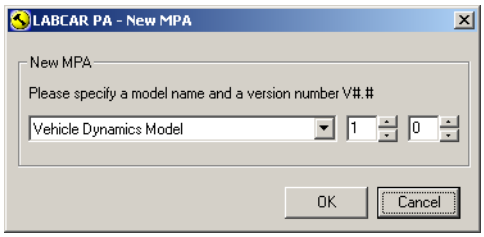
4.7.3 Beschreibung der Menüs

In diesem Abschnitt finden Sie eine allgemeine Beschreibung der Funktion der vorhandenen Menüeinträge.

Menü „File“

- **File → New**

  Legt eine neue (leere) Projektparameterdatei (*.mpa) an.



  Die Auswahl eines Modells oder einer Modellkomponente und der Version dient der Zuordnung der Parameterdaten in der erstellten Datei zu einem Modell oder einer Modellkomponente.
  Bei kompletten Modellen wird auch die gesamte Hierarchie miterstellt - der Modellname erscheint dann als oberstes Element in der Strukturansicht.

- **File → Open**

  Öffnet ein Dateiauswahlfenster, mit dem eine Projektparameterdatei (*.mpa) geladen werden kann.

  Die Festlegung eines Arbeitsverzeichnisses erfolgt über das Menü **Edit → Options** (siehe „Menü „Edit"" auf Seite 348).
- **File → Close**

  Schließt die geöffnete Projektparameterdatei.
- **File → Save**

  Speichert die momentan geöffnete Projektparameterdatei.
- **File → Save As**

  Speichert die aktuelle Projektparameterdatei unter einem anzugebenden Namen.
- **File → Open Reference**

  Hiermit kann eine Referenzdatei (siehe Abschnitt „Referenzdateien" auf Seite 363) geladen werden.
- **File → Set To Reference File**

  Hiermit kann die Parametrierung der aktuellen Datei als Referenzdatei gespeichert werden. Der voreingestellte Dateiname ist `<Dateiname>Reference_0.mpa`.
- **File → Close Reference**

  Schließt die aktuell geöffnete Referenzdatei.
- **File → Exit**

  Beendet das Programm.

Menü „Edit"

- **Edit → Edit Parameter**

  Zum Editieren des gewählten Parameters (siehe „Parameter editieren" auf Seite 357).
- **Edit → Filter Settings**

  Öffnet ein Dialogfenster, in dem Anzeigefilter und Sortierkriterien eingegeben werden können (siehe auch „Die Filter- und Sortierfunktion" auf Seite 369).
- **Edit → Search Model Parameters**

  Öffnet ein Dialogfenster, in dem nach Modellparametern gesucht werden kann (siehe „Die Suchfunktion für Modellparameter" auf Seite 372).
- **Edit → Options**

  Öffnet ein Dialogfenster, in dem Optionen festgelegt werden (siehe „Optionen einstellen" auf Seite 352).

Menü „View“

- **View → Initial Project Parameters**

  Wird diese Option gewählt, so werden nur diejenigen Parameter angezeigt, die für einen ersten Lauf eines LABCAR-Projektes notwendig sind.

- **View → Parameter Scope Binding**

  Öffnet eine Liste mit Parametern, die aus Blöcken exportiert und von anderen importiert sind. Zusätzlich werden die Namen der Blöcke angegeben, aus denen sie bzw. in die sie exportiert bzw. importiert werden (siehe „Parameter Scope Binding ermitteln“ auf Seite 374).

- **View → Basic Attributes**

  Wenn diese Option gewählt ist, werden in der Parameterliste nur die Attribute „Unit“, „Value“ und „Comment“ angezeigt (siehe auch „Umfang der Parameterliste“ auf Seite 343).

- **View → All Attributes**

  Bei dieser Option werden in der Parameterliste alle Attribute angezeigt (siehe auch „Umfang der Parameterliste“ auf Seite 343).

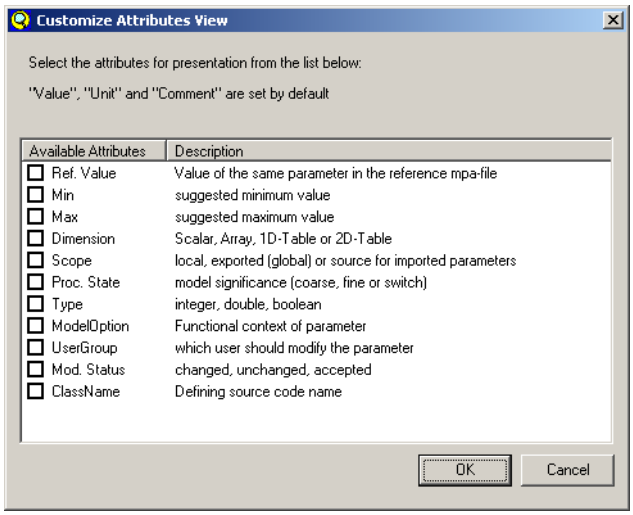
- **View → Custom Attributes**

  Bei dieser Option werden in der Parameterliste alle Attribute angezeigt, die bei **View → Customize Attribute View** ausgewählt wurden (siehe auch „Umfang der Parameterliste“ auf Seite 343).

- **View → Customize Attribute View**

  Hier können die Attribute ausgewählt werden, die mit der Option **View → Custom Attributes** angezeigt werden (siehe auch „Umfang der Parameterliste“ auf Seite 343).

  Die Bedeutung der Attribute ist in Abschnitt „Die Einträge in der Parameterliste“ auf Seite 343 beschrieben.



Menü „Import/Export“

- **Import/Export → Import/Export Parameter Data**

  Öffnet das Dialogfenster zum Import von Parametern aus oder zum Export in Parameterdateien (siehe „Parameter aus Datei importieren“ auf Seite 366 und „Parameter in Datei exportieren“ auf Seite 364).

Menü „Online“

- **Online → Go Online**

  Bei laufender Simulation wird geprüft, ob die Parameter der geöffneten mpa-Datei auch im Modell vorhanden sind (Kriterium: Gleichheit der ASAM-Label). Näheres zu diesem Thema siehe „Online-Parametrierung“ auf Seite 375.
- **Online → Synchronize ASAM Labels**

  Parameter in LabCar Developer-Modellen haben andere Namen als solche in Simulink-Modellen. Diese Funktion synchronisiert Parameter in mpa-Dateien mit aus Simulink-Modellen importierten Parametern und passt ggf. deren ASAM-Labels an.
- **Online → Read Parameter**
  - **Read Parameter from Target**

    Liest den Wert eines in der Parameterliste ausgewählten Parameters aus dem laufenden Modell.
  - **Read all Parameters from Target**

    Liest alle Werte der Parameter und der gewählten Hierarchie aus dem laufenden Modell.
- **Online → Write Parameter**
  - **Write Parameter to Target**

    Schreibt den Wert eines in der Parameterliste ausgewählten Parameters in das laufenden Modell. Diese Funktion wird allerdings schon beim Ändern des Wertes in der Parameterliste durchgeführt.
  - **Write all Parameters to Target**

    Schreibt alle Werte der Parameter und der gewählten Hierarchie in das laufenden Modell.
- **Online → Go Offline**

  Beendet die Online-Parametrierung.

4.7.4 Die Symbolleiste

Die folgenden der oben beschriebenen Menüfunktionen sind auch über eine Symbolleiste verfügbar:

- **Create new MPA file**

  Legt eine neue (leere) Projektparameterdatei (*.mpa) an. Gleichbedeutend mit **File → New**.

- **Open MPA file**

  Öffnet ein Dateiauswahlfenster, mit dem eine Projektparameterdatei (*.mpa) geladen werden kann. Gleichbedeutend mit **File → Open**.

- **Save MPA file**

  Speichert die momentan geöffnete Projektparameterdatei. Gleichbedeutend mit **File → Save**.

- **Filter settings**

  Öffnet ein Dialogfenster, in dem Anzeigefilter und Sortierkriterien eingegeben werden können. Gleichbedeutend mit **Edit → Filter Settings**.

- **Search parameters**

  Öffnet ein Dialogfenster, in dem nach Modellparametern gesucht werden kann. Gleichbedeutend mit **Edit → Search Model Parameters**.

- **Import/Export parameters**

  Öffnet das Dialogfenster zum Import von Parametern aus oder zum Export in Parameterdateien. Gleichbedeutend mit **Import/Export → Import/Export Parameter Data**.

- **Online parameterization**

  Bei laufender Simulation werden damit die aktuellen Parameter ins Modell geladen (Online-Parametrierung). Gleichbedeutend mit **Online → Go Online**.

- **Offline parameterization**

  Beendet die Online-Parametrierung. Gleichbedeutend mit **Online → Go Offline**.

- **View parameter scope binding**

  Öffnet eine Liste mit Parametern, die exportiert und importiert werden. Zusätzlich werden die Namen der Blöcke angegeben, aus denen sie exportiert bzw. in die sie importiert werden. Gleichbedeutend mit **View → Parameter Scope Binding**.

4.7.5 Arbeiten mit LABCAR-PA V1.0

In diesem Abschnitt finden Sie Informationen über das Arbeiten mit LABCAR-PA V1.0.

Im Einzelnen sind dies:

- „Optionen einstellen“ auf Seite 352

  In diesem Abschnitt wird die Einstellung verschiedener Optionen beschrieben.
- „Parameter editieren“ auf Seite 357

  In diesem Abschnitt erhalten Sie Informationen über das Editieren von Parametern.
- „Referenzdateien“ auf Seite 363

  Dieser Abschnitt gibt Ihnen Informationen über Referenzdateien.
- „Parameter in Datei exportieren“ auf Seite 364

  In diesem Abschnitt wird der Export von Modellparametern in M- oder DCM-Dateien beschrieben.
- „Parameter aus Datei importieren“ auf Seite 366

  In diesem Abschnitt wird der Import von Modellparametern aus M- oder DCM-Dateien beschrieben.
- „Die Filter- und Sortierfunktion“ auf Seite 369

  Dieser Abschnitt beschreibt die Filter- und Sortierfunktion für die Parameterliste.
- „Die Suchfunktion für Modellparameter“ auf Seite 372

  Dieser Abschnitt beschreibt die Suchfunktion für Modellparameter
- „Parameter Scope Binding ermitteln“ auf Seite 374

  In diesem Abschnitt wird beschrieben, wie Sie Parameter identifizieren, die von Blöcken des Modells im- oder exportiert werden.
- „Online-Parametrierung“ auf Seite 375

  In diesem Abschnitt wird die Parametrierung von laufenden Modellen beschreiben.

