ETAS

LABCAR-OPERATOR V5.4.0

ユーザーズガイド

DRIVING | EMBEDDED EXCELLENCE

著作権について

V5.4.0 R01 JP - 03.2016

目次

1 はじめに

本書は、自動車用 ECU の開発とテストに従事されているユーザーの方を対象としています。本書をお読みになる際は、計測、および ECU テクノロジーの分野に関する専門知識が必要です。

1.1 本書について

本書は、LABCAR-OPERATOR V5.4.0 の操作方法についての解説書です。

本章以降の部分は、主に以下の章で構成されています。

- **第 2 章 「LABCAR-IP ユーザーインターフェース」（12 ページ）**

  本章では、LABCAR-IP ユーザーインターフェースについて説明します。

  — 機能概要（12 ページ）
  — LABCAR-IP のメインメニュー（13 ページ）
  — ツールバー（15 ページ）
  — メインワークスペース（17 ページ）
  — プロジェクトエクスプローラ（"Project Explorer"）（18 ページ）
  — ログウィンドウ（18 ページ）
  — プロジェクトのプロパティ設定（18 ページ）

- **第 3 章 「LABCAR-IP の操作方法」（22 ページ）**

  本章では、LABCAR-OPERATOR プロジェクトにさまざまなタイプのモジュールを統合し、LABCAR-EE で実行される実験用のコードを生成する方法について説明します。

  — モジュール式シミュレーションコード（24 ページ）
  — Matlab/Simulink モデル（35 ページ）
  — ASCET モジュール（48 ページ）
  — C コードモジュール（55 ページ）
  — CAN モジュール（LABCAR-NIC — CAN ネットワークの統合）（85 ページ）
  — LIN モジュール（LABCAR-NIL — LIN ネットワークの統合）（132 ページ）
  — FlexRay モジュール（LABCAR-NIF V5.4.0 — FlexRay ネットワークの統合）（162 ページ）
  — FiL モジュール（183 ページ）
  — リアルタイムプラグイン（196 ページ）
  — 信号変換モジュール（209 ページ）
  — RTIO エディタによるハードウェア設定（214 ページ）
  — コネクションマネージャ（215 ページ）
  — リアルタイムオレーティングシステムの設定（OS コンフィギュレーション）（227 ページ）

- **第 4 章 「LABCAR-EE の概要」（248 ページ）**

  LABCAR-EE（Experiment Environment：実験環境）は、LABCAR-OPERATOR の実験を実行するための環境です。本章では LABCAR-EE の機能と GUI（Graphic User Interface）について概説します。

  — GUI の構成（249 ページ）
  — エクスペリメントエクスプローラ（"Experiment Explorer" ウィンドウ）（252 ページ）

- "Workspace Elements" ウィンドウ（254 ページ）
- メインワークスペース（260 ページ）
- パラメータの使用法（287 ページ）
- LABCAR-PA 1.0 を用いたパラメータファイルの編集（305 ページ）
- LABCAR-CCI V5.4.0（"Calibration Connector for INCA" ― INCA 用適合コネクタ）（341 ページ）

1.2 本書の使用法

1.2.1 表現について

ユーザーが実行するすべてのアクションは、いわゆる "Use-Case" 形式で記述されています。つまり以下に示すように、操作を行う目標がタイトルとして最初に簡潔に定義され、その下に、その目標を実現するために必要な操作手順が列挙されています。

目標の定義：

前置き ...

- 手順 1
  手順 1 についての説明 ...
- 手順 2
  手順 2 についての説明 ...
- 手順 3
  手順 3 についての説明 ...

まとめ ...

具体例：

新しいファイルを作成する：

新しいファイルを作成する際は、他のファイルをすべて閉じておきます。

- **File → New** コマンドを選択します。
  "Create file" ダイアログボックスが開きます。
- 新しいファイルの名前を、"File name" フィールドに入力します。
  ファイル名は 8 文字以内でなければなりません。
- **OK** をクリックします。

新しいファイルが作成され、ユーザーが指定した名前で保存されます。このファイルを使用して以後の操作を行います。

1.2.2 表記上の規則

本書は以下の規則に従って表記されています。

表記例	説明
File → Exit コマンドを選択して、…	メニューコマンドは、**青の太字**で表記します。
OK をクリックして、…	ユーザーインターフェース上のボタン名は、**青の太字**で表記します。
<Ctrl> を押して、…	キーボードの各キーは、**< >** で囲んで表記します。
"Open File" ダイアログボックスが 開きます。	プログラムウィンドウ、ダイアログボックス、入力フィールド等のタイトルは、" " で囲んで表記します。
`setup.exe` ファイルを選択します。	リストボックス、プログラムコード、ファイル名、パス名等のテキスト文字列は、`Courier` フォントで表記します。

その他、重要な語や新出の語は**太字**または*斜体*、または「 」で囲んで示されていて、特に重要な注意事項は、以下のように表記されています。

注記

ユーザー向けの重要な注意事項

また PDF 文書において、索引、および他の部分を参照する個所（例：「xxxx」を参照してください」の「xxxx」の部分）については、その参照先へのリンクが設けられているので、必要な参照個所を素早く見つけることができます。

2 LABCAR-IP ユーザーインターフェース

本章では、LABCAR-IP ユーザーインターフェースについて説明します。

本章は以下の項に分かれています。

2.1 機能概要

LABCAR-IP を起動すると、以下のようなユーザーインターフェースウィンドウ（メインウィンドウ）が開きます。

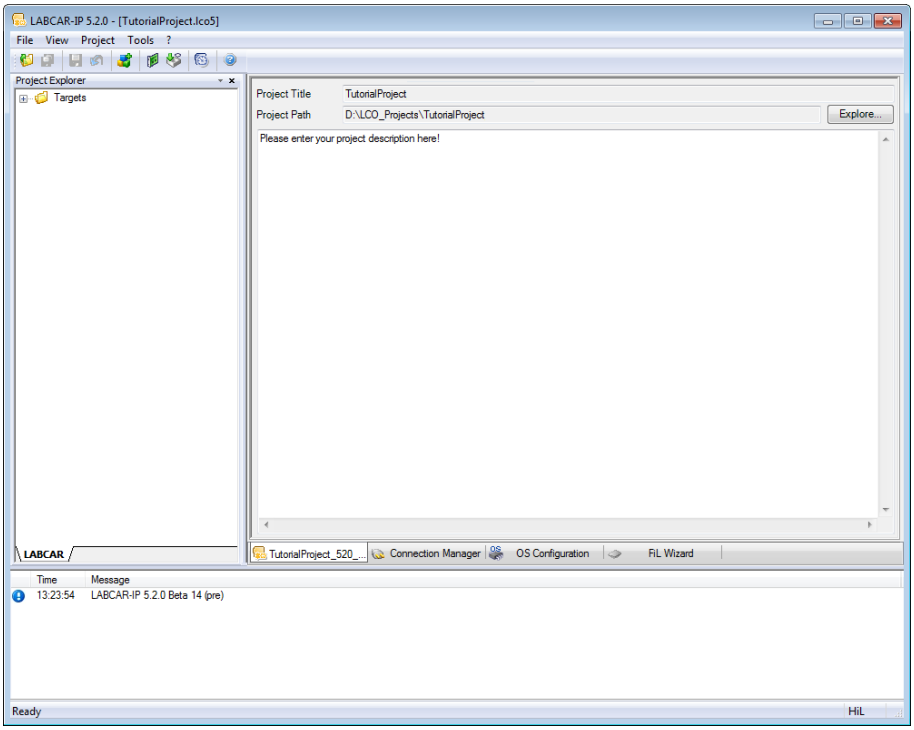


図 2-1 LABCAR-IP ユーザーインターフェース（プロジェクトを開いた状態）

このユーザーインターフェースには以下の操作エレメントが含まれています。

- メニューバー

  LABCAR-IP のメニュー機能についての説明は、13 ページの「LABCAR-IP のメインメニュー」という項を参照してください。

- ツールバー

  ツールバー内の各エレメントについての説明は、15 ページの「ツールバー」という項を参照してください。

- メインワークスペース

  複数のタブで構成され、各タブは、プロジェクト情報、コネクションマネージャ、CAN エディタ、OS コンフィギュレータ、およびシグナルジェネレータセット用に使用されます。
- プロジェクトエクスプローラ

  18 ページの「プロジェクトエクスプローラ（"Project Explorer"）」を参照してください。
- ログウィンドウ

  18 ページの「ログウィンドウ」を参照してください。

2.2 LABCAR-IP のメインメニュー

LABCAR-IP のメニューバーには以下のメインメニューが含まれています。

- "File" メニュー（13 ページ）

  このメニューには、ファイル固有のアクションを実行するためのコマンドが含まれています。
- "View" メニュー（14 ページ）

  このメニューで、個々のドッキングウィンドウの表示／非表示の制御を行います。
- "Project" メニュー（14 ページ）

  このメニューには、プロジェクトの編集や管理、そして実験の実行に必要なコマンドが含まれています。
- "Tools" メニュー（15 ページ）

  マクロレコーダ操作用のメニューです。
- "Help" メニュー（15 ページ）

  このメニューには、LABCAR-IP についての情報を表示するコマンドが含まれています。

注記

本章ではメニュー構成と各メニューコマンドの概要のみを紹介します。各コマンドの具体的な機能についての詳細は、関連する各章を参照してください。

2.2.1 "File" メニュー

このメニューには、ファイル固有のアクションを実行するためのコマンドが含まれています。

- **File → New Project**

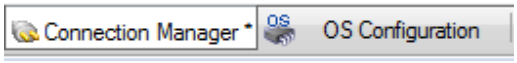
  新しいプロジェクトを作成します。
- **File → Open**

  既存のプロジェクトを開きます。
- **File → Close**

  現在開いているプロジェクトを閉じます。
- **File → Save All**

  プロジェクト全体（CAN エディタ、コネクションマネージャ等の変更内容をすべて含む）を保存します。

- **File → Save As**
  プロジェクト全体を別名で保存します。
- **File → Save**
  現在アクティブになっているドキュメント（CAN エディタ、コネクションマネージャなど）を保存します。タブのタイトルの右端にアスタリスクが表示されている場合、そのタブ上の変更内容がまだ保存されていないことを示しています。



- **File → Discard**
  現在アクティブになっているドキュメントについて、まだ保存されていない変更内容をすべて破棄します（**File → Save** を参照）。
- **File → Recent Projects**
  最近開いた 4 つのプロジェクトファイルを一覧表示します。
- **File → Exit**
  LABCAR-IP を終了します。

2.2.2 "View" メニュー

このメニューで、個々のドッキングウィンドウの表示／非表示の制御を行います。

- **View → Status Bar**
  メインウィンドウ最下部のステータスバーの表示／非表示を切り替えます。
- **View → Tool Bar**
  ツールバーの表示／非表示を切り替えます。
- **View → Project Explorer**
  プロジェクトエクスプローラの表示／非表示を切り替えます。
- **View → Log Window**
  "Log Window" の表示／非表示を切り替えます。

2.2.3 "Project" メニュー

このメニューには、プロジェクトの編集や管理、そして実験の実行に必要なコマンドが含まれています。

- **Project → Add Module**
  モジュール追加ウィザード（"Add Module Wizard"）を開きます。
- **Project → RTIO Editor**
  RTIO エディタを開き、接続されているハードウェアのコンフィギュレーション設定を行います。
- **Project → Build**
  コード生成用ダイアログボックスを開きます。
- **Project → Options**
  プロジェクト設定を行うためのダイアログボックスを開きます。
- **Project → ECU Pin List**
  "ECU Pin & HW Pin Properties" ダイアログボックスを開きます。

- **Project → OS Configuration**
  "OS Configurator" タブを開きます。

2.2.4 "Tools" メニュー

マクロレコーダ操作用のメニューです。

- **Tools → HSP Update Tool**
  最新バージョンの LABCAR-FWU を起動します。
- **Tools → Open LABCAR-IP Log File**
  詳細な情報が記録されたログファイルを開きます。
- **Tools → Zip Log Files**
  詳細な情報が記録されたログファイルを開きます。
- **Tools → Zip And Go**
  カレントプロジェクトを ZIP ファイルに圧縮します。
- **Tools → Open Experiment Environment**
  実験環境 LABCAR-EE を開きます。
- **Tools → View Version Info**
  すべてのソフトウェアコンポーネントのバージョンリストを表示します。
- **Tools → Options**
  LABCAR ID を編集するためのダイアログボックスを開きます。

2.2.5 "Help" メニュー

このメニューには、LABCAR-IP についての情報を表示するコマンドが含まれています。

- **? → Help**
  マニュアルやその他の情報へのリンクが設定されている HTML ファイルを開きます。
- **? → About**
  現在インストールされている LABCAR ソフトウェアとそのバージョンについての情報を表示します。
- **? → Contact**
  ETAS のサポート窓口についての情報を表示します。
- **? → License**
  ETAS ライセンスマネージャを開きます。

2.3 ツールバー

LABCAR-OPERATOR V5.4.0 のツールバーには以下の操作ボタンが含まれます。

1 Open
ドキュメントを開きます（**File → Open** と同じ機能です）。

2 Save All

プロジェクト全体（CAN エディタ、コネクションマネージャ等の変更内容をすべて含む）を保存します（**File → Save All** と同じ機能です）。

3 Save

現在開いているドキュメントを保存します（**File → Save** と同じ機能です）。

4 Discard

現在アクティブになっているドキュメントについて、まだ保存されていない変更内容をすべて破棄します（**File → Discard** と同じ機能です）。

5 Add Module

モジュール追加ウィザード（"Add Module Wizard"）を開きます。

6 Edit Hardware Configuration

RTIO エディタを起動します（**Project → RTIO Editor** と同じ機能です）。

7 Build LABCAR Project

コード生成を開始します（**Project → Build** と同じ機能です）。

8 Open Experiment Environment

LABCAR-EE（実験環境）を開きます（**Tool → Open Experiment Environment** と同じ機能です）。

9 Help

LABCAR-OPERATOR V5.4.0 についての情報を表示します（**? → Help** と同じ機能です）。

2.4 メインワークスペース

メインワークスペースには LABCAR-IP の主要な機能が含まれます。各機能は個々のタブに分かれて実装されています。

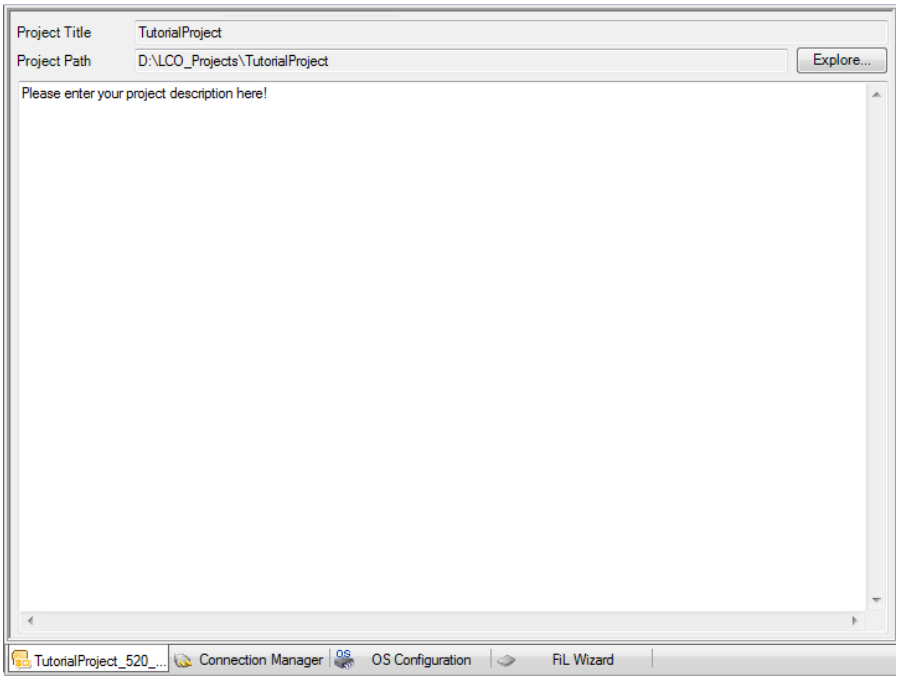


図 2-2 LABCAR-IP のメインワークスペース

"< プロジェクト名 >" タブ（プロジェクト情報タブ）

プロジェクトの名前と、プロジェクトファイルが格納されるパスが表示されます。さらにテキストフィールドに任意の情報を入力することもできます。

"Connection Manager" タブ

このタブにコネクションマネージャの機能がすべて含まれています（215 ページ「コネクションマネージャ」参照）。

"OS Configuration" タブ

リアルタイムオペレーションシステムの設定を行います（227 ページ「リアルタイムオペレーティングシステムの設定（OS コンフィギュレーション）」参照）。

"FiL Wizard" タブ

このタブで FiL モジュールの統合と統合を行います（183 ページ「FiL モジュール」参照）。

2.5 プロジェクトエクスプローラ（"Project Explorer"）

プロジェクトに関連するすべてのオブジェクトとドキュメントは、「プロジェクトエクスプローラ」というドッキングウィンドウで管理されます。

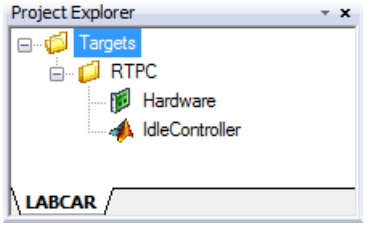


図 2-3 プロジェクトエクスプローラ

このフォルダ構成の最上位には以下のフォルダが表示されます。

"Targets" フォルダ

"Targets" フォルダには、各実験ターゲットのためのサブフォルダが作成されます。

- ***<シミュレーションターゲット名>*** （**"RPPC"**）

  このフォルダのショートカットメニューからは以下のコマンドを実行できます。

  — **Add Module**

  — **Edit OS Configuration**

  各サブフォルダには以下のアイテムが含まれます。

  — 各種モジュール（Simulink モデル、C コード、CAN、信号変換など）

  — ハードウェアコンフィギュレーション

  各アイテムのショートカットメニューからは以下のコマンドを実行できます。

  — **Edit**

  アイテムに対応するエディタを開きます。

  — **Update**

  モジュールが所定のエディタで開き、変更内容が更新されます。

  — **Remove**（ハードウェアコンフィギュレーションのショートカットメニューには表示されません）

2.6 ログウィンドウ

"Messages" タブ

LABCAR-OPERATOR V5.4.0 から出力された情報メッセージとエラーメッセージが表示されます。

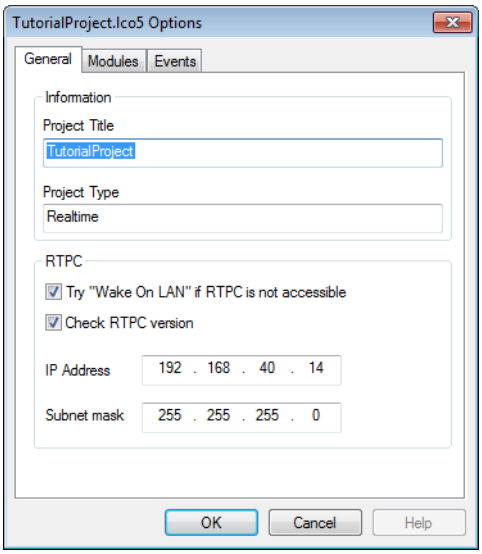
このウィンドウの表示内容を削除するには、このウィンドウを右クリックしてから **Clear Log** コマンドを選択します。

2.7 プロジェクトのプロパティ設定

LABCAR-OPERATOR には、他のコンポーネントと関連したさまざまなプロパティがあります。プロパティを表示／設定するには、メニューコマンド **Project → Options** でプロパティダイアログボックスを開きます。

2.7.1 "General" タブ

このタブの上部には、プロジェクト名（"Project Title"）とプロジェクトタイプ（"Project Type"）のフィールドが表示されます。

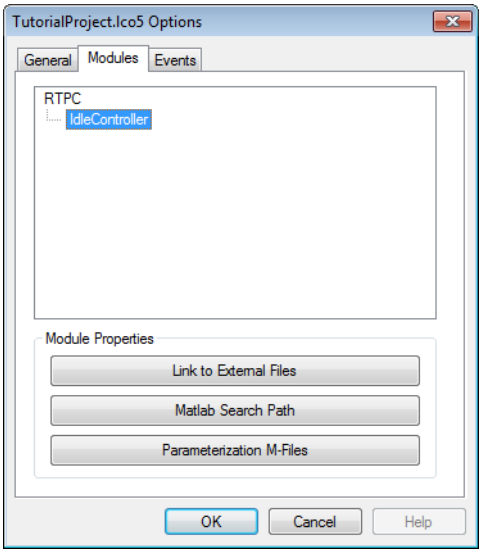


このタブの "RTPC" グループフィールドでは、Real-Time PC に関する以下のオプションを設定できます。

- **Try "Wake On LAN" if RTPC is not accessible**
  このオプションがオンになっている場合は、Real-Time PC が無応答になると、Wake On LAN 機能を使用して Real-Time PC の起動を試みます。
- **Check RTPC version**
  このオプションがオンになっていると、検出された Real-Time PC のバージョンがチェックされます。このチェックはビルド時に行われ、そのバージョンが LABCAR-OPERATOR のカレントバージョンで使用できるかが確認されます。
- **IP Address**
  カレントプロジェクトに割り当てられた Real-Time PC の IP アドレスです（マルチ RTPC プロジェクトの場合）。
- **Subnet Mask**
  IP アドレスのプレフィックスを決定するためのサブネットマスクです。

2.7.2 "Modules" タブ

Simulink 用モデリングコネクタ（Modeling Connector for Simulink）を使用している場合、このタブにプロジェクトに含まれる Simulink モデルや外部 C コードなどのモジュールが表示され、ここで Simulink モデルの管理を行うことができます。



設定内容については 41 ページの「Simulink モデルについての詳細設定」を参照してください。

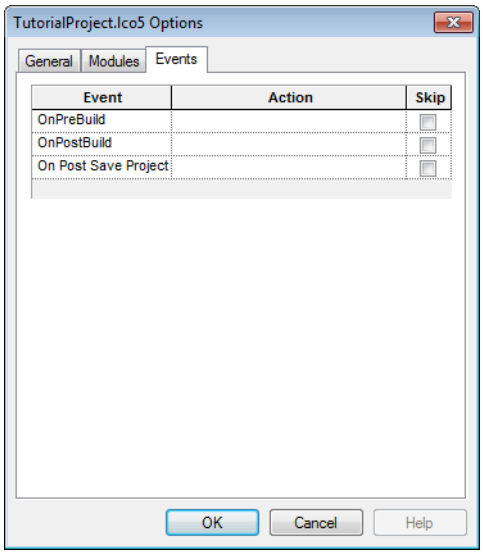
2.7.3 "Events" タブ

ビルド処理の前後に任意のスクリプトを実行させることができます。スクリプトは、Windows 上で実行できる形式（*.exe、*.bat、*.cmd）であればどのタイプのファイルでも使用できます。

イベントにスクリプトを割り当てる：

- **Project → Options** を選択します。
  "*Project_name* properties" ダイアログボックスが開きます。

- "Event" タブを選択します。
  スクリプトを割り当てることができるイベントの一覧が表示されます。



- スクリプトを割り当てたいイベントの "Action" セルをクリックします。
- セルの端の矢印をクリックして **<browse...>** を選択します。
  実行させるスクリプトファイルを選択するためのダイアログボックスが開きます。
- 割り当てたスクリプトを一時的に実行しないようにするには、"Skip" 列をチェックします。

3 LABCAR-IP の操作方法

本章では、LABCAR-OPERATOR プロジェクトにさまざまなタイプのモジュールを統合し、LABCAR-EE で実行される実験用のコードを生成する方法について説明します。

本章は以下の項で構成されています。

- 「モジュール式シミュレーションコード」（24 ページ）
  本項では、さまざまな方法で作成されたモジュールを LABCAR-OPERATOR プロジェクトに組み込む方法について簡潔に説明します。
- 「Matlab/Simulink モデル」（35 ページ）
  本項では Matlab/Simulink モデルの統合について以下の内容を説明します。
- 「ASCET モジュール」（48 ページ）
  本項には、ASCET モジュールの LABCAR-OPERATOR プロジェクトへの統合に関する以下の情報が含まれています。
- 「C コードモジュール」（55 ページ）
  LABCAR-IP では、Matlab/Simulink モデルや ASCET モジュール以外に、ハンドコーディングされた C モジュールをプロジェクトに統合することができます。
- 「CAN モジュール（LABCAR-NIC ― CAN ネットワークの統合）」（85 ページ）
  LABCAR-OPERATOR V5.4.0 のアドオン製品である LABCAR-NIC V5.4.0（Network Integration CAN）は、CAN 通信を含む ECU ファンクションのテストを容易に行うための機能を提供するものです。
- 「CAN シミュレーション用インストゥルメント」（125 ページ）
  LABCAR-EE（Experiment Environment）には、LABCAR-NIC V5.4.0 で CAN バスシミュレーションや CAN バスモニタを行うための CAN メッセージ用インストゥルメントが用意されています。
- 「LIN モジュール（LABCAR-NIL ― LIN ネットワークの統合）」（132 ページ）
  LABCAR-NIL（Network Integration LIN）は、LIN 2.0 や LIN 2.1 ／ LIN 2.2 に準拠する LIN 通信を使用した ECU ファンクションのテストを行うための LABCAR-OPERATOR 用アドオンです。
- 「FlexRay モジュール（LABCAR-NIF V5.4.0 ― FlexRay ネットワークの統合）」（162 ページ）
  LABCAR-OPERATOR V5.4.0 のアドオン製品である LABCAR-NIF V5.4.0（Network Integration FlexRay）は、FlexRay 通信を含む ECU ファンクションのテストを容易に行うための機能を提供するものです。
- 「EB tresos bmc ファイルの同期と NIF モジュールソースコードの生成を行う：」（165 ページ）
  LABCAR-OPERATOR V5.4.0 のアドオン製品である LABCAR-NIF V5.4.0（Network Integration FlexRay）は、FlexRay 通信を含む ECU ファンクションのテストを容易に行うための機能を提供するものです。
- 「FiL モジュール」（183 ページ）
  本項では FiL モジュールの作成について説明します。
- 「リアルタイムプラグイン」（196 ページ）
  本項では、「リアルタイムプラグイン」を作成して LABCAR-OPERATOR プロジェクトに組み込み、実験環境 LABCAR-EE で使用する方法について説明します。
- 「信号変換モジュール」（209 ページ）
  LABCAR-OPERATOR V5.4.0 に含まれている標準的な信号変換モジュールを用いることにより、汎用的な開ループコンフィギュレーションを実現することができます。

- 「RTIO エディタによるハードウェア設定」（214 ページ）
  RTIO エディタを使用してハードウェアの構成とパラメータを設定する方法は、『LABCAR-RTC V5.4.0 - ユーザーズガイド』に記述されています。
- 「コネクションマネージャ」（215 ページ）
  コネクションマネージャは、モジュール間の入力と出力とを接続することにより閉ループ操作を可能にするものです。
- 「リアルタイムオレーティングシステムの設定（OS コンフィギュレーション）」（227 ページ）
  LABCAR-OPERATOR プロジェクトを作成すると、自動的にデフォルトの OS コンフィギュレーションが生成され、プロジェクトに追加されます。
- 「マルチ RTPC ネットワークのセットアップ」（238 ページ）
  マルチ RTPC ネットワークのセットアップについて説明します。

3.1 モジュール式シミュレーションコード

本項では、さまざまな方法で作成されたモジュールを LABCAR-OPERATOR プロジェクトに組み込む方法について簡潔に説明します。

以下のトピックについての情報が含まれています。

- LABCAR モジュール（24 ページ）
- 統合可能なモジュールの種類（25 ページ）
- LABCAR-OPERATOR プロジェクトの作成とモジュールの統合（26 ページ）
- 外部コードの統合（28 ページ）
- 接続の管理（31 ページ）
- オペレーティングシステムの設定（"OS Settings" タブ）（32 ページ）
- 32 ページの「コード生成」

3.1.1 LABCAR モジュール

LABCAR-OPERATOR において、シミュレーションモデルコードは複数の「LABCAR モジュール」で構成され、各モジュールにプロジェクト内の 1 つのコンポーネントが記述されています。

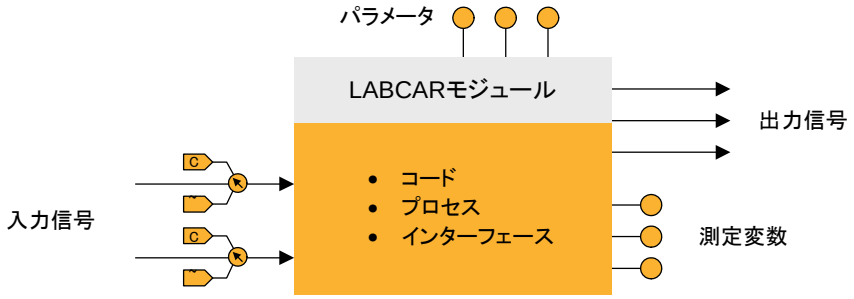


図 3-1 LABCAR モジュール

各 LABCAR モジュールは以下のもので構成されます。

- インターフェース情報
- 機能記述（標準 C コード形式）

インターフェース情報には、モジュールの入出力定義、測定 / 適合変数へのアクセス情報、モジュールを制御するプロセスについての情報が含まれます。

25 ページの図 3-2 は、複数のモジュールからなるプロジェクトの構成図です。プロジェクト内の各モジュールは、コネクションマネージャを用いて信号レベルで接続できます。このようにして信号パスを定義することにより、閉じた制御回路を構築できます。

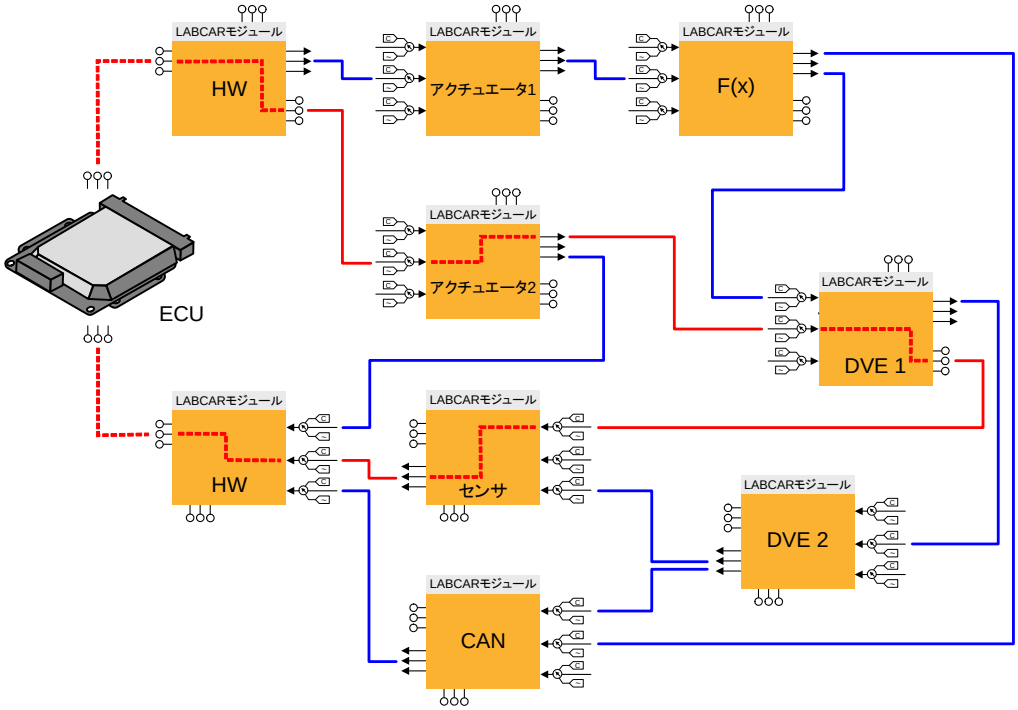


図 3-2 LABCAR-OPERATOR プロジェクト内のモジュール

実行時において各入力の信号パスを切り替え、定数値、生成信号、または信号トレース（テストドライブ時などに記録された一連の信号値）を入力信号として供給することができます（209 ページの「信号変換モジュール」を参照してください）。

C コード形式になっていないモジュールについては、コード生成時にまず C コードが作成されます。その後、個々の C コードパーツが統合されて完全なシミュレーションコードとなり、後にそのコードがシミュレーションターゲット（RTPC）で実行されます。

このアプローチにおいては、C コード生成機能を備えた任意のビヘイビア定義ツールを使用できるので、各ツールが持つ固有の長所（モデリング言語、未処理バスシミュレータ、C コードライブラリなど）を柔軟に活用することができます。

シミュレーションコードの実行

シミュレーションコードのリアルタイム演算は Real-Time PC という Intel® プロセッサアーキテクチャをベースとするシミュレーションターゲットで実行されます。このアーキテクチャはマルチコアプロセッサを包括的にサポートしているので、コードをモジュール単位で複数の CPU コアに分散させることが可能です。

3.1.2 統合可能なモジュールの種類

以下に、LABCAR-OPERATOR プロジェクトに統合できる各種モジュールを紹介します。これらのモジュールの統合に関する注意事項は本書で説明されています。

各モジュールは、プロジェクトエクスプローラに以下のアイコンで示されます。

- **ASCET および MATLAB/Simulink モジュール**
  統合についての詳細情報は、48 ページの「ASCET モジュール」と 35 ページの「Matlab/Simulink モデル」で説明されています。

- **C コードモジュール**
  統合についての詳細説明は、55 ページの「C コードモジュール」で説明されています。

- **CAN、LIN、FlexRay モジュール**
  統合についての詳細な説明は、85 ページの「CAN モジュール（LABCAR-NIC ― CAN ネットワークの統合）」、132 ページの「LIN モジュール（LABCAR-NIL ― LIN ネットワークの統合）」、162 ページの「FlexRay モジュール（LABCAR-NIF V5.4.0 ― FlexRay ネットワークの統合）」で説明されています。

- **FiL モジュール**
  統合についての詳細な説明は、183 ページの「FiL モジュール」で説明されています。

- **開ループアクセスおよび信号変換用のモジュール**
  作成および設定についての詳細な説明は、209 ページの「信号変換モジュール」で説明されています。

- **I/O ハードウェアモジュール**
  これらのモジュールは、モデルと ECU の間の I/O ハードウェアを定義するもので、シミュレーションコードの一部となります。RTIO エディタの使用方法や各ハードウェアの設定については、『LABCAR-RTC V5.4.0 - ユーザーズガイド』を参照してください。

統合手順はモジュールの種類により異なるので、それぞれについて後述します。

3.1.3 LABCAR-OPERATOR プロジェクトの作成とモジュールの統合

ここではプロジェクトを作成して各種モジュールを統合する方法を説明します。

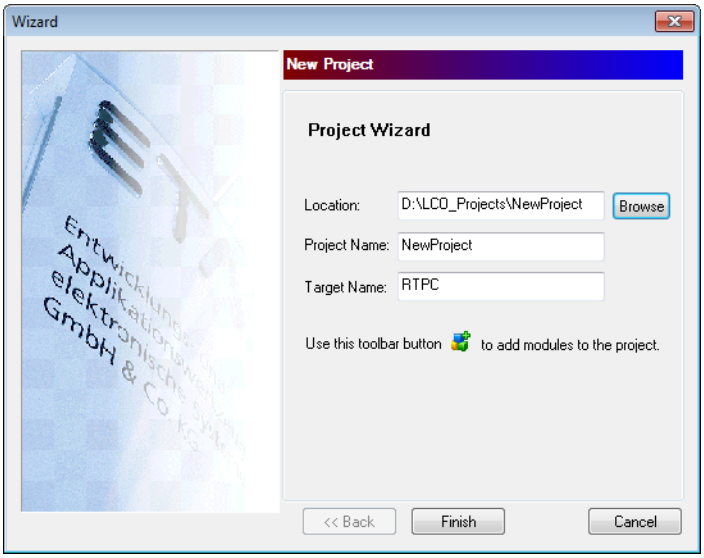
最初に Simulink モデル（新しい「空の」モデル、または既存のモデル）、あるいはモデルをまったく含まないプロジェクトを作成し、さらに必要に応じて C コードモジュールや ASCET モジュールを追加します。

以下にプロジェクトの一般的な作成手順を示します。

プロジェクトを作成する：

- メインメニューから **File → New Project** を選択します。
  プロジェクトウィザードが開きます。
- 新しいプロジェクトを作成するディレクトリを "Location" フィールドに設定します。

- プロジェクト名を "Project Name" フィールドに入力します。

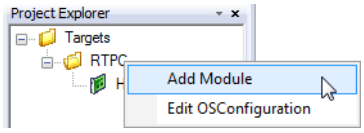


- 必要に応じて、"Target Name" フィールドにこのターゲットの任意の名前を入力します。
- **Finish** をクリックします。

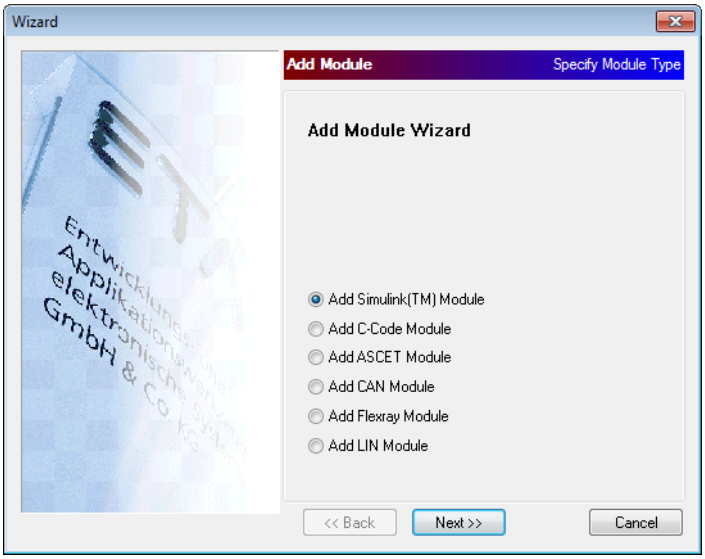
  LABCAR-OPERATOR プロジェクトが作成されます。ここにモジュールを追加していきます。

モジュールを追加する：

- プロジェクトエクスプローラで、実験ターゲットの名前の付いたフォルダを選択して右クリックします。
- ショートカットメニューから、**Add Module** を選択します。



- "Add Module Wizard" ダイアログボックスで、追加するモジュールのタイプを選択します。



以下のタイプのモジュールを追加するには、それぞれ異なる手順が必要です。

- ハードウェアモジュール
  LABCAR-OPERATOR プロジェクトを作成すると、自動的にそのプロジェクト内に「ハードウェアモジュール」が作成されます。このモジュールには、作成時においてはシミュレーションターゲット「RTPC」以外のハードウェアは含まれていません。モジュールのコンフィギュレーション設定（つまり I/O ハードウェアの統合）についての詳細は『LABCAR-RTC V5.4.0 - User's Guide』に記載されています。
- CAN モジュール
  CAN モジュールの作成と管理は LABCAR-IP の "CAN Editor" タブで行います。詳細は 85 ページの「CAN モジュール（LABCAR-NIC ー CAN ネットワークの統合）」を参照してください。
- FiL モジュール
  FiL モジュールの統合と管理は LABCAR-IP の "FiL Wizard" タブで行います。詳細は 183 ページの「FiL モジュール」を参照してください。

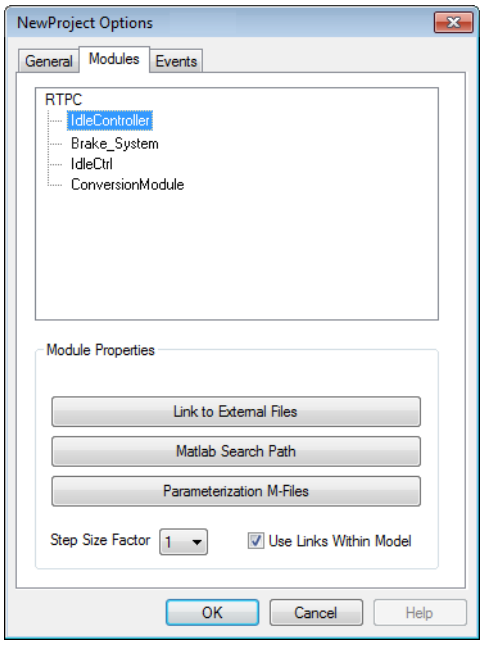
3.1.4 外部コードの統合

特定のタイプのモジュール（Simulink、CAN、信号変換モジュール）の場合、個々のモジュールを統合する以外に、モジュールコード生成時に追加のヘッダ、コード、オブジェクト、ライブラリファイルをコンパイルし、リンクすることができます。このため、頻繁に使用されるコードファイルとライブラリをプロジェクト間で交換して再利用することができます。

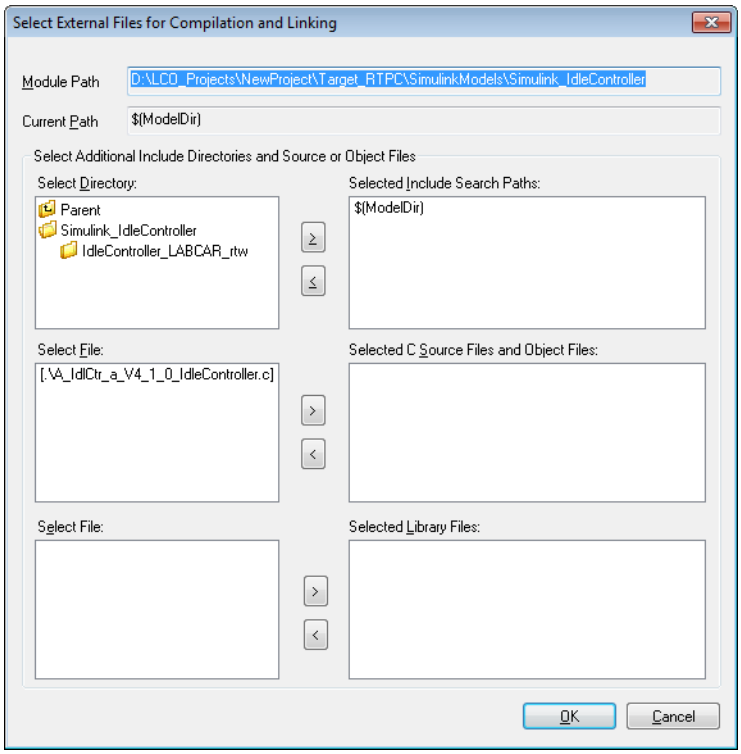
外部データソースを選択する：

- メインメニューから **Project** □**Options** を選択します。
  "*<project_name>* Options" ダイアログボックスが開きます。
- "Modules" タブを選択します。

- 外部ファイルをリンクするモジュールを選択します。



- **Link to External Files** をクリックします。
  "Select External Files for Compilation and Linking" ダイアログボックスが開きます。



このダイアログボックスでは、コンパイルとリンクの際に参照される以下のようなディレクトリとファイルを指定できます。

- ヘッダファイル（`*.h`）の検索場所となるディレクトリ

- ソースコードやオブジェクトコードが保存されているファイル（`*.c`、`*.o`）
- ライブラリファイル（`*.a`）

プロジェクトの移植性とコード生成の安定性を Real-Time PC 上で保証するため、モジュールパス（例：`$(ModuleDir)`）にあるディレクトリとファイルしか参照できないようになっています。

"user" ディレクトリ

プロジェクトディレクトリ *<project_name>*`\Target_RTPC\` の下に `\user` というフォルダが作成され、コンパイル／リンク処理において Real-Time PC 上で必要となるファイルが保存されます。

`\user` フォルダの内容は、ビルド処理中に Real-Time PC の以下のディレクトリに転送されます。

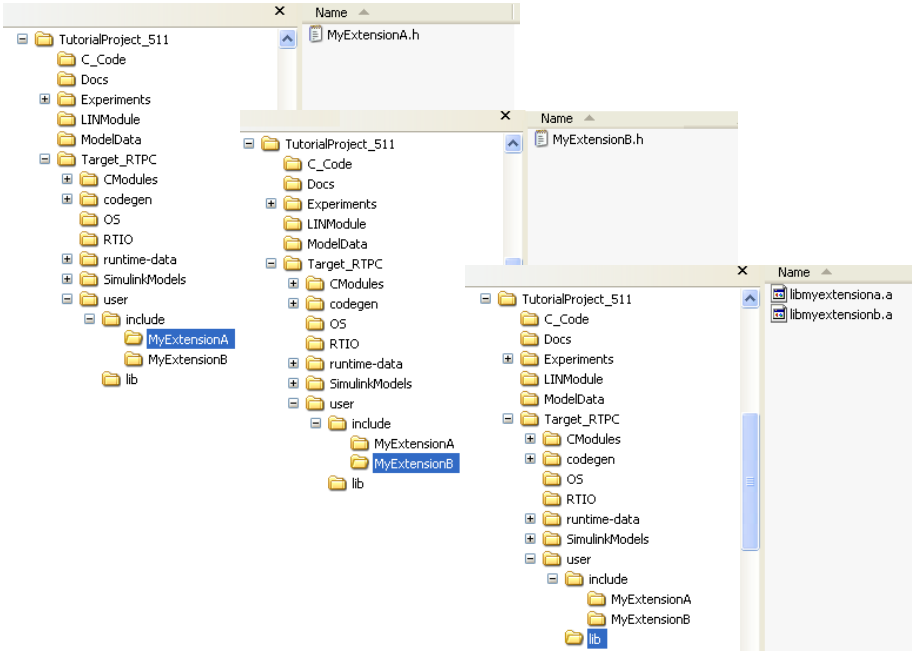
`/home/labcar/codegen/`

`"labcar"` は、ETAS RTPC オペレーティングシステムのデフォルトユーザーです。

`\user` フォルダには以下の 2 つのサブフォルダがあります。

- `include`
  使用されるライブラリの各ルーチンの宣言を含むヘッダファイルが保存されます。
  このフォルダには、さらにサブフォルダが作成されます。
- `lib`
  プロジェクトのリンク時に使用されるライブラリが保存されます。
  このフォルダにはサブフォルダは作成されません。

例：



LABCAR-OPERATOR モジュール（C コードモジュールなど）からライブラリ関数にアクセスできるようにするには、各モジュールのソースコードから所定のヘッダファイルを参照する必要があります。

例：

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

特定のモジュールに外部コードを追加する標準的な方法（**Project → Options**、"Modules" タブ : **Link to External Files**）とは異なり、`user` ディレクトリに保存されたライブラリとヘッダファイルは、すべてのモジュールからアクセスできます。

3.1.5 接続の管理

25 ページの図 3-2 には、プロジェクトモジュールだけでなく、プロジェクトモジュールの入出力間の接続も示されています。これらの接続はコネクションマネージャで作成され、閉ループ処理や信号トレースに用いられます。

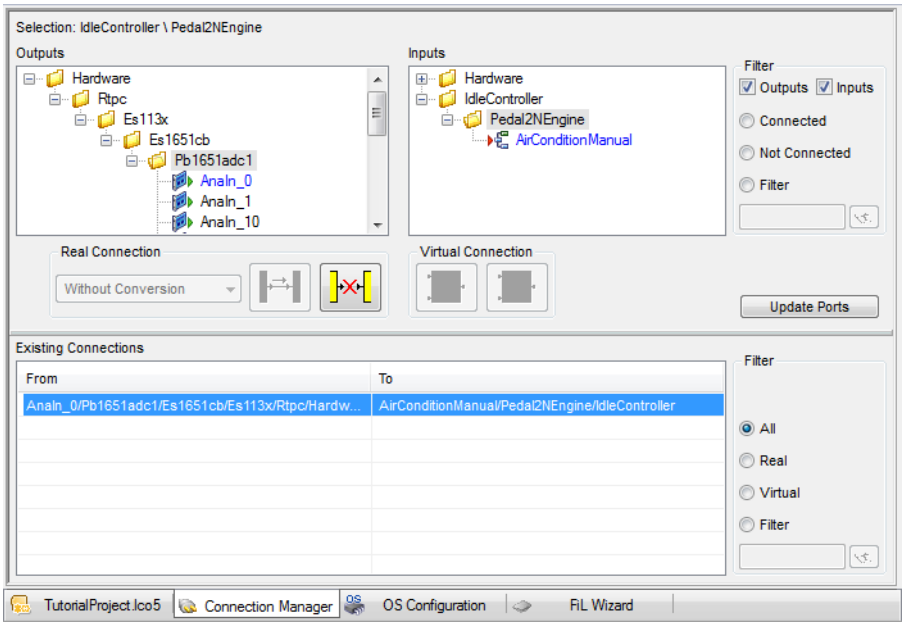


図 3-3 コネクションマネージャ

接続管理の詳細については、215 ページの「コネクションマネージャ」を参照してください。

「接続」を作成する際は、一方のモジュールの入力に信号変換モジュールを追加するかどうかを指定することができます。信号変換モジュールについては、信号変換のタイプを選択でき、さらに、他方のモジュールとの接続を切断し、実際の入力信号の代わりに、マニュアル操作で定義した信号やシグナルジェネレータで生成された信号を供給することができます（224 ページの「信号変換」を参照してください）。

これらのモジュールは実験環境内の専用のインストゥルメント（GUI）で設定します（209 ページの「信号変換モジュール」を参照してください）。

3.1.6 オペレーティングシステムの設定（"OS Settings" タブ）

モジュールをプロジェクトに統合すると、そのモジュール用のプロセスが OS コンフィギュレーション内に表示され、適切なタスクに割り当てられます。

すべてのモジュールを統合した後は、リアルタイムオペレーティングシステムの設定を調整する必要があります。つまり、タスクの追加や削除、各タスクのプロパティ、プロセスのタスクへの割当てやタスクからの削除などを行います。

これらの操作の詳細については、227 ページの「リアルタイムオレーティングシステムの設定（OS コンフィギュレーション）」を参照してください。

3.1.7 コード生成

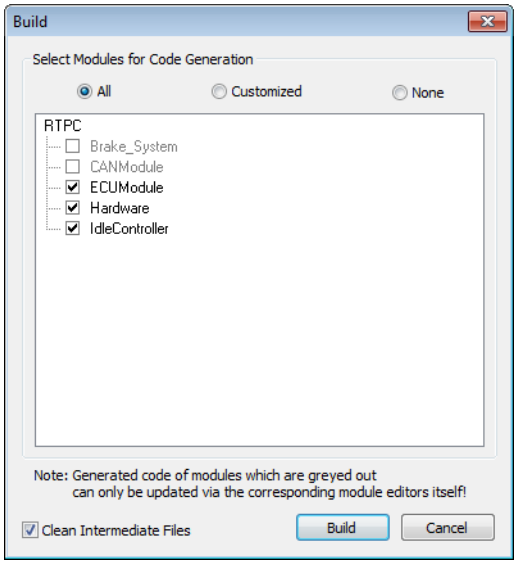
すべてのモジュールをプロジェクトに統合して、信号間を接続し、タスク実行についての設定を行うと、実験ターゲット用のシミュレーションコードを生成できる状態になります。

コードを生成する：

- **Project → Build** を選択します。

または

- **Build LABCAR Project** ボタンをクリックします。
- 以下のダイアログボックスから、コードを生成するモジュールを選択します。



次の 2 つのいずれかに該当するモジュールは、リスト内ではグレイアウトされ、選択できない状態になっています。

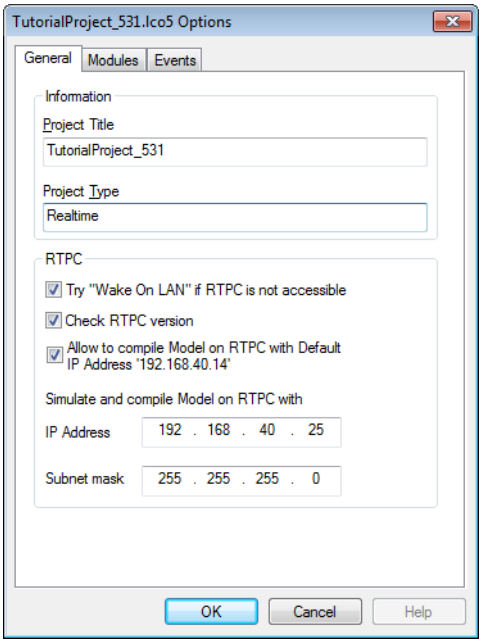
- C コードモジュール（コードは自動的に生成されます）
- Simulink モジュールまたは ASCET モジュールの場合で、対応するアドオン LABCAR-MCS（Modeling Connector for Simulink）または LABCAR-MCA（Modeling Connector for ASCET）がインストールされていない場合

Clean Intermediate Files オプションをオンにすると、コンパイルされたオブジェクトファイルはビルド処理終了後に削除されます。プロコンパイルされたコードを再利用しない場合は、これをオンにしておいてください。

- **Build** をクリックします。
  コード生成処理が開始され、進行状況が "Messages" ダイアログボックスに表示されます。

3.1.8 プロジェクトの一般オプション

Project → Options でオプションダイアログボックスを開き、"General" タブを選択すると、プロジェクトに関する重要なオプションが表示されます。



Information:

- **Project Title**
  プロジェクト作成時に定義されたプロジェクト名
- **Project Type**
  常に "Realtime"

RTPC:

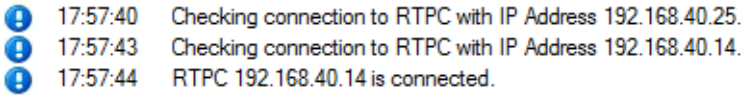
- **Try "Wake On LAN" if RTPC is not accessible**
  ネットワーク経由で Real-Time PC の起動を試みます（"Wake-on-LAN"）。
  これを使用するには、"magic packets" と呼ばれるものを送信するためのツール（例 : フリーツール "WOL - Magic Packet Sender 2007"）がインストールされたネットワーク PC が必要です。このツールの設定に必要なデータは、ETAS RTPC の web インターフェースの "Wake On LAN Settings"（**Main Page >> Power Control**）に表示されます。

- **Check RTPC version**
  LABCAR-OPERATOR の各バージョンは、基本的に、それぞれ特定のバージョンの ETAS RTPC にのみ互換性があります（詳しいバージョン情報は、リリースノートを参照してください）。このオプションにより、シミュレーションターゲットのバージョンが正しいかどうかを確認できます。

  非互換バージョンが検出されると、ビルド処理は中断されます。
- **Allow to compile model on RTPC with default IP address '192.168.40.14'**
  このオプションがオンになっていると、Real-Time PC の IP アドレスに関わらず、プロジェクトは常にデフォルト IP アドレスのシステム上でビルドされます。

  例 : IP アドレスが 192.168.40.25 の Real-Time PC 用にプロジェクトが設定されていたとしても（34 ページの「Simulate and compile model on RTPC with」を参照）、このオプションがオンになっていると、ビルド処理においてその IP アドレスは使用されません。

  設定された IP アドレスの代わりにデフォルトアドレスが使用されます。

  この処理は、以下のような手順で行われます。

  — 設定された IP アドレスをチェック

  設定された IP アドレスの Real-Time PC が見つかると、それを使用してビルド処理を行います。

  見つからない場合 :

  — フォールバック IP アドレス 192.168.40.14 をチェック

  フォールバックアドレスで Real-Time PC が見つかると、それを使用してビルド処理を行います。

  見つからない場合 :

  — "Wake On LAN" がオンの場合は、設定されたアドレスを使用して Real-Time PC のウェイクアップが試みられます。

  設定されたアドレスで Real-Time PC をウェイクアップできた場合は、それを使用してビルド処理を行います。

  ウェイクアップできない場合 :

  フォールバック IP アドレスでの Real-Time PC のウェイクアップを試みます。

  フォールバック IP アドレスで Real-Time PC をウェイクアップできた場合は、それを使用してビルド処理を行います。

  ウェイクアップできない場合 :

  — ビルド処理は実行されません。
- **Simulate and compile model on RTPC with**
  プロジェクトをコンパイルして実行する Real-Time PC の IP アドレス（推奨範囲 : 192.168.40.10 ～ 192.168.40.40）とサブネットマスクを指定します。

3.2 MATLAB/Simulink モデル

本項では MATLAB/Simulink モデルの統合について以下の内容を説明します。

- Simulink モデル（35 ページ）
  - ユーザー定義 C コード：パス名（35 ページ）
  - ファンクションコールサブシステム（36 ページ）
- Simulink モデルの使用についての制約（37 ページ）
  - S-Function（37 ページ）
  - 絶対時間を使用するブロック（37 ページ）
  - パラメータや出力が生成されないブロック（37 ページ）
- 新しいプロジェクトの作成（38 ページ）
- Simulink モデルについての詳細設定（41 ページ）

ソフトウェア要件

LABCAR プロジェクトに MATLAB/Simulink モデルを統合するには、以下の要件があります。

- MATLAB のインストール :
  サポートされているバージョンについては、リリースノート（**? → Help**）を参照してください。
- LABCAR-MCS V5.4.0（Simulink 用モデリングコネクタ）のライセンス

3.2.1 Simulink モデル

Simulink モデルの C コードは Real-Time Workshop によって生成されますが、この際、必ず守らなければならないポイントがいくつかあります。本項ではそれらについて説明します。

注記

コード生成においては、必ず 1 つのバージョンの Real-Time Workshop のみを使用するようにしてください。それぞれ異なるバージョンの MATLAB/Simulink で作成された複数のモデルをプロジェクトに統合しようとすると、不具合が生じる可能性があります。

ユーザー定義 C コード：パス名

Real-Time Workshop では、ユーザー定義されたコードを組み込むことができます。これには、S-function や S-function により呼び出されるその他のコードが含まれます。

既存のモデル用にプロジェクトを作成すると、そのオリジナルのディレクトリ全体（すべてのサブディレクトリを含む）が、プロジェクトディレクトリにコピーされるか、または参照され、モデルのバージョン管理が有効になります。

さらに、コンパイルとリンクを行うため、ビルド処理が行われるディレクトリに必要なファイルがコピーされます。この際、Simulink モデルに属するユーザー定義コードに関していくつかの制約条件が生じます。

- C コードの場合、インクルードファイルの標準の拡張子は `*.h` ですが、実際には #include 文はどのような拡張子を持つファイルでも参照することができます。
  しかし LABCAR-MCS V5.4.0 がサポートするファイル拡張子は `*.h` と `*.c` のみであるため、それ以外の拡張子が使用されていると、コンパイラがファイルを見つけられない場合があります。

- C コードでは、通常、#include 文にはファイル名だけを記述し、ファイルのパスはコンパイラスイッチによって指定しますが、下の例の 2 行目と 3 行目に示されているように、#include 文にパスを含むファイル名を記述することもできます。
  例：

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

  しかし LABCAR-MCS V5.4.0 は 1 行目と 2 行目の記述方法しかサポートしていないため、他の記述方法を使用すると、コンパイラがファイルを見つけられない場合があります。
- インクルードファイルの複数の検索パスに含まれるインクルードファイルが、コンパイル対象ディレクトリ、つまり、リンクする外部ファイルのコピー先と同じディレクトリにコピーされます（42 ページの「他のファイルにリンクする（"Link to External Files"）：」を参照してください）。
  この際、名前が同じファイルが複数存在しないことを確認してください。ファイル名が重複していると、コピーの順序によっては一部のファイルが上書きされてしまう可能性があります。

ファンクションコールサブシステム

使用するモデルに「ファンクションコールサブシステム」が含まれている場合や、短い演算周期で実行する必要のないモデルパート用にユーザーがファンクションコールサブシステムを作成した場合は、以下の点を考慮してください。

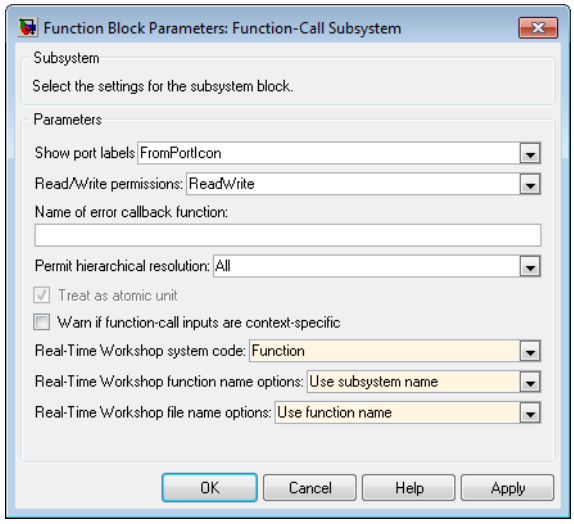
これらのブロックについてパラメータを適切に設定すると、Real-Time Workshop は、ブロック内のモデルパート用に独立したプロセスを生成します。このモデルパートはターゲット上の「シミュレーション演算周期」では実行されません。つまり、ユーザーがこのモデルパートをタスク（時間トリガ式かイベントトリガ式のもの）にアタッチするまではまったく実行されません。

このような独立したプロセスが生成されるようにするには、ファンクションコールサブシステムのパラメータを以下のように設定してください。

設定を選択する：

- Simulink モデル内の "Function-Call Subsystem" ブロックを選択して右クリックします。
- ショートカットメニューから **Subsystem parameters** を選択します。
  "Block Parameters: Function-Call Subsystems" ダイアログボックスが開きます。

- 下図のとおりに設定します（R2007b の場合）。



この設定により生成されたプロセスには、`<model_name>_<subsystem_block_name>` という名前が付きます。

注記

このプロセスは、LABCAR-OPERATOR においてプロジェクトがビルド（**Project → Build**）されるまで、OS コンフィギュレーション（227 ページ参照）には表示されません。

3.2.2 Simulink モデルの使用についての制約

Real-Time Workshop では、使用できる Simulink モデルについて、以下に示すいくつかの制約があります。

S-Function

Real-Time Workshop は、データ型 "double" のパラメータを持つ S-function しか処理できません。他のデータ型が見つかると、コード生成が強制終了します。

また、Real-Time Workshop はカスタムデータ型（"ssRegisterDataType" で宣言されるもの）の出力を持つ S-function を処理できません。この場合、C コードは正常に生成されますが、コンパイル時には、マクロが "ssRegisterDataType" 呼び出しを "ssRegisterDataType_cannot_be_used_in_RTW" に置き換えるため、エラーメッセージが発行されます。

絶対時間を使用するブロック

Real-Time Workshop は、絶対時間を使用するブロック（"Sine Wave"、"Clock"、"Pulse Generator"、"Signal Generator"、"Transport Delay" など）を処理することはできません。

このようなタイプのブロックを含むモデルについては、C コードの生成とコンパイルは正常に行えますが、実行時には、このブロックは誤った結果を返します。

パラメータや出力が生成されないブロック

モデルブロックのうち、以下のブロックにはパラメータや出力が生成されません。

- Scope

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

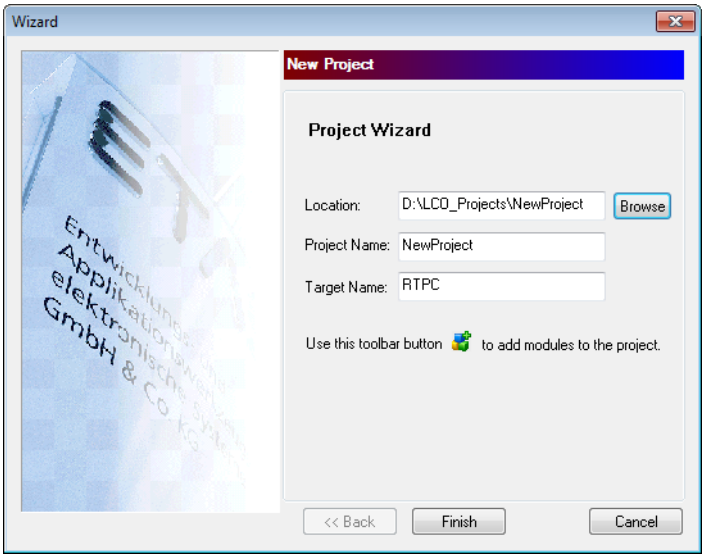
3.2.3 新しいプロジェクトの作成

本項では LABCAR-OPERATOR プロジェクトを作成するための情報を紹介します。

LABCAR-OPERATOR プロジェクトを作成する：

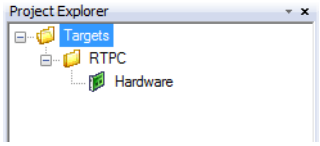
- メインメニューから **File** → **New Project** を選択します。

  "Project Wizard" ダイアログボックスが開きます。
- 新しいプロジェクトを作成するディレクトリを "Location" フィールドに設定します。
- プロジェクト名を "Project Name" フィールドに入力します。



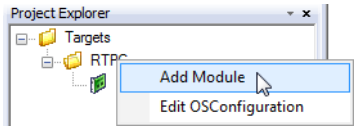
- 必要に応じて、"Target Name" フィールドにこのターゲットの任意の名前を入力します。
- **Finish** をクリックします。
- **Finish** をクリックします。

  デフォルトのハードウェアモジュールを含むプロジェクトが作成されます。

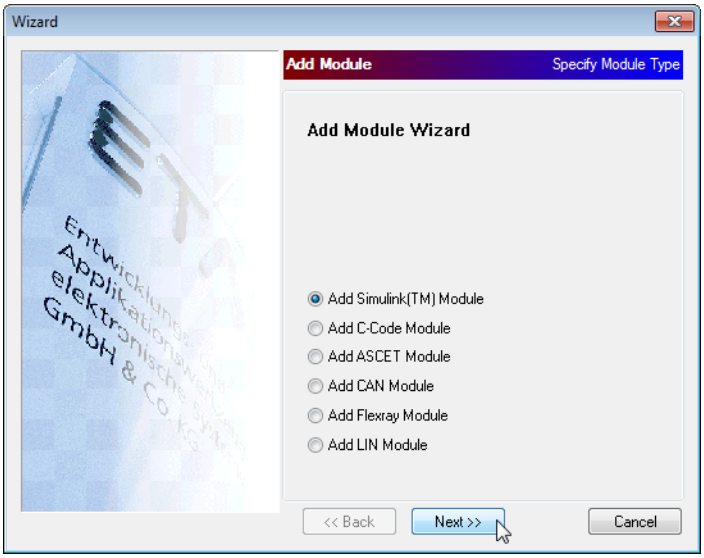


Simlink モデルをプロジェクトに追加する：

- プロジェクトエクスプローラで、実験ターゲットの名前が含まれるフォルダを選択して右クリックします。
- ショートカットメニューから **Add Module** を選択します。

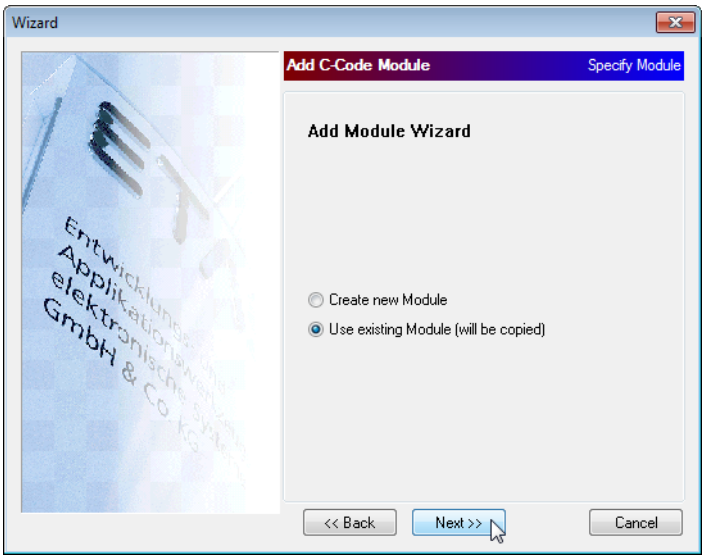


- "Add Module Wizard" ダイアログボックスで "Add Simulink Module" を選択します。

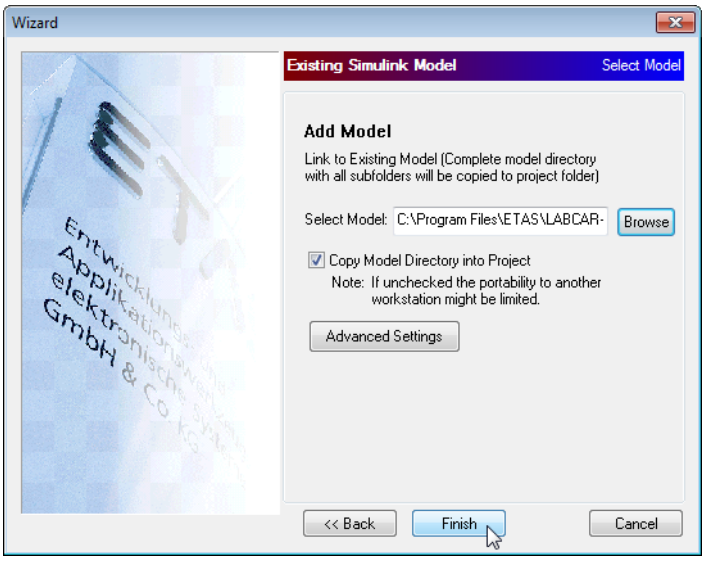


- **Next** をクリックします。

- "Use existing Simulink model" を選択します。



- **Next** をクリックします。
- **Browse** をクリックして、プロジェクトに追加したい Simulink モデルを選択します。



- **Copy Model Directory into Project** オプションをオンにすると、Simulink モデルとそれに関連するすべてのファイルがプロジェクトディレクトリにコピーされます。オフの場合は、モデルは参照されるのみとなります。

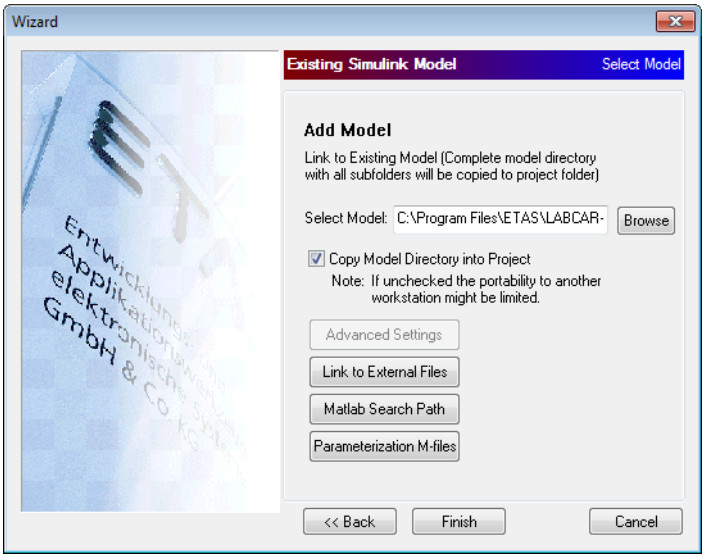
注記

モデルを参照のみとして使用する場合、LABCAR-OPERATOR プロジェクト用に合わせて参照を調整する必要があります。`<project_name>\Target_RTPC\SimulinkModels\Simulink_<model_name>` ディレクトリ内のテキストファイル `<model_name>.conf` にモデルディレクトリの絶対パスが保存されているので、このパスを LABCAR-OPERATOR プロジェクトの相対パスまたは絶対パスに変更することができます。

- **Finish** をクリックします。
  プロジェクトにモデルが追加されます。

3.2.4 Simulink モデルについての詳細設定

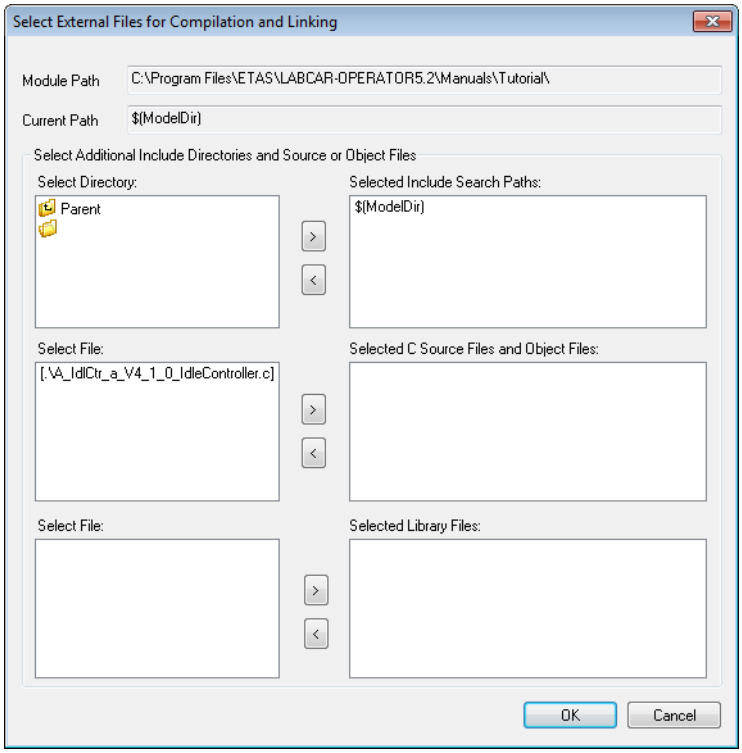
プロジェクト作成時、使用する Simulink モデルを指定する際に、同じダイアログボックスにある **Advanced Settings** ボタンを使うと、Simulink モデルについての詳細な設定を行うことができます。



さらに、コンパイル時に必要なインクルードディレクトリやソース／オブジェクトファイル、またビルド処理でリンクするオブジェクトライブラリもここで追加することができます。

他のファイルにリンクする（"Link to External Files"）：

- **Link to External Files** をクリックします。
  下のダイアログボックスが開きます。



注記

このリストに表示されるライブラリファイルは、拡張子が ".a" または ".lib" のファイルのみです。

- マウスを使用してディレクトリとファイルを選択し、**>** ボタンをクリックしてこれらのファイルをダイアログボックス右側のリストに追加します。

注記

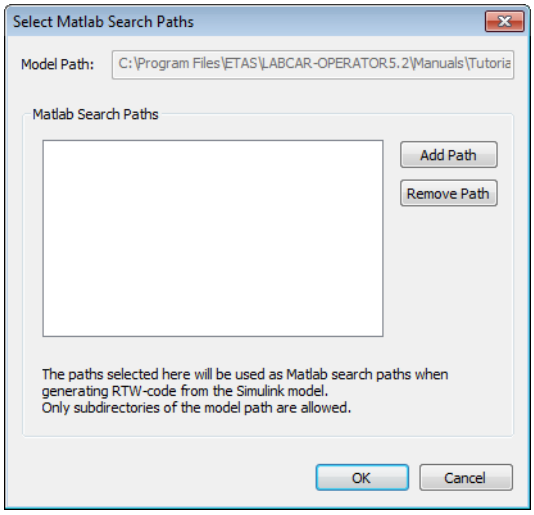
ここで選択できるのは、当該モデルが格納されているディレクトリのサブディレクトリに格納されているファイルとディレクトリだけです。

- **OK** をクリックします。

MATLAB 検索パスをプロジェクトに追加するには、以下の手順を実行してください。

MATLAB 検索パスを追加する（"Matlab Search Paths"）：

- **Matlab Search Paths** をクリックします。
  下のダイアログボックスが開きます。



- **Add Path** をクリックします。
  ディレクトリブラウザが開き、そこからディレクトリを選択することができます。

注記

ここで選択できるのは、モデルパス（"Model Path"）にあるディレクトリだけです。

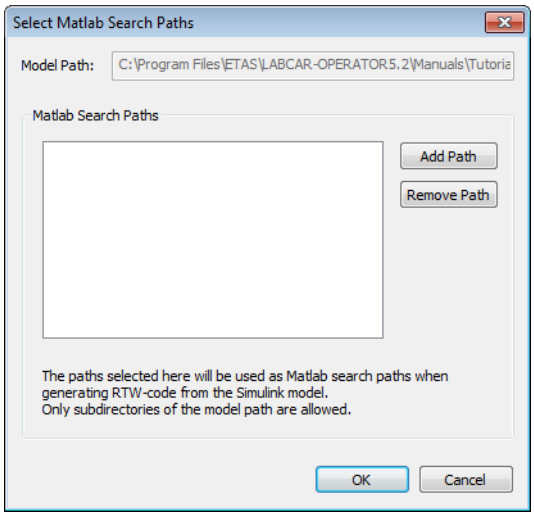
  選択されたパスがリストに表示されます。
- 検索パスをリストから削除するには、リストからそのパスを選択して **Remove Path** をクリックします。
- **OK** をクリックします。

Simulink モデルが開かれた時に評価されるパラメータ定義ファイルを選択するには、以下の手順を実行してください。

パラメータ定義ファイルを選択する（"Parameterization M-Files"）：

- **Parameterization M-Files** をクリックします。

- 下のダイアログボックスが開きます。



- **Browse** をクリックします。

  ファイル選択ダイアログボックスが開き、ここで、ファイルを選択することができます。

  選択されたファイルがリストに表示されます。
- ファイルをリストから削除するには、そのファイルを選択して **Remove from List** をクリックします。
- **OK** をクリックします。

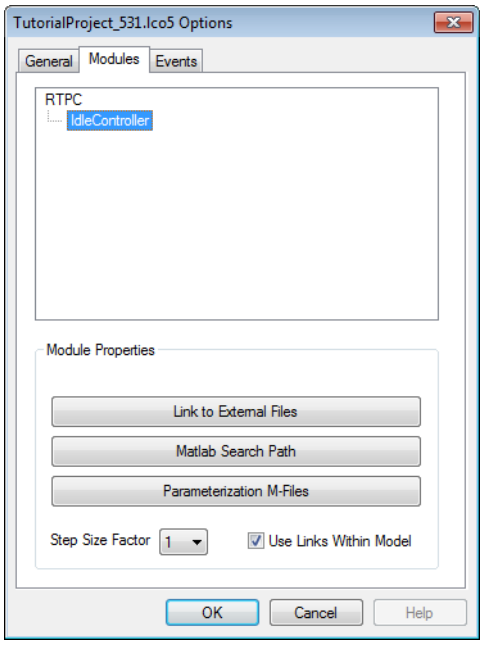
プロジェクト作成時だけでなく、すでに作成されたプロジェクトについて作業している時でも、上述の各ダイアログボックスには常にアクセスできるので、いつでも設定変更を行うことができます。

設定を変更する：

- **Project → Options** を選択します。

  "<project name> Options" ダイアログボックスが開きます。

- "Modules" タブを選択します。



当該モデル（ダイアログボックスのタイトル部に表示されています）について、上述のすべての設定を変更できます。

Simulink モデルの積分ステップ幅と LABCAR-EE のタスク周期を設定する：

一般的に、Simulink モデルの固定ステップ幅で定義されたモデルの演算は、実験環境で実行／表示される頻度（OS コンフィギュレーションのタスク周期）よりもはるかに高い頻度で実行することが好ましいため、タスク周期 dT_{EE} が積分ステップ幅 fss_{SL} の n 倍の値（factor n = 1 ～ 10）になるように設定することができます。

式 3-1 $fss_{SL} = dT_{EE} / n$

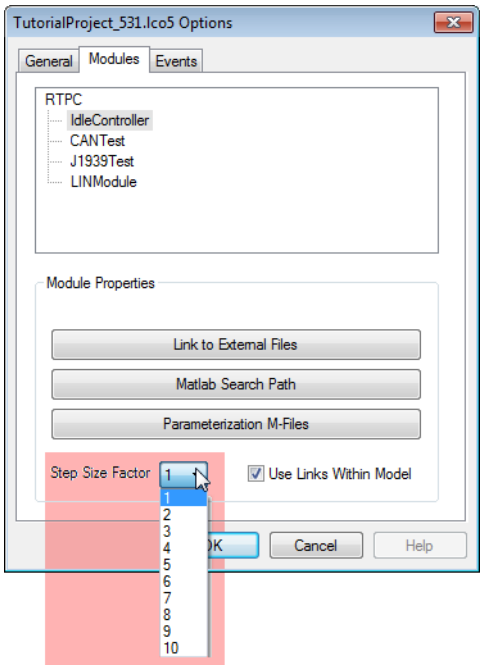
注記

上記の式で使用されている係数 n は、LABCAR-OPERATOR の Simulink モデルごとに異なります。

この設定は以下のように行います。

- **Project → Options** を選択します。
  "<project name> Options" ダイアログボックスが開きます。
- "Modules" タブを選択します。
- 設定を行いたい Simulink モデルを選択します。

- 係数 n を指定します。



- **OK** をクリックして設定を確定します。
- **File → Save** を選択して設定を保存します。

  これにより、Simulink モデルのステップ幅 fss が、OS コンフィギュレーション内に定義されたタスク周期の分数（1/n）に設定されます。

Matlab リンクのサポート機能を有効／無効にする：

V5.3.1 より、出力ポートと入力ポート用の Matlab リンクがデフォルトでサポートされるようになりました。それより前のバージョンのコードを生成するには、このサポート機能を無効にする必要があります。

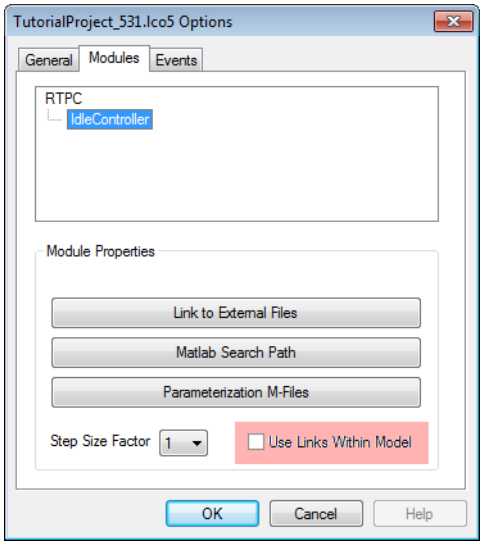
- **Project → Options** を選択します。

  "<project name> Options" ダイアログボックスが開きます。
- "Modules" タブを選択します。
- 設定を行いたい Simulink モデルを選択します。

注記

このサポート機能のオン／オフは、LABCAR-OPERATOR プロジェクト内の各 Simulink モデルごとに設定できます。

- **Use Links Within Model** オプションのオン／オフを切り替えます。オフにするとこの機能が無効になります。



- **OK** をクリックして設定を確定します。
- **File → Save** を選択して設定を保存します。

3.3 ASCET モジュール

本項には、ASCET モジュールの LABCAR-OPERATOR プロジェクトへの統合に関する以下の情報が含まれています。

- ASCET モジュールについての一般情報（48 ページ）
- 準備（48 ページ）
- ASCET モジュールの統合（49 ページ）
- EE で扱う測定変数とパラメータ（54 ページ）

3.3.1 ASCET モジュールについての一般情報

ASCET プロジェクトについて "RTPC" ターゲット用のコード生成と転送が実行されると、さまざまなコードファイルやヘッダファイルのほか、以下のファイルが作成されます。

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

`<project_name>.six` ファイルは SCOOP-IX ファイル[1] です。このファイルには、ASCET モジュールのすべてのインターフェースが記述されています。ASCET プロジェクトは、1 つのモジュールとして、このファイル内の情報とともに LABCAR-OPERATOR プロジェクトに組み込むことができます。

`<project_name>.oil` ファイルにはオペレーティングシステムのコンフィギュレーション（タスクやプロセスなど）が含まれています。このファイルは、ASCET プロジェクトを統合するときに必要に応じてインクルードできます。

3.3.2 準備

ASCET のインストール

ASCET プロジェクトをモジュールとして確実に LABCAR-OPERATOR に統合するためには、各コンポーネント（ASCET-SE、ASCET-MD、ASCET-RP）を含む ASCET（バージョン 6.0.1 以上）がインストールされている必要があります。

ASCET の準備

ASCET のバージョンに応じて、"RTPC" ターゲット用にプロジェクトをエクスポートするための所定のアドオン機能が必要です。

注記

ASCET V6.1.0 以降では、この機能を個別にインストールする必要はありません。

この機能をインストールするには、製品 DVD のインストール起動画面の **Installation** リンクをクリックして "Installation" ページを開きます。ここで **Install RTPC export for ASCET 6.x.y (LABCAR-ASC)** をクリックし、インストールプログラムの指示に従ってください。

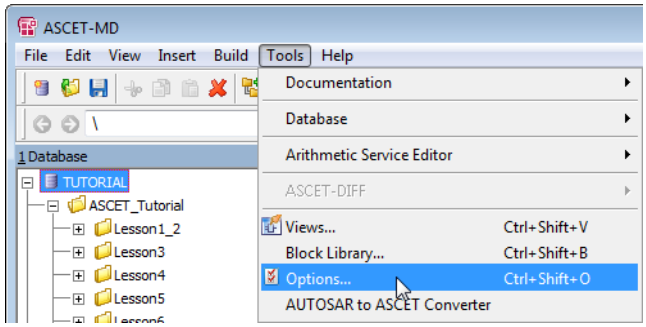
ASCET のオプション設定

プロジェクトを ASCET から転送するためには、ASCET オプションを以下のように設定する必要があります。

1. "SCOOP-IX" の詳細については、『INTECRIO V3.0 ユーザーズガイド』を参照してください。

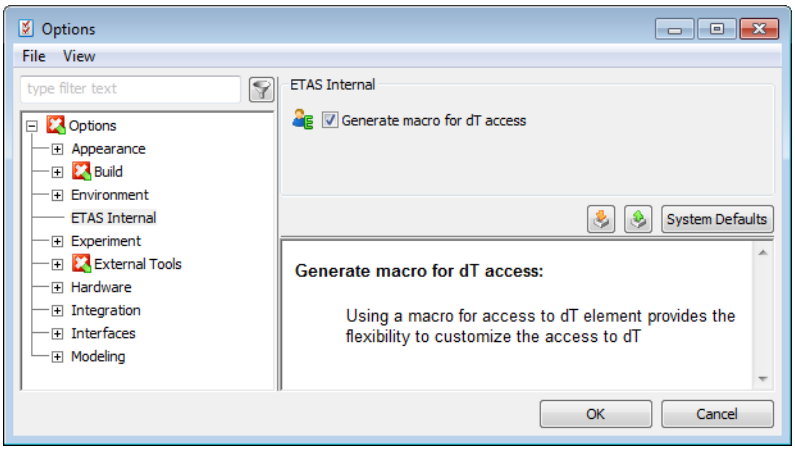
オプションを設定する：

- ASCET を起動します。
- メインメニューから **Tools → Options** を選択します。



"Options" ダイアログボックスが開きます。

- 左側のフィールドから "ETAS Internal" を選択し、右側に表示される "Generate macro for dT access" というオプションをオンにします。