4.7.6 Optionen einstellen

In diesem Abschnitt wird die Einstellung folgender Optionen beschrieben:

- „Allgemeine Optionen“ auf Seite 353
- „Einstellungen für Referenzdateien“ auf Seite 354
- „Pfade für Arbeits-, Modell- und Dokumentationsverzeichnis“ auf Seite 354
- „Optionen für Parameterimport und -export“ auf Seite 355
- „History-Optionen“ auf Seite 356

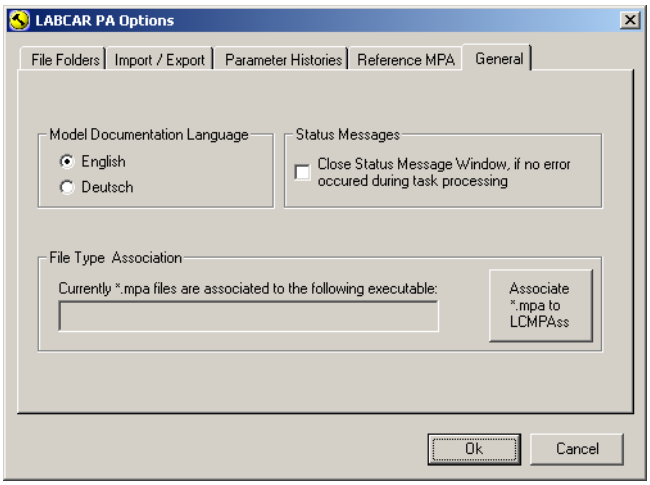
Allgemeine Optionen

Im Bereich „Allgemeine Optionen“ können folgende Einstellungen vorgenommen werden:

- Sprache der Modellblock-Dokumentation festlegen
- Verhalten des Fensters für Status Messages festlegen
- Verknüpfung des Dateityps `*.mpa` ändern

Um eine oder mehrere dieser Optionen zu ändern, gehen Sie wie folgt vor:

- Wählen Sie **Edit** → **Options**.
  Das Fenster „LABCAR-PA Options“ wird geöffnet.
- Wählen Sie das Register „General“.



Sprache der Modelldokumentation festlegen

- Wählen Sie unter „Model Documentation Language“ entweder „English“ oder „Deutsch“ aus.

Verhalten von Fenstern für Status Messages

- Wenn Sie wollen, dass das „Status Message Window“ nach der fehlerfreien Abarbeitung einer Task automatisch geschlossen wird, aktiveren Sie die Option „Close Status Message Window, if no error occured during task processing“.

Verknüpfung des Dateityps *.mpa ändern

Üblicherweise ist unter Microsoft Windows Betriebssystemen der Dateityp `*.mpa` mit Multimedia-Dateien assoziert, d.h. beim Doppelklick auf eine Datei mit dieser Extension wird ein Mutimedia-Player gestartet.

- Wenn Sie möchten, dass Dateien mit der Erweiterung „mpa“ mit LABCAR-PA V1.0 verknüpft werden, klicken Sie unter „File Type Association“ die Schaltfläche **Associate *.mpa to LCMPAss**.

Einstellungen für Referenzdateien

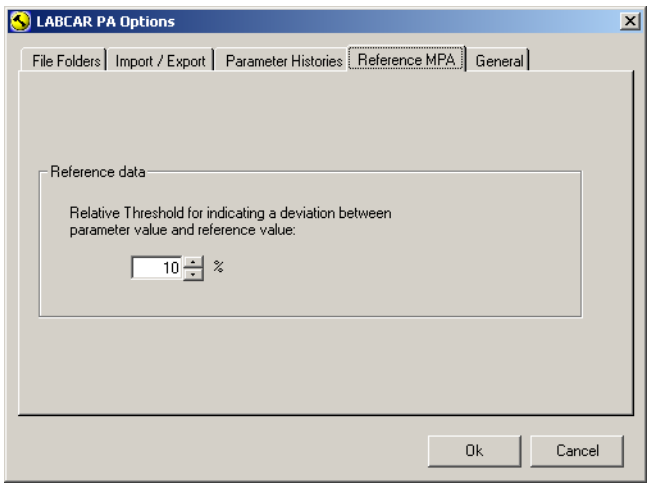
Wenn eine Referenzdatei geöffnet ist und ein Parameter der geladenen Parameterdatei um einen bestimmten Betrag von dem Betrag des Parameters in der Referenzdatei abweicht, wird der Eintrag in der Parameterliste mit einem Pfeil (nach oben oder nach unten) versehen.

Die Abweichung (in %), ab der es zu dieser Anzeige kommt, kann eingestellt werden.

Schwelle für Anzeige einer Abweichung einstellen

- Wählen Sie **Edit** → **Options**.

  Das Fenster „LABCAR-PA Options“ wird geöffnet.
- Wählen Sie das Register „Reference MPA“.



- Um die Schwelle einzustellen, klicken Sie auf die Pfeile oder geben Sie den gewünschten Prozentsatz in das Feld ein.

Pfade für Arbeits-, Modell- und Dokumentationsverzeichnis

Zum Arbeiten mit dem LABCAR-PA V1.0 müssen die Pfade für folgende Verzeichnisse richtig eingestellt sein:

- Arbeitsverzeichnis („Working Directory“)

  Hier befinden sich alle Dateien mit den Parametrisierdaten (*.mpa, *.dcm, *.m)
- Modellverzeichnis („Model Directory“)

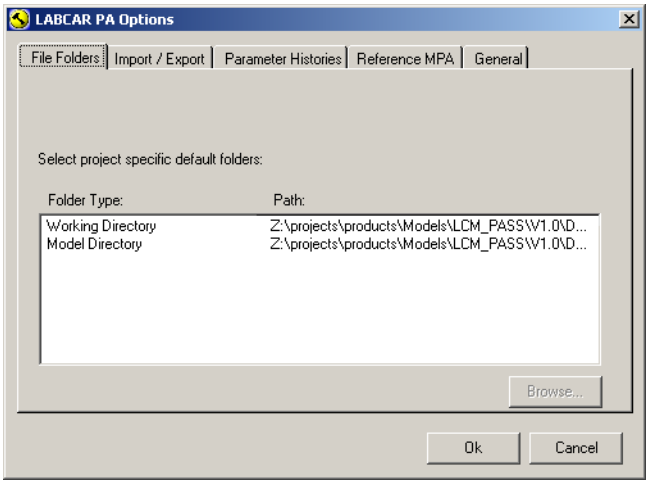
  Dieser Pfad wird in der aktuellen Version nicht verwendet.
- Verzeichnis für Modelldokumentation („Model Documentation Directory“)

  Pfad zur Block- und Modelldokumentation (*.pdf)

Arbeitsverzeichnis ändern

Um das Arbeitsverzeichnis zu ändern gehen Sie wie folgt vor:

- Wählen Sie **Edit** → **Options**.
  Das Fenster „LABCAR-PA Options“ wird geöffnet.



- Wählen Sie das Register „File Folders“.
- Wählen Sie mit einem Mausklick den Eintrag „Working Directory“.
- Klicken Sie die Schaltfläche **Browse**.
  Das Fenster „Select Folder“ wird geöffnet.
- Wählen Sie ein Laufwerk und ein Verzeichnis aus.
- Klicken Sie **OK**.

Die Änderung des Modell- und Dokumentationsverzeichnisses erfolgt auf die selbe Art und Weise.

Optionen für Parameterimport und -export

Für den Import oder Export von Parametern aus oder in Dateien lassen sich folgende Optionen festlegen:

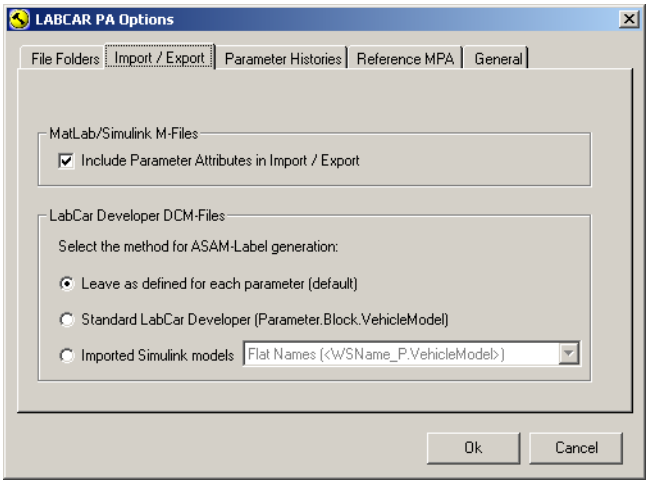
- Ob bei Import und Export aus und in Matlab Simulink M-Dateien ETAS-Attribute mitgenommen werden
- Die Methode zur Erzeugung von ASAM-Labeln

Import-/Exportoptionen ändern

Um diese Optionen zu ändern, gehen Sie wie folgt vor:

- Wählen Sie **Edit** → **Options**.
  Das Fenster „LABCAR-PA Options“ wird geöffnet.

- Wählen Sie das Register „Import/Export“.



Im Feld „Matlab/Simulink M-Files“ können Sie die Option „Include Parameter Attributes in Import/Export“ aktivieren. In diesem Fall werden die Parameterattribute mit in die M-Datei geschrieben oder aus dieser gelesen.

Im Feld „LabCar Developer DCM-Files“ kann die Methode für die Erzeugung von ASAM-Labeln bestimmt werden. Folgende Möglichkeiten stehen zur Auswahl:

- Leave as defined for each parameter (default)

  Das ASAM-Label, wie es in der mpa-Datei verwendet wird (Attribut „ModelPath“).
- Standard LabCar Developer (Parameter.Block.VehicleModel)

  LabCar Developer Syntax: Parametername.Blockname.Modellname
- Imported Simulink Models

  Optionen,die beim Simulink-Modellimport verwendet werden:
  - Flat Names (<WSName_P.VehicleModel>)

    Workspacename.Modellname
  - Standard SL Import

    Mit vollständiger Hierarchie
  - SL Import Reverse Quantity Names

    Mit vollständiger Hierarchie in umgekehrter Reihenfolge

History-Optionen

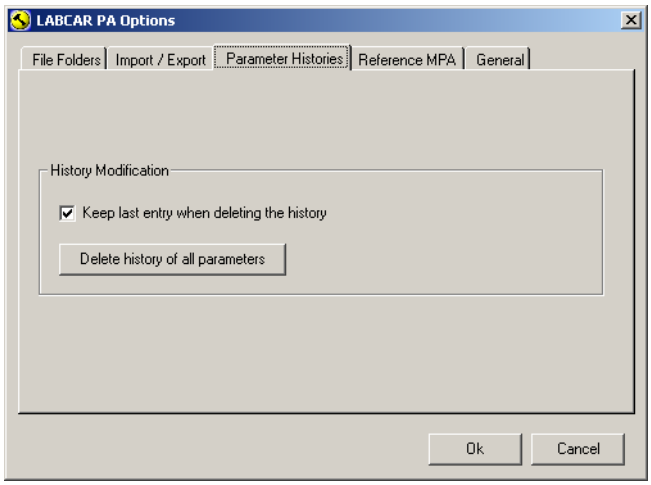
Bei jedem Editieren der Parameterdatei wird bei dem geänderten Parameter ein Historyeintrag hinzugefügt. Wird die Parameterdatei oft editiert, so nimmt deren Umfang zu - um den Umfang der Datei wieder zu reduzieren, können die Historyeinträge wieder gelöscht werden.

Als Option kann eingestellt werden, dass jeweils der letzte Eintrag nach der Löschung beibehalten wird.