- **OK** をクリックしてこのダイアログボックスを閉じます。

3.3.3 ASCET モジュールの統合

本項では、LABCAR-OPERATOR で ASCET プロジェクトを統合する方法について説明します。

注記

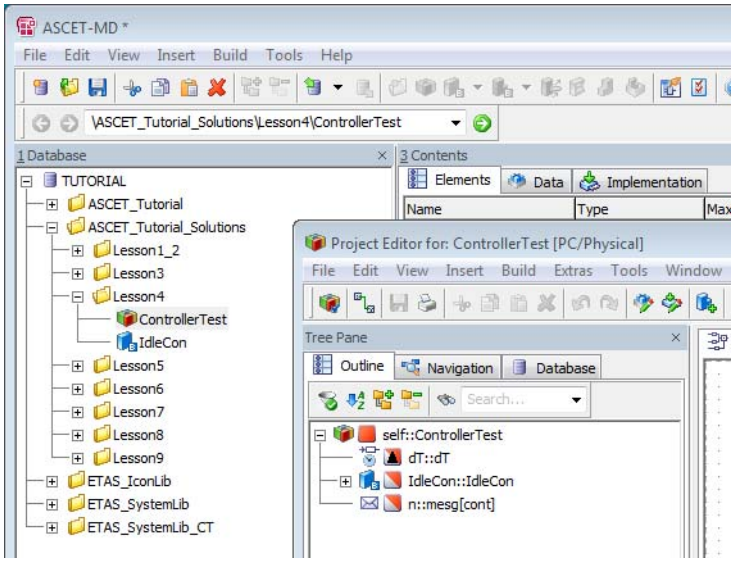
プロジェクトの ASCET バージョンが V6.0.1 より古い場合は、まず古いバージョンの ASCET でプロジェクト（またはデータベース全体）をエクスポートし、新しいバージョンにインポートする必要があります。

ASCET プロジェクトを転送する：

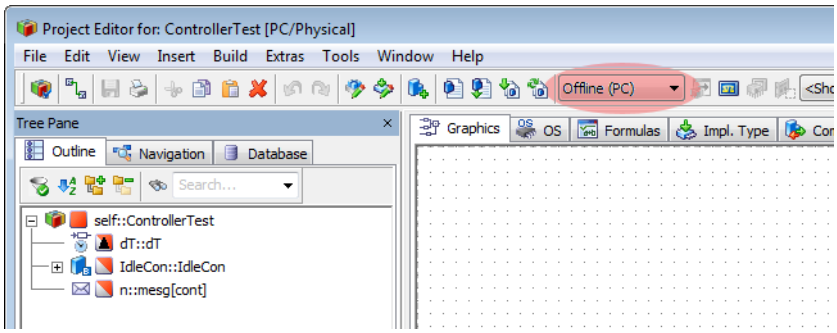
- 転送したいプロジェクトが含まれるデータベースを ASCET で開きます。

- プロジェクト（この例では "TUTORIAL" データベースの "Lesson4" フォルダにある "ControllerTest" プロジェクト）をダブルクリックします。

  そのプロジェクトがプロジェクトエディタで開きます。



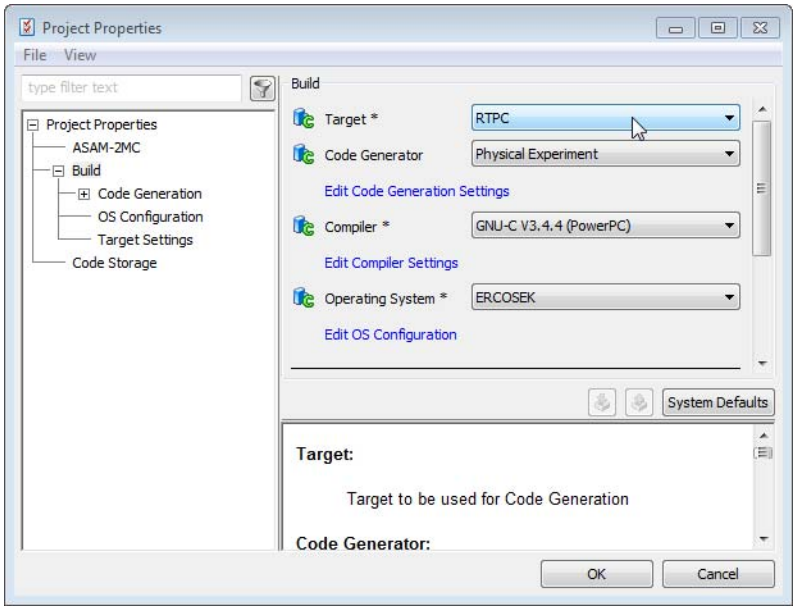
その時点では、PC でのオフラインシミュレーション用のターゲットが選択されています。



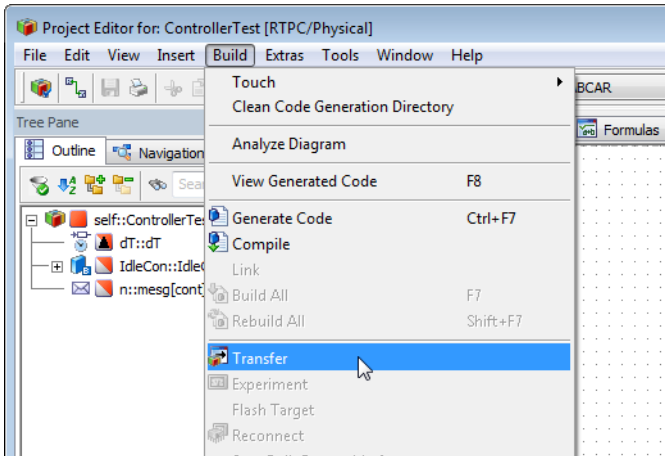
- プロジェクトエディタの **File → Properties** を選択します。

  "Project Properties" ダイアログボックスが開きます。

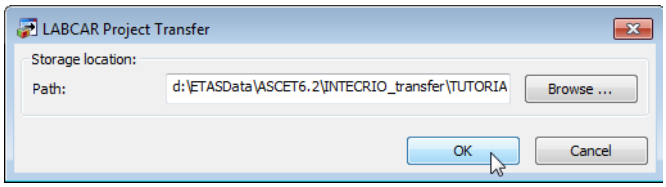
- 左側のフィールドから "Build" を選択し、右側に表示される "Target" オプションのコンボボックスから "RTPC" を選択します。



- **OK** をクリックします。
- ASCET メインウィンドウから **File → Save** を選択します。
- ASCET プロジェクトエディタのメインメニューから **Build → Touch → Recursive** を選択します。
- **Build → Transfer** を選択します。



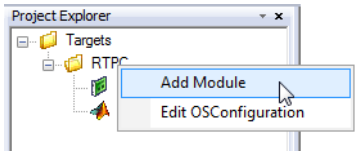
- 生成されたファイルを格納するディレクトリを指定します。



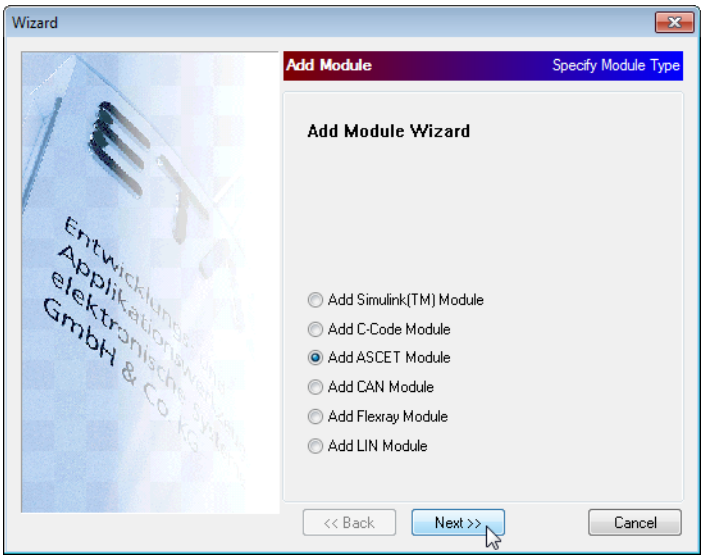
生成された各ファイル（`*.c`、`*.h`、`<project_name>.six`、`<project_name>.oil`）がこのディレクトリに格納されます。

ASCET モジュールを統合する：

- LABCAR-IP を起動し、ASCET モジュールを追加する LABCAR-OPERATOR プロジェクトを開きます。
- プロジェクトエクスプローラから、実験ターゲットの名前を持つフォルダを選択して右クリックします。
- ショートカットメニューから **Add Module** を選択します。

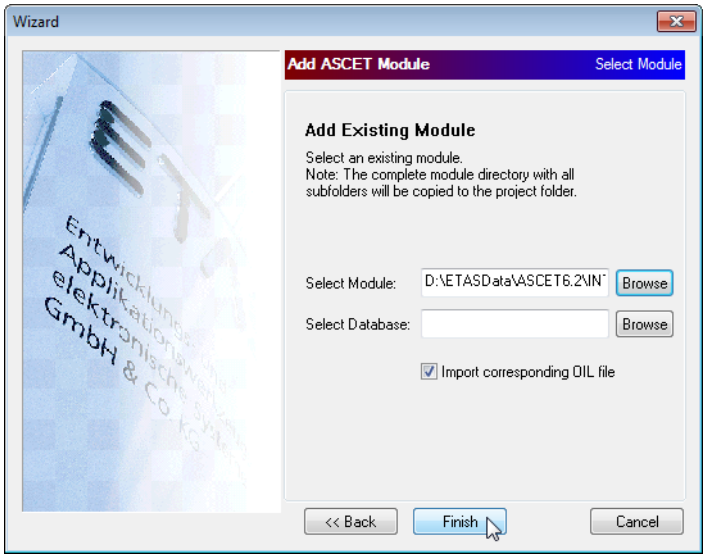


- "Add Module Wizard" ウィンドウで、"Add ASCET Module" を選択します。



- **Next** をクリックします。
- **Use existing ASCET model** を選択します。
- **Next** をクリックします。

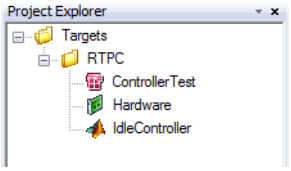
- 追加するモジュールのディスクリプションファイル（例： `ControllerTest.six`）を選択します。



- "Select Database" フィールドで、エクスポートされたモデルが含まれる ASCET データベースを指定しておくと、プロジェクトエクスプローラで ASCET モジュールをダブルクリックして開くことができます。

  データベースがここで選択されていない場合は、モジュールをダブルクリックして開く際に、パスを手入力する必要があります。
- **Finish** をクリックします。

  モジュールが統合され、プロジェクトエクスプローラに表示されます。



- プロジェクトを保存します。

  ASCET プロジェクトの受信メッセージのうち ASCET プロジェクトの送信メッセージに接続されていないものについては、インポート時に入力ポートが生成され、これをコネクションマネージャ内でプロジェクトの他の信号に接続することができます。
- コネクションマネージャ内で、他のモジュールへの接続を定義します。
- 実験用コードを生成します（**Project → Build**）。
- LABCAR-EE で実験を開きます（**Tools → Open Experiment Environment**）。

3.3.4 EE で扱う測定変数とパラメータ

ASCET プロジェクトと EE

前の例で使用した ASCET プロジェクトのコンポーネント（"TUTORIAL" データベースの "Lesson4" フォルダにあります）を図 3-4 に示します。

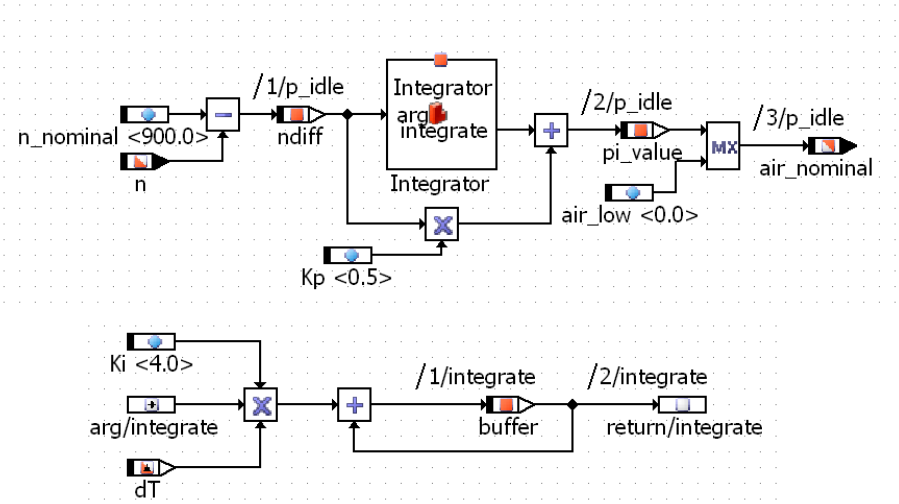


図 3-4 "ControllerTest" プロジェクトのコンポーネント "IdleCon"（上）と "Integrator"（下）

アイドルコントローラは実際の速度 "n" を目標速度 "n_nominal" と比較し、これらの値を使用して目標給気 "air_nominal" を調整します。

この ASCET プロジェクトのコンポーネント、および測定変数とパラメータを図 3-5 に示します。

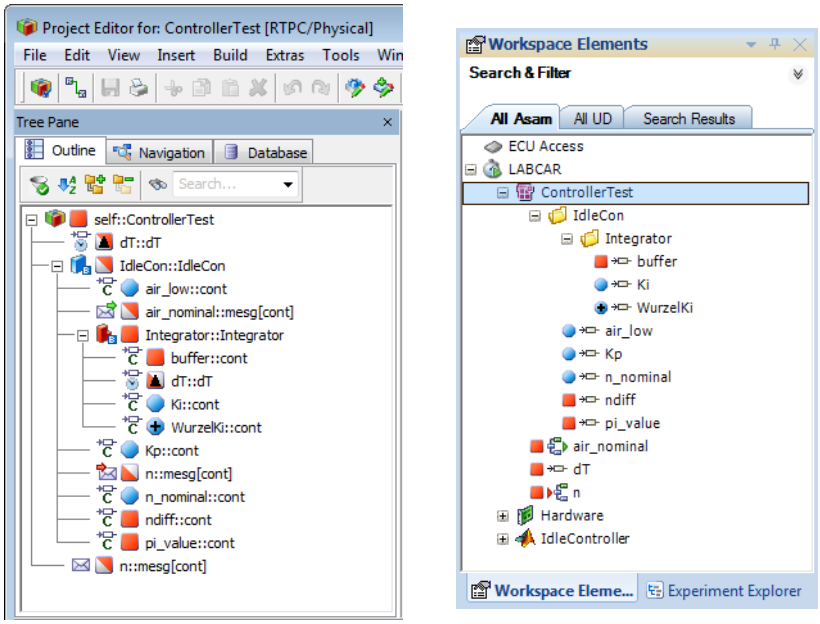


図 3-5 ASCET プロジェクト "ControllerTest"（左）と、統合されて LABCAR-EE の "Workspace Elements" ウィンドウに表示されているモジュール（右）

3.4 C コードモジュール

LABCAR-IP では、MATLAB/Simulink モデルや ASCET モジュール以外に、ハンドコーディングされた C モジュールをプロジェクトに統合することができます。

これにより、以下のようなことを実現できます。

- ユーザー定義されたあらゆる種類の機能を容易に組み込めるので、コストをかけずに柔軟なソリューションを実現できます。
- サードパーティのツールを用いて、LABCAR 用のシミュレーションモデルを作成できます。
- 既存の LABCAR のコードを再利用できます。
- 機能モックアップユニットの統合

本項では、「C コードモジュール」を作成して LABCAR プロジェクトに組み込むための以下のトピックについて、チュートリアル形式で具体的に説明します。

- C コードモジュールをハンドコーディングする（56 ページ）
- コードを記述する（63 ページ）
- C コードモジュールをオートメーションサーバーで作成する（66 ページ）
- C コードモジュールをエクスポートする（66 ページ）
- 既存の C コードモジュールを追加する（67 ページ）
- 既存の C コードモジュールを編集する（70 ページ）
- 外部 C コードをリンクする（70 ページ）
- 変数のラベル（72 ページ）
- 機能モックアップユニット（FMU）（72 ページ）

3.4.1 チュートリアル

以下に、「C コードモジュール」を用いて C コードを LABCAR プロジェクトに組み込む方法について、チュートリアル形式で説明します。

組み込みは以下の手順で行います。

- 以下のいずれかの方法で C コードモジュールのインターフェースを定義する
  - マニュアル操作で定義する
    （56 ページの「C コードモジュールをハンドコーディングする」を参照してください）
  - オートメーションインターフェースを使用する
    （66 ページの「C コードモジュールをオートメーションサーバーで作成する」を参照してください）
- 作成したインターフェースにコードを追加する

このチュートリアルでは、シンプルなコード例を用いてこれらの手順を実行します。

コード例

このチュートリアルでは、非常にシンプルなブレーキシステムの例を使用します。ブレーキシステムの入力ポートには、ブレーキペダルの設定値（0 ～ 1）が入力されます。出力ポートは 4 つの値（M_LF、M_RF、M_LR、M_RR）からなり、各車輪に影響を与えるブレーキトルクの値が出力されます。

出力ポートの値の算出方法は以下のとおりです。

まず、係数（`pedal_pressure_factor`）を用いてペダル設定値（`pedal`）を油圧ブレーキ圧に変換します。次に、別の変換係数（`pressure_torque_factor[ ]`）を用いて各ブレーキトルクを算出します。

このモジュールは、以下のような式で記述されます。

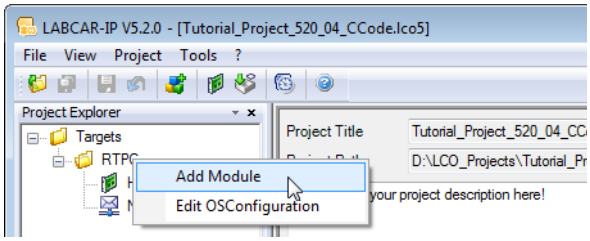
```
M_LF = pressure * pressure_torque_factor[0]
M_RF = pressure * pressure_torque_factor[1]
M_LR = pressure * pressure_torque_factor[2]
M_RR = pressure * pressure_torque_factor[3]
where
  pressure = pedal * pedal_pressure_factor
```

3.4.2 C コードモジュールをハンドコーディングする

LABCAR-IP のモジュール追加ウィザードを用いて、C コードモジュールをハンドコーディングすることができます。

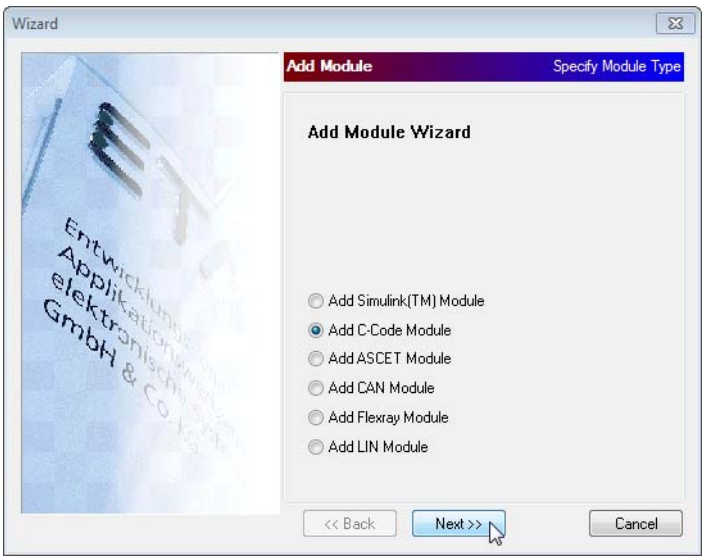
C コードモジュールを作成する：

- C コードモジュールを組み込みたい LABCAR プロジェクトを開きます。
- プロジェクトエクスプローラからターゲットを選択します。
- ショートカットメニューから **Add Module** を選択します。



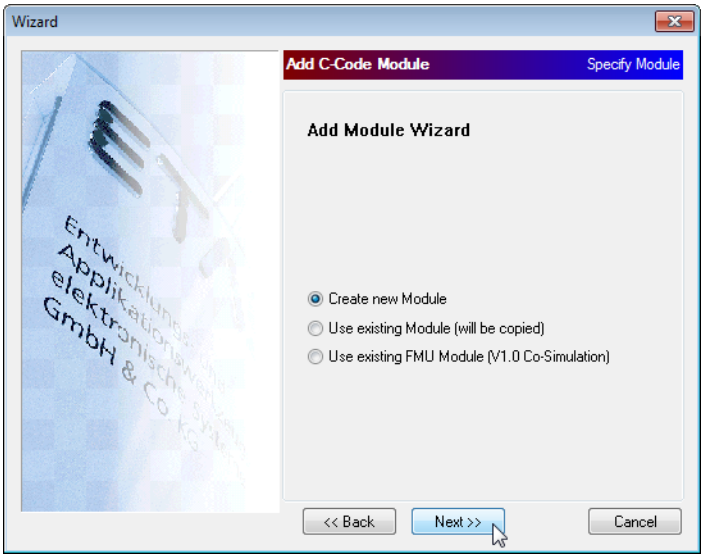
モジュール追加ウィザードが開きます。

- "Add C code Module" を選択します。

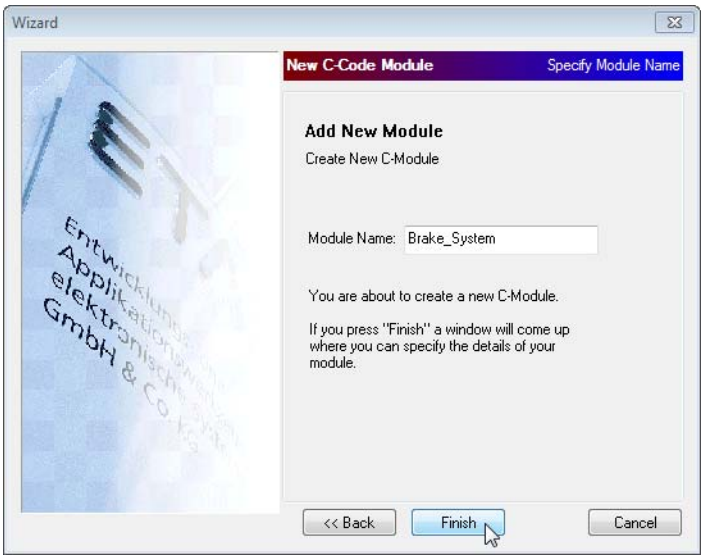


- **Next** をクリックします。

- "Create new Module" オプションを選択します。



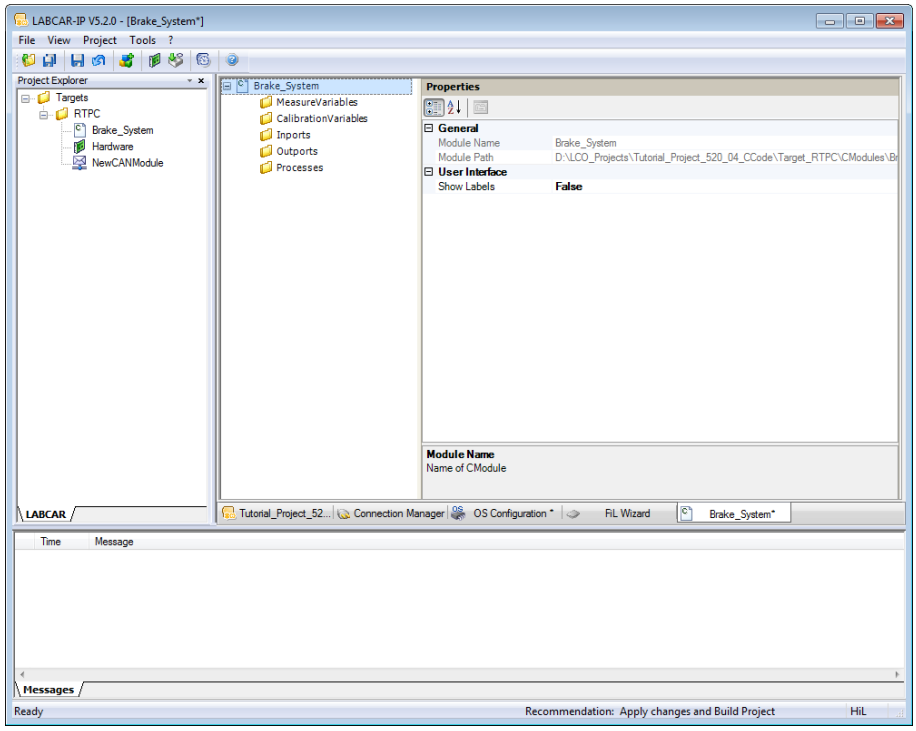
- **Next** をクリックします。
- 次のページで、作成するモジュールの名前（"Brake System"）を入力します。



- **Finish** をクリックします。

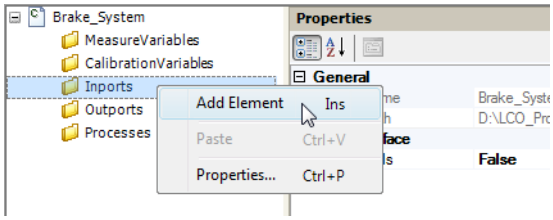
  C コードが生成され、プロジェクトエクスプローラに表示されます。

- モジュールをダブルクリックします。
  C コードエディタのタブが開きます。



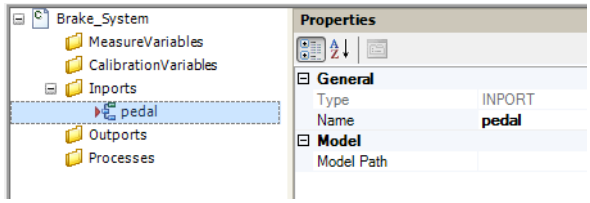
モジュールの入力ポートを定義する：

- モジュール入力ポートを定義するには、"Inports" フォルダを右クリックします。
- ショートカットメニューから **Add Element** を選択します。



"Inport" エレメントが作成されます。

- エレメントを選択し、"Properties" ウィンドウの "Name" フィールドに入力ポートの名前（"pedal"）を入力します。



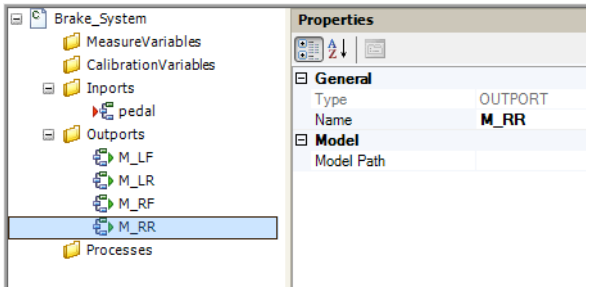
- **File → Save All** を選択します。

注記

入力ポートと出力ポートは必ず "double" データ型のスカラ値なので、他の詳細情報を入力する必要はありません。

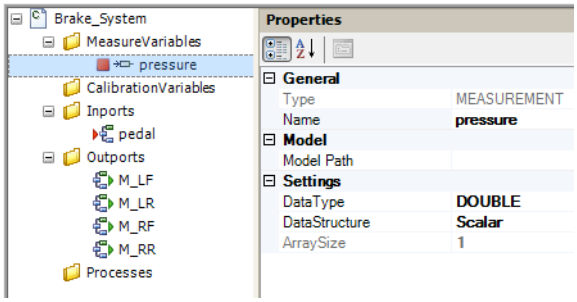
モジュールの出力ポートを定義する：

- モジュールの出力ポートを定義するには、"Outports" フォルダを右クリックします。
- ショートカットメニューから **Add Element** を選択します。
- 出力ポートの名前（"M_LF"）を入力します。
- 同じ方法、または **Copy** & **Paste** で、"M_RF"、"M_LR"、"M_RR" という名前の 3 つの出力ポートを作成します。



測定変数を定義する：

- 測定変数を定義するには、"MeasureVariables" フィールドを右クリックします。
- ショートカットメニューから **Add Element** を選択します。
- 出力ポートの名前（"pressure"）を入力します。
- "Data Type" から "DOUBLE" を選択します。
- "DataStructures" から "Scalar" を選択します。

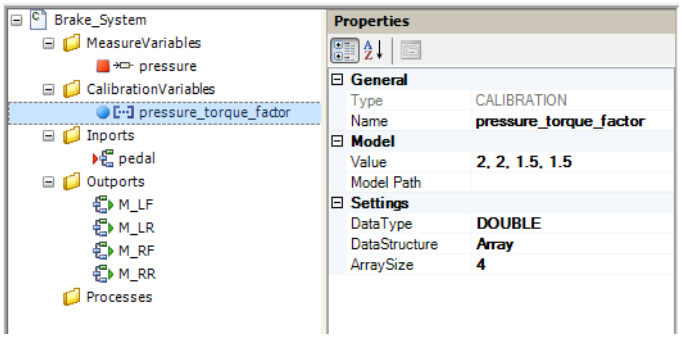


適合変数を定義する：

- 適合変数を定義するには、"CalibrationVariables" フォルダを右クリックします。

- ショートカットメニューから **Add Element** を選択します。

  適合変数の場合は、測定変数を定義する際に設定する情報に加え、値を割り当てる必要があります。



- 変数の名前（"pressure_torque_factor"）を入力します。
- データ型として "DOUBLE" を選択します。
- "Array" を選択し、"ArraySize" を 4 にします。
- 値を "Value" に設定します。値の間はカンマで区切ります。
- もう一つ "pedal_pressure_factor" という名前の適合変数を作成し、その値として 200 を設定します。

プロセス

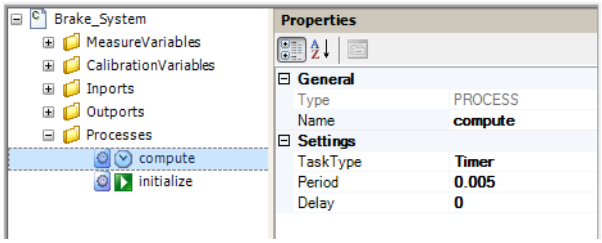
プロセスは、オペレーティングシステムにより呼び出される関数です。呼び出しのタイプはプロセスのタイプにより異なります。

- Init プロセスは、実験開始時に一度呼び出されます。
- Exit プロセスは、実験終了時に一度呼び出されます。
- Timer プロセスは、特定のタイミング（"Period" および "Delay" により定義されます）で呼び出されます。
- プロセスは必ずタスクに割り当てなければならないわけではありません。タスクに割り当てられていないプロセスは、"Event" プロセスと呼ばれます。

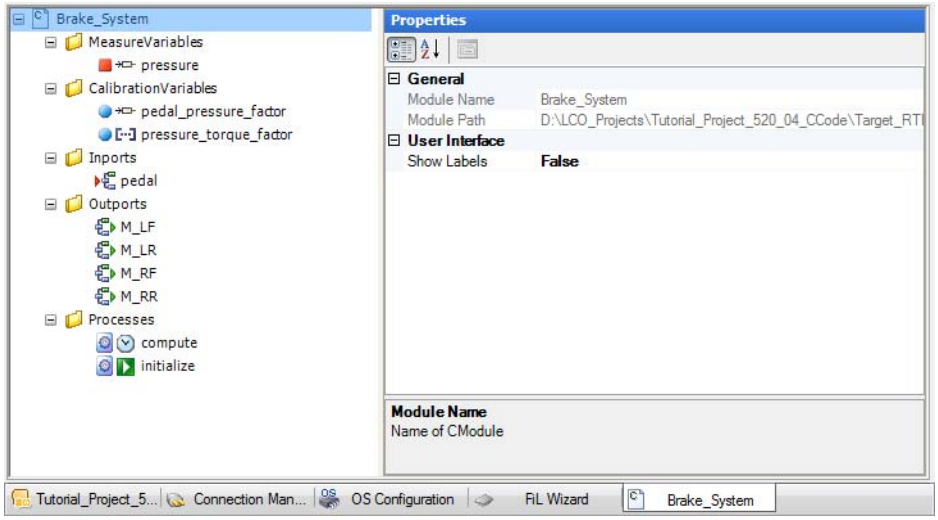
プロセスを追加する：

- プロセスを定義するには、"Processes" フィールドを右クリックします。
- ショートカットメニューから **Add Element** を選択します。
- "Process Name" にプロセス名（"initialize"）を入力します。
- この新しいプロセスをタスクに割り当てるため、**Specify Task (optional)** を選択します。
- タスクタイプとして "Init" を選択します。
- **OK** をクリックします。
- さらに、"compute" という名前のプロセスを定義します。
- "Task Type" として "Timer" を選択し、以下の値を設定します。

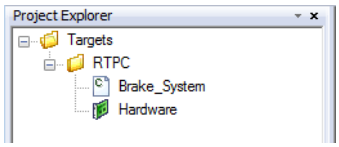
— Period = 0.005 s
— Delay = 0 s



"Brake_System" タブは、以下のようになります。



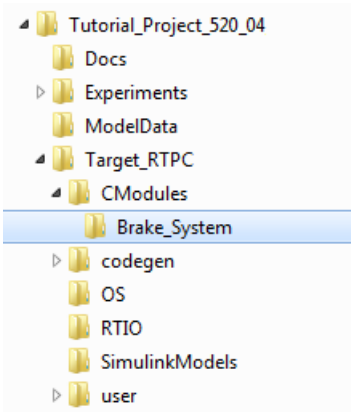
- **File → Save** を選択します。
  作成されたモジュールが保存され、プロジェクトに追加されます。



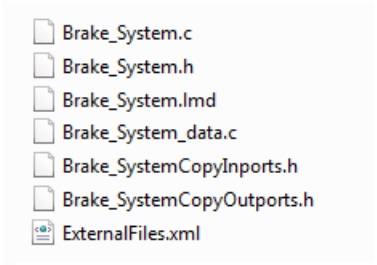
プロジェクトを保存した時点で、以下の処理が実行されます。

- データの整合性チェック
  モジュール名の妥当性、変数名やプロセス名の一意性などがチェックされます。

- ディレクトリの作成
  プロジェクトディレクトリに、"Brake_System" というモジュール用のディレクトリが作成されます。



- ファイルの作成
  上述のディレクトリに、以下のファイルが作成されます。

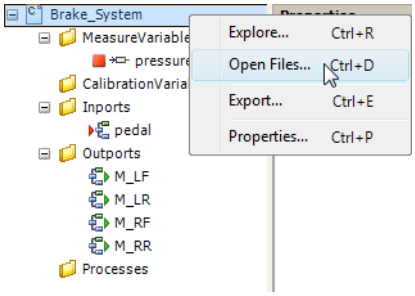


  C コードや詳細定義を含むファイル（`*.c` および `*.h`）と `*.lmd` ファイルが作成されます。
- LABCAR-OPERATOR のデータモデルへの統合

ソースファイルとインクルードファイルを編集する：

エディタ（Visual Studio など）でソース（*.c）とインクルードファイルを開くには、以下のように操作します。

- C コードモジュールのショートカットメニューから **Open Files** を選択します。



モジュール用の各ファイルが、選択されたエディタで開きます。

3.4.3 コードを記述する

ここまでの操作で作成されたコードは、Brake_System.c というファイルに格納されます。このファイルは空のプロセスが定義されたテンプレートファイルなので、ここに任意のユーザーコードを追加します。

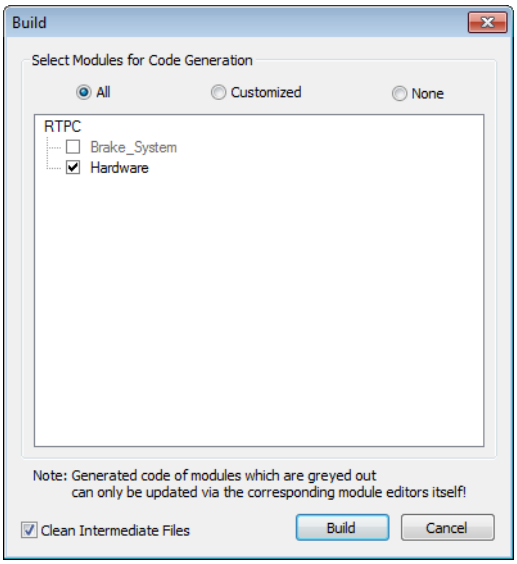
このファイルの内容を以下に示します。太字の部分が、ブレーキシステム用に記述されたコードです。

```
// 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"
}
```

ユーザーが記述する部分では、C コードモジュール作成時に定義した変数を使用できます。これらの変数は Brake_System.h というファイルで宣言されています。このコードは C コードモジュールの作成方法を説明するためのものであるため、ここでは "initialize" 関数のコードを記述する必要ありません。

ここまでで C コードモジュールの作成は終了しました。

次に、**Project → Build** を選択してプロジェクトをビルドします。

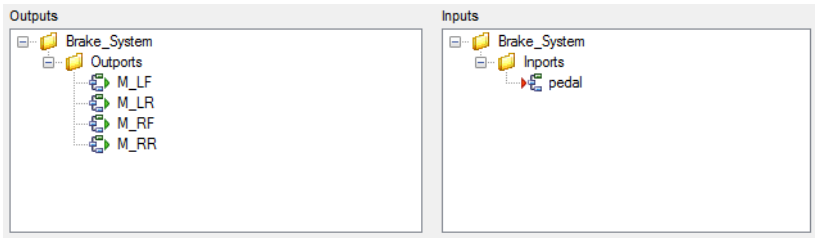


C コードモジュールについてコード生成を行う必要はないため、この "Build" ダイアログボックスにおいて C コードモジュール "Brake_System" は選択できません。**Build** をクリックしてビルドを実行します。

ビルド処理が正常に終了すると、すべての入力ポートと出力ポート、および測定変数と適合変数が LABCAR-OPERATOR ユーザーインターフェースの各ウィンドウに表示されます。

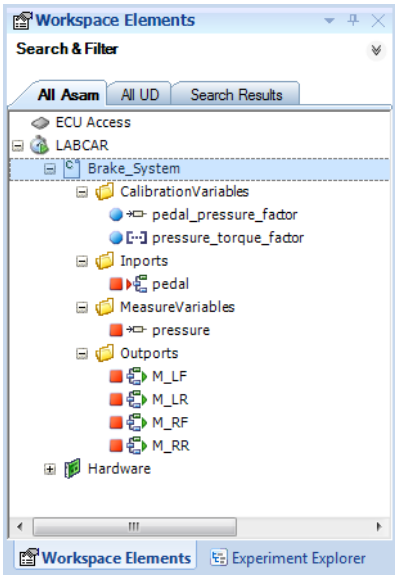
コネクションマネージャ

入力ポートと出力ポートは LABCAR-IP のコネクションマネージャに表示され、他のモジュールとの接続を行うことができます。必要に応じて、**Update Ports** を選択してください。



"Workspace Elements" ウィンドウ

LABCAR-EE で実験を行う際、入力ポートと出力ポートは、モデルの測定変数や適合変数とともに "Workspace Elements" ウィンドウに一覧表示されます。

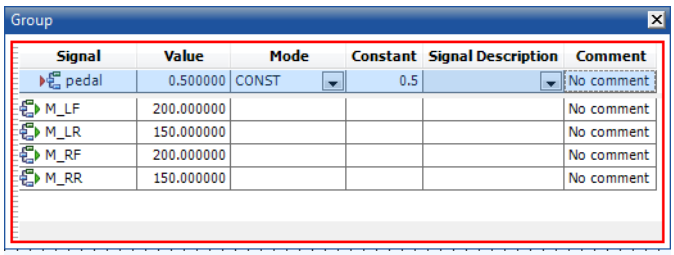


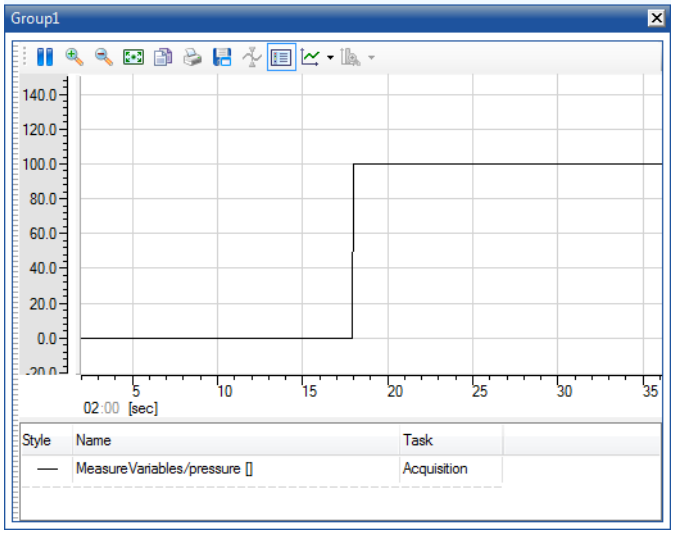
モジュールの実験を行うには、ビルドを実行し、作成された実験を実験環境で開きます。

シミュレーションを実行する：

- 信号リストを作成します。
- 入力ポートと 4 つの出力ポートを、この信号リストに追加します。
- オシロスコープを作成して、そこに測定変数 "pressure" を追加します。
- 実験を開始します。
- 入力ポート "pedal" において "Mode" を "CONST" にセットし、入力ポートに値 0.5 を代入します。

- **<Enter>** を押します。





"pressure" の値が "100" になります。2 つの前輪のブレーキトルクは 200 になり、2 つの後輪のブレーキトルクは 150 になります。

3.4.4 C コードモジュールをオートメーションサーバーで作成する

C コードモジュールの作成（入力ポート／出力ポートや変数などの追加）は、前項で行った方法以外に、API 関数を使って行うこともできます。"ICModuleManager Interface" には、これらの操作に必要なインターフェース関数が用意されています。

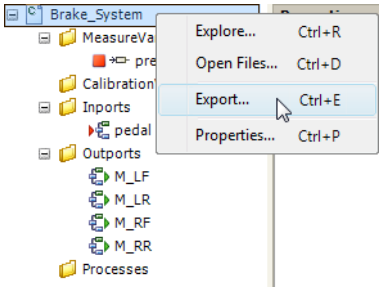
? → **Help** を選択して "API Document" を選択すると、API についての詳細なドキュメント（*.chm）が開きます。

3.4.5 C コードモジュールをエクスポートする

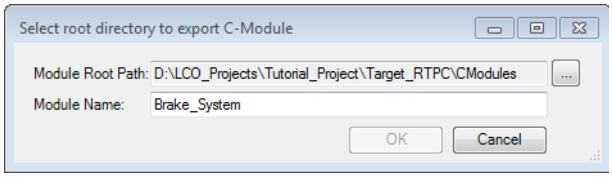
C コードモジュール（ファイルの種類などは、62 ページの「ファイルの作成」を参照してください）は、任意の名前でエクスポートし、別のプロジェクトに追加することができます。

C コードモジュールをエクスポートする：

- C コードモジュールのショートカットメニューから **Export** を選択します。



- 以下のダイアログボックスが開くので、パスを選択し、必要に応じて別の名前を指定します。



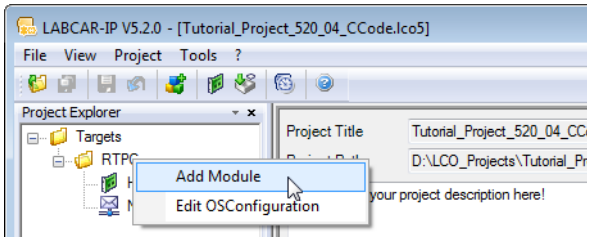
モジュールが指定の名前でエクスポートされます。

3.4.6 既存の C コードモジュールを追加する

既存の C コードモジュールをプロジェクトに追加することもできます。

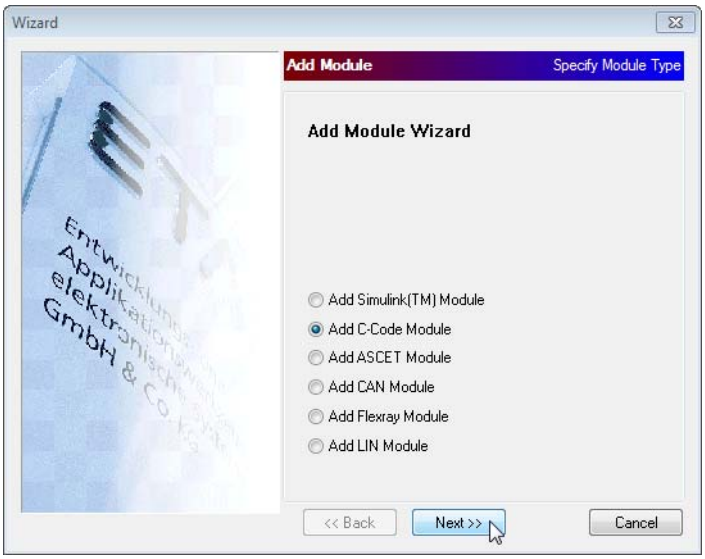
モジュールを追加する：

- C コードモジュールを追加したい LABCAR プロジェクトを開きます。
- プロジェクトエクスプローラで、"RTPC" をターゲットとして選択します。
- ショートカットメニューから、**Add Module** を選択します。

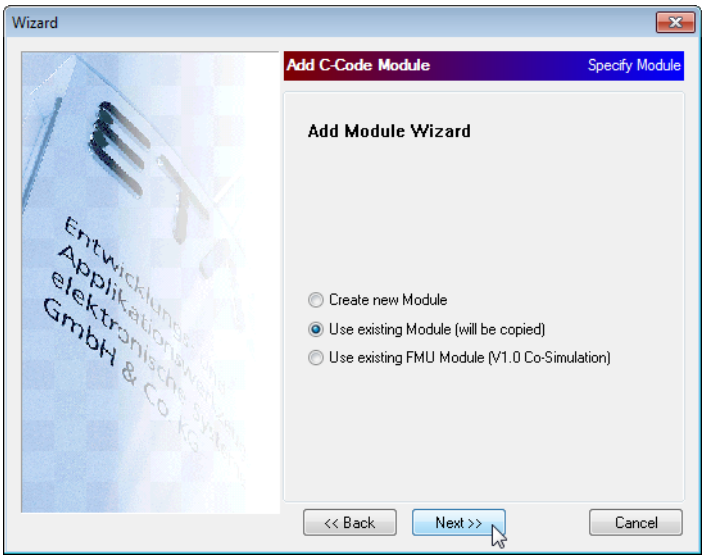


モジュール追加ウィザードが開きます。

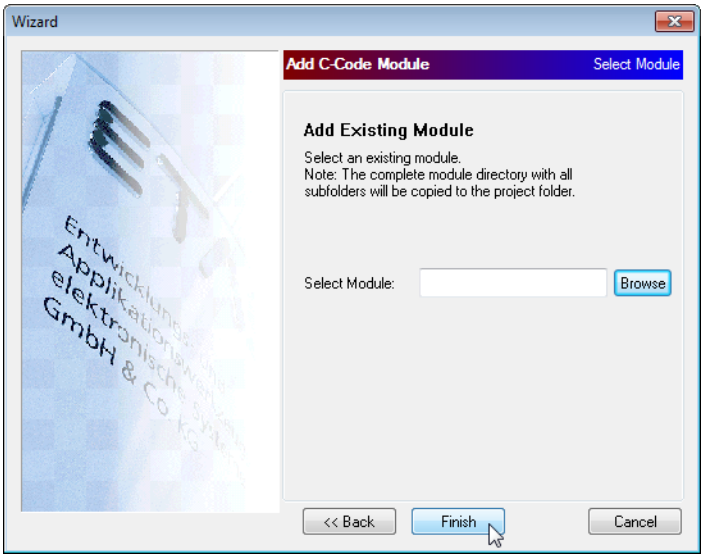
- "Add C-Code Module" を選択します。



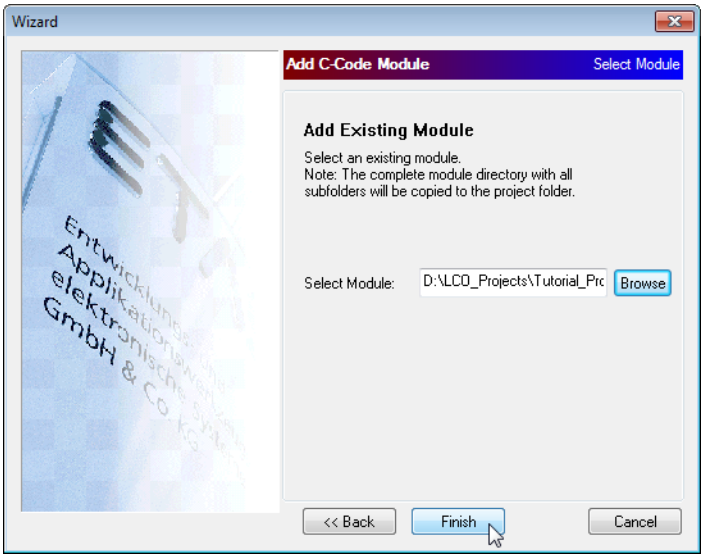
- **Next** をクリックします。
- "Use existing Module" を選択します。



- **Next** をクリックします。
  モジュールを選択するページが開きます。

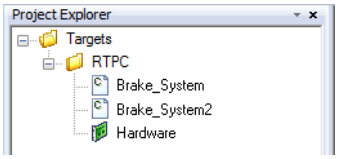


- **Browse** をクリックします。
  ファイル選択ダイアログボックスが開きます。
- ファイルを選択して **Open** をクリックします。



- **Finish** をクリックします。

"Brake-System2" というモジュールが追加され、プロジェクトエクスプローラ内の "Brake-System" という既存のモジュールと同じレベルに表示されます。



最初に作成したモジュールと同じコードをこのモジュールに追加すると、同じシミュレーション結果が得られます。ただし、LABCAR-EE の "Workspace Elements" ウィンドウ内の測定変数と適合変数のパスは異なります。

3.4.7 既存の C コードモジュールを編集する

既存の C コードモジュールを変更すると、インターフェースの変更（変数やプロセスの追加、編集、削除など）が必要となる可能性があります。

C コードを変更した後に **File → Save** を選択すると、前述の 6 個のファイルのうち 5 個が新しく作成されます。モジュールの C コードが格納されている `<Module_name>.c` というファイルに対しては、所定のルールに基づき、必要に応じてコードの追加や拡張などが行われます。

適用されるルールは以下のとおりです。

- 変数だけが変更された場合、C ファイルは変更されません。
- 新しいプロセスが追加された場合、それに対応する関数のテンプレートが追加されます。
- プロセスが削除された場合、それに対応する関数はそのまま残されます。
- プロセス名が変更された場合は、プロセスが削除されて新しいプロセスが追加されたものとして処理されます。つまり新しい空の関数テンプレートが追加されるので、古い関数のコードをこのテンプレート内にマニュアル操作で移す必要があります。

上記のどのケースにおいても、コードが自動的に削除されることは絶対にありません。つまり削除されたプロセスの関数はファイルに残りますが、その関数が呼び出されない限りコードは意味を持たないため、特に削除する必要はありません。

ただし、あるプロセスを削除した後に同じ名前のプロセスを作成した場合は例外です。この場合は同じ名前の関数が C コード内に 2 つ存在してしまうことになり、一意性が損なわれてしまうので、古いコードをマニュアル操作で削除する必要があります。

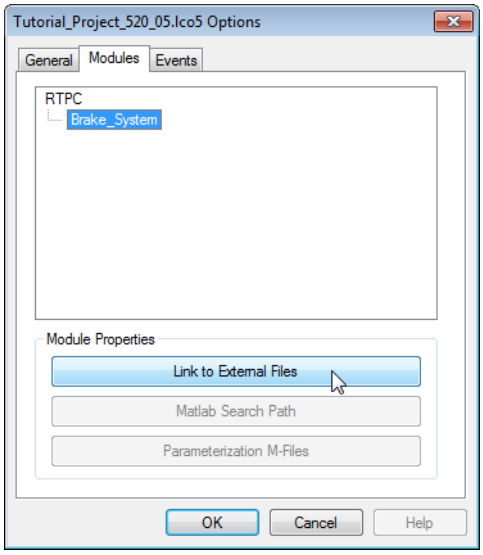
ファイル内にコードが追加される際は、必ず前もってバックアップコピーが作成されます。バックアップコピーのファイル名は、元のファイル名に連番が付いたものになります。

3.4.8 外部 C コードをリンクする

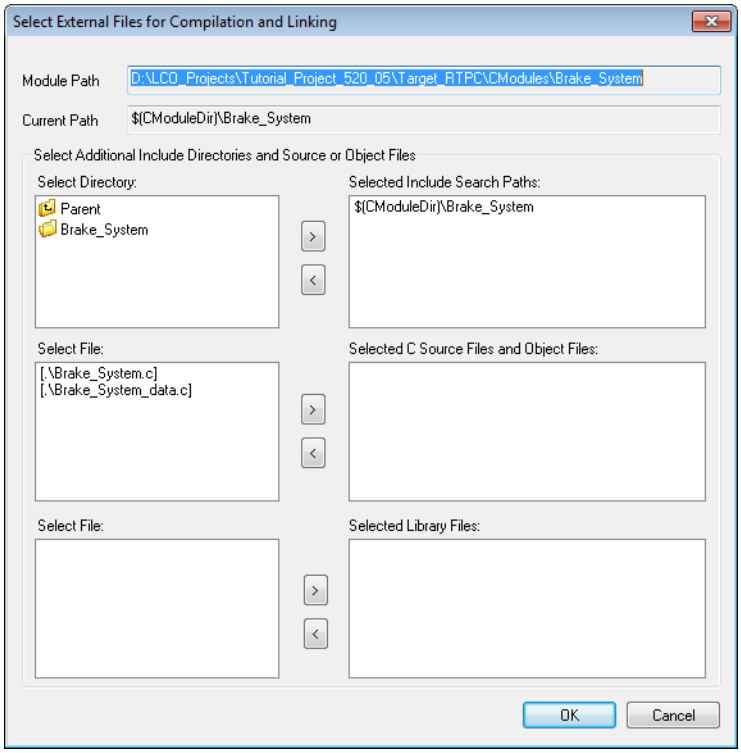
エディタで作成した C コードファイルには、ユーザー定義された C コードを追加する必要があります。この際、コードを直接挿入することもできますが、外部ソースへの参照を使用することもできます。

- LABCAR-OPERATOR のメインメニューから **Project → Options** を選択します。
- "Modules" タブを選択します。
- 外部ファイルをリンクするモジュールを選択します。

- **Link to External Files** をクリックします。



検索パス、ソースファイル、オブジェクトファイル、ライブラリを選択するためのダイアログボックスが開きます。



- 使用するファイルを選択します。
- **OK** を 2 回クリックして、2 つのダイアログボックスを閉じます。

3.4.9 変数のラベル

LABCAR-OPERATOR では、標準のラベルだけでなく、ユーザー定義のラベルも使用できます。これは、C コードモジュールをマニュアル操作で作成する場合、および API 関数により作成する場合のいずれにおいても可能です。

このラベルは、"Workspace Elements" ウィンドウ（LABCAR-EE）またはコネクションマネージャ（入力ポートと出力ポート）において、変数、入力、出力が表示されるパスとして使用されます。

ラベルを明示的に指定しない場合は、以下のデフォルトラベルが使用されます。

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

たとえば、ユーザーが測定変数用に `User/Defined/Path` というような具体的なラベルを指定すると、その変数は以下のような構成で表示されます。

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

3.4.10 機能モックアップユニット（FMU）

本項では、機能モックアップユニット（FMU: Functional Mock-up Unit）を LABCAR-OPERATOR プロジェクトにインポートする方法を説明しています。特に、Linux 用共有オブジェクトファイルを作成する際の注意事項をよくお読みください。

本項には以下の内容が含まれます。

概要

機能モックアップインターフェース（FMI: Functional Mock-up Interface）は、コシミュレーション（co-simulation: 相互シミュレーション）との標準的なツール非依存インターフェースと、タイプが異なる複数のソースとのモデル交換を定義するものです。

FMU（Functional Mock-up Unit）と呼ばれるソフトウェアライブラリを作成することにより、あるモデリングツールで作成されたモデルを異なるツール環境上で実行することが可能になります。

FMU には以下のようなタイプのものがあります。

- モデル交換（Model Exchange）
  FMU のインポート先のツールのソルバを使用します。
- コシミレーション（Co-Simulation）
  モデルのインポート元のシミュレーションツールのソルバを使用します。
  コシミュレーションには以下の 2 つのタイプがあります。
  - スタンドアロン型
    FMU に組み込まれたソルバを使用します。

— ツール交換型

シミュレーションツールのソルバを使用するメソッドを呼び出します。

注記

LABCAR-OPERATOR V5.4.0 は、スタンドアロン型コシミュレーション V1.0 をサポートしています。

各 FMU モデルは、以下のような内容の ZIP ファイル（拡張子は `*.fmu`）として提供されます。

- `sources`
  このフォルダに、モデル方程式の C 関数を含むソースファイルが含まれます。
- `binaries`
  このフォルダのサブフォルダに、各プラットフォーム用バイナリファイルが含まれます。
  — `win32 / win64`
  — `linux32 / linux64`
- `modelDescription.xml`
  モデルのすべての変数とその他のモデル情報が定義されたファイルです。
- そのほか、モデルに必要なデータ（パラメータテーブル、ユーザーインターフェースなど）のための各種フォルダ

LABCAR-IP に FMU を統合する

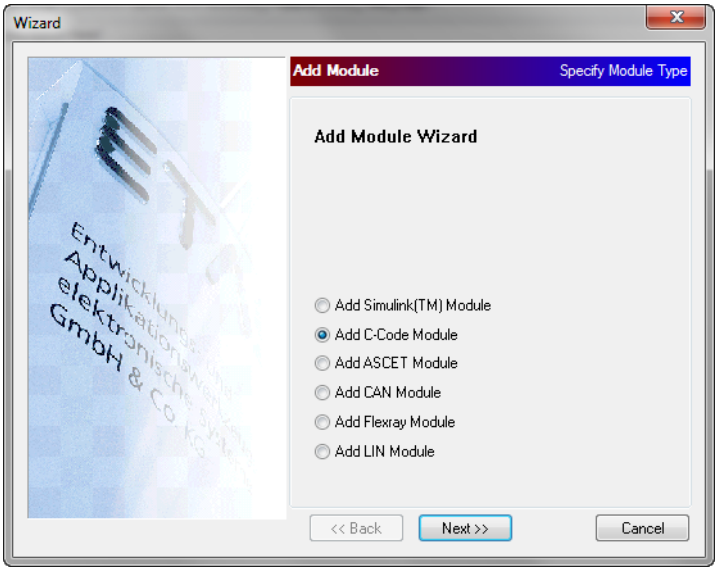
以下に、FMU を LABCAR-IP に統合する方法を説明します。

注記

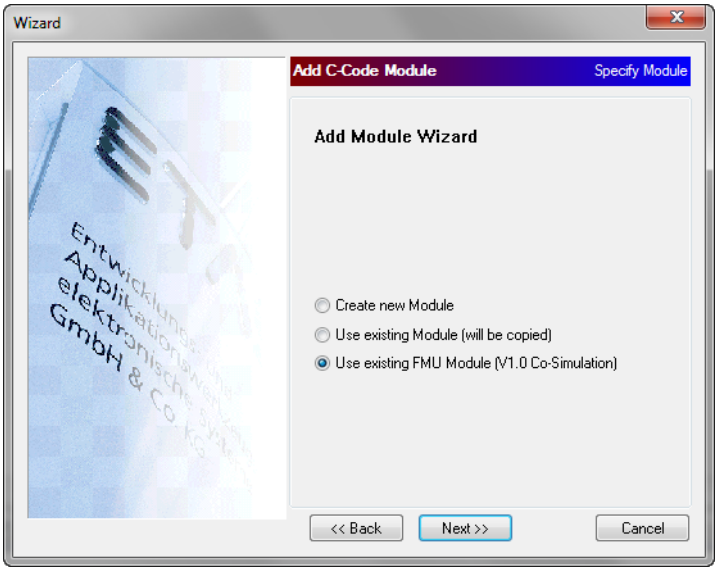
FMU モジュールを統合するには、FMU ソースファイルが必要です。以下に示された方法と異なる方法で Linux バイナリファイル（共有オブジェクトファイル）がすでに作成されていると、LABCAR-OPERATOR プロジェクトが正しく動作しない場合があります。
それを避けるため、「共有オブジェクトファイル（.so）の生成」（76 ページ）に示された方法で Linux バイナリファイルを作成することを強くお勧めします。

LABCAR-IP に FMU を統合する：

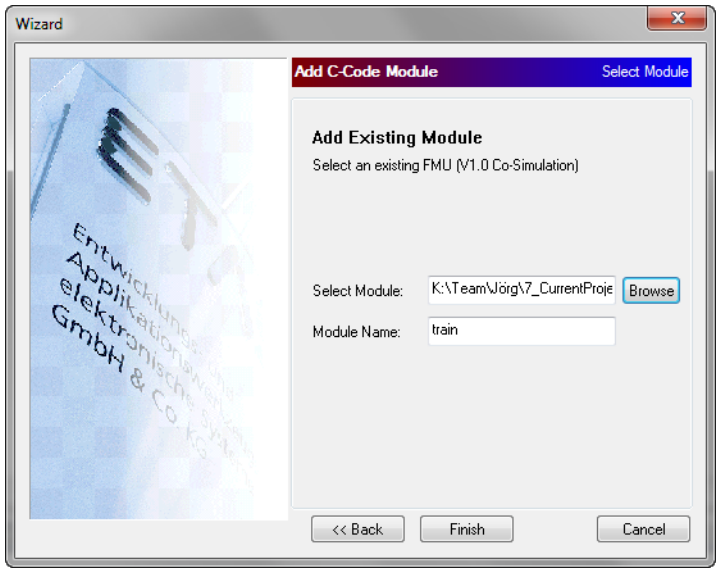
- C コードモジュールを統合するには、**Add Module** を選択します。
  モジュール追加ウィザードが開きます。



- **Add C-Code Module** を選択して、**Next** をクリックします。

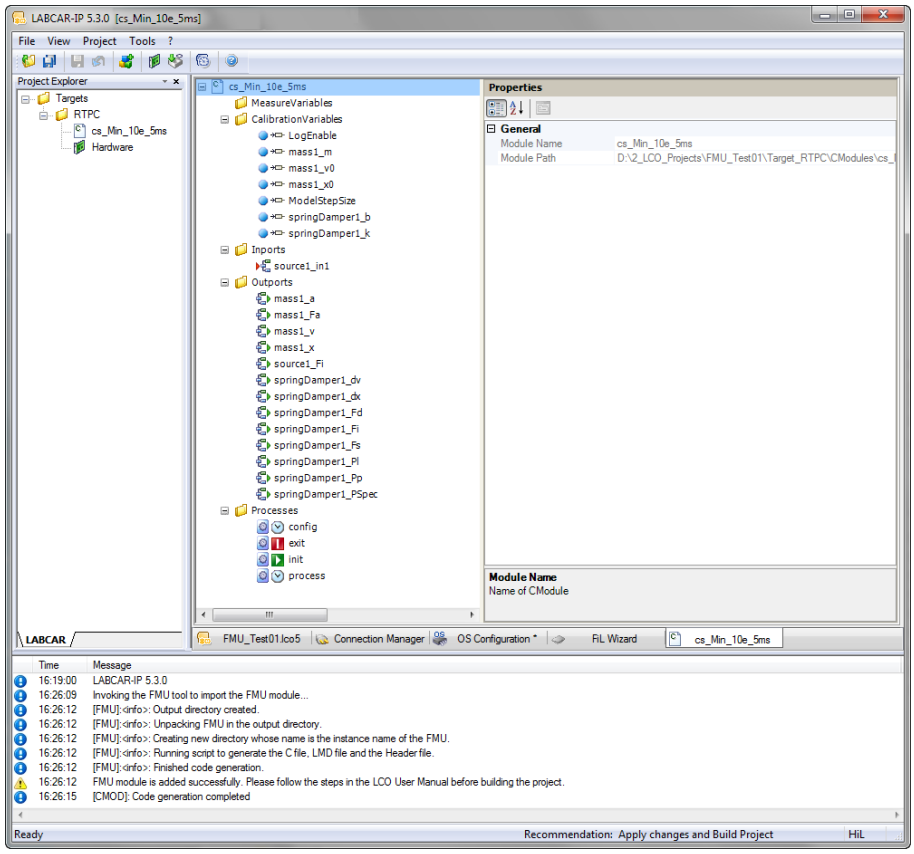


- **Use existing FMU Module** を選択して、**Next** をクリックします。



- `*.fmu` ファイルへのパスを指定し、モジュール名（有効な C 識別子）を入力します。

- **Finish** をクリックします。
  モジュールが統合され、LABCAR-IP に表示されます。



プロジェクトをビルドするには、以下のように、生成する共有オブジェクトファイル（.so）を前もってプロジェクトにコピーしておく必要があります（76 ページの「共有オブジェクトファイル（.so）の生成」を参照してください。

- .so ファイルを以下のディレクトリにコピーします。
  `Target_<TargetName>\User\lib`
  `Target_<TargetName>\runtime-data\lib`
- プロジェクトをビルドします。

共有オブジェクトファイル（.so）の生成

「共有オブジェクトファイル」は、FMU のすべての関数が含まれるバイナリファイルで、以下に示された方法でソースファイルから生成する必要があります。

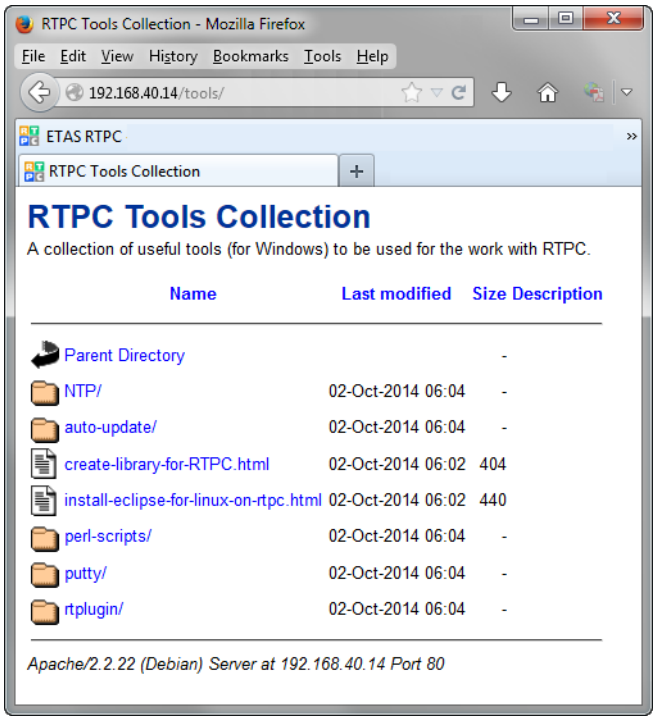
LABCAR-IP に FMU を統合すると、FMU モジュールフォルダ内のプロジェクトディレクトリに C ソースファイルがコピーされます。たとえば、LABCAR-OPERATOR プロジェクトが `D:\Temp` ディレクトリに作成されていて、FMU モジュールが "Engine" の場合は、FMU のソースファイルは以下のディレクトリにコピーされます。

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

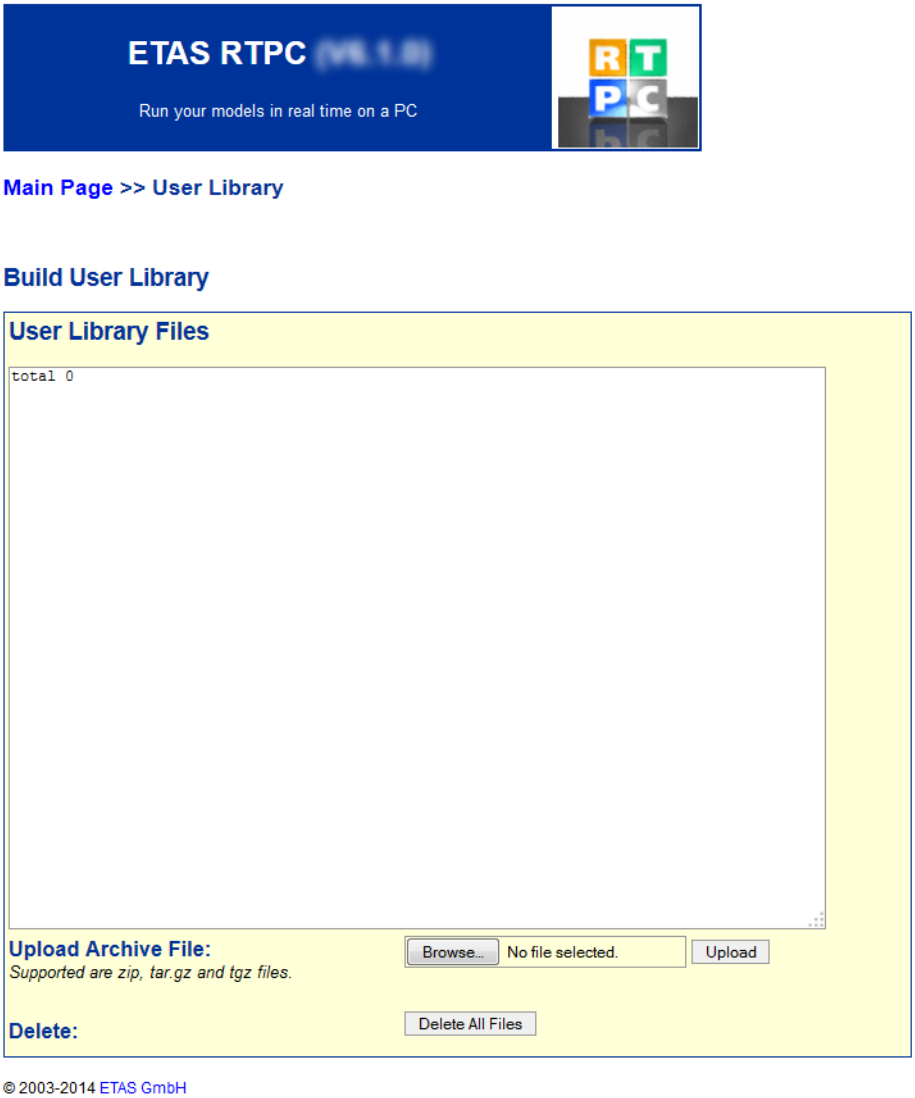
共有オブジェクトファイルを生成する：

- `FMU-Sources` フォルダ内のすべてのファイルを 1 つの ZIP ファイルに圧縮します。この際、フォルダ自体は ZIP ファイルに含めません。

  これによって Make ファイル `makefile_labcar` が正しくメインディレクトリに含まれます。
- ETAS RTPC の Web インターフェースを開きます。
- シミュレーションコントローラを停止します。
- メインページに戻り、**Tools** をクリックします。



- **create-library-for-RTPC.html** というリンクをクリックします。



- **Delete All Files** をクリックします。
- **Choose File** をクリックします。
- 作成してあった ZIP ファイル（`FMU_Sources.zip`）を選択します。
- **Upload** をクリックして、ファイルを Real-Time PC にアップロードします。
- "Build User Library" の **Build** をクリックします。

  ビルド処理が正常に終了すると、生成されたソースオブジェクトファイルが "Download Library Files" セクションに表示されます。

  ビルド時にエラーが発生した場合は、ログウィンドウにメッセージが出力されます。詳しくは 79 ページの「トラブルシューティング」を参照してください。

- ソースオブジェクトファイルを PC にダウンロードするには、"Download Library File" セクション内のリンクをクリックします。

注記

Make ファイル `makefile_labcar` はテンプレートとして提供されているもので、`FMU_Sources` フォルダ内のすべての C ファイルが考慮されており、複数の FMU を統合するための各種フラグも含まれています。
また、ビルド処理で使用するコンパイラフラグ、各種定義、インクルードディレクトリなどを追加したり、コンパイル時に特定の C ファイルが除外されるようにしたりすることが必要となる場合があります。

トラブルシューティング

質問： ソースファイルに `fmiFunctions.h` ファイルまたは `fmiPlatformTypes.h` ファイルが含まれていません。どこから入手できますか？

答え： JModelica のホームページからダウンロードできます。

`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

質問： どのようにすれば変数の名前を変更できますか？

答え： 変数の名前は、以下のようにして変更することができます。

- "Properties" ウィンドウで変数名を変更します。
- `Target_<TargetName>/CModules/instanceName` フォルダ内の `<FMU_InstanceName>_FMU.c` ファイルを開きます。
- `cmod_process_<InstanceName>FMU()` メソッドに含まれる変更前の変数名を探して、新しい名前に変更します。

（例）コードに以下のような行が含まれていて、変数名を "value" から "valueone" に変更した場合：

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

この行を以下のように変更する必要があります。

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

注記

プロジェクトをビルドする前に、必ず LABCAR-OPERATOR プロジェクトディレクトリ下の `FMU_Sources` フォルダが削除されていることを確認してください。

質問： FMU_Sources.zip ファイルがアップロードされると、以下のメッセージが出力されず、

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

ビルド処理中に以下のメッセージが出力されます。

```
Missing Arguments
```

答え： \FMU_Sources フォルダ内のソースファイルを圧縮した ZIP ファイルが正しい ZIP ファイルではないことが原因である可能性があります。圧縮処理が正しく実行された場合は、圧縮終了後に Web インターフェースの "Build User Library" セクション（**Main Page → User Library**）にその内容が表示されます。

質問： ビルド処理を実行すると、コンパイラのエラーメッセージが出力され、処理が中断されます。

答え： ビルド処理が中断される原因としては、さまざまなものが考えられます。FMU が作成されたツールに応じて、以下のいずれかの対策を試してみてください。

注記

一般的には、FMU の作成者による Make ファイルを参考にして Make ファイル makefile_labcar を調整していくことをお勧めします。

対策 1: FMU が MapleSim® で作成されたものであり、ビルド処理がコンパイラエラーにより中断した場合は、ソースファイルの中に fmuTemplate.c というファイルが含まれている可能性があり、他のファイルが #include 文でこのファイルを使用していると、エラーが発生する可能性があります。

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:
```

このエラーを解消するには、Make ファイルの内容を以下のように変更してください。

- Make ファイルから以下の行を削除します。

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

- Make ファイルの OBJECTS 変数から以下の部分を削除します。

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

Make ファイルの変更内容は、以下のようになります。

変更前：

```
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)
```

変更後：

```
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)
```

Make ファイルの変更後もエラーが発生する場合は、ヘッダファイルが不足している可能性があります。その場合は、FMU の作成者から入手してください。

対策 2: FMU が SimulationX® で作成されたものであり、以下のようなエラーメッセージが出力された場合は、Make ファイル内の定義文に不足がある可能性があります。

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'
```

CFLAGS 変数に SimulationX® ツール用の以下の定義が含まれいないと、エラーが発生する可能性があります。

```
-DITI_CVODE_EXT
```

このエラーを解消するには、Make ファイルの内容を以下のように変更してください。

- 以下の変数を追加します。

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

質問： .so ファイルのビルド中に、以下のエラーメッセージが出力されます。

```
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'
```

答え： このエラーメッセージは、ソースコードに `#define FMI_COSIMULATION` というマクロが含まれていないことを示しています。

メインソースファイル（通常は FMU の名前が付いた C ファイル）を開き、ファイルの先頭部分に以下のようなマクロを追加してください。

```
/* 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
```

質問： 共有オブジェクトファイルのビルド処理は正常に終了しましたが、複数の FMU を TLABCAR-OPERATOR プロジェクトに追加すると、ビルト処理が正常に終了しません。

答え： このエラーは、互いに異なる FMU で作成されたメソッド間の矛盾点が原因である可能性があり、これを解消するにはソースファイルを調整する必要があります。

\FMU_Source ディレクトリに fmiFunctions.h というファイルが含まれていれば、この調整は LABCAR-IP によって自動的に行われます。

このファイルが存在しない場合は、以下のような手操作による調整が必要です。

- "fmiInitializeSlave" という文字列を含むヘッダファイルを探します。
- このファイルを開き以下の部分を変更します。
  `#define DllExport`

変更前：

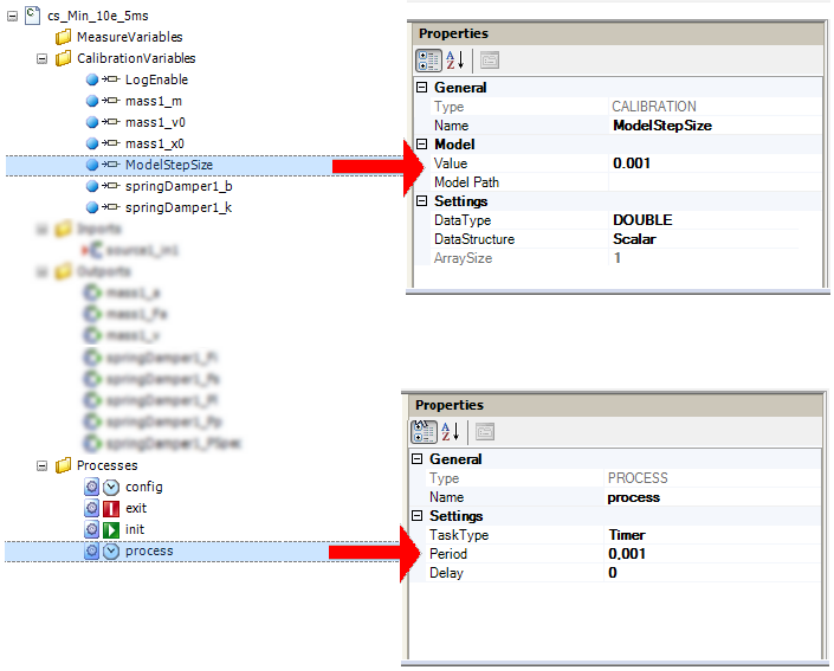
```
/* Export fmi functions on Windows */
#ifdef _MSC_VER
#define DllExport __declspec( dllexport )
#else
#define DllExport
#endif
```

変更後：

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

制限事項に関する注意点

- **共有オブジェクトファイルの命名規則**
  共有オブジェクトファイルの名前には、"lib" とモジュール名を合わせたものは使用できません。また文字は小文字しか使用できません。
  例： プロジェクトに "gear" および "Engine" という 2 つのモジュールが含まれる場合は、`libgear.so` および `libEngine.so` という名前は使用できません。
- **タスク周期とモデルのステップ幅**
  "process" プロセスの周期は、モデル（Type: "CALIBRATION"）のステップ幅と同じである必要があります。



- **適合変数（Calibration Variables）**
  実験を実行する前に、実験環境において各適合変数の値をそれぞれ定義しておく必要があります。実験の実行中にはこれらの値は変更できません。
- **入力ポートと出力ポート**
  "string"、"boolean"、"enum" 型の入力ポートと出力ポートは、LABCAR-IP でサポートされていないため、無視されます。また配列型のものも無視されます。LABCAR-IP で FMU 用にサポートされているのは、スカラ型のものだけです。
- **適合変数と測定変数**
  LABCAR-IP は、文字列型と enum 型の適合変数と測定変数をサポートしていません。

3.5 CAN モジュール（LABCAR-NIC ー CAN ネットワークの統合）

LABCAR-OPERATOR V5.4.0 のアドオン製品である LABCAR-NIC V5.4.0（Network Integration CAN）は、CAN 通信を含む ECU ファンクションのテストを容易に行うための機能を提供するものです。

CAN バス通信の仕様を 1 つまたは複数の CANdb ファイルから読み込み、その中から実際に存在する CAN ノード（複数選択可）を UuT（**U**nit **u**nder **T**est：テスト対象）として選択すると、CAN バス内の残りの部分をシミュレートするためのコードが LABCAR-NIC V5.4.0 によって自動的に生成されます。生成されたコードにはユーザーコード（カウンタ、チェックサム計算など）を追加することができます。

選択されたメッセージに使用される信号は、コネクションマネージャに表示され、ここでモデル入力（受信メッセージ）やモデル出力（送信メッセージ）に接続できます。また各信号は、すべての OLC 情報と共にシグナルセンターにも表示されます。

シミュレートされるメッセージについては、LABCAR-EE 上に専用のインストゥルメント（GUI）が作成され、CAN モニタの機能を利用できます。

LABCAR-NIC のバージョン 5.0 より、J1939 ネットワークプロトコルがサポートされています（119 ページの「ユーザー定義の C コード」を参照してください）。

ハードウェア要件

LABCAR-NIC V5.4.0 を使用するには、ETAS RTPC V6.2.0 がインストールされた Real-Time PC が必要で、さらにその PC に 1 枚以上の CAN ボード（IXXAT 社の iPC-I XC16/PCI または CAN-IB200/PCIe）がインストールされている必要があります。

CAN ボードを RTIO（LABCAR-RTC V5.4.0）に組み込む必要はなく、CAN ボードのコンフィギュレーションは、99 ページの「バス」を参照して設定してください。

3.5.1 概要

ここでは、4 つのノード（Node A、B、C、D）と CAN メッセージ 1 ～ 5 を含む CAN ネットワークを例に、LABCAR-NIC の機能概要を説明します。CANdb ファイルには、図 3-6 に示されるような通信仕様が記述されているものとします。

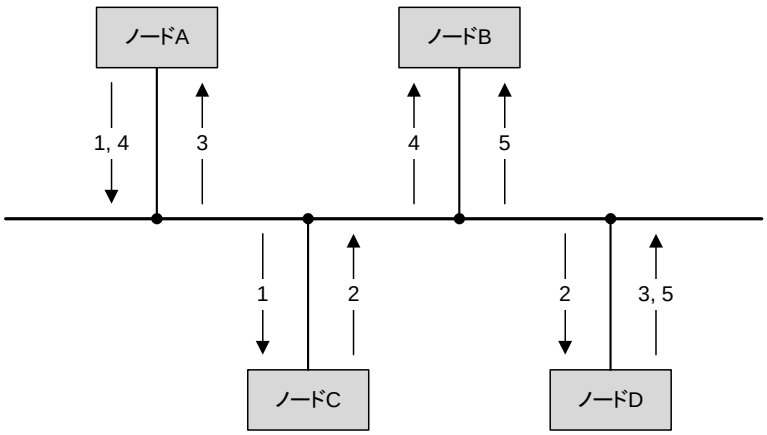


図 3-6 4 つのノードと 5 個のメッセージが含まれる CAN ネットワーク

これら 4 つのノードのうちの 2 つ（C と D）は実際に存在するので、これらを UuT とします。この UuT のテストを行うには、3 個のメッセージ（メッセージ 1、3、5）をシミュレートする必要があります。

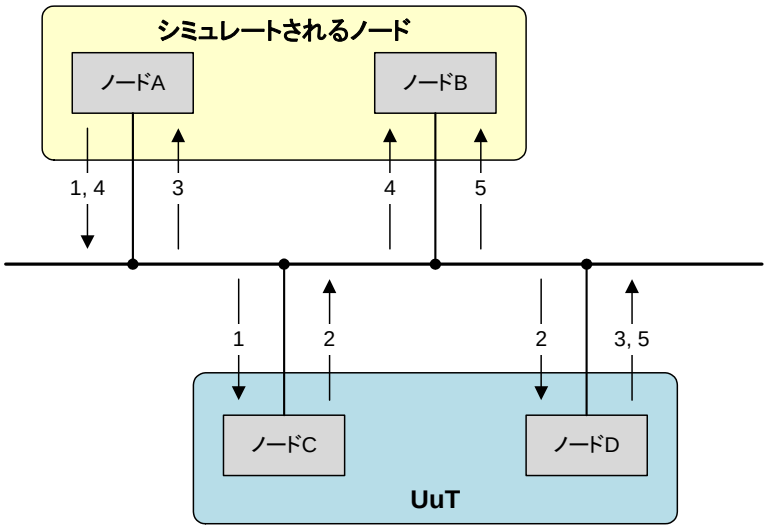


図 3-7 物理的に存在するノード（UuT）とシミュレートされるノード

メッセージ 4 は、シミュレートされるノード間で送受信されるものであるため、UuT の通信には関係しません。このメッセージと UuT を除いた部分、つまりシミュレーションのためのコードが必要な部分は「UuT 外のバス」（"residual bus"）と呼ばれます。

この部分からは UuT に対してメッセージ 1 が送信され、また UuT からこの部分にメッセージ 3 および 5 を送信する必要があります。

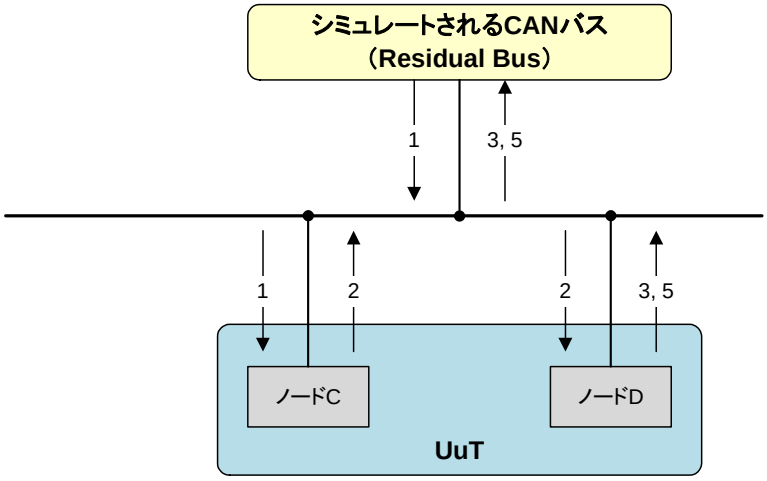


図 3-8 シミュレートされる架空のバス（"Residual Bus"）との間で送受信されるメッセージ

3.5.2 J1939 プロトコルのサポート機能

ネットワークプロトコル J1939 は、SAE（Society of Automotive Engineers）の各種規格をベースに策定されたものです。J1939 は、商用車のドライブトレインやシャシーなどに使用される CAN バス上の通信を規定するもので、ISO11898 に準拠する CAN 高速通信を用いた物理層を使用します。

J1939 には以下のような特徴があります。

- CAN 2.0B に基づく 29 ビット識別子（拡張フォーマット）を使用
- ピアツーピア通信とブロードキャスト通信をサポート
- 最大 1785 バイトのデータ転送が可能なトランスポートプロトコルである BAM（Broadcast Announce Message）および CMDT（Connection Mode Data Transfer）をサポート
- 分散型ネットワーク管理
- 情報は「パラメータ」（「シグナル」）と呼ばれ、複数のパラメータ（シグナル）が、ユニークな番号（PGN: Parameter Group Number）で識別されるパラメータグループにまとめられて転送される

J1939 に対応するための LABCAR-NIC V5.4.0 の拡張機能

LABCAR-NIC V5.4.0 は、J1939 規格に準じたバスディスクリプションを処理します。これには以下のような機能が含まれます。

- J1939 のインタープリテーションスキームに基づく 29 ビットメッセージ識別子の扱い
- ネットワーク管理とデータ転送プロトコル
- J1939 用バスディスクリプションのインポート

注記

この機能を使用するには、CANalyzer V6.0 以降に対応する "J1939 PG (ext. ID)" フォーマットの DBC ファイルが必要です。完全な 29 ビット識別子は、このフォーマットでのみ使用されます。

- バスシミュレーション用パラメータの設定
- Real-Time PC に組み込まれた IXXAT CAN ボード用のコード生成

大きなサイズの CAN メッセージの扱い

LABCAR-NIC V5.4.0 は、プロトコルで定義された最大ペイロード長 1785 バイトをサポートしています。そのような長いメッセージを LABCAR-IP の CAN エディタで作成して編集することができ、コネクションマネージャにおいて標準の CAN メッセージと同様に扱うことができます。

注記

CAN コントローラが J1939 をデコードできるようにするには、CAN エディタにおいて、そのコントローラに対して長い（8 バイト以上の）J1939 メッセージ（送信メッセージ、受信メッセージ、ゲートウェイメッセージのいずれか）を定義する必要があります。

可変長の CAN メッセージは、現時点においてはサポートされていません。

多重セッションのサポート（J1939-21）

長いメッセージの受信を可能にするため、多重セッションがサポートされています。同時に受信できるメッセージ数について、特に上限は定義されていません。

ただし、個々の CAN インターフェースは同時に 1 つしか長いメッセージを送信できません。複数の長いメッセージの送信が重複してスケジュールされると、メッセージは 1 つずつ順に送信されます。複数の長いメッセージを同時に送信するには、メッセージごとに異なる CAN インターフェースを使用する必要があります。

サイクルタイム（送信周期）

長いメッセージの周期的送信をスケジュールするには、各メッセージの先頭パケット間の時間をサイクルタイムとして定義します。

例：

- メッセージ 0 のパケット 0 が送信されるタイミング : t_{00}
- メッセージ 0 のパケット 1 が送信されるタイミング : t_{01}
- メッセージ 0 のパケット 2 が送信されるタイミング : t_{02}

  .....

- メッセージ 1 のパケット 0 が送信されるタイミング : t_{10}

上記の場合、サイクルタイムは t_{10} - t_{00} です。

3.5.3 CAN エディタ

この項では、CAN バスシミュレーションの仕様を設定するための CAN エディタの使用方法について説明します。CAN エディタは、メインワークスペースの "CAN Editor" タブ上に表示されます。

CAN エディタを開く：

- プロジェクトエディタで、エディタで開きたい CAN モジュールをダブルクリックします。

  メインウィンドウの "CAN Editor" タブに CAN モジュールが表示されます。

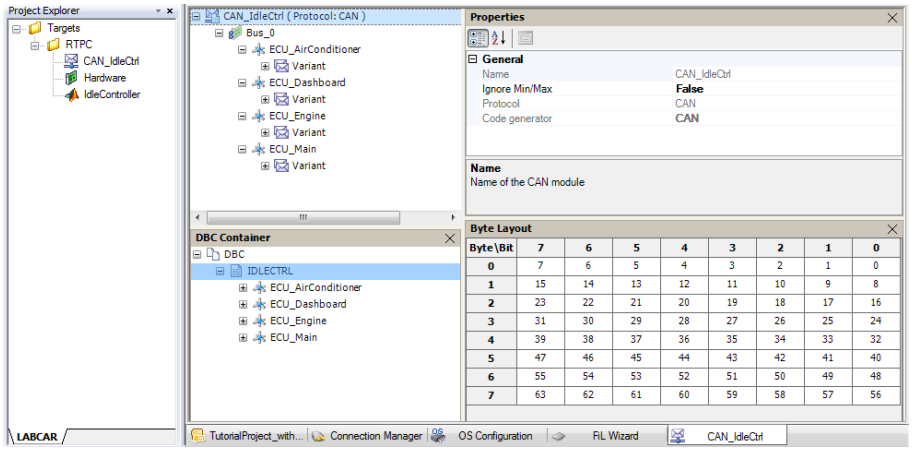


図 3-9 CAN エディタ

CAN エディタには以下のものが表示されます。

- CAN ネットワーク
- DBC コンテナ
- プロパティ（"Properties" ウィンドウ）
- フレームのバイトレイアウト（"Byte Lauout" ウィンドウ）

CAN ネットワークペイン

ボードとコントローラから構成されたネットワークの階層が表示され、各種下位エレメントが含まれます。

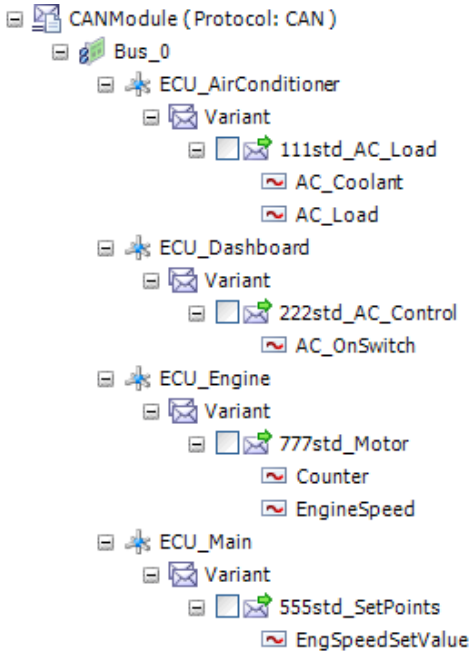


図 3-10 CAN ネットワーク

ネットワーク構成についての詳細は、88 ページの「CAN エディタ」を参照してください。

"Properties" ウィンドウ

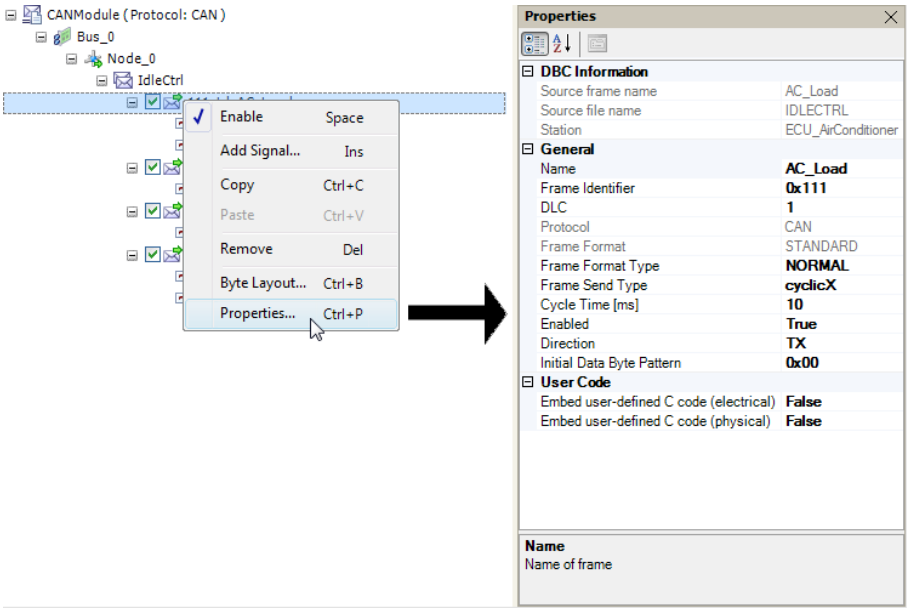
ネットワークペイン内のエレメントを選択すると、そのエレメントのプロパティが "Properties" ウィンドウに表示されます。編集可能なプロパティは、ここで直接編集できます。

プロパティが自動的に表示されない場合は、以下のように操作して表示できます。

プロパティを表示する：

- プロパティを表示したいエレメントを右クリックします。

- ショートカットメニューから **Properties** を選択します。



選択されたエレメントのプロパティが "Properties" ウィンドウに表示されます。

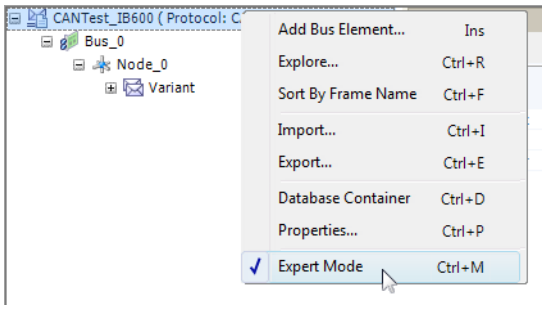
プロパティの編集についての詳細は、88 ページの「CAN エディタ」を参照してください。

"Expert Mode" オプション

"Properties" ウィンドウでは、デフォルトにおいては、通常使用する一般的なプロパティのみが表示されます。すべてのプロパティを表示するには、"Expert Mode" オプションをオンにします。

エキスパートモードに切り替える：

- CAN ネットワークの最上位エレメントを右クリックします。
- ショートカットメニューから **Expert Mode** を選択します。



CAN ネットワークの各エレメントのプロパティについては、98 ページの「CAN ネットワークのコンポーネント」を参照してください。

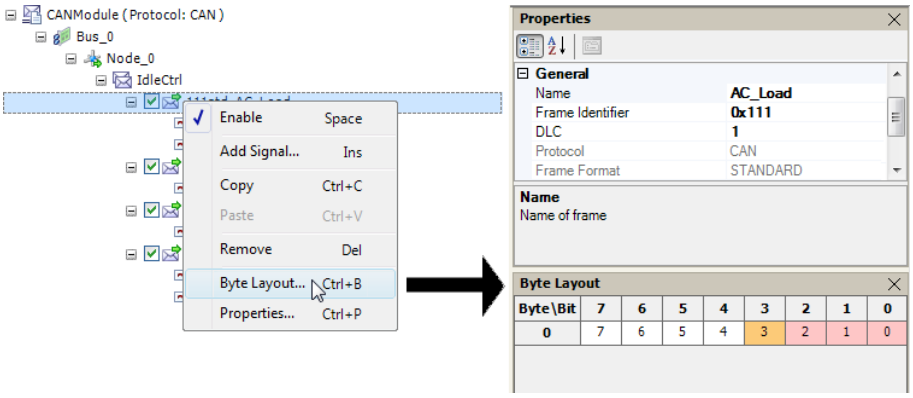
フレーム（バイトレイアウト）ウィンドウ

ネットワークペイン内で選択されているフレームの内容が表示されます。

バイトレイアウトを表示する：

- レイアウトを表示したいフレームを右クリックします。
- ショートカットメニューから **Byte Layout** を選択します。

  フレームのバイトレイアウトが表示されます。



3.5.4 CAN ネットワークの編集

CAN ネットワークを編集するには、ネットワーク内の編集対象のエレメントを右クリックしてショートカットメニューを開き、そこに含まれるコマンドを使用します。ネットワークに含まれるエレメントについての詳細は、98 ページの「CAN ネットワークのコンポーネント」を参照してください。

同種のエレメントを複数選択（ハイライト表示）してからショートカットメニューを開くと、それらのエレメントについて同時に同じコマンドを実行することもできます。

エレメントを追加する：

- 追加したいエレメントの上位のエレメントを右クリックして、ショートカットメニューから **Add (Element)** を選択します。

注記

パート、フレーム、信号の名前に使用できるのは、ANSI-C 識別子として使用できる文字に限られます。

エレメントを削除する：

- 追加したいエレメントを右クリックして、ショートカットメニューから **Remove** を選択します。

エレメントをコピーする：

- 追加したいエレメントを右クリックして、ショートカットメニューから **Copy** を選択します。
- 追加先の上位エレメントを右クリックして、ショートカットメニューから **Paste** を選択します。

エレメントのプロパティをコピーする：

- 追加したいエレメントを右クリックして、ショートカットメニューから **Copy** を選択します。

- 追加先のエレメントを右クリックして、ショートカットメニューから **Paste** を選択します。

エレメントを移動する：

- 移動したいエレメントを移動先の上位エレメントまでマウスでドラッグ & ドロップします。

注記

上記の移動操作は、状況に応じて行えない場合があります。その場合は、ショートカットメニューに含まれる同機能のコマンドも、グレイアウトされているか、または表示されていません。

プロパティの編集

CAN ネットワーク内の各エレメントのプロパティは、"Properties" ウィンドウで編集します。プロパティの中には自動的に決定されるものがあり、それらは "Properties" ウィンドウ上ではグレイアウトされます。

"Properties" ウィンドウを開く：

- 対象のエレメントを右クリックして、ショートカットメニューから **Properties** を選択します。

  CAN エディタ内に "Properties" ウィンドウが表示されます。

複数のエレメントのプロパティを同時に編集する：

ネットワーク内の同じタイプのエレメントを複数選択すると、それらのエレメントで同じ設定が行われているプロパティの内容が表示されます。

- "Properties" ウィンドウを開きます。
- **<Ctrl>** を押し下げたまま、設定したい複数のエレメントを選択します。

注記

複数エレメントの同時編集において、上位のエレメント内に含まれるエレメントを特定するプロパティ（例：フレーム内の信号名）が変更されると、同じ上位エレメント内の別のエレメントのプロパティにも変更後の値が設定される、という内容のエラーメッセージが出力される場合があります。

検証

CAN モジュールを保存する際には、CAN ネットワークの妥当性確認が行われます。その際には、たとえばコード生成される内容が CAN 仕様に準じているか、などがチェックされます。エラーメッセージはログウィンドウに表示されます。

注記

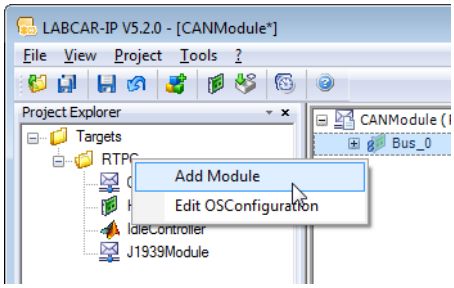
上記のエラーメッセージを無視すると、プロジェクトのビルド処理においてコンパイラエラーが発生したり、予期しないランタイム挙動が生じる可能性があります。

3.5.5 CAN ネットワークの作成

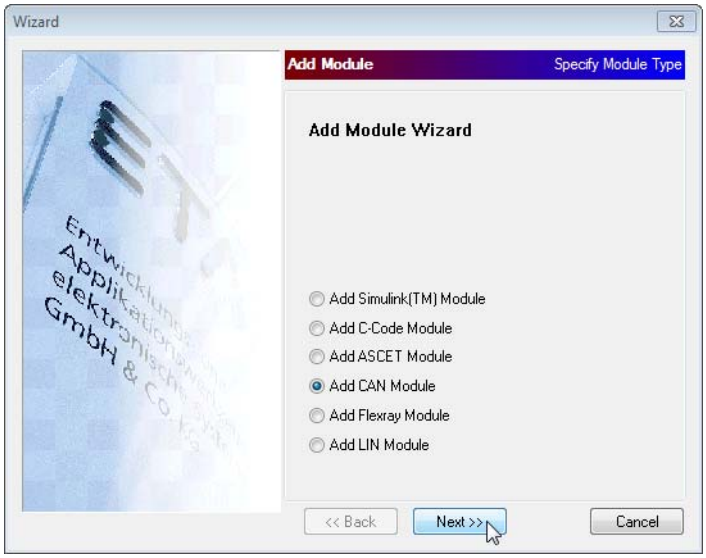
本項では、CANdb ファイルをインポートして CAN ネットワークを作成する方法について説明します。

CAN モジュールを 追加する：

- プロジェクトエクスプローラで "RTPC" ターゲットを右クリックします。
- ショートカットメニューから **Add Module** を選択します。

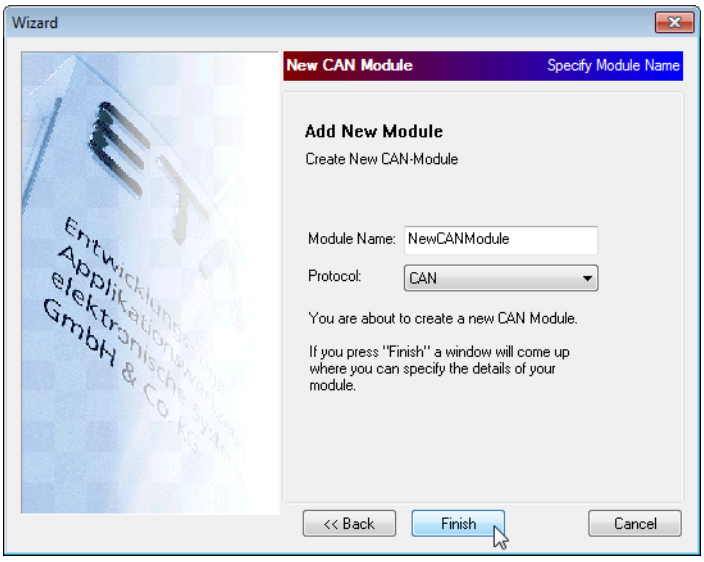


モジュール追加ウィザードが開きます。



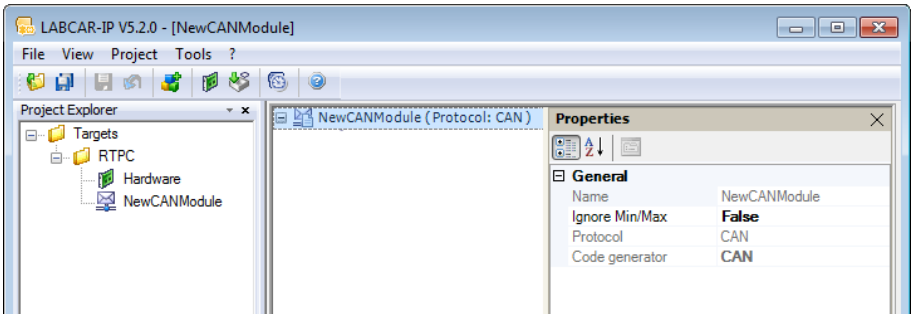
- **Add CAN Module** を選択して **Next** をクリックします。

- モジュール名を入力し、プロトコル（"CAN" または "J1939" を選択します。



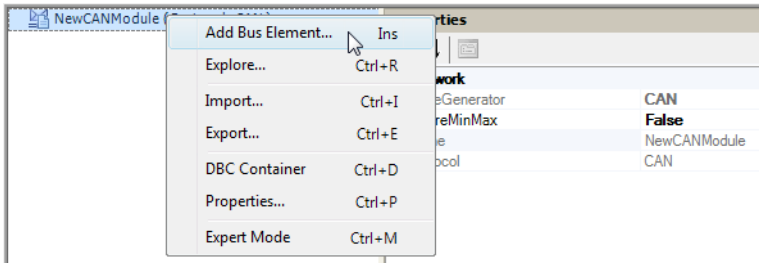
- **Finish** をクリックします。

  新しい CAN モジュールが作成され、エディタ上に表示されます。

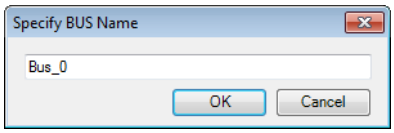


バスを追加する：

- CAN モジュールを 右クリックします。
- ショートカットメニューから **Add Bus Element** を選択します。



- バスエレメントの名前を入力して **OK** をクリックします。



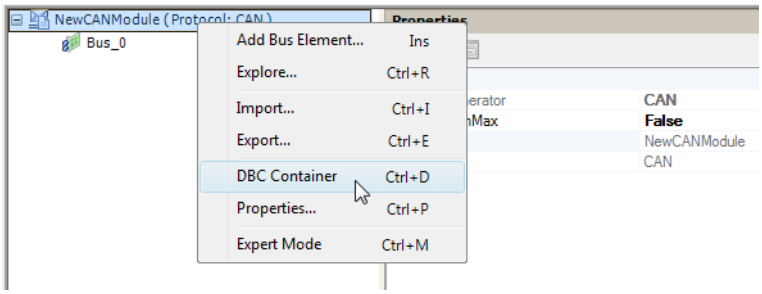
バスエレメントが作成されます。

- **File → Save All** を選択します。

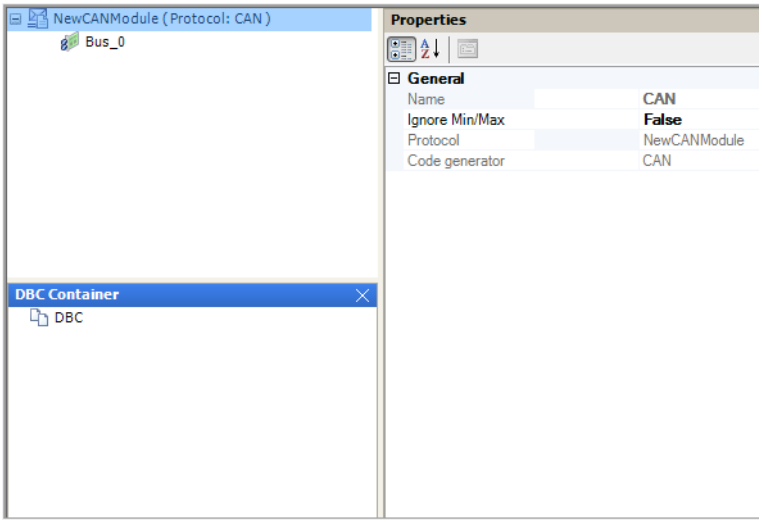
CAN モジュールが作成されたので、91 ページの「CAN ネットワークの編集」の方法で編集するか、または DBC ファイルを読み込みます。

DBC コンテナを開く：

- CAN モジュールを選択します。
- ショートカットメニューから **DBC Container** を選択します。



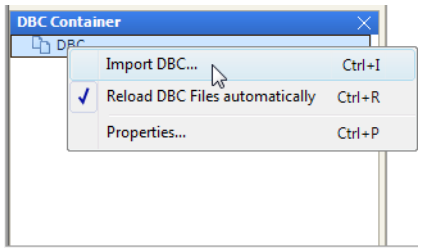
"DBC Container" ウィンドウが開きます。



DBC ファイルをロードする：

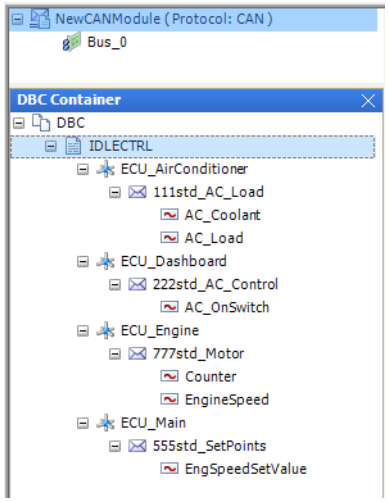
- DBC コンテナ内の "DBC" エレメントを選択します。

- ショートカットメニューから **Import DBC** を選択します。



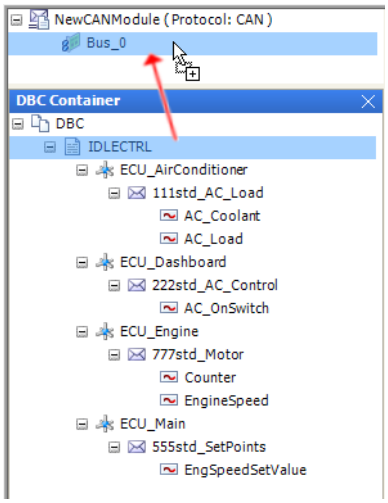
- DBC ファイル（*.dbc）を選択して、**OK** をクリックします。

  ファイルに含まれる情報が DBC コンテナにインポートされます。



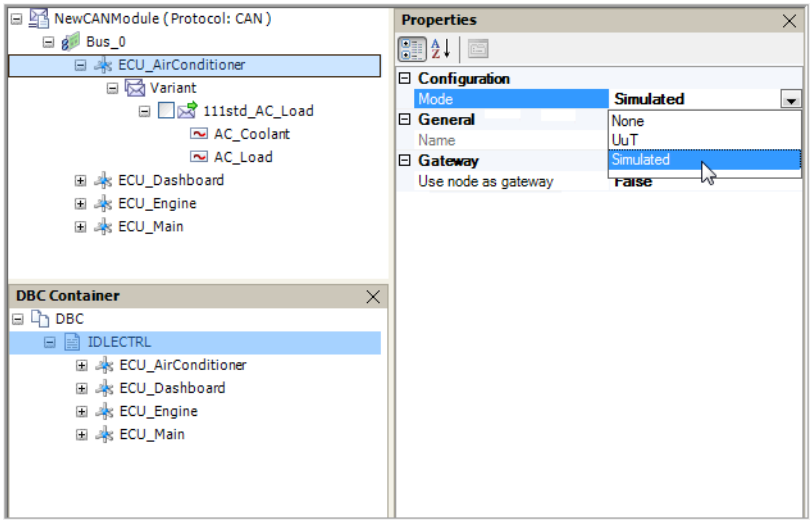
DBC をバスに追加する：

- インポートされた DBC を、バスエレメントにドラッグ & ドロップします。



DBC ファイルに保存されていたすべてのノードがバスに追加されます。さらに、追加されたノードとフレームの間の階層に、パート（"Variant"）が 1 つずつ追加されます。

- 各ノードを選択し、プロパティウィンドウの "Configuration" / "Mode" で、そのノードが実在するか（"UuT"）、またはシミュレートされるものであるか（"Simulated"）を指定します。



CAN コンフィギュレーション内でのドラッグ& ドロップ

エレメントをマウスでドラッグ & ドロップした際に、コピーにすでにエレメントが存在していた場合、以下の 2 とおりが行われます。

1. バージョニンの割り当て
   以下の場合、バージョン番号が付加されて、操作が完了します。

— **DBC File → Bus**

同名のノードが存在する場合

— **Node → Bus**

同名のノードが存在する場合

— **Frame → Part**

同名のフレームが存在する場合

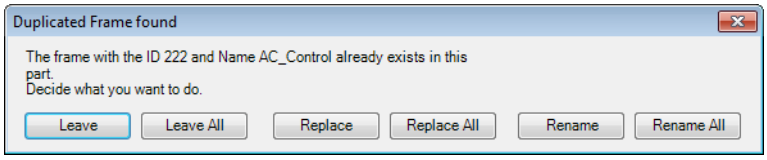
— **Signals → Frames**

同名の信号が存在する場合

2. 操作確認

— **Node → Node/Part**

ソース側ノードのフレームのうち、ターゲット側のノート／パート内に存在しないものだけがコピーされます。同一フレーム[1] が見つかると、そのたびに以下の確認が行われます。



各ボタンで以下の処理が行われます。

- **Leave**
  フレームはインポートされません。**Leave All** をクリックすると、存在する他のフレームはすべてインポートされません。
- **Replace**
  存在するフレームは、インポートされるフレームに置き換えられます。**Replace All** をクリックすると、存在する他のすべてのフレームに対して同じ処理が行われます。
- **Rename**
  存在するフレームにバージョン番号が割り当てられてインポートされます。**Rename All** をクリックすると、存在する他のすべてのフレームに対して同じ処理が行われます。

3.5.6 CAN ネットワークのコンポーネント

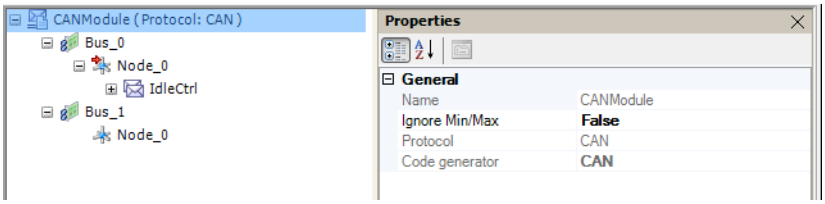
本項では、LABCAR-IP の CAN ネットワークの以下のコンポーネントについて説明します。

- CAN ネットワーク（99 ページ）
- バス（99 ページ）
- ノード（104 ページ）
- パート（107 ページ）
- フレーム（109 ページ）
- 信号（シグナル）（115 ページ）

[1] ID/PGN と名前が同じ場合、両フレームは「同一」とみなされます。

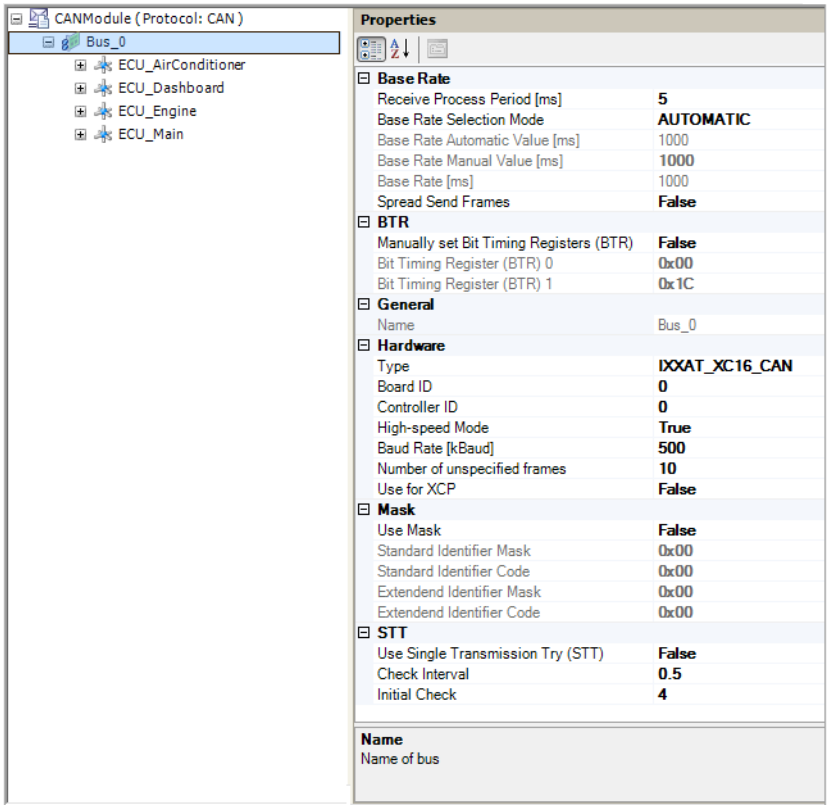
CAN ネットワーク

CAN ネットワークを選択すると、そのプロパティが "Properties" フィールドに表示されます。



- **Name**
  CAN モジュール（CAN ネットワーク）の名前
- **Ignore Min/Max**
  信号の定義済み上下限値を無視することができます（118 ページの「最小値と最大値」を参照してください）。
- **Protocol**
  CAN または J1939（ネットワーク管理を含む）
- **Code generator**
  使用するコードジェネレータ

バス



Base Rate:

コントローラの **Base Rate** （基本レート）プロパティは、選択された CAN モジュールコントローラと LABCAR-OPERATOR プロジェクトとの間の通信レートを定義するものです。

すべての CAN 送信フレームは、最高速のタスク（例：1ms 周期のタスク）を基準に処理されます。この「基本タスク」の周期の整数倍のタイミング（n*t）で 1 個のフレームの送信タスクがトリガされます。

これを一例をあげて説明します。たとえば、3 個の送信フレームがあり、その送信周期がそれぞれ 5ms、10ms、2ms であったとします。これらすべてのメッセージを送信できるようにするには、基本タスクの起動周期は各周期の最大公約数、つまり 1ms となります。

実験の実行中にフレームの送信周期を自動設定されたレート以外の周期に変更する必要が生じた場合は、この周期をマニュアル操作で変更することができます。

自動設定（"Automatic"）の場合は各周期の最小公分母が算出され、これが基本レートとして使用されます。"Automatic 2x" または "Automatic 4x" が選択されていると、自動的に算出された基本レートがさらに 2 または 4 で除算され、"Spread Send Frame" オプションがオンの場合、この周期で送信フレームが分散されて送信されます。

- **Receive Process Period [ms]**
  バスからの受信処理を呼び出す周期です。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。
- **Base Rate Selection Mode**
  Base Rate（基本レート）を決定する方法を選択します。

  — **MANUAL**（マニュアル）

  — **AUTOMATIC**（自動）

  自動計算では、2 倍または 4 倍のオーバークロッキング（**AUTOMATIC2x** / **AUTOMATIC4x**）を選択できます。
- **Base Rate Automatic Value [ms]**
  自動計算された基本レートの値です。選択されているコントローラに含まれるアクティブフレームで使用されているすべての周期の最大公約数から、モードに応じて決定されます。
- **Base Rate Manual Value [ms]**
  任意に入力できる基本レートの値です。
- **Base Rate [ms]**
  現在使用されている基本レートです。

- **Spread Send Frames**
  CAN カードの送信バッファには限られた数のフレームしか格納できないため、送信フレームが蓄積されるような送信設定になっている場合、送信フレームの一部が消失してしまう可能性があります。

  たとえば、10ms ごとに 5 個のフレームが送信され、さらに 20ms ごとに 5 つ、100ms ごとに 5 個のフレームが送信される場合、n × 20ms（n = 0, 1, 2,...）ごとにフレームが蓄積されることになり、特に n × 100ms（n = 0, 1, 2,...）においては蓄積量が著しく増大します。

  このようなフレームの蓄積によるデータ損失を防ぐには、**Spread Send Frames** オプションを使用して、各フレームが少しずつ遅れて送信されるようにします。

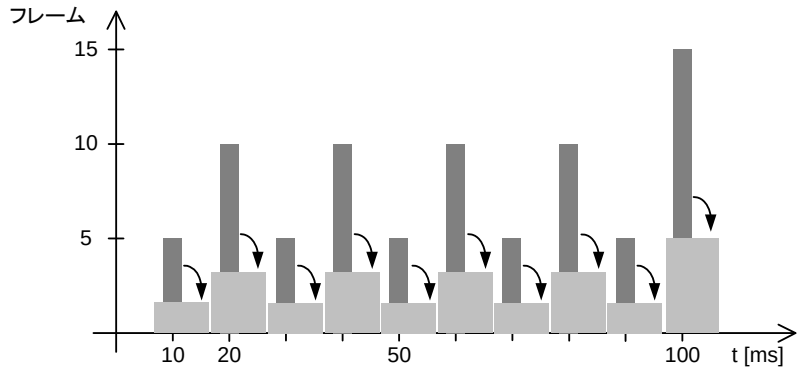


図 3-11 "Spread Send Frames" オプションの効果

BTR:

- **Manually set Bit Timing Registers (BTR)**
  このオプションを **True** にすると、コントローラのビットタイミングレジスタ（BTR: Bit Timing Registers: ）に直接アクセスすることが可能になり、**Baud Rate [kBaud]** はグレイアウトされます。
- **Bit Timing Register (BTR) 0**
  ビットタイミングレジスタ 0
- **Bit Timing Register (BTR) 1**
  ビットタイミングレジスタ 1

注記

各 BTR レジスタの内容についての詳細な情報は、CAN コントローラのデータシートをご参照ください。

CAN-FD - Arbitration Bit Rate:

- **CAN-FD - Arbitration Bit Rate [kBaud]**
  コントローラのアービトレーションビットレート（単位 : kBaud、デフォルト : 500）
- **Arbitration Timing Mode**
  アービトレーションビットレートのタイミングモード
- **Time Segment 1**
  タイムセグメント 1 の期間（単位 : TIME QUANTA）
- **Time Segment 2**
  タイムセグメント 2 の期間（単位 : TIME QUANTA）

- **Re-synchronisation Jump Width**
  再同期ジャンプ幅（単位 : TIME QUANTA）
- **Transceiver Delay Compensation**
  トランシーバ遅延補正のオフセット（単位 : TIME QUANTA、0= 無効）

CAN-FD - Fast Data Bit Rate:

- **CAN-FD - Fast Data Rate [kBaud]**
  コントローラの高速データレート（単位 : kBaud、デフォルト : 2000）
- **Arbitration Timing Mode**
  アービトレーションビットレートのタイミングモード
- **Time Segment 1**
  タイムセグメント 1 の期間（単位 : TIME QUANTA）
- **Time Segment 2**
  タイムセグメント 2 の期間（単位 : TIME QUANTA）
- **Re-synchronisation Jump Width**
  再同期ジャンプ幅（単位 : TIME QUANTA）
- **Transceiver Delay Compensation**
  トランシーバ遅延補正のオフセット（単位 : TIME QUANTA、0= 無効）

General:

- **Name**
  バスの名前

Hardware:

- **Type**
  使用する CAN ボードのタイプ（"IXXAT_XC16_CAN" / "IXXAT_IB200_CAN" / "IXXAT_IB600_CANFD"）
- **Board ID / Controller ID**
  ボードとコントローラのユニークな識別子
- **High-speed Mode**
  CAN コントローラを高速インターフェースとして定義するためのプロパティ
- **Baud Rate [kBaud]**
  CAN コントローラの転送レート
- **Number of unspecified frames**
  実験においてまだ定義されていない受信可能フレームの数。LABCAR-EE の "Workspace Elements" タブに、これらのフレームに対応する測定変数が作成されます。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。
- **Use for XCP**
  コントローラを、XCP に使用するなどの理由により CAN バスのシミュレーション部分から除外するためのプロパティ

Mask:

- **Use Mask**
  メッセージに適用するマスクを設定します。このオプションが **True** になっていると、標準 ID と拡張 ID 用にそれぞれマスクとコードを定義することができます。

注記

ここに設定する値や、Acceptance Code Register と Acceptance Mask Register についての詳細は、CAN コントローラのデータシートをお読みください。

SST:

エラー処理に関するプロパティを設定します。

- **Use Single Transmission Try (SST)**
  コントローラの動作モード "Single Transmission Mode" の有効化／無効化を行います。この動作モードにおいては、通信エラーが発生すると、送信フレームはキューに登録されません。

  この動作モードは、コントローラが完全な CAN ネットワークに接続されていない場合（接続された ECU が一時的に電源オフ状態になっているなど）に役立ちます。

  通常、コントローラはすべてのフレームを、送信が正常に終了するまでキューに入れておきます。もしも一時的に電源オフになっていた ECU の電源が再投入されると、一度に 50 個の CAN フレームがすべてコントローラの送信バッファ内に蓄積されて連続的に送信されるため、フレーム間のタイミングが失われてしまう可能性があります。
- **Check Interval**
  CAN バスのステータスをチェックする間隔を浮動小数点の秒単位で指定します。チェック時にバスの不正な状態が検出されると、リブートが行われます。

  この間隔を 0 にセットすると、チェックは行われません。
- **Initial Check**
  初期化の失敗を検出するまでのチェック回数を指定します。最小値は 2 で、デフォルト値は 4 です。

  "Check Interval" で指定された間隔でステータスチェックが行われ、ここで指定された回数分のチェックが行われてもステートが "Initialization" のままであった場合、エラーが発生します。

J1939 Protocol 用の専用プロパティ

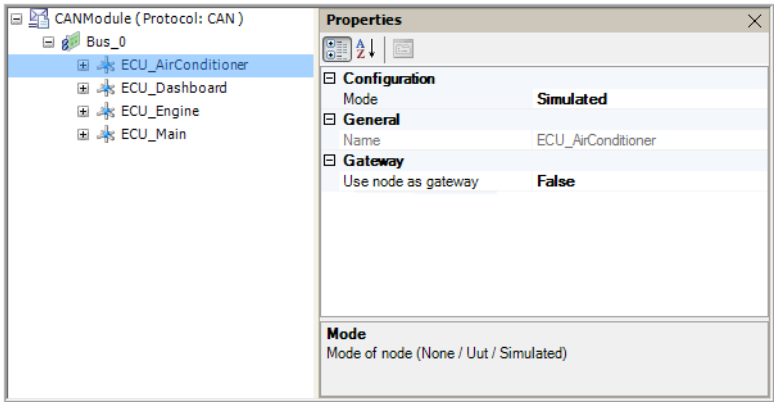
これらのプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。

DTC:

- **Number of Diagnostic Trouble Codes (DTCs)**
  J1939 ノードは DTC を扱うことができ、バッファリングできる DTC の数をこのプロパティで指定します。これらのバッファは、ダイナミックに追加されたノードについてのみ使用可能です（103 ページの「Number of unspecified nodes」を参照）。

  この機能には大量のメモリを必要とするため、設定可能な最大値は 50 です。
- **Number of unspecified nodes**
  ダイナミックに追加できるノードの最大数。これらのノードは CAN ネットワーク内でアドレスを宣言する（つまり「アドレスクレーム」を行う）ことによって登録できます。

ノード



Configuration:

- **Mode**
  ノードのコンフィギュレーション（86 ページの図 3-7 を参照してください）
  - **Simulated**
    シミュレートされるノード
  - **UuT**
    実在するノード
  - **None**
    完全にユーザーの定義に基づいて制御されるか、または他のノードのコンフィギュレーションによってリモート制御される「ニュートラル」なノード

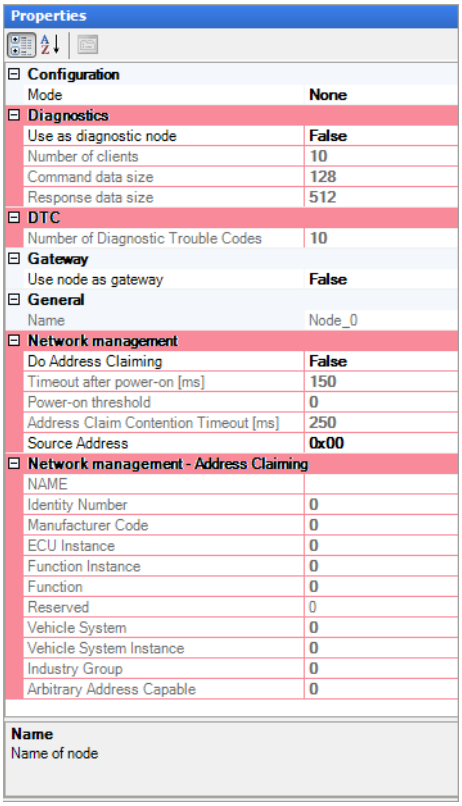
General:

- **Name**
  ノードの名前

Gateway:

- **Use node as gateway**
  ノードをゲートウェイとして設定します。

J1939 プロトコルについてのその他のプロパティ



Diagnostics:

- **Use as diagnostic node**
  ノードを「診断ノード」として使用します。

> **注記**
>
> 1 つのバスには診断ノードを 1 つしか使用できません。

以下の 3 つのプロパティは、**Use as diagnostic node** が **True** の場合にのみ有効となります。

- **Number of clients**
  LABCAR-EE と LABCAR-AUTOMATION に含まれる、診断機能を持つクライアントの数を指定します。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。

- **Command data size**
  診断コマンドは、常に配列として送信されます。このプロパティはその配列のサイズ、つまり送信されるコマンドの数を指定するものです。各コマンドに含まれるデータ量はそれぞれ異なる可能性があり、それに応じて実際に送信できるコマンド数も変わります。

  このサイズは、同時に送信するコマンド数を増やす必要がある場合にのみ大きくするようにしてください。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。
- **Response data size**
  診断応答は、常に配列として送信されます。このプロパティはその配列のサイズ、つまり保存される応答の数を指定するものです。

  コマンドに応じて応答に含まれるデータ量は異なるため、必要なコマンドに対する応答がすべて保存できない場合にのみ大きくするようにしてください。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。

DTC:

- **Number of Diagnostic Trouble Codes**
  このノード用にバッファリングできる ETC の数を指定します。

  このプロパティは、**Expert Mode** オプション（90 ページの「"Expert Mode" オプション」を参照してください）がオンになっている場合のみ表示されます。

Network management:

- **Do Address Claiming**
  アドレスクレームを送信してアドレス宣言を行うかどうかを指定します。[1]
- **Timeout after power-on [ms]**
  シミュレートされるノードは、ここで指定された時間（単位 : ms）だけ待機した後にアドレスクレームを送信します。これはパワーオンセルフテスト（POST: Power On Self Test）をシミュレートするためのものです。
- **Power On Threshold**
  シミュレートされるノードには、このしきい値と比較するための入力／ IN ポートがあり、入力値がしきい値を超えた時点でアドレスクレームが実行されます。
- **Address Claim Contention Timeout [ms]**
  シミュレートされるノードがアドレスクレームを送信した後に、他に同じアドレスを宣言するノードがないかを確認する時間（単位 : ms）を指定します。この時間内に他のノードから同じアドレスが宣言されなければ、宣言したアドレスが承認されたものと判断し、そのアドレスを使用します。
- **Source Address**
  ソースアドレス

[1] 現時点においては、以下に記述されたアドレス宣言のみサポートされています。"SAE J1939-81, Appendix D, Figure D2 - State Transition Diagram for Initialization of Single Address CAs"

Network management - Address Claiming:

- **NAME**
  ノード（ECU）の識別子
  この識別子には以下の 10 個のフィールドが含まれます。
- **Identity Number**
  21 ビット : ECU 製造元が定義した識別番号
- **Manufacturer Code**
  11 ビット : ECU 製造元を識別するための情報
- **ECU Instance**
  3 ビット : ECU の各インスタンスを識別するための情報（複数の ABS 用 ECU が存在する場合など）
- **Function Instance**
  5 ビット : ファンクションの各インスタンスを識別するための情報（例 : ABS #1）
- **Function**
  8 ビット : ファンクションを識別するための情報（例 : ABS）
- **Reserved**
  SAE によって将来の用途向けに予約されたフィールド
- **Vehicle System**
  7 ビット : 車両システムを識別するための情報（例 : トレーラー）
- **Vehicle System Instance**
  4 ビット : 車両システムの各インスタンスを識別するための情報
- **Industry Group**
  3 ビット : 特定の産業分野を定義する情報（例 : "On-Highway"、"Agricultural"）
- **Arbitrary Address Capable**
  1 ビット : ECU がアドレスクレーム中にアドレスの競合が生じた場合、別のアドレスを選択できるようにするかどうかを指定します。

パート

ノードのバリアントである「パート」は、ノードの直下に割り当て、その下にフレームを割り当てます。パートはインポートとエクスポートが可能です。各パートの内容は、それぞれ個別のファイル（`.lcp`）に保存されます。

各パートには、ネットワーク内でユニークな名前を割り当てます。すでに使用されている名前を指定すると、名前の末尾に連番が付きます。

パートの優先順位付けと有効化／無効化

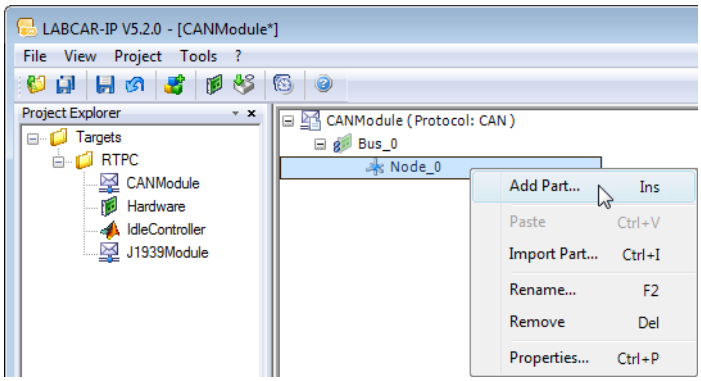
ネットワークノードの複数のバリアントをモデリングするには、割り当てられたすべてのフレームと信号を含むパートの複製を作成します。

各パートは **Enabled** プロパティを使用して有効化／無効化することができます。各パートのフレームと信号の処理は、**Priority** によって制御できます。各シミュレーションサイクルにおいて、優先度の高いパートは優先度の低いパートの入力と出力を上書きします。

パートを作成する：

- 新しいパートを割り当てたいノードを右クリックします。

- ショートカットメニューから **Add Part** を選択します。

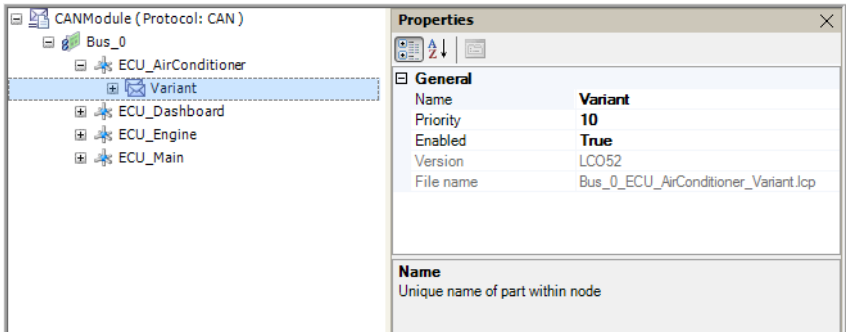


"Variant" という名前のパートが作成されます。

注記

パート、フレーム、信号の名前に使用できるのは、ANSI-C 識別子として使用できる文字に限られます。

作成されたパートには、エクスポート済みのパートをインポートすることができます（109 ページの「パートをインポートする：」を参照してください）。



General:

- **Name**
  パートの名前
- **Priority**
  パートの優先順位を指定します。各パートは優先順位に基づいて実行されるので、この値はすべてユニークである必要があります。
- **Enabled**
  このプロパティにより、実験の実行時に各パートを個別に有効化／無効化することができます。無効になっているパートに含まれるフレームは、送信されません。
- **Version**
  パートのバージョン

- **File name**
  パートの内容が保存されるファイルの名前
  （<Bus name>_<Node name>_<Part name>.lcp）

注記

実験実行中のバリアントの扱いには、**Enabled** および **Priority** プロパティを使用します。

パートをエクスポートする：

パートをエクスポートすると、そこに含まれるすべてのフレームとそのプロパティが XML ファイルに保存されます。

- エクスポートしたいパートを右クリックします。
- ショートカットメニューから **Export Part** を選択します。
  ファイル選択ダイアログボックスが開きます。
- エクスポートファイルのパスとファイル名（*.lcp）を指定して、**OK** をクリックします。
  パートの内容がファイルにエクスポートされます。

パートを削除する：

- 削除したいパートを右クリックします。
- ショートカットメニューから **Remove** を選択します。

パートをインポートする：

エクスポートされたポートは、CAN コンフィギュレーションに含まれるポートにインポートすることができます。

- エクスポートされたパートをインポートしたいノードを右クリックします。
- ショートカットメニューから **Import Part** を選択します。
  ファイル選択ダイアログボックスが開きます。
- インポートするファイルのパスとファイル名（*.lcp）を指定して、**OK** をクリックします。
  ファイルから読み込まれた内容を含むパートが作成されます。

フレーム

フレームはパートに割り当て、フレームに信号を割り当てます。

各フレームについては、"Properties" ウィンドウに加え "Byte Layout" ウィンドウが表示され、フレーム内の各信号の割り付けが示されます。

各フレームには、その方向に応じて以下のアイコンが表示されます。

 ― 送信フレーム（TX）

 ― 受信フレーム（RX）

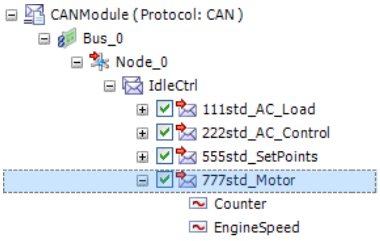
 ― ゲイトウェイフレーム（GW）
（104 ページの「Use node as gateway」を参照してください）

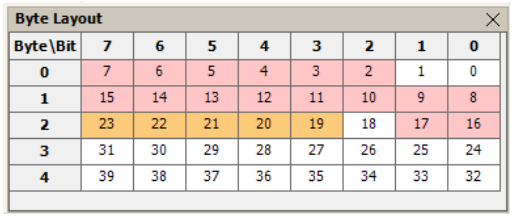
注記

ここに示される「メッセージの方向」は、シミュレートされる部分から見た方向です（86 ページの図 3-8 を参照してください）。

バイトレイアウトを表示する：

- フレームを右クリックします。
- ショートカットメニューから **Byte Layout** を選択します。





上記の例では、フレームには 16 ビット信号 "EngineSpeed"（ピンク色）と 5 ビット信号 "Couter"（オレンジ色）が含まれています。

- 上図のように表示されたビットフィールドの 1 つをダブルクリックすると、そのフィールドに対応する信号がメッセージリスト内で選択され、そのプロパティが "Properties of Signals" フィールドに表示されます。

注記

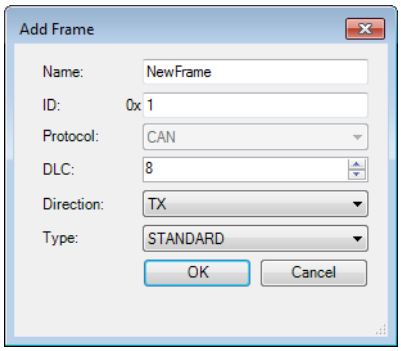
バイトレイアウト内にエラー（2 つの信号が重複していたり、信号やフレームの長さが正しく指定されていない、など）があると、該当するビットが赤で表示されます。

ユーザー定義フレームを作成する：

CANdb ファイル、または CAN モジュールやパートからインポートされたもの以外に、ユーザー定義されたフレームを作成することもできます。

- フレームを作成したいパートを右クリックします。

- ショートカットメニューから **Add Frame** を選択します。



注記

1 つのパート内では、フレームの ID と名前の組み合わせは、ユニークである必要があります。

- 以下の情報を入力します（113 ページの「General」を参照してください）。
  - **Name**: フレームの名前
  - **ID**: フレームの ID
  - **Protocol**: プロトコル

    プロトコルは、CAN モジュールを作成する際に選択されています（93 ページの「CAN モジュールを 追加する：」を参照してください）。
  - **DLC**: フレームのサイズ
  - **Direction**: 転送方向
  - **Type**: フレームタイプ
- **OK** をクリックします。

  新しく作成されたフレームが、CAN コンフィギュレーションに追加されます。

複数の CAN ボードを使用している場合、CAN コンフィギュレーションを設定する際には以下の点に注意してください。

注記

メッセージが割り当てられているボードを削除した際には、割り当ての自動変更は行われません。変更はマニュアル操作で行う必要があります。

フレームを有効化／無効化する：

1 つのパート内に、名前が異なっていて ID が同じである複数のフレームを定義することができます。ただしコード生成の際には、その中のいずれか 1 つのみを有効にして、他のフレームはコード生成から除外する必要があります。フレームの有効化／無効化は、以下のいずれかの方法で行います。

- **ネットワークツリービューを使用する場合**：フレームアイコンの左側のチェックボックスのオン／オフを切り替えます。

- **ショートカットメニューを使用する場合** : フレームを右クリックしてショートカットメニューからを開き、**Enable** コマンドを選択します。
- **"Properties" ウィンドウを使用する場合** : フレームの "Properties" ウィンドウを開き、**Enabled** プロパティ（**True** ／ **False**）を変更します。

フレームを削除する：

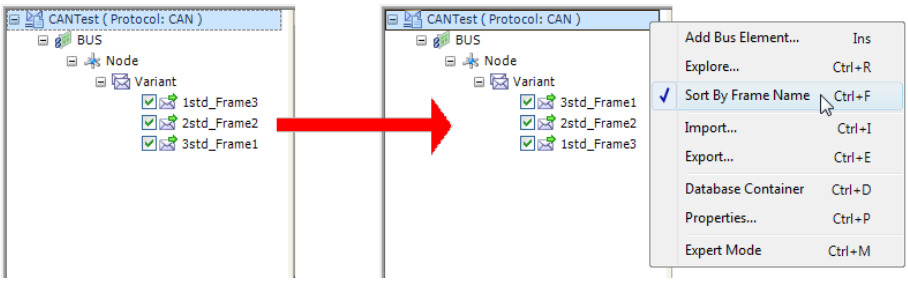
- 削除したいフレームのショートカットメニューから **Remove** を選択します。

  フレームが削除されます。

フレームを名前順にソートする：

デフォルト状態において、フレームはフレーム ID の順で表示されます。

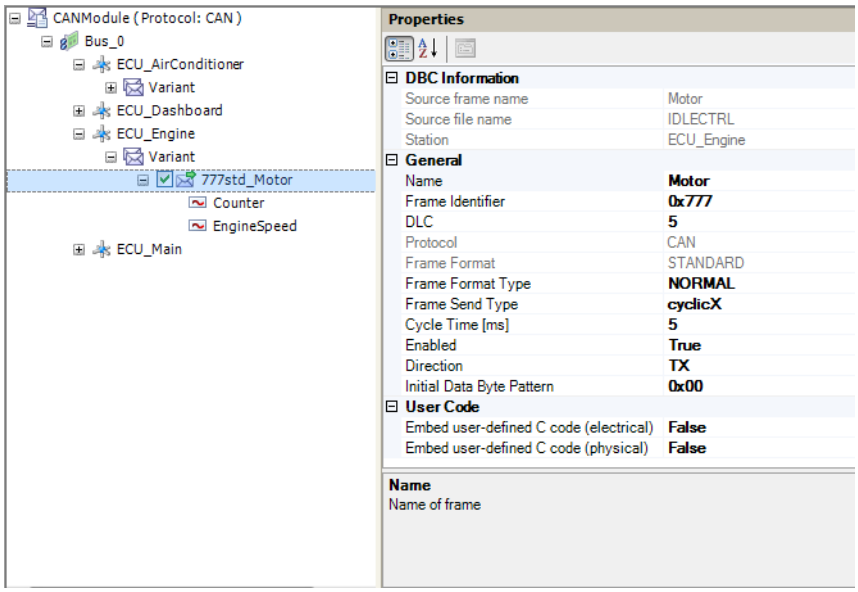
- フレームを名前でソートするには、最上位の CAN モジュールを右クリックします。
- ショートカットメニューから **Sort by Frame Name** を選択します。



フレームのプロパティを表示する：

- フレームを右クリックします。

- ショートカットメニューから **Properties** を選択します。
  フレームのプロパティが "Properties" ウィンドウに表示されます。



DBC Information:

CANdb ファイルからインポートされたフレームの場合は、以下のプロパティがあります。

General:

- **Name**
  フレームの名前
- **Frame Identifier**
  フレーム ID（HEX）
- **DLC**
  フレームのデータ長コード（Data Length Code）
- **Protocol**
  プロトコル（**CAN** ／ **J1939**）
- **Frame Format**
  フレームのフォーマット（**STANDARD** / **STANDARD_REMOTE** / **EXTENDED** / **EXTENDED_REMOTE**）
- **Frame Format Type**
  NORMAL / **REMOTE**
- **Frame Send Type**
  送信モード（送信フレームの場合のみ）
  - **cyclicX**
    周期ラスタで送信されます。

– **spontanX**
いずれかの信号値が変化した時点で送信されます。

注記

spontanX タイプは、いずれかの信号の送信モードが **On Change** である場合にのみ送信されます（117 ページの「Send Type」を参照してください）。

– **cyclicAndSpontanX**
上記の両方のタイミングにおいて送信されます。

- **Cycle Time [ms]**
フレームの送信周期（単位 : ms）
- **Enabled**
フレームの有効化／無効化（111 ページの「フレームを有効化／無効化する：」を参照してください）。
- **Direction**
モデル側からみた転送方向（**TX** = 送信、**RX** = 受信）
- **CAN-FD**
フレームを CAN-FD フレームとして扱うかどうかを指定します（**True**: CAN-FD フレーム、**False**: 標準フレーム）。
- **Bit Rate Switch (BRS)**
CAN-FD フレームの送信中に高速のビットレートに切り替えるかどうかを指定します。
- **Initial Data Byte Pattern**
フレーム初期化時に使用するバイトパターン

User Code:

- **Embed user-defined C code (electrical)**
Embed user-defined C code (physical)
ユーザー定義された C コードをフレームに追加するかどうかを指定します。（119 ページ「ユーザー定義の C コード」参照）。

J1939 プロトコルについてのその他のプロパティ

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

メッセージの CAN ID（29 ビット）で、以下のものが含まれます。

– **Priority** （3 ビット）
– **Parameter Group Number** （18 ビット）
– **Source Address** （3 ビット）

- **Parameter Group Number**
PGN（パラメータグループ番号）には以下のものが含まれます。

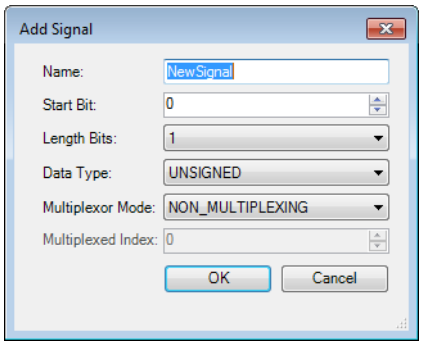
  - **Data Page**（1 ビット）
  - **PDU Format**（8 ビット）
  - **PDU Specific Field**（8 ビット）
- **Data Page**
  データページビット
- **PDU Format**
  PDU フォーマットを指定する 8 ビットデータ
  - < 240: **PDU Specific Field** が "Destination Address"（"PDU1"）
  - >= 240: **PDU Specific Field** が "Group Extension"（"PDU2"）
- **PDU Specific Field**
  8 ビットデータ（**PDU Format** 参照）
- **Priority**
  バスへの転送時のレイテンシを最適化するための 3 ビットデータ（LABCAR-OPERATOR には無視されます）
- **Source Address**
  フレームが属するノードのソースアドレス（8 ビット）
- **Frame specific source address**
  True の場合、フレームのソースアドレスと異なるアドレスをフレームのソースアドレスとして設定することができます。
- **PGN Type**
  PGN タイプ
- **PDU Format Type**
  PDU1 / **PDU2**（**PDU Format** 参照）
- **PDU Specific Field Functionality**
  PDU Specific Field の機能（**PDU Format** 参照）

信号（シグナル）

信号は、各フレームに割り当てます。1 つのフレーム内の信号の名前はすべてユニークである必要があります。

ユーザー定義信号を作成する：

- フレームを右クリックします。
- ショートカットメニューから **Add Signal** を選択します。



- 以下の情報を入力します（117 ページ「Scaling」と 116 ページの「General」を参照してください）。
  - **Name**: 信号の名前

- **Start Bit**: 先頭ビット
- **Lengh Bits**: 信号長（ビット数）
- **Data Type**: データ型
- **Multiplexor Mode**: マルチプレクサモード
- **Multiplexed Index**: マルチプレクサインデックス

- **OK** をクリックします。
  新しく作成された信号がフレームに追加されます。

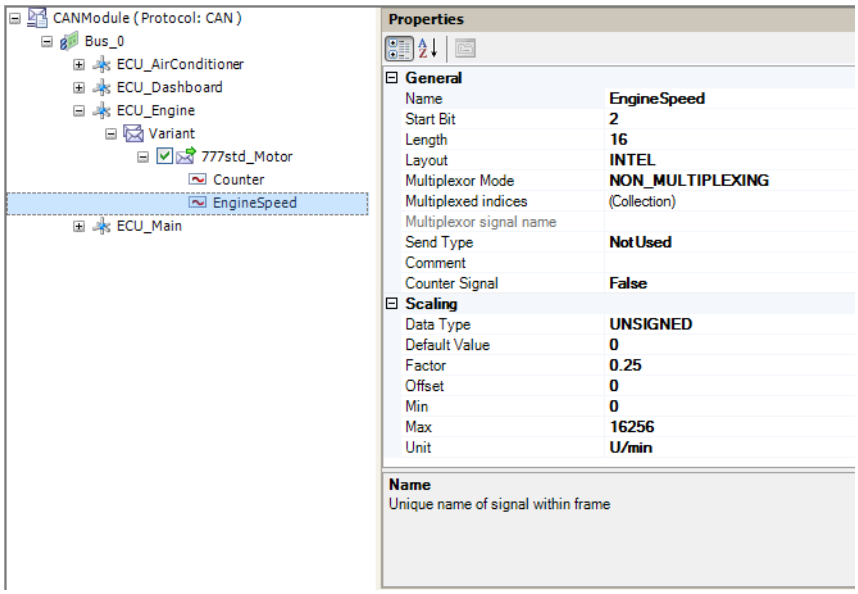
信号を削除する：

フレームに追加した信号は、以下のようにして削除できます。

- 信号を右クリックします。
- ショートカットメニューから **Remove** を選択します。
  信号がフレームから削除されます。

信号のプロパティを表示する：

- 信号を右クリックします。
- ショートカットメニューから **Properties** を選択します。



"Properties" ウィンドウに信号のプロパティが表示されます。

General:

- **Name**
  信号の名前
- **Start Bit**
  信号の先頭ビット
- **Length**
  信号長（ビット数）

- **Layout**
  バイトオーダー（**INTEL**: リトルエンディアン、**MOTOROLA**: ビッグエンディアン）
- **Multiplexor Mode**
  フレームの多重化モード
  - **NON_MULTIPLEXING**
    標準（非多重化）フレーム
  - **MULTIPLEXOR**
    マルチプレクサ（多重化フレームの制御シグナル）
  - **MULTIPLEXED_SIGNAL**
    多重化信号（制御シグナルの値に応じてデータの意味が変わる信号）
- **Multiplexed indices**
  多重化フレームに多重化信号（**MULTIPLEXED_SIGNAL**）の割り当てを行うためのコード。
   をクリックするとコレクションエディタが開き、複数の値を割り当てることができます。
- **Multiplexor signal name**
  信号を使用するかどうかを決定するためのマルチプレクサの名前
- **Send Type**
  フレームの送信タイプ
  - **NotUsed**
    送信トリガの発生時には送信されません。
  - **OnChange**
    送信トリガの発生時に送信されます。
- **Comment**
  信号についてのコメント
- **Counter Signal**
  118 ページの「信号をカウンタとして使用する：」を参照してください。

Scaling:

- **Data Type**
  データ型： **UNSIGNED** / **SIGNED** / **IEEE32** / **IEEE64**
- **Default Value**
  送信フレームの信号にはデフォルト値を定義することができます。ただしコネクションマネージャにおいて別の信号に接続されている信号については、この値は無視されます。

  デフォルト値を定義しておくと、CAN モジュールをエクスポートして別の LABCAR-OPERATOR プロジェクトにインポートした際、その値もインポートされます。
- **Factor** / **Offset**
  信号のスケーリングに使用する係数 a とオフセット b
- **Min, Max**
  妥当性チェックに使用する最大値と最小値（118 ページ「最小値と最大値」参照）

  信号を右クリックしてショートカットメニューから **Limit Min/Max to bit length** を選択すると、**Data Type** と **Length** から自動計算されます。

- **Unit**
  信号の物理単位

最小値と最大値

CAN のコンフィギュレーションが保存される際には、必ず「最小値 < 最大値」の条件が満たされているかどうかがチェックされます。CAN モジュール（CAN ツリーの最上位）の "Properties" フィールド内の "**Ignore min / max** オプション（99 ページの「Ignore Min/Max」参照）が **True** になっていない場合は、この条件が満たされていないと警告が発行されます。

注記

物理値については 2 通りのチェックが行われ、チェック条件が満たされない場合、CAN で転送される値には元の値と異なる値が設定される可能性があります。

値のチェックは以下のように行われます。

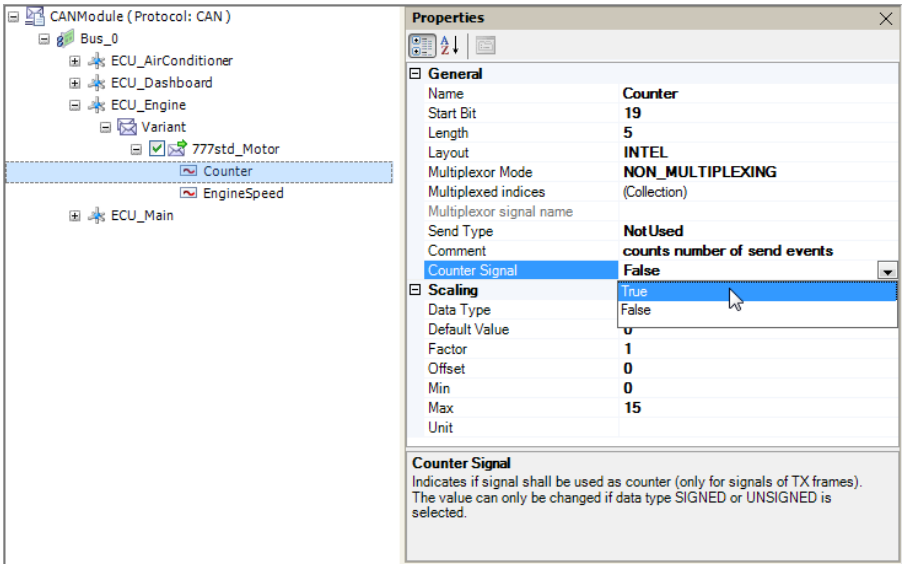
1. 「最小値 ≤ 物理値 ≤ 最大値」の条件が満たされているか？
   この条件が満たされていない場合、値は最小値または最大値に設定されます。
2. 物理値が HEX 値に変換された後、その値が、信号のデータ型に応じたサイズに納まるかどうかがチェックされます。たとえば、6 ビットの signed integer として定義されている信号の場合、変換された値は「-32 ≤ 値 ≤ +31」という条件を満たしている必要があります。
   この条件が満たされていない場合、この信号の値は -32 または 32 に設定されます。

信号をカウンタとして使用する：

以下のようにして、送信フレーム内の 1 つの信号をカウンタとして使用することができます。

- 信号の "Properties" ウィンドウを開きます

- **Counter Signal** プロパティを **True** に設定します。



その信号がカウンタとして使用されます。信号アイコンの左側側に、カウンタであることを示す "123" というシンボルが表示されます。

注記

Use as Counter プロパティは、整数型のメッセージにのみ使用できます。

- カウンタとして使用することをやめるには、**Use as Counter** プロパティを **False** に設定します。

注記

信号プロパティの最小値と最大値は、カウンタ信号にも適用されます。

3.5.7 ユーザー定義の C コード

LABCAR-NIC では、自動生成されるメッセージコードにユーザー定義された C コードを追加することができ、これによってメッセージの管理やメッセージ内容の拡張を行ったりすることができます。ユーザーコードを使用して、モデルから出力される「物理」信号と、最終的に CAN バスに出力されるバイト配列の両方に対して処理を行えます。

以下に、ユーザーコードを用いたアクセス方法を示します。

送信メッセージの信号フローとアクセス

図 3-12 は、モデルから CAN バスへの信号フローを示しています。

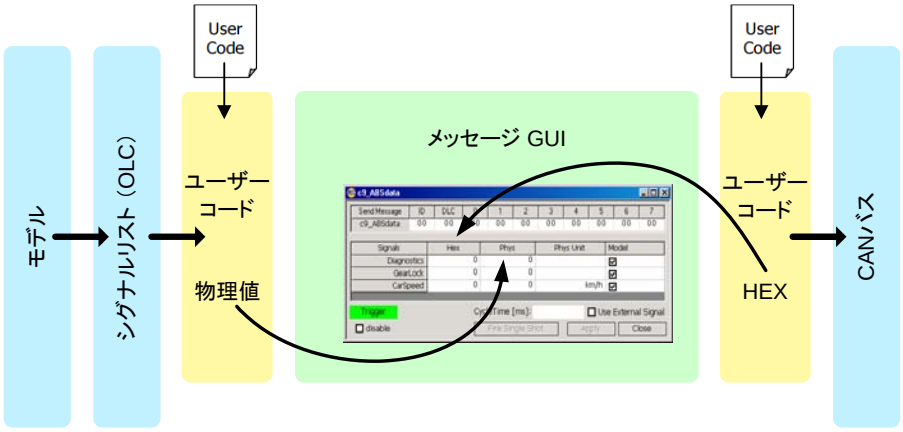


図 3-12 送信メッセージの信号フロー

信号リスト以外に、メッセージ GUI（125 ページ「CAN シミュレーション用インストゥルメント」参照）またはユーザー定義 C コード（以下の項を参照）を用いてメッセージの内容を操作することができます。

注記

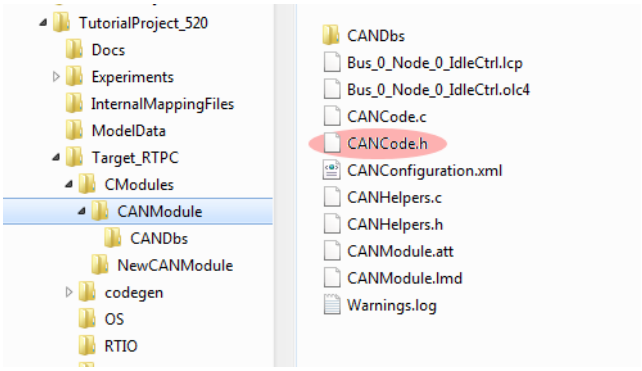
アクセス権により、それまでに変更されたコードは上書きされるので、注意が必要です。

受信メッセージの場合も同様で、上図の左方向への変更により以前の変更内容が上書きされます。

以下に、サンプルコードと、そのコードをメッセージコードに追加する方法について説明します。

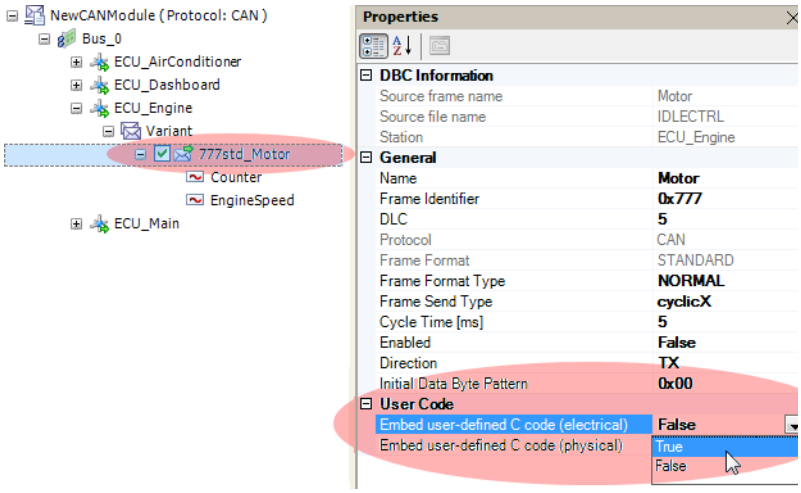
ユーザーコードの追加手順

CAN コンフィギュレーションが保存されると、下図のようなフォルダ内に、関数宣言や信号用の構造体の宣言が含まれるヘッダファイル `CANCode.h` とが作成されます。



ユーザー定義コードを追加する：

- ハンドコーディングしたユーザーコードを追加するには、**Embed user-defined C code (electrical)** または **Embed user-defined C code (physical)** を **True** に設定します。



- **Save** をクリックします。

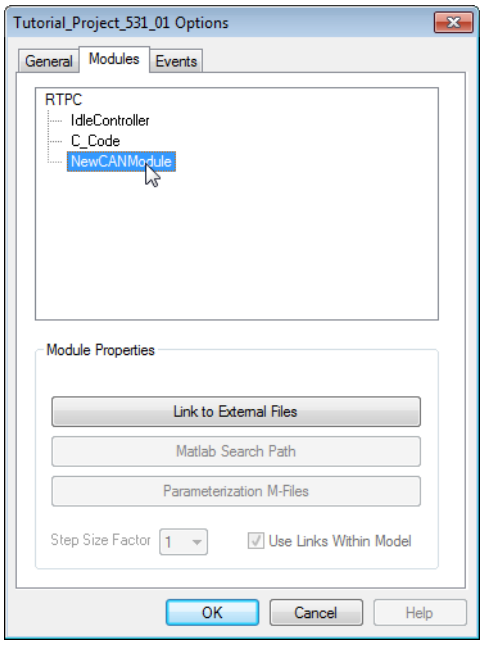
  設定が有効になります。

- `CANCode.h` というファイルを、テキストエディタなどで開きます。

  このファイルの最後部には、設定に応じて下記のような関数（物理信号用とバイト配列用）が宣言されています。これらの関数を使用して物理信号とバイト配列にアクセスすることができます。またこの `CANCode.h` には、シミュレートされる各信号の構造体も宣言されています。

- ユーザーの C ファイルに `"#include CANCode.h"` という文を追加して、関数宣言と構造体宣言がユーザーコード内で認識されるようにします。

- メインメニューから **Project** → **Options** を選択し、"Modules" タブを開きます。



- CAN モジュールを選択して **Link to External Files** ボタンをクリックします。

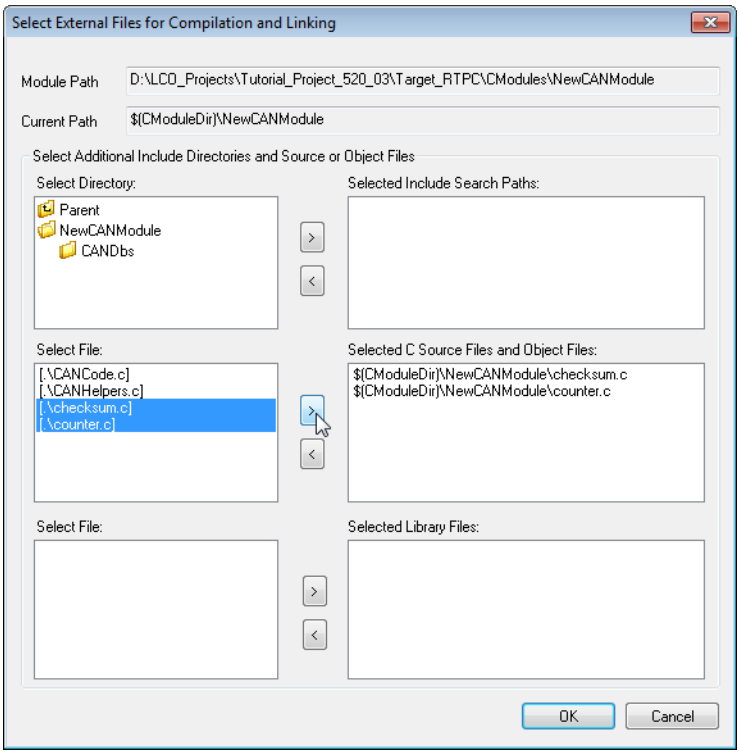
  "Select External files for Compilation and Linking" ダイアログボックスが開きます。

- リンクするソースファイルを選択して（複数可）、**>** ボタンをクリックします。ソースファイルの例は 123 ページの「ユーザーコードの例」を参照してください。

注記

ここで選択できる C ファイルは、`¥Target_RTPC¥CModules¥CAN_Name_of_CANdb_file` というフォルダとそのサブフォルダに保存されているファイルのみです。

- 選択されたファイルがダイアログボックス内の右側のフィールドに表示されます。



- **OK** をクリックします。
  以後、コンパイルとリンクが行われる際は、これらのファイルを含めた処理が行われます。
- **OK** をクリックして "Modules" タブを閉じます。

ユーザーコードの例

以下に、ユーザーコードの例を 2 つ紹介します。

メッセージカウンタ（counter.c）

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

   void userManipulationPhysical_NewCANModule_Bus_0_ECU_Engine_
   777std_Motor ( int32_t inputTrigger, NewCANMod-
   ule_Bus_0_ECU_Engine_777std_Motor_PhysValues_t* signalStructure )
   {
      static int counter = 0;
      signalStructure -> Signal_Counter = counter;
```