History löschen

Um die Parameter-History zu löschen, gehen Sie wie folgt vor:

- Wählen Sie **Edit → Options**.
  Das Fenster „LABCAR-PA Options“ wird geöffnet.
- Wählen Sie das Register „Parameter Histories“.



- Wenn Sie den jeweils letzten Eintrag in der Parameter-History behalten möchten, so aktivieren Sie „Keep last entry when deleting the history“.
- Zum Löschen klicken Sie **Delete history of all parameters**.

4.7.7 Parameter editieren

In diesem Abschnitt erhalten Sie Informationen über das Editieren von Parametern. Im Einzelnen sind dies:

- „Parameterwerte editieren“ auf Seite 358
  Hier finden Sie eine Beschreibung, wie Parameter je nach ihrem Datenformat editiert werden.
- „Editieren von Minimal- und Maximalwerten“ auf Seite 360
  enthält Hinweise zum Editieren von Minimal- und Maximalwerten.
- „Der Tabelleneditor“ auf Seite 360
  beschreibt den Umgang mit dem Tabelleneditor.
- „Parameter History einsehen und editieren“ auf Seite 362
  gibt Informationen zum Umgang mit der Parameter History.

Parameterwerte editieren

Da Parameter unterschiedliche Formate besitzen können, unterscheidet sich auch die Art, wie diese editiert werden.

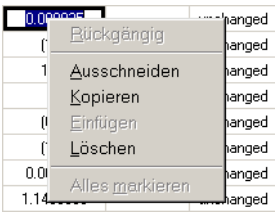
Folgende Fälle werden beschrieben:

- „Parameter ist Ganz- oder Fließkommazahl“ auf Seite 358
- „Parameter ist 1D- (Kennlinie) oder 2D-Table (Kennfeld)“ auf Seite 359
- „Parameter ist logischer Wert“ auf Seite 359
- „Parameter ist importiert“ auf Seite 359

Parameter ist Ganz- oder Fließkommazahl:

Zum Editieren eines Parameters klicken Sie mit der linken Maustaste in die entsprechenden Tabellenzelle in der Spalte „Value“. Die Tabellenzelle geht dann in den Editiermodus und kann geändert werden.

Wenn sich die Zelle im Editiermodus befindet, wird mit einem rechten Mausklick folgendes Kontextmenü geöffnet:



Über dieses Menü sind folgende Operationen möglich:

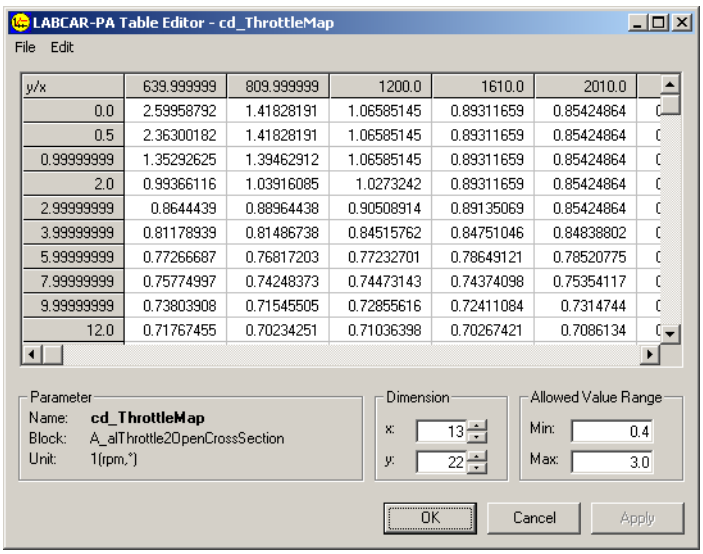
- Rückgängig machen einer eben durchgeführten Änderung
- Ausschneiden des Wertes
- Kopieren des Wertes oder ganzer Bereiche
- Löschen des Wertes
- Ganzen String markieren

Hinweis

Je nach Vorgeschichte sind jeweils nur bestimmte Menüeinträge aktiv.

Parameter ist 1D- (Kennlinie) oder 2D-Table (Kennfeld):

Zum Editieren einer Tabelle doppelklicken Sie mit der linken Maustaste in die entsprechenden Tabellenzelle in der Spalte „Value“. Es wird dann der Tabelleneditor für diesen Parameter geöffnet.



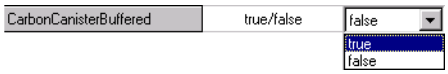
y/x	639.999999	809.999999	1200.0	1610.0	2010.0
0.0	2.59958792	1.41828191	1.06585145	0.89311659	0.85424864
0.5	2.36300182	1.41828191	1.06585145	0.89311659	0.85424864
0.99999999	1.35292625	1.39462912	1.06585145	0.89311659	0.85424864
2.0	0.99366116	1.03916085	1.0273242	0.89311659	0.85424864
2.99999999	0.8644439	0.88964438	0.90508914	0.89135069	0.85424864
3.99999999	0.81178939	0.81486738	0.84515762	0.84751046	0.84838802
5.99999999	0.77266687	0.76817203	0.77232701	0.78649121	0.78520775
7.99999999	0.75774997	0.74248373	0.74473143	0.74374098	0.75354117
9.99999999	0.73803908	0.71545505	0.72855616	0.72411084	0.7314744
12.0	0.71767455	0.70234251	0.71036398	0.70267421	0.7086134

Abb. 4-15 Der Tabelleneditor

Die Bedienung des Tabelleneditors ist im Abschnitt „Der Tabelleneditor“ auf Seite 360 beschreiben.

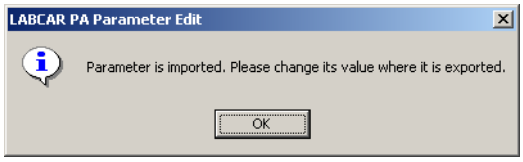
Parameter ist logischer Wert:

Wenn der Parameter aus einem Wahrheitswert (true/false) besteht, öffnet sich bei einem Mausklick eine Dropdown-Liste, aus der der gewünschte Wert gewählt werden kann.



Parameter ist importiert:

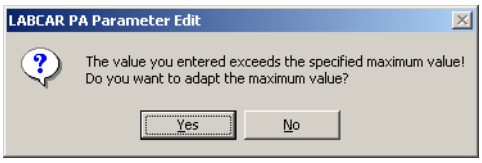
Wenn ein Parameter importiert ist, ist er in diesem Block nicht editierbar, was auch an der grauen Hervorhebung erkennbar ist. Wird trotzdem versucht, ihn zu editieren, erhalten Sie folgende Meldung:



Der Parameter kann nur in dem Block editiert werden, aus dem er exportiert worden ist. Um den Block zu ermitteln, aus dem dieser Parameter exportiert wurde, wählen Sie im Kontextmenü **Select source parameter (“exported”)**. Der entsprechende Block wird dann aktiviert und seine Parameter in der Liste angezeigt - der exportierte Parameter selbst ist blau hinterlegt.

Editieren von Minimal- und Maximalwerten

Die Minimal- und Maximalwertvorgaben für die Parameterwerte stellen sinnvolle Grenzen für das Funktionieren des Modells dar. Wird in LABCAR-PA bei der Eingabe eines Wertes ein solcher Grenzwert über- oder unterschritten, so wird eine Warnmeldung ausgegeben.



Der Anwender hat dann die Möglichkeit, diese Änderung abzubrechen oder trotzdem durchzuführen.

Bei Parametern vom Typ „Ganz- oder Fließkommazahl“ besteht zudem die Möglichkeit, diese Werte direkt zu ändern.

Der Tabelleneditor

Beim Doppelklick auf den Wert eines Parameters, der aus einer ein- oder zweidimensionalen Tabelle besteht, wird der Tabelleneditor geöffnet.

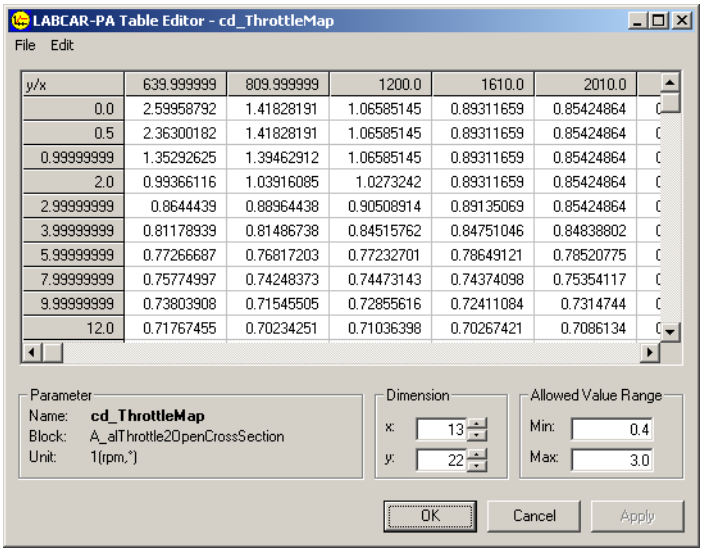
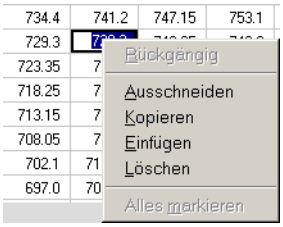


Abb. 4-16 Der Tabelleneditor mit Parameter „cd_ThrottleMap“

Eine einzelne Tabellenzelle kann mit einem linken Mausklick editiert werden.

Wenn sich die Zelle im Editiermodus befindet, wird mit einem rechten Mausklick folgendes Kontextmenü geöffnet.



Über dieses Menü sind folgende Operationen möglich:

- Rückgängig machen einer eben durchgeführten Änderung
- Ausschneiden des Wertes
- Kopieren des Wertes
- Löschen des Wertes
- Ganzen String markieren

Hinweis

Je nach Vorgeschichte sind jeweils nur bestimmte Menüeinträge aktiv.

Die Benutzeroberfläche des Tabelleneditors besteht aus einem Menü und mehreren Feldern, die im folgenden beschrieben werden.

Menü:

Das Menü des Tabelleneditors enthält folgende Einträge

- **File → Open Reference Table Viewer**

  Hiermit kann die entsprechende Tabelle der Referenzdatei geöffnet werden. Diese enthält einen Menüpunkt **Data → Copy Reference Data to Table**, der den gesamten Inhalt der Referenztabelle in die aktuelle Tabelle kopiert.

- **Edit → Edit Axes**

  Hiermit können die x- bzw. y-Achsenwerte der Tabelle verändert werden (siehe auch „Feld „Dimension“ “ auf Seite 361).

- **File → Import from File**

  Hiermit können Daten aus einer M- oder DCM-Datei importiert werden.

Feld „Parameter“:

Dieses Feld enthält Informationen zum Parameter.

- Name

  Der Name des Parameters

- Block

  Der Name des Blocks, zu dem der Parameter gehört

- Unit

  Die physikalische Einheit des Parameters

Feld „Dimension“:

In diesem Feld wird die Zahl der Spalten und Reihen der Tabelle dargestellt - zudem kann diese hier geändert werden.