```
    (counter <= 14) ? (counter++): (counter = 0);
}
```

注記

"counter.c" に記述されたコード例は、周期的に送信されるフレーム（**Frame Send Type** = cyclicX、113 ページを参照してください）についてのみ機能します。

チェックサムの算出（checksum.c）

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

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

3.6 CAN シミュレーション用インストゥルメント

LABCAR-EE（Experiment Environment）には、LABCAR-NIC V5.4.0 で CAN バスシミュレーションや CAN バスモニタを行うための CAN メッセージ用インストゥルメントが用意されています。

3.6.1 "Workspace Elements" ウィンドウ内の CAN モジュール

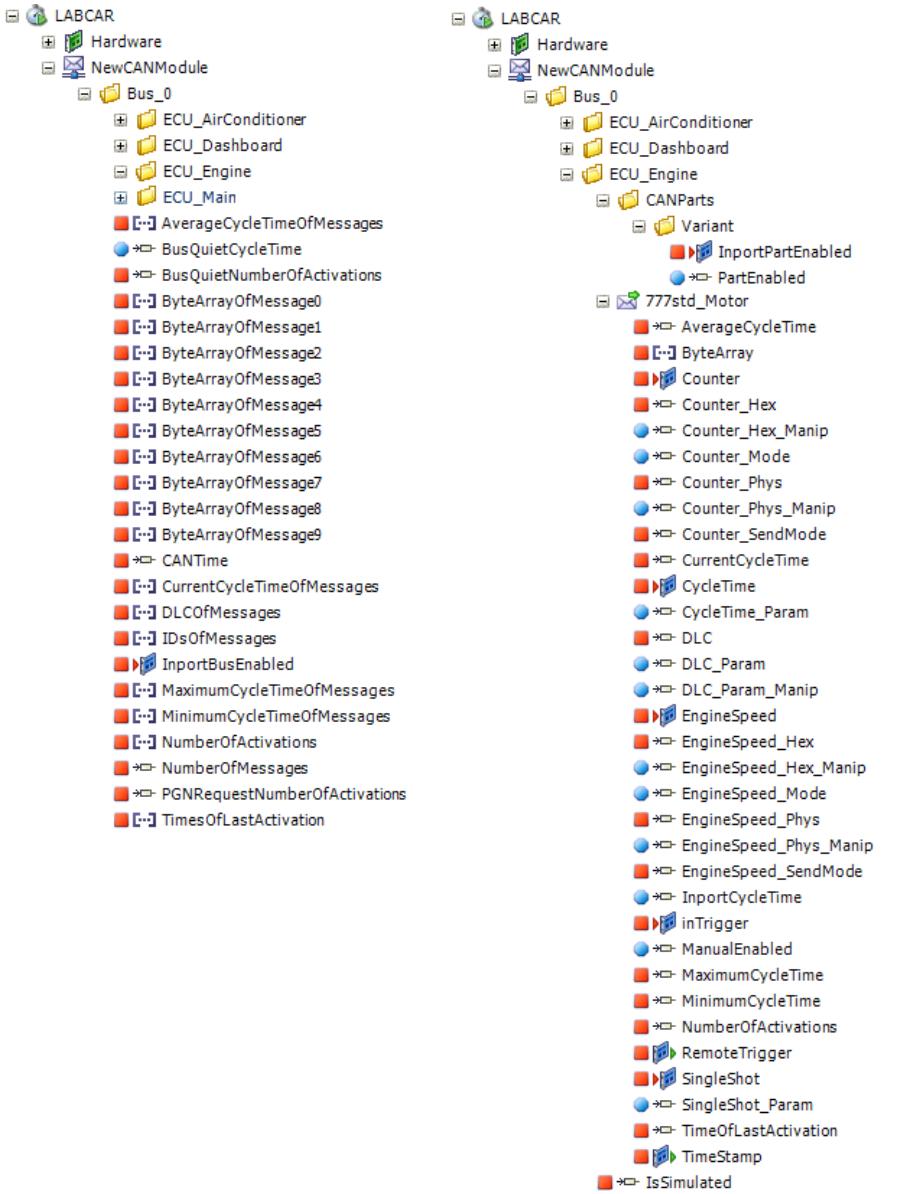


図 3-13 "Workspace elements" ウィンドウに表示される CAN モジュール

"Workspace Elements" ウィンドウに表示される CAN モジュールには、以下のアイテムが含まれます。

バス

最上位にバスを表すフォルダがあり、以下の測定変数とパラメータ（適合変数）が含まれます（125 ページの図 3-13 の左側を参照してください）。

- **AverageCycleTimeOfMessages**
  n 個の要素を持つ配列で、n 個の未定義メッセージの平均周期が含まれます。
- **BusQuietCycleTime**
  BusQuiet（DM13）メッセージの転送周期
- **BusQuietNumberOfActivations**
  DM13 メッセージの送信回数
- **ByteArrayOfMessage*n***
  未定義メッセージが含まれる配列。配列の数は、LABCAR-IP の CAN エディタにおいて、CAN バスの "Properties" ウィンドウ内の **No. of Unspecified Messages** プロパティで設定します。
- **CANTime**
  シミュレートされる CAN バス部分のタイムスケールです。CAN 送信処理が最初に実行される時点でこの値がゼロになります。デフォルトにおいて CAN バスシミュレーションは、LABCAR-EE の実験開始と同時に開始されます。
- **CurrentCycleTimeOfMessages**
  n 個の未定義メッセージの現在の周期を含む、要素数 n の配列。
- **DLCOfMessages**
  n 個の未定義メッセージの DLC を含む、要素数 n の配列。
- **IDsOfMessages**
  n 個の未定義メッセージの ID を含む、要素数 n の配列。
- **InportBusEnabled**
  バスの有効化／無効化を行うための入力ポート
  - - 0.5 < 入力ポートの値 < + 0.5 = False：バスは無効化されます
  - 上記以外は True（デフォルト = 1.0）：バスは有効化されます
- **MaximumCycleTimeOfMessages**
  n 個の未定義メッセージの最大周期を含む、要素数 n の配列。
- **MinimumCycleTimeOfMessages**
  n 個の未定義メッセージの最小周期を含む、要素数 n の配列。
- **NumberOfActivations**
  n 個の未定義メッセージの起動回数を含む、要素数 n の配列。
- **NumberOfMessages**
  未定義メッセージの数 n
- **PGNRequestNumberOfActivations**
  PGN 要求メッセージの送信回数
- **TimesOfLastActivations**
  n 個の未定義メッセージの最後の起動時刻を含む、要素数 n の配列。

ノード

ノードのフォルダには、ノードのバリアントである「パート」が含まれる "CANParts" フォルダと、ノードのメッセージが含まれます。

- **InportPartEnabled**
  バスの有効化／無効化を行うための入力ポート
  - - 0.5 < 入力ポートの値 < + 0.5 = False：バスは無効化されます
  - 上記以外は True（デフォルト = 1.0）：バスは有効化されます
- **PartEnabled**
  ランタイムにおいてパートの有効化／無効化を行うためのパラメータです。

メッセージ

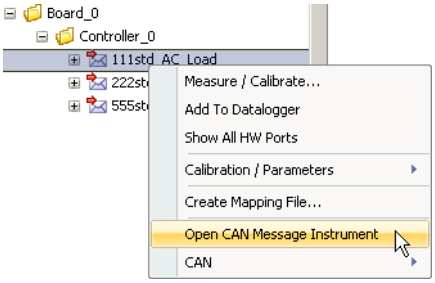
各メッセージには以下の測定変数とパラメータ（適合変数）が含まれます。ただし * の付いたものは、送信メッセージにのみ含まれます。

- **<Signalname>**
  ハードウェア信号の名前
- **<Signalname>_Hex**
  信号の HEX 値
- **<Signalname>_Hex_Manip***
  信号の HEX 値（調整された値）
- **<Signalname>_Mode**
  信号のモード（以下のいずれか）を表します。
  - 0: モデルから受け取った値
  - 1: 変更された物理値
  - 2: 変更された HEX 値
- **<Signalname>_Phys**
  信号の物理値
- **<Signalname>_Phys_Manip***
  信号の物理値（調整された値）
- **<Signalname>_SendMode**
  メッセージの送信タイプを表す測定変数（117 ページ「Send Type」参照）
  0 = NotUsed、**1** = OnChange.
- **AverageCycleTime**
  平均周期
- **ByteArray**
  メッセージのバイト配列
- **CurrentCycleTime**
  現在測定されている周期
- **CycleTime***
  周期の入力ポート（127 ページ「InportCycleTime」参照）
- **CycleTime_Param***
  周期を指定するためのパラメータ
- **DLC**
  メッセージのデータ長
- **DLC_param**
  ユーザー定義のメッセージ用 DLC パラメータ
- **DLC_param_manip**
  ユーザー定義された DLC（**1**）と LABCAR-IP の CAN エディタで定義された値（**0**）のどちらを有効にするかを決定する論理値パラメータ
- **InportCycleTime**
  周期を **CycleTime_Param** パラメータ で定義するか（=**False**）、または入力ポート **CycleTime** からスティミュレートするか（=**True**）を指定します。
- **InTrigger***
  メッセージ送信の有効化／無効化を行います（ 129 ページの「送信メッセージの有効化／無効化」参照）。**0** は **False**、その他の値は **True** を表します。

- **ManualEnabled**
  メッセージ送信をマニュアル操作で行うためのパラメータです。メッセージ GUI の **disabled** オプションに該当します（ 129 ページの「送信メッセージの有効化／無効化」参照）。
- **MaximumCycleTime***
  測定された最大周期
- **MinimumCycleTime**
  測定された最小周期
- **NumberOfActivations**
  メッセージの起動回数
- **RemoteTrigger***
  メッセージがリモートで何回起動されたかをカウントする出力ポート
- **InputTrigger**
  InTrigger 信号用測定変数
- **SingleShot***
  シングルショット送信を有効にするための論理型パラメータです。**True** に設定するとメッセージが 1 回送信され、その後 **False** にリセットされます。
- **SingleShot_Param***
  シングルショット送信をトリガするパラメータです。メッセージ GUI 内の **Transmit Once** ボタンに相当します。
- **TimeOfLastActivation**
  メッセージの最後の起動時刻
- **TimeStamp**
  現在の **CANTime** から決定される、メッセージ送受信時のタイムスタンプ（126 ページの「CANTime」を参照）

3.6.2 CAN メッセージ用インストゥルメント

メッセージ表示用のインストゥルメントを作成するには、メッセージを右クリックして **Open CAN Message Instrument** を選択します。



メッセージタイプに合ったインストゥルメント（GUI）が作成され、メッセージの内容が表示されます。

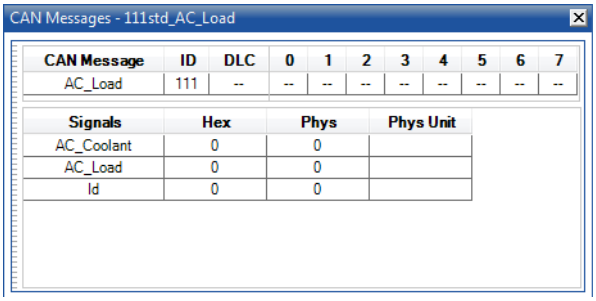


図 3-14 受信メッセージ用のメッセージ GUI

受信メッセージの場合、メッセージとその信号の内容が表示され、信号の編集は行えません。

送信メッセージの場合は、**Model** オプションを無効にすると信号の値（HEX 値または物理値）を編集することができます。変更内容を確定するには **Apply** をクリックします。

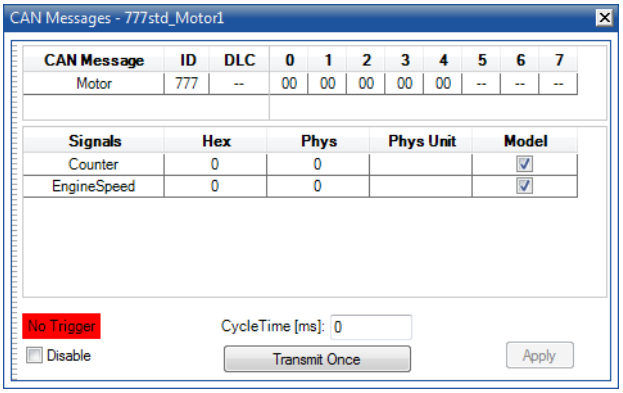


図 3-15 送信メッセージ用のメッセージ GUI

Transmit Once をクリックすると、CAN バスに対してメッセージが 1 回だけ送信されます（130 ページの「シングルショット送信」を参照してください）。

"Cycle Time [ms]" フィールドには送信メッセージの送信周期を入力します。

注記

モデルからの "CycleTime" シグナルを使用する場合、値は秒単位で指定してください。

3.6.3 送信メッセージの有効化／無効化

"Trigger" が緑の背景で表示されている場合は、各送信メッセージに含まれる "inTrigger" シグナルの値が論理値 "True" になっています。このシグナルは CANdb ファイルから読み込まれたものではなく、LABCAR-OPERATOR によって追加されたものです。これは一般的に、モデルからの送信メッセージの送信を制御するために使用されます。

なお、インストゥルメント上の **disable** オプションをオンにすることにより、"inTrigger" シグナルによるメッセージ送信を無効にすることもできます。具体的には、メッセージ送信の条件には "inTrigger" シグナルと "disable" オプションの AND 演算が用いられます。つまり、"disable = false" および "Trigger = true" の条件が共に満たされている場合にのみ、メッセージが送信されます。

シングルショット送信

上記の AND 演算の結果が **False** である場合、タイマによるメッセージ送信の制御は行われません。この状態においては **Fire Single Shot** ボタンが有効になり、このボタンをクリックすると、メッセージが 1 回だけ送信されます。

また、各送信メッセージには、"InTrigger" 以外に "SingleShot**"** というシグナルがあり、このシグナルの値が 0 → 1 に変化した際にも、メッセージが 1 回だけ送信されます。

3.6.4 CAN モニタ

LABCAR-OPERATOR には、CAN バス上のトラフィックを監視するために "CAN Bus Monitor" というインストゥルメントが用意されています。

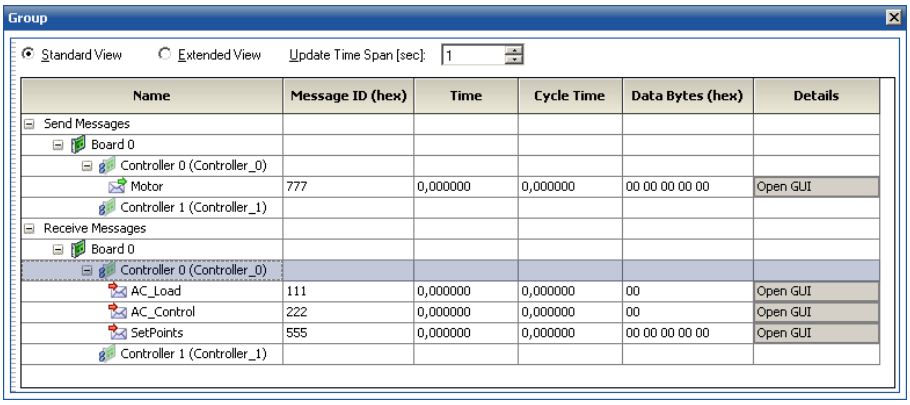


図 3-16 "CAN Bus Monitor" インストゥルメント

このウィンドウには各 CAN メッセージが階層的に（"Send/Receive Message / Board / Controller"）表示され、各メッセージの名前のほか、各列に以下のような情報が表示されます。

- **Message ID (hex)**
  メッセージ ID
- **Time**
  メッセージのタイムスタンプ（シミュレーション開始からの経過時間）
- **Cycle Time**
  メッセージ周期
- **Data Bytes (hex)**
  メッセージのデータバイト数
- **Details**
  メッセージがプロジェクト内に定義されている場合、この列の **Open GUI** ボタンをクリックしてメッセージ GUI（129 ページの図 3-14 と 129 ページの図 3-15 を参照してください）を開くことができます。

ウィンドウ上部の **Extended View** というオプションをオンにすると、CAN メッセージについて、以下のような詳細な情報が表示されます。

- **Message Count**
  送信されたメッセージの数

- **Mean Cycle Time**
  平均の送信周期
- **Min. Cycle Time**
  最小の送信周期
- **Max. Cycle Time**
  最大の送信周期

送信メッセージについては、CAN エディタで追加されたメッセージがすべてここに表示されます。メッセージが最初に送信されると、メッセージ名の右側の 2 つのカラムに値が表示されます。

受信メッセージの場合は、各コントローラで実際に受信されたメッセージのみが表示されます。同じメッセージが CAN エディタで追加されている場合は、メッセージ名が表示され、それ以外のメッセージは "n/a" と表示されます。

このウィンドウの情報を更新する周期は、"Update Time Span [sec]" で指定します。

3.7 LIN モジュール（LABCAR-NIL ― LIN ネットワークの統合）

LABCAR-NIL（Network Integration LIN）は、LIN 2.0 や LIN 2.1 ／ LIN 2.2 に準拠する LIN[1] 通信を使用した ECU ファンクションのテストを行うための LABCAR-OPERATOR 用アドオンです。

LIN バス通信の仕様は 1 つまたは複数の LDF[2] ファイルから読み込むことができ、マニュアル操作での調整や拡張も可能です。送受信する LIN フレームをユーザーが定義すると、LABCAR-NIL がコードを自動生成し、これにユーザー定義コードを追加することもできます。

選択されたフレームの信号はコネクションマネージャに表示され、ここでモデル入力（受信フレーム）やモデル出力（送信フレーム）に接続できます。

ハードウェア要件

LABCAR-NIL を使用するには、最新バージョンの ETAS RTPC がインストールされた Real-Time PC が必要で、さらにその PC には以下の IXXAT 社製 LIN ボードのいずれかが 1 枚以上インストールされている必要があります。

- iPC-I XC16/PCI（LIN 2.0 のみ）
- CAN-IB200/PCIe、CAN-IB600/PCIe（LIN 2.0、LIN 2.1 ／ LIN 2.2）

この LIN ボードは、LABCAR-RTC（Real-Time Execution Connector）に組み込む必要はありません。

LIN バスのコンフィギュレーションを設定するには、LABCAR-IP において、LIN バス全体のシミュレーション情報を含む LIN モジュールを作成します。

各 LIN モジュールには、最大 16 個の LIN ボードと 1 つの LIN コントローラ（iPC-I XC16/PCI の場合）、または、最大 8 個の LIN ボードと 2 つ（CAN-IB600/PCIe の場合）または 4 つ（CAN-IB200/PCIe の場合）の LIN コントローラを組み込むことができ、LIN マスタノードまたは LIN スレーブノードとしてコード生成を行うことが可能です。

あるいは、各 LIN モジュールにコントローラを 1 個ずつ割り当てて、複数の LIN モジュールと実際の LIN コントローラで LIN バスを形成することもできます。

LIN ボードには、外部電源から LIN コントローラ接続経由で電力を供給する必要があります。

3.7.1 LIN モジュールの作成

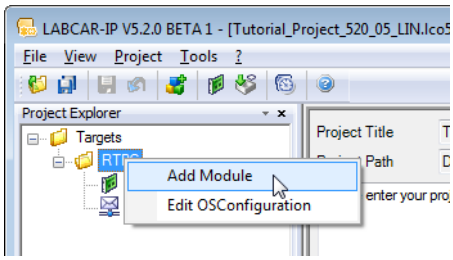
新しい LIN モジュールの作成は、モジュール追加ウィザードを使用して行います。

LIN モジュールを作成する：

- LIN モジュールを組み込む LABCAR プロジェクトを開きます。
- プロジェクトエクスプローラで、"RTPC" というターゲットを右クリックします。

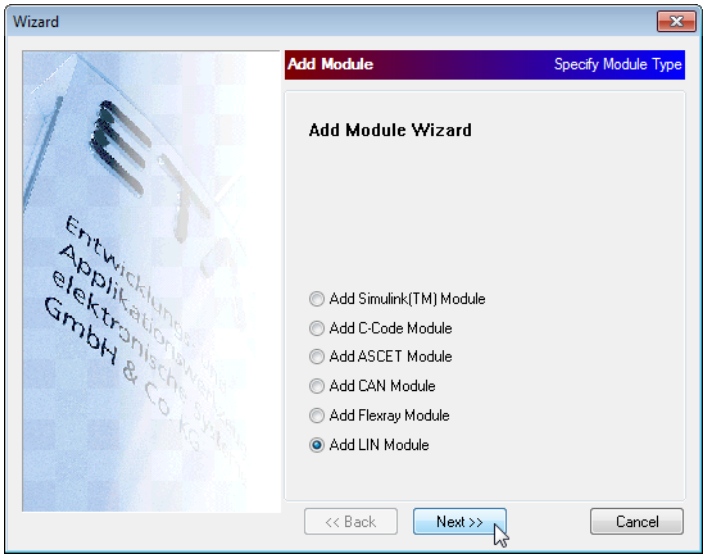
1. **L**ocal **I**nterconnect **N**etwork
2. LIN ディスクリプションファイル

- ショートカットメニューから **Add Module** を選択します。



モジュール追加ウィザードが開きます。

- "Add LIN Module" を選択します。



- **Next** をクリックします。
- モジュールの名前を入力します。
- プロトコルバージョンを選択します。
- **Finish** をクリックします。

  LIN モジュールが作成され、プロジェクトエクスプローラに表示されます。

3.7.2 LIN エディタ

LIN エディタを開くには、プロジェクトエクスプローラ内の LIN モジュールをダブルクリックします。

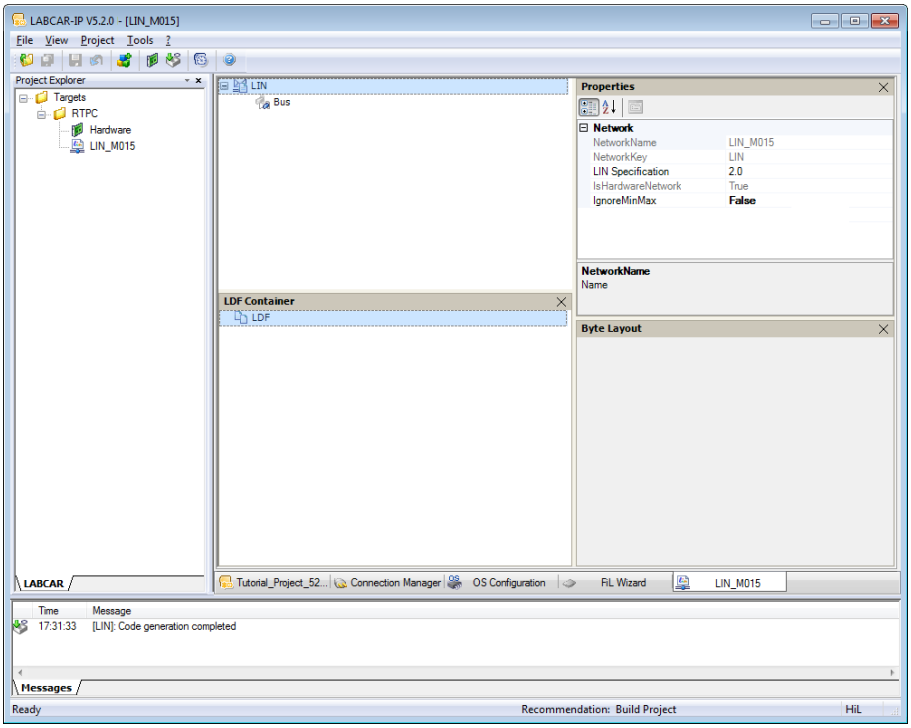


図 3-17 LIN エディタ

LIN エディタには以下のものが表示されます。

- LIN ネットワークペイン（134 ページ参照。）
- "LDF Container" ウィンドウ（135 ページ参照）
- プロパティ（"Properties" ウィンドウ、135 ページ参照）
- フレームのバイトレイアウト（"Byte Lauout" ウィンドウ、136 ページ参照）

LIN ネットワークペイン

ネットワークを構成するエレメントが階層表示されます。

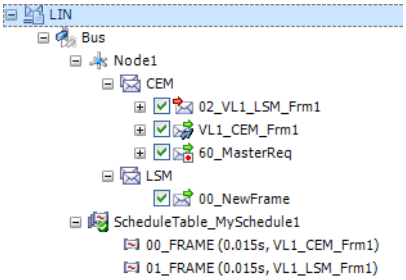


図 3-18 LIN ネットワーク

ネットワークの構造については、137 ページの「LIN ネットワークの編集」を参照してください。また、個々のコンポーネントについては、139 ページの「LIN ネットワークのコンポーネント」を参照してください。

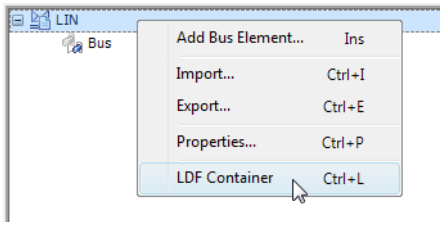
"LDF Container" ウィンドウ

ここには、インポートされた LDF ファイルの内容が表示されます。

LDF コンテナを表示する：

LDF コンテナが LIN エディタに自動表示されない場合は、以下の手順を実行してください。

- LIN ネットワークノードを右クリックします。
- ショートカットメニューから **LDF Container** を選択します。

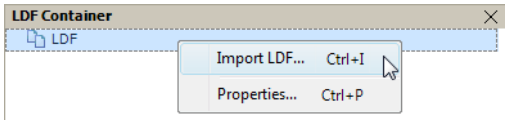


"LDF Container" ウィンドウが表示されます。

LDF ファイルをインポートする：

LDF ファイルのインポートは LDF コンテナのショートカットメニューから行い、LDF コンテナ内のエレメントは読み取りになります。インポートする LDF ファイルのプロトコルバージョンが LIN モジュールと適合していることを確認してください。

- コンテナを右クリックします。
- ショートカットメニューから **Import LDF** を選択します。



ファイル選択ウィンドウが開きます。

- インポートするファイルを選択します。

  選択されたファイルがインポートされ、その内容が LDF コンテナの下に表示されます。

  インポートされた LDF ファイルの名前が LDF コンテナに割り当てられます。
- 必要に応じて上記のステップ（ファイルの選択）を繰り返し、必要なファイルをすべてインポートします。
- インポートされている LDF ファイルを削除するには、コンテナのショートカットメニューから **Remove** を選択します。

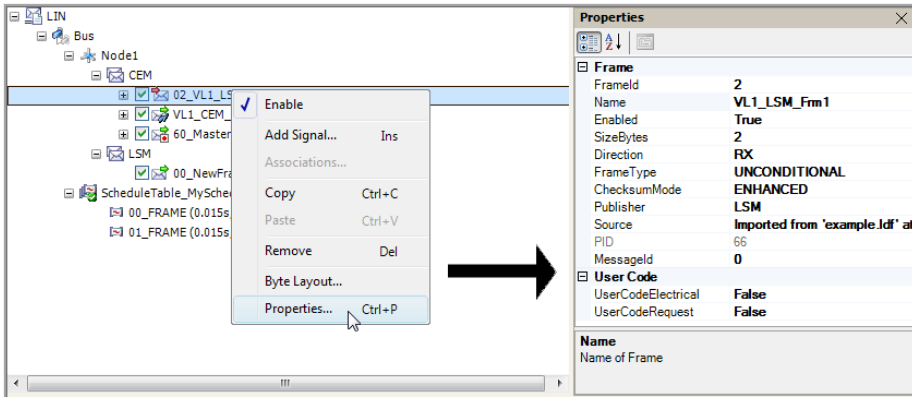
"Properties" ウィンドウ

ネットワークペイン内のエレメントを選択すると、そのエレメントのプロパティが "Properties" ウィンドウに表示されます。編集可能なプロパティは、ここで直接編集できます。

プロパティが自動的に表示されない場合は、以下のように操作して表示できます。

プロパティを表示する：

- プロパティを表示したいエレメントを右クリックします。
- ショートカットメニューから **Properties** を選択します。



選択されたエレメントのプロパティが "Properties" ウィンドウに表示されます。

プロパティの編集の詳細については、138 ページの「プロパティの編集」を参照してください。

フレームのバイトレイアウト

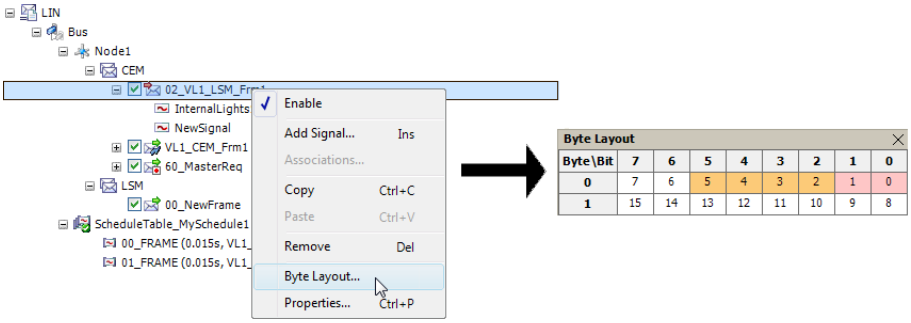
このエリアには、ネットワークビュー内で選択されているフレームの内容が表示されます。

バイトレイアウトを表示する

フレームのバイトレイアウトを表示するには、以下の手順を実行してください。

- レイアウトを表示したいフレームを右クリックします。
- ショートカットメニューから **Byte Layout** を選択します。

  フレームのバイトレイアウトが表示されます。



3.7.3 LIN ネットワークの編集

LIN ネットワークの編集は、ネットワークに含まれる各エレメントのショートカットメニューから行います。各エレメントについては 139 ページの「LIN ネットワークのコンポーネント」で説明します。

注記

LDF コンテナのエレメントのプロパティを編集することはできませんが、LDF コンテナのエレメントをコピーしたり移動したりして、同様のエレメントを LIN ネットワーク内に作成することはできます。

各エレメントのショートカットメニューにはネットワーク編集用のメニューコマンドが含まれ、一部のコマンドは、同じ種類の複数のエレメントを選択した状態で実行することができます。

エレメントを追加する：

- エレメントを追加したい上位エレメントを右クリックします。
- ショートカットメニューから **Add (Element)** を選択します。

注記

パート、フレーム、信号の名前に使用できるのは、ANSI-C 識別子として使用できる文字に限られます。

エレメントを削除する：

- 削除したいエレメントを右クリックして、ショートカットメニューから **Remove** を選択します。

エレメントをコピーする：

- コピーしたいエレメント（ソースエレメント）を右クリックします。
- ショートカットメニューから **Copy** を選択します。
- コピー先の上位エレメントになるエレメント（ターゲットエレメント）を右クリックします。
- ショートカットメニューから **Paste** を選択します。

エレメントのプロパティをコピーする：

- コピーしたいプロパティを持つエレメント（ソースエレメント）を右クリックします。
- ショートカットメニューから **Copy** を選択します。
- プロパティのコピー先となるエレメント（ターゲットエレメント）を右クリックします。
- ショートカットメニューから **Paste** を選択します。

エレメントを移動する：

- 移動したいエレメント（ソースエレメント）を、移動先の上位エレメントになるエレメント（ターゲットエレメント）までマウスでドラッグして離します。

注記

上述の各種操作は、その操作が可能なエレメントに対してのみ行えます。

プロパティの編集

LIN ネットワークのエレメントのプロパティは、"Properties" ウィンドウで編集することができます。ただし自動計算されるプロパティ（LIN フレームの PID など）はグレイアウト表示され、編集できません。LDF コンテナのエレメントのプロパティも編集できません。

"Properties" ウィンドウを開く：

- プロパティを編集したいエレメントを右クリックします。
- ショートカットメニューから **Properties** を選択します。

  LIN エディタ内に "Properties" ウィンドウが開きます。

複数のエレメントのプロパティを編集する：

ネットワーク内の同じ種類のエレメントを 2 個以上選択（ハイライト表示）すると、選択されているすべてのエレメントについて同じ設定が行われているプロパティの値のみが表示されます。

- "Properties" ウィンドウを開きます。
- **<Ctrl>** キーを押し下げたまま、複数のエレメントをクリックして選択します。

注記

2 個以上のエレメントを編集する場合、上位エレメントに含まれるエレメントを識別するためのプロパティ（フレーム内の信号名など）を変更する際に、入力した内容が同じ上位エレメントに属する別のエレメントのプロパティと同じ値になっていると、エラーメッセージが表示される場合があります。

妥当性確認

LIN モジュールを保存する際には、LIN ネットワークの妥当性が確認されます。その際には、たとえば正しいコード生成が可能なネットワーク構成になっているか、などがチェックされ、問題がある場合はログウィンドウにエラーがメッセージが表示されます。

注記

このエラーメッセージを無視すると、プロジェクトのビルド処理中にコンパイラエラーが発生したり、実行時に不正な挙動が生じる可能性があります。

モジュール

1 つの LABCAR-OPERATOR プロジェクトの中に複数の LIN モジュールを作成することができます。各モジュールのコンフィギュレーションは互いに無関係に設定できますが、1 つのモジュール内では 1 つの LIN ボードしか使用できません。

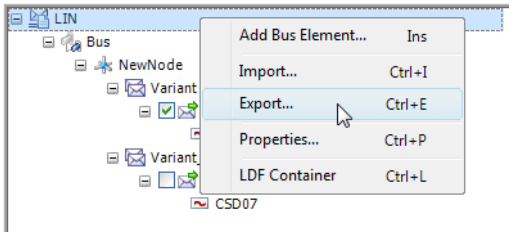
ネットワーク

LABCAR-OPERATOR LIN モジュール内では、ツリービューのルートノードが LIN ネットワークに相当し、バスエレメントはネットワークの下に配置されます。LIN ネットワークはインポートしたりエクスポートしたりすることができます。

モジュールのハードウェア構成は `LINConfiguration.xml` ファイルに格納されます。このファイルは、ネットワークに含まれるパート（`*.llp`）とスケジュールテーブル（`*.lsc`）のコンフィギュレーションファイルを参照し、これらのファイルはすべてモジュールディレクトリ内に格納されます。

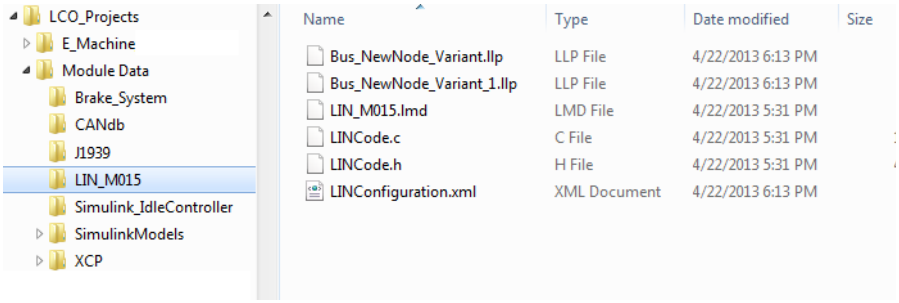
ネットワークをエクスポートする：

- LIN ネットワークを右クリックします。
- ショートカットメニューから **Export** を選択します。



- 既存のフォルダを選択するか新しいフォルダを作成して、**OK** をクリックします。

  LIN モジュールの名前と同じ名前のディレクトリに、各種ファイルが作成されます。



ネットワークをインポートする：

- LIN ネットワークを右クリックします。
- ショートカットメニューから **Import** を選択します。

  ファイル選択ウィンドウが開きます。
- ファイル `LINConfiguration.xml` を選択します。

注記

ユーザー定義の C コード（151 ページの「ユーザー定義 C コード」参照）を含むファイル（`*.c` と `*.h`）をエクスポートしたりインポートしたりすることはできません。このようなファイルが必要な場合はマニュアル操作で転送する必要があります。

3.7.4 LIN ネットワークのコンポーネント

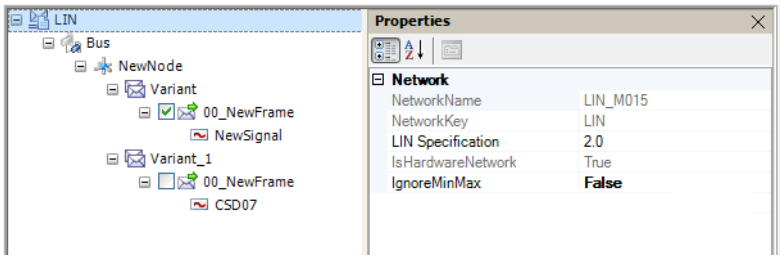
本項では、LABCAR-IP の LIN ネットワークの以下のコンポーネントについて説明します。

- LIN ネットワーク（140 ページ）
- バス（140 ページ）
- ノード（142 ページ）
- スケジュールテーブル（142 ページ）
- コマンド（143 ページ）
- パート（143 ページ）
- フレーム（145 ページ）

- 信号（シグナル）（148 ページ）

LIN ネットワーク

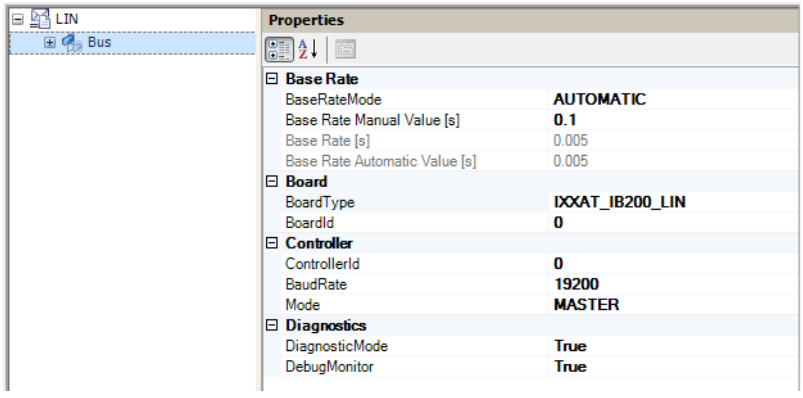
LIN ネットワークを選択すると、"Properties" ウィンドウにはその LIN ネットワークのグローバルプロパティが表示されます。



Network:

- **NetworkName**
  LIN ネットワークの名前
- **NetworkKey**
  ネットワーク ID
- **LIN Specification**
  LIN 2.0 または LIN 2.1/2.2
- **IsHardwareNetwork**
  ハードウェアが存在する実ネットワークか、それとも LDF ファイルに定義されたネットワークなのかを指定します。
- **Ignore Min/Max**
  信号値の下限値を無視することができます。このオプションが **True** になっていると、物理信号値が電気的な信号値に変換される際に、信号プロパティで設定された上下限値に制限されません。

バス



Base Rate:

コントローラの **Base Rate** プロパティは、（基本レート）プロパティは、選択された LIN モジュールコントローラと LABCAR-OPERATOR プロジェクトとの間の通信レートを定義するものです。

基本レートの値は、以下の各プロパティに従って決定されます。

- **BaseRateMode**
  Base Rate（基本レート）を決定する方法を選択します。

- **MANUAL**（マニュアル）
- **AUTOMATIC**（自動）

自動計算では、2 倍または 4 倍のオーバークロッキング（**AUTOMATIC2x** / **AUTOMATIC4x**）を選択できます。

- **Base Rate Manual Value [s]**
  任意に入力できる基本レートの値です。
- **Base Rate [s]**
  現在使用されている基本レートです。
- **Base Rate Automatic Value [s]**
  この値は、選択されているコントローラ下のすべてのコマンドの "Delay" プロパティの値とすべてのノードの **Base Rate** プロパティの値の最大公約数です。各ノードの **Base Rate** プロパティは、インポートされた LDF ファイルの "Master Time Base" パラメータで定義されますが、マニュアル入力で変更することができます。

Board:

- **BoardType**
  使用されているボードのタイプ（**IXXAT_XC16_LIN** / **IXXAT_IB200_LIN** / **IXXAT_IB600_LIN**）
- **BoardID**
  ネットワークコントローラ内でのボードの識別子

Controller:

- **ControllerID**
  ボード上のコントローラの識別子

注記

IXXAT_IB200_LIN では、チャンネル 1 がコントローラ 1 にマッピングされ、チャンネル 2 がコントローラ 0 にマッピングされます！

- **BaudRate**
  LIN コントローラの転送レート
- **Mode**
  MASTER または **SLAVE**

注記

MASTER モードのコントローラは、1 つのモジュールに 1 つしか使用できません。

Diagnostics:

以下のプロパティは、診断作業のサポートに関連するものです。

- **DiagnosticMode**
  このプロパティを **True** にすると、論理型の適合変数が生成されます。これを使用して、実験実行時の非診断フレームによる通信を拒否する（適合変数 = **True**）か、または許容する（適合変数 = **False**）かを指定することができます。

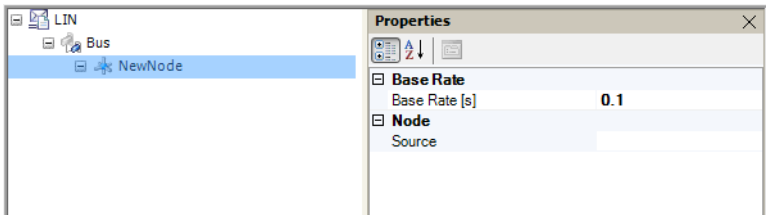
- **DebugMonitor**
  このプロパティを **True** にすると、測定変数が生成され、それを使用して実験実行時にモジュールとコントローラとの間の通信を監視することができます。

  送信フレームと受信フレームについて（マスタとスレーブ）、および要求フレームの送信フレームヘッダについて（マスタのみ）、それぞれ対応する測定変数が作成されます。これらの変数を見れば、前回転送されたすべてのデータバイトの内容がわかります。

注記

モジュールとコントローラとの間の通信は、設定されている基本レートよりも高いレートで行われ、1 サイクルごとに複数フレームのデータが交換されるので、上記の測定変数は、通信内容をすべて記録するのには適していません。

ノード



Base Rate:

時間挙動に関する以下のプロパティを使用できます。

- **Base Rate [s]**
  ノードの基本レートは、インポートされる LDF ファイルのパラメータ "Master Time Base" により定義され、マニュアル操作により変更することもできます。141 ページの「Base Rate Automatic Value [s]」コントローラの "BaseRateAutomaticValue" プロパティ（141 ページの「Base Rate Automatic Value [s]」参照）はこの基本レートを使用して計算されます。

Node:

- **Source**
  ノード（インポート後）のソース

スケジュールテーブル

スケジュールテーブルはマスタモードのコントローラに割り当てることができ、その下にコマンドを配置します（143 ページの「コマンド」を参照してください）。モジュール内では、スケジュールテーブルの内容は `.lsc` という拡張子の付いたファイルに格納されます。

スケジュールテーブル名はネットワーク内でユニークである必要があります。すでに使用されている名前を指定すると、名前の末尾に連番が付きます。

起動時の挙動を修正する：

- スケジュールテーブルを右クリックして、ショートカットメニュー内の **Use as Startup** をオンまたはオフにします。

注記

各コントローラ内の 1 個のスケジュールテーブルだけを "Startup" として指定できます。別のスケジュールテーブルが "Startup" として指定されると、前回の指定は取り消されます。

コマンド

コマンドはスケジュールテーブルに割り当てます。コマンドはスケジュールテーブル内に配置された位置で識別され、その順に処理されます。

位置を変更する：

- 位置を変更したいコマンドを右クリックします。
- ショートカットメニューから **Move Up** または **Move Down** を選択します。

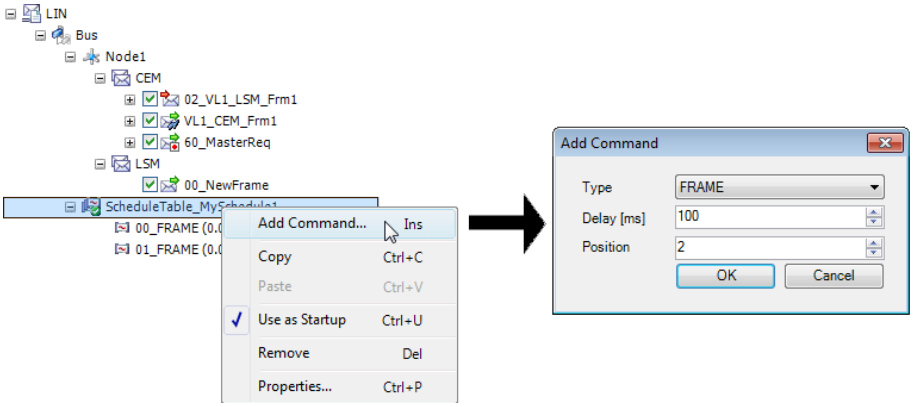
注記

コマンドの位置変更は、"Properties" ウィンドウからも行えます。1 つのコマンドの位置が変わると、それに応じて他のコマンドも移動します。コマンドの "Position" プロパティの値は、スケジュールテーブル内においてユニークです。

ユーザー定義コマンドを作成する：

コマンドは LDF ファイルからをインポートできるほか、ユーザー定義のものを作成することもできます。

- 作成されるコマンドを配置したいスケジュールテーブルを右クリックします。
- ショートカットメニューから **Add Command** を選択します。



注記

ユーザー定義コマンドを作成する際には、最初に **Delay** と **Position** という一般パラメータを定義し、その後、コマンドのタイプに応じたプロパティを "Properties" フィールドで定義します。

パート

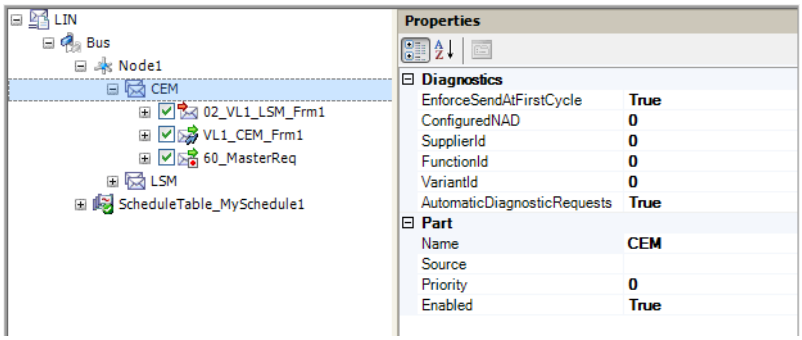
ノードのバリアントである「パート」は、ノードの直下に割り当て、その下にフレームを割り当てます。パートはインポートとエクスポートが可能です。各パートの内容は、それぞれ個別のファイル（`.llp`）に保存されます。

各パートには、ネットワーク内でユニークな名前を割り当てます。すでに使用されている名前を指定すると、名前の末尾に連番が付きます。

パートの優先順位付けと有効化／無効化

ネットワークノードの複数のバリアントをモデリングするには、割り当てられたすべてのフレームと信号を含むパートの複製を作成します。

各パートは **Enabled** プロパティを使用して有効化／無効化することができます。各パートのフレームと信号の処理は、**Priority** によって制御できます。優先度の高いパートは、各シミュレーション周期において優先度の低いパートの入力と出力を上書きします。



Part:

- **Name**
  パートの名前
- **Source**
  パート（インポート後）のソース
- **Priority**
  パートの優先順位を指定します。各パートは優先順位に基づいて実行されるので、この値はすべてユニークである必要があります。
- **Enabled**
  このプロパティにより、実験の実行時に各パートを個別に有効化／無効化することができます。無効になっているパートに含まれるフレームは、送信されません。

注記

実験実行中のバリアントの扱いには、**Enabled** および **Priority** プロパティを使用します。

Diagnostics:

以下のプロパティは、診断作業の実行に関連するものです。

- **EnforceSendAtFirstCycle**
  パフォーマンスの最適化のため、前回の転送以降に 1 つ以上のフレーム信号が変化した場合にだけ送信フレームのデータをコントローラ転送バッファに書き込むコードが生成されます。

  シミュレーションの最初の周期においては、定義済みの初期値が代入されたフレームが転送バッファに書き込まれますが、**EnforceSendAtFirstCycle** プロパティが **False** になっていると、この初期処理は行われません。この機能を利用すると、シミュレーション開始時にネットワーク通信に参加していないパートをシミュレートすることができます。
- **ConfiguredNAD**
- **SupplierId**
- **FunctionId**
- **VariantId**
  これらのプロパティは、ネットワーク内のパートによって定義されるノードを識別します。これは `ASSIGN_NAD` コマンドと `LIN PRODUCT ACTIVATION` コマンドの自動処理を行う場合に必要です。

- **AutomaticDiagnosticRequest**
  このプロパティを使用すると、パートについて、LIN 仕様に従って ASSIGN_NAD コマンドと LIN PRODUCT ACTIVATION コマンドの自動処理を行うコードを生成することができます。これらのコマンドの自動処理は、スレーブモードのコントローラでしか行われません。

注記

このプロパティが **True** になっているパートには、ネットワークアドレス **Current_NAD** のカレント値を示す測定変数が作成されます。

パートをエクスポートする：

- エクスポートしたいパートを右クリックします。
- ショートカットメニューから **Export Part** を選択します。
  ファイル選択ダイアログボックスが開きます。
- エクスポートファイルのパスとファイル名（*.llp）を指定して、**OK** をクリックします。
  パートの内容がファイルにエクスポートされます。

パートをインポートする：

- インポート先とするコントローラを右クリックします。
- ショートカットメニューから **Import Part** を選択します。
  ファイル選択ダイアログボックスが開きます。
- インポートするファイルのパスとファイル名（*.llp）を指定して、**OK** をクリックします。
  パートの内容がファイルからインポートされます。

フレーム

フレームはパートに割り当て、フレームに信号を割り当てます。

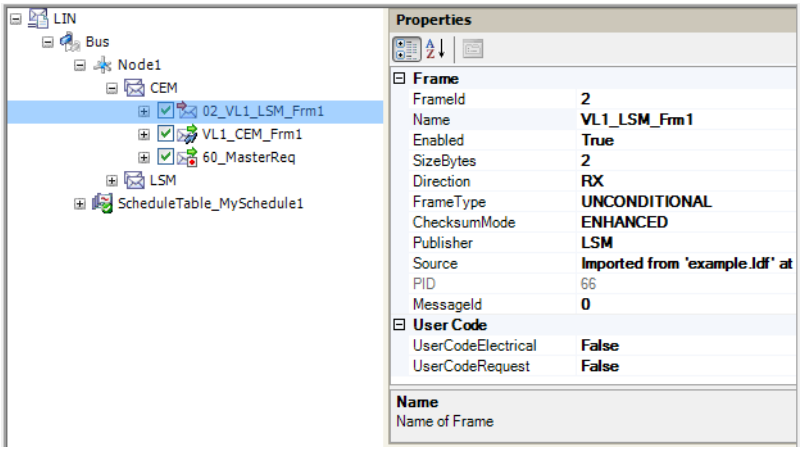
各フレームについては、"Properties" ウィンドウに加え "Byte Layout" ウィンドウが表示され、フレーム内の各信号の割り付けが示されます。

バイトレイアウトを表示する：

- フレームを右クリックします。
- ショートカットメニューから **Byte Layout** を選択します。

注記

バイトレイアウト内にエラー（2 つの信号が重複していたり、信号やフレームの長さが正しく指定されていない、など）があると、該当するビットが赤で表示されます。



Frame:

- **FrameId**
  フレームの ID
- **Name**
  フレームの名前
- **Enabled**
  シミュレーションにおいてフレームを有効化する（**True**）かしない（**False**）かを指定します。
- **SizeBytes**
  フレームのサイズ（バイト数）
- **Direction**
  転送方向（**TX** = 送信、**RX** = 受信）
- **FrameType**
  フレームのタイプ（**UNCONDITIONAL** / **SPORADIC** / **EVENTTRIGGERED** / **DIAGNOSTIC** / **USERDEFINED**）
- **ChecksumMode**
  チェックサム計算のモード（**CLASSIC** / **ENHANCED**）
- **Publisher**
  送信側のノードの名前（LDF からインポートされ、編集も可能）
- **Source**
  エレメントのソース
- **PID**
  フレームの保護識別子（計算値）
- **MessageId**
  ASSIGN_FRAME_ID コマンドを使用するための、フレームのメッセージ ID（任意）

User Code:

ここでは、ユーザー定義 C コードをフレームに追加するかどうかを指定できます（151 ページ「ユーザー定義 C コード」参照）。

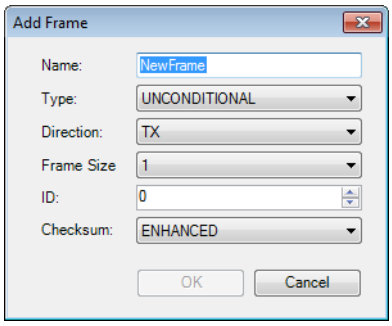
- **UserCodeElectrical**
  フレームの送信または受信を行う前に呼び出されるコードです。フレームのペイロードにアクセスするためのものです。

- **UserCodeRequest**
  フレーム要求の受信後に呼び出されるコードです。

ユーザー定義フレームを作成する：

LDF ファイルや、モジュール、パートからインポートされたもの以外に、ユーザー定義されたフレームを作成することもできます。

- フレームを作成したいパートを右クリックします。
- ショートカットメニューから **Add Frame** を選択します。



注記

1 つのパート内では、フレームの ID と名前の組み合わせは、ユニークである必要があります。

注記

フレームのパラメータは、すべての組み合わせが許容されているわけではありません。
たとえば、以下のプロパティを組み合わせると、自動的にフレーム ID が 60 になります。
- Type = DIAGNOSTIC
- Direction = TX（送信）
- コントローラモード

設定の組み合わせに応じて、入力フィールドが無効化されたり値の範囲が制限されたりします。入力内容がコンフィギュレーション内の深刻なエラーにつながりそうな場合は、ダイアログボックスの **OK** ボタンはグレイアウトされます。

フレームを有効化／無効化する：

1 つのパート内に、名前が異なっていて ID が同じである複数のフレームを定義することができます。ただしコード生成の際には、その中のいずれか 1 つのみを有効にして、他のフレームはコード生成から除外する必要があります。フレームの有効化／無効化は、以下のいずれかの方法で行います。

- **ネットワークツリービューを使用する場合**：フレームアイコンの左側のチェックボックスのオン／オフを切り替えます。
- **ショートカットメニューを使用する場合**：フレームを右クリックしてショートカットメニューからを開き、**Enable** コマンドを選択します。
- **"Properties" ウィンドウを使用する場合**：フレームの "Properties" ウィンドウを開き、**Enabled** プロパティ（**True** ／ **False**）を変更します。

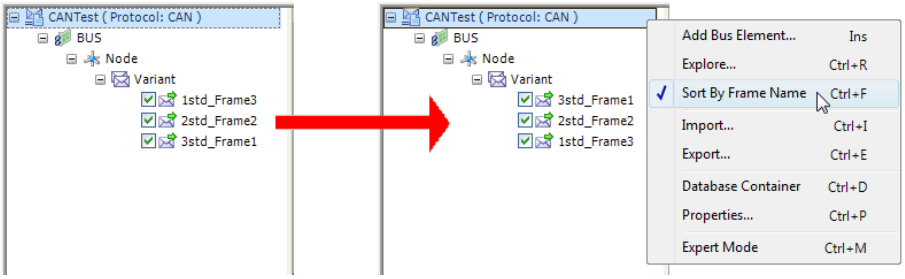
フレームを削除する：

- 削除したいフレームを右クリックします。
- ショートカットメニューから **Remove** を選択します。
  フレームが削除されます。

フレームを名前順にソートする：

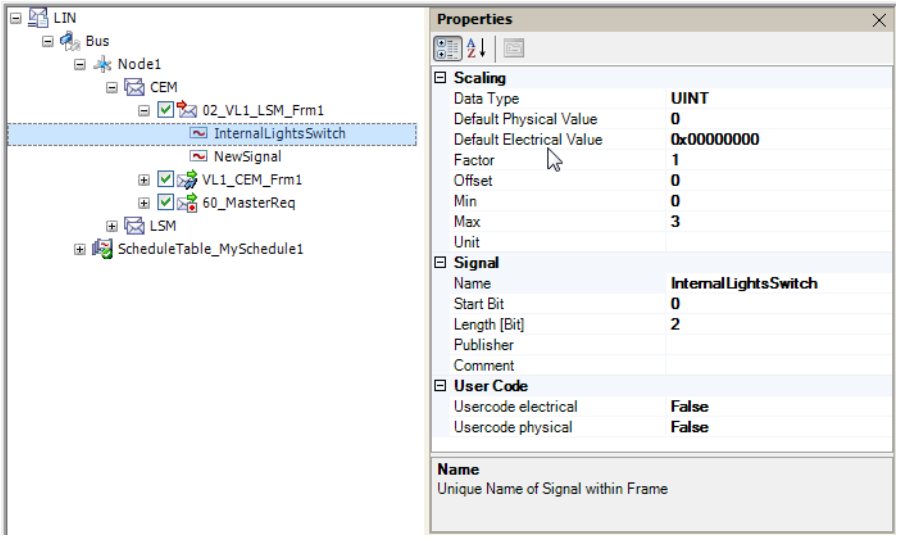
デフォルト状態において、フレームはフレーム ID の順で表示されます。

- フレームを名前でソートするには、最上位の CAN モジュールを右クリックして、ショートカットメニューから **Sort by Frame Name** を選択します。



信号（シグナル）

信号は、各フレームに割り当てます。各 1 つのフレーム内の信号の名前はすべてユニークである必要があります。



Scaling:

- **Data Type**
  "UINT"、"BOOLEAN" または "BYTEARRAY"

- **Default Physical Value**
- **Default Electrical Value**
  送信フレームの信号にはデフォルト値を定義することができます。ただしコネクションマネージャにおいて別の信号に接続されている信号については、この値は無視されます。

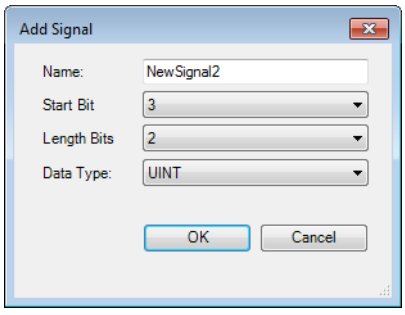
  デフォルト値を定義しておくと、LIN モジュールをエクスポートして別の LABCAR-OPERATOR プロジェクトにインポートした際、その値もインポートされます。
- **Factor / Offset**
  信号のスケーリングに使用する係数 a とオフセット b
- **Min / Max**
  妥当性チェックに使用する上下限値（118 ページの「最小値と最大値」参照）
- **Unit**
  信号の物理単位

Signal:

- **Name**
  信号の名前
- **Start Bit**
  信号の先頭ビット
- **Length**
  信号長（ビット数）
- **Publisher**
  送信側のノードの名前（LDF からインポートされ、編集も可能）
- **Comment**
  コメント（任意）

ユーザー定義信号を作成する：

- フレームを右クリックします。
- ショートカットメニューから **Add Signal** を選択します。



注記

信号のパラメータは、すべての組み合わせが許容されているわけではありません。たとえば、プロパティ **Data Type** を **BOOLEAN** にすると、**Length Bit** は自動的に **1** になります。
設定の組み合わせに応じて、入力フィールドが無効化されたり値の範囲が制限されたりします。入力内容がコンフィギュレーション内の深刻なエラーにつながりそうな場合は、ダイアログボックスの **OK** ボタンはグレイアウトされます。

アソシエーション

「アソシエーション」とは、無条件フレーム（**UNCONDITIONAL** フレーム）をイベントトリガフレーム（**EVENTTRIGGERED** フレーム）または散発フレーム（**SPORADIC**）にリンクさせるためのものです。

ネットワークツリービュー内では、アソシエーションは割り当てられたイベントトリガフレームまたは散発フレームの下に、無条件フレームへのリンクとして表示されます。

上の例は、2 つの無条件フレーム（ID 11 および ID 12）とのアソシエーションを持つイベントトリガフレーム（ID 13）を示しています。

イベントトリガフレームまたは散発フレームには複数の無条件フレームとのアソシエーションを持たせることができますが、これは同じパート内のイベントトリガフレームまたは散発フレームに限られます。そのため、無条件フレームを別のパートに移動すると、そのフレームへのアソシエーションはグレイアウト表示になります。

アソシエーションを生成する：

- 無条件フレームを、イベントトリガフレームまたは散発フレームにマウスでドラッグしてから離します。

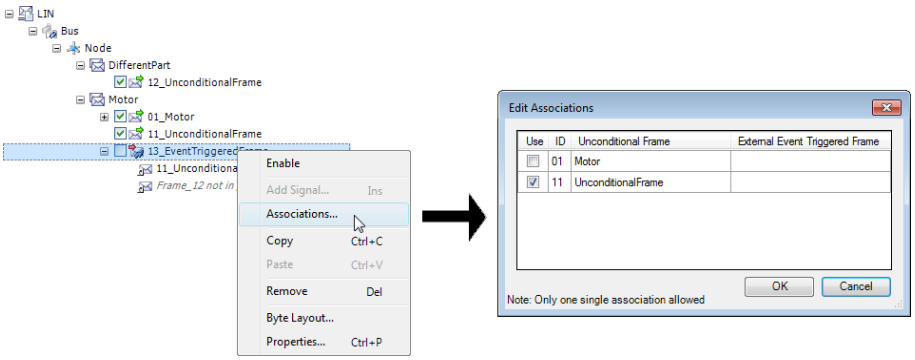
アソシエーションを削除する：

- 削除したいアソシエーションを右クリックします。
- ショートカットメニューから **Remove** を選択します。

アソシエーションを編集する：

- イベントトリガフレームまたは散発フレームを右クリックして、ショートカットメニューから **Associations** を選択します。

  "Edit Associations" ダイアログボックスが開きます。



- イベントトリガフレームまたは散発フレームとリンクさせる、同じパート内の無条件フレームを選択します。選択できる無条件フレームは 1 つだけです。

注記

パートの無条件フレームと他のパートのイベントトリガフレームまたは散発フレームとのアソシエーションはグレイアウト表示になります。これらのアソシエーションを変更するには、当該のイベントトリガフレームまたは散発フレームを編集する必要があります。

イベントトリガフレーム

無条件フレームとイベントトリガフレームとのアソシエーションにおいては、以下のルールが適用されます。

- イベントトリガフレームとのアソシエーションの場合、マスタとこのアソシエーションの影響を受けるスレーブの両方がこの関係を認識している必要があります。そのため、影響を受けるすべてのパート内に同じアソシエーションを定義する必要があります。
- イベントトリガフレームと無条件フレームの方向は、マスタモードのコントローラの下にあるパートでは RX（読み取り）、スレーブモードのコントローラの下にあるパートでは TX（送信）にする必要があります。

散発フレーム

無条件フレームと散発フレームとのアソシエーションにおいては、以下のルールが適用されます。

- 散発フレームとのアソシエーションの場合、送信側のパートだけがこの関係を認識する必要があり、受信側はその必要はありません。
- この種類の関係が発生するすべてのパートで、散発フレームと無条件フレームの方向は TX（送信）にする必要があります。

3.7.5 ユーザー定義 C コード

LABCAR-NIL では、自動生成されるメッセージコードにユーザー定義の C コードを追加して、フレームや信号の内容を操作したり拡張したりすることができます。信号の場合、物理値（スケーリングされ物理単位が付いた値）と電気値（スケーリングされず物理単位が付いていない値）のどちらについても可能です。

- **送信フレーム**
  フレームの "UserCodeElectrical" プロパティを使用して、送信直前にフレームのペイロードを操作することができます。
- **受信フレーム**
  フレームの "UserCodeElectrical" プロパティを使用して、受信直後にフレームのペイロードを操作することができます。
- **フレームヘッダ**
  マスタモードのコントローラの場合、フレームの "UserCodeRequest" プロパティを使用して、送信直前に要求フレームのヘッダを操作することができます。
- **送信信号**
  信号の "UserCodePhysical" プロパティを使用して、入力ポートから double 型の値として読み取った直後の信号を操作することができます。

  信号の "UserCodeElectrical" プロパティを使用して、uint64 型の電気値（スケーリングされず物理単位が付いていない値）としてフレームのペイロードに追加される直前の信号を操作することができます。

- **受信信号**
  信号の "UserCodeElectrical" プロパティを使用して、uint64 型の電気値（スケーリングされず物理単位が付いていない値）としてフレームのペイロードから抽出された直後の信号を操作することができます。

  信号の "UserCodePhysical" プロパティを使用して、LIN モジュールの出力ポートに double 型の物理値（スケーリングされ物理単位が付いた値）として書き込まれる直前の信号を操作できます。

手順

LIN コンフィギュレーションが保存されると、モジュールディレクトリには `LINCode.h` というヘッダファイルが格納されます。このファイルにはユーザーコードを挿入するための関数宣言が含まれています。

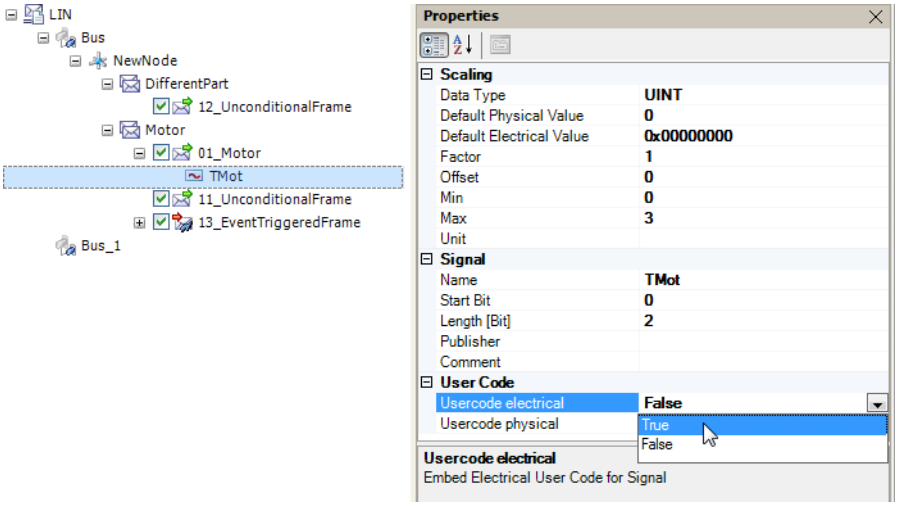
`LINCode.h` ファイルの最後の部分は、以下のようなコメント行になっています。

```
// Declarations for user code insertion:
```

このコメント行の下に、ユーザーコード内に定義されたユーザー関数の呼び出しを宣言します。

ユーザーコードを宣言する：

- ユーザーコードを追加したいフレームと信号について、以下のプロパティを編集します。
  - "UserCodePhysical"
  - "UserCodeElectrical"
  - "UserCodeRequest"

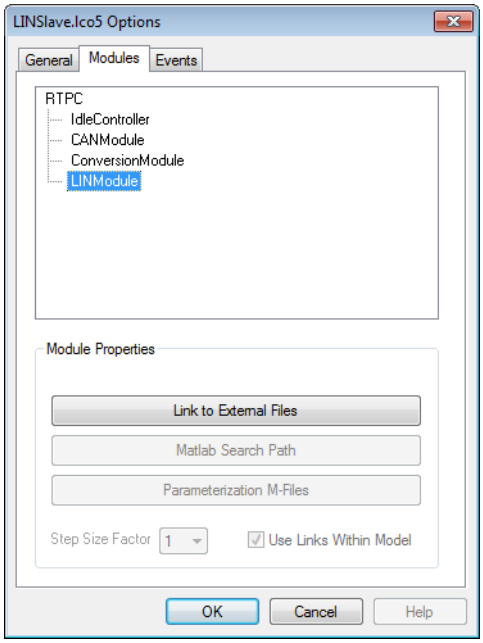


- **File → Save** を選択してモジュールを保存します。
- テキストエディタなどを使用して、モジュールディレクトリ内の `LINCode.h` ファイルを開きます。
- ユーザーコードの関数宣言をテキストファイルにコピーします。
- このテキストファイルに適切な名前（例：`UserCode.c`）な名前を付け、LIN モジュールディレクトリに保存します。

- 関数と構造体の宣言を認識させるため、ユーザー C ファイル `UserCode.c` に以下の行を追加します。

  ```
  #include "LINCode.h"
  ```

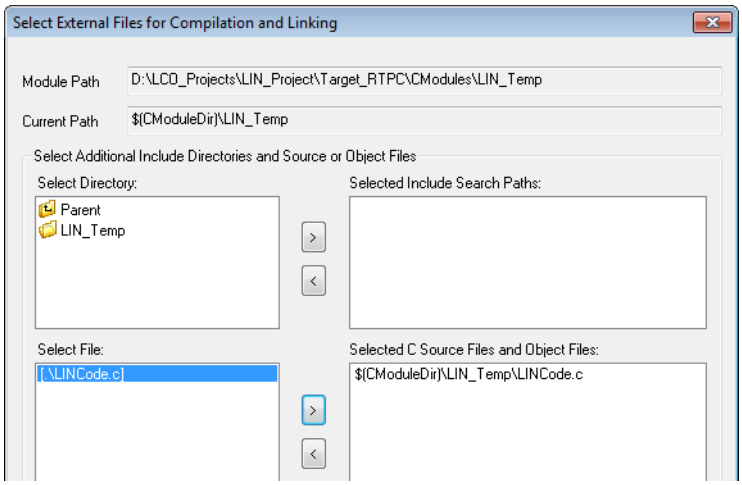
- ユーザーコードを `UserCode.c` に実装します。
- メインメニューから **Project → Options** を選択します。
- "Options" ダイアログボックスの "Modules" タブを選択します。



- "LINModules" を選択して **Link to External Files** をクリックします。

  "Select External Files for Compilation and Linking" ダイアログボックスが開きます。

- 組み込むソースファイル（複数可）を選択します。



注記

選択できるのは、モジュールディレクトリ内のファイルのみです。

選択されたファイルが "Selected C Source Files and Object Files" フィールドに表示されます。

- **OK** をクリックします。

  以後、コンパイルとリンクが行われる際は、これらのファイルを含めた処理が行われます。

- **OK** をクリックして "Options" ダイアログボックスを閉じます。

データ型と構造体

ユーザーコードの関数宣言においては、フレームを操作するための以下のデータ型と構造体を宣言します。

定数

```
/* 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
```

データ転送：送信と受信

```
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;
```

データ転送：受信（リターンエラー）

```
typedef struct {
   unsigned short errorcode;     // LIN error code (receive only)
} ixxat_lin_load_error_t;
```

データ転送：受信（リターンステータス）

```
typedef struct {
   unsigned short status;        // LIN status (receive only)
} ixxat_lin_load_status_t;
```

データ転送：受信（要求）

```
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;
```

データ転送：送信と受信（フレームのペイロード）

```
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 LABCAR-EE における LIN モジュール

本項では、LABCAR-EE（実験環境）で LIN モジュールを操作する方法について説明します。

作成したプロジェクトをターゲット上でコンパイルすると、プロジェクト内の各 LIN モジュールについて、制御や通信を行うためのさまざまな入力と出力が生成されます。エクスペリメントエクスプローラでは、これらは各モジュールの下に表示されます。

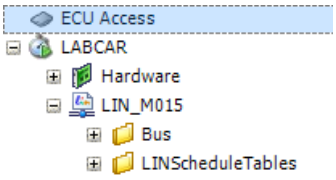
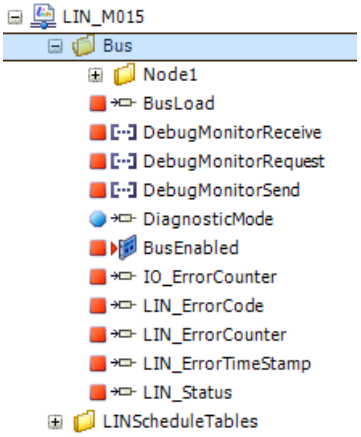


図 3-19 "Workspace Elements" ウィンドウ内の LIN モジュール

"Workspace Elements" ウィンドウにおいて各 LIN モジュールは、それぞれモジュール名の付いたノード下に表示されます。このモジュールノードの下には、設定されているバスエレメント用のバスエレメント（"Bus" フォルダ）が表示されます。バスエレメントまたはそれに含まれるエレメントを制御するための入力と出力は、すべてこのバスエレメントノードの下に表示されます。

バス

LIN コントローラとの通信の制御と監視を行うための入力と出力は、"Bus" フォルダの下に表示されます。



- **BusLoad**
  LIN バスの現在の負荷（パーセント）

- **DebugMonitorReceive**
- **DebugMonitorRequest**
- **DebugMonitorSend**
  LIN モジュールとコントローラとの間で最後に転送されたバイトデータを表示するための測定変数

注記

モジュールとコントローラとの間の通信は、設定されている基本レートよりも高いレートで行われ、1 サイクルごとに複数フレームのデータが交換されるので、上記の測定変数は、通信内容をすべて記録するのには適していません。

- **DiagnosticMode**
  コントローラの診断モードを有効化／無効化するための適合変数
- **BusEnabled**
  バスの有効化／無効化を行うための入力ポート
  - - 0.5 < 入力ポートの値 < + 0.5 = False：バスは無効化されます
  - 上記以外は True（デフォルト = 1.0）：バスは有効化されます

注記

バスの起動／停止は、それぞれのボードにより制御されます。ある 1 つコントローラの通信ステートが変化すると、他のコントローラのバスへの干渉が生じる可能性があります。このような干渉は、ステートが変化する限り続きます。

- **IO_ErrorCounter**
  選択されたコントローラとの通信に関するエラーカウンタ

  通信エラーが頻繁に発生する原因としては、"BoardId" プロパティと "ControllerId" プロパティの不適切な設定や、LIN ボードファームウェア内のエラーが考えられます。
- **LIN_ErrorCounter**
  IXXAT ドライバによる LIN 通信のエラーに関するエラーカウンタ
- **LIN_ErrorTimeStamp**
  前回 "LIN_ErrorCode" が発生した時のタイムスタンプ。タイムスタンプは、LIN 通信開始からの経過時間（単位はマイクロ秒）で表され、LIN ボードにより決定されます。

- **LIN_ErrorCode**
  LIN プロトコルによる通信のエラーコード

  通信エラーが頻繁に発生する原因としては、LIN ネットワークのエレメントの時間挙動やその他のプロパティの不適切な設定が考えられます。

```
#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**
  LIN ボードとの通信のステータスに関するエラーコード

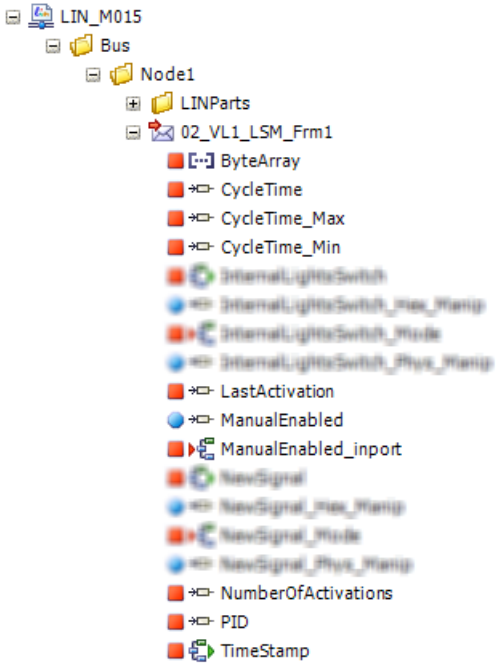
```
// status 0x00 means communication is running
#define BCI_LIN_STATUS_OVRRUN  0x01 /* data overrun occurred */
#define BCI_LIN_STATUS_ININIT  0x10 /* init mode active */
```

ノード

バスの下には LIN ネットワーク内の個々のノードが表示され、その下に、ノードやノード内のエレメントを制御するための入力と出力が表示されます。

フレーム

ノードの下には、LIN ノードのすべてのパートに含まれるフレームが表示されます。



- **ByteArray**
  フレームのペイロードのバイト配列
- **CycleTime**
- **CycleTime_Max**
- **CycleTime_Min**
  フレームの平均／最大／最小周期（単位はマイクロ秒）。周期はフレームの起動から次の起動までの時間です。
- **ManualEnabled**
- **ManualEnabled_inport**
  実験実行時にフレームを有効化／無効化するための適合変数と入力ポート
- **LastActivation**
- **NumberOfActivations**
  フレームの前回起動時のタイムスタンプ（単位は秒）と起動回数

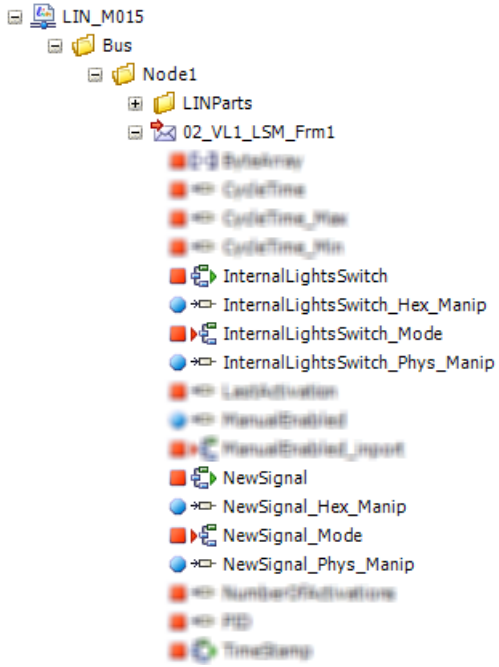
  LIN バスに対してフレームの読み書きが行われている場合、そのフレームは「起動されている状態」とみなされます。"LastActivation" の決定にはリアルタイム PC のプロセッサクロックが使用されます。
- **PID**
  "FrameId" から自動計算される、フレームの保護識別子

受信フレームの場合は "TimeStamp" シグナルも作成されます。

- **TimeStamp**
  フレームの前回の受信時のタイムスタンプ（マイクロ秒）。タイムスタンプは LIN ボードにより決定されます。

信号（シグナル）

信号制御用の入力と出力は、フレームの下に作成されます。



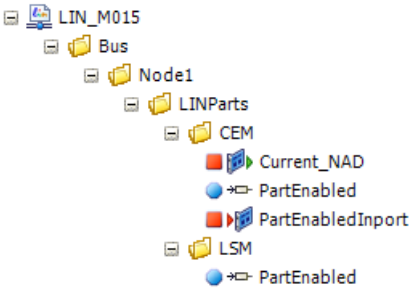
- **<SignalName>**
  信号の物理値を定義します。LIN バス上の信号の値か、マニュアル入力された代用値のいずれかが使用されます。
- **<SignalName>_Mode**
  <SignalName> の値を決定するための選択モード

```
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**
  マニュアル入力された <SignalName> の代用値（電気値）
- **<SignalName>_Phys_Manip**
  マニュアル入力された <SignalName> の代用値（物理値）

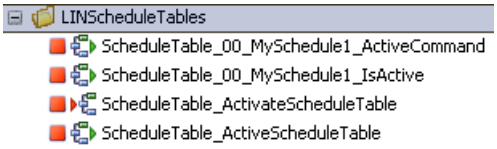
パート

各パートは、同名のフォルダとして表示されます。



- **Current_NAD**
  "AutomaticDiagnosticRequest" プロパティがオンになっているパートの "NAD" プロパティのカレント値
- **PartEnabled**
  パートを有効化／無効化するための適合変数
- **PartEnabledInport**
  パートの起動／停止を行うための入力ポート

LINScheduleTables（マスタモードのコントローラの場合のみ）

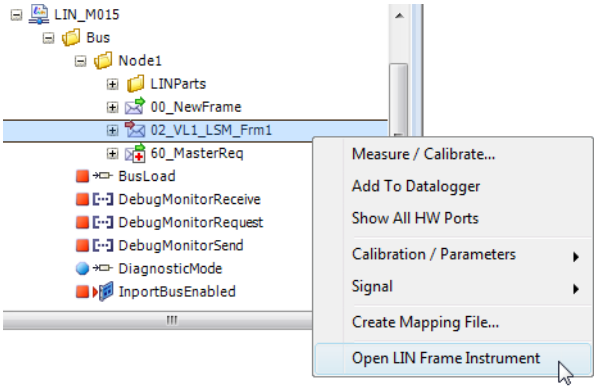


- **<ScheduleTableIndex>_<ScheduleTableName>_ActiveCommand**
  スケジュールテーブル内で現在実行されているコマンドの位置
- **<ScheduleTableIndex>_<ScheduleTableName>_IsActive**
  選択されているスケジュールテーブルの現在の実行ステータス（0 = 非起動状態、1 = 起動状態）。
- **ActivateScheduleTable**
  起動されるスケジュールテーブルの "Schedule Table Index" のエントリ

- **ActiveScheduleTable**
  現在起動されているスケジュールテーブルの "Schedule Table Index" の出力

3.7.7 LIN フレーム用のインストゥルメント

メッセージ表示用のインストゥルメントを作成するには、メッセージを右クリックしてショートカットメニューを開き、**Open LIN Frame Instrument** を選択します。



選択されたメッセージの内容を表示するインストゥルメントが作成されます。

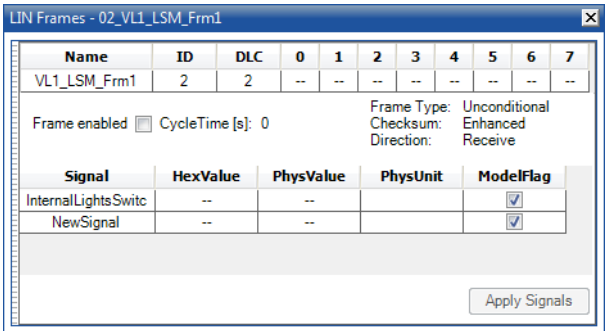


図 3-20 LIN フレーム用のインストゥルメント

受信メッセージの場合、メッセージとその信号の内容が表示され、信号の編集は行えません。

送信メッセージの場合は、**ModelFlag** オプションを無効にすると信号の値（HEX 値または物理値）を編集することができます。変更内容を確定するには **Apply Signals** をクリックします

3.8 FlexRay モジュール（LABCAR-NIF V5.4.0 ― FlexRay ネットワークの統合）

LABCAR-OPERATOR V5.4.0 のアドオン製品である LABCAR-NIF V5.4.0（Network Integration FlexRay）は、FlexRay 通信を含む ECU ファンクションのテストを容易に行うための機能を提供するものです。

この機能を利用するには、まず FlexRay バス通信の使用を、Elektrobit 社の EB tresos Busmirror というソフトウェアで作成されたデータモデルから読み込みます。続いて、「NIF モジュール」（FlexRay モジュール）としてシミュレートされるバス部分のソースコードが LABCAR-NIF V5.4.0 によって自動的に生成されます。生成されたコードにはユーザーコード（メッセージカウンタ、チェックサム計算など）を追加することができます。

FlexRay バス上の各信号に相当する NIF モジュールの信号は、コネクションマネージャに表示され、ここで他のモジュールの信号に接続できます。

注記

LABCAR-NIF V5.4.0 は "StringTemplate.NET" および "ANTLR" のライブラリを使用します。そのため、346 ページの「付録」に記載されているライセンス条件を考慮する必要があります。

ハードウェア要件

FlexRay バスシミュレーションを実行するには、Real-Time PC シミュレーションターゲット上に、Elektrobit Automotive 社の PCI ボード EB 5100 または EB 5200 がが 1 枚以上インストールされている必要があります。異なるタイプのボートを混在させることは可能で、1 枚のボードが 1 つの NIF モジュールに対応します。

この FlexRay インターフェースは ETAS からもご購入いただけます。品名と品番は以下のとおりです。

品名	品番
EB 5100 Elektrobit FlexRay Interface Solution	F-00K-106-407
EB 5200 Elektrobit FlexRay Interface Solution	F-00K-108-467

ソフトウェア要件

バスシミュレーションを行う際にインポートするファイルを作成するには、"EB tresos Busmirror" ソフトウェア（4.6.x 以降）のライセンスを Elektrobit 社から取得する必要があります。

注記

ETAS RTPC オペレーティングシステムの 64 ビットへの移行に伴い、4.6 より前のバージョンは使用できなくなりました。旧バージョンの LABCAR-OPERATOR プロジェクトを現行バージョン用に移行すると、EB tresos のバスミラーバージョンはサポートされなくなり、変更できなくなります。これらのモジュールは、編集や Real-Time PC 上での実行は行えなくなるので、新しい NIF モジュールに置き換える必要があります。

3.8.1 EB tresos Busmirror によるファイルの作成

LABCAR-OPERATOR に FlexRay バスコンフィギュレーションを統合するには、データモデルとバイナリファイル（EB 5100 または EB 5200 ボードで実行できるもの）を EB tresos Bursmirror プロジェクトからインポートする必要があります。

- `BMCfg.tdb`

- `BUSMIRROR_TypeConversion.tcs`
- `Firmware.ttc`

これらのファイルの作成方法は、EB tresos Busmirror の資料を参照してください。

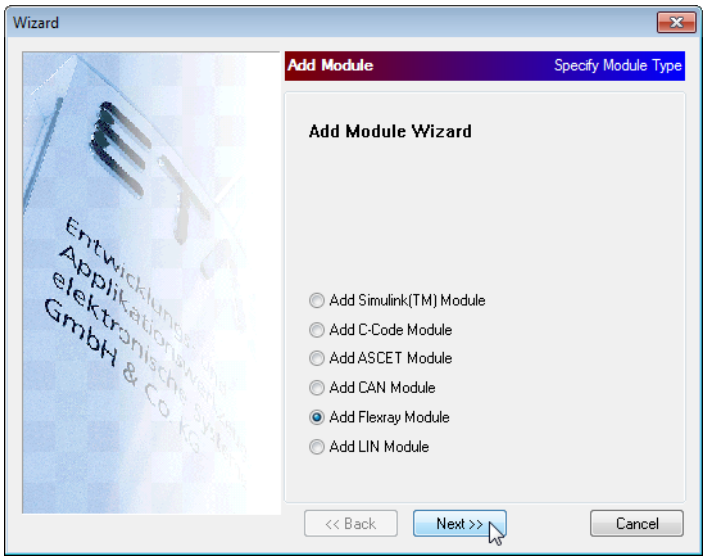
3.8.2 FlexRay モジュールの統合

EB tresos Busmirror でデータモデルと実行形式のバイナリファイルを生成したら、それらを以下の手順で LABCAR-OPERATOR プロジェクトに統合します。

FlexRay モジュールを作成する：

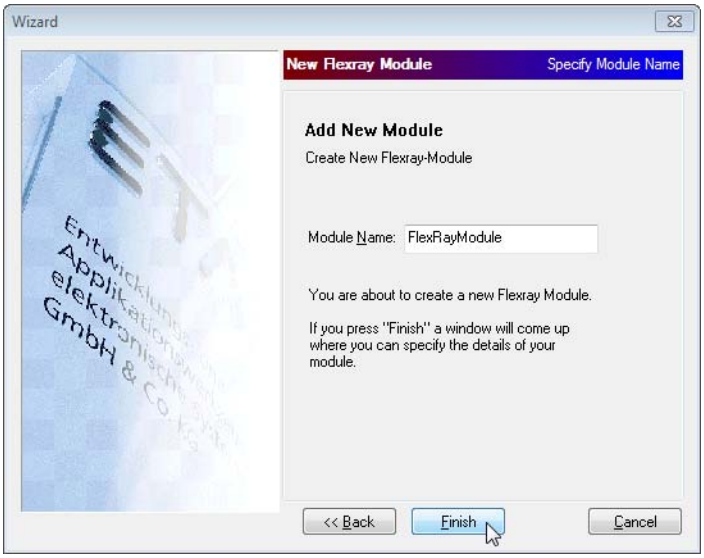
- FlexRayモジュールを作成したいLABCARプロジェクトを開きます。
- プロジェクトエクスプローラで、ターゲットを選択して右クリックします。
- ショートカットメニューから **Add Module** を選択します。

  モジュール追加ウィザードが開きます。



- **Add Flexray Module** を選択します。
- **Next** をクリックします。

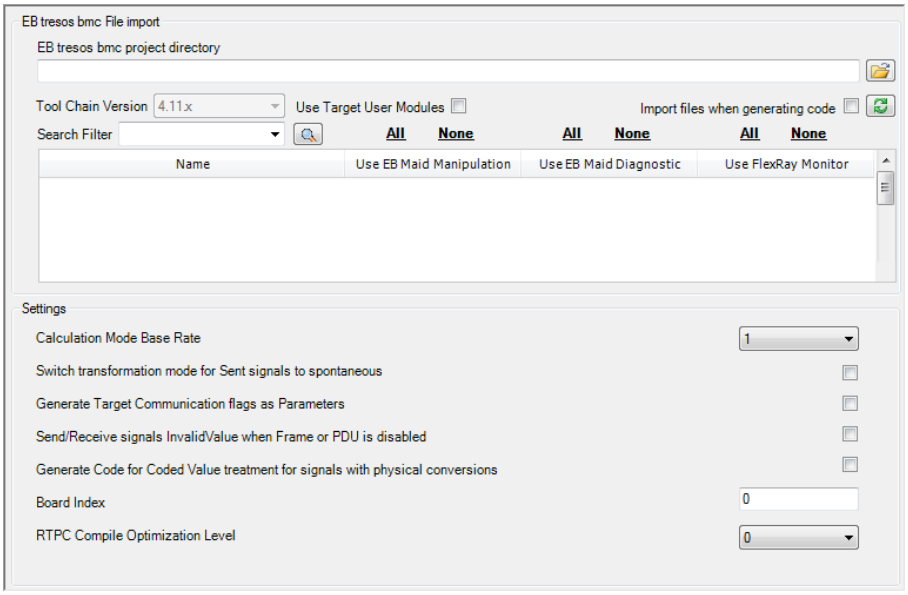
- 次のページで、作成するモジュールの名前を入力します。



- **Finish** をクリックします。
  LABCAR-IP のメインウィンドウに新しいタブが追加されるので、ここで必要なファイルをインポートします。

3.8.3 NIF モジュールの EB tresos bmc ツールスイートへのリンク

NIF モジュール用ウィンドウペイン上部の "EB bmc tresos File import" グループフィールドで、NIF モジュールの EB tresos bmc ツールスイートへのリンク情報を設定します。



- "EB tresos bmc project directory" フィールドで、データモデルと実行形式のバイナリファイルを FlexRay モジュールにインポートする EB tresos プロジェクトを選択します。
- FlexLay モジュールを保存します。

EB tresos bmc ファイルの同期と NIF モジュールソースコードの生成を行う：

モジュールの保存や外部ファイルの同期が行われると、必ずソースコードが生成されます。NIF モジュール用のソースコード生成時には、一般に、インポートされた EB tresos bmc ファイルのワーキングコピーがアクセスされます。同期メカニズムにより、これらの外部ファイルを NIF モジュールに再インポートすることができます。

この手順は、以下のようにして自動化することができます。

- 自動同期機能を有効にするには、**Import files when generating code** オプションを使用します。
  - オン：コードが再生成される前に必ずファイル同期され、同期されたファイルを使用してコードが生成されます。
  - オフ：ファイルの同期は、マニュアル操作でのみ行われます。ボタン操作によって同期とコード生成が行われます。

注記

ユーザー定義コードが NIF モジュール内で使用される場合、そのコードはコード生成時に NIF モジュールのソースコードに統合されます。

ターゲットユーザーモジュールを統合する（EB MAID）：

EB tresos bmc では、ターゲットユーザーモジュール（TUM）を使用して実行形式のバイナリファイルに機能を統合することができます。TUM のインターフェースが格納された TUM ディスクリプションを使用することにより、NIF モジュールに追加するソースコードを生成して、実行時に TUM と通信を行うことを可能にすることができます。

- "Use Target User Modules" オプションをオンにして TUM のサポート機能を有効にしてください。

注記

このオプションがオフになっていると、TUM ディスクリプションは無視され、EB MAID サポート用のコード生成は行われません。

3.8.4 モジュールコンフィギュレーション

NIF モジュール用ウィンドウペイン下部の "Settings" グループフィールドで、NIF モジュールの各種オプションを設定します。

Real-Time PC 上の NIF モジュールのクロックタイムを調整する：

送受信タスクのクロックタイムは通常 1ms で、NIF モジュールとボード上のアプリケーションとの信号交換を行うために必要な呼び出しは、5 つの計算スロットに均等に配分されます。

送受信タスクが実行されるたびに 1 つの演算スロットが実行され、信号交換全体におけるそのスロットの分担率は、NIF モジュールとボードの間で同期されます。個々のフレームまたは PDU の信号を交換するための呼び出しも、FlexRay のフレームトリガリングのサイクル反復により調節されるので、Real-Time PC とボードは可能な限り均等に使用されます。

つまり、信号交換全体は 5ms 経過後に完了します。クロックタイムを変更すると、パフォーマンスが最適化されるか（クロックタイム > 1ms の場合）、または同期化が向上する（クロックタイム < 1ms の場合）可能性があります。

- "Settings" フィールドの "**Calculation Mode Base Rate** オプションで、周期用の除数を選択します。

  選択可能な値：**1/8、1/4、1/2、1、2、4、8**

  1/8 = パフォーマンス向上（実質的には 8ms）

  8 = 同期化向上（実質的には 125 µs）

注記

フレームまたは PDU の送受信は、実際には NIF モジュールのクロックタイムの整数倍のタイミングで実行されます。フレームまたは PDU のクロックタイムは、それぞれのデータモデルから取得されます。

パフォーマンスを最適化し、変化していない信号値の送信を回避する：

送信信号交換用の関数は、実行回数を最小限に抑えるため、送信信号の値が前回呼び出された時の値と異なる場合だけ呼び出されます。これによりパフォーマンスを最適化することができます。

- **Switch transformation mode for Sent signals to spontaneous** をオンにして最適化オプションを有効にします。

ターゲット通信フラグをパラメータとして生成する：

Generate Target Communication flags as Parameters オプションをオンにすると、フレームまたは PDU の通信制御やバス通信制御の目的で、フレームまたは PDU ごとに制御シグナル "Enable" および "Idle" が生成され、バス通信については "CommEnable" と "HardBoundaries" が生成されます。このオプションは、これらのシグナルを入力ポートとして生成するか、それともパラメータとして生成するかを選択するためのものです。

受信 PDU の前回変更時のタイムスタンプを出力するためのシグナルは、常に出力ポートとして生成されます。

ペイロードが無効な場合に信号値を InvalidValue にする：

Send/Receive signals InvalidValue when Frame or PDU is disabled オプションは、実行時においてフレームまたは PDU が "Enable" シグナルによって無効化されているときに、各信号の代用値として FlexRay データモデルの "InvalidValue" を送受信するかどうかを制御します。

Real-Time PC 内の複数の EB 5100 ／ EB 5200 をサポートする：

この機能を利用して、複数の独立した LABCAR-NIF モジュールを生成することができます。EB 5100 と EB 5200（LABCAR-OPERATOR プロジェクトのディレクトリ構造内のもの）はボードインデックスにより識別されます。

- **Board Index** オプションにインデックスを入力します。

コンパイル最適化レベルを指定する：

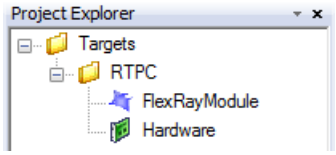
RTPC Compile Optimization Level オプションを使用して、Real-Time PC 上のビルド環境の最適化レベルを指定できます。このオプションは Real-Time PC の web インターフェースから設定できる最適化レベルに相当しますが、NIF モジュール用に生成されるソースコードにのみ適用されます。

3.8.5 NIF モジュールの LABCAR-OPERATOR プロジェクトへのリンク

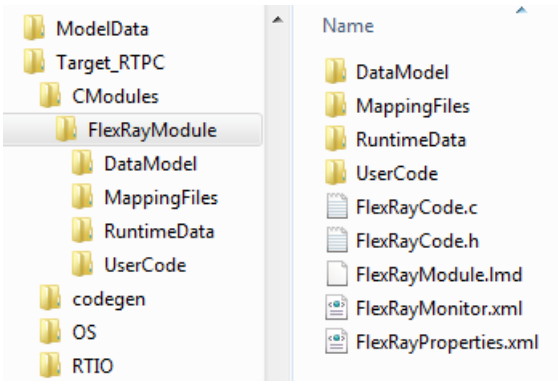
モジュールコンフィギュレーションを保存する：

- **File → Save** を選択します。

  モジュールコンフィギュレーションが保存され、モジュールのソースコードが生成されます。モジュールは LABCAR-OPERATOR プロジェクトに統合され、プロジェクトエクスプローラに表示されます。



コード生成が正常終了すると、NIF モジュール用に以下のファイルとディレクトリを含むディレクトリが作成されます。



OS コンフィギュレーションとコネクションマネージャを更新する：

- "OS Configuration" タブに切り替えます。
- **Update Processes** をクリックします。

  NIF モジュール用に作成されたタスクとプロセスが表示されます。
- "Connection Manager" タブに切り替えます。
- **Update Ports** をクリックします。

  データモデルのすべての入力（送信ペイロード）と出力（受信ペイロード）が表示されます。

モジュールをコンパイルする：

- **Project → Build** を選択します。

または

- **Build LABCAR Project** アイコンをクリックします。

3.8.6 ユーザー定義コードとカスタマイズ

データモデルから自動生成された NIF モジュールソースコードの拡張機能として、ハンドコーディングされたソースコード（「コードフラグメント」）を統合することができます。コードフラグメントは、生成されたコード内に配置されたダミーの部分に挿入します。

コードフラグメント用のファイル

NIF モジュールディレクトリ内には、ハンドコーディングされたソースコードを統合するためのサブディレクトリがあります。

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

このディレクトリ内にあってファイル名が `.fragment.c` で終わっているファイルは、すべてコードフラグメントとして評価されます。これらのファイルの形式は以下のとおりです。

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

注記

ユーザー定義コードが NIF モジュール内で使用される場合、そのコードはコード生成時に NIF モジュールのソースコードに統合されます。

コードロケーションインジケータ

コードフラグメントの開始位置は、コードロケーションインジケータにより指定されます。コードロケーションインジケータは、ソースコード内のコメント行に大括弧付きで指定されます。

コードロケーションインジケータの後に、統合されるソースコードが続きます。次のコードロケーションインジケータの前までのコードが、コードジェネレータにより NIF モジュールに挿入されます。コードロケーションインジケータの形式には以下のバリアントがあります。

グローバルコードロケーションインジケータ

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

コードポイント（`CodePoint`）は、生成されるソースコード内の、ユーザーコードの挿入位置を示すテキストマーカーです。グローバルテキストマーカーは、生成されるソースコード内に 1 回しか使用できません。グローバルテキストマーカーの場合、位置を指定するための情報を追加する必要はありません。

このバリアントには以下のテキストマーカーを使用できます。

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

コンテキストとオブジェクトを伴う特定コードロケーションインジケータ

その他のテキストマーカーとしては、ペイロード用のものやシグナル用のものなどがあります。これらのテキストマーカーについては、コードロケーションインジケータが指すコンテキストとオブジェクトを以下のように指定する必要があります。

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

このバリアントには以下のテキストマーカーを使用できます。

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

テキストマーカーのコンテキスト（`context`）には、プログラム内の以下の処理を指定できます。

```
SendSignal
ReceiveSignal
SendPdu
ReceivePdu
```

最後に、テキストマーカーが指すオブジェクト（`object`）を指定します。その際には信号や PDU の名前を使用することができます。

注記

`UserCode` ディレクトリ内の `SignalList.txt` および `PduList.txt` というファイルの中に、すべての名前のリストが生成されます。コード生成時に、有効なすべてのコードロケーションインジケータが、コメント行のテキストマーカーに従って NIF モジュールのソースコードに挿入されます。

コンテキスト、オブジェクト、関数を伴う特定コードのロケーションインジケータ

その時の状態に応じて実行する関数が異なるオブジェクトの場合、さらに詳細な指定ができます。

現時点では、これは "ToHost" を送信する EB MAID のターゲットユーザーモジュールについてのみサポートされています。

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

このバリアントには以下のテキストマーカーを使用できます。

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

現時点では、このテキストマーカーのコンテキスト（`context`）に指定できるのはプログラム内の以下の処理だけです。

```
ReceiveTum
```

このテキストマーカーが指すオブジェクト（`object`）を指定する必要があり、こでは TUM の名前を指定します。さらに、TUM により実行されている関数（`function`）を指定する必要があります。

コードポイント、コンテキスト、関数の組み合わせは、以下の表に示す 17 とおりが可能です。

		context				
		Send Signal	**Receive Signal**	**Send Pdu**	**Receive Pdu**	**Receive Tum**
					仮想	
コードポイント	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	

表 3-1 コードポイント、コンテキスト、関数の組み合わせ

例 :

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

上記のコードロケーションインジケータは、NIF モジュールのグローバル変数の宣言の後に挿入されるソースコードを示しています。この例では、`counter` という変数が宣言されています。

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

上のコードロケーションインジケータは、指定された PDU が送信される前に実行されるソースコードを示しています。このユーザーコードは前に宣言された `counter` という変数をインクリメントします。

コードロケーションインジケータとプログラムの流れ

本項では、コードロケーションインジケータの正確な挿入位置をプログラムの流れの中で説明します。角括弧付きの名前 `<text>` は、挿入されるユーザーコードに対応するダミーを示しています。

グローバルコンテキスト :

```
<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 コンテキスト :

```
//[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 コンテキスト :

```
//[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 コンテキスト :

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

ReceivePDU コンテキスト :

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

ユーザー定義のパラメータ、測定変数、入力ポート、出力ポート

ユーザー定義コード内で計算された値にアクセスできるようにするため、NIF モジュールにパラメータやポートを追加することができます。これらには実験環境（EE）からアクセスできます。

パラメータ：

パラメータは以下のファイルに定義します。

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

パラメータは 1 行に 1 つずつ、以下の形式で定義します。

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

`<type>` には以下のいずれかを使用できます。

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

また、アンダースコア 2 文字をセパレータにして階層変数を宣言することもできます。たとえば、以下のように定義すると、

```
real64 setting__var = 42.0;
```

以下のパラメータが作成され、

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

指定した値に初期設定されます。

注記

EB tresos BMC のデータモデルの信号名または PDU 名が使用されている場合は、アンダースコア 2 文字が含まれていてもそれは階層レベルとして解釈されません。

このパラメータは、ソースコード内では変数 `<name>` によりアクセスされます。

測定変数：

測定変数は以下のファイルに定義します。

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

測定変数は 1 行に 1 つずつ、以下の形式で定義します。

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

入力ポート：

入力ポートは以下のファイルに定義します。

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

入力ポートは 1 行に 1 つずつ、以下の形式で定義します。

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

入力ポートは、ソースコード内では変数 `<name>` によりアクセスされます。入力ポートは、変数と同様に階層的に宣言することもできます。

出力ポート：

出力ポートは以下のファイルに定義します。

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

出力ポートは 1 行に 1 つずつ、以下の形式で定義します。

```
TPortObj <module_name>_UserCode_outport_<name>;
```

出力ポートは、ソースコード内では変数 <name> によりアクセスされます。出力ポートは、変数や測定変数と同様に階層的に宣言することもできます。

例：

以下の例では、ユーザーコードを使用して NIF モジュールに機能を追加する方法を紹介します。

以下の信号の値をバスから読み取ります。

```
sint32 water_temperature
```

測定変数を作成します。

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

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

コードフラグメントを作成します。

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

これにより、最後に受信した信号値が呼び出され、宣言されていた変数に代入されます。

受信した信号を直ちに使用できるように、コードポイントとして `afterReceive` が選択されています。この信号はバスから読み取られるので、適切なコンテキストは `ReceiveSignal` です。挿入されたコードで扱う信号の名前は `water_temperature` で、この名前は `Signals.txt` ファイルから読み取られます。

LABCAR-IP でコードが生成されると、LABCAR-EE 実験環境においてこの新しい測定変数を使用できるようになります。

注記

ユーザー定義コードが NIF モジュール内で使用される場合、そのコードはコード生成時に NIF モジュールのソースコードに統合されます。

3.8.7 ETAS バス通信モニタ（BCM: Bus Communication Monitor）へのリンク

ETAS バス通信モニタ（BCM: Bus Communication Monitor）を使用すると、たとえばマニュアル操作で指定された値で信号値を送信前に上書きするなど、NIF モジュール内の信号フローを操作することができます。

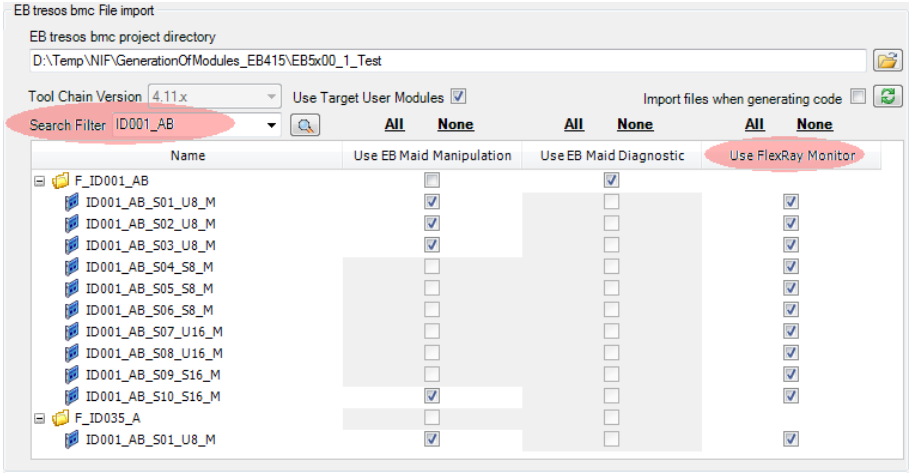
パフォーマンスを最適化するため、ETAS バス通信モニタの操作対象とするためのコードが生成される信号を、ユーザーインターフェース上で限定することができます。

ETAS バス通信モニタの設定は以下のコンフィギュレーションファイルに格納されています。

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

Bus Communication Monitor に信号を表示する：

"Bus Communication Monitor" ウィンドウで個々の信号にアクセスできるようにするには、ユーザーインターフェースで信号を選択する必要があります。



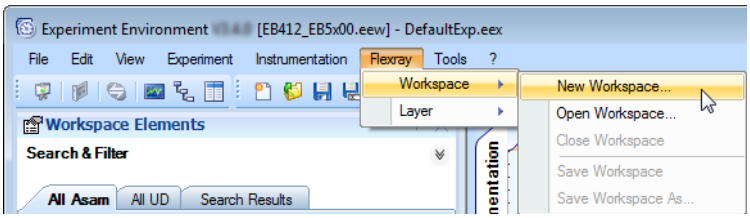
注記

Bus Communication Monitor がサポートしているのはスカラ信号と信号長が 32 ビット以内の信号だけです。ByteArray 信号や 32 ビットより長い信号はサポートしていません！

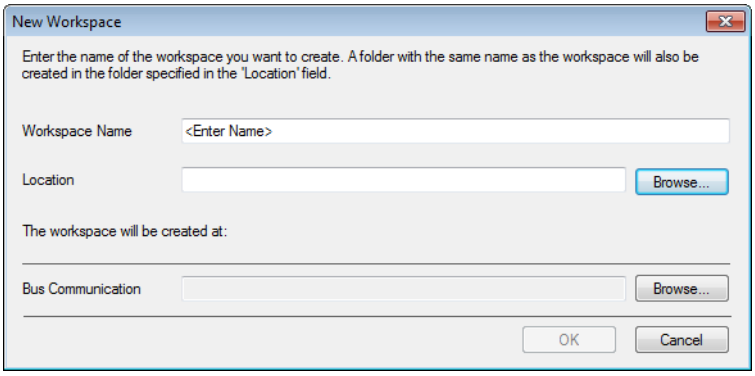
- 個々の信号を "Bus Communication Monitor" ウィンドウで操作できるようにするために、上記ユーザーインターフェースの **Use for FlexRay Monitor** オプションをオンにします。
  - 検索フィルタ（"Search Filter" フィールド）に文字列を正規表現で入力することにより、表示される信号を限定することができます。
  - **All** または **None** オプションを使用すると、検索条件に該当する信号をすべて有効化／無効化することができます。

    有効化／無効化は、選択された信号のすべてのインスタンスに影響します。
- モジュールを保存します。
- **Project** → **Build** を選択してコードを再生成します。
- **View** → **Flexray Elements** を選択して FlexRay エレメントを表示します。
- **View** → **Bus Communication Monitor** を選択して "Bus Communication Monitor" ウィンドウを開きます。

- **Flexray → Workspace → New Workspace** を選択して、FlexRay 専用ワークスペースを作成します。



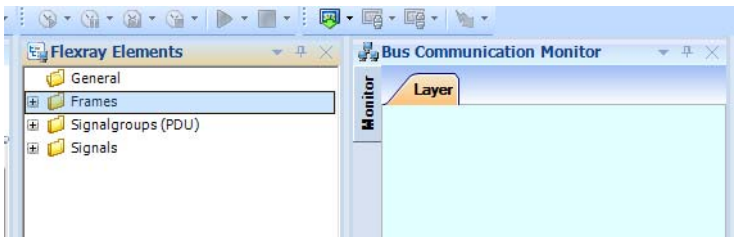
- 新しいワークスペースの名前と場所を指定します。
- "Bus Communication" ウィンドウから、上述のファイル `FlexRayMonitor.xml` を選択します。



または

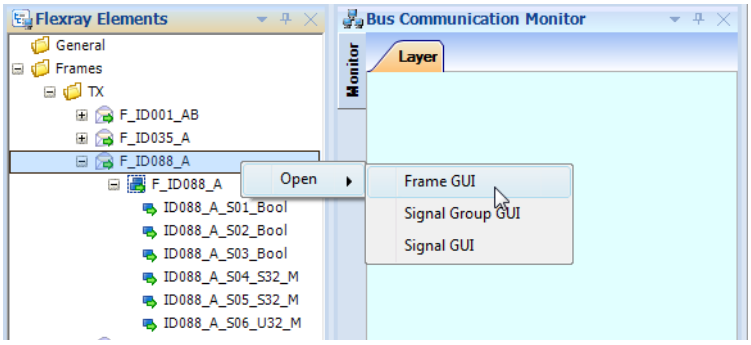
- **Flexray → Workspace → Open Workspace** を選択して、既存のワークスペース（`*.bcw`）を開きます。

  "Flexray Elements" ウィンドウには、有効化されているフレーム、信号グループ、信号が表示されます。



- "Flexray Elements" ウィンドウで、"Bus Communication Monitor" ウィンドウに表示したい信号が属するフレームを右クリックします。

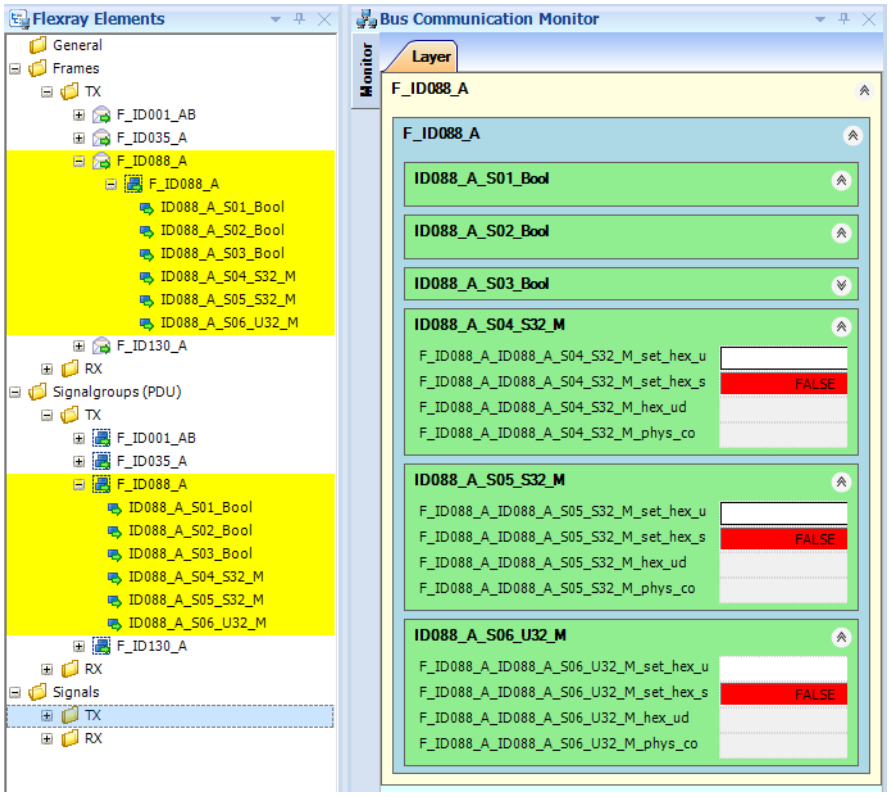
- ショートカットメニューから **Open** → **Frame GUI** を選択します。



そのフレームが "Bus Communication Monitor" ウィンドウのカレントレイヤに表示されます。

- 個々のセクションを表示します。

  ここでは、このフレーム内でリリースされている 3 つの信号のみ表示されます。



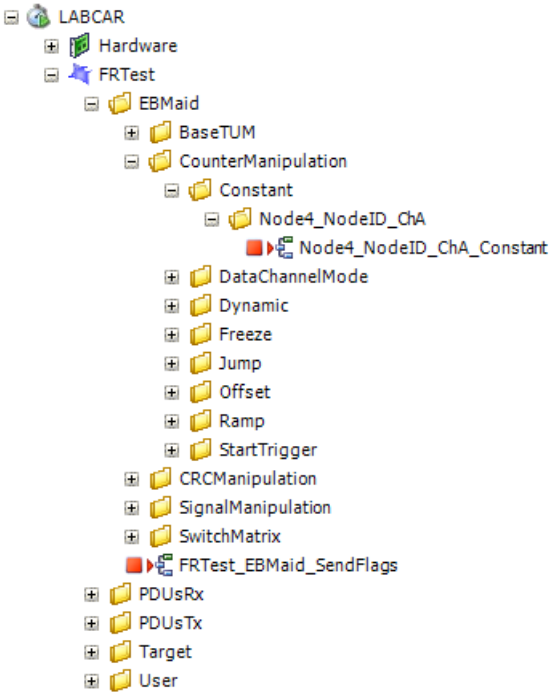
"Bus Communication Monitor" ウィンドウに表示されるように選択された信号は、"Workspace Elements" ウィンドウでは黄色でハイライト表示されます。

3.8.8 ターゲットユーザーモジュール（Target User Modules）

FlexRay 通信についての詳細情報（ライブカウンタや CRC 計算など）が必要となる場合、これらの機能はボードに直接実装する必要があります。たとえば CRC の計算に必要なデータは Real-Time PC 上では見ることができないので、FlexRay ボード上で処理を実行する必要があります。

ボード上で実行されるユーザー定義機能は、「ターゲットユーザーモジュール」（TUM: Target User Modules）として実装されます。TUM-API はフレーム送受信など特定のイベント用のコールバックを記録する機能を提供します。また、NIF モジュールとメッセージを交換する手段も提供するので、実験環境などとの通信が可能となります。ターゲットユーザーモジュールの詳細については、Elektrobit Busmirror TUM の資料を参照してください。

TUM サポート機能が有効になっていると、インポートされる TUM ディスクリプション内に定義された信号用のポートが作成され、TUM の制御に使用されます。EE の "Workspace Elements" ウィンドウ内の各 NIF モジュール下に以下の信号が作成されます。



このノード下の相対パスは、TUM 内に割り当てられた機能を反映しています。

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

第 1 レベルは、サポートされている以下のタイプの TUM の名前を示します。

```
SYSTEM
PROTOCOL
PROTOCOL-SAFEGUARDING
MANIPULATION
SIGNAL
```

第 2 レベルは TUM に割り当てられた機能（例：`CounterManipulation`）です。第 3 レベルは TUM が機能を提供する対象の信号の名前で、その下はその機能を制御するために必要なパラメータです。

信号と PDU についての TUM サポート機能の編集

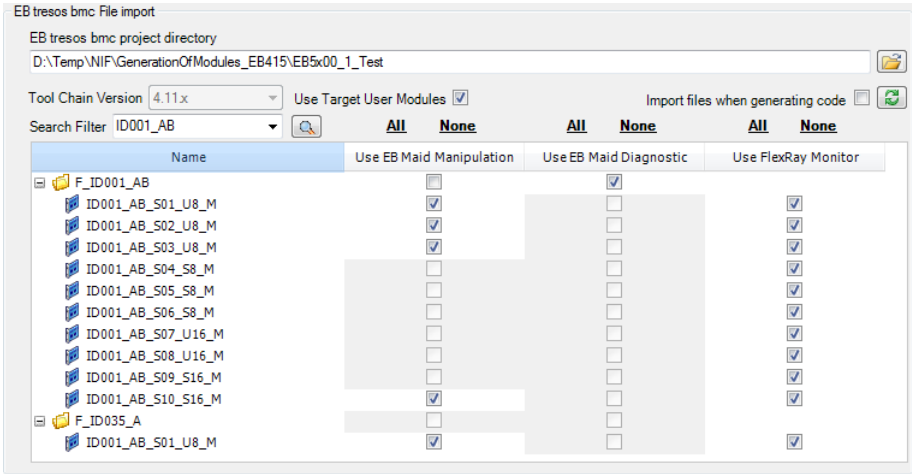
TUM サポート機能は、EB tresos BMC データモデル内の個々の信号と PDU ごとに有効にすることができます。これによりパフォーマンスが向上し、データが明確化されます。

関連する TUM が EB tresos BMC プロジェクトの一部である限り、EB tresos BMC データモデルに依存しないすべての TUM 信号（BaseTum や SwitchMatrix の信号など）は常にサポートされます。

- EB tresos BMC プロジェクトを編集して、アプリケーション用 TUM 機能をプロジェクトに追加します。
- 必要に応じて NIF モジュールを同期して、データモデルを更新します。
- ユーザーインターフェースで、**Use Target User Modules** オプションをオンにします。
- 各信号ごとの TUM サポート機能にアクセスできるようにするため、**Use EB Maid Manipulation**（Real-Time PC からボードに送信）または **Use EB Maid Diagnostic**（ボードから Real-Time PC に送信）オプションをオンにします。
  - 検索フィルタ（"Search Filter" フィールド）に文字列を正規表現で入力することにより、表示される信号を限定することができます。
  - **All** または **None** オプションを使用すると、検索条件に該当する信号をすべて有効化／無効化することができます。

    有効化／無効化は、選択された信号のすべてのインスタンスに影響します。

    4 バイト以上の信号は、出力ポートとして送信されません。これらの信号用にはバイト配列のコードが生成され、この配列をユーザーコードから処理することができます。



注記

TUM サポート機能は、EB tresos BMC プロジェクトの 1 つ以上の TUM 内に含まれる信号と PDU についてのみ使用できます。

- モジュールを保存します。

信号用 TUM 機能の選択

複数の機能（Dynamic Value、Hold、Ramp など）を持つ TUM 内の信号については、同時に 1 つの機能しか実行できません。この機能は、以下のノード下の入力ポート経由で選択できます。

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

この入力ポートの値が、実行すべき関数を指定します。有効な値と関数についての詳細は、TUM のドキュメントを参照してください。

値 0 は `OFF`（関数のデフォルトステート）を示し、その場合、TUM 内の関数は実行されません。

信号の TUM 機能の開始と終了

TUM は周期的には実行されません。関数呼び出し時に TUM の実行に必要なすべてのパラメータが渡され、別の関数呼び出しによって終了されるまで TUM が実行されます。

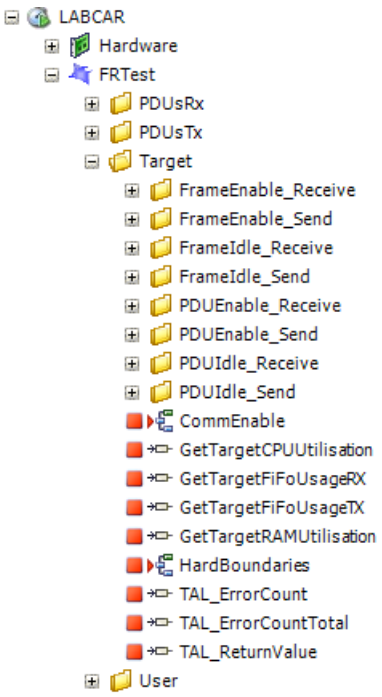
関数の実行開始には以下の入力ポートを使用します。

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

この開始トリガの値が 0 から 1 に変化することにより関数が実行されます。終了するには、`DataChannelMode` = 0 を設定し、さらに開始トリガを 0 から 1 に変化させます。

3.8.9 実験環境

実験環境の "Workspace Elements" ウィンドウには、使用されている EB tresos bmc のバージョンに応じて、NIF モジュールとの通信に使用される各種測定変数、パラメータ、入力ポートがツリー表示されます。



Target

"Target" フォルダには、通信を制御するための各種シグナルが含まれます。これらは、**Generate Target Communication flags as Parameters** オプションの設定に応じて、パラメータまたは入力ポートとして生成されます。

- *<module name>*/Target/CommEnable
  バス通信の許可／不許可を切り替えます。このパラメータの値を 0 にするとバスとの通信が行われなくなります。
- *<module name>*/Target/HardBoundaries
  信号の制限値を切り替えます。この値を 0 にすると、データモデルに記述された MinVal と MaxVal が信号の「ソフトバウンダリ」として使用され、1 にすると、データモデル内で定義されたビット長から計算された上下限値が「ハードバウンダリ」として使用されます。信号値は、各サイクルごとにバウンダリ内に制限されます。
- *<module name>*/Target/GetTargetCPUUtilisation
  ボードの CPU の現在の使用率（%）
- *<module name>*/Target/GetTargetRAMUtilisation
  ボードの RAM の現在の使用率（%）。実行時に動的にアロケートされるメモリのみがカウントされ、実行コード内でプログラムされた部分（静的メモリ部分）はカウントされないため、"GetTargetRAMUtilisation" の値は 0% になる場合があります。
- *<module name>*/Target/TAL_ErrorCount
  シミュレーション中の通信サイクル（1 サイクル）において発生した TAL エラーの数をカウントします。
- *<module_name>*/Target/TAL_ErrorCountTotal
  シミュレーション開始以降、TAL 呼び出しで発生したエラーの総数です。通信の再開によりリセットされます。
- *<module_name>*/Target/TAL_ReturnValue
  前回の通信サイクル内の TAL 呼び出しにおいて最初に発生したエラーのエラーコードです。TAL 関数のエラーコードについての詳細は、TAL のユーザードキュメントを参照してください。
- *<module_name>*/Target/GetTargetFiFoUsageRX
  Real-Time PC とボードの RAM メモリとの間の通信に使用される受信 FiFo の現在の使用率（%）
- *<module_name>*/Target/GetTargetFiFoUsageTX
  Real-Time PC とボードの RAM メモリとの間の通信に使用される送信 FiFo の現在の使用率（%）

Target/FrameEnable_Receive/Send

このフォルダには、NIF モジュールと FlexRay バスとの間で行われるフレーム単位の信号送信を許可するための各種シグナルが含まれます。

- *<module_name>*/Target/FrameEnable_Send/FrameEnable_Send *_<frame_name>*
  特定のフレームについて、バスへの送信を許可します。この機能は TAL 呼び出しによって実行されます。
  - 値 = 1: フレームは送信されます。
  - 値 = 0: フレームは送信されません。最後にバスに送信された値がそのまま保持されるか、または信号値が InvalidValue に置き換えられます（ペイロード無効時の **Signal InvalidValue** オプションがオンの場合）。
  - 値 = -1: ヌルフレームが送信されます。

- *<module_name>*/Target/FrameEnable_Receive/FrameEnable_Receive _*<frame_name>*
  バスからの特定のフレームの受信を許可します。この機能は TAL 呼び出しによって実行されます。
  - 値 = 1: フレームは受信されます。
  - 値 = 0: フレームは受信されません。ただしペイロード無効時の **Signal InvalidValue** オプションがオンの場合は、信号値が信号の InvalidValue に置き換えられます。

Target/PDUEnable_Receive/Send

このフォルダには、NIF モジュールと FlexRay バスとの間で行われる PDU 単位の信号送信を許可するための各種シグナルが含まれます。

- *<module_name>*/Target/PDUEnable_Send/PDUEnable_Send_*<PDU_name>*
  特定の PDU について、バスへの送信を許可します。この機能は TAL 呼び出しによって実行されます。
  - 値 = 1: PDU は送信されます。
  - 値 = 0: PDU は送信されません。最後にバスに送信された値がそのまま保持されるか、または信号値が InvalidValue に置き換えられます（ペイロード無効時の **Signal InvalidValue** オプションがオンの場合）。
- *<module_name>*/Target/PDUEnable_Receive/PDUEnable_Receive _*<PDU_name>*
  バスからの特定の PDU の受信を許可します。この機能は TAL 呼び出しによって実行されます。
  - 値 = 1: PDU は受信されます。
  - 値 = 0: PDU は受信されません。ただしペイロード無効時の **Signal InvalidValue** オプションがオンの場合は、信号値が信号の InvalidValue に置き換えられます。

Target/FrameIdle_Receive/Send

このフォルダには、LABCAR-OPERATOR と NIF モジュールとの間で行われるフレーム単位の信号送信を拒否するための各種シグナルが含まれます。

- *<module_name>*/Target/FrameIdle_Send/FrameIdle_Send_*<frame_name>*
  LABCAR-OPERATOR プロジェクトから送信されるフレームの信号値を拒否します。
  - 値 = 0: プロジェクトからのフレームの信号値を受理します。
  - 値 = 1: プロジェクトからのフレームの信号値を拒否します。最後に許可された値が保持されます。
- *<module_name>*/Target/FrameIdle_Receive/FrameIdle_Receive _*<frame_name>*
  LABCAR-OPERATOR プロジェクトへのフレームの送信を拒否します。
  - 値 = 0: プロジェクトに対してフレームの信号値を送信します。
  - 値 = 1: プロジェクトに対してフレームの信号値を送信しません。最後に送信された値が保持されます。

Target/PDUIdle_Receive/Send（EB tresos Busmirror 4.1.x 以降のみ）

このフォルダには、LABCAR-OPERATOR と NIF モジュールとの間で行われる PDU 単位の信号送信を拒否するための各種シグナルが含まれます。

- <*module_name*>/Target/PDUIdle_Send/PDUIdle_Send_<*PDU_name*>
  LABCAR-OPERATOR プロジェクトから送信される PDU の信号値を拒否します。
  - 値 = 0: プロジェクトからの PDU の信号を受理します。
  - 値 = 1: プロジェクトからの PDU の信号を拒否します。最後に許可された値が保持されます。
- <*module_name*>/Target/PDUIdle_Receive/PDUIdle_Receive_<*PDU_name*>
  LABCAR-OPERATOR プロジェクトへの PDU の送信を拒否します。
  - 値 = 0: プロジェクトに対して PDU の信号値を送信します。
  - 値 = 1: プロジェクトに対して PDU の信号値を送信しません。最後に送信された値が保持されます。

Target/PDUTiming_Receive

このフォルダには、各 PDU ごとに、最後に受信した信号のタイムスタンプを発行するためのシグナルが含まれます。タイムスタンプの単位はミリ秒です。

これらのシグナルは、**Generate Target communication Flags as Parameters** オプションの設定内容にかかわらず、必ず出力シグナルとして生成されます。

Send/ReceiveFrames

このノード下に、送信する信号値用の入力ポートと、受信する信号値用の出力ポートがあります。現行バージョンにおいては、各信号は、その信号が属する PDU フォルダ下に分類されて表示されます。

User

このフォルダには、ユーザー定義の測定変数、パラメータ、入力ポート、出力ポートが含まれます。ユーザーが定義したモジュールコードに応じた内容となります。

3.8.10 オートメーションサーバーを用いて FlexRay モジュールを作成する

NIF モジュールは、LABCAR-OPERATOR の API 関数を用いて作成することもできます。API には、ユーザーが GUI から実行できるすべての機能が含まれています。ユーザー定義コードの自動的な挿入や調整は行われないため、オートメーション操作においては、これらの処理は LABCAR-OPERATOR の外側で行う必要があります。

NIF モジュール構成の定義や問い合わせを行うための各種 Set/Get メソッドが用意されています。Set メッセージは、エラーが発生すると "False" を返し、正常終了すると "True" を返します。

API に関する詳細な情報（CHM 形式のヘルプファイル）を開くには、**? → Help** を選択して LABCAR-OPERATOR のドキュメントページを開き、"API Documentation (CHM)" をクリックします。

3.9 FiL モジュール

本項では FiL モジュールの作成について説明します。

Function-in-the-Loop (FiL)

従来の HiL システム（図 3-21）は、閉ループ制御回路内で相互接続された ECU とシミュレーションモデルとで構成されています。ECU の出力は、シミュレーション PC 用の信号を収集するボードに接続されています。シミュレーション PC 上ではシミュレーションモデルがリアルタイム実行されます。このモデルは測定された入力値に基づいて出力値を算出し、その出力値が電気信号に変換されて ECU 入力に接続されます。

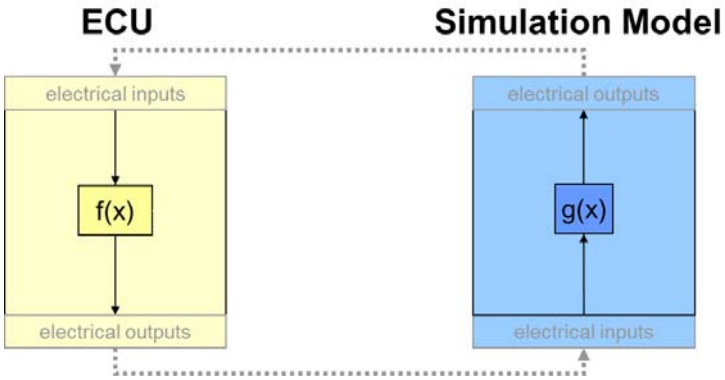


図 3-21 HiL システムの構成

それに対して FiL システム（図 3-22）では、HiL システムにおける電気的接続の一部（またはすべて）がバイパスされ、ECU メモリが直接アクセスされます。そのため、ECU には ETK または XETK を搭載しておく必要があります。[1]

つまり FiL システムでは、入力段と出力段、およびシミュレーションモデルとのハードウェア接続がバイパスされることになります。

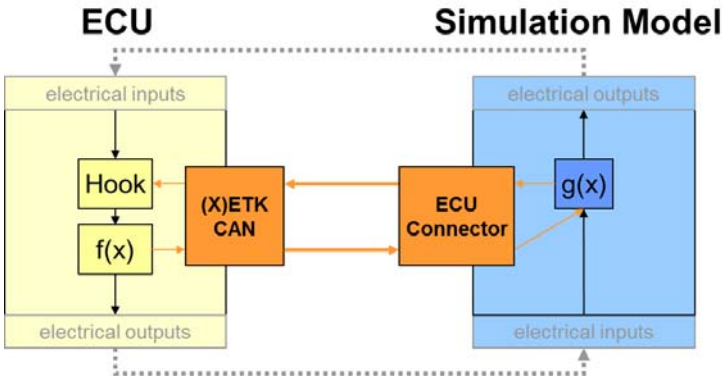


図 3-22 FiL システム

FiL システムを構築するには以下の条件が満たされている必要があります。

- ECU に ETK または XETK が搭載されていること
- ETK ／ XETK により読み取られる ECU 出力がシミュレーションモデルに接続されていること

1. XCPonCAN 経由のアクセスも可能ですが、FiL システムとしては速度が遅すぎます。

- モデルによりスティミュレートされる ECU ファンクションへの入力に、ソフトウェアフックを付ける必要があります。「フック」とは、ECU ファンクションの入力を電気的入力または前の ECU ファンクションから切り離し、ETK ／ XETK を使用してモデルに接続するためのスイッチです。
- ECU の内部ファンクションブロックについての十分な知識
- ファンクション入力に適切なスティミュレーションを行えるようにするために、ECU の診断機能を無効にしなければならない場合があります。

FiL システムの設定は LABCAR-IP で行い、FiL モジュールとの通信は実験環境 LABCAR-EE で行います。

3.9.1 LABCAR-IP での設定

下図に FiL ウィザードのスタートページを示します。

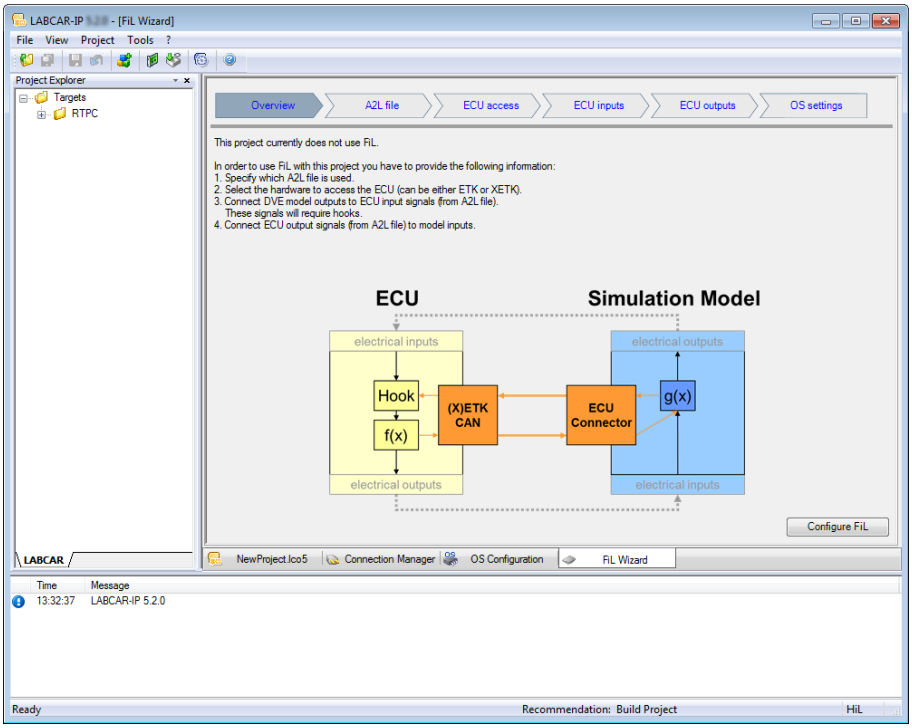


図 3-23 FiL ウィザードのスタートページ

ここでは、実行するステップの概要を一目で把握することができます。

1. ECU ファイルを定義します（185 ページ「A2L ファイルの選択」を参照してください）。
2. Real-Time PC から ECU までの接続を定義します（186 ページ「ECU アクセス用ハードウェアの定義」を参照してください）。
3. モデルによりスティミュレートされる ECU 値を定義します（188 ページの「スティミュレーション用 ECU 変数の選択」を参照してください）。
4. シミュレーションモデルの入力に接続される ECU 値を定義します（190 ページの「モデルに接続する ECU 出力の選択」を参照してください）。
5. ECU ラスタを OS コンフィギュレーション内のタスクに割り当てます（191 ページの「OS の設定」を参照してください）。

上記の各ステップ用のタブが、ウィザードウィンドウの上部に順に並んでいます。

ステータスとエラーの表示

各タブには対応するステップのステータス（OK、ワーニング、エラー）も表示されます。

設定プロセスにおいて発生したワーニングとエラーについての詳細メッセージは "Messages" ウィンドウに出力され、ログファイルに書き込まれます。

3.9.2 A2L ファイルの選択

まず最初に ECU ディスクリプションファイルを定義します。

ECU ディスクリプションファイル（A2L ファイル）を選択する：

- "A2L file" タブを選択します。
- **Browse** をクリックし、A2L ファイルを選択します。

  選択したファイルがロードされ、その内容（パラメータ、測定変数、使用可能なフック）が表示されます。

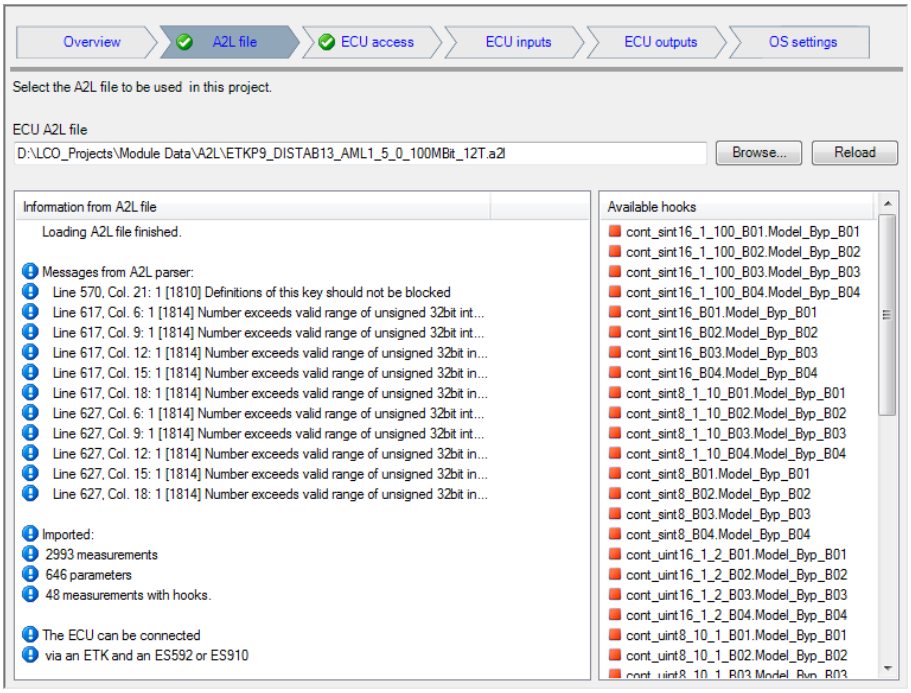


図 3-24 A2L ファイルを選択する

ファイルが読み取られると、以下の情報が表示されます。

- **Information from A2L file**

  各種パーサーメッセージがここに出力されます。

  ファイルをロードできなかった場合は、エラーの一覧が "Messages" ウィンドウに表示されます。
- **Available hooks**

  A2L 内にフックが定義されている信号がすべて表示されます。

A2L ファイルの名前やディレクトリは変更されずに内容のみが変更された場合は、**Reload** によりそのファイルを再読み込みすることができます。

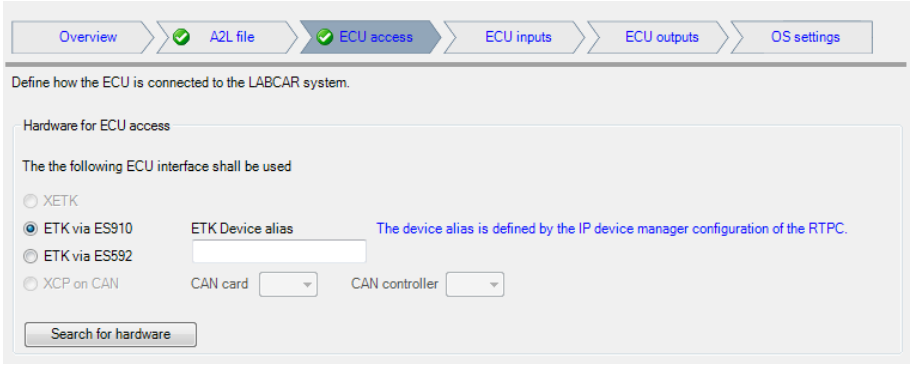
3.9.3 ECU アクセス用ハードウェアの定義

次のステップでは、ECU へのアクセスに使用するハードウェアを定義します。

- **XETK**
  XETK を搭載した ECU の場合、XETK をイーサネット経由で Real-Time PC に直接接続します。
- **ETK**
  ETK を搭載した ECU の場合、ETK を ES592 または ES910 に接続し、これらのモジュールをイーサネット経由で Real-Time PC に接続します。
- **XCP on CAN**

ハードウェアを選択する：

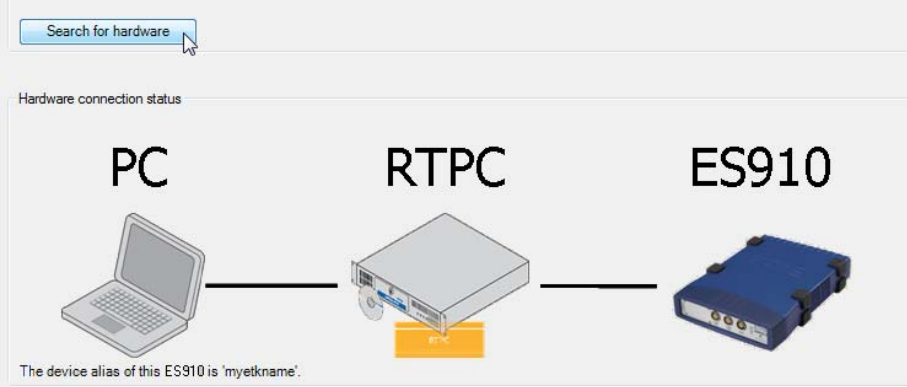
- "ECU access" タブを開きます。
- ECU アクセス用ハードウェアを選択します。



- ES910 または ES592 経由でアクセスする場合は、"ETK Device alias" に入力します（187 ページの「ETK デバイスエイリアス」を参照してください）。

- 選択したハードウェアが正しく接続されているかどうかを調べるには、**Search for hardware** をクリックします。

  選択したハードウェアが使用可能な場合は、そのハードウェアがアイコン（場合によってはデバイスエイリアス）により表示されます。



注記

検索されるハードウェアは、"Hardware for ECU access" フィールドで選択されたハードウェアだけです。それ以外のハードウェアが接続されていても、検索結果には表示されません。

ETK デバイスエイリアス

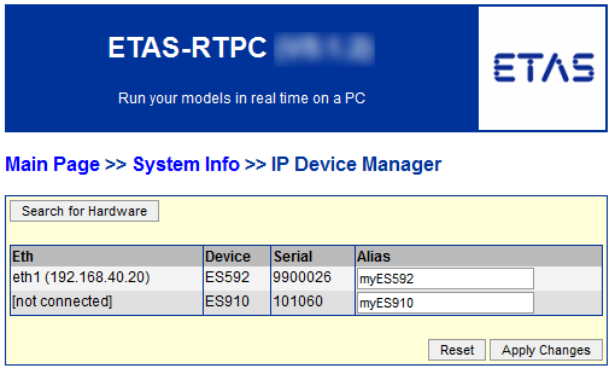
ETK デバイスエイリアス（"ETK Device Alias"）は、「ハード的な配線」を ES592 または ES910 の IP アドレスに置き換えるためのものです。これによりプロジェクトの移植が容易になります。

注記

ETK 経由でアクセスする場合は、前もって ETAS RTPC 上で "ETK Device Alias" を定義しておく必要があります。

ETAS RTPC にデバイスエイリアスを定義する：

- リンクをクリックすると、ブラウザインスタンスが開き、ETAS RTPC の web インターフェースの "IP Device Manager" ページが表示されます。



- **Search for Hardware** をクリックして、リアルタイム通信用に設定されているイーサネット接続（"rtudp_*n*"）を検索します。

  検索が成功すると、以下の情報が表示されます。

  — **Eth**

  Real-Time PC のイーサネットアダプタ。ハードウェアはここに接続されています。

  — **Device**

  デバイスタイプ

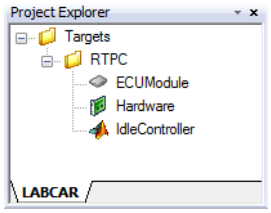
  — **Serial**

  デバイスのシリアル番号
- **Alias** に識別子（C 準拠、32 文字以内）を入力します。
- **Reset** をクリックすると、設定内容がリセットされます。
- 入力した内容を適用するには、**Apply Changes** をクリックします。
- webインターフェースを終了してLABCAR-IPに戻ります。

プロジェクトを保存する：

- **File** □**Save** を選択します。

  設定された FiL モジュールが保存され、Project Explorer に "ECUModule" として表示されます。



3.9.4 スティミュレーション用 ECU 変数の選択

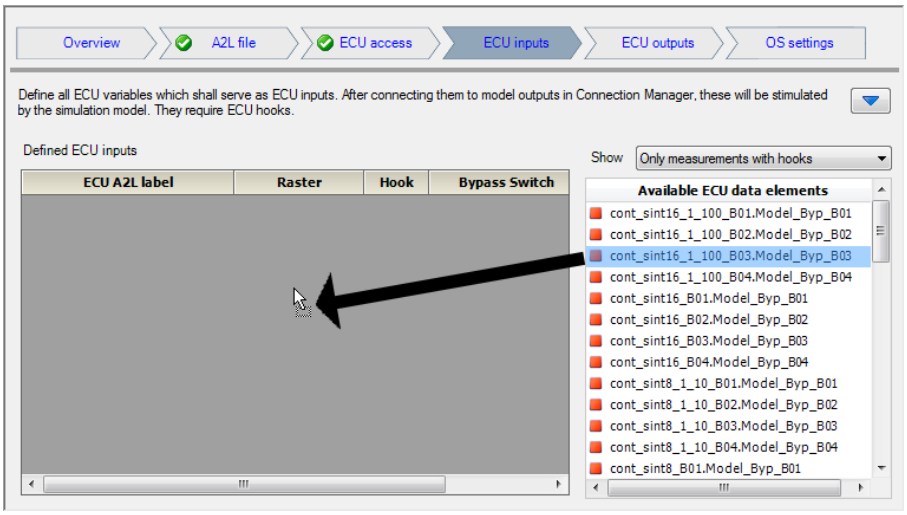
このステップでは、入力として使用する ECU 変数を定義します。これらの変数はシミュレーションモデルの出力によりスティミュレートされるので、そのための ECU フックが必要です。

これらのフックにより、当該ファンクションブロックは１つ前のブロックの出力（または ECU の入力）から切り離され、代わりに、ETK ／ XETK から提供される入力値がそのブロックに接続されます。つまり、フックは外部バイパスファンクションの有効／無効を切り替えるスイッチとして機能します。

入力を選択する：

- "ECU inputs" タブを選択します。
- "Show" フィールドで、フィルタとして **Only measurements with hooks** を選択します。
- "Available ECU data elements" リストから測定変数を選択します。

- その測定変数を左のリストにドラッグ＆ドロップで移動します。



測定変数が左のリストに追加されます。

測定変数をリストから削除する：

- リストから測定変数を削除するには、その測定変数を右クリックして **Delete** を選択します。

各列の意味と機能

- **ECU A2L label**
  データエレメントのラベル
- **Raster**
  このデータエレメントを収集する ECU ラスタを選択できます。
- **Hook**
  このデータエレメントにフックを使用できるかどうかが表示されます。
- **Bypass Switch**
  バイパスを有効化／無効化するパラメータです。これは "Available ECU data elements" リスト（フィルタ："Characteristics"）からドラッグ＆ドロップで選択できます。
- **Initial Value**
  "Bypass Switch" 列のパラメータの初期値。
  実験環境には、すべてのバイパスをデフォルトの初期値で有効化できる機能を持つ「通信操作用インストゥルメント」（195 ページ「Apply bypass switch settings」を参照）が用意されています。

ここで選択されたデータエレメントは、コネクションマネージャで保存した後に入力として使用できるようになります。使用可能にならない場合は、コネクションマネージャの **Update Ports** をクリックしてください）。

3.9.5 モデルに接続する ECU 出力の選択

このステップでは、後に測定用としてモデル入力に接続する ECU のデータエレメントを選択します。

下図に "ECU outputs" タブを示します。

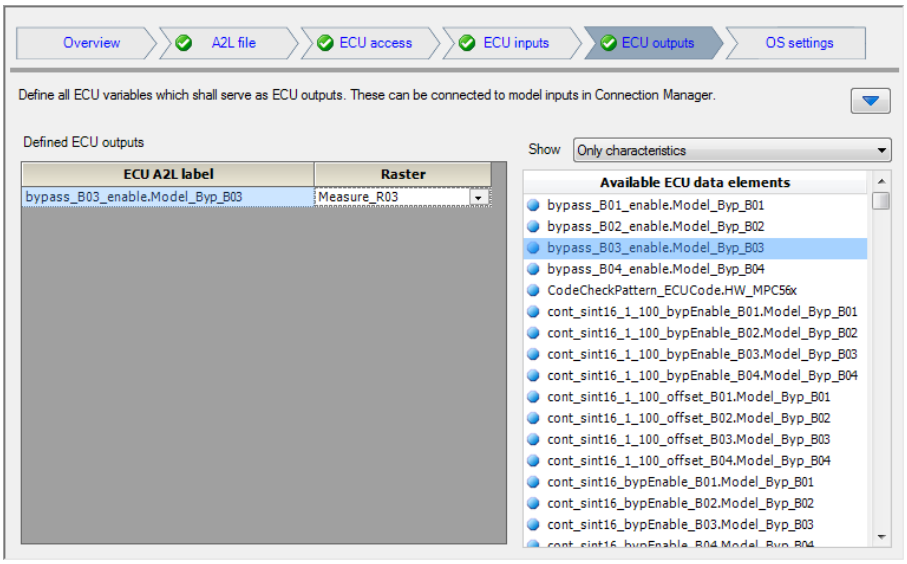


図 3-25 ECU 出力をモデル入力に接続できるようにする

データエレメントを選択する：

- 右側のリストからデータエレメントを選択し、左側のリストにドラッグします。

  選択したエレメントが "ECU A2L label" 列に追加されます。
- "Raster" 列でラスタを選択します。

ここで選択されたデータエレメントは、コネクションマネージャで保存した後に出力として使用できるようになります。使用可能にならない場合は、コネクションマネージャの **Update Ports** をクリックしてください)。

3.9.6 OS の設定

このステップでは、ECU ラスタを Real-Time PC のタスクにリンクします。これにより、シミュレーションと ECU ラスタを緊密にリンクさせることができます。

下図に "OS settings" タブを示します。

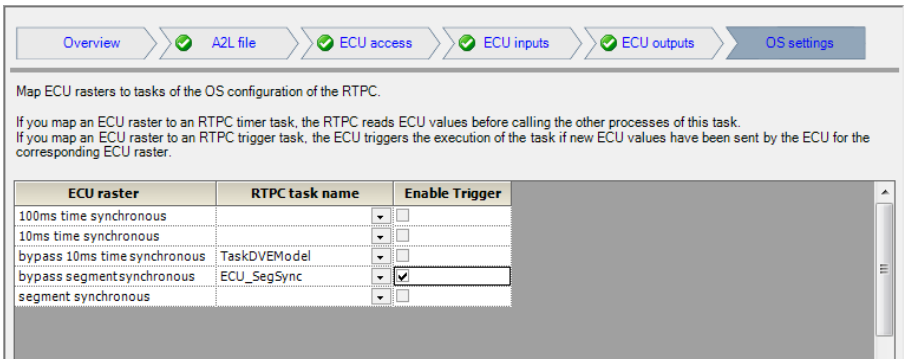
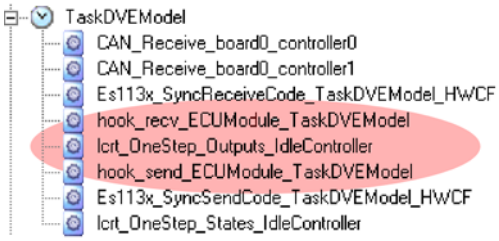


図 3-26 タスクを ECU ラスタに割り当てる

この設定は。タイマタスクとトリガタスクについて行います。

タイマタスク

Real-Time PC のモデル演算を行うタイマタスクに ECU ラスタをマッピングすると、モデル演算の前に演算ステップの dt データが ECU から読み取られてから（hook_recv_ECUModule_[task name]）、モデル演算が行われ（lcrt_OneStep_Outputs_...）、その後、演算結果が ECU に書き込まれます（hook_send_ECUModule_[task name]）。

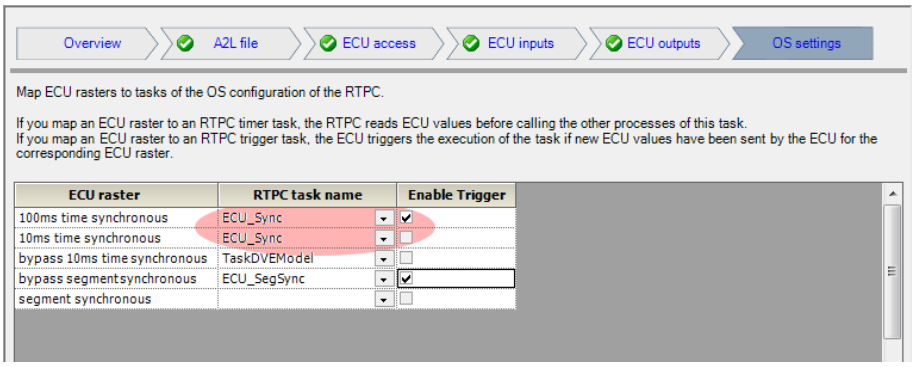


トリガタスク

トリガタスクに ECU ラスタをマッピングすると、モデル演算と ECU との緊密なリンクが実現します。つまり ECU をシミュレーション用タイマとして使用し、ECU からのデータ受信を待ってから Real-Time PC のモデル演算が開始されるようにすることができます。

ECU をクロックジェネレータとして使用するには、トリガタスクを割り当てて **Enable Trigger** オプションをオンにする必要があります（195 ページの図 3-27 を参照してください）。このオプションがオンになっていないと、ECU がデータを送信しても Real-Time PC のタスクはトリガされません。

そのため、**Enable Trigger** オプションはオンにしておくのが一般的ですが、同じトリガタスクに複数の ECU ラスタを割り当てる場合は、いずれか 1 つのラスタについてのみこのオプションをオンにします（下図参照）。

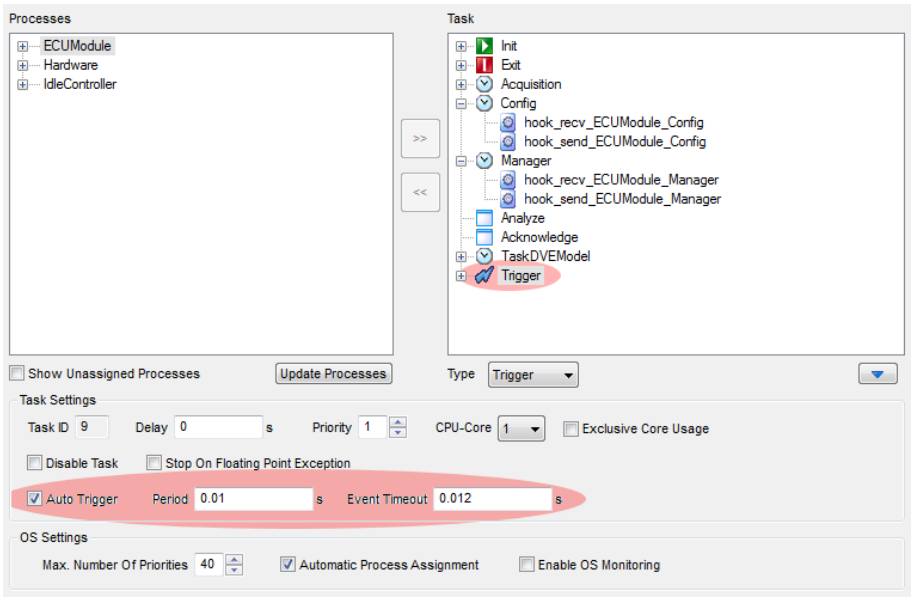


そのようにしないと、1 つの dT 周期内で ECU がそのトリガタスク（上図の例では ECU_Sync）を複数回トリガしてしまうため、応答時間が不規則になり、ECU において不定期なリードサイクルが発生してしまいます。

"Auto Trigger" オプション

ECU からデータが来ないとシミュレーションは静止状態になってしまいますが、そのような状況を避けるため、"Trigger" タイプのタスクには "Auto Trigger" というオプションがあります。

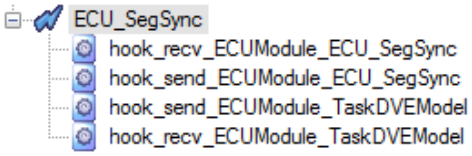
"OS Configuration" タブから Trigger タスクを選択すると、ウィンドウ下部の "Auto Trigger" オプションをオンにすることができます。



上図の例のように設定すると、12ms（"Event Timeout"）経過しても ECU からデータが届かない場合、再び ECU からデータが届き始めるまで 10ms（"Period"）間隔で演算が続けられます。

"RTPC Task Name" 列で Real-Time PC のタスクを割り当てると（191 ページの図 3-26 を参照）、各タスクに以下のプロセスが追加されます（LABCAR-IP メインウィンドウの "OS Configuration" タブ）。

- hook_recv_ECUModule_[*Taskname*]
- hook_send_ECUModule_[*Taskname*]



注記

設定の最後に、ECU 通信に使用されるすべてのタイマタスクとトリガタスク内の各プロセスの順序が正しいことを確認してください。

3.9.7 コネクションマネージャでの接続の作成

スティミュレーション用およびデータ収集用のデータエレメントを選択して、ECU ラスタをタスクにマッピングすると、FiL モジュールの設定は完了です。

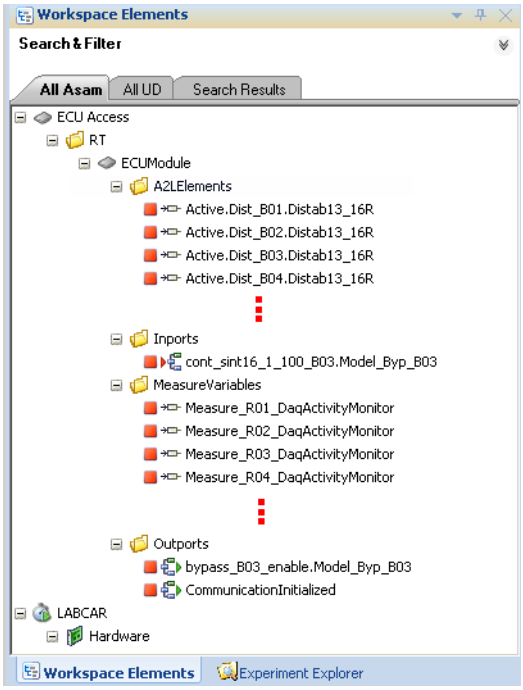
このモジュールを実際に使用できるようにするには、その入力と出力をコネクションマネージャで DVE モデルの入力と出力に接続する必要があります。

"CommunicationInitialized" 出力

コネクションマネージャには、本章で設定した出力以外に "CommunicationInitialized" という出力もあります。これは "Boolean" 値で、ECU が通信可能になるとこの値が 0 から 1 に変化します。

3.9.8 LABCAR-EE での操作

プロジェクトを再ビルドして実験環境で開くと、下図のように "Workspace Elements" ウィンドウに FiL モジュールが表示されます。



このモジュールには以下のサブフォルダがあります。

- **A2LElements**

  ECU ディスクリプションファイル（A2L ファイル）内のすべてのラベル（測定変数と適合変数）が含まれています。
- **Inports**

  設定時に定義された、モジュールの入力です。
- **MeasureVariables**

  各 ECU ラスタごとに 1 つずつ定義されている変数です。ラスタが ECU にトリガされるたびにインクリメントされます。
- **Outports**

  設定時に定義された、モジュールの出力です。このフォルダには "CommunicationInitialized" という出力もあります（193 ページの「"CommunicationInitialized" 出力」を参照してください）。

3.9.9 "RT ECU Access" インストゥルメント

測定変数とパラメータは、標準インストゥルメントを使用して表示／編集を行うことができます。FiL ファンクションを制御するための専用の "RT ECU Access" というインストゥルメントが用意されています。

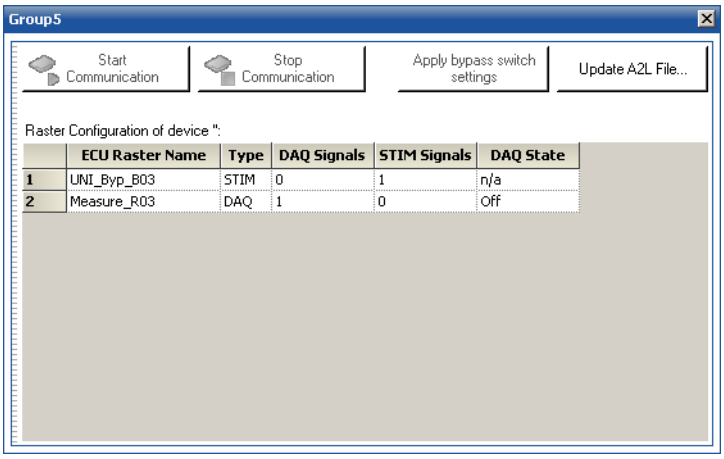


図 3-27 FiL ファンクション制御用インストゥルメント

このインストゥルメントの上部には以下の通信制御ボタンがあります。

- **Start Communication** / **Stop Communication**
  FiL モジュールと ECU との通信を開始／停止します。
- **Apply bypass switch settings**
  各バイパススイッチにデフォルト値をセットします（189 ページの「Initial Value」を参照してください）。
- **Update A2L File**
  ソフトウェアの更新に伴い A2L ファイルが新しくなった場合、この機能を使用することにより、LABCAR-IP でコンフィギュレーションを開かなくても A2L ファイルを更新することができます。

注記

この更新機能を実行するには、新しい A2L ファイルにモジュールのラスタと ECU ラベルがすべて定義されている必要があります。

インストゥルメントの下部には ECU ラスタのコンフィギュレーションが表示されます。各列の意味は以下のとおりです。

- **ECU Raster name**
  ラスタの名前
- **Type**
  タイプ："DAQ" または "STIM"
- **DAQ Signals** / **STIM Signals**
  当該タスクで測定／スティミュレートされる信号の数
- **DAQ State**
  ECU により DAQ ラスタ用の値が正しく送信されているかどうかを表します。"n/a"、"Off"、"Running" のいずれかの値が表示されます。

3.10 リアルタイムプラグイン

本項では、「リアルタイムプラグイン」を作成して LABCAR-OPERATOR プロジェクトに組み込み、実験環境 LABCAR-EE で使用する方法について説明します。

リアルタイムプラグイン

「リアルタイムプラグイン」は、動的にロードされる Real-Time PC 用プログラムライブラリです。実行時においてこのプラグインの関数がリアルタイムオペレーティングシステムに統合され、実行されます。

RT プラグインビルダ（"RT-Plugin Builder"）は、リアルタイムプラグインプロジェクトを作成し、それに関連するソースコードを管理するためのツールです。このツールでは、Real-Time PC 上で行われるリアルタイムプラグインのコンパイルとリンクもコントロールでき、リアルタイムプラグイン用のパケットファイルも生成できます。

ETAS RTPC

プラグインのソースコードは Real-Time PC 上でコンパイル／リンクされ、動的にリロード可能なプログラムライブラリ（共有オブジェクトライブラリ）になります。Real-Time PC はリアルタイムプラグインに必要な以下のような機能と環境を提供します。

- リアルタイムプラグインのロード／アンロードを行う
- フックプロセスを生成する
- フックプロセスにプラグインの関数をリンクさせる
- データエレメント識別子をメモリアドレス内で解決する
- リアルタイムコンテキストでフックプロセスを呼び出す

オペレーティングシステムの設定

LABCAR-IP の "OS Configuration" タブにおいて、リアルタイムオペレーティングシステムのタスクにフックを追加します。

実験環境（LABCAR-EE）

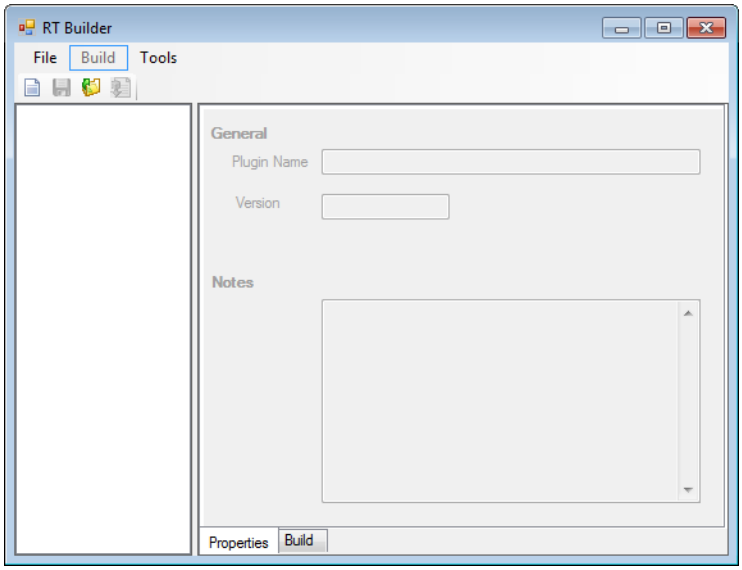
リアルタイムプラグインパケットファイルの操作（追加、有効化、無効化、削除）は実験環境で行えます。

3.10.1 RT プラグインビルダ（"RT-Plugin Builder"）

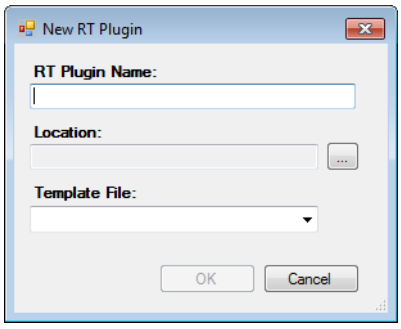
RT プラグインビルダは、リアルタイムプラグインを作成するためのアプリケーションで、プラグインに必要なコールバック関数用の関数ルートを含むディレクトリ構造とファイルを生成します。

リアルタイムプラグインプロジェクトを作成する：

- Windows のスタートメニューから **すべてのプログラム → ETAS → LABCAR-OPERATOR X.Y → RT-Plugin Builder** を選択します。
  RT プラグインビルダ（"RT Builder" ウィンドウ）が開きます。

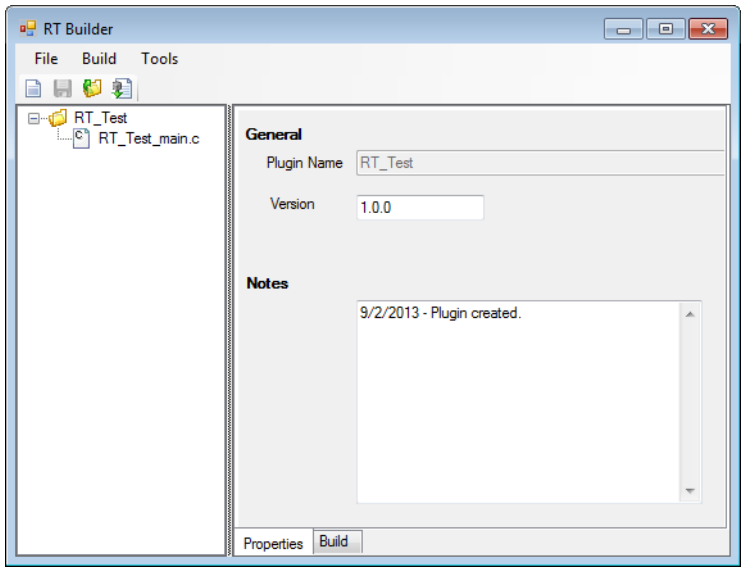


- **File → New** を選択します。
  "New RT Plugin" ウィンドウが開きます。



- RT プラグインの名前と、プロジェクトファイルの保存先ディレクトリを入力します。
- テンプレートファイルを選択します。

- **OK** をクリックします。
  プロジェクトが作成されます。

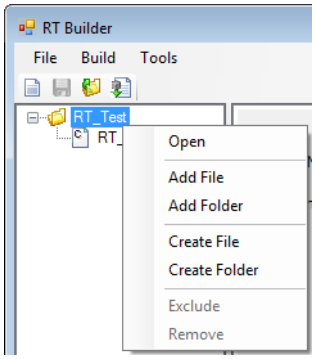


- 必要に応じて、バージョン番号（"Version"）を変更します。
- **File → Save** を選択してプロジェクトを保存します。

リアルタイムプラグインのソースコードを管理する：

1 つのプラグインに属するすべてのファイルは、RT プラグインビルダ内で 1 つの論理的なツリー構造として管理されます。

- プロジェクトフォルダを右クリックします。
  ショートカットメニューが開きます。

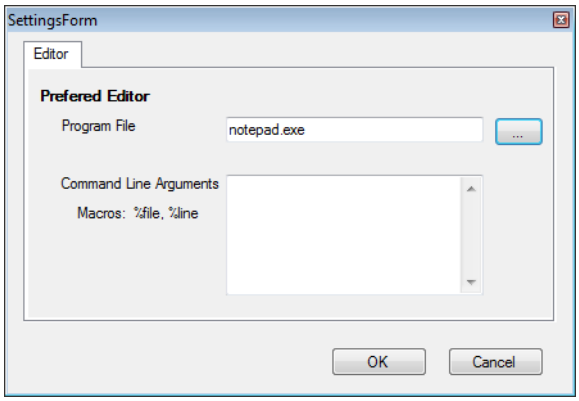


- **Create...** を選択すると、新しいファイルまたはフォルダを作成できます。
- プロジェクトフォルダ内で **Add...** を選択すると、既存のファイルまたはフォルダを追加できます。
- **Exclude** を選択すると、ファイルまたはサブフォルダをプロジェクトから除外できます。
  除外されたファイルやフォルダはプロジェクト内に保持されているので、**Add File / Add Folder** を選択して再びプロジェクトに追加することができます。

- ファイルまたはサブフォルダを削除するには **Remove** を選択します。
  指定されたファイルやフォルダが物理的に削除されます。

ファイルを編集する：

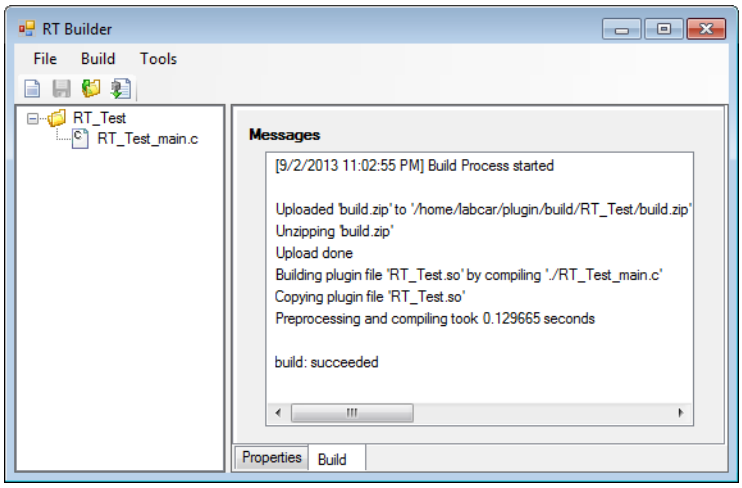
- ツリー内の、編集するファイルをダブルクリックします。
  プリセットされているエディタでそのファイルが開きます。
- **Tools → Settings** を選択すると、使用するエディタを変更することができます。



- "Command Line Arguments" フィールドには、エディタ起動時に使用する引数を入力することができます。

リアルタイムプラグインをコンパイルしてリンクする：

- "Build" タブに切り替えます。
- **Build □Build Plugin** を選択します。
  ファイルが Real-Time PC に転送され、そこで処理されます。



ビルド処理の結果が "Messages" フィールドに表示されます。

リアルタイムプラグインパッケージ

ビルドを実行すると、ソースコードが記述されたファイルから共有オブジェクトライブラリが生成されます。RT プラグインビルダはこれらのファイルを他のファイルと共に 1 つのアーカイブファイルに格納し、このアーカイブファイルは「リアルタイムプラグインパッケージ」と呼ばれます。

リアルタイムプラグインパッケージには以下のファイルが含まれます。

- ツールやバージョンなどの情報が記述されたメタデータファイル（xml）
- プラグインの共有オブジェクトライブラリ

3.10.2 コード例

以下のコード例は、各コマンド（"enable"、"disable"、"reset"）を使用してコントロールされる単純なカウンタを実装するものです。

このプラグインは LABCAR-OPERATOR プロジェクトの C コードモジュールとやりとりを行います。つまり、C コードモジュールから倍率を読み取り、この倍率でスケーリングしたカウンタ値をモジュールパラメータ "In" に書き込みます。すると C コードモジュールは出力 "Out" から "In" を出力します。

```
#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)
{
}
```

この例の内容についての詳細は、以降の説明を参照してください。

3.10.3 LABCAR-IP の OS コンフィギュレーションにフックを定義する

「フック」は、LABCAR プロジェクトの OS コンフィギュレーション内の 1 つ（または複数）のプロセスを表すワイルドカードとして機能します。つまり、プロセスと同様に、タスクの任意のポイントに追加したり移動や削除を行うことができます。またプロセスとは異なり、フックの名前はユーザーが自由に定義／変更することができます。

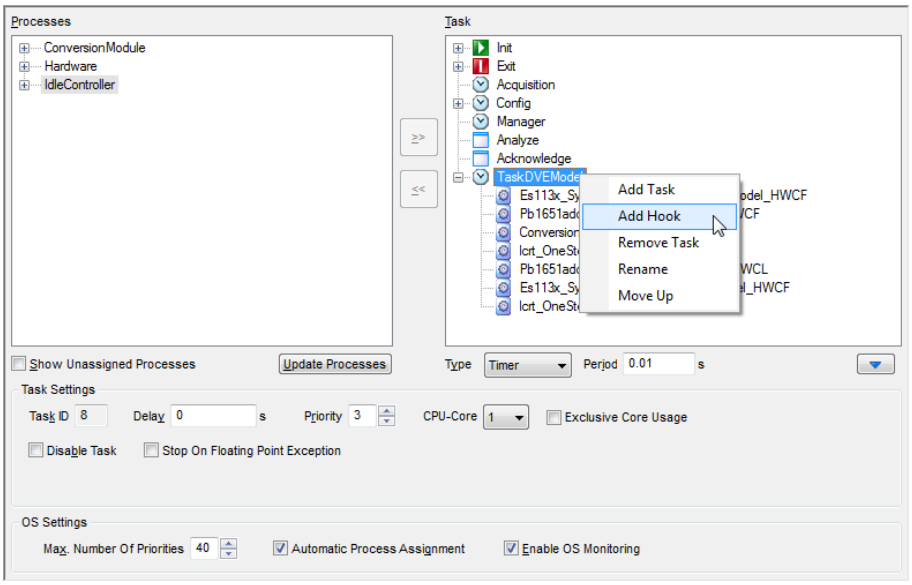


図 3-28 OS コンフィギュレーションにフックを追加する

1 つのフックを 2 回以上使用することもできます。つまり、1 つのフックを同じ名前で 1 つ（または複数）のタスク内の複数の場所に追加することができます。

3.10.4 プラグインプロセスをフックにアタッチする

プロセスをフックにアタッチする処理は、リアルタイムプラグインにより行われます。リアルタイムプラグインをロードすると、そのプラグインの

```
on_load
```

という名前のメソッドが呼び出されます。

その中の以下の関数を使用してプロセスを適切なフックにアタッチすることができます。

```
rtos_hook_attach
```

もしも必要なフックが使用可能な状態になっていない場合、実装された `on_load` メソッドは、そのリアルタイムプラグインのロードを中断しなければならないかどうか（つまり続行不可能であるかどうか）を判断します。

1 つのフックに複数のプロセスをアタッチすることができます。デフォルトでは、新しいプロセスは必ずフックのプロセスリストの末尾にアタッチされます。フックプロセスリスト内の末尾以外の任意の位置にプロセスを追加することはできません。

リアルタイムプラグインをアンロードする際は、

```
on_unload
```

メソッドにより、アタッチされていたすべてのプロセスをフックのプロセスリストから除外する必要があります。これには以下の関数を使用します。

```
rtos_hook_detach
```

3.10.5 ラベルとマッピング

LABCAR-OPERATOR プロジェクト内のすべてのデータエレメントには、ユニークなアドレッシングを実現するための「ラベル」が付けられています。

データエレメントのアドレッシング

LABCAR-OPERATOR プロジェクトのデータエレメントのアドレッシングは、リアルタイムプラグイン内でコンパイラ指示文を使用して行われます。

この指示文の構文は以下のとおりです。

```
#typedlabel  <type> <variable identifier> <data element
label>
```

データエレメントへのアクセスは、プラグインのソースコードから、指定された変数識別子を使用して行います。実行時においてラベルは、プラグインがロードされてからプラグイン関数 `on_load` が呼び出される前に、データエレメントのメモリアドレスに解決されます。

そのラベルが LABCAR-OPERATOR プロジェクト内で未知の場合、またはそのデータエレメントのデータ型が「予期しないもの」である場合は、リアルタイムプラグインはロードされず、所定のエラーエントリが ETAS RTPC のログファイルに書き込まれます。

ラベルのマッピング

データエレメントに対してユーザー定義されたラベルは、マッピングファイルと SuT マッピングファイルを使用して LABCAR-OPERATOR 内で実現化されます。ユーザー定義ラベルをリアルタイムプラグイン内で使用することにより、プロジェクトに依存しないプラグインを作成することができます。

実行時においてユーザー定義ラベルは、プラグインがロードされてからプラグイン関数 `on_load` が呼び出される前に、データエレメントのメモリアドレスに解決されます。

ラベルが LABCAR-OPERATOR プロジェクト内で未知の場合や、マッピング情報がないために解決できない場合、またはデータエレメントのデータ型が予期しないものである場合は、リアルタイムプラグインはロードされず、所定のエラーエントリが ETAS RTPC のログファイルに書き込まれます。

バックグラウンドサービス

リアルタイムコンテキストでは実行されない関数（ファイルアクセス用関数など）は、`rtos_hook_attach` という関数（対応するフック識別子は `rtos_hook_background`）を使用してオペレーティングシステムのバックグラウンドタスクに割り当てることができます。

3.10.6 リアルタイムプラグインのライフサイクル

図 3-29 のアクティビティ図は、リアルタイムプラグインをロード／アンロードする際の処理の流れを示しています。

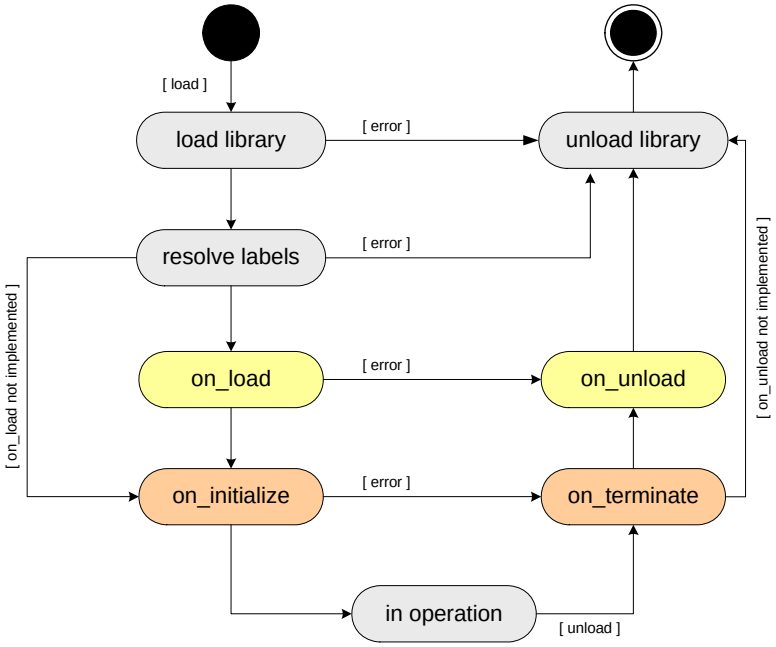


図 3-29 アクティビティ図

- **load library**

  共有オブジェクトライブラリをロードし、プラグインのコールバック関数用の関数ポインタを解決します。

  — `on_load`（任意）

  — `on_initialize`（必須）

  — `on_terminate`（必須）

  — `on_unload`（任意）

  エラーが発生するケース : 必要なコールバック関数の関数ポインタを解決できない場合

- **resolve labels**

  データエレメントのラベルをメモリアドレスに解決します。

  エラーが発生するケース : データエレメントのラベルを解決できない場合、またはデータエレメントの型が適合していない場合

- **on_load**

  コールバック関数 `on_load` が実装されている場合は、それを呼び出します。この関数は依存関係を処理し、システムリソースを予約します。

  エラーが発生するケース : コールバック関数の戻り値が 0 未満の場合

- **on_initialize**

  コールバック関数 `on_initialize` を呼び出します。この関数の中では、リアルタイム関数がそれぞれ特定のフックに割り当てられ、バックグラウンド関数とコマンドハンドラが登録されます。

  エラーが発生するケース : コールバック関数の戻り値が 0 未満の場合

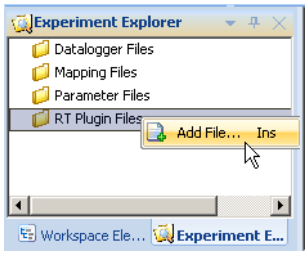
- **in operation**
  登録されているリアルタイム関数と非リアルタイム関数を Real-Time PC のオペレーティングシステムから呼び出します。
- **on_terminate**
  コールバック関数 `on_terminate` を呼び出します。この関数の中では、前に `on_initialize` 内で登録された関数が終了します。
- **on_unload**
  コールバック関数 `on_unload` が実装されている場合は、それを呼び出します。前に `on_load` 内で予約されたシステムリソースは、この関数内で解放されます。
- **unload library**
  リアルタイムプラグインの共有オブジェクトライブラリが解放され、アンロードされます。

3.10.7 実験環境でリアルタイムプラグインを使用する

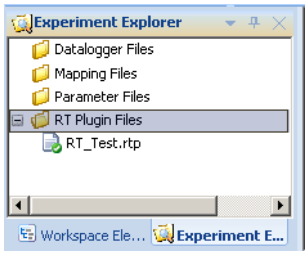
リアルタイムプラグインは、実験環境において、パラメータファイルやマッピングファイルと同じように管理されます。つまり、リアルタイムプラグインの追加、除外、有効化、無効化は、LABCAR-EE のエクスペリメントエクスプローラ（"Experiment Explorer"）で行われます。

RT プラグインを実験に追加する

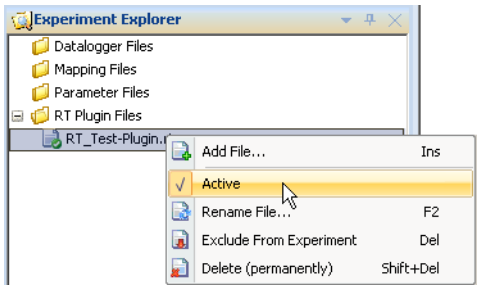
- エクスペリメントエクスプローラの "RT Plugin Files" フォルダを右クリックします。
- **Add File** を選択します。



- ファイル選択ウィンドウからプラグインファイル（*.rtp）を選択します。
  リアルタイムプラグインが実験に追加されます。



- このプラグインがアクティブになっていない場合は、そのファイルを右クリックして **Active** を選択します。



- "RT Plugin Files" フォルダからファイルを除外するには、**Exclude From Experiment** を選択します。
- ファイルを削除するには、**Delete (permanently)** を選択します。

リアルタイムプラグインを使用する実験を実行する：

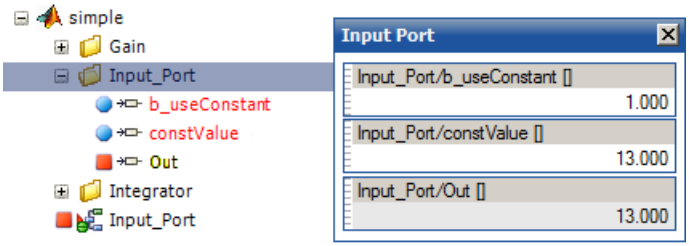
- **Experiment → Download → LABCAR** を選択して、実験をシミュレーションターゲットにダウンロードします。

  プラグインの状態が "Initialized" になります。
- **Experiment → Start Simulation → LABCAR** を選択して、シミュレーションを開始します。

  実験が実行されます。
- "RT Plugins" タブに切り替えます。

LABCAR のポートを Simulink モデル内で使用する際の注意事項

Simulink モデル内の LABCAR 入力ポートには、リアルタイムプラグインから直接書き込みを行わないでください。代わりに、パラメータ "B_useConstant" を "1.0" にセットして、リアルタイムプラグインからパラメータ "constValue" に書き込みます。



OS コンフィギュレーション内でフックが追加されている場合、上記のようにすることにより、測定変数 "Out"（"simple/Input_Port/Out"）、および関連するすべての Simulink ブロックが正しい影響を受けます。

同様に、LABCAR 出力ポートにもリアルタイムプラグインからは書き込まないようにしてください。ただし C コードモジュールの入力ポートへの書き込みは可能です。

3.10.8 "RT Plugins" タブ

下図に "RT Plugins" タブを示します。

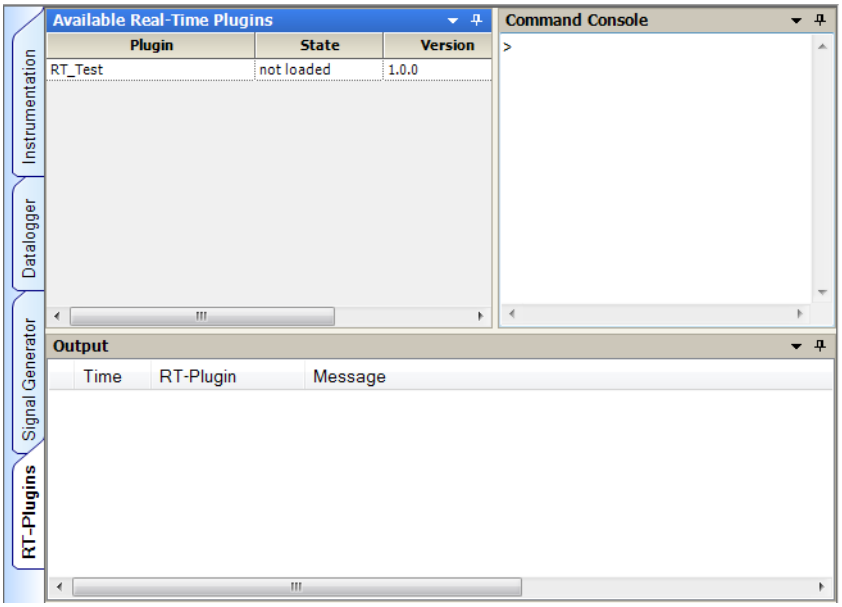


図 3-30 LABCAR-EE の "RT Plugins" タブ

このタブは以下の 3 つのサブウィンドウに分かれています。

- "Available real-time plugins"
- "Output"（207 ページ「"Output" サブウィンドウ」を参照）
- "Command console"（208 ページ「"Command Console" サブウィンドウ」を参照）

"Available Real-Time Plugins" サブウィンドウ

実験に含まれるすべてのリアルタイムプラグインが一覧表示されます。

- **Plugin**
  プラグインの名前
- **State**
  "State" 列には各プラグインの現在の状態が表示されます。状態の問い合わせは約 500ms のラスタで周期的に行われます。
  プラグインのロード状態を示すステート（"loaded" と "not loaded"）が事前定義されています。ステートのユーザー定義は、プラグインのソースコード内で行います。
- **Version**
  プラグインのバージョン（RT プラグインビルダでプラグインを作成する際に割り当てられらもの）

"Output" サブウィンドウ

"Output" サブウィンドウ（"RT Plugins" タブの下半分）には、リアルタイムプラグインが生成するログメッセージが表示されます。

"Command Console" サブウィンドウ

このコマンドコンソールには、各リアルタイムプラグイン用のコマンドプロトコル（ASCII 文字列をベースとするもの）を実装することができます。そのためには、転送された文字列を解釈するための「プロトコルハンドラ」をプラグイン内に用意しておく必要があります。

プロトコルハンドラの登録は、リアルタイムプラグインのロード時に、以下を使用して行います。