In Abb. 4-16 auf Seite 360 ist ein Kennfeld für die Kraftstoffdichte gezeigt. Wenn Sie dieses Kennfeld um eine Spalte erweitern wollen, erhöhen Sie im Feld „Dimension“ den x-Wert um „1“.

Es wird eine neue Spalte eingefügt mit einem um „1“ inkrementierten x-Wert - die y-Werte dieser neuen Spalte werden durch Extrapolation gewonnen.

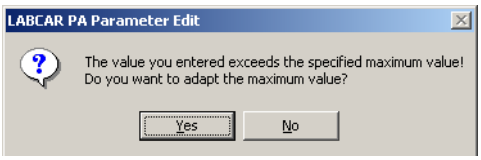
Möchten Sie einen neuen x-Wert „2200“, so wählen Sie aus dem Menü **Edit → Edit Axes** und ändern den Eintrag „2001.0“ auf „2200.0“. Die neuen y-Werte für diesen x-Wert müssen dann entsprechend gändert werden.

Dasselbe gilt für auch für das Hinzufügen von Spalten.

Wird Spalten/Reihenzahl verringert, wird jeweils die letzte Reihe/Spalte gelöscht.

Feld „Allowed Value Range“:

Hier werden die minimalen und maximalen Funktionswerte eingestellt. Wird eine dieser Schwellen über- bzw. unterschritten, so kommt es zu folgender Warnmeldung:



Wenn Sie **No** wählen, so wird die Änderung zurückgenommen - wählen Sie **Yes**, wird der jeweilige Minimal- oder Maximalwert der jetzigen Änderung angepasst.

Parameter History einsehen und editieren

Ein rechter Mausklick auf eine Zeile der Parameterliste öffnet ein Kontextmenü, das den Eintrag **View History** enthält. Wird dieser gewählt, öffnet sich das „Parameter History“ Dialogfenster.

Dieses Fenster enthält neben dem Namen und dem Block, zu dem der Parameter gehört, die Änderungsgeschichte des Parameters.

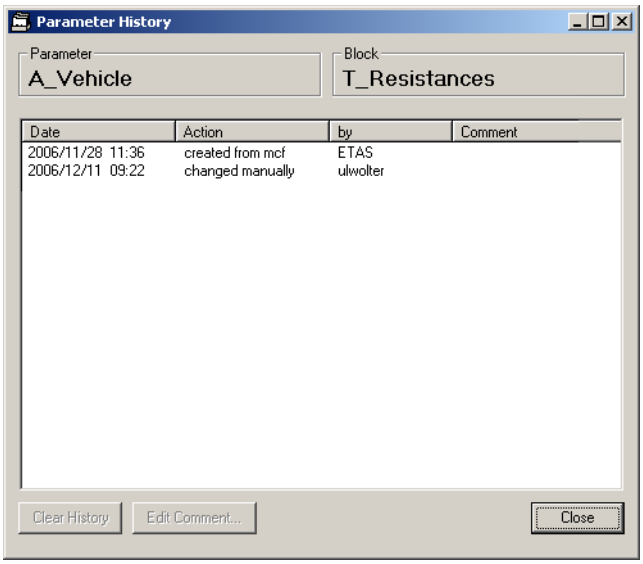
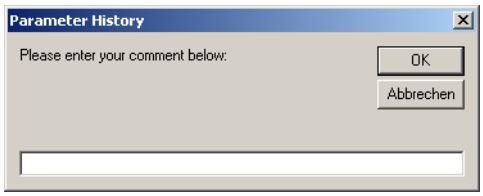


Abb. 4-17 Parameter History

Die Einträge können gelöscht werden durch Klicken der Schaltfläche **Clear History**. In den Optionen kann eingestellt werden, ob dabei alle Einträge gelöscht werden oder ob der jeweils letzte Eintrag erhalten wird (siehe Abschnitt auf Seite 356).

Wenn Sie einen Eintrag in der Liste mit einem Mausklick auswählen, wird die Schaltfäche **Edit Comment** aktiv.

Wenn Sie diese Schaltfläche anklicken, wird ein Dialogfenster geöffnet, in das Sie einen Kommentar zu der jeweiligen Änderung eingeben können.



4.7.8 Referenzdateien

Referenzdateien dienen zur Unterstützung bei der Parametrierung Ihres Modells. Sie können beispielsweise einen Satz „bewährter" Parameter enthalten, von dem aus Sie Änderungen vornehmen.

Wenn Sie eine Parametrierung als Referenz abspeichern wollen, wählen Sie **File → Set To Reference File**. In dem Dateiauswahlfenster wird Ihnen dann als Dateiname der Name der Datei plus die Zeichenkette „Reference_*N*" angeboten.

Um eine gespeicherte Referenzdatei zu laden, wählen Sie **File → Open Reference**. In dem Dateiauswahlfenster können Sie dann die gewünschte Datei wählen.

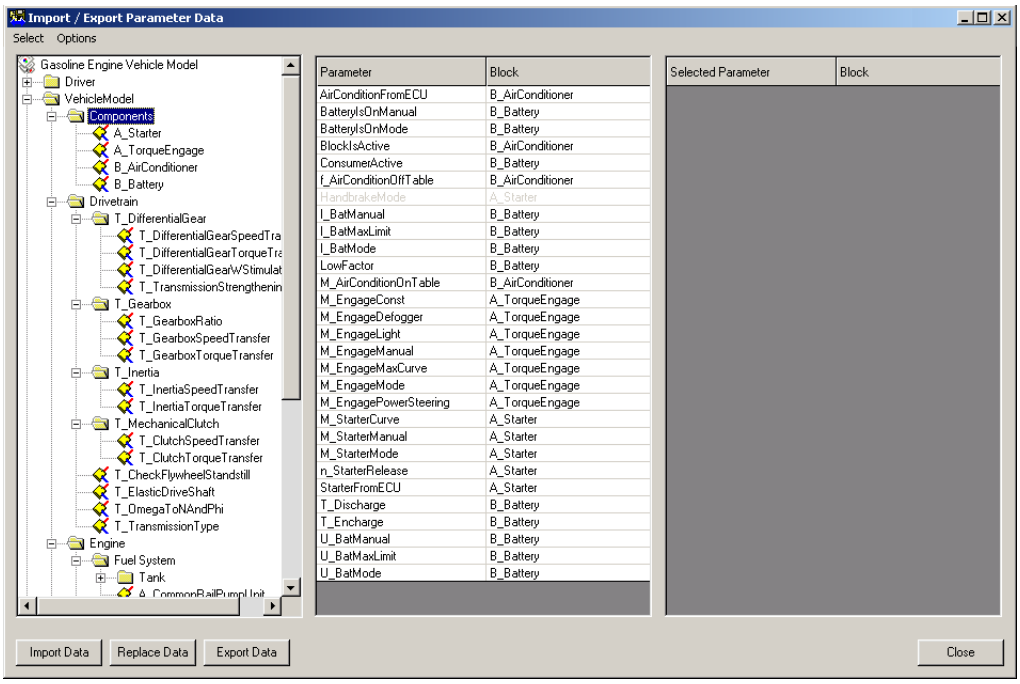
4.7.9 Parameter in Datei exportieren

In diesem Abschnitt wird der Export von Parametern in M- oder DCM-Dateien beschrieben.

Parameter in eine neue Datei exportieren („Export Data“)

- Um Parameter in m-, DCM- oder mpa-Dateien zu exportieren, wählen Sie **Import/Export → Import/Export Parameter Data**.

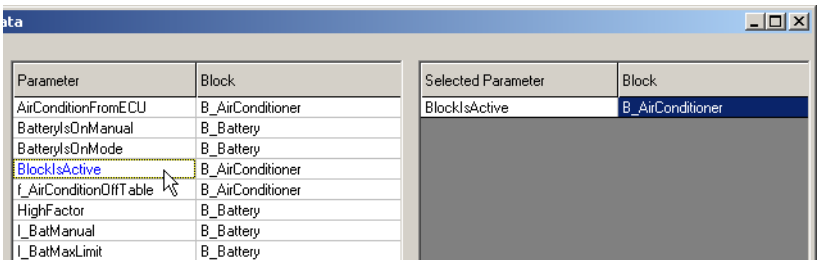
  Es wird dann das Dialogfenster „Import/Export Parameter Data“ geöffnet.



Im linken Teil des Fensters ist wieder die hierachische Modellansicht dargestellt. Im mittleren Teil sind alle Parameter des ausgewählten Modellteils (hier: GEVM → VehicleModel → Components) aufgelistet. Rechts werden diejenigen Parameter angezeigt, die zum Export ausgewählt wurden.

Um beispielsweise den Parameter „BlockIsActive“ des Blockes „B_AirConditioner“ zu exportieren, klicken Sie auf diesen Eintrag. Der ausgewählte Para-

meter wird dann unter „Selected Parameters“ eingetragen. Parameter, die dieser Liste hinzugefügt wurden, werden blau markiert.

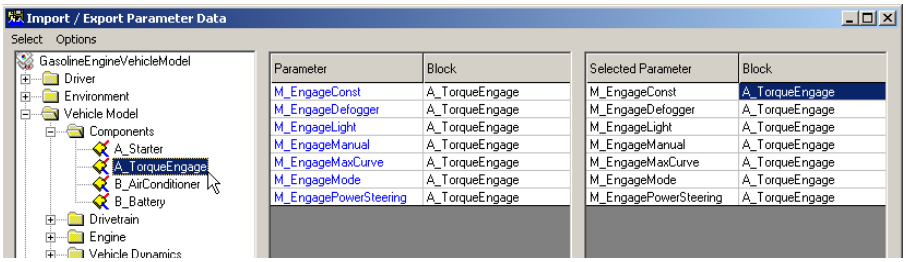


- Um die Auswahl eines Parameters wieder rückgängig zu machen, klicken Sie auf diesen Parameter unter „Selected Parameters“.
- Um alle Parameter eines ausgewählten Blockes oder Modellteils (z.B. des Blockes „A_TorqueEngage“) auszuwählen, aktivieren Sie diesen Block oder Modellteil).
- Wählen Sie **Select** → **Select All**.

oder

- Drücken Sie <STRG+A>.

  Alle Parameter werden dann der Liste „Selected Parameters“ hinzugefügt.



- Umgekehrt werden mit **Select** → **Deselect All** alle bis dato gewählten Parameter aus der Spalte „Selected Parameters“ gelöscht.
- Klicken auf die Schaltfläche **Export Data** öffnet ein Dateiauswahlfenster.
- Geben Sie einen Dateinamen an oder wählen Sie eine bereits vorhandene Datei aus.
- Wählen Sie den Dateityp.
- Klicken Sie **Speichern**.

  Die neue Datei wird angelegt.

  Wenn die Datei bereits vorhanden ist, werden Sie gefragt, ob Sie die vorhandene Datei ersetzen wollen.

Parameter in eine bereits existierende Datei exportieren („Replace Data“)

- Um Parameter in bereits vorhandene M- oder DCM-Dateien zu exportieren, gehen Sie vor wie oben beschrieben.
- Zum Export klicken Sie die Schaltfläche **Replace Data**.

  Wenn die gewählten Parameter bereits in der Datei vorhanden sind, werden diese Einträge überschrieben, andernfalls werden Sie in der Datei eingefügt.

Optionen für Parameterexport einstellen

Mit **Options → Im-/Export Attributes in Matlab M-Files** kann eingestellt werden, ob beim Export von Parametern ausschließlich der Parameter selbst in die Datei geschrieben wird (deaktiviert) oder zusätzlich weitere Attribute wie „Min.“, „Max.“ oder „Modification State“ etc. (aktiviert).

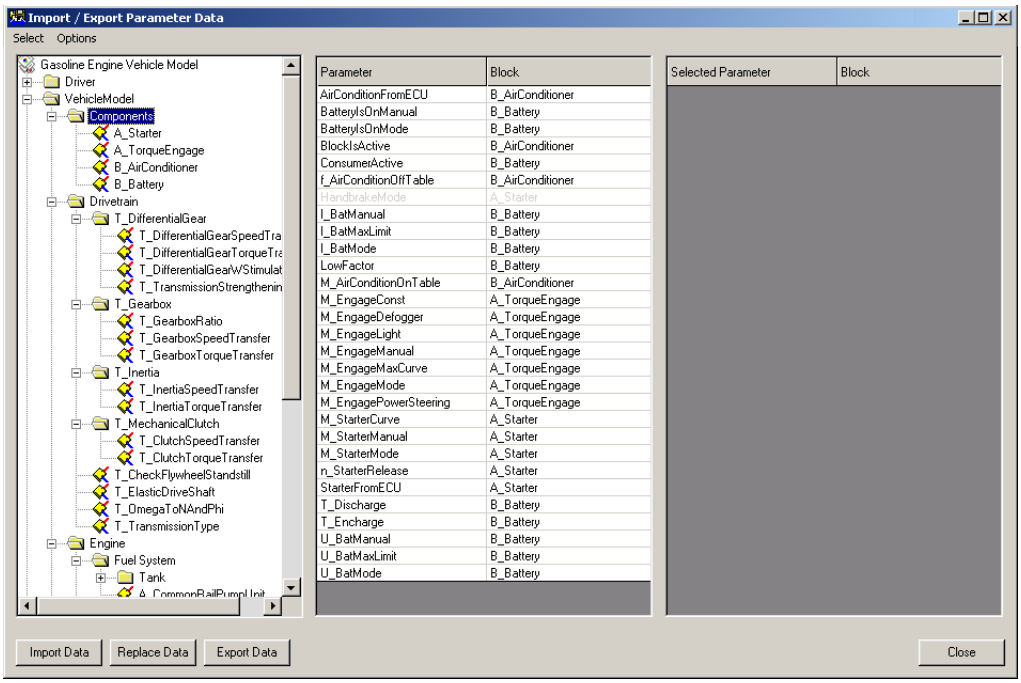
4.7.10 Parameter aus Datei importieren

Auf die selbe Art und Weise können Parameter aus einer Datei importiert werden. Voraussetzung dafür ist allerdings, dass der gewählte Parameter auch in der Datei vorhanden ist - ansonsten kommt es zu einer Fehlermeldung.

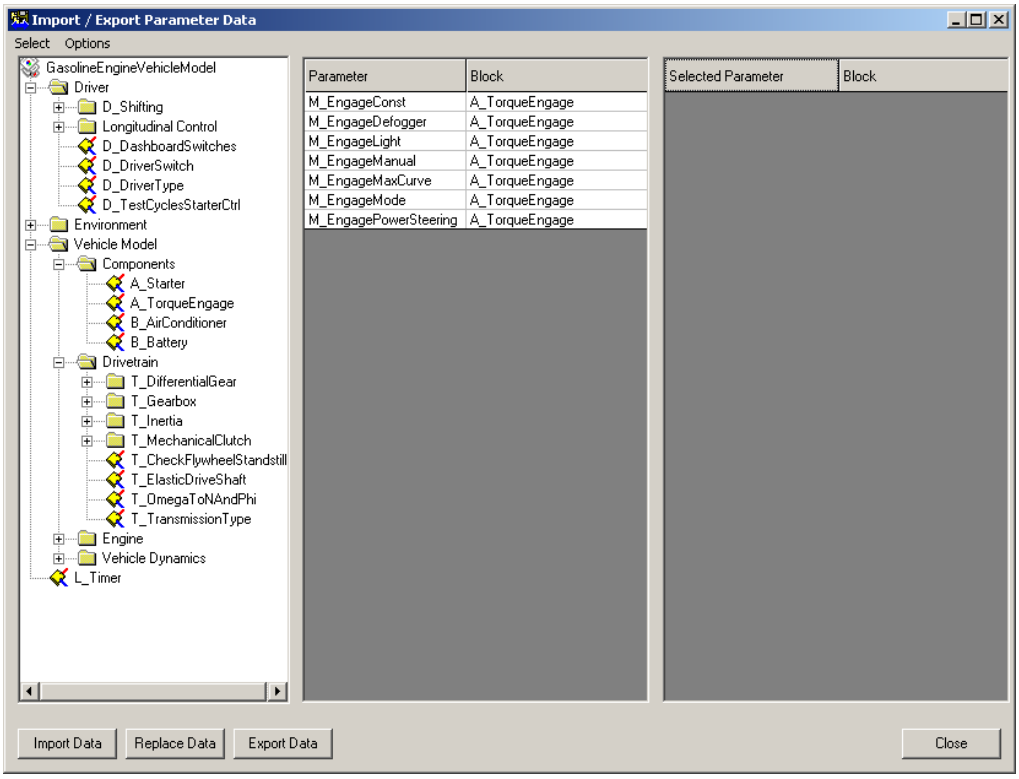
Zu importierende Parameter auswählen

- Um Parameter aus einer M- oder DCM-Datei zu exportieren, wählen Sie **Import/Export → Import/Export Parameter Data**.

  Es wird dann das Dialogfenster „Import/Export Parameter Data“ geöffnet.



- Über den Menüpunkt **Options → Im-/Export Attributes in Matlab M-File** haben Sie an dieser Stelle nochmal die Möglichkeit, die Import/Export-Optionen (siehe auf Seite 355) zu ändern.
- Um beispielsweise Parameter des Blockes „A_TorqueEngage“ zu importieren, klicken Sie auf diesen Eintrag.

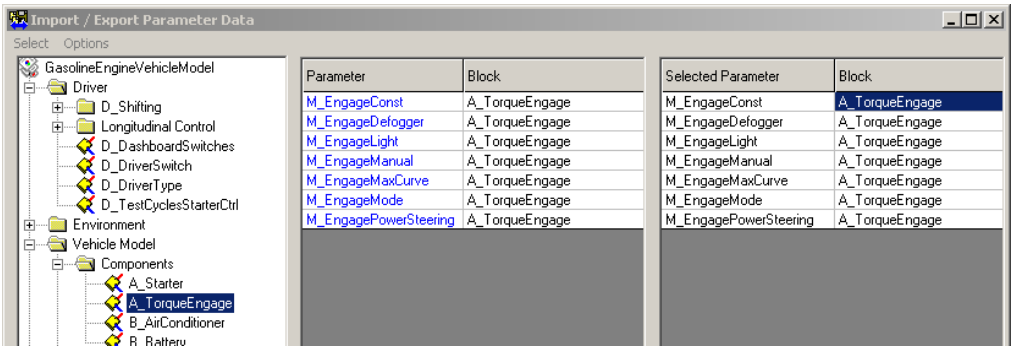


Die Parameter des Blockes werden in der mittleren Spalte des Fensters aufgelistet.

Parameter mit dem Scope „imported“ werden in der Liste grau dargestellt.

- Wählen Sie **Select** → **Select All**.

  Die ausgewählten Parameter (hier: alle Parameter des Blockes „A_TorqueEngage“) werden dann unter „Selected Parameters“ eingetragen. Parameter, die dieser Liste hinzugefügt wurden, werden blau markiert.



Datei auswählen

- Klicken Sie die Schaltfläche **Import Data**.

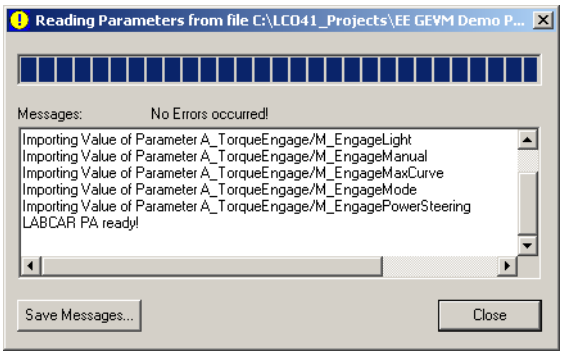
  Es öffnet sich ein Dateiauswahlfenster.
- Wählen Sie den Dateityp.
- Geben Sie einen Dateinamen an oder wählen Sie eine Datei aus.
- Klicken Sie **Öffnen**.

  Die Parameter werden importiert.

Beim Import werden die Werte der gewählten Parameter durch jene aus der importierten Datei ersetzt. Dabei sind zwei Szenarien denkbar:

- Der neue Parameterwert ist mit dem alten identisch.
- Der neue Parameterwert ist mit dem alten nicht identisch.

Dies wird in dem Status-Fenster „Import Parameters“ noch einmal angezeigt.

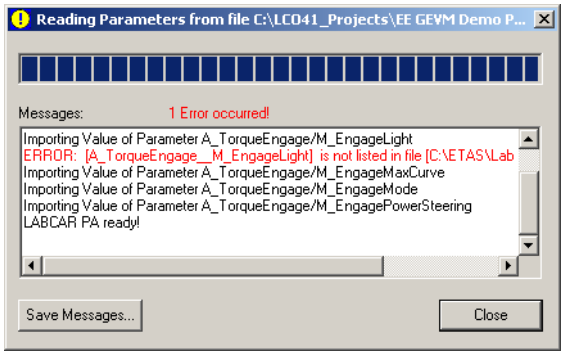


Hinweis

Da es sich hierbei um keine Fehlermeldungen handelt, bleibt das Fenster nach Beendigung des Imports nur geöffnet, wenn die entsprechende Option (siehe „Verhalten von Fenstern für Status Messages“ auf Seite 353) deaktiviert ist.

Fehlermeldungen

Wenn der ausgewählte Parameter in der Datei, aus der importiert werden soll, nicht vorhanden ist, oder wenn beim einlesen andere Fehler auftreten, wird eine ensprechende Fehlermeldung ausgegeben.



Diese Meldungen können auch in ASCII- oder RTF-Dateien abgespeichert werden.

Fehlermeldung abspeichern

- Klicken Sie die Schaltfläche **Save Errors**.
  Es wird ein Dateiauswahlfenster geöffnet.
- Wählen Sie den Dateityp (ASCII oder Rich Text Format)
- Geben Sie einen Dateinnamen an.
- Klicken Sie **Speichern**.