```
rtos_plugin_command
```

使用できるコマンドは以下のとおりです。

at または @

指定されたコマンドを指定された RT プラグインに送ります。
例：`at ExamplePlugin disable`

clear

コマンドコンソールの内容を消去します。

info

ターゲット上にロードされているすべてのプラグインを一覧表示します。

help

ヘルプを表示します。

load

指定されたプラグインをロードします（`on_load`）。
例：`load ExamplePlugin`

unload

指定されたプラグインをアンロードします（`on_unload`）
例：`unload ExamplePlugin`

init

プラグインを初期化します（`on_initialize`）。
例：`init ExamplePlugin`

terminate

プラグインを終了します（`on_terminate`）。
例：`terminate ExamplePlugin`

<Tab>

オートコンプリート（大文字と小文字は区別されます）

<Arrow up>

コマンド履歴

<Arrow down>

コマンド履歴

3.11 信号変換モジュール

LABCAR-OPERATOR V5.4.0 に含まれている標準的な信号変換モジュールを用いることにより、汎用的な開ループコンフィギュレーションを実現することができます。

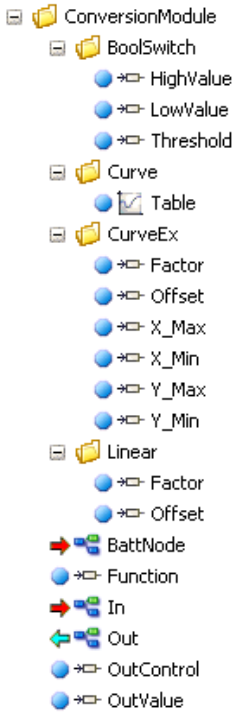
このモジュールは、コネクションマネージャで接続できるすべてのポート間に挿入でき、これによって I/O ハードウェアと DVE モデル間のセンサ／アクチュエータモデルを有効にすることができます。

3.11.1 モジュールの挿入

ポート間に信号変換用モジュールを挿入する方法は、224 ページの「信号変換」を参照してください。

3.11.2 パラメータ

信号変換用モジュールが挿入されると、そのモジュールのすべての入力、出力、パラメータと共に LABCAR-EE の "Workspace Elements" ウィンドウに追加されます。

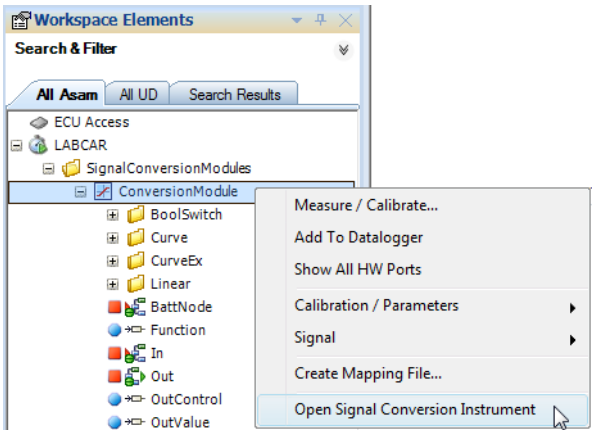


パラメータファイルによるパラメータ設定のほか、LABCAR-EE のインストゥルメンテーションの GUI を使用してモジュールのパラメータを適合することができます。

パラメータの意味（これがモジュールの機能を表します）を GUI を用いて以下に説明します。

3.11.3 信号変換制御用 GUI

このタイプのウィンドウを作成するには、"Workspace Elements" ウィンドウに表示されているツリービュー内のモジュールを右クリックし、**Open Signal Conversion GUI** を選択します。



"Instrumentation" タブのアクティブレイヤ内にインストゥルメント（GUI）を作成し、"Type" フィールドで以下のいずれかの以下のいずれかの変換タイプを選択します。

- Boolean
- Curve
- CurveEX
- Identity
- Linear

出力信号の制御

どの変換タイプを選択しても出力の制御を行えます。出力の制御には "Out Control" フィールドを使用します。

出力値として以下の値を選択できます。

- On
  常に出力値が演算されます。
- Off
  出力値を求める演算は行われません。指定された値がそのまま出力されます。

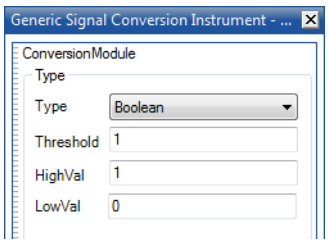
- Node
  出力値を求める演算は、バッテリノードの状態（"Workspace Elements" ウィンドウおよびコネクションマネージャに表示される "BattNode" 入力）に応じて行われます。選択されたバッテリノードのスイッチがオフになると、カーブ演算は行われなくなります。

注記

シミュレーション実行時に信号の値が正しく演算されない場合、信号演算がバッテリノードの状態に依存していないか、または常にオフになっていないかを確認してください。

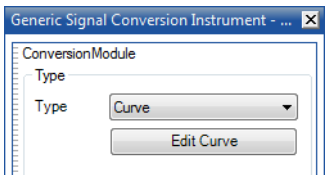
"Boolean"

入力信号のしきい値 "Threshold" を定義し、入力値がそれより低いレベルである場合に出力される値 "LowVal" と高いレベルを表す出力値 "HighVal" の値を定義します。

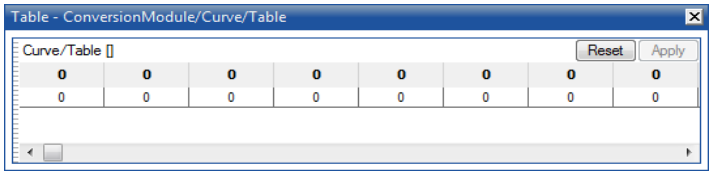


"Curve"

Edit Curve ボタンをクリックして、カーブ用のテーブルを編集します。

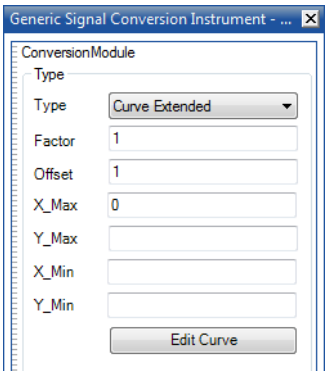


パラメータ "ConversionModule/Curve/Table" を編集するための GUI が開きます。



"Curve Extended"

拡張された "Curve" です。各種パラメータにより信号をさらに変換することができます。



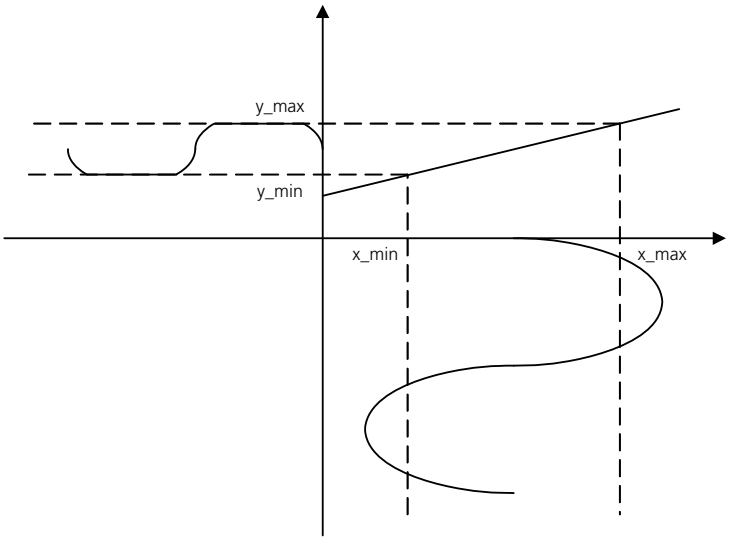
信号は、以下のようなさまざまなレベルで変換できます。

1. ファクタ（または減衰）a（"Factor"）とオフセット b（"Offset"）を使用して 1 次関数 v = a * u + b による変換を行います。デフォルトは Offset = 0、Factor = 1 で、入力と出力は等しくなります。

注記

このマッピングは、次のステップで定義される This mapping acts upon the **input values** of the characteristic defined in the next step!

2. **Edit Curve** で開くテーブル（211 ページの「"Curve"」を参照）で定義された適合値による変換を行います。
3. 信号値を最小値 "y_min" と最大値 "y_max" の範囲内に制限し、さらに "x_min"、"x_max"、"y_min"、"y_max" を指定して一次変換を行います。以下の図は、"x_min"、"x_max"、"y_min"、"y_max" で定義されたリミッタ機能の効果を示したものです。



x_min と x_max の値を指定すると、入力信号はその範囲の値に制限されますが、x_min、x_max、y_min、y_max をすべて指定すると、信号のファクタとオフセットを使用する 1 次方程式による変換が行われます。

信号の範囲が制限されないようにするには、最小値と最大値を十分な範囲の値に設定しておいてください（デフォルト値：-10000 と 10000）。

ファクタとオフセットが不要な場合は、y_min = x_min、y_max = x_max というように設定してください。

上記以外の設定になっていると、ファクタ（1 次方程式 (1) の傾き）、オフセット（1 次方程式 (1) のパラメータ b）、および x_max、x_min、x_max、y_min の値には、以下の相関関係が適用されます。

$$y = a*x + b \qquad (1)$$

$$y = ((y\_max - y\_min)/(x\_max - x\_min))*x - ((y\_max - y\_min)/(x\_max - x\_min))*x_1 + y_1$$

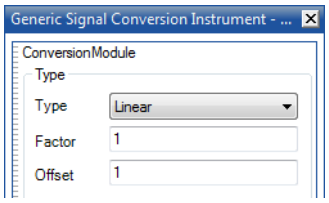
ここで、x_1 と y_1 には、（x_max, y_max）または（x_min,y_min）のいずれかのペアの値が使用されます。

"Identity"

センサ信号は 1:1 変換により物理値から電気値に、または電気値から物理値に変換されます。

"Linear"

ファクタ "Factor" とオフセット "Offset" を使用する 1 次方程式により、入力値から出力値への変換が行われます。



3.12 RTIO エディタによるハードウェア設定

RTIO エディタを使用してハードウェアの構成とパラメータを設定する方法は、『LABCAR-RTC V5.4.0 - ユーザーズガイド』に記述されています。

また『LABCAR-OPERATOR V5.4.0 - 入門ガイド』のチュートリアルの章には、ハードウェア設定を行う演習が含まれています。

3.13 コネクションマネージャ

コネクションマネージャは、モジュール間の入力と出力とを接続することにより閉ループ操作を可能にするものです。

コネクションマネージャには以下のような機能が含まれます。

- 接続できるすべての入力と出力を表示
- 個々の入力と出力の接続と切断
- 信号トレースのための仮想コネクションの作成
- 存在するコネクションのハイライト表示
- エクスポート／インポートにより、他のプロジェクトとコネクションリストを交換
- コネクションリストの表示／ソートオプション
- フィルタリングにより特定のタイプのコネクションのみを表示

以下に。モデル入力および出力の割り当て、コネクションの作成、信号変換モジュールの挿入の方法について説明します。

3.13.1 実際のコネクションと仮想コネクション

信号がコネクションマネージャに表示されるには、コネクションマネージャが各信号を認識する必要があります。これは、LABCAR-IP で統合されるモジュールの場合、モジュールの入力と出力はモジュールの一部であるため、特に配慮する必要はありません。ここでの「モジュール」とは、記述済みの ASCET モデルや Simulink モデル、CAN ／ FlexRay モジュール、ユーザー定義の C コードモジュール、信号パス調整用のモジュール（信号変換や閉ループのオープンを行う）、LABCAR-RTC におけるハードウェアコンフィギュレーションなどを指します。

これらのモジュールの信号は、コネクションマネージャ内で扱われ、コード生成後は LABCAR-EE 内の実験で扱われます。以下の図は HiL 実験の構成を示し、一般的なコンポーネント（ハードウェアと信号変換）と、アクチュエータとセンサのシミュレーションを行うモデルが含まれています。

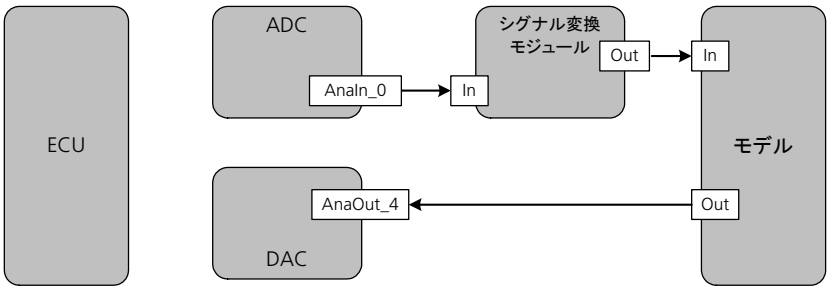


図 3-31 実験用信号

ECU からのアクチュエータ信号は、ここでは最初に ADC モジュールの出力信号として得られ、信号変換モジュールを介してモデルの入力に供給されています。センサの場合は、モデルの出力から DAC モジュールの入力までが信号パスとなります。

この時点では以下の情報が不明です。

- ECU と RTIO ハードウェアとの間の接続
- 各モジュール内の接続

つまりこれでは閉ループ実験における信号パスが完全には閉じていないことになりますが、実験環境においてはそれをトレースすることが必要です。

上記の 1 番目の情報は ECU ピンと HW ピンのリストによって補足され、2 番目の情報は「仮想コネクション」によって補足されます。

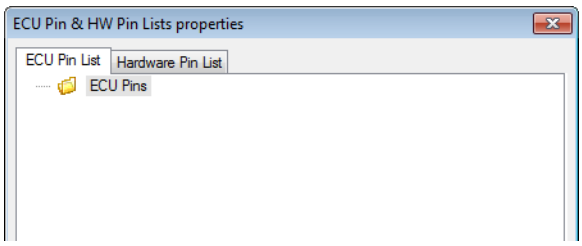
ECU ピンと HW ピン

ECU ピンリストやハードウェアピンリストは、以下のようにして作成することができます。これは LABCAR-IP のスクリプティング API を使用して自動的に行うこともできます。この API についての情報はオンラインヘルプ（**? → Help**）を参照してください。

ECU ピンのリストを作成する：

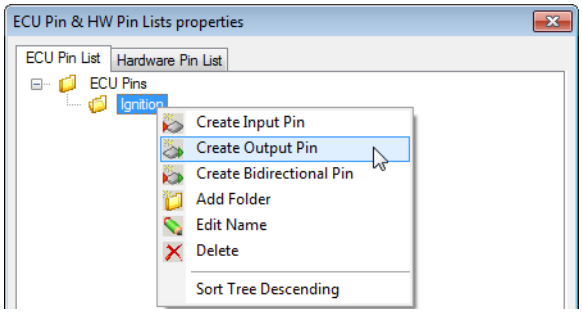
- LABCAR-IP のメインメニューから **Project → ECU Pin List** を選択します。

  "ECU Pin & HW Pin Lists properties" ダイアログボックスが開きます。



- "ECU Pin List" タブを選択します。
- "ECU Pins" という最上位エントリを右クリックします。
- **Add Folder** を選択します。

  新しいフォルダが作成され、名前はここで自由に変更できます。
- このフォルダを右クリックして各コマンド（**Create Output Pin** など）を選択します。



HW ピンのリストを作成する：

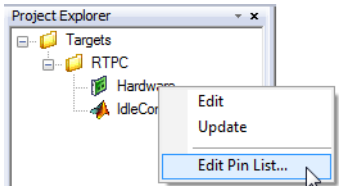
- LABCAR-IP のメインメニューから **Project → ECU Pin List** を選択し、"ECU Pin & HW Pin Lists properties" ダイアログボックスで "Hareware Pin List" タブを選択します。

または

- 以下のように操作しても同じ機能にアクセスできます。

  – プロジェクトエクスプローラ内のハードウェアモジュールを右クリックします。

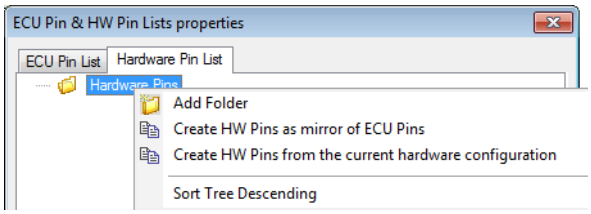
— **Edit Pin List** を選択します。



"ECU Pin & HW Pin Lists properties" ダイアログボックスが開きます。

— "Hareware Pin List" タブを選択します。

- "Hardware" という最上位エントリを右クリックすると、ショートカットメニューから以下のような機能を実行できます。



- ピンリストの編集を以下のように行います。
  - — ECU ピンリストの場合と同様、フォルダを作成できます。
  - — 既存の ECU ピンリストと対になるハードウェアピンリストを作成することができます。
  - — 現在のハードウェアコンフィギュレーション内の信号からハードウェアピンリストを作成することができます。この場合、ハードウェアピンと各ハードウェア信号との仮想的な接続（「仮想コネクション」）も自動的に生成されます。

上記の操作により、コネクションマネージャ内で ECU とハードウェアとの接続が利用可能になり、接続することができるようになります。

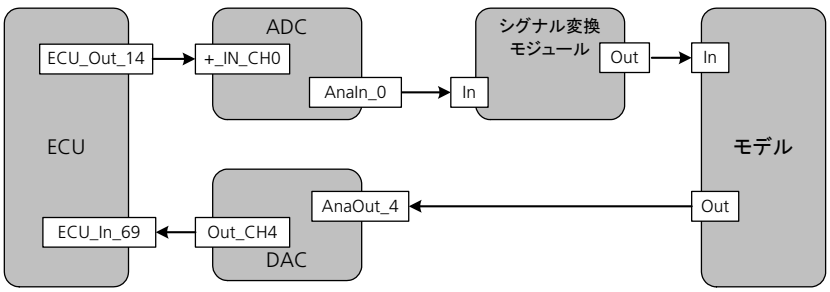


図 3-32 実験用信号（ECU とハードウェアの信号とピンを含む）

仮想コネクション

実験の際は、各モジュール内の信号トレースも明確になっている必要があります。

1 枚のハードウェアボートを簡単な例として挙げ、この HW ピンのリストは既存のコンフィギュレーションから自動的に生成されたものとします。この場合は、ピンと信号が自動的に正しく物理接続されます。また信号変換モジュールの入力と出力も互いに内部で接続されます。

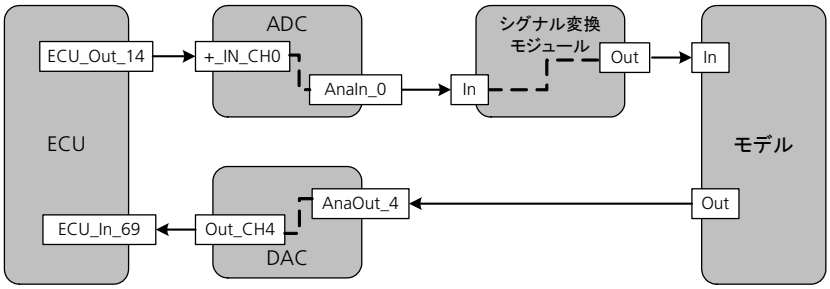


図 3-33 実験用信号（仮想コネクション - 破線部分 - を含む）

注記

ECU と LABCAR ハードウェアとの間のコネクションは、ECU とハードウェア間で実際に接続されるケーブルに対応します。
これらのコネクションについては、コンフィギュレーションマネージャにおいて仮想コネクション（図 3-33 内の "ECU_Out_14"、"+_IN_CH0"、"ECU_In_69"、"Out_CH4"）も作成する必要があります。これらにより、ECU と LABCAR ハードウェアとの間の信号のフルトレースが可能になります。

仮想コネクションの作成方法は、222 ページの「仮想コネクションを作成する：」を参照してください。

3.13.2 Simulink モデルへの入力ポートと出力ポートの追加

モデル信号を HIL アプリケーション用のハードウェア信号に接続する場合、モデル信号が LABCAR-OPERATOR からアクセス可能な状態になっている必要があります。モデル信号へのアクセスには、「LABCAR 入力ポート」および「LABCAR 出力ポート」と呼ばれるものを使用します。

これらのブロックは、Simulink モデルの任意の階層レベルに配置できます。

MIL アプリケーションでは、入力ポートに定数値を割り当てることにより、入力ポートを Simulink の「Constant ブロック」のように機能させることができます。この場合、LABCAR の出力ポートは Simulink の「Terminator ブロック」のように機能します。

これらの入力ポートと出力ポートは "LABCAR Port Blocks" ライブラリに含まれています（図 3-34）。

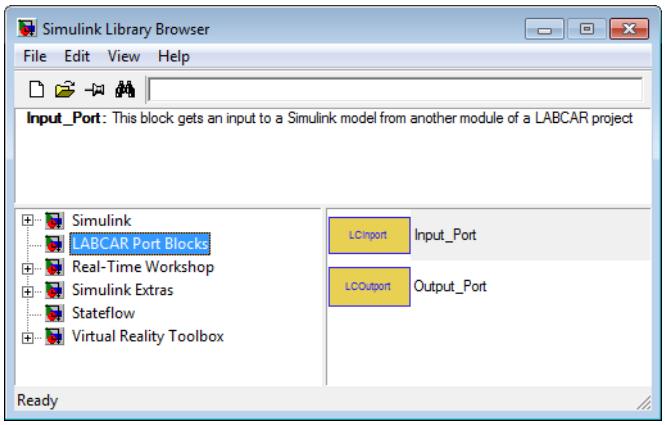


図 3-34 LABCAR Port Blocks

信号の接続は、コネクションマネージャで簡単に行えます（220 ページの「コネクションマネージャ内での信号の接続」を参照してください）。

このようにして接続を行えば、使用されるハードウェアに依存しないインターフェースを定義できるので、定義されたインターフェースを MIL アプリケーションにもそのまま使用することができます。つまり、モデルコードの生成後にハードウェア構成を変更した場合でも、モデルコードを再生成する必要がありません。

注記

これらのポートブロックがライブラリブラウザ内に表示されない場合は、以下の説明をお読みください。

ライブラリブラウザに LABCAR ポートブロックを追加する

Simulink ブラウザが開く際、Matlab によって、所定のパスから `slblocks.m` という名前のファイルが検索されます。この M ファイルは、Simulink モデルブラウザでエレメントを開く際の拡張機能が記述されたものです。

LABCAR-OPERATOR の場合、このファイルは、`<drive>:\Program Files\ETAS\LABCAR-OPERATORX.Y\SiCo` というディレクトリ（デフォルトディレクトリにインストールされている場合）に格納されています。

Simulink モデルを開く操作を LABCAR-OPERATOR から行わなかった場合は、このファイルが認識されないため、LABCAR ポートブロックはライブラリブラウザに表示されません。その場合は、Matlab コマンドウィンドウに下記のコマンドを入力し、パスを登録してください。

```
>> addpath('<drive>:\Program Files\ETAS\
      LABCAR-OPERATORX.Y\SiCo');
```

なお、Simulink モデルを編集する PC に LABCAR-OPERATOR がインストールされていない場合は、その PC に `¥SiCo` ディレクトリの内容をすべてコピーし、上記の方法でそのパスを MATLAB に登録してください。

データ型

LABCAR-RTC は "double" 型のデータしか処理しないので、それ以外の型と Simulink の基本データ型との間で自動的にキャスティングが行われるようにする必要があります。

- double（LABCAR 入力ポート）→ double、float、boolean、signed/unsigned int8、int16、int32、int64

- double、float、boolean、signed/unsigned int8、int16、int32、int64 → double（LABCAR 出力ポート）

3.13.3 コネクションマネージャ内での信号の接続

コネクションマネージャは、プロジェクトに含まれる全モジュールの入出力についての情報が格納されている XML ファイルから情報を取得します。

以下の図はコネクションマネージャのユーザーインターフェースで、メインワークスペースの " コネクションマネージャ " タブに表示されます。

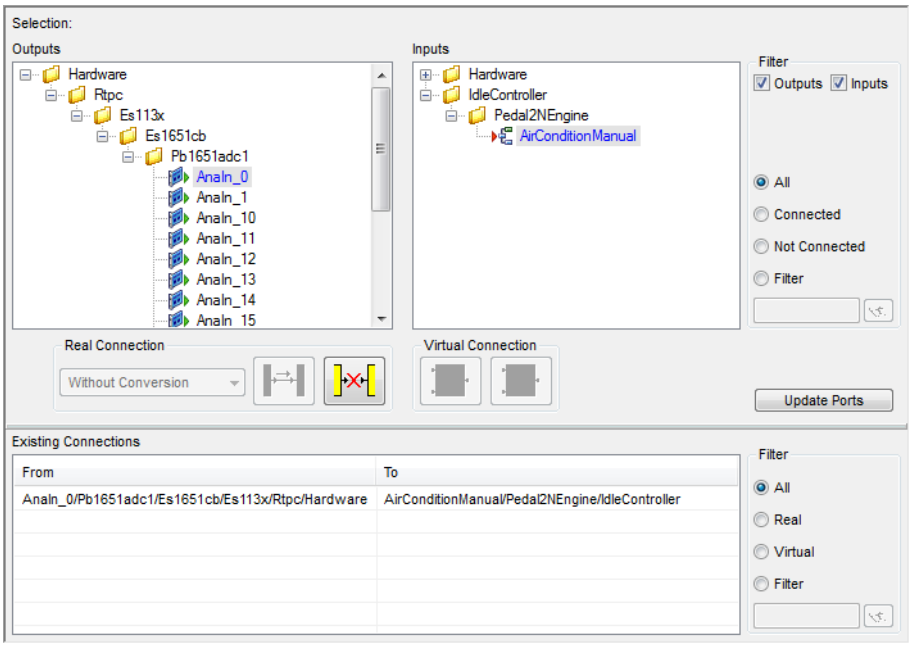


図 3-35 コネクションマネージャのユーザーインターフェース

このダイアログボックスの一番上には、存在するすべてのモデルの入力と出力が表示されます。

信号をエディタに表示する：

- 出力ポートまたは入力ポートのリストから、信号を選択します。
- マウスを右クリックし、ショートカットメニューから **Goto** を選択します。

  信号のタイプに応じたエディタが開き、選択された信号が強調表示されます。

表示のフィルタリング

コネクションマネージャでは、フィルタ（"Filters" フィールド）を指定することにより、特定の信号のみを表示することができます。

以下のフィルタ条件を使用できます。

- **All**

  使用可能な入力と出力がすべて表示されます。
- **Connected**

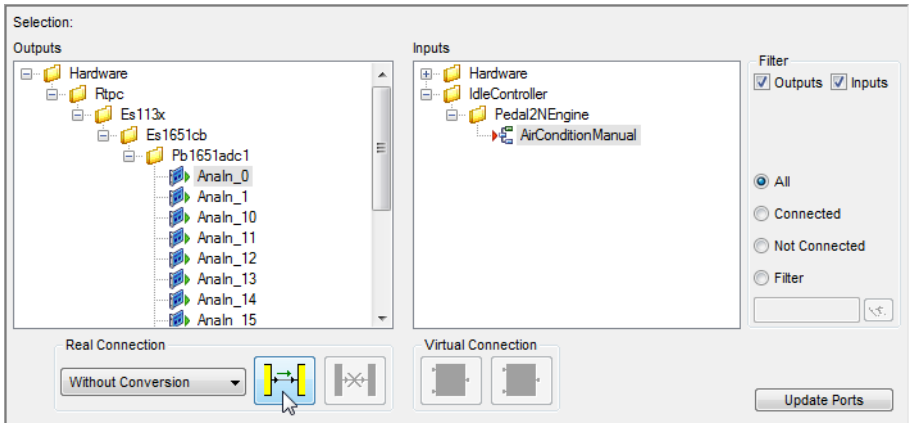
  接続されている入力と出力だけが表示されます。

- **Not Connected**
  接続されていない入力と出力だけが表示されます。
- **Filter**
  ここに、名前をフィルタリングするための文字列を入力できます。大文字と小文字は区別されません。

また、フィルタ条件の適用対象として、出力のみ（"For Outputs"）、入力のみ（"For Inputs"）、またはその両方を選択できます。

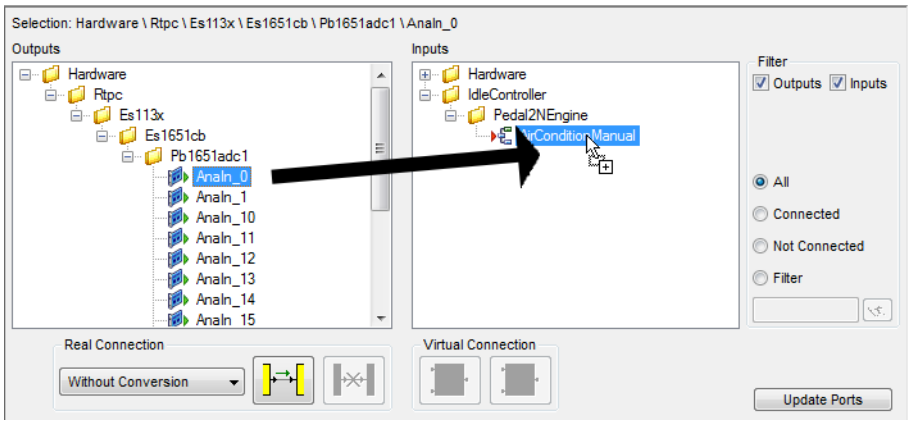
信号を接続する：

- 接続する入力と出力をマウスで選択します。
- 接続ボタンをクリックします。

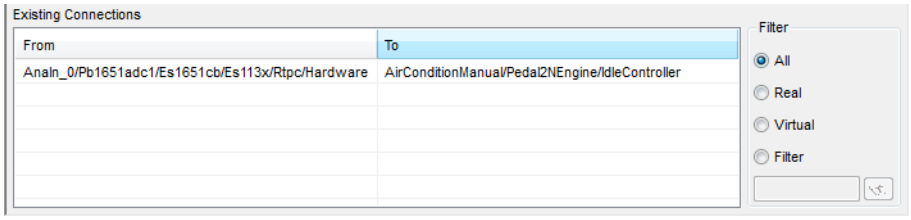


または

- マウスを使用して、入力／出力を接続先の出力／入力の位置に移動します。

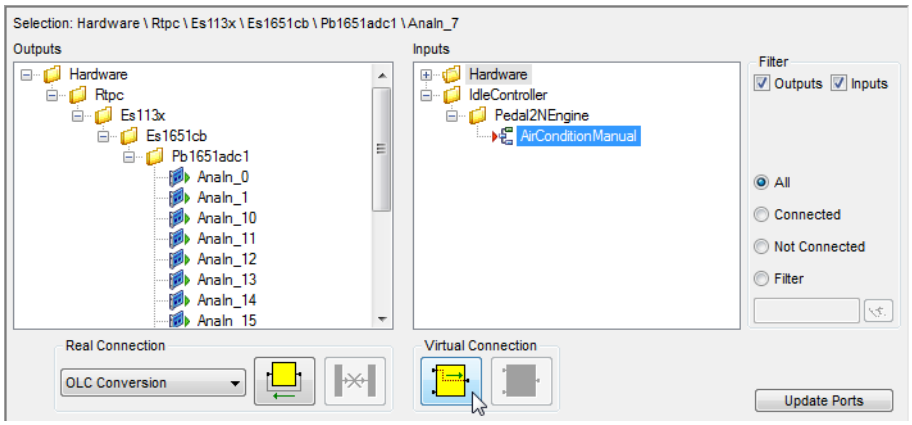


「コネクション」が作成され、"Existing Connections" フィールドにその内容が表示されます（新しく作成されたコネクションを表示しないようにするフィルタが有効になっていない場合のみ）。



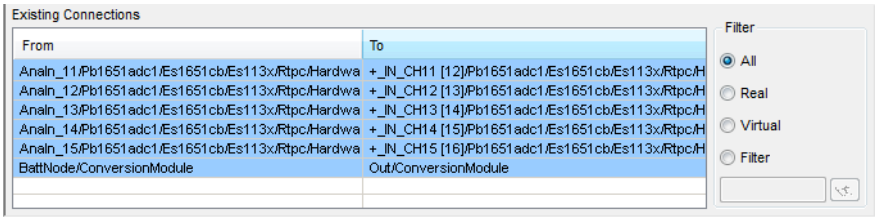
仮想コネクションを作成する：

同じモジュールの入力と出力の間の仮想コネクションを作成する方法は、実際のコネクションの場合と同じ（上記操作方法を参照してください）ですが、使用する接続ボタンのみが異なります。



右側のボタンを使用すると、同じモジュールの出力から入力への実際のコネクションを作成でき、これをフィードバック信号として使用することができます。

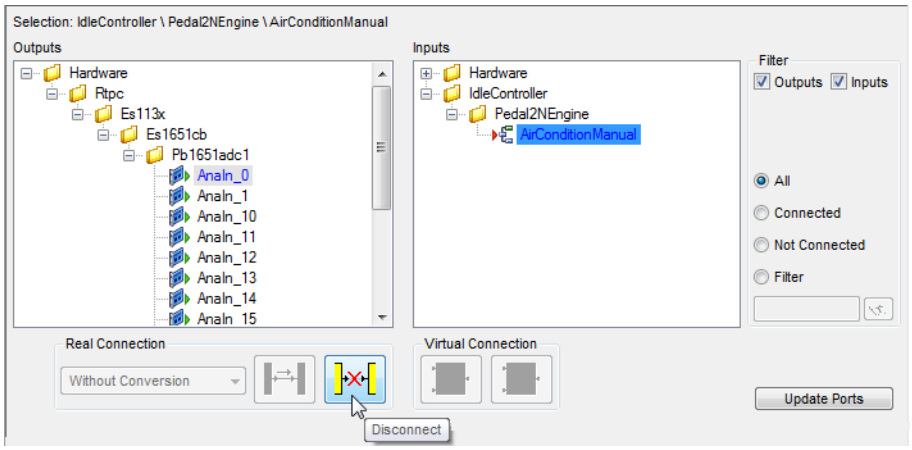
仮想コネクションは、コネクションマネージャに青の背景で表示されます。



コネクションを切断する：

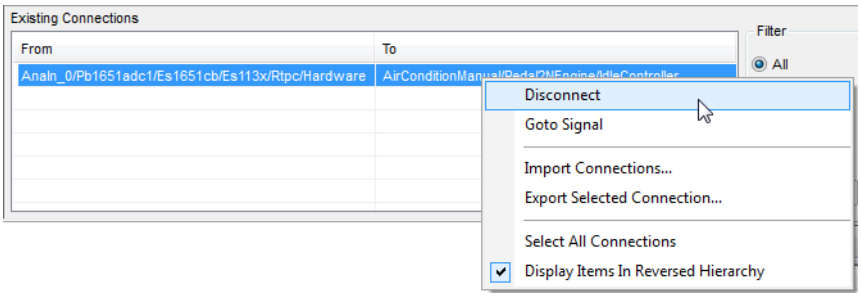
- 接続を切断する 2 つの信号の一方を選択します。

- 切断ボタンをクリックします。



または

- "Existing Connections" から、既存のコネクションを選択します。
- 右マウスボタンをクリックし、**Disconnect** を選択します。



コネクションが切断され、接続が解除されます。

モジュールの内容が変更された場合は、以下のようにして表示内容を更新できます。

表示を更新する：

- **Update Ports** を選択します。

  表示が更新されます。

コネクションリストをソートする：

- コネクションリストのヘッダ部分のタイトル "Output" または "Input" をダブルクリックします。

  コネクションリストの内容が、所定のカラムを基準にアルファベット順（昇順または降順）に並び替わります。

信号名の構成を逆にする：

- コネクションリストのショートカットメニューから **Display Items in Reversed Hierarchy** を選択します。

  信号名の構成が逆になり、信号名、モジュール名の順になります。

接続設定をエクスポートする：

- コネクションリストから、エクスポートしたいコネクションを選択します。複数のコネクションを選択するには、**<Ctrl>** または **<Shift>** キーを使用します。
- コネクションリストのショートカットメニューから **Export Selected Connection** を選択します。
  ファイル選択ダイアログボックスが開きます。
- ファイル名を入力します。
  選択されたコネクションがファイルに保存されます。

エクスポートされてファイルに保存された接続設定は、以下のようにしてインポートすることができます。

接続設定をインポートする：

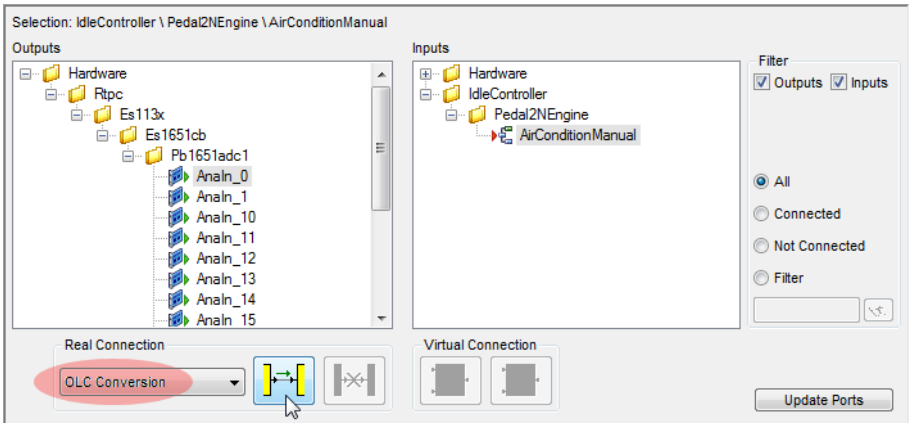
- コネクションリストのショートカットメニューから **Import Connections** を選択します。
  ファイル選択ダイアログボックスが開きます。
- インポートしたいコネクションデータが保存されているファイル選択します。
  コネクションが読み込まれます。

3.13.4 信号変換

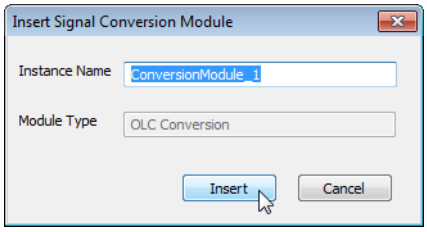
コネクションマネージャでは、信号間のコネクション上に汎用的な信号変換モジュール（アクチュエータ／センサ）を挿入することができます（209 ページの「信号変換モジュール」を参照してください）。これらのモジュールの設定は、LABCAR-EE の専用 GUI で行います。

信号変換モジュールを含むコネクションを作成する：

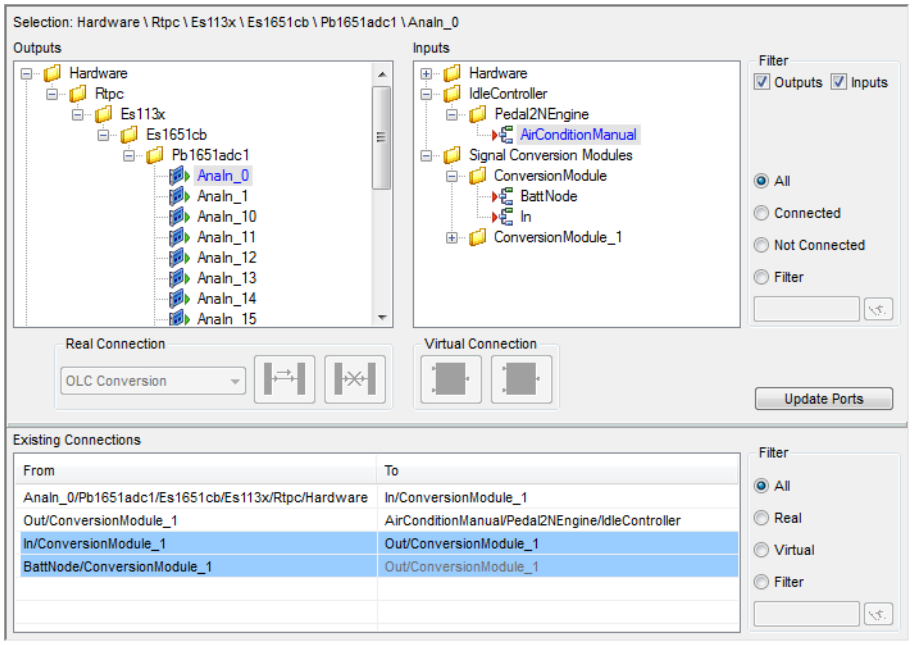
- 接続する 2 つの信号を選択します。
- コンボボックスから **OLC Conversion** を選択し、接続ボタンをクリックします。



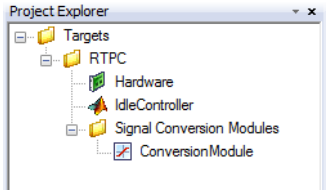
- 以下のダイアログボックスで、モジュール名を入力して **Insert** をクリックします。



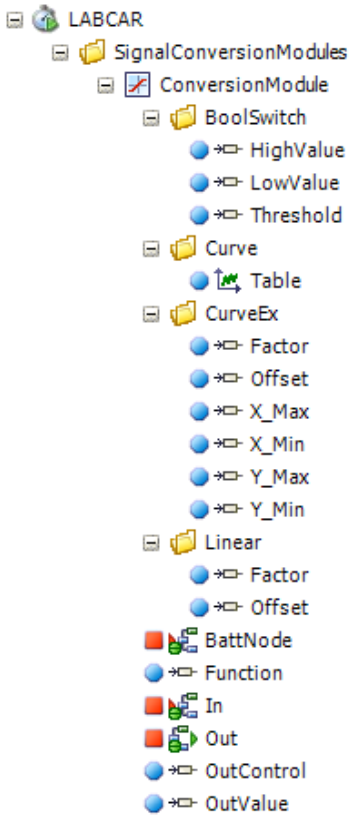
2 つの信号が、変換モジュールを経由して接続されます。これにより、たとえばハードウェア出力を変換モジュール入力に接続し、変換モジュールの出力をモデル入力に接続できます。



上図のように、変換モジュール内の仮想コネクションも青い背景で表示されます。
挿入された変換モジュールはプロジェクトエクスプローラ上にも表示されます。



また LABCAR-EE の "Workspace Elements" ウィンドウからは、挿入された変換モジュールのすべての入力と出力、およびパラメータにアクセスできます。



3.14 リアルタイムオレーティングシステムの設定（OS コンフィギュレーション）

LABCAR-OPERATOR プロジェクトを作成すると、自動的にデフォルトの OS コンフィギュレーションが生成され、プロジェクトに追加されます。

リアルタイムオペレーティングシステムのコンフィギュレーションは LABCAR-IP の OS コンフィギュレータで設定します。ここでは以下の設定を行えます。

- タスクの追加と削除
- タスクプロパティの編集
- プロセスのタスクへの割り当て／タスクからの削除

OS コンフィギュレータのユーザーインターフェースの使用方法は、232 ページの「OS コンフィギュレーションの編集」という項で説明します。

プロジェクトに新しいモジュールがリンクされると、適切なプロセスが自動的に OS コンフィギュレーションに組み込まれ、適切なタスクに割り当てられます。

3.14.1 OS コンフィギュレータのエレメント

以下に、"OS Configuration" タブの各フィールドで設定するオプションを紹介します。

グローバル設定

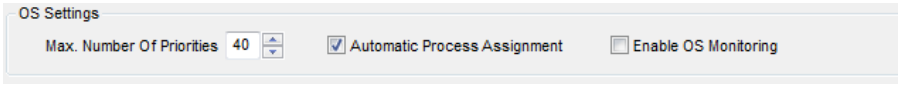


図 3-36 グローバル設定

- **Max. Number of Priorities**
  優先度レベルの最大数を指定します。
- **Automatic Process Assignment**
  新しいモジュールを LABCAR-OPERATOR プロジェクトに追加する際に、追加される新しいモジュール内のすべてのプロセスがそれぞれ対応するタスクに自動的に割り当てられます。

注記

ここでは、他のモジュールのプロセスの割り当ては影響されません。
自動割り当て後に行ったマニュアル操作によってエラーが発生した場合、そのそれを解決するには、**すべてのプロセス**をタスクから削除して、プロジェクトの次回のビルド処理において自動割り当てが行われるようにする必要があります。それ以外の解決策はありません。

- **Enable OS Monitoring**
  このオプションをオンにすると、次にビルドを行った後、監視用の測定変数のセットが LABCAR-EE の "Workspace Elements" ウィンドウ内の "OSMonitoring" の下に追加されます。

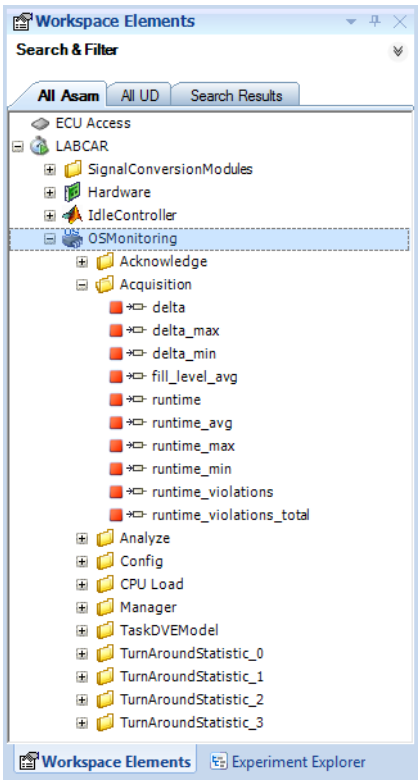


図 3-37 "Enable OS Monitoring" によって追加される測定変数

タスクの OS モニタリング

測定変数には以下のようなものがあります。

- delta / delta_max / delta_min
  タスクの実行間隔 dt（単位 : ns）と、その最大値と最小値
- fill_level_avg
  フィルレベル（平均の実行時間を周期で割った割合）
  例：100ms 周期で呼び出されるタスクの平均実行時間が 20ms の場合、フィルレベルは 20% になります。
- runtime (runtime_avg, runtime_max, runtime_min)
  タスクの現在の実行時間（単位 : ns）と、その平均値、最大値、最小値
- runtime_violations
  実行時間（"runtime"）が実行間隔（"delta"）を超えるたびにインクリメントされる整数値。
  この変数は定期的にリセットされます。リセットの間隔は、ETAS RTPC の web インターフェース（**Main Page → Configuration**）の RTPC_TASK_TIMING_STATISTIC パラメータ（"RTPC_TASK_TIMING_STATISTIC = 0" で記録オフ）で定義します。
- runtime_violations_total
  実験の実行時間全体における "runtime violations" の数

TurnAroundStatistic_n

"TurnAroundStatistic_*n*" フォルダ（*n* は VME シャシー番号で、Real-Time PC への接続に使用するイーサネットアダプタ "ETHn" に対応します）に 4 つの変数 "runtime"、"runtime_avg"、"runtime_min"、"runtime_max" が含まれ、各タスク内で使用されます。

"runtime" は下図のようにして算出されます。

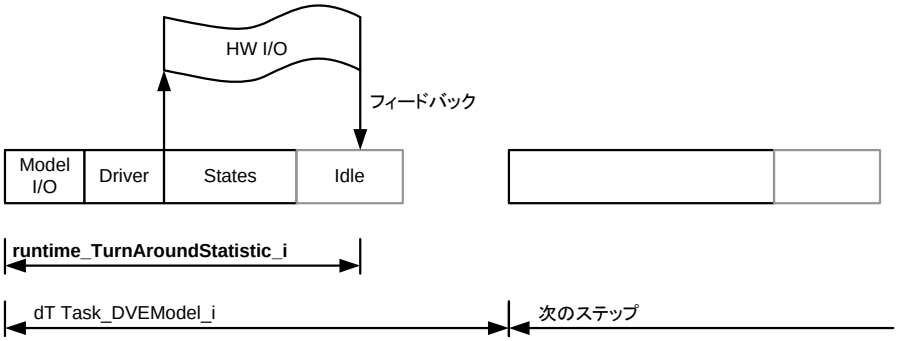


図 3-38 "runtime_TurnaroundStatistic" の算出

CPU 負荷

"CPU Load" フォルダ内の測定変数 "Core *n*" には、各プロセッサコアの負荷がセットされます。これを参照して、タスクを複数のコアに効率的に分散させることができます（230 ページの「CPU Core」を参照してください）。

タスク設定

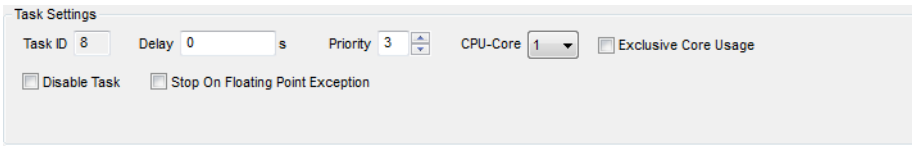


図 3-39 タスク設定

- **Show Unassigned Processes**
  このオプションがオンになっていると、タスクに割り当てられていないプロセス（231 ページ「Processes」を参照）だけが表示されます。
- **Update Processes**
  プロジェクト内のすべてのモジュールに含まれるプロセスが再検索されます。
- **Type**
  タスクタイプ（231 ページの「タスクとそのタイプ」を参照してください。）
- **Period**（タイマタスクの場合のみ）
  タイマタスクの周期（秒単位）
- **Task ID**
  タスクの ID（編集不可）。ソフトウェアタスクの呼び出しなどの際にこの ID が用いられます（231 ページの「タスクとそのタイプ」を参照してください）。
- **Delay**
  タスク起動の遅延時間（秒単位）

- **Priority**
  タスクの優先度を設定できます。数値が大きいほど優先度が高くなります。使用できる優先度レベルの数は、"OS Settings" グループフィールド内の **Max. Number of Priorities** オプションで設定できます（227 ページの「グローバル設定」を参照してください）。

注記

シミュレーションターゲット RTPC の各タスクはプリエンプティブスケジューリングで動作します。つまり、所定のタイムスライス内において、より高い優先度のタスクは常に指定されたプロセッサコアで処理されます。

- **CPU Core**
  タスクの演算を行うプロセッサコアの番号

注記

シミュレーションを分散させる（つまり各タスクを複数のコアに割り当てる）場合は、2 つ以上のコアが必要です。

- **Exclusive Core Usage**
  このオプションがオンになっていると、指定したプロセッサコアがこのタスク専用に確保されます。

注記

Exclusive Core Usage は、周期が非常に短いタスク（ハードウェアの I/O 処理など）を最短のレイテンシで実行できるようにするためのものです。このモードのタスクのアイドル時間は "Busy-Waiting" 状態になり、対応するプロセッサコアは常に 100% の稼働率となります。

- **Disable Task**
  タスクのステート："enabled" ／ "disabled"
- **Stop Simulation on Floating Point Exception**
  このオプションは、Real-Time PC がシミュレーションターゲットとして使用されている場合にのみ使用されます。
  このオプションがオンになっていると、浮動小数点値の値が異常な値（"Not-a-number"）になった場合、浮動小数点例外が発生し、ターゲットは停止します。エラー処理に関する詳細は、ETAS RTPC のユーザーズガイドを参照してください。

トリガタスク（タスクタイプ : **Trigger**）の場合は、さらに以下のオプションがあります。

- **Auto Trigger**
  このオプションがオンになっていると、トリガが発生していなくてもタスクが実行されます。これによりシミュレーションが停止してしまうのを防ぐことができます。
- **Period**
  トリガイベントが発生しない場合、タスクはこの周期で実行されます。
- **Event Timeout**
  この時間が経過してもトリガイベントが発生しない場合、自動トリガ機能が有効になります。

Processes

ユーザーインターフェースに表示されるプロセスは以下のとおりです。

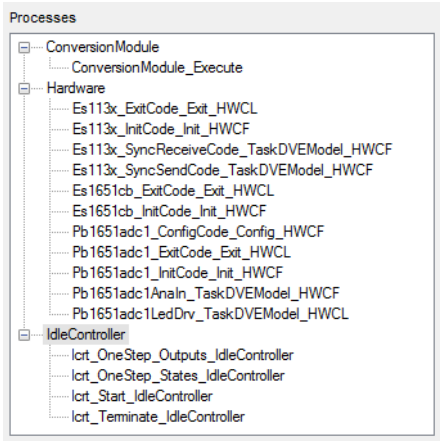


図 3-40 "OS Configuration" タブの "Processes" フィールド

タスクとそのタイプ

ユーザーインターフェースに表示されるタスクとそのタイプは以下のとおりです。

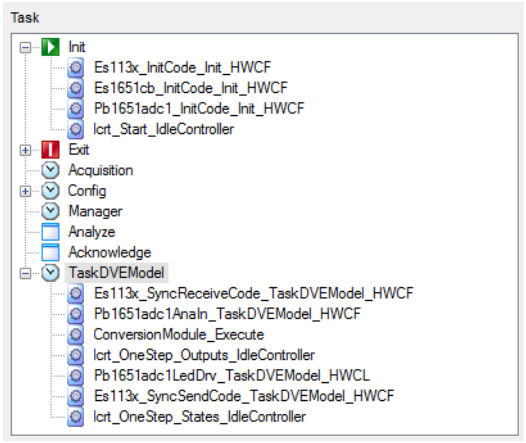


図 3-41 "OS Configuration" タブの "Tasks" フィールド

- **Init**
  シミュレーション開始時に必ず実行されます。

- **Exit**
  シミュレーション終了時に必ず実行されます。

- **Timer**
  固有の周期（"Period"）で実行されます。

- **Event**
  イベントによりコントロールされる通信を行うために ETAS RTPC が使用する特別なタスクです。タスク内のプロセスは、ハードウェアまたはネットワークからの入力を待つ「ブロッキングコール」を実行しなくてはなりません。
  このタスクはタイムアウト付きの CAN アクセスなどに利用されます。`rtos_comm_read()` を呼び出すと、CAN メッセージ到着の割り込みが発生するまでタスクはウェイト状態（ブロック状態）となります。これにより、CAN メッセージ受信に対する応答時間を非常に早くすることができます。

- **Software**
  `activateTaskWithId(uint32 taskId)` で起動できるタスクです。

- **Trigger**
  外部イベント（ハードウェアやネットワークなど）からトリガできるタスクです。

下の表は、デフォルトの OS コンフィギュレーションに含まれるタスク、およびそのタイプと設定内容をまとめたものです。

タスク名	タイプ	周期（秒）	優先度	用途
Acquisition	Timer	0.1	1	測定用
Init	Init	-		RTIO
Exit	Exit	-		RTIO
Config	Timer	0.1	1	RTIO
Manager	Timer	0.01	3	RTIO

表 3-2 デフォルトの OS コンフィギュレーションに含まれるタスク

処理順序

基本的に、プロセスは追加された順に実行されます（231 ページの図 3-41「"OS Configuration" タブの "Tasks" フィールド」の例を参照してください）。

複数の Simulink モデルが実行される場合、処理順序を変更することにより実行時間を最適化できる可能性があります。

注記

"Outputs" の処理は、必ず "States" より**前**に行われる必要があります。

3.14.2 OS コンフィギュレーションの編集

本項では、OS コンフィギュレーションの編集方法について、以下の各項目で説明します。

- OS コンフィギュレーションを編集する：（232 ページ）
- 詳細設定を表示する：（233 ページ）
- 割り当て先のタスクを確認する：（233 ページ）
- タスクを追加する：（234 ページ）
- タスク名を変更する：（235 ページ）
- タスク設定を割り当てる：（235 ページ）
- タスクを削除する：（235 ページ）
- タスクの順序を変更する：（235 ページ）
- タスクにプロセスを追加する：（236 ページ）
- タスクへのプロセス割り当てを解除する：（236 ページ）

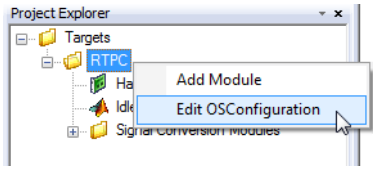
OS コンフィギュレーションを編集する：

- "OS Configuration" タブを選択します。

または

- プロジェクトエクスプローラからターゲット（例：RTPC）を選択します。

- 右クリックでショートカットメニューを開き、**Edit OS Configuration** を選択します。



"OS Configuration" タブが開き、プロジェクトの現在の OS コンフィギュレーションの内容が表示されます。

- すべてのプロセス、または各タスクとそのタスクに割り当てられているプロセスを表示するには、当該アイテムを展開します。

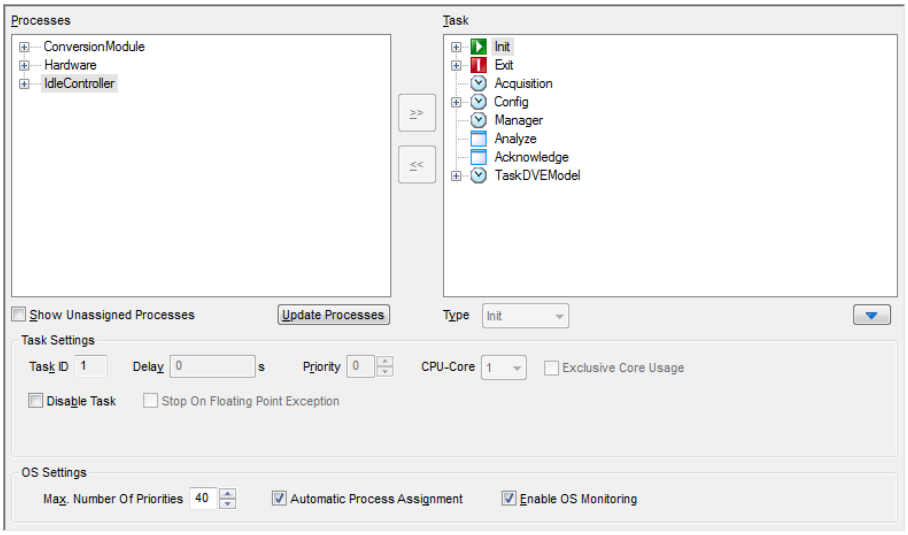


図 3-42 "OS Configuration" タブ（詳細設定ビュー）

詳細設定を表示する：

- ▲ ボタンをクリックします。

"Task Settings" および "OS Settings" というフィールドがダイアログボックスに追加されます。

"Processes" フィールド

このフィールドには、ハードウェアとモデルに属するすべてのプロセスがツリー表示されます。タスクに割り当てられていないプロセスを表示するには、**Show Unassigned Processes** オプションをオンにします。

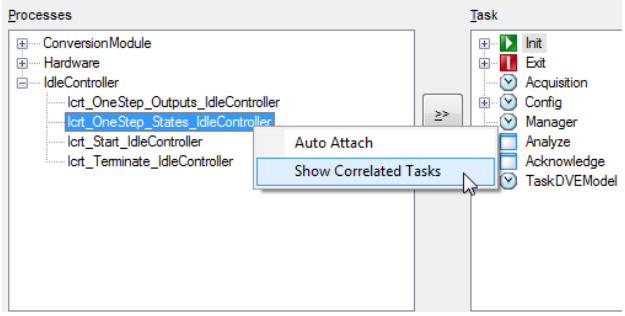
まだタスクに割り当てられていないプロセスについては、ショートカットメニューから **Auto Attach** コマンドを実行してください。

その際、プロセスがどのタスクに割り当てられるかを前もって確認するには、以下のように操作してください。

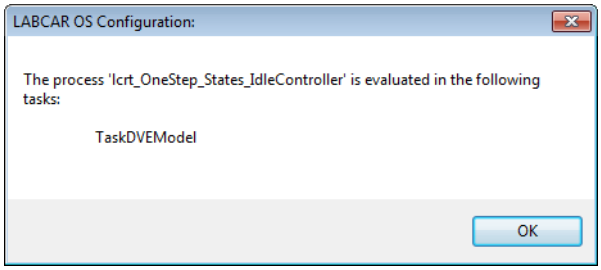
割り当て先のタスクを確認する：

- "Processes" フィールドからプロセスを選択します。

- ショートカットメニューから **Show Correlated Tasks** を選択します。



選択されたプロセスを割り当てるタスクが判定され、ダイアログボックスに表示されます。



"Tasks" フィールド

このフィールドには、すべてのタスクとそれらに割り当てられているプロセスがツリー表示されます。

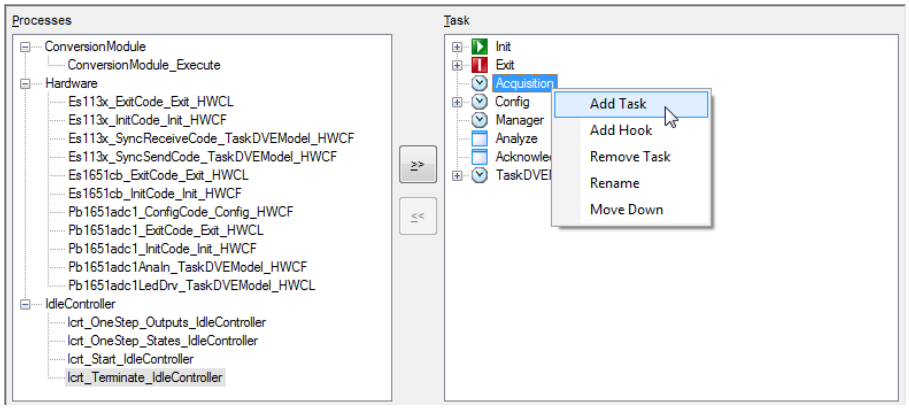
タスクを追加する：

- 新しいタスクを追加する場所のすぐ上に表示されているタスクを選択します。

注記

タスクの順序は後で変更することもできます（235 ページの「タスクの順序を変更する：」を参照してください）。

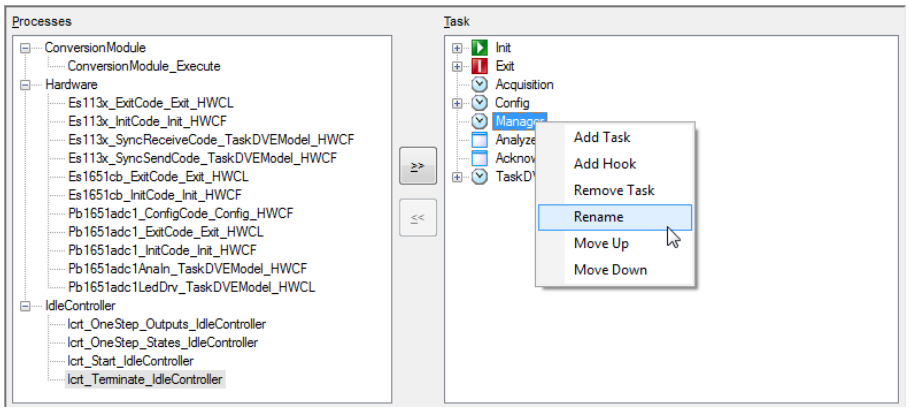
- ショートカットメニューから **Add Task** を選択します。



"Task_*n*"（n は連続番号）という名前の新しいタスクが追加されます。

タスク名を変更する：

- 名前を変更するタスクを選択します。
- ショートカットメニューから **Rename** を選択します。



タスク名が編集可能になります。

タスク設定を割り当てる：

- "Type" で新しいタスクのタイプを選択します。
- "Task Period" を指定します。

タスクを削除する：

- 削除するタスクを選択します。
- ショートカットメニューから **Remove Task** を選択します。

  タスクが削除されます。

タスクの順序を変更する：

- 移動するタスクを選択します。

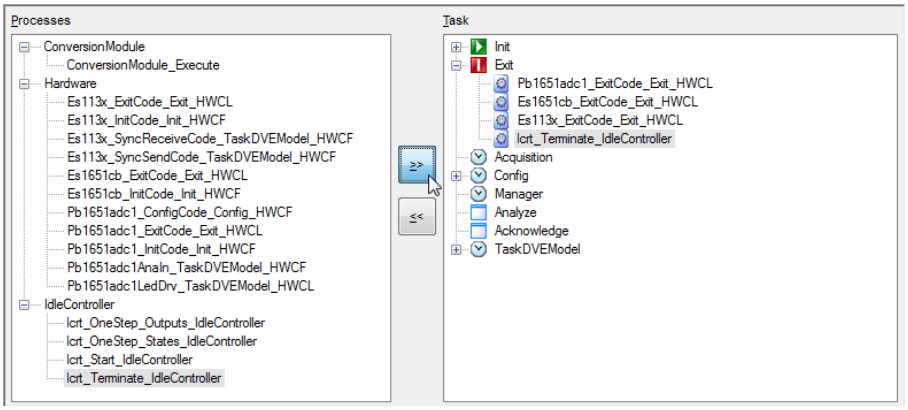
- ショートカットメニューから **Move Up** または **Move Down** を選択します。

  選択したメニューコマンドに従って当該タスクが移動します。

タスクにプロセスを追加する：

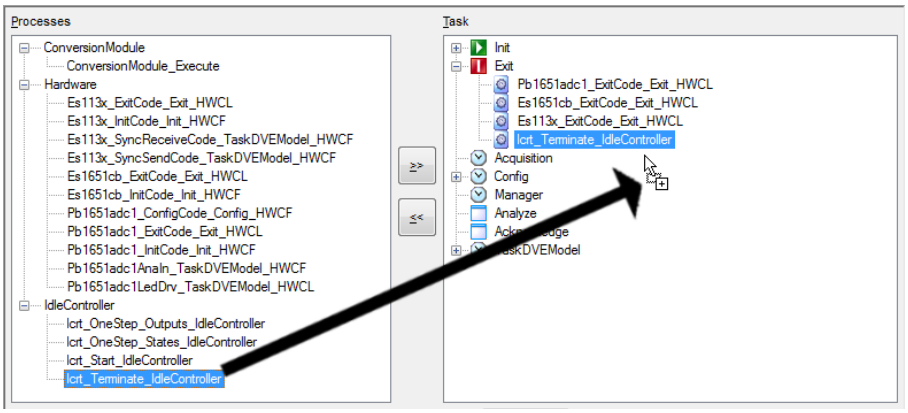
例として "Icrt_Terminate_IdleController" プロセスを "Exit" タスクに割り当てる手順を以下に示します。

- 割り当てるプロセスを、"Processes" フィールドから選択します。
- "Tasks" フィールドから、プロセスを割り当てるタスク内の、追加したいプロセスを挿入する場所のすぐ上のプロセスを選択します。
- **>>** ボタンをクリックします。



または

- 割り当てるプロセスを選択します。
- マウスボタンを押し下げたまま、挿入する場所のすぐ上のプロセスまでプロセスをドラッグします。



プロセスが、"Tasks" フィールド内の選択されたプロセスの下に追加されます。

タスクへのプロセス割り当てを解除する：

- 割り当てを解除するプロセスを選択します。

- ショートカットメニューから **Remove Process** を選択します。
  選択されたプロセスがタスクから削除されます。

タスク内にソフトウェアフックを作成するには、以下のように操作します。

フックを作成する：

- フックを追加したいタスクを選択し、右クリックします。
- ショートカットメニューから **Add Hook** を選択します。
  フックが作成されます。

フックの名前を変更する：

- フックの名前を変更するには、フックのショートカットメニューから **Rename** を選択します。

フックを削除する：

- フックを削除するには、フックのショートカットメニューから **Remove** を選択します。

3.15 マルチ RTPC ネットワークのセットアップ

図 3-43 は、複数の RTPC を使用した「マルチ RTPC」ネットワークの構成とコンポーネントの概略図です。

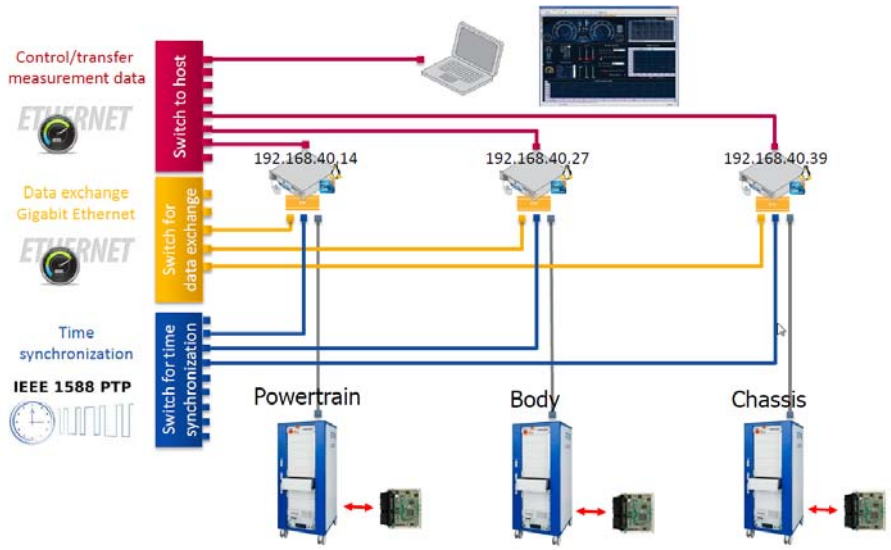


図 3-43 マルチ RTPC システムのアーキテクチャ

ここでは以下のコンポーネントと機能が必要です。

- ユーザー PC と各 Real-Time PC との間の測定データの制御と転送
- 各 Real-Time PC 間のデータ交換
- 各 Real-Time PC 間の時刻同期

ハードウェア面では、各機能やコンポーネントの接続はネットワークスイッチを使用して行います。

PTP 1588 による Real-Time PC の時刻同期

時刻の同期には PTP（**P**recision **T**ime **P**rotocol、IEC 61588 の一部として IEEE 1588 に定義されたもの）が使用されます。このプロトコルを使用すると、ネットワーク内のデバイスをより正確に同期させることができます。時刻の誤差は、PTP 対応のネットワークアダプタを使用する場合は 1 µs 未満、ソフトウェアで実装する場合は 1ms 未満になります。

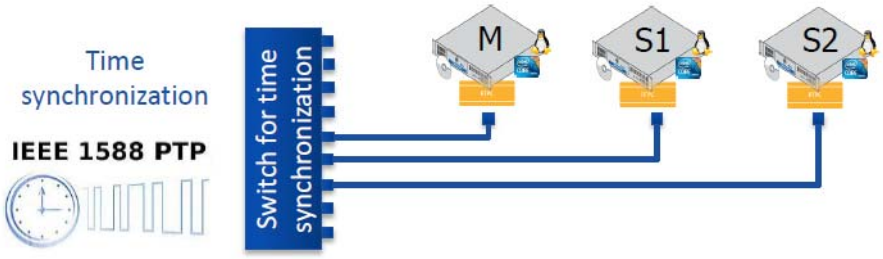


図 3-44 PTP による時刻同期

各 Real-Time PC 間のリアルタイムのデータ交換

複数の Real-Time PC 上で実行されるモデル／プロジェクト間では、データはリアルタイムインターフェースを経由して交換されます。UDP データのリアルタイム転送を行うためのライブラリは、ETAS RTPC の製品パッケージに含まれています。

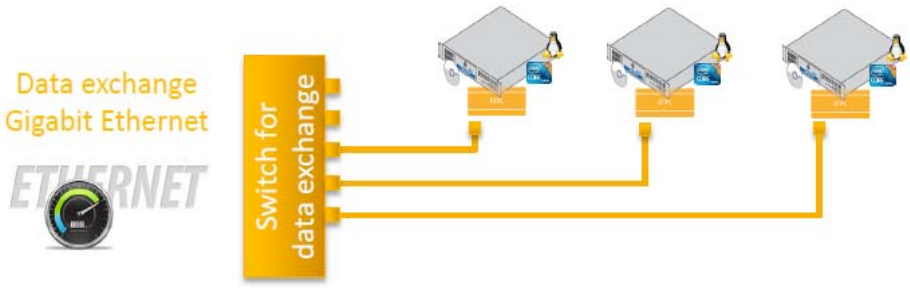


図 3-45 Real-Time PC 間のデータ交換

実現できる転送速度は PC の台数とメッセージタイプ（ポイントツーポイント、マルチキャスト／ブロードキャスト）により異なります。

ハードウェア要件

一般的に、マルチ RTPC ネットワークのハードウェア要件は以下のようになります。

- イーサネットポート（3 チャンネル以上）を備えた PTP 対応イーサネットアダプタが搭載された Real-Time PC（台数制限なし）
- 「Real-Time PC の台数 + 1」と同じ数のポートを備えた 3 個のイーサネットスイッチ
- イーサネットケーブル（Real-Time PC の台数 x 2 +1）
- LABCAR-OPERATOR がインストールされたユーザー PC

3.15.1 ハードウェアの接続

ユーザー PC と各 Real-Time PC との間の接続のためのスイッチ

各 Real-Time PC の eth0 ポートを対応するスイッチに接続し、それをユーザー PC に接続します。

PTP による各 Real-Time PC 間の時刻同期のためのスイッチ

各 Real-Time PC のいずれかのイーサネットポート（例：eth1）を適切なスイッチに接続します。

Real-Time PC 間のデータ転送のためのスイッチ

各 Real-Time PC のいずれかのイーサネットポート（例：eth2）を適切なスイッチに接続します。

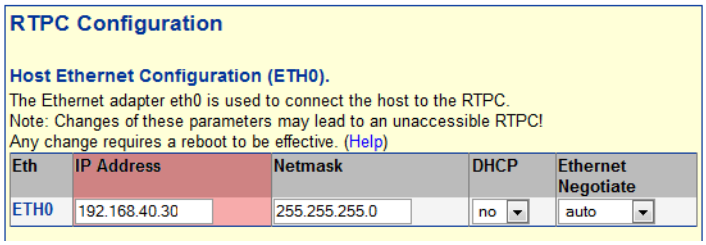
3.15.2 Real-Time PC の IP アドレスの設定

ユーザー PC への接続には、イーサネットアダプタ "eth0" を使用します。

注記

以下のように操作して、すべての Real-Time PC の IP アドレスが連続するように設定してください！

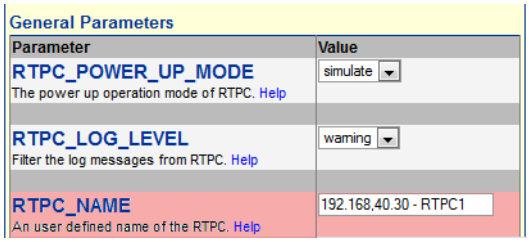
- Real-Time PC をユーザー PC に接続します。
- Real-Time PC を起動します。
- ETAS RTPC の web インターフェースを開き、"RTPC Configuration" セクションに進みます。



注記

192.168.40.14 は Real-Time PC 用のデフォルトアドレスです。設定時には 192.168.40.30 以降のアドレスを指定してください！

- 各 Real-Time PC に識別しやすい名前を付けます。



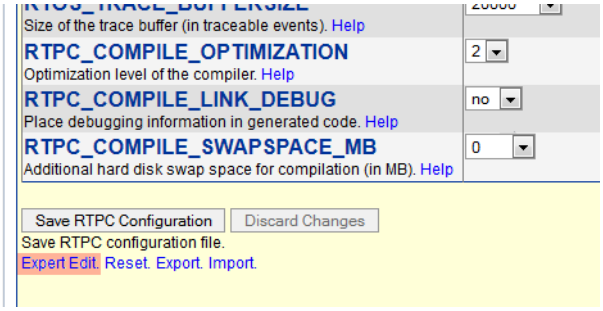
- RTPC を再起動します（**Main Page → Power Control → Reboot RTPC**）。

3.15.3 web インターフェースでの PTP 接続とデータ接続の設定

ETAS RTPC で PTP を有効化するには、以下のように操作してください。

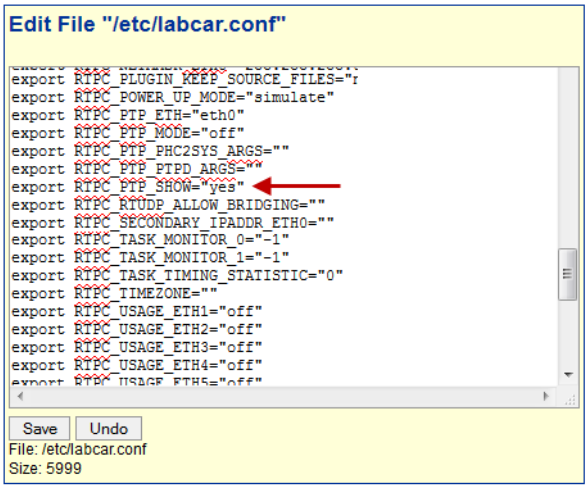
PTP を有効化する：

- ETAS RTPC の web インターフェース内をナビゲートして、セクション "RTPC Configuration" に進みます。
- **Expert Edit** をクリックします。

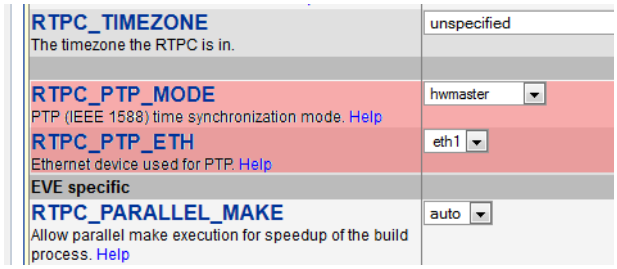


エディタフィールドにコンフィギュレーションファイルが開きます。

Configuration >> Expert Edit



- `RTPC_PTP_SHOW` パラメータの値を `yes` にします。
- **Save** をクリックして **Configuration** に戻ります。
  追加された 2 つのパラメータが表示されます。



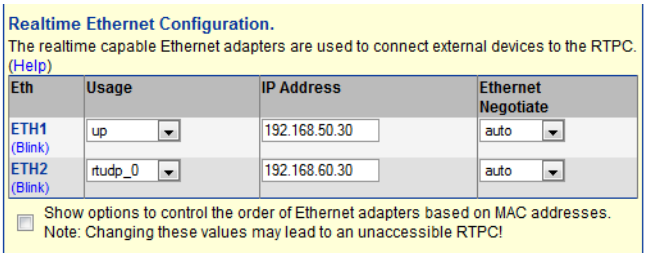
- 詳しい情報を見るには **Help** をクリックします。

ETAS RTPC で PTP 接続とデータ接続を有効化するには、以下のように操作してください。

PTP 接続とデータ接続を設定する：

- **RTPC_PTP_MODE** パラメータは、PTP-Master とする Real-Time PC では **hwmaster** を選択し、他の Real-Time PC では **hwslave** を選択します。
- "Realtime Ethernet Configuration" フィールドでは、PTP スイッチ用のポートに IP アドレスを割り当てて、**Usage** を **up** に設定します。

  下図の例では、"ETH1" の IP アドレスが 192.168.50.30 に設定されています。



- すべての Real-Time PC について、データ交換用イーサネットポート（この例では ETH2、192.168.60.30）の **Usage** を **rtudp_0** に設定します。

注記

すべての PTP ポートとデータポートはそれぞれ同じスイッチに接続されるので、各 IP アドレスはすべて異なってる必要があります。1 台の Real-Time PC 上のすべてのポートについて、アドレスの最後の 1 バイトを同じ値（上の例では 30）にすることを推奨します。

- **Save RTPC Configuration** をクリックします。
- 再起動します。

243 ページの図 3-46 にコンフィギュレーションパラメータをまとめて示しています。

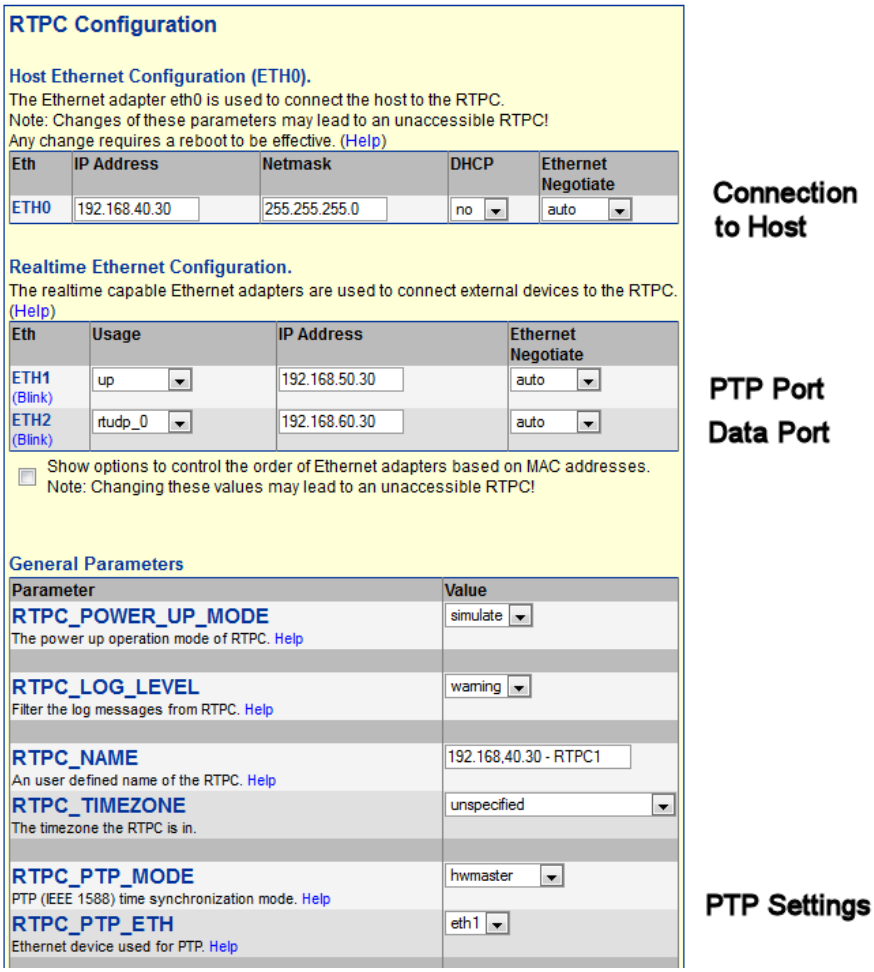
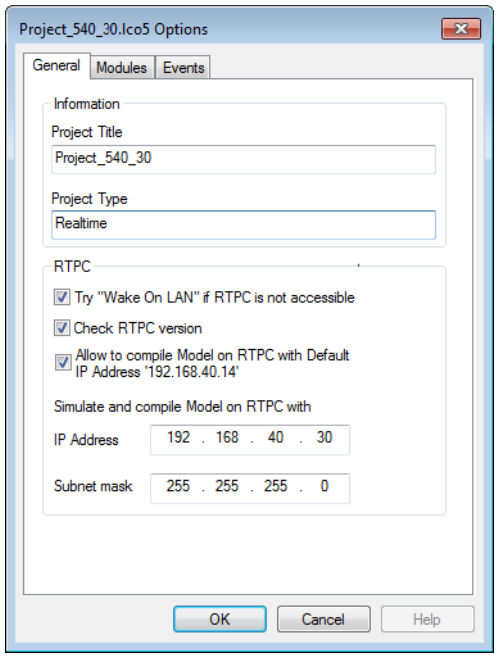


図 3-46 web インターフェースでの Real-Time PC の設定

3.15.4 LABCAR-OPERATOR プロジェクトの作成

LABCAR-OPERATOR プロジェクトは、ネットワーク内のすべての Real-Time PC にそれぞれ 1 つずつ作成する必要があります。

Project → **Options** を選択すると、プロジェクトが作成されて実行される Real-Time PC の IP アドレスを設定することができます。



ネットワークターゲットが使用できない状態である場合は、**Allow to compile...** オプションをオンにすると、IP アドレスが 192.168.40.14 の「デフォルトターゲット」上でプロジェクトをコンパイルすることができます。

注記

すべてのプロジェクトのモジュール（ハードウェアモジュールを含む）の名前は、互いに異なっている必要があります。

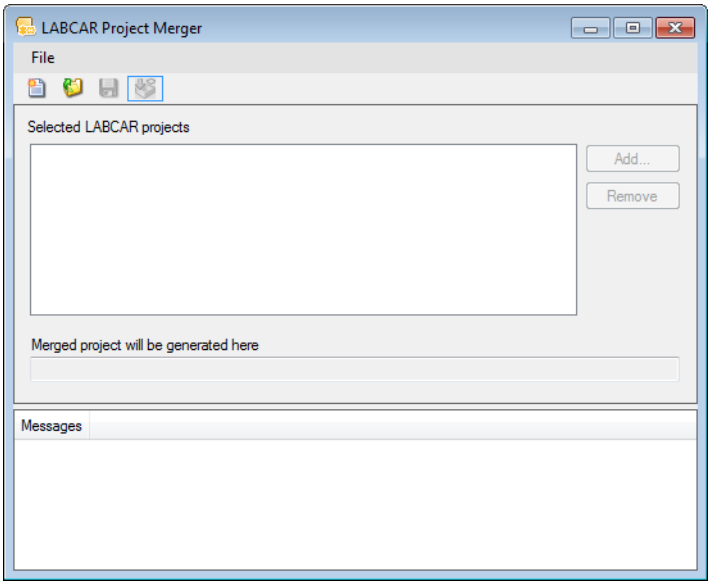
3.15.5 プロジェクトのマージ

複数のプロジェクトをマージするには、以下のように操作してください。

プロジェクトコンフィギュレーションを作成する：

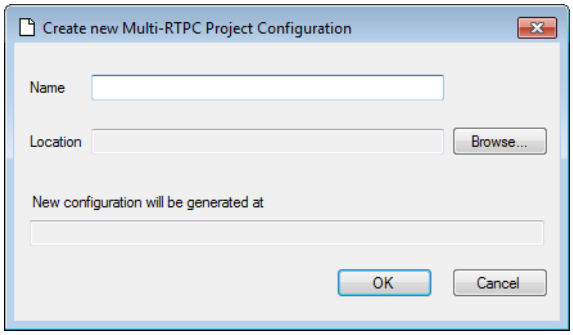
- `<LABCAR-OPERATOR Installation directory>/ProjectMerger/ProjectMerger.exe` というプログラムを起動します。

  LABCAR プロジェクトマージャ（LABCAR Project Merger）が起動されます。



- **File → New** を選択します。

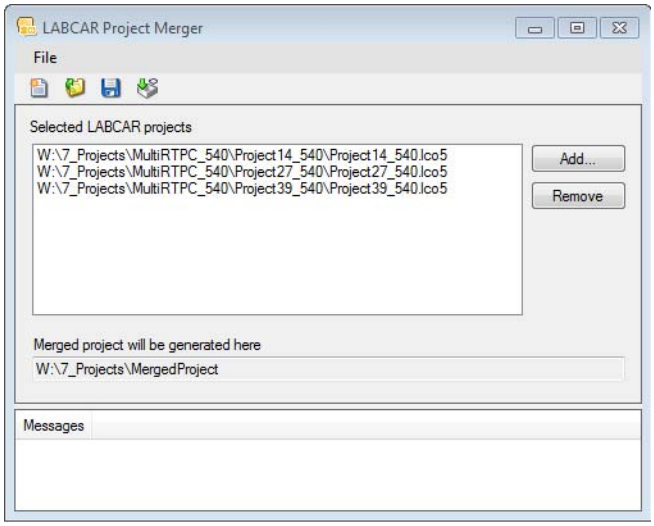
  "Create New Multi-RTPC Project Configuration" ダイアログボックスが開きます。



- プロジェクトコンフィギュレーションの名前とディレクトリを指定します。
- **OK** をクリックします。

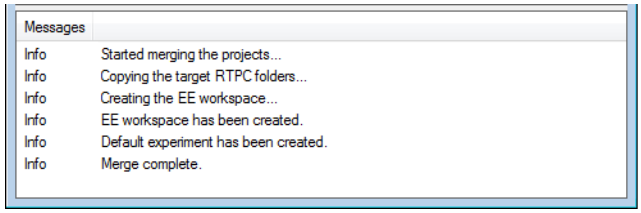
  選択されたディレクトリパスに、指定された名前のディレクトリが作成され、その中にコンフィギュレーションファイル `<Name>.multirtpc` が作成されます。

- **Add** をクリックして、作成するマルチ RTPC プロジェクトにマージしたい LABCAR-OPERATOR プロジェクトを選択します。



- **File** → **Save** を選択してコンフィギュレーションを保存します。
- **File** → **Merge** を選択します。

  マルチ RTPC プロジェクトが作成され、通知（エラーメッセージなど）が "Messages" セクションに表示されます。



4 LABCAR-EE の概要

LABCAR-EE（**E**xperiment **E**nvironment：実験環境）は、LABCAR-OPERATOR の実験を実行するための環境です。本章では LABCAR-EE の機能と GUI（**G**raphic **U**ser **I**nterface）について概説します。

注記

本章は LABCAR-EE の概要をまとめたものです。詳細な情報は LABCAR-EE のオンラインヘルプをご参照ください。

本章は以下の項に分かれています。

4.1 GUI の構成

LABCAR-IP から実験を開くと、以下のような LABCAR-EE の GUI（グラフィックユーザーインターフェース）が起動します。

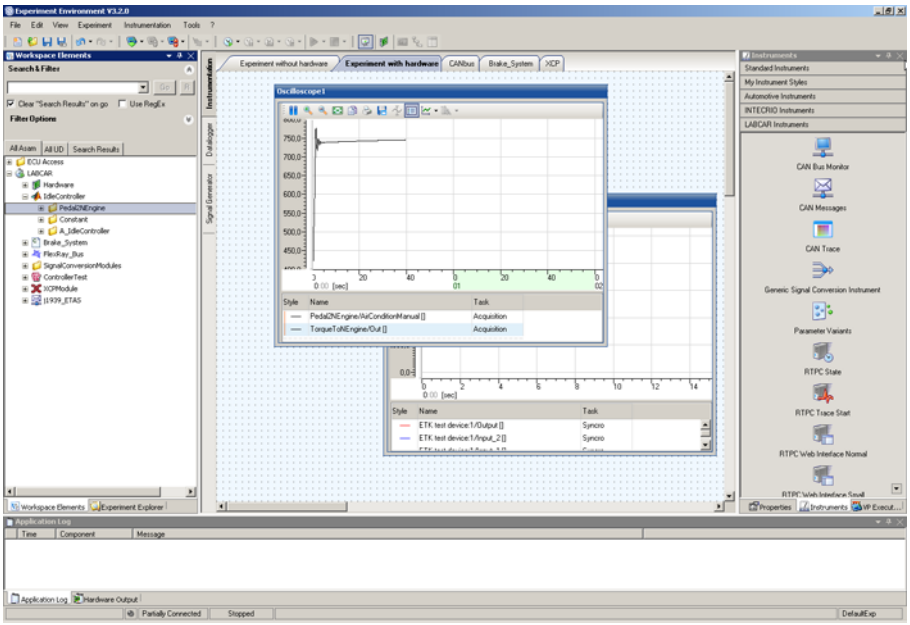


図 4-1 LABCAR-EE の GUI

以下に、この GUI の各パーツについて説明します。

4.1.1 "Workspace Elements" ウィンドウ

この "Workspace Elements" ウィンドウから、実験に含まれる各モジュールのすべての測定変数、パラメータ、信号入力／出力にアクセスできます。

またこのウィンドウには、階層、エレメント、データ、オブジェクトタイプを基準にフィルタリングを行う機能もあり、また正規表現による検索の機能も備えています。

4.1.2 エクスペリメントエクスプローラ（"Experiment Explorer" ウィンドウ）

このウィンドウで、コンフィギュレーション設定や実験の実行に必要なすべてのファイル（データロガーやシグナルジェネレータセット用のパラメータファイル、コンフィギュレーションファイルなど）を管理します。

4.1.3 メインワークスペース

実験の主要な GUI にはメインワークスペースの各タブからアクセスします。

"Instrumentation" タブ

複数のレイヤを作成し、各レイヤにさまざまなインストゥルメント（GUI）を割り当てて実験を実行します。

"Signal Generators" タブ

実験で使用するシグナルジェネレータの設定と操作を行います。

"Datalogger" タブ

データロガーの設定と操作を行います。

4.1.4 "Instruments" ウィンドウ

このタブに複数のレイヤを作成し、各レイヤにさまざまなインストゥルメント（GUI）を割り当てます。

4.1.5 "Properties" ウィンドウ

このウィンドウには、GUI 上で選択されているオブジェクト（パラメータ、インスゥルメントなど）のプロパティとメタデータが表示されます。

4.1.6 "Application Log" ／ "Hardware Output" ウィンドウ

これらのウィンドウには、アプリケーションから出力される情報、ワーニング、エラーメッセージなどが表示されます。

4.1.7 LABCAR-EE のメインメニュー

注記

本項は LABCAR-EE のメニューバーに含まれる各メニューの概要をまとめたものです。詳細な情報は LABCAR-EE のオンラインヘルプをご参照ください。

"File" メニュー

このメニューにはファイル（ワークスペースファイルと実験ファイル）用のアクションコマンドが含まれます。

"Edit" メニュー

Undo および **Redo** コマンドが含まれます。

"View" メニュー

このメニューで、GUI 上の各種コンポーネントの表示／非表示を切り替えたり、ウィンドウサイズの最大化（**<F11>** キーにも同じ機能が割り当てられています）を行うことができます。

"Experiment" メニュー

このメニューには、シミュレーションコードをターゲットにダウンロードしたりシミュレーションを制御したりするために必要なコマンドがすべて含まれています。

"Instrumentation" メニュー

このメニューで、"Instrumentation" タブ内のレイヤ管理を行います。

"Tools" メニュー

このメニューで、実験用の各種オプション設定を行えます。

"?" メニュー

このメニューから、オンラインヘルプ、ライセンス情報、製品情報、サポート情報にアクセスできます。

4.1.8 ツールバー

ツールバーの操作ボタンは、以下のグループに分類されます。

File

実験の管理に関するコマンド用ボタンが含まれます。

Experiment

ターゲットへのダウンロードや接続を行うためのコマンド用ボタンが含まれます。

Simulation

シミュレーションと測定に関する操作を行うためのコマンド用ボタンが含まれます。

Instrumentation

インストゥルメントの操作を行う際に頻繁に使用されるボタンが含まれます。

4.2 エクスペリメントエクスプローラ（"Experiment Explorer" ウィンドウ）

エクスペリメントエクスプローラは、コンフィギュレーション設定や実験の実行に必要なすべてのファイル（データロガーやシグナルジェネレータセット用のパラメータファイル、コンフィギュレーションファイルなど）を管理するためのウィンドウです。

ここで、実験に含まれる各データのアクティブ化、編集、削除などを行えます。

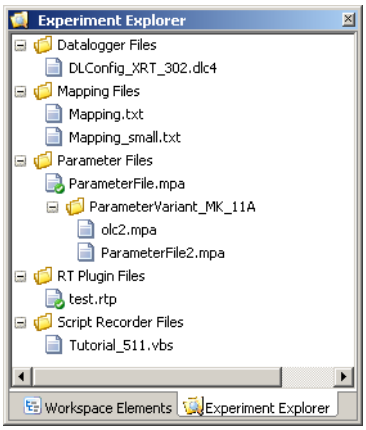


図 4-2 エクスペリメントエクスプローラ

4.2.1 エクスペリメントエクスプローラの機能

このウィンドウに表示された各種オブジェクト（フォルダやファイル）を右クリックしてショートカットメニューを開くと、以下のようなコマンドを使用できます。

全フォルダ用コマンド

- **Add File**
  ファイル選択ダイアログボックスでファイルを選択し、追加します。

特定のフォルダに関するコマンド

"Parameter Files" フォルダ：

- **Set Display Order**
  ダイアログボックスで、既存のパラメータファイルの表示順、つまりターゲットにダウンロードされる順番を指定します。
- **Download all active files**
  実験への接続が確立されているときに、アクティブなパラメータファイルをすべてターゲットにダウンロードします。
- **Add New Parameter Variants**
  サブフォルダを作成し、そこにパラメータファイル（「バリアント」）を追加します。

全ファイル用コマンド

- **Add File**
  ファイル選択ダイアログボックスでファイルを選択し、追加します。
- **Rename File**
  ファイル名を変更します。

- **Exclude From Experiment**
  ファイルを実験内で無効化します。ファイル自体は削除されません。
- **Delete (permanently)**
  ファイルを実験から除外し、削除します。

特定のタイプのファイルに関するコマンド

- **Edit File**
  - "Mapping Files" フォルダ内のファイル（`*.txt`、`*.smf`）
    マッピングファイルを SUT マッピングファイルエディタで開きます。
  - "Parameter Files" フォルダ内のファイル
    `*.mpa` の場合：パラメータファイルを LABCAR-PA（Parameterization Assistant）で開きます（305 ページの「LABCAR-PA 1.0 を用いたパラメータファイルの編集」を参照してください）。
    `*.dcm` の場合：ファイルをテキストエディタで開きます。
- **Active**
  各タイプのファイルを「アクティブ」にします。アクティブになったファイルはこのコマンド実行時にダウンロードされ、その後も、実験がターゲットにダウンロードされるたびに他のすべてのアクティブなファイルとともにダウンロードされます。
  - "Datalogger Files" フォルダ内のファイル（`*.dlc4`）
    選択されたデータロガーコンフィギュレーションをアクティブにします。
  - "Mapping Files" フォルダ内のファイル（`*.txt`）
    選択されたマッピングファイルをアクティブにします。
  - "Parameter Files" フォルダ内のファイル（`*.mpa`、`*.dcm`、`*.cdfx`、`*.olc4`）
    選択されたパラメータファイルまたは開ループコンフィギュレーションをアクティブにします。
- **Set Display Order**
  - "Parameter Files" フォルダ内のファイル
    ダイアログボックスで、既存のパラメータファイルの表示順、つまりターゲットにダウンロードされる順番を指定します。
- **Reload Mapping**
  - "Mapping Files" フォルダ内のファイル（`*.txt`）
    選択したファイルの内容に従ってマッピングを更新します。

4.3 "Workspace Elements" ウィンドウ

"Workspace Elements" ウィンドウは実験環境 LABCAR-EE の主要なウィンドウの 1 つです。ここにはプロジェクト内の各エレメントが以下のフォルダに分類されて表示され、さまざまな管理を行えます。

- Simulink モデルと ASCET モデル
- C モジュール（信号変換モジュールを含む）
- CAN モジュールと FlexRay モジュール
- FiL モジュール
- ハードウェア
- ECU アクセス

これらのフォルダ内で以下のことを行えます。

- すべてのパラメータおよび測定変数へのアクセス
- モジュールのすべての入力と出力（信号）へのアクセス
- ハードウェアと ECU ポートへのアクセス

またフィルタや検索機能を使用して、特定のタイプのオブジェクトのみを表示することができます。

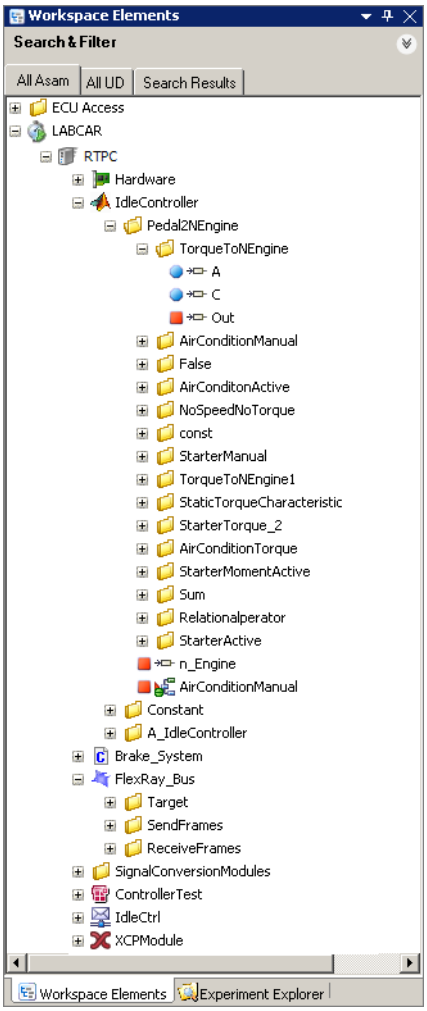


図 4-3 "Workspace Elements" ウィンドウ

4.3.1 "Workspace Elements" ウィンドウの各タブ

"Workspace Elements" ウィンドウには以下のタブがあり、エレメントが異なるビューで表示されます。

- "All ASAM" タブ

  ASAM ラベルまたはモジュール仕様に基づき、全エレメントがツリー形式で階層表示されます。

- "All UD" タブ

  マッピングファイル内のユーザー定義名が "All ASAM" タブと同じ形式で表示されます。

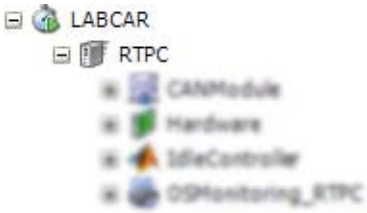
- "Search Result" タブ

  最後に実行された検索の実行結果が表示されます（258 ページの「検索機能とフィルタ機能」を参照してください）。

4.3.2 "Workspace Elements" ウィンドウの機能

254 ページの図 4-3 には、さまざまなタイプのエレメントが各種アイコンと共に表示されています。

最上位の "LABCAR" エレメントの下に、プロジェクトが割り当てられた Real-Time PC を示すエレメント（"RTPC"）が表示されます。マルチ RTPC プロジェクトの場合は、プロジェクト内に複数の Real-Time PC が含まれます。



Real-Time PC の下には複数のプロジェクトモジュールが存在します。

モジュール

モジュールは、以下のアイコンと共に表示されます。

- ASCET モジュールと MATLAB/Simulink モジュール

- C コードモジュール

- CAN、LIN、FlexRay モジュール

- FiL モジュール

- 開ループアクセスモジュールと信号変換モジュール

- I/O ハードウェアモジュール

各モジュールの測定変数とパラメータ

各モジュールの測定変数とパラメータは、以下のアイコンと共に表示されます。


- 測定変数


- パラメータ（適合変数）


- モジュール出力（測定変数）


- モジュール入力（測定変数）


- モジュールに対するハードウェア出力（測定変数）


- モジュールからのハードウェア入力（測定変数）


- ハードウェア出力ピン


- ハードウェア入力ピン


- ECU 出力ピン


- ECU 入力ピン


- CAN 送信メッセージ


- CAN 受信メッセージ

注記

パラメータの扱いについての詳細は 287 ページの「パラメータの使用法」を参照してください。

ショートカットメニュー

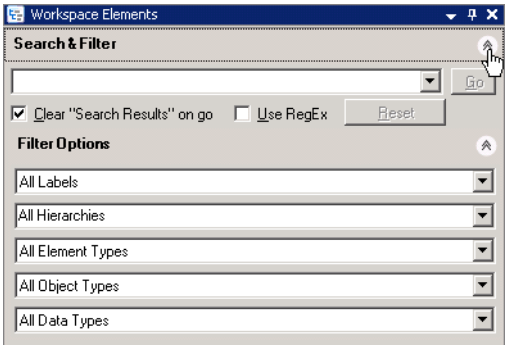
このウィンドウでエレメントを操作するには、ショートカットメニューを使用します。エレメントのタイプに応じて以下のようなコマンドが用意されています。

- **Measure / Calibrate**
  "Create Instruments" ダイアログボックスを開き、エレメントを既存のインストゥルメント、または新しいインストゥルメントに割り当てます。
- **Find in Instruments**
  測定変数やパラメータがインスゥルメントに割り当てられているかを調べ、結果を "Application Log" ウィンドウに出力します。見つかったエレメントをダブルクリックすると、そのエレメントが含まれるレイヤが最前面に表示されます。
- **Show All HW Ports**
  すべてのハードウェア信号（監視用信号、測定モード制御用信号等）を "Workspace Elements" ウィンドウに表示します。

- **Add to Datalogger**
  エレメントをデータロガーに追加します。複数のデータロガーが存在する場合はいずれか 1 つを選択します。
- **Calibration / Parameters**
  このコマンドはパラメータについてのみ有効で、実験がターゲットにダウンロードされた後に使用可能となります。
  - **Copy Values**
    パラメータの値をクリップボードにコピーします。
  - **Paste Values**
    クリップボード上の値をパラメータにコピーします。
  - **Save**
    ダイアログボックスを開き、パラメータを `.mpa` ファイルに保存します。**<Shift>** キーを利用して複数のパラメータを選択することができます。
  - **Set State To Modified/Unmodified**
    パラメータのステートを、"Modified"（赤い文字で表示）または "Unmodified"（黒い文字で表示）に変更します。
- **Signal**
  このコマンドはハードウェアの入力と出力についてのみ使用可能で、サブメニューから以下のコマンドを実行できます。
  - **Trace**
    信号トレース用のダイアログボックスを開きます。
  - **Save Input Signal Settings**
    入力の設定と値を `.olc4` ファイルに保存します。
  - **Reset Full Path to "Model"**
    パス上の信号を "MODEL" にセットし、開ループをすべて閉じて閉ループにします。
  - **Reset Input Setting To**
    OLC 設定の値を変更します。ダイアログボックスで "MODEL"、"CONST" などの値を選択できます。
- **CAN**
  CAN メッセージについて、**Disable/Enable all Messages** コマンドでシミュレーション時のメッセージ送信を有効／無効にできます。
- **Reload Mapping**
  アクティブなマッピングファイルを再ロードします。

4.3.3 検索機能とフィルタ機能

"Workspace Elements" ウィンドウでは、エレメントの検索や、エレメント表示のフィルタリングを行うことができます。



検索機能

検索を行うには、検索文字列を入力して **Go** をクリックします。検索されたエレメントは "Search Results" タブに表示されます。

Clear Search Results on Go オプションがオンになっていると、検索を行うたびに前回の検索結果が "Search Results" タブから消去され、オフになっていると検索結果が追加されます。また **Use RegEx** オプションがオンになっていると、検索文字列に正規表現を使用できます。

フィルタ機能

フィルタ機能を利用することにより、以下のようなフィルタを使用して特定のタイプのエレメントのみを表示することができます。

- ラベルタイプ
  表示するラベルタイプを指定します。
  - All Labels
  - All ASAM
  - All UD
- 階層
  表示する階層を指定します。
- エレメントタイプ
  表示するエレメントタイプを指定します。
  - Inputs
  - Outputs
  - Parameters
  - Measure Elements
  - Pins
- オブジェクトタイプ
  表示するオブジェクトを指定します。
  - Scalar
  - Array
  - Matrix
  - 1D-Table

  - 2D-Table
- データタイプ
  表示するデータタイプを指定します。
  - Logical
  - Discrete
  - Unsigned Discrete
  - Continuous
  - Enumeration

結果を新しいタブに保存する

"Search Results" タブに表示された検索結果を新しいタブに保存することができます。それには "Search Results" タブのタイトル部分を右クリックして **Export Result to New Tab** を選択します。新しいタブの名前は任意に指定できます。

4.4 メインワークスペース

LABCAR-EE の主要な機能は、メインワークスペースの 3 つのタブに実装されています。

- "Instrumentation" タブ（260 ページ）
  実験用インストゥルメント（表示エレメントと操作エレメント）を作成して操作します。
- "Datalogger" タブ（262 ページ）
  データロガーを設定して操作します。
- "Signal Generator" タブ（265 ページ）
  シグナルジェネレータの作成、管理、操作を行います。
- "RT-Plugins" タブ（281 ページ）
  RT-Plugin の管理を行います。

4.4.1 "Instrumentation" タブ

このタブは複数のレイヤで構成され、各レイヤにさまざまなインストゥルメント（表示エレメントや操作エレメント）を割り当てます。

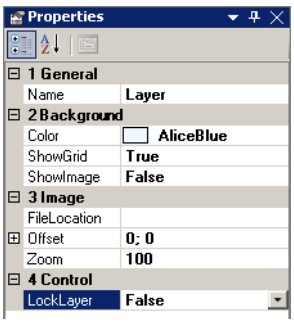
レイヤ

レイヤのタイトルをクリックすると、以下のコマンドを含むショートカットメニューが開きます。

- **Create Layer**
  新しいレイヤを作成します。
- **Import Layer**
  エクスポートされたレイヤコンフィギュレーションをインポートして新しいレイヤを作成します。
- **Export Layer**
  レイヤコンフィギュレーションをファイルに保存します。
- **Rename Layer**
  レイヤの名前を変更します。
- **Delete Layer**
  レイヤを削除します。

レイヤのプロパティ

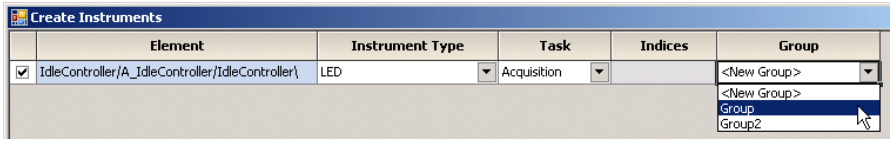
レイヤのタイトル部分をクリックすると、"Properties" ウィンドウにそのレイヤのプロパティが表示され、編集することもできます。



グループ

同じタイプのインストゥルメントは、複数のものを 1 つのグループにまとめて表示することができます。

"Workspace Elements" ウィンドウのショートカットメニューに含まれる **Measure / Calibrate** コマンドを使用して測定変数やパラメータ（適合変数）を表示する際、"Create Instruments" ダイアログボックス内で、そのインストゥルメント用に新しいグループを作成するか、またはインストゥルメントを既存のグループに追加するかを指定することができます。



各レイヤに表示されているグループのタイトル部分を右クリックすると、ショートカットメニューから以下のコマンドを使用することができます。

- **Bring to Front**
  グループを最前面に表示します。
- **Send to Back**
  グループを最後面に表示します。
- **Cut / Copy / Paste**
  レイヤ内のグループの切り取り／コピーを行い、他のレイヤに貼り付けます。
- **Move to Layer**
  グループを別のレイヤに移動します。
- **Hide Group Frame / Show Group Frame**
  グループフレームの表示／非表示を切り替えます。
- **Delete**
  グループを削除します。

グループのプロパティ

グループのタイトル部分をクリックすると、"Properties" ウィンドウにそのグループのプロパティが表示され、編集することもできます。

インストゥルメント

インストゥルメントを実験に追加するには、"Instrumentation" ウィンドウに表示されているインストゥルメントを現在表示されているレイヤにドラッグアンドドロップするか、または "Workspace Elements" ウィンドウのショートカットメニューを使用します。

インストゥルメントのプロパティ

インストゥルメントのタイトル部分をクリックすると、"Properties" ウィンドウにそのグループのプロパティが表示され、編集することもできます。

4.4.2 "Datalogger" タブ

このタブではデータロガーの作成と設定を行います。

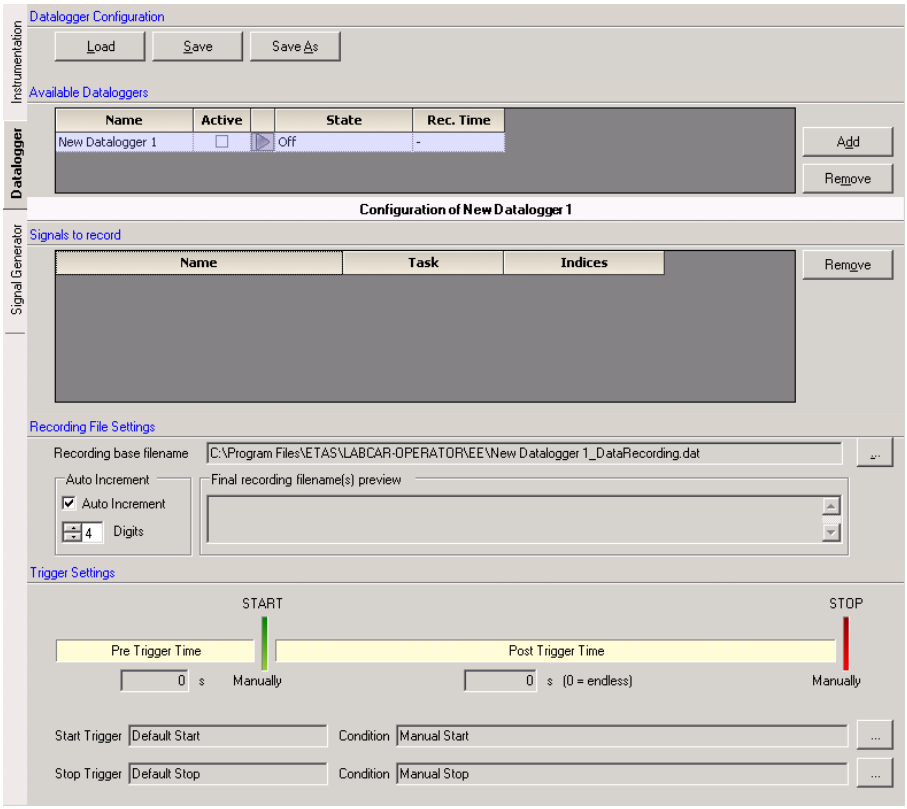


図 4-4 "Datalogger" タブ

以下にこのタブの各フィールドについて説明します。

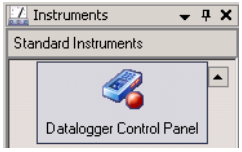
"Datalogger Configuration" フィールド

このフィールドでは、データロガーのコンフィギュレーションを保存したり、すでに保存されているコンフィギュレーションをロードすることができます。このファイル（`.dlc4`）はエクスペリメントエクスプローラの "Datalogger Files" フォルダ内で管理されます。

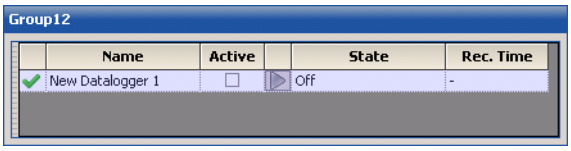
"Available Dataloggers" フィールド

利用できるデータロガーがすべて表示されます。**Add** ボタンで新しいデータロガーを追加したり **Remove** ボタンでデータロガー削除できます。

データロガーの実際の操作は、"Instrumentation" タブ上に "Datalogger Control Panel"（"Standard Instruments" リストに含まれています）を作成して行います。

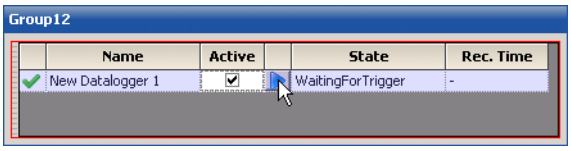


このインストゥルメントでは、"Datalogger" タブで行うものと同じ操作を行えます。



"Signals to record" リストに信号が追加されると、データロガーが有効であることを示す緑のチェックマークが左端の列に表示されます。

実験の実行中は常にデータロガーの "Active" フィールドをアクティブ状態にしておき、青い三角形のマークをクリックすることにより実際の記録が開始されます。

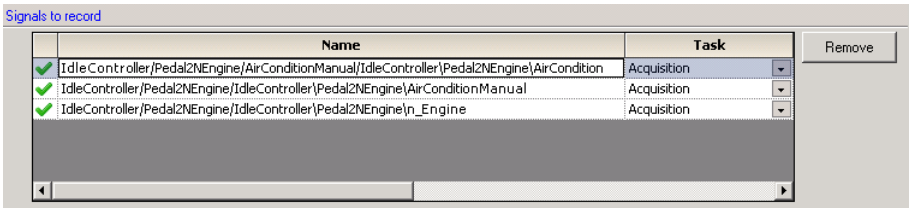


データロガーの稼働中は青い三角形マークが四角形になり、これをクリックするとデータロガーが停止します。

データロガー稼働中は、"State" 列にデータロガーの現在のステートが表示され、さらに "Rec. Time" 列に記録時間が表示されます。

"Signals to record" フィールド

このリスト内にデータロガーで記録する信号が表示されます。ここに信号を追加するには、"Workspace Elements" ウィンドウからドラッグアンドドロップ操作で信号をコピーします。またここでは、記録処理を実行するタスクを各信号ごとに個別に指定できます。



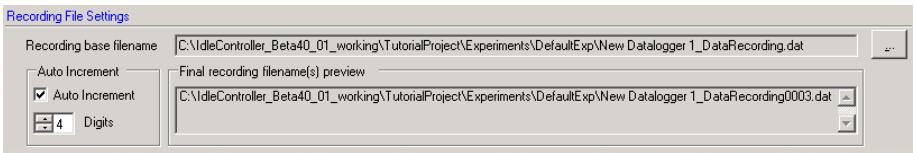
信号を削除するには **Remove** をクリックします。

"Recording File Settings" フィールド

以下の方法でログファイル名を指定します。

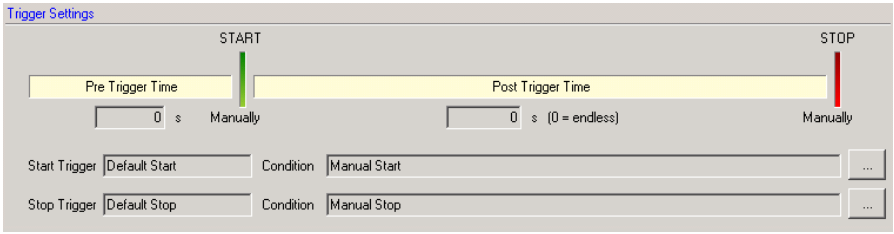
- "Recording base filename" フィールドで、ファイルの名前（ベースネーム）とパスを定義します。
- 記録を行うたびにインクリメントされる番号をファイル名に付加する場合、"Auto Increment" フィールドで設定します。

上記の設定によって決まるファイル名とパスが "Auto Increment" フィールドに表示されます。



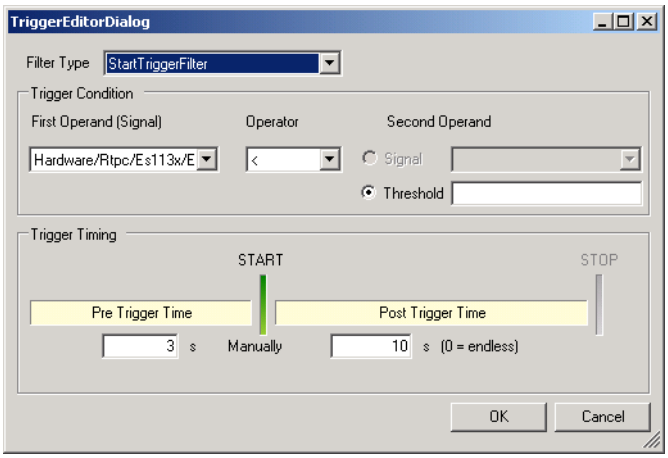
"Trigger Settings" フィールド

このフィールドではデータロガーの記録を自動開始するためのトリガ条件を定義できます。



データロガーの記録開始と終了のためのトリガ条件を定義するには、右側の ボタンをクリックします。

以下のダイアログボックスで各トリガの条件（以下の図は開始トリガ）を設定します。



- **Filter Type**
  マニュアル、またはトリガ条件
- **Trigger Condition**
  - **First Operand**
    トリガ条件として使用する信号
  - **Operator**
    参照値との比較に使用する演算子
  - **Second Operand**
    トリガ信号と比較する信号または値
  - マニュアル、またはトリガ条件
- **Trigger Timing**
  - **Pre-Trigger Time**
    トリガ条件が一致した時点から遡って記録する時間
  - **Post-Trigger Time**
    トリガ条件が一致した後に記録を行う時間（0 は無期限を意味し、マニュアル操作で停止するまで記録を続けます）

4.4.3 "Signal Generator" タブ

このタブでは、シグナルジェネレータの作成、管理、操作を行います

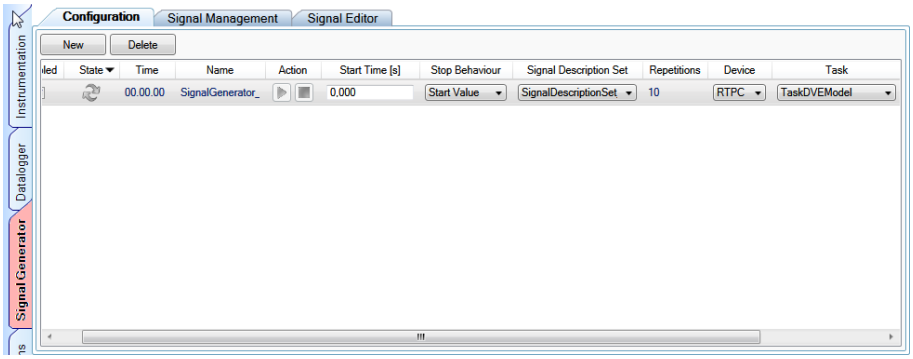


図 4-5 "Signal Generator" タブ

シグナルジェネレータのユーザーインターフェースは、以下の 3 つのタブで構成されています。

- **Configuration**
  このタブでシグナルジェネレータを作成して編集し、操作します。詳しくは 265 ページの「"Configuration" タブ」を参照してください。
- **Signal Management**
  このタブでシグナルジェネレータ用の信号ディスクリプションセットを作成して設定します。詳しくは 268 ページの「"Signal Management" タブ」を参照してください。
- **Signal Editor**
  このタブで信号ディスクリプションの各セグメントを編集します[1] 。詳しくは 276 ページの「"Signal Editor" タブ」を参照してください。

"Configuration" タブ

このタブで、シグナルジェネレータを作成して編集し、操作します。

シグナルジェネレータを作成する：

- **New** をクリックします。
  新しいシグナルジェネレータがリストに追加されます。
  "SignalGenerator"（または "SignalGenerator_1"、"SignalGenerator_2"、...）という名前のシグナルジェネレータが作成されます。

シグナルジェネレータの名前を変更する：

- リスト内のシグナルジェネレータを選択します。
- 以下のいずれかを行います。
  - ショートカットメニューから **Rename** を選択します。
  - **<F2>** を押します。

  名前が編集可能になります。

1. 信号ディスクリプションの編集機能が選択されている場合にのみ表示されます。

シグナルジェネレータを削除する：

- リスト内のシグナルジェネレータを選択して、**Delete** をクリックします。
  シグナルジェネレータが削除されます。

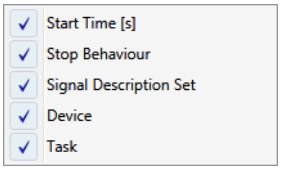
プロパティと操作機能：

リスト内の各シグナルジェネレータには、以下のようなプロパティと操作機能が含まれています。

注記

シグナルジェネレータの操作方法については、280 ページの「シグナルジェネレータを操作する：」を参照してください。

表の見出し部分を右クリックすると、各列の表示／非表示を切り替えるためのショートカットメニューが開きます。



各列の表示順は、ドラッグ & ドロップで任意に変更できます。

- **Enabled**
  シグナルジェネレータのオン／オフを切り替えます。
- **State**
  シグナルジェネレータの状態を示すアイコンで表示されます。
  - **Invalid**
    シグナルジェネレータコンフィギュレーションが無効（信号が割り当てられていない、など）であることを示すアイコンです。
    このアイコンはグレイアウト状態で表示され、このアイコンが表示されると、"Action" 列の **Start** アイコンもグレイアウトされます。
  - **Stopped**
    このアイコンが表示されていると、**Start** アイコンによってシグナルジェネレータを実行することができます。
  - **Running**
    シグナルジェネレータが稼働中の場合、回転するアイコンが表示されます。
  - **Paused**
    シグナルジェネレータが一時停止中の場合、点滅する一時停止アイコンが表示されます。
- **Name**
  シグナルジェネレータの名前
- **Action**
  この列でシグナルジェネレータを操作できます。詳しくは 280 ページの「シグナルジェネレータを操作する：」を参照してください。
  - **Start**
    シグナルジェネレータの実行を開始します。開始トリガが定義されている場合は、定義された条件が満たされてトリガが発生するまで開始されません。

  - **Stop**
    シグナルジェネレータの実行を終了します。
  - **Pause**
    シグナルジェネレータを一時停止します。その後は、モデルは最後に生成された値でスティミュレートされます。**Play** ボタンでジェネレータの実行を再開すると、一時停止した時点から信号生成が続行されます。
- **Start Time**
  シグナルジェネレータが信号出力を開始してからの経過時間（デフォルト : 0.000）
- **Stop Behaviour**
  シグナルジェネレータが一時停止した際の、出力信号の挙動を指定します。
  - **Start Value**（デフォルト）
    信号ディスクリプション内の開始値が出力されます。
  - **0 (zero)**
    値 "0" が出力されます。
- **Signal Description Set**
  信号ディスクリプションセットを選択します。各セットには、シグナルジェネレータが出力する信号の内容が含まれます。"Signal Management" タブ内のすべての信号ディスクリプションセットから選択できます。
- **Repetitions**
  信号ディスクリプションセットの繰り返し回数
- **Device**
  シミュレーションターゲット（RTPC）
- **Task**
  ここで選択されたタスクの実行パターンで、シグナルジェネレータの演算処理が実行されます。

"Signal Management" タブ

このタブで、シグナルジェネレータが生成する信号を定義する信号ディスクリプションセットを作成して編集します。

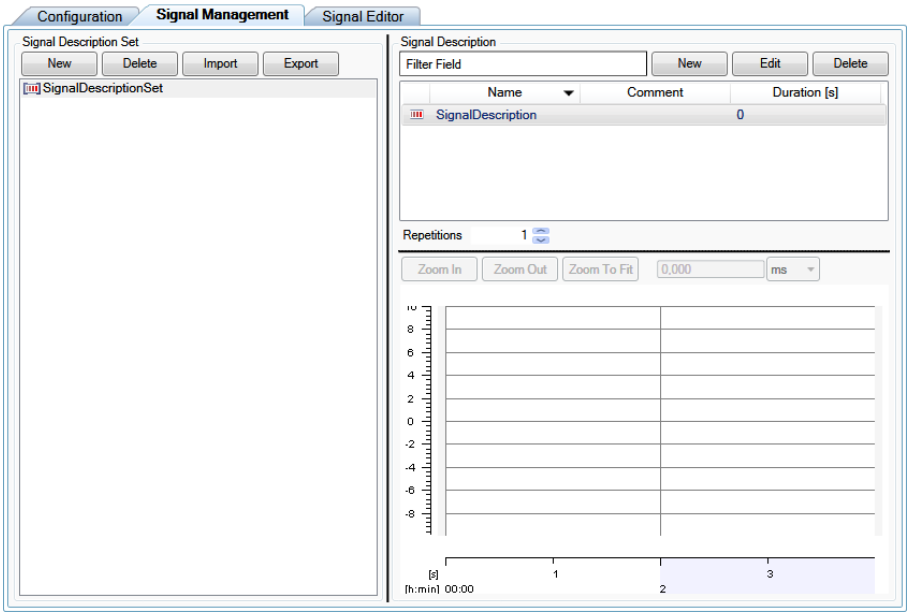


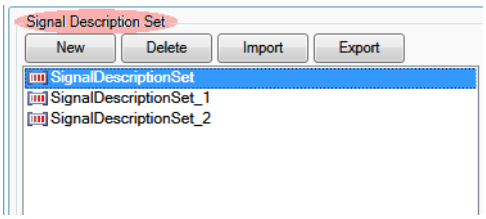
図 4-6 "Signal Management" タブ

このタブは以下の 2 つの領域に分かれます。

- 左側の "Signal Description Set" 領域：実験で使用する信号ディスクリプションセットを管理します。詳細は 268 ページの「"Signal Description Set" 領域」を参照してください。
- 右側の "Signal Description" 領域：各セット内の信号ディスクリプションを管理します。詳細は 273 ページの「"Signal Description" 領域」を参照してください。

"Signal Description Set" 領域：

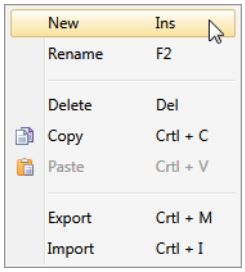
この領域で、現在の実験で使用される信号ディスクリプションセットの作成と管理を行います。



信号ディスクリプションセットを作成する：

- 以下のいずれかを行います。
  - ー **New** をクリックします。

– この領域を右クリックして、ショートカットメニューから **New** を選択します。



– **<Ins>** を押します。

新しい信号ディスクリプションセットが作成されます。

信号ディスクリプションセットを複製する：

- リストから、コピーしたい 信号ディスクリプションセットを選択します
- 以下のいずれかを行います。
  – 選択した信号ディスクリプションセットを右クリックして、ショートカットメニューから **Copy** を選択します。
  – **<Ctrl> + <C>** を押します。

  信号ディスクリプションセットがコピーされます。
- ショートカットメニューから **Paste** を選択するか、または **<Ctrl> + <V>** を押します。

  信号ディスクリプションセットの複製が作成されます。

信号ディスクリプションセットの名前を変更する：

- リストから、名前を変更したい 信号ディスクリプションセットを選択します。
- 以下のいずれかを行います。
  – 選択した信号ディスクリプションセットを右クリックして、ショートカットメニューから **Rename** を選択します。
  – **<F2>** を押します。

  信号ディスクリプションセットの名前が編集モードになります。

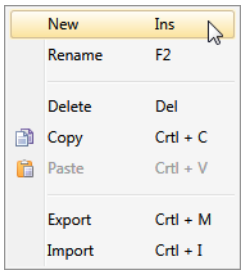
信号ディスクリプションセットを削除する：

- リストから、削除したい 信号ディスクリプションセットを選択して、以下のいずれかを行います。
  – **Delete** をクリックします。
  – ショートカットメニューから **Delete** を選択します。
  – **<Del>** を押します。

  信号ディスクリプションセットが削除されます。

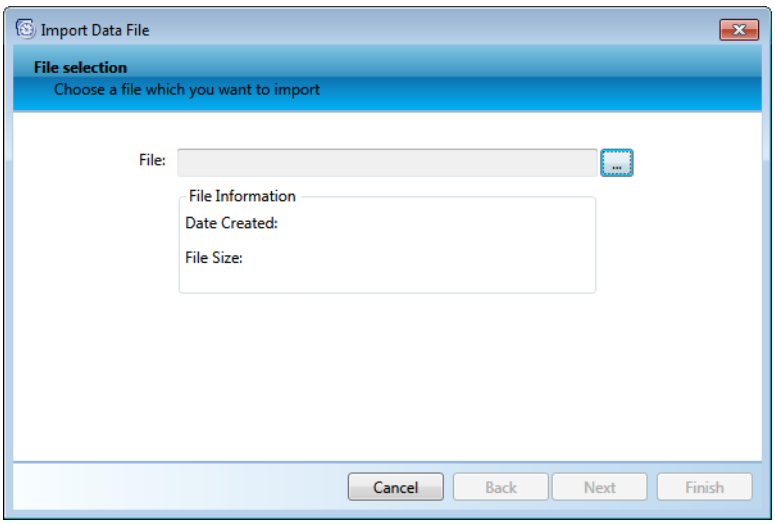
信号ディスクリプションセットをインポートする：

- 以下のいずれかを行います。
  - **Import** をクリックします。
  - この領域のショートカットメニューから **Import** を選択します。



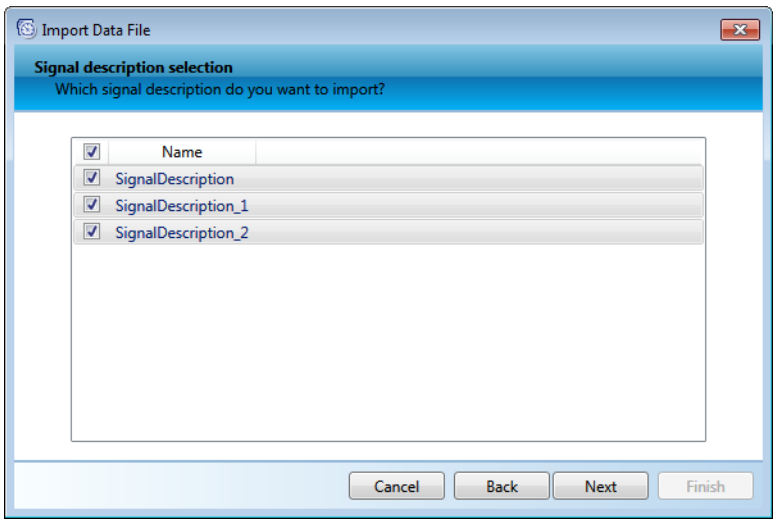
  - **<Ctrl> + <I>** を押します。

"Import Data File" ウィンドウが開きます。



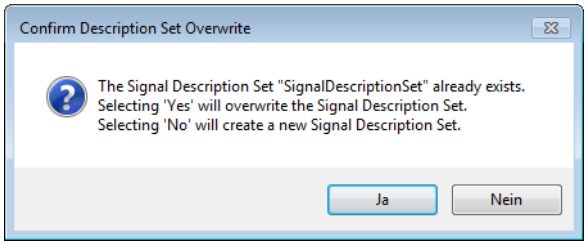
- **...** をクリックしてファイル選択ダイアログボックスを開きます。
- 以下のいずれかのフォーマットのファイルを
- 選択して、**Next** をクリックします。
  - ETAS スティミュラスファイル（`*.esti`）
  - HIL API スティミュラスファイル（`*.sti`）
  - 測定データファイル（`*.dat`）
  - LABCAR スティミュラスファイル（`*.lcs`）
  - MAT ファイル（`*.mat`）

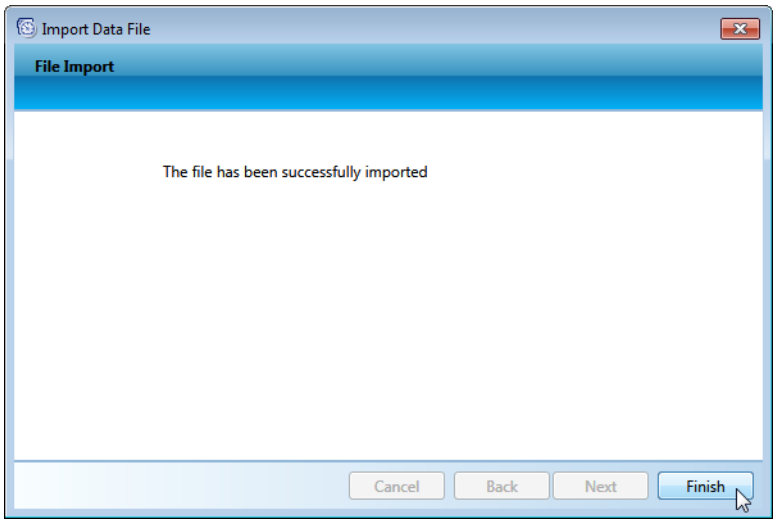
"Import Data File" ウィンドウが開きます。



必要な信号ディスクリプションを選択します。

同名の信号ディスクリプションセットがすでに実験内に存在する場合は、以下のダイアログボックスが開き、既存のセットを上書きするか（**Yes**）、または別名（名前の末尾に連番を付けたもの）のセットを作成するか（**No**）が尋ねられます。





- **Finish** をクリックします。

  信号ディスクリプションセットが作成（または上書き）されます。

MATLAB ファイル（*.mat）をインポートする：

MATLAB の MAT ファイルをインポートする際には、以下の 2 つのフォーマットが使用できます。

- **行列ベースフォーマット**
  時間スケールの異なる信号を保存するためのフォーマットです。各信号は行列として保存され、1 つの MAT ファイルに任意の数の n x 2 行列（n は各信号で異なっていてかまいません）を含めることができます。

  - MATLAB でシグナルジェネレータセットを生成するには、MATLAB オペレーティングウィンドウに以下の例のようなコードを入力します。

    ```
    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;
    ```

    上記の例では、2 つの信号（つまり 2 つの行列）X および Y を含む MATLAB ファイル `MatrixBasedFormat.mat` が生成されます。

    信号 1：

    ```
    0         2
    0.5       3
    1         4
    1.5       5
    2         8
    ```

    信号 2：

    ```
    0         1
    0.5       4
    1         5
    1.5       8
    2         9
    ```

- **インターバルベースフォーマット**（シーケンスフォーマット）
  時間スケールの同じ信号を保存するためのフォーマットです。

  インターバルベクトルにインターバルの間隔（すべて同じ値）が含まれ、データベクトルには各インターバルにおける信号値が含まれます。インターバルベクトルとデータベクトルの長さは同じである必要があり、インターバルベクトルの名前は必ず "Intervals" にします。

- MATLAB オペレーティングウィンドウに以下の例のようなコードを入力します。