Hinweis

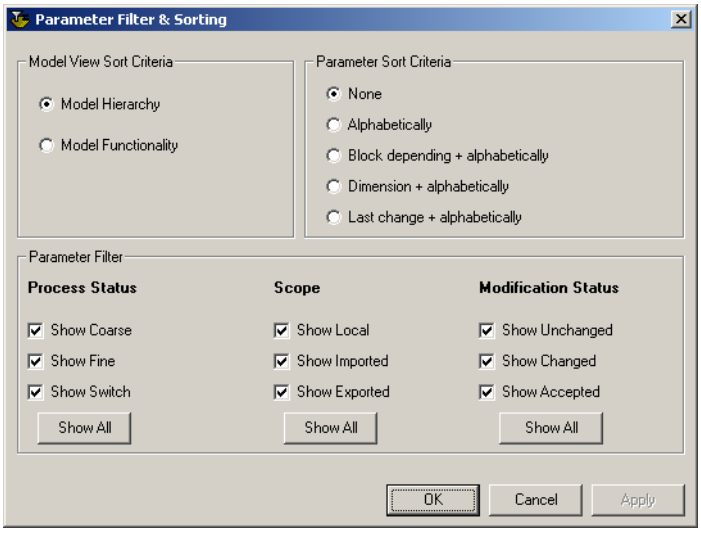
Bei einer RTF-Datei sind die jeweiligen Fehlermeldungen mit roter Schrift hervorgehoben, was bei einer ASCII-Datei nicht möglich ist.

4.7.11 Die Filter- und Sortierfunktion

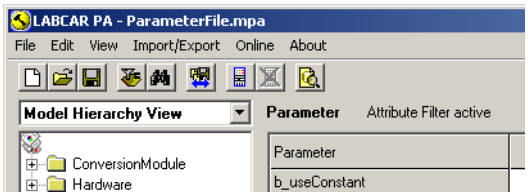
Über die in der Parameterliste angezeigten Parameter kann ein Filter gelegt werden, der beispielsweise dazu verwendet wird, nur alle Parameter mit dem Status „changed“ anzuzeigen oder solche mit dem Scope „exported“.

Zudem können die Strukturansicht des Modells und die Ordnungskriterien für die Parameter in der Liste festgelegt werden.

Die Funktion wird aufgerufen über **Edit → Filter Settings**.



Wenn ein Filter gesetzt ist, wird dies in der Bedienoberfläche oberhalb der Parameterliste angezeigt („Attribute Filter Active").



Die Bedeutung der einzelnen Filterkriterien wird im Folgenden beschrieben.

Model View Sort Criteria

Hier können Sie festlegen, wie das Modell im linken Teil der Benutzeroberfläche dargestellt wird.

- **Model Hierarchy**

  Diese Ansicht stellt das Modell in seiner hierarchischen Struktur dar.

- **Model Functionality**

  Diese Ansicht stellt das Modell nach seinen funktionalen Einheiten (z.B. Driver, Engine, usw.) gruppiert dar.

Diese Funktion ist identisch mit der Auswahl in dem Dropdown-Menü links oben in der Benutzeroberfläche.

Parameter Sort Criteria

Hier werden die Sortierkriterien für die Modellkomponenten und der im rechten Teil der Benutzeroberfläche aufgelisteten Parameter festgelegt.

- **None**

  Keine explizite Sortierung - die Parameter werden in der Reihenfolge aufgelistet, in der sie in den einzelnen Blöcken in der Modellansicht vorkommen (entspricht der Reihenfolge in der mpa-Datei).

- **Alphabetically**

  Alphabetische Sortierung

- **Block depending and alphabetically**

  Erstes Sortierkriterium ist der Blockname - die Parameter eines Blockes sind alphabetisch geordnet.

- **Dimension and alphabetically**

  Erstes Sortierkriterium ist die Dimension des Parameters (2D-Table, 1D-Table, Scalar, Array) - bei Parametern mit der selben Dimension ist die Sortierung alphabetisch.

- **Last change and alphabetically**

  Erstes Sortierkriterium ist das Änderungsdatum, zweites Sortierkriterium ist der Parametername (alphabetisch).

Parameter Filter

Hier können Parameter nach bestimmten Kriterien in der Liste eingeblendet werden.

Process Status

- Show Coarse

  Alle Parameter mit dem Parametrisierstatus „coarse“ (Grundparametrisierung) werden angezeigt.

- Show Fine

  Alle Parameter mit dem Parametrisierstatus „fine“ (Feinparametrisierung) werden angezeigt.

- Show Switch

  Alle Parameter mit dem Parametrisierstatus „switch“ (Schalter) werden angezeigt.

Mit der Schaltfläche **Show All** werden alle drei Filteroptionen auf einmal aktiviert, was einer Deaktivierung des Filters entspricht.

Scope

- Show Local

  Alle Parameter mit dem Scope „local“ werden angezeigt.

- Show Imported

  Alle Parameter mit dem Scope „imported“ werden angezeigt.

- Show Exported

  Alle Parameter mit dem Scope „exported“ werden angezeigt.

Mit der Schaltfläche **Show All** werden alle drei Filteroptionen auf einmal aktiviert, was einer Deaktivierung des Filters entspricht.

Modification Status

- Show Unchanged

  Alle Parameter mit dem Status „unchanged“ werden angezeigt.

- Show Changed

  Alle Parameter mit dem Status „changed“ werden angezeigt.

- Show Accepted

  Alle Parameter mit dem Status „accepted“ werden angezeigt.

Mit der Schaltfläche **Show All** werden alle drei Filteroptionen auf einmal aktiviert, was einer Deaktivierung des Filters entspricht.

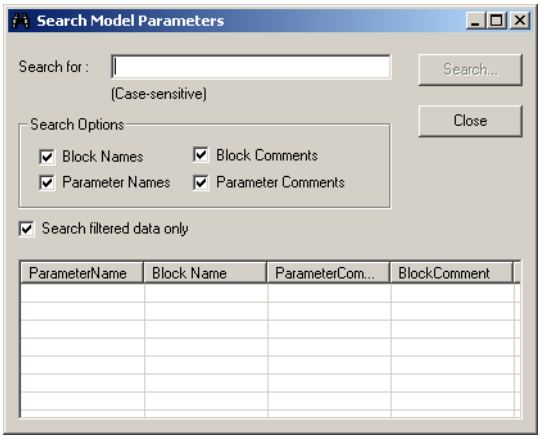
Hinweis

Es macht keinen Sinn, alle drei Filteroptionen innerhalb einer Gruppe zu deaktivieren: dies würde zu einer leeren Liste führen und die Kriterien der beiden andernen Gruppen wären wirkungslos.
Deshalb hat die Deaktivierung aller drei Filteroptionen eines Kriteriums keine Wirkung, und es werden alle Parameter angezeigt.

4.7.12 Die Suchfunktion für Modellparameter

Mit der Suchfunktion können Parameter im Modell mit verschiedenen Optionen gesucht werden.

Die Funktion wird aufgerufen über **Edit → Search Model Parameters** oder mit <STRG+F>,



Gesucht wird nach einem String, der in bestimmten Attributen des Parameters vorkommt. Diese Attribute sind:

Search Options

- **Block Names**

  Der String muss im Attibut „Block Name“ enthalten sein.
- **Block Comments**

  Der String muss im Attribute „Block Comment“ enthalten sein (derselbe String, der im Tooltip angezeigt wird, wenn der Mauszeiger auf einen Block zeigt).
- **Parameter Names**

  Der String muss im Parameternamen enthalten sein.
- **Parameter Comments**

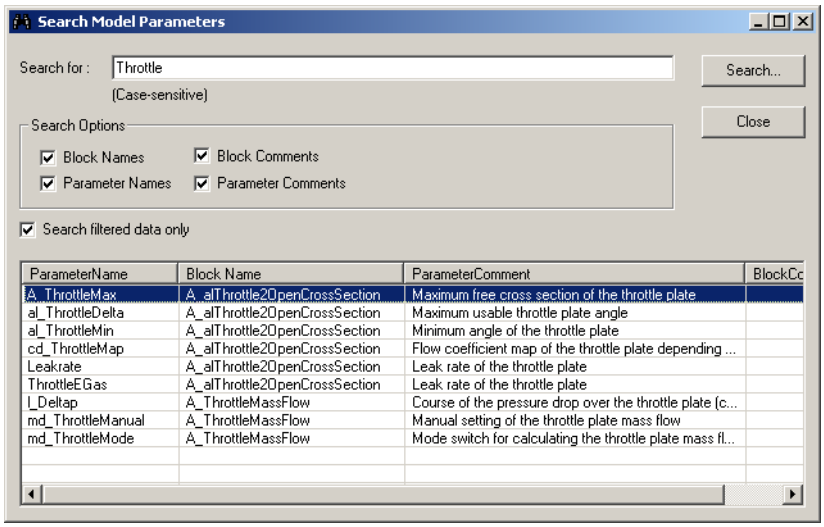
  Der String muss im Attribut „Comment“ enthalten sein.

Search filtered data only

Wenn ein Filter über die Parameterliste gelegt ist, bewirkt diese Option, dass die Suchkriterien lediglich auf die angezeigten Parameter angewendet wird.

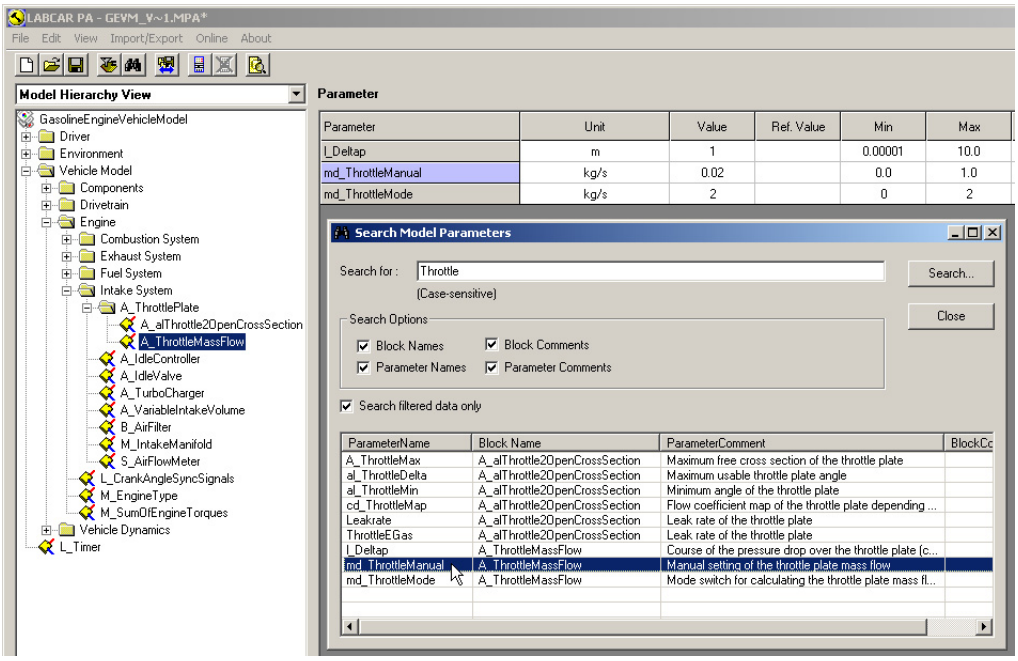
Beispiel „Throttle“

In diesem Besipiel wird nach dem String „Throttle“ gesucht - alle Kriterien sind aktiv.



Würde der String beispielsweise nur in den Parameternamen gesucht, würde die Parameter „Leakrate“ und „l_Deltap“ nicht in der Ergebnisliste erscheinen.

Wenn Sie einen Parameter in der Ergebnisliste anklicken, wird der Block in der Strukturansicht ausgewählt. In der Parameterliste erscheinen dann alle Parameter dieses Blockes - der in der Suchliste ausgewählte wird blau hinterlegt.



4.7.13 Parameter Scope Binding ermitteln

Da der Wert importierter Parameter nur in dem Block geändert werden kann, aus dem er exportiert wurde, gibt es die Möglichkeit, eine Liste von exportierten und importierten Parametern zu erzeugen.

Die Funktion wird aufgerufen über **View → Parameter Scope Binding**.

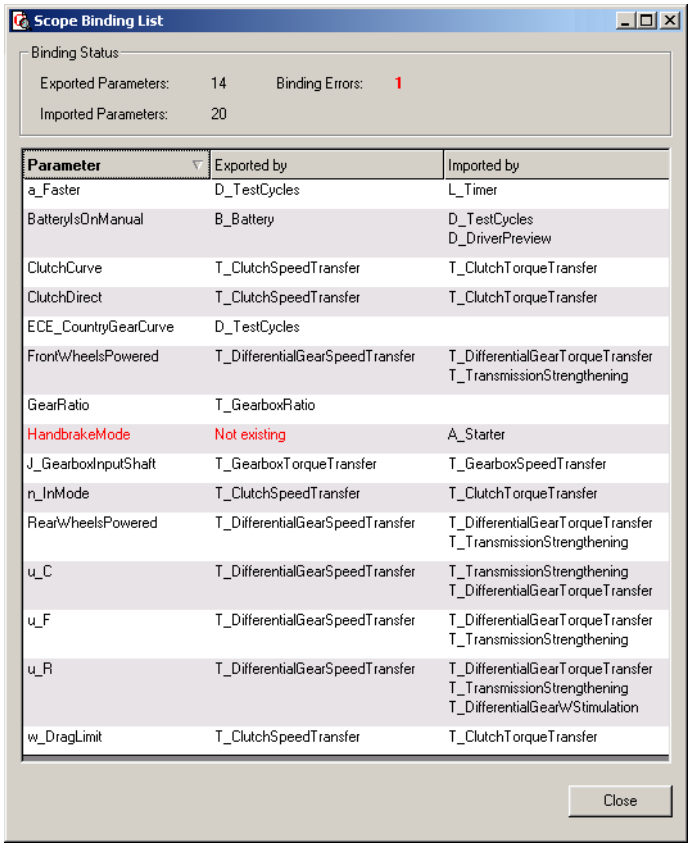


Abb. 4-18 Scope Binding List

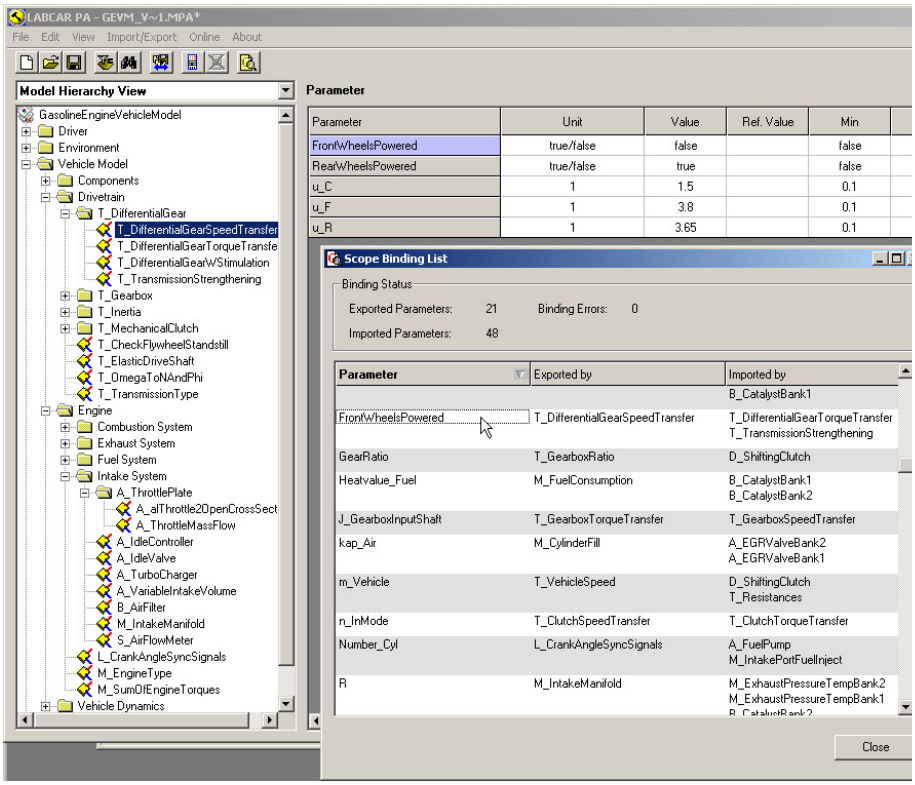
In der Spalte „Parameter“ sind alle Parameter aufgeführt, die aus einem Block exportiert werden (Spalte „Exported by“) und in andere importiert werden (Spalte „Imported by“).

Die Parameter sind alphabetisch auf- oder absteigend sortierbar (MausKlick auf die Spaltenüberschrift „Parameter“).

Die Liste in Abb. 4-18 auf Seite 374 zeigt zusätzlich, wenn es beim „Scope Binding“ Inkonsistenzen gibt: Der Parameter „Handbrake Mode“ wird vom Block „A_Starter“ importiert, aber aus keinem Block exportiert. Er wird dann rot dargestellt.

Inkonsistenzen dieser Art werden im Übrigen schon beim Öffnen der Datei überprüft und dem Anwender mitgeteilt.

Mit einem Klick auf den Parameternamen oder auf den Namen des exportierenden Blockes wird der Block in der Strukturansicht ausgewählt. In der Parameterliste erscheinen dann alle Parameter dieses Blockes - der exportierte Parameter wird blau hinterlegt.



Entsprechendes gilt beim Klicken auf den Namen des importierenden Blockes.

4.7.14 Online-Parametrierung

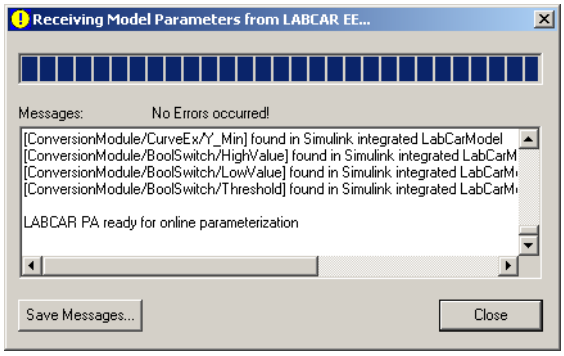
Bei der Online-Parametrierung werden in LABCAR-PA V1.0 geänderte Parameter direkt in ein in ETAS EE laufendes Experiment geschrieben.

Online-Parametrierung starten

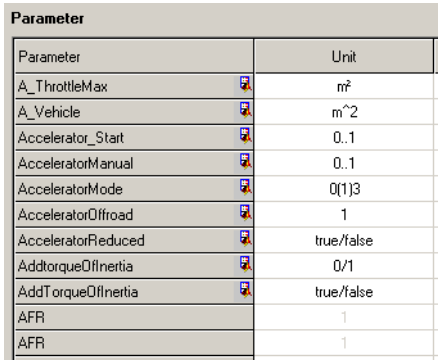
- Starten Sie das Experiment in ETAS EE.
- Öffnen Sie im LABCAR-PA die Parameterdatei (*.mpa) des Modells.

- Wählen Sie **Online** → **Go Online**.

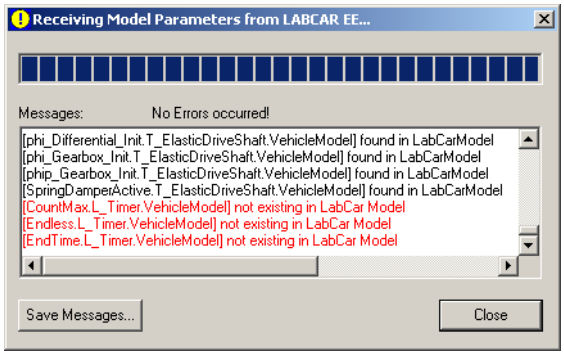
  Es wird eine Verbindung zur Experimentierumgebung aufgebaut und im laufenden Projekt nach allen Parametern gesucht, die in der aktuellen Parameterdatei vorkommen.



  Die gefundenen Parameter werden in der Parameterliste mit dem „Online"-Icon versehen.



  Zudem werden Fehlermeldungen ausgegeben, wenn ein Parameter nicht gefunden wurde.



- Wählen Sie **Close**.

  Sie können nun Parameter des laufenden Modells in der Parameterliste von LABCAR-PA ändern.

Fehler: Parameter wurde nicht gefunden

Wenn ein Parameter aus der Parameterdatei nicht im laufenden Modell gefunden wird, kann dies zwei Gründe haben:

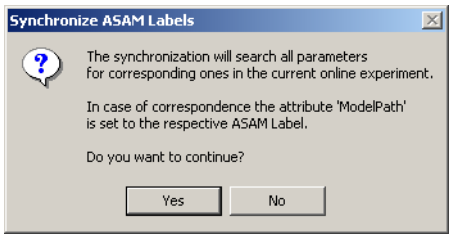
- Die ASAM-Labels stimmen nicht überein (etwa aufgrund anderer Benennung nach Simulink-Import).

  Dies kann behoben werden durch eine Synchronisation der Labels (siehe weiter unten).
- Der Parameter existiert nicht im laufenden Modell

ASAM-Labels synchronisieren

- Wählen Sie **Online → Synchronize ASAM Labels**.

  Das laufende Experiment wird nun nach ASAM-Labels durchsucht, die zu Parametern der Parameterdatei passen.

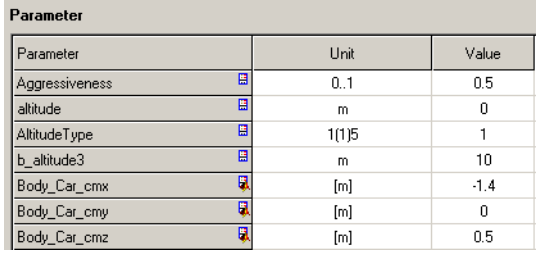


Hinweis

Die Synchronisation kann nur in den Fällen erfolgreich sein, wenn die ASAM-Labels des laufenden Modells (etwa nach einem Simulink-Import) nach den in Abschnitt „Optionen für Parameterimport und -export“ auf Seite 355 beschriebenen Regeln für die Erzeugung von ASAM-Labeln erzeugt wurden.