```
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;
```

この例では、2 つの信号ベクトル "Signal1_Value" および "Signal2_Value" を含む MATLAB ファイル `IntervalsBasedFormat.mat` が生成されます。

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

信号ディスクリプションセットをエクスポートする：

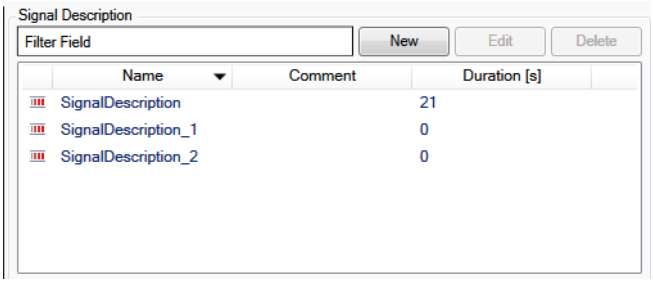
- リストから、エクスポートしたい信号ディスクリプションセットを選択して、以下のいずれかを行います。
  - **Export** をクリックします。
  - この領域のショートカットメニューから **Export** を選択します。
  - **<Ctrl> + <M>** を押します。

  ファイル選択ダイアログボックスが開きます。
- 以下のいずれかのファイルフォーマットを選択し、ディレクトリとファイル名を指定します。
  - ETAS スティミュラスファイル（`*.esti`）
  - HIL API スティミュラスファイル（`*.sti`）
- **保存** をクリックします。

  信号ディスクリプションセットが指定のフォーマットで保存されます。

"Signal Description" 領域：

この領域で、信号ディスクリプションセットに含まれる信号ディスクリプションの作成・編集・管理を行います。



- **Name**
  信号ディスクリプションの名前（277 ページ「Name」参照）
- **Comment**
  コメント（277 ページ「Comment」参照）

- **Duration**
  信号ディスクリプションの全体の時間幅（個々のシーケンスから計算される合計時間）

左上のフィルタフィールド（**Filter Field** フィールド）にフィルタ文字列を入力することにより、信号ディスクリプションを検索することができます。このフィールドが空になっていると。すべての信号ディスクリプションが表示されます。

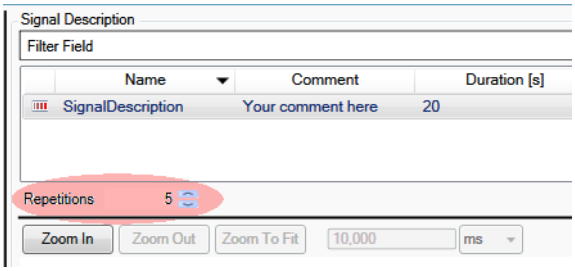
このフィールドに文字を入力していくと、該当する信号ディスクリプションが自動的にインクリメントサーチされ、リストに表示されます。たとえば "abc" と入力すると、"abc" で始まるすべての信号ディスクリプションが表示されます。

検索時には、任意の文字を意味する以下のワイルドカード文字が使用できます。

- アスタリスク（"*"）は、0 文字以上のすべての文字に相当します。たとえば "*_TRK" は、"_TRK" で終わるすべての名前が該当します。
- クエスチョンマーク（"?"）は任意の 1 文字に相当します。たとえば "?TRK" は、任意の 1 文字とそれに続く "TRK" で構成される名前が該当します。

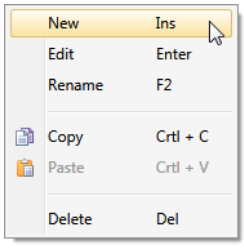
ここでは大文字と小文字は区別されません。

Repetitions フィールドでは、シーケンスを実行する回数を指定できます。



信号ディスクリプションを作成する：

- 以下のいずれかを行います。
  - **New** をクリックします。
  - この領域のショートカットメニューから **New** を選択します。



  - **<Ins>** を押します。

  信号ディスクリプション が作成されます。

信号ディスクリプションを編集する：

- リストから、編集したい信号ディスクリプションを選択して、以下のいずれかを行います。
  - **Edit** をクリックします。
  - この領域のショートカットメニューから **Edit** を選択します。

— **<Enter>** を押します。

信号ディスクリプションに含まれる信号を編集するための "Signal Editor" タブに切り替わります（276 ページの「"Signal Editor" タブ」を参照してください）。

信号ディスクリプションの名前を変更する：

- リストから、名前を変更したい信号ディスクリプションを選択して、以下のいずれかを行います。
  - — この領域のショートカットメニューから **Rename** を選択します。
  - — **<F2>** を押します。

  信号ディスクリプションの名前が編集モードになります。

信号ディスクリプションを複製する：

- リストから、コピーしたい 信号ディスクリプションを選択します
- 以下のいずれかを行います。
  - — 選択した信号ディスクリプションを右クリックして、ショートカットメニューから **Copy** を選択します。
  - — **<Ctrl> + <C>** を押します。

  信号ディスクリプションがコピーされます。
- ショートカットメニューから **Paste** を選択するか、または **<Ctrl> + <V>** を押します。

  信号ディスクリプションの複製が作成されます。

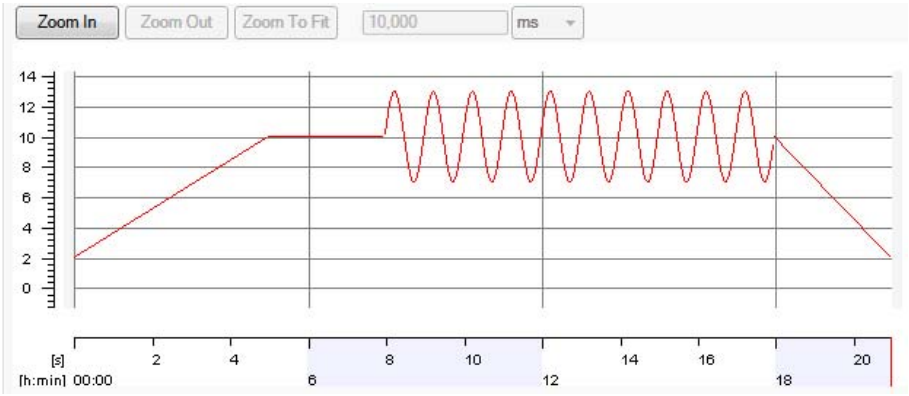
信号ディスクリプションを削除する：

- リストから、削除したい 信号ディスクリプションを選択して、以下のいずれかを行います。
  - — **Delete** をクリックします。
  - — ショートカットメニューから **Delete** を選択します。
  - — **<Del>** を押します。

  信号ディスクリプションが削除されます。

信号ビュー：

信号ビューには、"Signal Editor" タブで定義された信号トレースの概要が表示されます。



各軸をマウスでドラッグすると表示部分が移動し、**<Ctrl>** を押しながらドラッグすると、軸のスケールが変わります。

ここにはリスト内で選択されている信号ディスクリプションが表示されます。**<Ctrl>** を押しながら複数の信号ディスクリプションを選択すると、それぞれの信号トレースが異なる色で表示されます。

また、**Zoom In** / **Zoom Out** をクリックすると時間軸のスケールが増減し、**Zoom To Fit** をクリックすると、信号トレース全体が表示されます。

"Signal Editor" タブ

信号ディスクリプションの信号トレースを編集するには、"Signal Management" タブの "Signal Description" フィールドで信号ディスクリプションを選択して **Edit** をクリックするか、または信号ディスクリプションをダブルクリックして、"Signal Editor" タブを開きます。

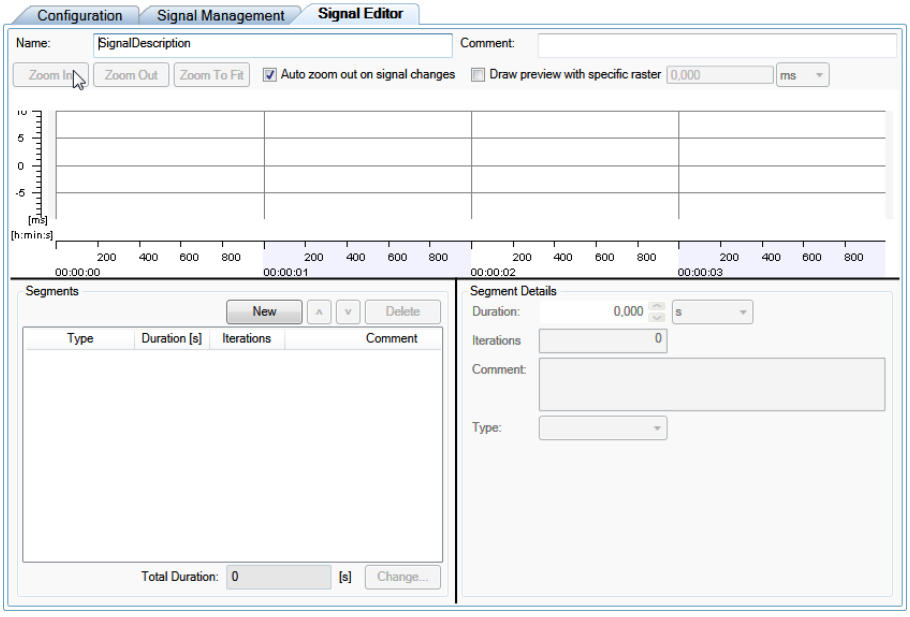


図 4-7 "Signal Editor" タブ

- **Name**
  信号ディスクリプションの名前を変更できます。
- **Comment**
  信号ディスクリプションについてのコメントを入力できます。

信号ビュー :

信号ビューには、このタブで定義された信号セグメントの概要が表示されます。各軸をマウスでドラッグすると表示部分が移動し、**<Ctrl>** を押しながらドラッグすると、軸のスケールが変わります。

また、**Zoom In** / **Zoom Out** をクリックすると時間軸のスケールが増減し、**Zoom To Fit** をクリックすると、信号トレース全体が表示されます。

さらに以下のオプションも用意されています。

- **Auto Zoom out on Signal Changes**
  信号の時間や値の範囲が変更されると各軸の表示範囲も自動的に変更されます。
- **Draw Preview with Specific Raster**
  時間軸のサンプリングレートを指定できます。

"Segments" 領域 :

この領域で、信号ディスクリプションに含まれる信号セグメントの作成、編集、管理を行います。

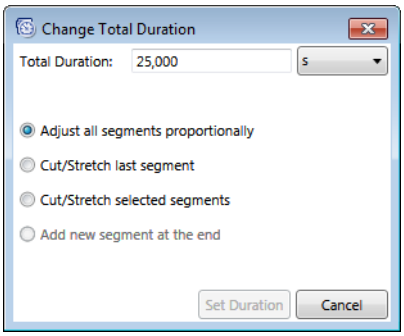
- **Type**
  "Segment Detail" 領域内の "Types" フィールドで選択された、セグメントのタイプが表示されます。
- **Duration**
  "Segment Detail" 領域内の "Duration" フィールドで設定された、セグメントの時間幅が表示されます。
- **Iterations**
  "Segment Detail" 領域内の "Iterations" フィールドで設定された、セグメントの繰り返し回数が表示されます。
- **Comment**
  "Segment Detail" 領域内の "Comment" フィールドに入力されたコメントが表示されます。

セグメントを作成するには **New** をクリックし、セグメントを削除するには、削除したいセグメントを選択して **Delete** をクリックします。

セグメントの位置を移動するには、移動したいセグメントを選択して上下の矢印ボタンをクリックします。

"Total Duration" フィールドには、全セグメントの合計時間が表示されます。
Change をクリックすると、この合計時間を任意のモードで変更することができます。

以下の 4 種類のオプションのいずれかを選択してから、"Total Duration" の値を変更します。



- **Adjust all signals proportionally**
  現在の比率を保持したまま全セグメントの時間を変更します。
- **Cut/Stretch last segment**
  最後のセグメントの時間のみを変更します。
- **Cut/Stretch selected segments**
  選択されたセグメントの時間のみを変更します。
- **Add new segment at the end**
  現在の合計時間より長い時間が入力されると、新しいセグメントが末尾に追加されます。

各セグメントの詳細な編集は "Segment Details" 領域で行います。

"Segment Details" 領域 :
この領域では、"Segments" 領域で選択されたセグメントについて詳細な設定を行います。

- **Duration**
  セグメントの時間幅
- **Iteration**
  セグメントの繰り返し回数
- **Comment**
  セグメントについてのコメント
- **Type**
  信号値のタイプ
  - **Constant**
    定数値（"Value"）
  - **Data**
    ファイル（測定データファイル (*.dat) または MAT ファイル (*.mat)）から読み込んだ値
  - **Pulse**
    PWM 信号 ― 周期（"Period"）、増幅（"Period"）、デューティサイクル（"Duty Cycle"）、オフセット（"Offset"）で定義されます。
  - **Ramp**
    リニアに上昇／下降する値 ― 開始値（"Start Value"）、時間幅（"Duration"）、終了値（"Stop Value"）、傾斜（"Slope"）で定義されます。

- **Random**

  乱数値 ー 増幅値（"Amplitude"）とオフセット（"Offset"）を伴います。

- **Saw**

  鋸波 ー 増幅（"Amplitude"）、周期（"Period"）、オフセット（"Offset"）で定義されます。

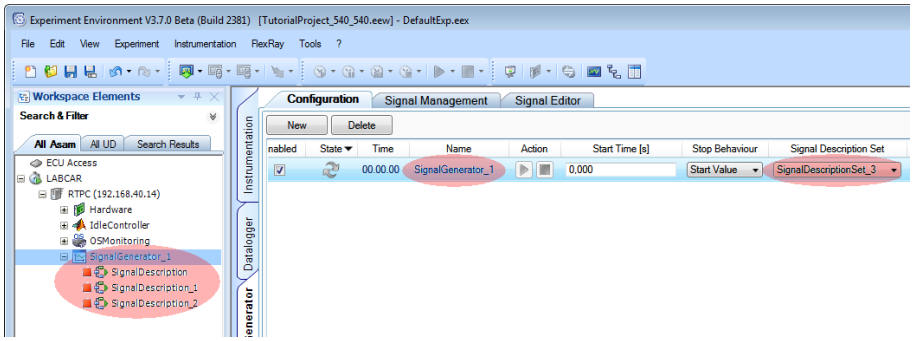
- **Sine**

  正弦波 ー 増幅（"Amplitude"）、周期（"Period"）、移相（"Phase"）、オフセット（"Offset"）で定義されます。

シグナルジェネレータの使用

シグナルジェネレータを使用するには、実験を設定する際に、信号ディスクリプションを含む信号ディスクリプションセットをシグナルジェネレータに割り当てます。

割り当てられた信号ディスクリプションは、"Workspace Elements" ウィンドウのエレメントリスト内の各シグナルジェネレータの下に、モジュール出力として表示され、信号リスト内の他の入力に接続することができます。



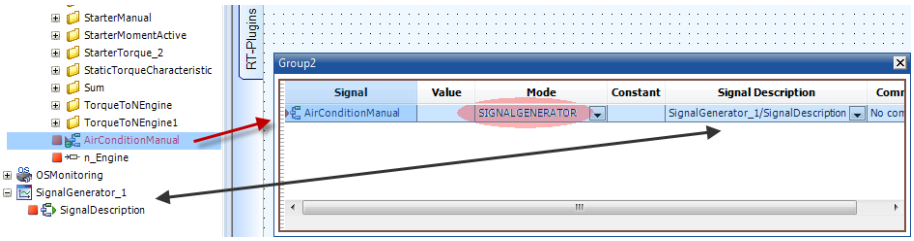
注記

シグナルジェネレータとその信号ディスクリプションについては、完全に定義されたもののみがエレメントリストに表示されます。つまり、信号ディスクリプションが定義された信号ディスクリプションセットがシグナルジェネレータに割り当てられ、さらにシグナルジェネレータにタスクが割り当てられている必要があります。

シグナルジェネレータを接続する：

- "Instrumentation" タブの任意のレイヤ上に信号リストを作成します。
- "Workspace Elements" ウィンドウで、シグナルジェネレータでスティミュレートしたいモジュール入力を選択し、信号リストにドラッグします。
- "Mode" 列で、"Signal Generator" を選択します。

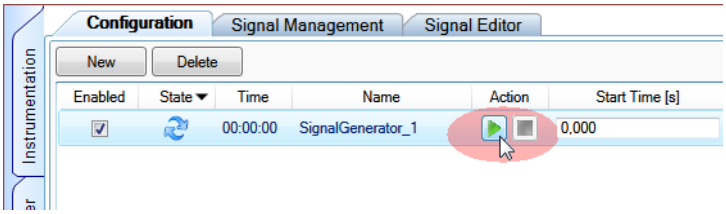
- "Signal Description" 列で、モジュール入力へのスティミュレーションに使用したいシグナルジェネレータと信号ディスクリプションを選択します。



- 実験を保存します。

シグナルジェネレータを操作する：

- Real-Time PC に実験をダウンロードします。
- 実験の実行を開始します。
- "Signal Generator" タブに切り替えます。
- 使用するシグナルジェネレータを選択して、**START** ボタンをクリックします。



実行中のシグナルジェネレータには実行中を示すアイコンが表示されます。

"Time" 列には、シグナルジェネレータの最後の実行開始時からの経過時間が表示されます。

- **Pause** をクリックすると、シグナルジェネレータが一時停止します。一時停止中はシグナルジェネレータの最後の出力値が出力されます。
- **Stop and reset signal generator** をクリックすると、シグナルジェネレータの実行が終了し、出力値は "Stop Behaviour" 列で定義された値にリセットされます。

4.4.4 "RT-Plugins" タブ

このタブでは RT プラグインを管理します。このタブについては、207 ページの「"RT Plugins" タブ」の説明をご覧ください。

4.4.5 "Instruments" ウィンドウ

"Instruments" ウィンドウには、実験において各種測定や操作を行うためさまざまなインストゥルメントが用意されています。

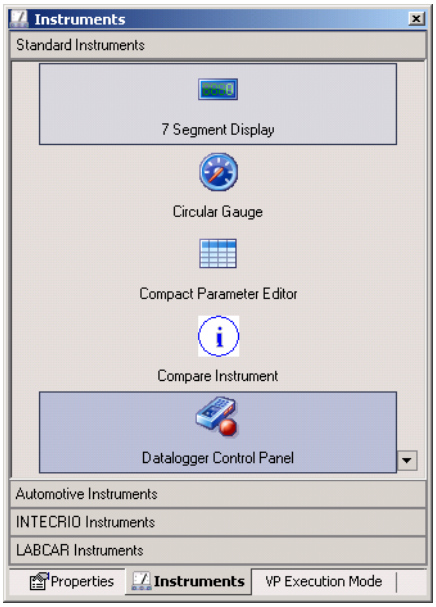


図 4-8 "Instruments" ウィンドウ

各インストゥルメントは以下のグループに分類されています。

- "Standard Instruments"（標準インストゥルメント）
- "Automotive Instruments"（車両インストゥルメント）
- "LABCAR Instruments"（LABCAR インストゥルメント）

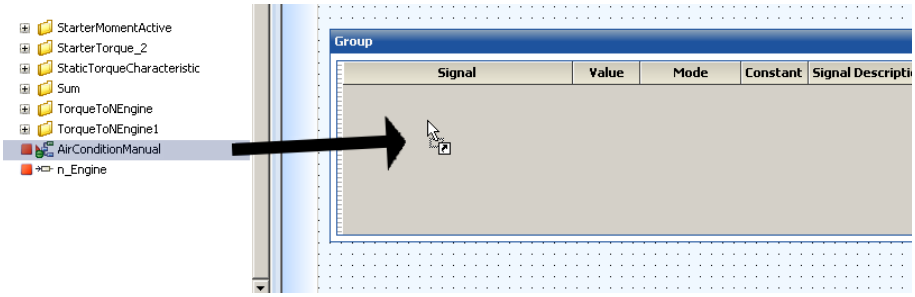
インストゥルメントを実験で使用するには、任意のインストゥルメントを "Instrumentation" ウィンドウの各レイヤにドラッグアンドドロップします。そのインストゥルメントに測定変数やパラメータを割り当てるには、"Workspace Elements" ウィンドウ内の測定変数またはパラメータを選択し、ショートカットメニューから **Measure/Calibrate** コマンドを選択するか、またはドラッグアンドドロップを行います。

4.4.6 信号リストへの信号の追加

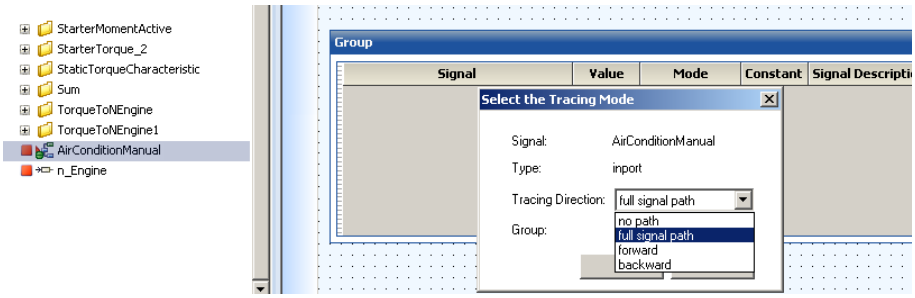
LABCAR インストゥルメントに含まれる信号リスト（"Signal List" インストゥルメント）を作成した後は、以下の 2 とおりの方法で信号リストに信号を追加することができます。

信号をトレース情報とともに追加する：

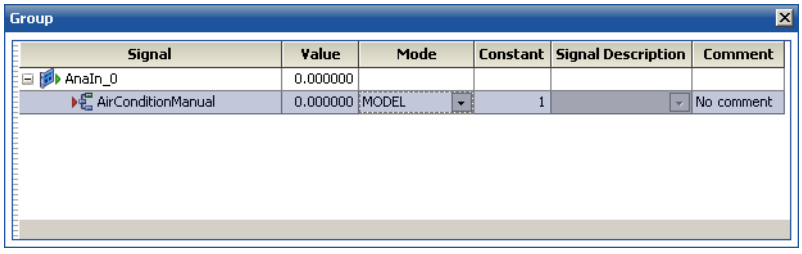
- 追加したい信号を信号リストまでドラッグします。



- "Select the Tracing Mode" ダイアログボックスの "Tracing Direction" フィールドで、信号をトレースしたい方向（"full signal path"、"forward"、"backward"）を選択します。



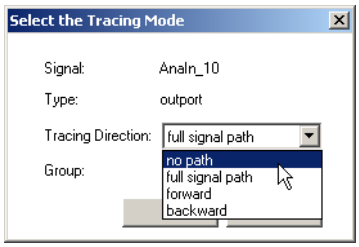
追加した信号が、選択された方向に接続されているすべての信号とともに信号リストに表示されます。



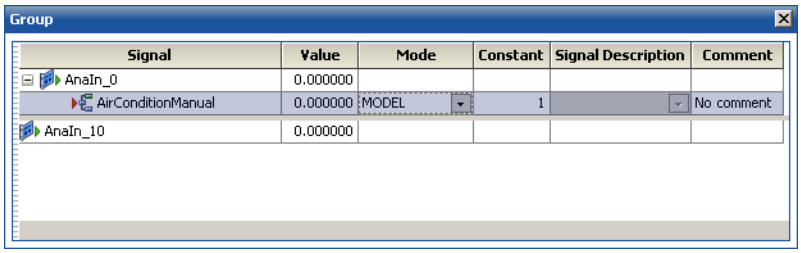
この表示では、入力をモデルから切り離して定数値を代入することもできます（"Mode" 列）。

信号を単独で追加する：

- 信号リストに信号だけを表示し、トレース情報（信号パス）を表示しない場合は、"Select the Tracing Mode" ダイアログボックスの "Tracing Direction" フィールドで "no path" を選択します。



信号リストに信号が単独で表示されます。



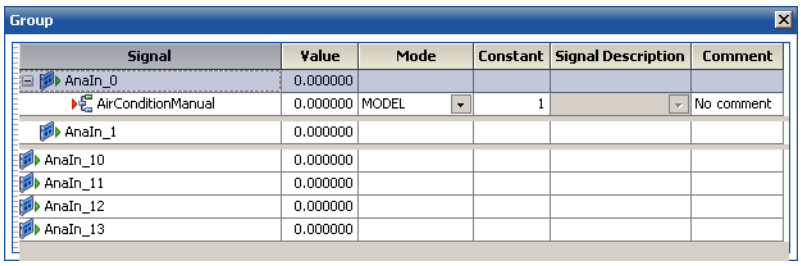
信号リストに信号を追加する：

既存の信号リストに新しい信号をドラッグして追加する際、ドラッグ先の位置に応じてその信号のトレースモードが変わります。

- 追加したい信号を、信号パスとともに表示されている信号の行までドラッグします。

  "Select the Tracing Mode" ダイアログボックスが開くので、トレースタイプを指定します。
- 追加したい信号を、単独で表示されている信号の行までドラッグします。

  ドラッグされた信号が単独でリストに追加されます。



信号リスト内の信号表示には以下のルールも適用されます。

- 複数の信号を "Workspace Elements" ウィンドウで選択して信号リストにドラッグすると、それらは単独の信号として追加されます。
- 信号リスト内の信号のショートカットメニューから **Trace Signal** を選択して "Select the Tracing Mode" ダイアログボックスを開き、プロパティを適切に選択することにより、その信号の表示モード（トレースあり／なし）を変更したり、新しい信号リストにコピーしたりすることができます。

- リスト内に単独で表示されている信号は、他の信号リストにドラッグしてコピーすることができます。トレースモードはコピー先のリスト内で指定できます。

4.5 スクリプトレコーダ

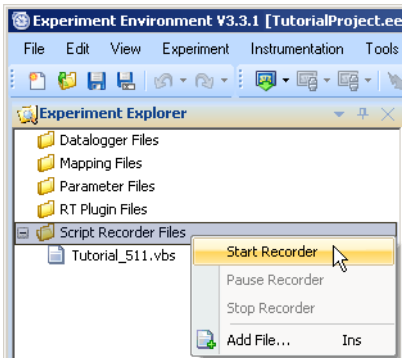
実験を実行する際には、GUI から各種操作（パラメータ値の変更、パラメータファイルのロードなど）が頻繁に繰り返し行われますが、LABCAR-OPERATOR V5.4.0 のスクリプトレコーダ機能を利用することにより、このような繰り返し操作を自動化することができます。

操作を記録したマクロは Visual Basic Script（*.vbs）として保存されます。1 つのプロジェクトに複数のスクリプトを割り当てることができ、1 つのスクリプトを複数のプロジェクトで使用することもできます。

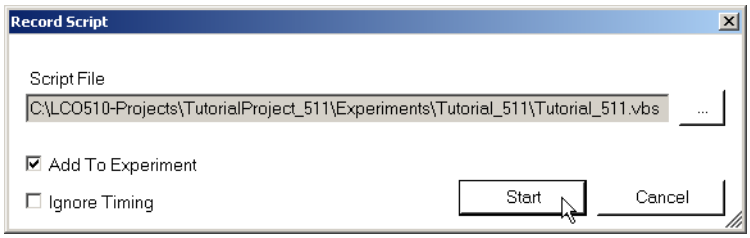
LABCAR-OPERATOR V5.4.0 スクリプトレコーダでは、LABCAR-EE スクリプティング API でサポートされるすべてのユーザーアクションを記録することができます。

スクリプトを記録する：

- エクスペリメントエクスプローラ内の "Script Recorder Files" フォルダを右クリックします。
- ショートカットメニューから **Start Recorder** を選択します。



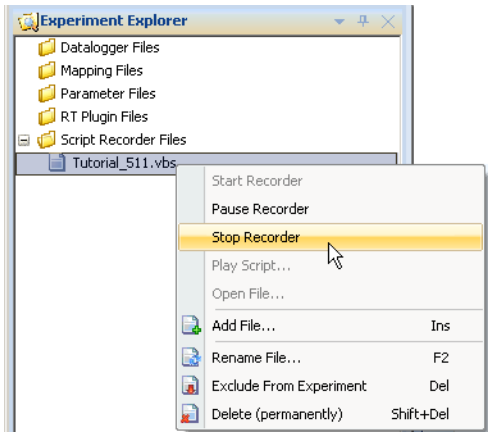
- 以下のダイアログボックスからファイルを選択します。すでにプロジェクトに割り当てられているスクリプトファイルでもかまいません。



  - そのファイルを実験（"Script Recorder Files" ディレクトリ内）に追加する場合は "Add to Experiment" をオンにします。
  - "Ignore Timing" オプションをオンにすると、各ユーザーアクション間の経過時間は記録されません。
- **Start** をクリックします。

記録が始まります。スクリプトレコーダのステータスと記録開始からの経過時間が実験環境のメインウィンドウの下端に表示されます。

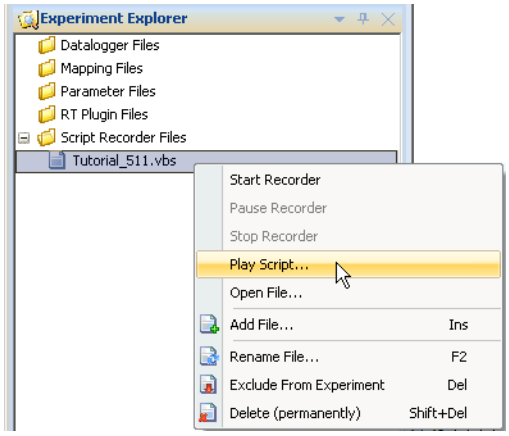
- 記録を終了するには、ショートカットメニューから **Stop Recorder** を選択します。

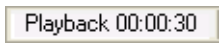


記録が終了し、スクリプトが保存されます。

スクリプトを実行（再生）する：

- 実行するスクリプトをエクスペリメントエクスプローラから選択します。
- ショートカットメニューから **Play Script** を選択します。





スクリプトが実行されます。

スクリプトを管理する：

各スクリプトのショートカットメニューから、以下のような操作が行えます。

- スクリプトをプロジェクトに追加するには、**Add File** を選択します。
- スクリプト名を変更するには、**Rename File** を選択します。
- 実験に対するスクリプトファイルの割り当てを解除するには、**Exclude From Experiment** を選択します。
- スクリプトファイルを削除するには、**Delete (permanently)** を選択します。

4.6 パラメータの使用法

本項では、LABCAR-OPERATOR プロジェクトのパラメータを実験環境 LABCAR-EE で使用する方法について説明します。

以下の情報が含まれています。

- LABCAR-EE でのパラメータ管理（287 ページ）
- ショートカットメニュー（289 ページ）
- 実験中に行うパラメータ値の変更（290 ページ）
- パラメータセットの保存（293 ページ）
- エクスペリメントエクスプローラで行うパラメータファイル管理（294 ページ）
- バリアントとパラメータコンビネーション（297 ページ）
- マッピングファイル（301 ページ）
- マッピングファイルの作成と管理（301 ページ）

4.6.1 LABCAR-EE でのパラメータ管理

LABCAR-IP でプロジェクトに追加される各モジュールには、一連の新しいパラメータが含まれています。LABCAR-EE では、これらのパラメータとそれに割り当てられた値（以下「パラメータ値」と呼びます）のアクティブ化、変更、保存、エクスポートを行えます。

LABCAR-EE では、1 つのプロジェクトのパラメータと測定変数が "Workspace Elements" ウィンドウに一覧表示されます。このウィンドウの詳細については、254 ページの「"Workspace Elements" ウィンドウ」を参照してください。

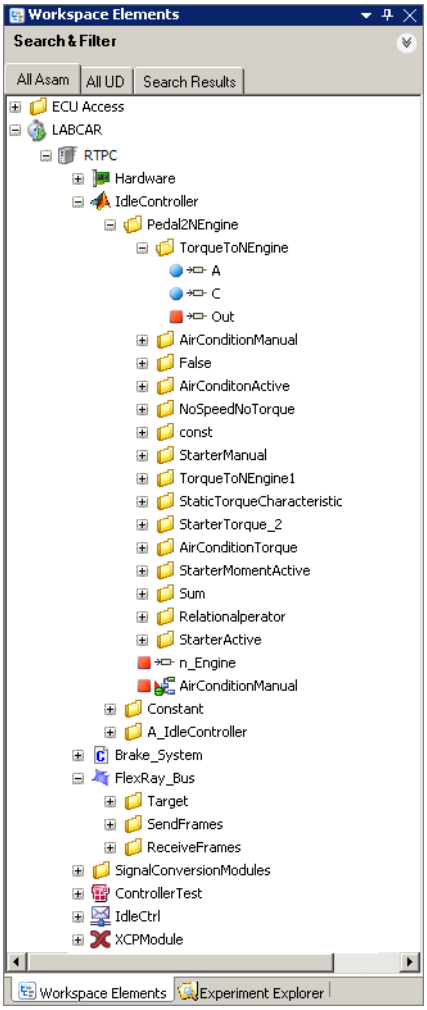


図 4-9 "Workspace Elements" ウィンドウ

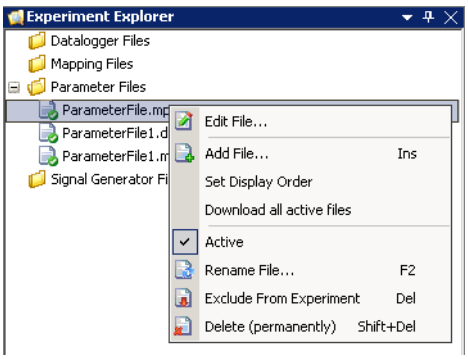
パラメータは "All Asam" タブに表示されます（他の 2 つのタブについては 255 ページの「"Workspace Elements" ウィンドウの各タブ」を参照してください）。

このタブには、すべてのモジュール、およびそのパラメータと測定変数が "LABCAR" フォルダ内に表示されます。また "ECU Access" フォルダには ECU ピンの定義や、INCA 適合コネクタにより並行して行う INCA 実験のパラメータと測定変数の定義が含まれています。

各モジュールには、モジュールタイプを表す専用のアイコンが付記されます。これらは、ショートカットメニューを使用してモジュールフォルダやその内容に対して処理を実行するためのものです（255 ページの「"Workspace Elements" ウィンドウの機能」を参照してください）。

パラメータファイルの管理

パラメータ、およびそれに割り当てられた値が格納されているファイルは "Experiment Explorer" ウィンドウで管理されます。各機能はショートカットメニューから実行します。



ファイルタイプには以下のものがあります。

- `*.mpa` と `*.dcm`

  パラメータとその値が定義されているファイル
- `*.olc`

  信号経路とカーブが定義されているファイル
- `*.pac`

  パラメータバリアント（パラメータコンビネーション）：複数の `*.mpa`、`*.dcm`、`*.olc` ファイルを集めたもの

詳細については、294 ページの「エクスペリメントエクスプローラで行うパラメータファイル管理」を参照してください。

4.6.2 ショートカットメニュー

"All Asam" タブのショートカットメニューについての一般情報は 255 ページの「"Workspace Elements" ウィンドウの機能」に示されているので、ここではパラメータに関するメニューコマンドの使用法に絞って説明します。

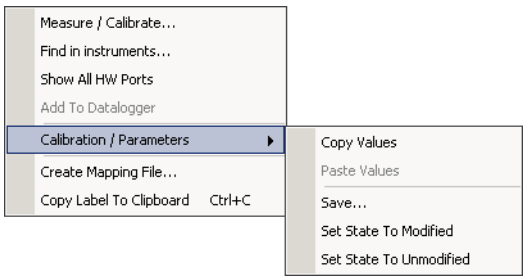


図 4-10 パラメータ用ショートカットメニュー（アクティブな実験について）

このショートカットメニューには以下の機能に対応するコマンドが含まれています。

- 各パラメータの値を変更するためのインストゥルメンテーションの作成（**Measure / Calibrate**）

  290 ページの「実験中に行うパラメータ値の変更」を参照してください。
- パラメータセットの保存（**Calibration / Parameters → Save**）

  293 ページの「パラメータセットの保存」を参照してください。

- パラメータセット（*.mpa）全体の編集（295 ページの「パラメータファイルを編集する：」を参照してください。）

このショートカットメニューでは、個々のパラメータやパラメータセット全体を変更するだけでなく、以下のアクションも実行できます。

- 各パラメータ値のコピー＆ペースト（**Calibration / Parameters → Copy/Paste**）
- パラメータ／測定変数の変更ステータスの編集（**Calibration / Parameters → Set State To Modified/Unmodified**）
- マッピングファイルの作成（**Create Mapping File**）（301 ページの「マッピングファイル」を参照してください。）
- ラベルをクリップボードにコピー（**Copy Label To Clipboard**）
- ハードウェアパラメータをすべて表示（**Show all HW Ports**）
- 特定のパラメータ／測定変数が使用されているインストゥルメントの検索（**Find in instruments**）

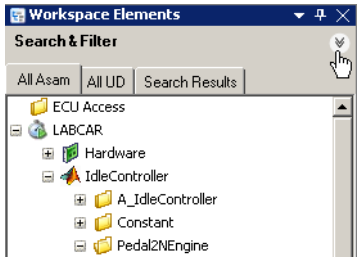
4.6.3 実験中に行うパラメータ値の変更

パラメータ値の表示と変更は、LABCAR-EE のディスプレイインストゥルメント（"Instrumentation" ウィンドウ）で行われます。通常、値の変更には「エディットボックス」（下図）が使用されます。

実験には非常に多くのパラメータと測定変数が使用される可能性がありますが、検索機能によって目的のアイテムを素早く見つけ出すことができます。

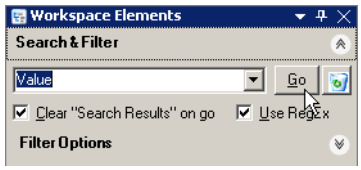
パラメータを検索する：

- "Workspace Elements" ウィンドウ内の下記のアイコンをクリックします。

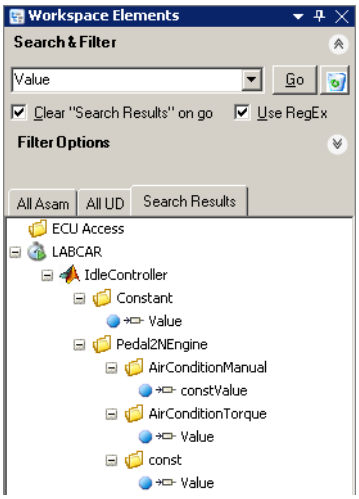


"Search & Filter" フィールドが展開されます。

- 検索文字列を入力して **Go** をクリックします。



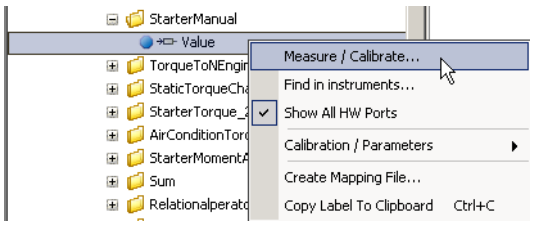
検索結果を表示する "Search Results" タブがアクティブになります。



フィルタ設定により（"Filter Options"）、検索結果をさらに絞り込むことができます。

インストゥルメントを作成する：

- 表示／編集したいパラメータを右クリックします。

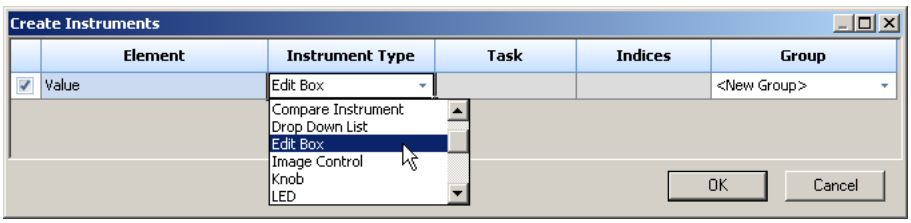


注記

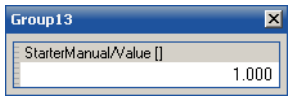
2 個以上のパラメータを選択するには、**<Shift>** または **<Ctrl>** を使用してください。また、フォルダ全体を選択することもできます。

- ショートカットメニューから **Measure / Calibrate** を選択します。

  "Create Instruments" ダイアログボックスが開きます。

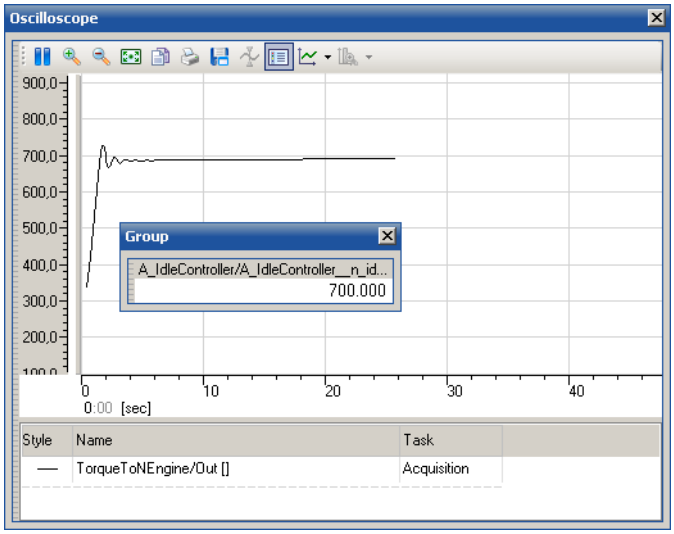


- "Instrument Type" 列で "Edit Box" を選択します。
  インストゥルメントが作成され、パラメータのカレント値が表示されます。



実験においてパラメータの値を変更する：

- 実験を開始します。
- 値を変更したいパラメータが割り当てられているインストゥルメントをクリックします。
  パラメータが編集可能な状態になります。

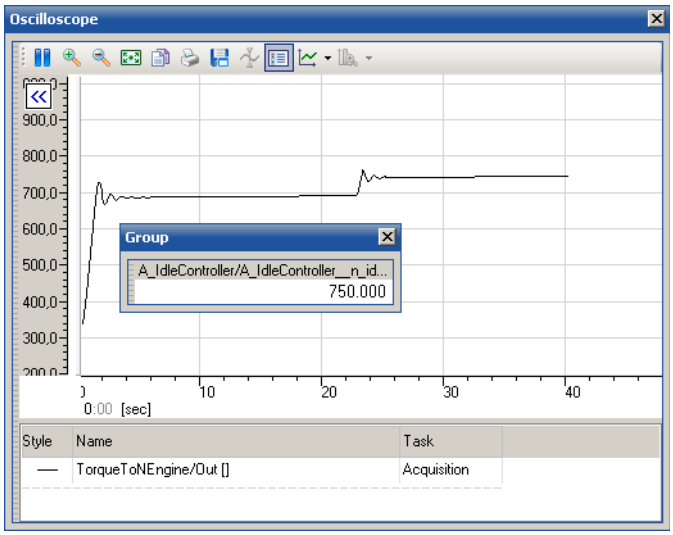


- 新しい値を入力します。
- **<Enter>** を押します。

または

- インストゥルメントの外側をクリックします。

- 現在のシミュレーションにおけるパラメータ値が変更されます。



変更ステータスの表示とリセット

パラメータの初期値を含むコードをターゲットにダウンロードした後、実行を開始してパラメータ値を変更すると、その値は赤色で表示されます。

この赤色表示は、ショートカットメニューの **Calibration / Parameters → Set State To Modified** または **Calibration / Parameters → Set State To Unmodified** で、任意にセット／リセットできます。

4.6.4 パラメータセットの保存

実験中に変更したパラメータ値を「パラメータセット」として保存するには、"Workspace Elements" ウィンドウから以下のいずれかを選択して 1 つのファイルに保存します。

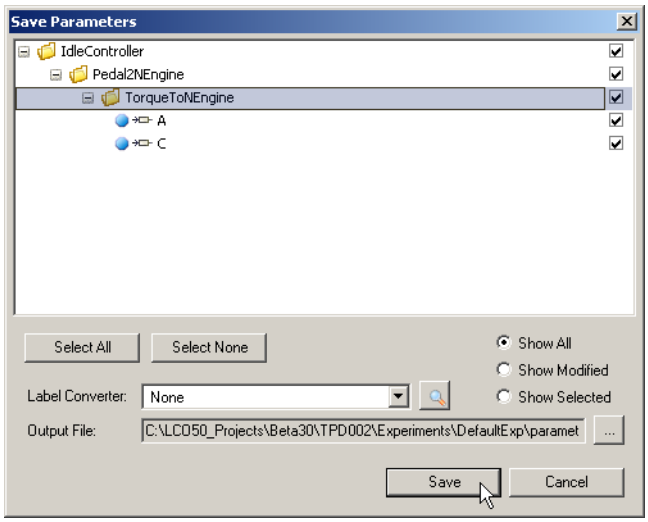
- 1 つのフォルダ内の 1 つまたは複数のパラメータ
- 複数のフォルダ
- リスト全体

パラメータセットを保存する；

- 値をファイルに保存したいパラメータ／フォルダ／モジュールを選択します。

- 右クリックして **Calibration / Parameters → Save** を選択します。

  "Save Parameters" ダイアログボックスが開きます。



選択された階層／フォルダのすべてのパラメータが表示され、選択状態になります。**Select All** または **Select None** ボタンで、この選択状態を切り替えることができます。また **Show All**、**Show Modified**、**Show Selected** ボタンで、エレメントリストをフィルタリングすることもできます。

パラメータファイルを選択するには "Output File" の **...** をクリックしてファイル選択ダイアログボックスを開き、ファイルの名前とフォーマット（`*.mpa`、`*.dcm`、`*.cdfx`）を指定します。

このファイル選択ダイアログボックスでは、以下の 2 つのオプションも使用できます。

- Add To Experiment

  選択されたファイルが実験に自動的に追加されます。つまり、選択されたファイルがエクスペリメントエクスプローラの "Parameter Files" フォルダに表示され、利用可能になります。

- Activate File

  選択されたファイルがアクティブになります（296 ページの「パラメータファイルをアクティブにする：」を参照してください）。

4.6.5 エクスペリメントエクスプローラで行うパラメータファイル管理

エクスペリメントエクスプローラの "Parameter Files" フォルダには、実験に属しているすべてのパラメータファイルがリストアップされます。

以下の各項で、ショートカットメニューを使用したアクションの実行手順を説明します。

- パラメータファイルを編集する：（295 ページ）
- パラメータファイルを追加する：（295 ページ）
- パラメータファイルの順序を指定する：（296 ページ）
- パラメータを実験にロードする：（296 ページ）
- パラメータファイルをアクティブにする：（296 ページ）
- パラメータファイル名を変更する：（297 ページ）
- パラメータファイルを実験から除外する：（297 ページ）

- パラメータファイルを削除する：（297 ページ）

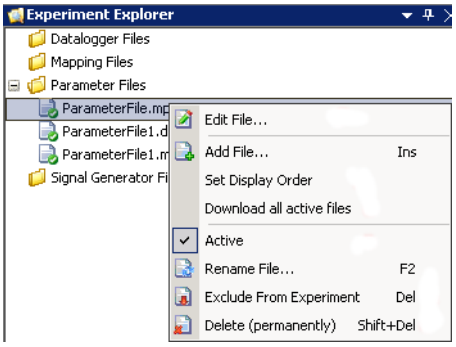


図 4-11 "Parameter Files" フォルダのショートカットメニュー

ファイルに保存されているパラメータ（293 ページの「パラメータセットの保存」を参照してください）を、ここで開いて編集することができます。

パラメータファイルを編集する：

- **Edit File** を選択するか、またはファイルをダブルクリックします。

  *.mpa ファイルの場合は、LABCAR-PA 1.0 - Parameterization Assistant が開き、選択されたパラメータファイルがロードされます。

パラメータファイルを追加する：

- パラメータファイルを実験に追加するには、**Add File** を選択します。
- ファイル選択ダイアログボックスが開きます。
- 追加するファイルを選択します。複数のファイルを選択するには **<Shift>** または **<Ctrl>** を使用します。
- **Open** をクリックします。

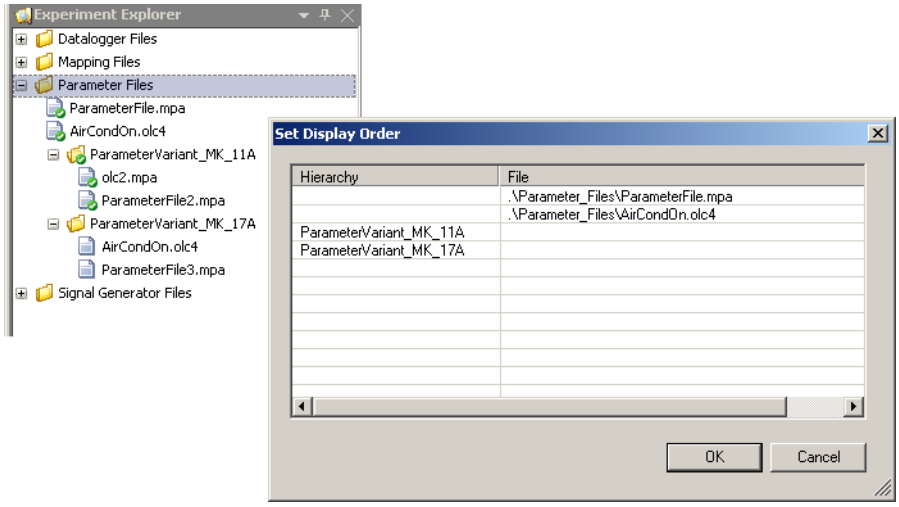
  選択されたパラメータファイルが "Parameter Files" フォルダに追加されます。

シミュレーションコードがロードされるときには、パラメータファイルは "Parameter Files" フォルダ内に表示されている順序で実験ターゲットにロードされます。

パラメータファイルの順序を指定する：

- この順序を変更するには **Set Display Order** を選択します。

  "Set Display Order" ダイアログボックスが開きます。



  このダイアログボックスでは、バリアント（パラメータコンビネーション（*.pac）とも呼ばれます。297 ページの「バリアントとパラメータコンビネーション」を参照してください）のフォルダが "Hierarchy" 列にリストアップされ、トップレベルのパラメータファイルが "File" 列にリストアップされます。

- ドラッグ＆ドロップ操作でフォルダまたはファイルを上下に移動します。
- **OK** をクリックします。

  エクスペリメントエクスプローラ内のフォルダ／ファイルの順序が、指定されたとおりに変更されます。

パラメータを実験にロードする：

- アクティブになっているすべてのパラメータファイルを実験にダウンロードするには、**Download all active files** を選択します。

パラメータファイルをアクティブにする：

- パラメータファイルをアクティブにするには、ショートカットメニューの **Active** を選択します。

  このファイルがアクティブになり、ファイル内のパラメータ値がターゲットに自動的にロードされます。

- この処理を取り消すには、もう一度 **Active** を選択します。

パラメータファイル名を変更する：

- パラメータファイル名を変更するには、**Rename File** を選択します。

  ダイアログボックスが開くので、新しいファイル名を入力します。

パラメータファイルを実験から除外する：

- パラメータファイルを実験から除外するには、**Exclude From Experiment** を選択します。

  そのパラメータファイルの実験への割当てが解除され、エクスペリメントエクスプローラから消去されます。

注記

除外されたファイルは削除されません。そのファイルのプロジェクトへの割り当てが解除されるだけです。

パラメータファイルを削除する：

- パラメータファイルを完全に削除するには、**Delete (permanently)** を選択します。

"Parameter Files" フォルダのショートカットメニューから、次項で説明するパラメータバリアントを作成することもできます。

4.6.6 バリアントとパラメータコンビネーション

一般的な ECU テストにおいては、細部が異なる複数の ECU をテストする必要が生じる場合があります。その場合、ソフトウェアバージョンも ECU ごとに異なります。なおここでは、そのような細部の異なる ECU またはソフトウェアを、それぞれ「バリアント」（= バリエーション）と呼びます。

このようなテストを行うには、ECU バリアントおよびソフトウェアバリアントごとにそれぞれ「テストプロジェクトバリアント」を用意する必要があります。この構成を図 4-12 に示します。

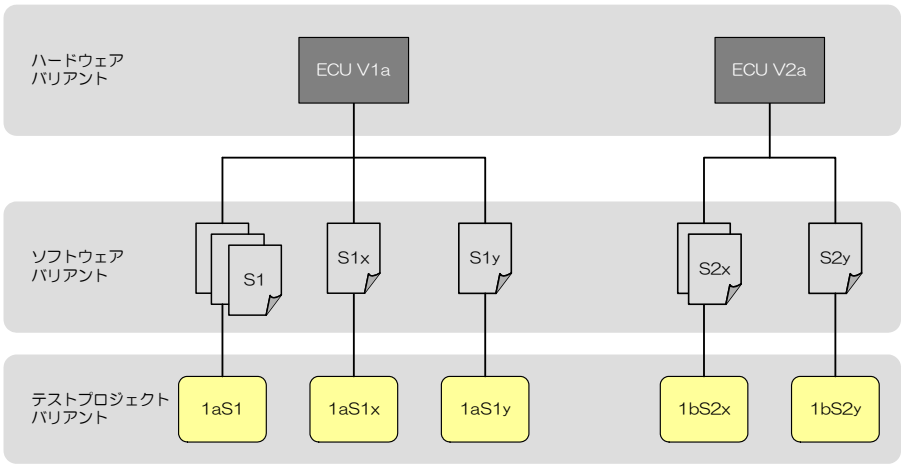


図 4-12 ECU ハードウェアと ECU ソフトウェアのバリアント、およびそれらに対応するテストプロジェクトバリアント

上図に示されるようなバリアント管理においては、データの冗長性が莫大なものとなり、また、全バリアントに共通する変更点が生じた場合は、膨大な工数を費やしてすべてのバリアントを変更しなければなりません。

各バリアント間では、テストプロジェクト内の以下のようなプロジェクトデータにおいて差異が存在する可能性があります。

- モデルパラメータ
- 開ループコンフィギュレーション（センサカーブなど）
- I/O ハードウェアのコンフィギュレーション（測定のモードや精度など）
- ECU と I/O ハードウェアの接続データ

この問題は、グローバルデータと固有データを別に保持することにより解決できます（図 4-13 を参照してください）。

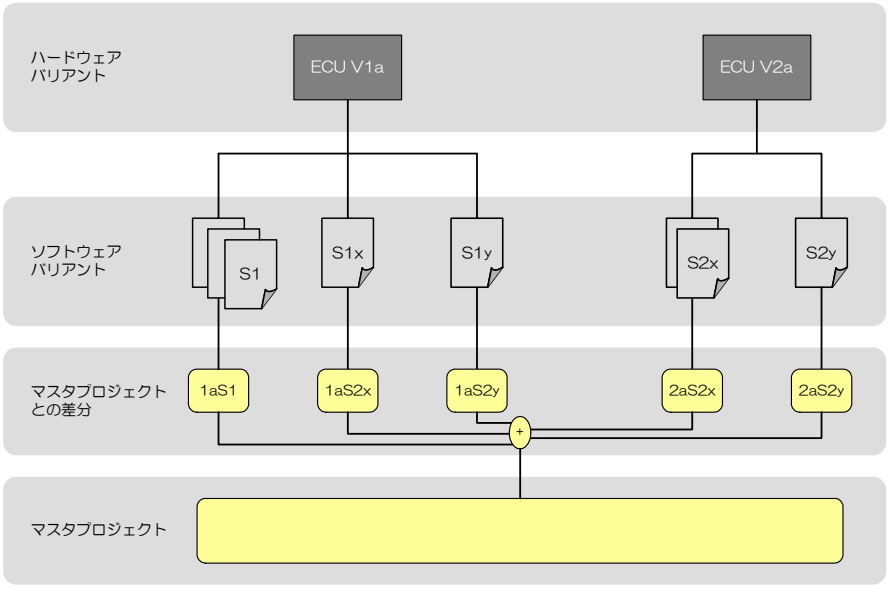


図 4-13 マスタプロジェクト、および各バリアント用の差分データを格納したファイル

マスタプロジェクト

マスタプロジェクトは、デフォルトのパラメータ値を内包する完全な LABCAR-OPERATOR プロジェクトです。これはシングルソースで、冗長データは含まれません。

差分

差分ファイルには、ある UuT バリアントをその UuT 専用のソフトウェアバリアントを使用してテストするための、専用のプロジェクトパラメータが含まれています。これらのファイルは「バリアント」と呼ばれています。

バリアントに含まれるデータ

バリアントには以下のようなデータが含まれます。

- モデルパラメータ

  例：異なるエンジンコンポーネント用のパラメータ（保存先：`*.mpa`、`*.dcm`、`*.cdfx` ファイル）
- OLC パラメータ

  例 : 異なる O2 センサ用のセンサカーブ（保存先： `*.olc` ファイル）

- ハードウェアコンフィギュレーションのパラメータ（エレメントリスト）

  例：異なる I/O ハードウェア設定（保存先：`*.mpa`、または `*.dcm` ファイル）
- 接続データ : ECU ― I/O ハードウェア

  （保存先：`properties.ecu`）

これらのうち最初の 3 種類は「パラメータコンビネーション」として結合され、各ファイルをアクティブにすることにより、実験にロードされます。

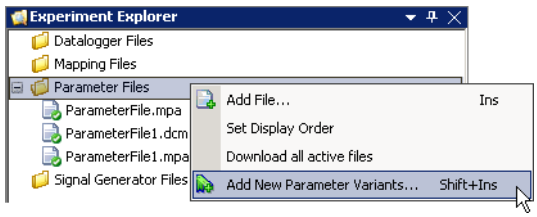
パラメータコンビネーション

モデルパラメータをオンライン実験にロードする際、パラメータファイルは "Parameter Files" フォルダに表示されている順序でターゲットにロードされます。

この順番は、「パラメータコンビネーション」（`*.pac`）としてエクスポートすることができます。各プロジェクトに、これらのコンビネーションを任意の数だけ割り当てることができます。

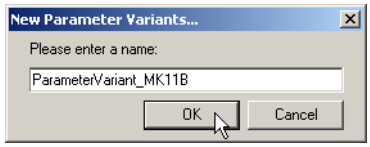
新しいバリアントを作成する：

- "Parameter Files" フォルダを右クリックして **Add new Parameter Variants** を選択します。



"New Parameter Variants" ダイアログボックスが開きます。

- バリアントの名前を入力して **OK** をクリックします。



ファイル選択ダイアログボックスが開きます。

- 新しいバリアントに含めたいファイルを選択して **Open** をクリックします。

  バリアントが作成され、実験に追加されます。

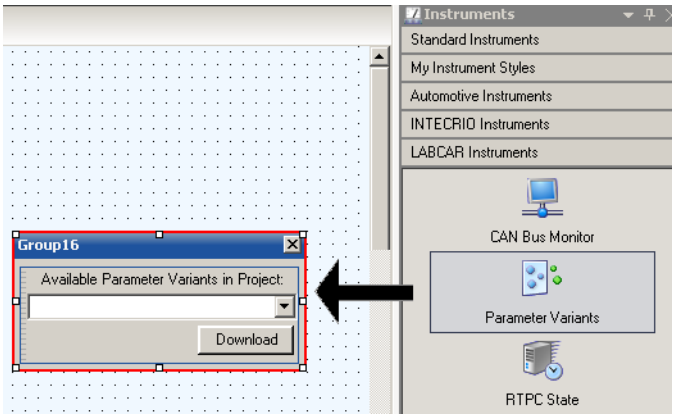
パラメータバリアントについては以下のような管理を行えます。詳しくは 294 ページの「エクスペリメントエクスプローラで行うパラメータファイル管理」を参照してください。

- バリアントにファイルを追加する（**Add File**）
- 順序を変更する（**Set Display Order**）
- アクティブなすべてのファイルをダウンロードする（**Download all Active Files**）
- アクティブ／非アクティブを切り替える（**Active**）
- ファイル名を変更する（**Rename File**）
- ファイルをプロジェクトから除外する（**Exclude from Experiment**）
- ファイルを削除する（**Delete (permanently)**）

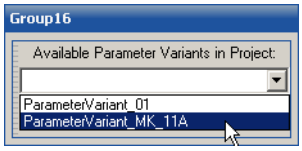
前述のようにショートカットメニューにより実験のすべてのパラメータバリアントをダウンロードできます（**Download all Active Files**）。またインストゥルメントを使用して、任意のタイミングでバリアントを 1 つだけ選択してロードすることもできます。

任意のバリアントを実験にロードする：

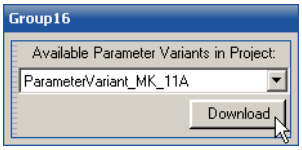
- "Instruments" ウィンドウ（"LABCAR Instruments" タブ）から "Parameter Variants" インストゥルメントを選択してレイヤにドラッグします。



- リストから、プロジェクトに含まれているバリアントの 1 つを選択します。



- **Download** をクリックします。



選択されたバリアントが実験にロードされます。

バリアントは、ファイル（*.pac）の形でエクスポートして実験に追加することもできます。

バリアントを *.pac 形式でエクスポートする：

- パラメータバリアントをエクスポートするフォルダを選択します。
- ショートカットメニューから **Export to PAC File** を選択します。

  ファイル選択ダイアログボックスが開きます。
- ファイル名（*.pac）を入力して **Save** をクリックします。

バリアントをインポートする：

- パラメータバリアントを実験に追加するには、**Add File** を選択します。
  ファイル選択ダイアログボックスが開きます。
- 追加したいファイル（`*.pac`）を選択して **Open** をクリックします。
  選択されたバリアントが実験に追加されます。

4.6.7 マッピングファイル

マッピングファイル内で ASAM ラベルに任意のユーザー定義名を割り当て、ラベルを扱いやすくすることができます。

ETAS LABCAR モデルにはインストゥルメント用のマッピングファイルが付属している場合がありますが、ユーザーがマッピングファイルを独自に作成することもできます。

プロジェクトには任意の数のマッピングファイルを割り当てることができ、それらを LABCAR-EE の エクスペリメントエクスプローラで管理することができます。マッピングファイルの編集は SUT マッピングファイルエディタという専用のエディタで行われます。

マッピングの "All UD" タブへの表示

一例として、ユーザーはモジュール内の測定変数の

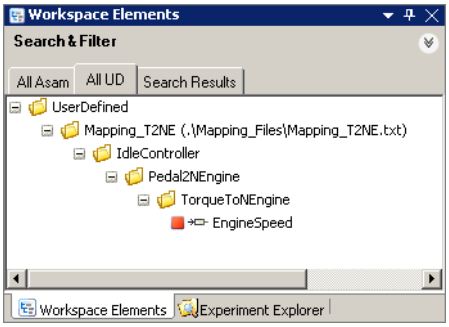
```
IdleController/Pedal2NEngine/TorqueToNEngine/Out
```

というラベル（ツールラベル）を