Falls eine Übereinstimmung gefunden wird, erhält das ASAM-Label in der Parameterdatei (Attribut „ModelPath“) den im laufenden Modell verwendeten Namen.

Wenn die ASAM-Labels aus einem importierten Simulink-Modell stammen, werden diese synchronisierten ASAM-Labels in der Parameterliste mit einem modifizierten „Online“-Icon versehen.



4.8 LABCAR-CCI V5.4.1 (Calibration Connector for INCA)

LABCAR-CCI V5.4.1 ist ein Add-On zu LABCAR-OPERATOR V5.4.1, das den Zugriff auf INCA-Geräte wie ETK ermöglicht. Insbesondere können damit Teile der INCA-Funktionalität von LABCAR-OPERATOR aus gesteuert werden. Die ASAM-Labels des INCA-Experiments stehen in der Experimentierumgebung (ETAS EE) im Fenster „Workspace Elements“ zur Verfügung.

Die folgende Abbildung verdeutlicht die Integration des LABCAR-CCI V5.4.1 in LABCAR-OPERATOR V5.4.1.

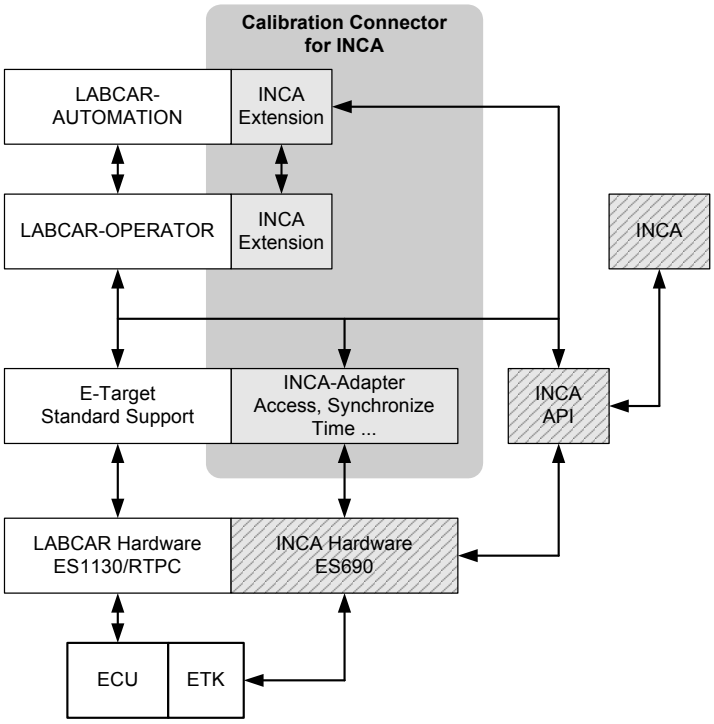


Abb. 4-19 Die Integration von LABCAR-CCI V5.4.1 in LABCAR-OPERATOR

LABCAR-CCI V5.4.1 besteht aus drei Hauptkomponenten:

- Die INCA-Erweiterung in LABCAR-OPERATOR V5.4.1 zum Messen und Verstellen mit INCA-Geräten.

  Diese enthält auch weitere Funktionalität wie das Öffnen einer bestimmten INCA-Datenbank, Initialisieren von INCA-Geräten, Starten und Anhalten einer INCA-Messung.
- Die INCA-Erweiterung in LABCAR-AUTOMATION stellt eine Anpassung an die neue Architektur dar, insbesondere bezüglich Zeitsynchronisation und E-Target.
- Die INCA-Erweiterung zur Zugriffsverwaltung auf INCA-Geräte und zur Synchronisierung der Zeitstempel von E-Target einerseits und INCA-Messwerten andererseits.

In diesem Kapitel finden Sie Informationen zu folgenden Themen:

- „Systemanforderungen“ auf Seite 379

  In diesem Abschnitt werden die Software- und Hardwareanforderungen für den Einsatz von LABCAR-CCI V5.4.1 beschrieben.

- „Arbeiten mit LABCAR-CCI V5.4.1“ auf Seite 379

  In diesem Abschnitt finden Sie Informationen zum Arbeiten mit LABCAR-CCI.

4.8.1 Systemanforderungen

In diesem Abschnitt werden die für den Einsatz von LABCAR-CCI V5.4.1 erforderlichen Soft- und Hardwarekomponenten beschrieben.

INCA Version

Die aktuelle Liste der für den Betrieb von LABCAR-CCI V5.4.1 erforderlichen INCA-Versionen finden Sie im Menü **?** → **Help** im Dokument „LABCAR-OPERATOR 5.x.y - Software Compatibility List“.

Hinweis

INCA selbst ist nicht Bestandteil von LABCAR-CCI V5.4.1. Die INCA-Lizenz muss separat erworden werden. Ohne die Installation von INCA kann LABCAR-CCI V5.4.1 nicht verwendet werden.

INCA kann auf demselben Computer installiert werden wie LABCAR-OPERATOR.

Hardware

Der Zugriff auf das Steuergerät wird im INCA-Experiment spezifiziert und kann beispielsweise über die CAN-, K-Line- oder ETK-Schnittstelle eines ES690 Kompaktmoduls erfolgen.

4.8.2 Arbeiten mit LABCAR-CCI V5.4.1

Der Zugriff auf ein bestimmtes INCA-Experiment erfolgt wie bei LABCAR-OPERATOR-Experimenten in der Experimentierumgebung ETAS EE.

Gehen Sie dazu wie folgt vor:

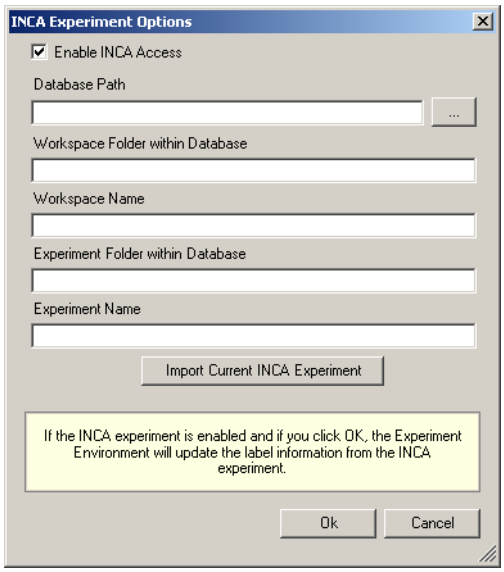
Das INCA-Experiment spezifizieren

- Starten Sie ETAS EE.
- Laden Sie ggf. Ihr LABCAR-OPERATOR-Experiment zum Target herunter und starten Sie (jetzt oder später) die Simulation.
- Wählen Sie **Experiment** → **INCA Options**.

  Das Fenster „INCA Experiment Options“ wird geöffnet.

- Aktivieren Sie die Option „Enable INCA Access“.

  Die verschiedenen Felder zur Spezifikation weiterer Daten werden aktiviert.



- Sie können nun entweder die notwendigen Daten per Hand eingeben:
  - Database Path: Pfad zur INCA-Datenbank
  - Workspace Folder within Database
  - Workspace Name
  - Experiment Folder within Database
  - Experiment Name

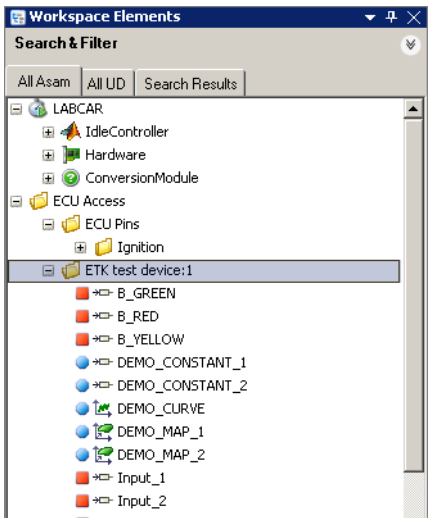
oder

- Öffnen Sie das Experiment in INCA und klicken Sie **Import Current INCA Experiment**.

  Die Daten des Experiments werden in die entsprechenden Felder (s.o.) übernommen.

- Klicken Sie **OK**.

  Um die Messgrößen/Parameter Ihres INCA-Experimentes zu ermitteln, wird jetzt automatisch ein „Connect“ zum Experiment durchgeführt.



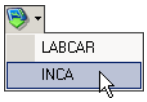
Wenn Sie anschließend das Experiment in ETAS EE speichern (**File** → **Save Experiment**), wird die Verknüpfung zu dem weiter oben spezifizierte INCA-Experiment ebenfalls gespeichert.

Das INCA-Experiment verbinden

Um das INCA-Experiment erneut zu verbinden, gehen Sie wie folgt vor:

- Starten Sie die Messung mit **Experiment** → **Download** → **INCA**.

oder



- Klicken Sie das entsprechende Symbol in der Werkzeugleiste.

  Das Experiment wird in INCA geöffnet und die ASAM-Labels des INCA-Experimentes sind in ETAS EE zugänglich.

INCA-Hardware initialisieren

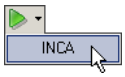
Wenn Ihre INCA-Hardware nach einem Download nicht korrekt initialisiert ist (z.B. weil die Hardware oder das Steuergerät ausgeschaltet waren), gehen Sie wie folgt vor:

- Zur Initialisierung der INCA-Hardware wählen Sie **Experiment** → **Initialize INCA Hardware**.

Messung starten

- Starten Sie die Messung mit **Experiment** → **Start Measurement** → **INCA**.

oder



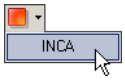
- Klicken Sie das entsprechende Symbol in der Werkzeugleiste.

  Die Messung wird gestartet.

Messung stoppen

- Stoppen Sie die Messung mit **Experiment** → **Stop Measurement** → **INCA**.

oder



- Klicken Sie das entsprechende Symbol in der Werkzeugleiste.

  Die Messung wird gestoppt.

Die Verbindung zum INCA-Experiment unterbrechen

- Um die Verbindung zum Experiment zu unterbrechen, wählen Sie **Experiment** → **Stop Measurement** → **INCA**.

oder

- Klicken Sie das entsprechende Symbol in der Werkzeugleiste.

  Das INCA-Experiment wird geschlossen .

5 Appendix

Die von LABCAR-NIF V5.4.1 und LABCAR-LCX V5.4.1 verwendeten Bibliotheken "StringTemplate" und "ANTLR" unterliegen folgenden Lizenzen:

StringTemplate Software-Lizenz

[The BSD License]

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:
```

6 ETAS Kontaktinformation

ETAS Hauptsitz

ETAS GmbH

Borsigstraße 14 Telefon: +49 711 3423-0

70469 Stuttgart Telefax: +49 711 3423-2106

Deutschland WWW: www.etas.com

ETAS Regionalgesellschaften und Technischer Support

Informationen zu Ihrem lokalen Vertrieb und zu Ihrem lokalen Technischen Support bzw. den Produkt-Hotlines finden Sie im Internet:

ETAS Regionalgesellschaften WWW: www.etas.com/de/contact.php

ETAS Technischer Support WWW: www.etas.com/de/hotlines.php

Abbildungsverzeichnis

Index