```
EngineSpeed
```

というラベル（テストラベル）に割り当てることができます。

これらの「ユーザー定義ラベル」は "Workspace Elements" ウィンドウの "All UD" タブに階層表示され、上位階層にファイル名も表示されます。



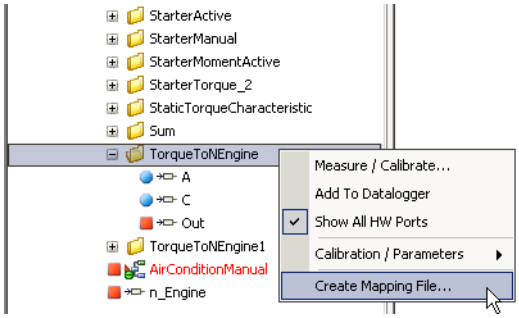
4.6.8 マッピングファイルの作成と管理

本項では、エクスペリメントエクスプローラのショートカットメニューを使用してマッピングファイルを作成し管理する方法について説明します。

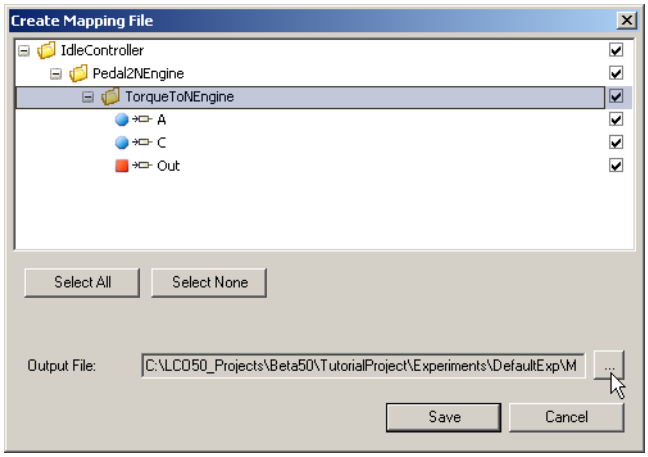
マッピングファイルを作成する：

- "Workspace Elements" タブで、マッピングファイルを作成したいパラメータまたは測定変数を選択します。
  ここで、階層の一部（=フォルダ）またはメインの "LABCAR" ノードを選択することもできます。

- ショートカットメニューから **Create Mapping File** を選択します。



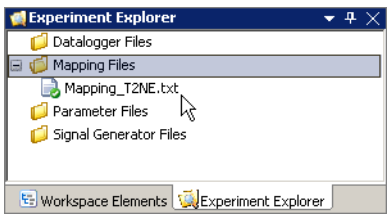
"Create Mapping File" ダイアログボックスが開きます。



- マッピングファイルに含めたいパラメータと測定変数を選択します。
- マッピングの保存先とするファイルを指定するには、**...** をクリックします。

  ファイル選択ダイアログボックスが開きます。
- ファイル名を入力して **Save** をクリックし、ファイル名を確定します。
- "Create Mapping File" ダイアログボックスの **Save** をクリックしてマッピングファイルを保存します。
- "Experiment Explorer" タブに切り替え、"Mapping Files" フォルダをダブルクリックします。

  作成されたマッピングファイルが実験に追加されてアクティブになります（フォルダアイコンの横に緑色のチェックマークが表示されます）。



エクスペリメントエクスプローラの "Mapping Files" フォルダには、実験に属するすべてのマッピングファイルが含まれています。

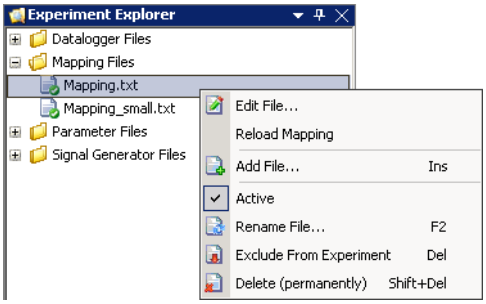


図 4-14 "Mapping Files" フォルダのショートカットメニュー

このフォルダのショートカットメニューから、マッピングファイルの編集や管理を行えます。これらの機能について以下の各項で詳しく説明します。

- マッピングファイルを編集する：(303 ページ)
- マッピングを再度ロードする：(303 ページ)
- マッピングファイルをプロジェクトに追加する：(303 ページ)
- マッピングファイルをアクティブにする：(304 ページ)
- マッピングファイル名を変更する：(304 ページ)
- マッピングファイルをプロジェクトから除外する：(304 ページ)
- マッピングファイルを完全に削除する：(304 ページ)

マッピングファイルを編集する：

- エクスペリメントエクスプローラで、編集したいファイルをダブルクリックします。

または

- そのファイルを右クリックして **Edit File** を選択します。

  SUT マッピングファイルエディタが開きます。

マッピングファイルをプロジェクトに追加する：

- エクスペリメントエクスプローラの "Mapping Files" を右クリックします。
- **Add File** を選択します。

  ファイル選択ダイアログボックスが開きます。
- 追加したいマッピングファイル（`*.txt`、`*.smf`）を選択して **OK** をクリックします。

  指定したマッピングファイルがプロジェクトに追加されます。

マッピングを再度ロードする：

- エクスペリメントエクスプローラで、ロードしたいファイルを右クリックします。
- **Reload Mapping** を選択します。

  選択されたマッピングファイルが実験に再度ロードされます。

マッピングファイルをアクティブにする：

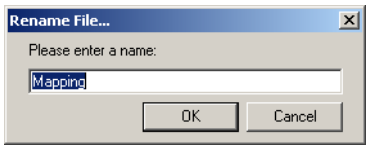
- エクスペリメントエクスプローラで、アクティブにしたいファイルを右クリックします。
- **Active** を選択します。

選択されたマッピングファイルがアクティブになります（チェックマークが表示されます）。

マッピングファイル名を変更する：

- エクスペリメントエクスプローラで、名前を変更したいファイルを右クリックします。
- **Rename File** を選択します。

"Rename File" ダイアログボックスが開きます。



- 新しい名前を入力して **OK** をクリックします。

マッピングファイルをプロジェクトから除外する：

- エクスペリメントエクスプローラで、除外したいファイルを右クリックします。
- **Exclude From Experiment** を選択します。

そのマッピングファイルがプロジェクトから除外されます。

注記

除外されたファイルは削除されません。そのファイルのプロジェクトへの割り当てが解除されるだけです。

マッピングファイルを完全に削除する：

- エクスペリメントエクスプローラで、削除したいファイルを右クリックします。
- **Delete (permanently)** を選択します。

そのマッピングファイルが完全に削除されます。

4.7 LABCAR-PA 1.0 を用いたパラメータファイルの編集

LABCAR-PA 1.0（Parameterization Assistant）を用いることにより、パラメータファイルの内容をわかりやすく表示し、値を編集することができます。

LABCAR-PA には以下のような特長があります。

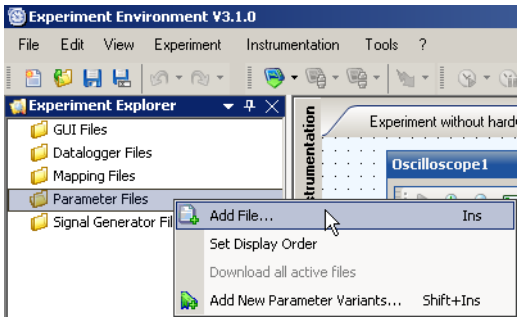
- すべてのモデルパラメータとそれらの属性を構造的に概観できます。
  - モデル階層または機能単位に準じた表示
  - フィルタ機能とソート機能
- パラメータに簡単にアクセスできるため、設定作業を快適に行うことができます。
  - オンライン操作とオフライン操作が可能
  - パラメータの設定状態と属性の表示
  - 変更の有無の表示
  - パラメータの設定値を参照値と比較することが可能
- モデルのパラメータ設定を XML ファイル形式で一元管理できます。
- パラメータを DCM ファイルや M ファイルからインポートしたり、DCM ファイルや M ファイルにエクスポートしたりできます。
- モデルコンポーネントを簡単に交換できます。
- モデルを簡単に更新できます。
- ドキュメント（リファレンスマニュアル）に直接アクセスできます。

4.7.1 パラメータファイルの管理

LABCAR-EE のエクスペリメントエクスプローラには "Parameter Files" というフォルダが含まれ、ここで実験で使用するパラメータファイルの管理を行うことができます。

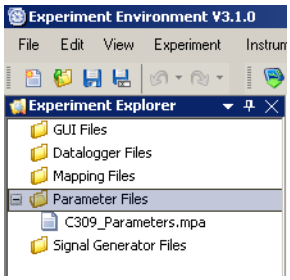
パラメータファイルを実験に追加する：

- エクスペリメントエクスプローラで "Parameter Files" を右クリックしてショートカットメニューを開きます。
- **Add File** を選択します。



- ファイル選択ダイアログボックスで、追加したいパラメータファイルを選択します。

  ファイルが追加されます。

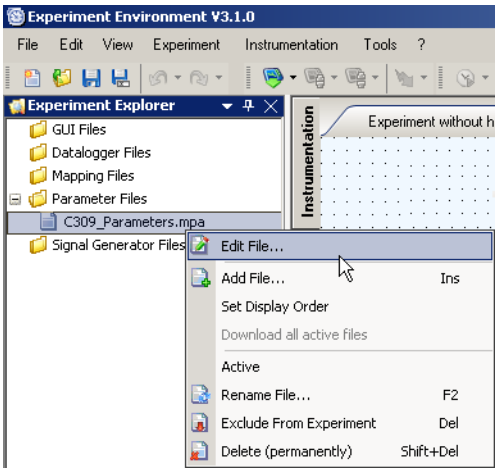


パラメータファイルを編集する：

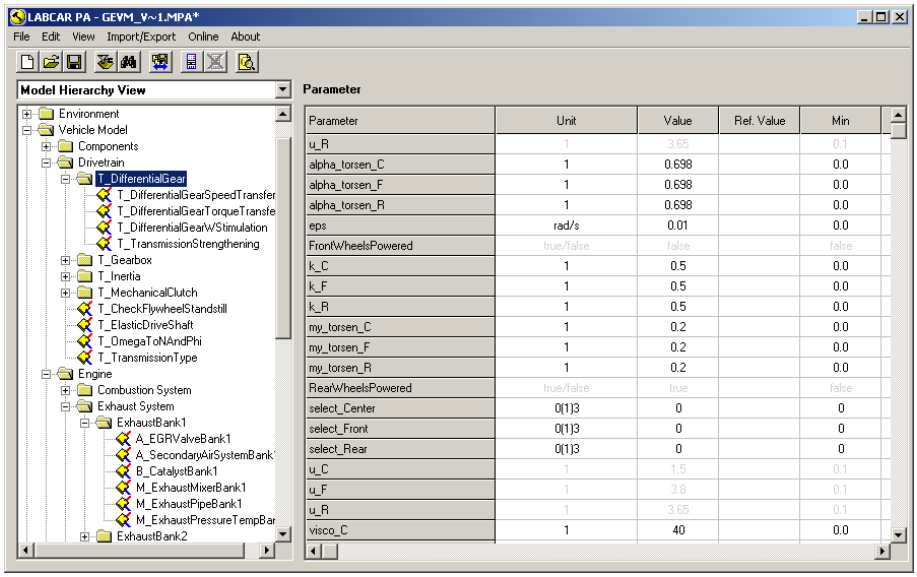
- 編集したいパラメータファイルをダブルクリックします。

または

- 編集したいパラメータファイルのショートカットメニューから **Edit File** を選択します。



LABCAR-PA が開き、選択されたファイルがロードされます。



以下に、LABCAR-PA のユーザーインターフェースについての説明と操作方法をまとめます。

4.7.2 パラメータテーブル

本項では、パラメータテーブルの内容と機能について説明します。説明には、以下の情報が含まれます。

- パラメータテーブルの表示項目の変更（307 ページ）
- パラメータテーブル内の各表示項目（308 ページ）
- ユーザーが編集できる属性（309 ページ）
- パラメータテーブルのショートカットメニュー（309 ページ）
- パラメータテーブルの表示オプション（310 ページ）

パラメータの編集方法については 321 ページの「パラメータの編集」の項で説明します。

パラメータテーブルの表示項目の変更

パラメータテーブルの各列に表示される属性の数を変更することができます（308 ページの「パラメータテーブル内の各表示項目」を参照してください）。この設定は **View** メニューで行います（313 ページの「"View" メニュー」を参照してください）。

以下の設定が可能です。

- **Basic Attributes**

  パラメータ名以外に、"Unit"、"Value"、"Comment" 属性が表示されます。

- **All Attributes**

  項に記載されているすべての属性が表示されます。

- **Custom Attributes**

  View → Customize Attribute View を用いて選択したすべての属性が表示されます（313 ページの「"View" メニュー」を参照してください）。

パラメータテーブル内の各表示項目

ここでは、パラメータテーブル内の各列に表示されるパラメータ属性の意味について説明します。

- Parameter
  モデルブロック内で用いられるパラメータの名前です。
- Unit
  パラメータの物理単位（km/h や Pa/s など）です。
- Value
  スカラパラメータの値、または非スカラパラメータの最初（先頭）の値です。
- Ref. Value
  参照ファイルに定義されているパラメータ値です。
- Min
  パラメータ値の推奨範囲の最小値です。
- Max
  パラメータ値の推奨範囲の最大値です。
- Dimension
  パラメータのディメンションです。
  — Scalar （スカラ値）
  — Array（固定配列）
  — 1D-Table（カーブ）
  — 2D-Table（マップ）
- Scope
  パラメータの有効範囲です。
  — local
  — exported
  — imported
  パラメータがインポートされるもの（= imported）である場合、この列にはそのパラメータがエクスポートされているブロック名が表示されます。
- Proc. State
  パラメータの設定タイプです。
  — coarse（基本パラメータ）
  — fine（ファインチューニング用パラメータ）
  — switch（モードスイッチ）
- Type
  パラメータのデータ型です。
  — sdisc（整数）
  — cont（浮動小数点）
  — log（ブール値）
- ModelOption
  パラメータが使用されるモデルのタイプ（general、VVLT、GDI など）を示します。

- UserGroup

  "initial" と表示されている場合、このパラメータが、LABCAR プロジェクトを初めて実行する前に設定しておく必要があることを示します。このようなパラメータは、**View → Initial Project Parameters** により表示できます。

  現バージョンにおいては、"initial" 以外の値は意味を持ちません。
- Mod. Status

  パラメータの変更ステータスを示します。

  — unchanged

  — changed

  — accepted

  パラメータの変更ステータスを示す文字色については、310 ページ「パラメータテーブルの表示オプション」で説明します。
- Comment

  パラメータについての注釈が表示されます。

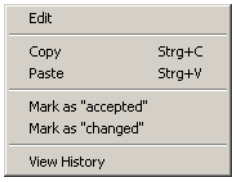
ユーザーが編集できる属性

ユーザーが編集できるのは、以下の属性の値です。

- Value
- Min
- Max
- Proc. State
- Mod. Status

パラメータテーブルのショートカットメニュー

テーブルの任意のフィールドを選択して右クリックすると、ショートカットメニューが開き、値の変更、変更ステータス（changed、accepted）の設定、変更履歴の表示などを行うためのメニューコマンドが表示されます。



- Edit

  現在選択されているフィールドを編集します。

  注記

  "Value" 列のパラメータ値のフィールドをクリックすれば、その値を直接変更することができます。
- Copy, Paste

  値、または選択範囲全体のコピーと貼り付けを行います。
- Mark as accepted

  パラメータの値を承認し、変更ステータスを "accepted" にします。パラメータ名とその値が緑の文字で表示されます。

- Mark as changed

  パラメータの変更ステータスを "changed"（変更済み）にします。パラメータ名とその値が赤い文字で表示されます。

- View History

  "Parameter History" ダイアログボックスを開きます。ここにはパラメータの変更履歴が表示され、各レコードに注釈を入力することもできます（詳細は 325 ページの「変更履歴の表示と編集」を参照してください）。

注記

各操作においてエラーメッセージが表示される場合、ドキュメントファイルへのパスが正しく設定されていない（318 ページの「パス設定」を参照してください）か、あるいはドキュメントファイルが存在していない場合があります。

パラメータテーブルの表示オプション

パラメータテーブル内の各パラメータの表示スタイルは、そのパラメータの状態（以下の 6 通りがあります）によって異なります。

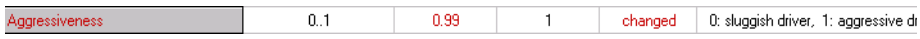
- 変更されたパラメータ
- 変更されて承認されたパラメータ
- インポートパラメータ
- 変更されたインポートパラメータ
- 変更されて承認されたインポートパラメータ
- 変更されたパラメータ（参照ファイルが開いている場合）

変更されたパラメータ :

パラメータの内容が変更されると、以下の列のフィールドの文字色が赤になります。

- Parameter Name
- Parameter Value
- Modification Status

また "Modification Status" の表示内容が "changed" に変わります。



変更されて承認されたパラメータ :

変更されたパラメータ（"Modification Status" が "changed" であるパラメータ）の値が承認されたものであることを示すためには、"Modification Status" を "accepted"（「承認済み」）に設定します（309 ページの「パラメータテーブルのショートカットメニュー」を参照してください）。

赤い文字で表示されていたフィールドが、緑色の文字になります。

"Modification Status" の値が "unchanged" のパラメータも、同様に "accepted" に設定することができます。

インポートパラメータ :

インポートされるパラメータを変更することはできないため、インポートパラメータは灰色で表示されます。

インポートパラメータの内容は、そのパラメータのエクスポート元のブロックでしか変更できません。エクスポート元のブロックを見つける方法は、336 ページの「パラメータスコープのバインディング」の項で説明します。

変更されたインポートパラメータ：

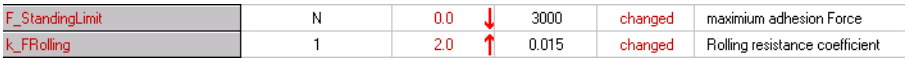
インポートパラメータの内容がエクスポート元のブロックで変更された場合、そのパラメータ名は赤い文字で表示され、さらに "Value" と "Modification Status" がオレンジ色で表示されます。

変更されて承認されたインポートパラメータ：

インポートパラメータの内容がエクスポート元のブロックで変更されて承認された場合、そのパラメータ名は緑の文字で表示され、さらに "Value" と "Modification Status" が薄緑色で表示されます。

変更されたパラメータ（参照ファイルが開いている場合）

パラメータが変更されていて、かつ参照ファイルが開いている場合、参照ファイルの値とパラメータ値を比較した結果がパラメータ値右側の矢印で示されます。



変更されたパラメータがスカラ値でない場合は、そのパラメータ内に参照値よりも大きい値と小さい値が混在している可能性があります。そのような場合は上下の両方向を示す矢印で示されます。

注記

矢印は、新しい値と参照値の差がユーザー設定された割合（%）を超えた場合にのみ表示されます（317 ページの「参照ファイルに関する設定」を参照してください）。

注記

"Enumeration" 型のパラメータの場合、変更内容は表示されません。

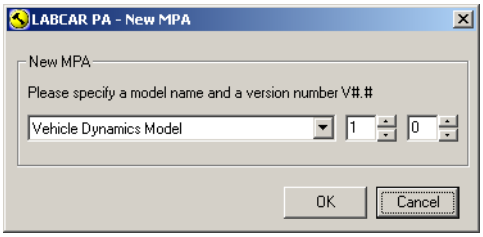
4.7.3 メニューの説明

本項では各メニューコマンドの機能について概説します。

"File" メニュー

- **File → New**

  新しい空のプロジェクトパラメータファイル（*.mpa）を作成します。

  ここで任意のモデルまたはモデルコンポーネントとバージョンを選択すると、作成されたファイル内のパラメータデータをモデルまたはモデルコンポーネントに割り当てるのに役立ちます。

  完全なモデルについての全階層が作成され、ブロック構成リストの最上位アイテムとしてモデル名が表示されます。

- **File → Open**

  ファイル選択ダイアログを開いて既存のプロジェクトパラメータファイル（*.mpa）をロードします。

  ワーキングディレクトリは **Edit → Options** メニューコマンドで指定します（312 ページの「"Edit" メニュー」を参照してください）。

- **File → Close**

  現在開いているプロジェクトパラメータファイルを閉じます。

- **File → Save**

  現在開いているプロジェクトパラメータファイルを保存します。

- **File → Save As**

  現在開いているプロジェクトパラメータファイルを、名前を付けて保存します。

- **File → Open Reference**

  参照ファイルをロードします（326 ページの「参照ファイル」の項を参照してください）。

- **File → Set To Reference File**

  現在のパラメータ設定の内容を参照ファイルとして保存します。デフォルトファイル名は *<file_name>*Reference_0.mpa です。

- **File → Close Reference**

  現在開いている参照ファイルを閉じます。

- **File → Exit**

  プログラムを終了します。

"Edit" メニュー

- **Edit → Edit Parameter**

  選択されているパラメータの値を編集します（321 ページの「パラメータの編集」を参照してください）。

- **Edit → Filter Settings**

  パラメータの表示フィルタとソート方法を指定するためのダイアログボックスが開きます（332 ページの「フィルタとソートの機能」を参照してください）。

- **Edit → Search Model Parameters**

  モデルパラメータを検索するためのダイアログボックスが開きます（334 ページの「モデルパラメータ検索機能」を参照してください）。

- **Edit → Options**

  各種オプション設定を行うためのダイアログボックスが開きます（316 ページの「オプションの設定」を参照してください）。

"View" メニュー

- **View → Initial Project Parameters**

  この項目を選択すると、LABCAR プロジェクトを初めて実行するときに必要なパラメータだけが表示されます。

- **View → Parameter Scope Binding**

  他のブロックからエクスポートされるインポートパラメータの一覧を開きます。エクスポート元やインポート先のブロック名もそれぞれ明示されます（336 ページの「パラメータスコープのバインディング」を参照してください）。

- **View → Basic Attributes**

  この項目を選択すると、"Unit"、"Value"、"Comment" の属性のみがパラメータテーブルに表示されます（307 ページの「パラメータテーブルの表示項目の変更」を参照してください）。

- **View → All Attributes**

  この項目を選択すると、すべての属性がパラメータテーブルに表示されます（307 ページの「パラメータテーブルの表示項目の変更」を参照してください）。

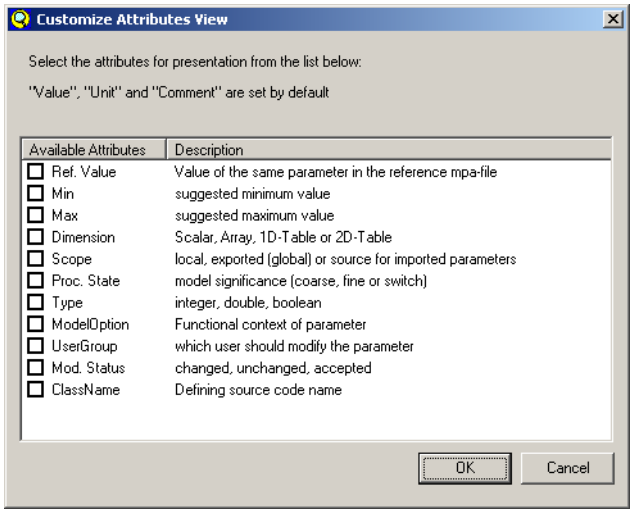
- **View → Custom Attributes**

  この項目を選択すると、**View → Customize Attribute View** で選択したすべての属性がパラメータテーブルに表示されます（307 ページの「パラメータテーブルの表示項目の変更」を参照してください）。

- **View → Customize Attribute View**

  ここでは、**View → Custom Attributes** で表示される属性を選択することができます（307 ページの「パラメータテーブルの表示項目の変更」を参照してください）。

  これらの属性の意味については、308 ページの「パラメータテーブル内の各表示項目」を参照してください。



"Import/Export" メニュー

- **Import/Export → Import/Export Parameter Data**

  パラメータをパラメータファイルからインポート、またはパラメータファイルにエクスポートするためのダイアログボックスを開きます（329 ページの「パラメータをファイルからインポートする」と 326 ページの「パラメータをファイルにエクスポートする」を参照してください）。

"Online" メニュー

- **Online → Go Online**

  実行中のシミュレーションにおいて、開いている mpa ファイルのパラメータがシミュレーションモデル内で使用されているかどうかのチェックが行われます（ASAM-MCD-2MC ラベルで比較されます）。詳細は、338 ページの「オンラインでのパラメータ設定」を参照してください。

- **Online → Synchronize ASAM Labels**

  LabCar Developer モデル内のパラメータ名は、Simulink モデルのパラメータ名とは異なるため、この同期機能によって Simulink モデルからインポートされたパラメータと mpa ファイル内のパラメータを同期させ、必要に応じて ASAM-MCD-2MC ラベルを適合させます。

- **Online → Read Parameter**
  - **Read Parameter from Target**

    パラメータテーブル内で選択されているパラメータの値を、実行中のモデルから読み取ります。
  - **Read all Parameters from Target**

    選択されている階層内のすべてのパラメータの値を、実行中のモデルから読み取ります。

- **Online → Write Parameter**
  - **Write Parameter to Target**

    パラメータテーブル内で選択されているパラメータの値を、実行中のモデルに書き込みます。ただし、パラメータテーブルで値が変更されている場合、この機能はすでに自動的に実行されています。

— **Write all Parameters to Target**

選択されている階層内のすべてのパラメータの値を、実行中のモデルに書き込みます。

- **Online → Go Offline**

  オンラインでのパラメータ設定を終了します。

4.7.4 ツールバー

これまでに説明したメニュー機能の中には、ツールバーのボタンから実行できるものもあります。

- **Create new MPA file**

  新しい空のプロジェクトパラメータファイル（*.mpa）を作成します。**File → New** と同じ機能です。

- **Open MPA file**

  プロジェクトパラメータファイル（*.mpa）をロードするためのファイル選択ダイアログを開きます。**File → Open** と同じ機能です。

- **Save MPA file**

  現在開いているプロジェクトパラメータファイルを保存します。**File → Save** と同じです。

- **Filter settings**

  表示フィルタとソート基準を入力するためのダイアログボックスを開きます。**Edit → Filter Settings** と同じ機能です。

- **Search parameters**

  モデルパラメータを検索するためのダイアログボックスを開きます。**Edit → Search Model Parameters** と同じ機能です。

- **Import/Export parameters**

  パラメータをパラメータファイルからインポート、またはパラメータファイルにエクスポートするためのダイアログボックスを開きます。**Import/Export → Import/Export Parameter Data** と同じ機能です。

- **Online parameterization**

  現在のパラメータの値を実行中のモデルにロードします（オンラインでのパラメータ設定）。**Online → Go Online** と同じ機能です。

- **Offline parameterization**

  オンラインでのパラメータ設定を終了します。**Online Go Offline** と同じ機能です。

- **View parameter scope binding**

  エクスポート、インポートされるパラメータの一覧を開きます。エクスポート元やインポート先のブロック名もそれぞれ明示されます。**View → Parameter Scope Binding** と同じ機能です。

4.7.5 LABCAR-PA 1.0 の操作方法

以下の項で、LABCAR-PA 1.0 の操作方法について説明します。

内容は以下のとおりです。

- オプションの設定（316 ページ）

  各種オプションの設定方法について説明します。

- パラメータの編集（321 ページ）

  各パラメータの編集方法について説明します。

- 参照ファイル（326 ページ）
  参照ファイルの使用方法について説明します。
- パラメータをファイルにエクスポートする（326 ページ）
  モデルパラメータを M ファイルまたは DCM ファイルにエクスポートする方法について説明します。
- パラメータをファイルからインポートする（329 ページ）
  モデルパラメータを M ファイルまたは DCM ファイルからインポートする方法について説明します。
- フィルタとソートの機能（332 ページ）
  パラメータテーブルの表示フィルタとソートの機能について説明します。
- モデルパラメータ検索機能（334 ページ）
  モデルパラメータの検索機能について説明します。
- パラメータスコープのバインディング（336 ページ）
  モデルのブロックにインポート、またはモデルのブロックからエクスポートされるパラメータの識別方法について説明します。
- オンラインでのパラメータ設定（338 ページ）
  モデルを実行しながらパラメータを設定する方法について説明します。

4.7.6 オプションの設定

本項では、以下のオプションの設定について説明します。

- 全般的なオプション（316 ページ）
- 参照ファイルに関する設定（317 ページ）
- パス設定（318 ページ）
- パラメータのインポートとエクスポートに関するオプション（319 ページ）
- 履歴オプション（320 ページ）

全般的なオプション

全般的なオプションとして、以下の設定を行うことができます。

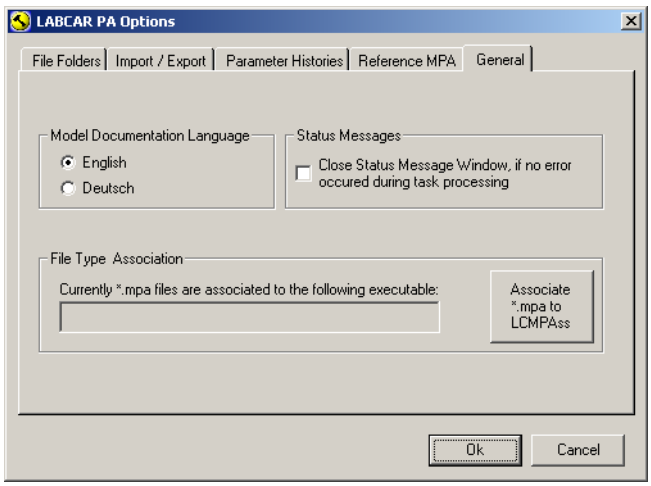
- モデルブロックのドキュメントの言語を指定する
- ステータスメッセージウィンドウの表示モードを指定する
- ファイルタイプ `*.mpa` の関連付けを変更する

これらのオプション設定を変更するには、以下の手順を実行します。

全般的なオプション設定のダイアログを開く：

- **Edit → Options** を選択します。
  "LABCAR-PA Options" ダイアログボックスが開きます。

- "General" タブを選択します。



モデルのリファレンスガイドの言語を指定する：

- "Model Documentation Language" フィールドの "English"（英語）と "Deutsch"（ドイツ語）のいずれかを選択します。

注記

日本語のモデル説明は、日本語ユーザーズガイドを参照してください。日本語ユーザーズガイドが製品に含まれてない場合は、イータスの LABCAR ホットラインまでお問い合わせください。

ステータスメッセージウィンドウの表示モードを指定する：

- "Close Status Message Window, if no error occurred during task processing" というオプションを有効にしておくと、実行された処理が正常に終了した場合は、処理終了後に "Status Message Window" が自動的に閉じます。

通常、Microsoft Windows オペレーティングシステムでは、`*.mpa` というファイルタイプはマルチメディアデータとして扱われます。つまり、このようなファイルをダブルクリックすると、デフォルト状態においては LABCAR-PA ではなくマルチメディアプレーヤが起動してしまいます。

ファイルタイプ *.mpa の関連付けを変更する：

- `*.mpa` という拡張子を持つファイルを LABCAR-PA 1.0 に関連付けるには、"File Type Association" の **Associate *.mpa to LCMPAss** ボタンをクリックします。

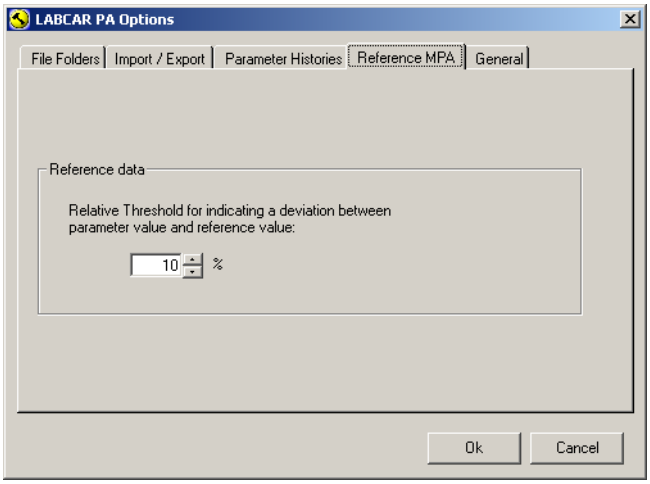
参照ファイルに関する設定

参照ファイルが開いていると、ロードされているパラメータファイル内の値と参照ファイル内の値との差が一定の割合を超えているパラメータについては、そのパラメータの値の右側にに矢印（上向きまたは下向き）が表示されます。

矢印を表示される基準となる割合は、任意に設定することができます。

参照ファイルの値との差の許容範囲を設定する：

- **Edit □Options** を選択します。
  "LABCAR-PA Options" ダイアログボックスが開きます。
- "Reference MPA" タブを選択します。



- エディットボックスの右にある上下の矢印をクリックしてしきい値を調整するか、またはその値を直接キー入力します。

パス設定

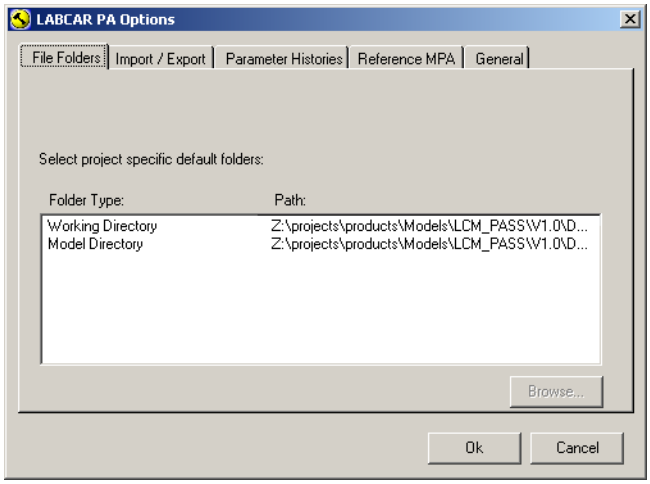
LABCAR-PA 1.0 を操作するためには、以下のディレクトリのパスが正しく設定されている必要があります。

- ワーキングディレクトリ
  設定されたパラメータを格納するすべてのファイル（`*.mpa`、`*.dcm`、`*.m`）に使用するディレクトリです。
- モデルディレクトリ
  このパスは本バージョンでは使用されません。
- モデルドキュメントディレクトリ
  ブロックとモデルに関するリファレンスガイド（`*.pdf`）用のディレクトリです。

ワーキングディレクトリを変更するには、以下の手順を実行します。

ワーキングディレクトリを変更する：

- **Edit □Options** を選択します。
  "LABCAR-PA Options" ダイアログボックスが開きます。



- "File Folders" タブを選択します。
- マウスで "Working Directory" を選択します。
- **Browse** ボタンをクリックします。
  "Select Folder" ダイアログボックスが開きます。
- ドライブとフォルダを選択します。
- **OK** をクリックします。

モデルとモデルドキュメントのディレクトリの変更も、同様の手順で行います。

パラメータのインポートとエクスポートに関するオプション

パラメータをファイルからインポート、またはファイルにエクスポートする際に使用される以下のオプションを指定することができます。

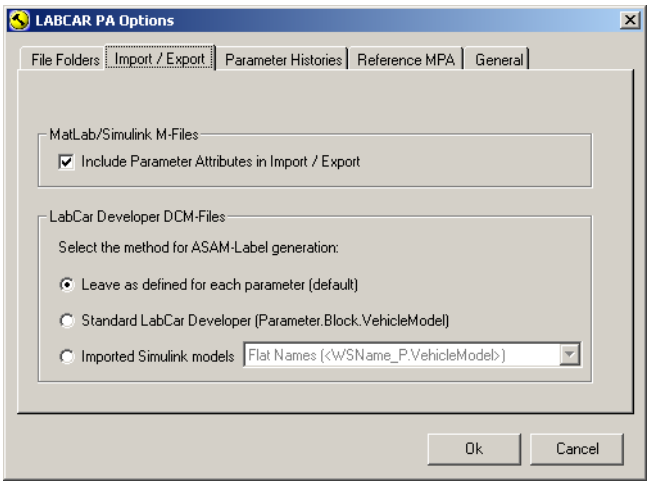
- Matlab Simulink の M ファイルからのインポート、または M ファイルへのエクスポート時に ETAS の属性を含めるかどうか
- ASAM-MCD-2MC ラベルを生成する方法

これらのオプションを変更するには、以下の手順を実行します。

インポート／エクスポートのオプションを変更する：

- **Edit □Options** を選択します。
  "LABCAR-PA Options" ダイアログボックスが開きます。

- "Import/Export" タブを選択します。



"Matlab/Simulink M-Files" ボックス内の "Include Parameter Attributes in Import/Export" オプションを有効にすると、エクスポート時にはパラメータ属性が M ファイルに書き込まれ、インポート時にはパラメータ属性が M ファイルから読み込まれます。

"LabCar Developer DCM-Files" ボックスでは、各パラメータ用の ASAM-MCD-2MC ラベルを生成する方法を指定します。以下のオプションの中から選択できます。

- Leave as defined for each parameter（default）

  mpa ファイル内で用いられているものをそのまま ASAM-MCD-2MC ラベルとして使用します（"ModelPath" 属性）。
- Standard LabCar Developer（Parameter.Block.VehicleModel）

  LabCar Developer の構文（Parameter_name.Block_name.Model_name）を使用します。
- Imported Simulink Models

  Simulink モデルをインポートする際に使用されるオプションです。
  - Flat Names（<WSName_P.VehicleModel>）

    Workspace_name.Model_name
  - Standard SL Import

    完全な階層でインポートします。
  - SL Import Reverse Quantity Names

    逆順の完全な階層でインポートします。

履歴オプション

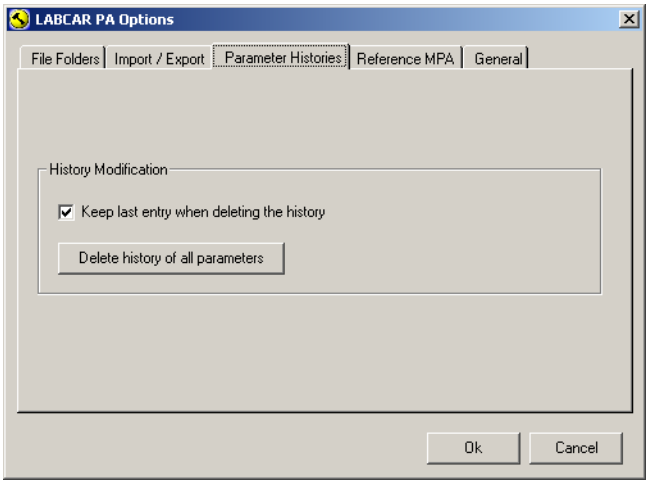
履歴ファイルには、パラメータファイルが編集されるたびに変更履歴レコードが追加されるため、パラメータを頻繁に変更するとファイルはかなり大きくなってしまいます。そのような場合、履歴レコードを削除してファイルを小さくすることができます。

最新のレコードのみを残して他のレコードがすべて削除されるように設定することも可能です。

履歴レコードを削除する：

変更履歴のレコードを削除するには、以下の手順を実行します。

- **Edit → Options** を選択します。
  "LABCAR-PA Options" ダイアログボックスが開きます。
- "Parameter Histories" タブを選択します。



- "Keep last entry when deleting the history" を有効にしておくと、履歴レコードをすべて削除する際、最後のレコードは削除されずに残ります。
- **Delete history of all parameters** をクリックすると、履歴レコードが削除されます。

4.7.7 パラメータの編集

本項では、パラメータの編集に関して、以下の内容について説明します。

- パラメータ値の編集（321 ページ）
  パラメータのタイプに応じた編集方法を説明します。
- 最小値と最大値の編集（323 ページ）
  最小値と最大値の編集に関するヒントを紹介します。
- テーブルエディタ（324 ページ）
  テーブルエディタの操作方法について説明します。
- 変更履歴の表示と編集（325 ページ）
  パラメータ履歴の取り扱いについて説明します。

パラメータ値の編集

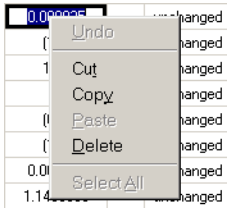
パラメータにはさまざまなタイプがあり、それぞれ編集方法も異なります。
以下のケースについて説明します。

- パラメータが整数または浮動小数点数である場合（322 ページ）
- パラメータが 1D テーブル（カーブ）または 2D テーブル（マップ）である場合（322 ページ）
- パラメータが論理値である場合（323 ページ）
- パラメータがインポートされるものである場合（323 ページ）

パラメータが整数または浮動小数点数である場合：

パラメータ値を編集するには、そのパラメータの行の "Value" 列のフィールドを左クリックします。すると、そのフィールドが編集モードになり、値の変更が可能になります。

編集モードになっているフィールドを右クリックすると、下図のようなショートカットメニューが開きます。



このメニューから、以下の操作を実行できます。

- 直前に行った入力の取り消し
- 値の切り取り
- 値または選択範囲のコピー
- 値の削除
- 文字列全体の選択

注記

直前に行われていた操作に応じて、特定のメニューコマンドのみが選択可能になります。

パラメータが 1D テーブル（カーブ）または 2D テーブル（マップ）である場合：

テーブルタイプのパラメータの値を編集するには、そのパラメータの行の "Value" 列のフィールドダブルクリックします。すると、このパラメータ用のテーブルエディタが開きます。

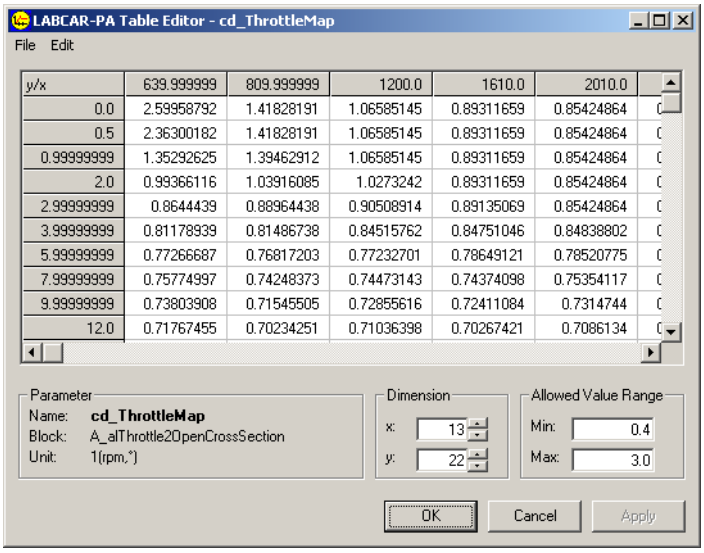
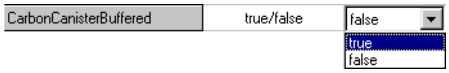


図 4-15 テーブルエディタ

テーブルエディタの操作方法については、324 ページの「テーブルエディタ」の項で説明します。

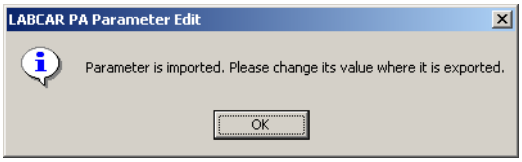
パラメータが論理値である場合：

パラメータの値が true か false である場合、そのフィールドでマウスボタンをクリックするとドロップダウンリストが開き、値を選択することができます。



パラメータがインポートされるものである場合：

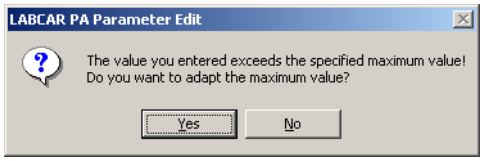
インポートされるパラメータはそのブロック内で編集することはできないため、そのデータはグレイアウト表示となります。パラメータを編集しようとすると以下のメッセージが表示されます。



インポートパラメータの内容は、そのパラメータをエクスポートするブロックでしか変更できません。パラメータのエクスポート元のブロックを明らかにするには、ショートカットメニューから **Select source parameter（"exported"）**を選択します。すると、エクスポート元のブロックがアクティブになり、そのブロックのパラメータがテーブルに表示されます。エクスポートされるパラメータ名の背景が青色になります。

最小値と最大値の編集

パラメータ値の最小値と最大値は、モデルを適切に機能させるために設定された限界値です。この限界値を超える値をパラメータ値として入力しようとすると、以下のような警告メッセージが発行されます。



この場合、変更を取り消すか実行するかを選択できます。ここで変更を実行した場合、その値に応じて限界値も変わります。

テーブルエディタ

1D または 2D のテーブルで構成されるパラメータの値をダブルクリックすると、テーブルエディタが開きます。

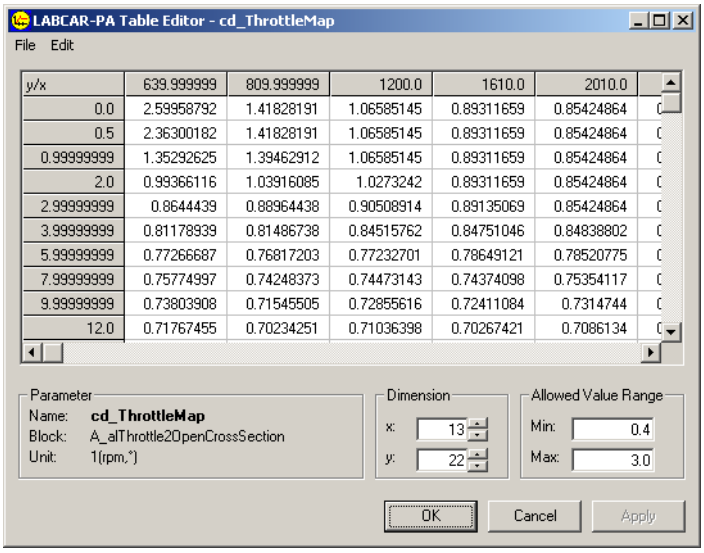
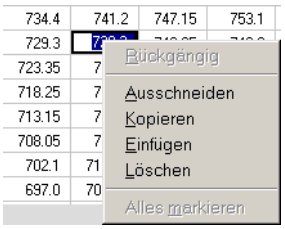


図 4-16 "cd_ThrottleMap" というパラメータを表示したテーブルエディタ

テーブル内のデータフィールドを左クリックするとその内容を編集できます。

編集モードになっているフィールドを右クリックすると、下図のようなショートカットメニューが開きます。



このメニューから、以下の操作が可能です。

- 直前に行ったデータ入力の取り消し
- 値の切り取り
- 値のコピー
- 値の削除
- 文字列全体の選択

注記

前に行われていた操作に応じて、特定のメニューコマンドのみが選択可能になる場合があります。

テーブルエディタのユーザーインターフェースは、メニューといくつかのフィールドで構成されています。それらについて、以下に説明します。

メニュー：

テーブルエディタのメニューには以下の項目が含まれます。

- **File → Open Reference Table Viewer**

  別のテーブルエディタが開き、参照ファイル内の同じテーブルのデータが表示されます。このエディタには **Data → Copy Reference Data to Table** というメニューコマンドがあり、これを選択すると参照ファイルのテーブルの内容がすべて現在編集中のテーブルにコピーされます。

- **Edit → Edit Axes**

  テーブルの座標ポイントの x/y 座標値を変更します（325 ページの「"Dimension" フィールド」を参照してください）。

- **File → Import from File**

  M ファイルまたは DCM ファイルからデータをインポートします。

"Parameter" フィールド

このフィールドには、パラメータについての情報が含まれます。

- Name

  パラメータ名

- Block

  パラメータが属するブロック名

- Unit

  パラメータの物理単位

"Dimension" フィールド

このフィールドには、テーブルの列数と行数が表示されます。この値は変更可能です。

324 ページの図 4-16 は、燃料濃度マップ（2D テーブル）を示しています。このマップに 1 列を追加するには、"Dimension" フィールドの x の値を "1" 大きくします。

指定の数の列が最後部に追加され、新しい列の x 座標値は外挿法により算出されます。

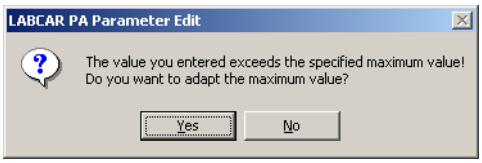
"2200" という新しい x 座標値が必要な場合は、**Edit → Edit Axes** コマンドを選択し、"2001.0" を "2200.0" に変更します。この際、この x 座標値に合わせて z 値（出力値）も変更する必要があります。

行の追加も同様に行えます。

列／行の数を減らすと、最後部の行／列が削除されます。

"Allowed Value Range" フィールド

ここでテーブルの出力値（z 値）の限界値（最小値と最大値）を設定します。これらを超える値を出力値として入力しようとすると、下図のような警告メッセージが表示されます。



No を選択すると、変更は取り消されます。**Yes** を選択すると入力された限界値が有効になります。

変更履歴の表示と編集

パラメータテーブルの任意の行を選択して右クリックしてショートカットメニューを開き、**View History** というメニューコマンドを選択すると、"Parameter History" ダイアログボックスが開きます。

このダイアログボックスには選択されているパラメータの変更履歴が表示され、パラメータが属するブロックの名前も表示されます。

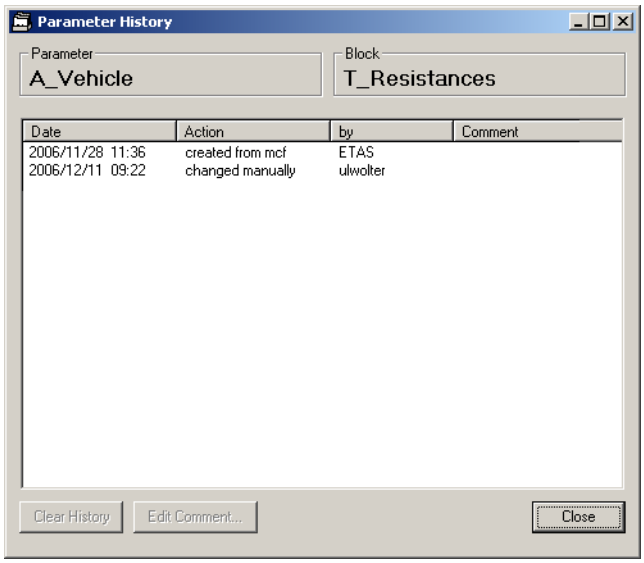
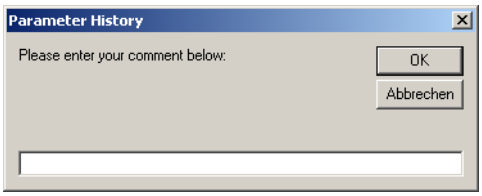


図 4-17 パラメータの履歴

Clear History ボタンをクリックすると、そのパラメータのすべての履歴レコードが削除されます。ただし、オプション設定で、最後のレコードのみ残すように設定しておくことができます（320 ページの「履歴オプション」の項を参照してください）。

マウスでいずれかのレコードを選択すると、**Edit Comment** ボタンがアクティブになります。

このボタンをクリックすると別のダイアログボックスが開き、そこで任意に注釈を入力することができます。



4.7.8 参照ファイル

モデルのパラメータ設定を行う際、充分にテストされた一連のパラメータ値を参照ファイルから読み込んで、その値を参考に変更を加えていけば、より確実で迅速なパラメータ設定を行うことができます。

設定されたパラメータを参照ファイルとして保存するには、**File → Set To Reference File** を選択します。するとファイル選択ダイアログボックスが開き、元の mpa ファイル名に文字列 "Reference_*N*" を加えた名前が、参照ファイル名として表示されます。

保存されている参照ファイルをロードするには、**File → Open Reference** を選択し、ファイル選択ダイアログから必要なファイルを選択してロードします。

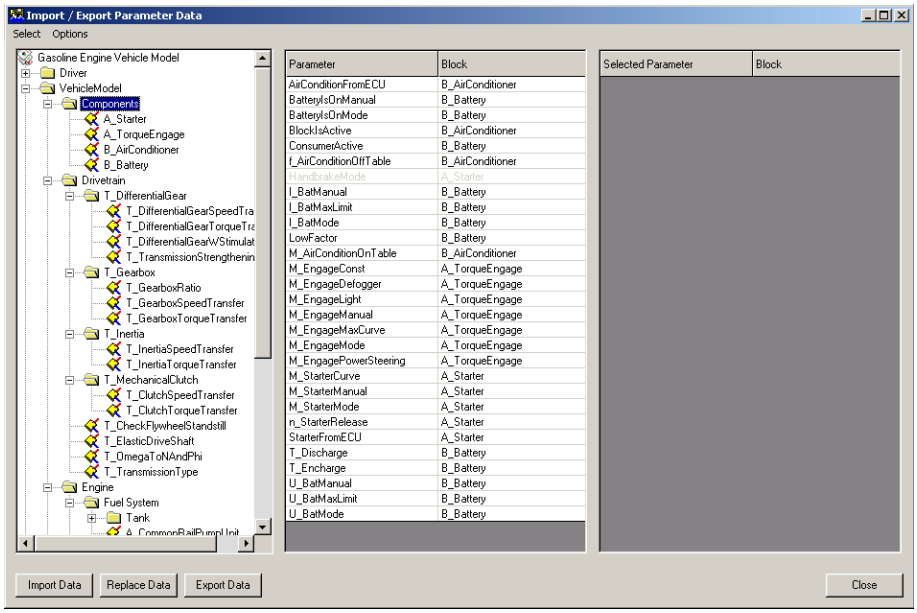
4.7.9 パラメータをファイルにエクスポートする

本項では、パラメータを M ファイルまたは DCM ファイルにエクスポートする方法について説明します。

パラメータを新しいファイルにエクスポートする（"Export Data"）：

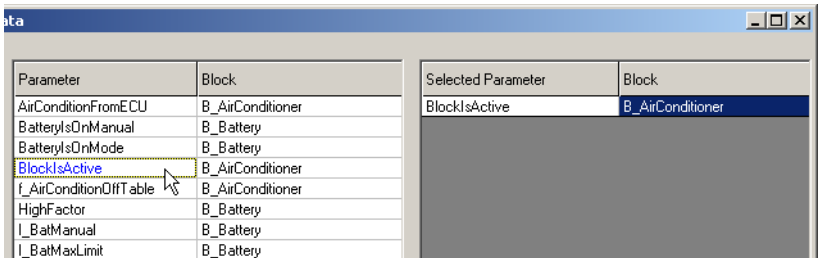
- パラメータを新しい M ファイル、DCM ファイル、mpa ファイルのいずれかにエクスポートするには、**Import/Export → Import/Export Parameter Data** を選択します。

  "Import/Export Parameter Data" ダイアログボックスが開きます。



ダイアログボックスの左側部分にはモデルのブロック構成リストが表示され、中央部には、ブロック構成リストの中で選択されたモデルコンポーネント（上図の場合：GEVM → VehicleModel → Components）に含まれるすべてのパラメータが表示されます。エクスポート対象として選択されたパラメータは、右側に表示されます。

たとえば、"B_AirConditioner" ブロックの "BlockIsActive" パラメータをエクスポートするには、まずブロック構成リスト内の "B_Battery" アイテムをクリックし、ダイアログボックス中央部に表示された "ConsumerActive" パラメータを選択します。すると、そのパラメータ名の文字色が青に変わり、同じパラメータがダイアログボックス右側の "Selected Parameters" 列にも表示されます。



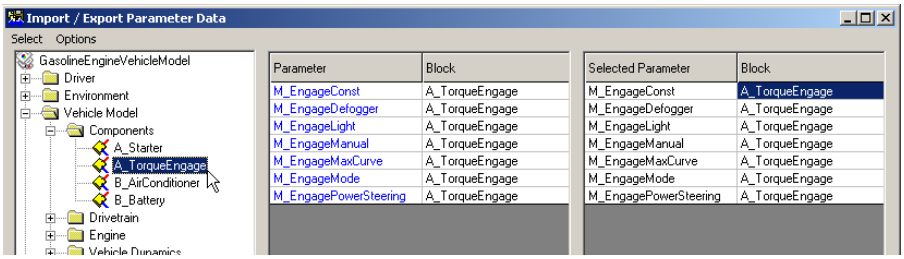
- "Selected Parameters" 列のパラメータ名をクリックすると、そのパラメータの選択が解除されます。

- あるブロックまたはモデルコンポーネント（たとえば、"A_TorqueEngage" ブロック）に含まれるすべてのパラメータを選択するには、まずブロック構成リスト内でこのブロックまたはモデルコンポーネントを選択します。
- **Select → Select All** を選択します。

または

- **<CTRL+A>** を押します。

  すると、すべてのパラメータが "Selected Parameters" のリストに追加されます。



- **Select → Deselect All** を選択すると、現在選択されているすべてのパラメータの選択が解除され、"Selected Parameters" 列から消去されます。
- パラメータの選択が終了したら、**Export Data** ボタンをクリックしてファイル選択ダイアログボックスを開きます。
- ファイル名を任意に入力するか、既存のファイルを選択します。
- ファイルのタイプを選択します。
- **Save** をクリックします。

  新しいファイルが作成されます。

  同名のファイルがすでに存在している場合、既存のファイルを上書きしてよいかを確認するメッセージが表示されます。上書きを行うと、それまで保存されていたファイルの内容はすべて失われます。

パラメータを既存のファイルにエクスポートする（"Replace Data"）：

- パラメータを既存の M ファイルまたは DCM ファイルにエクスポートする場合、上記と同様の手順でパラメータを選択します。
- **Export Data** ボタンの代わりに **Replace Data** ボタンをクリックします。

  エクスポートされるパラメータがファイル内にすでに存在する場合はその値が更新（上書き）され、存在しない場合はそのパラメータがファイルに追加されます。

パラメータのエクスポートに関するオプションを設定する：

メニューの **Options → Im-/Export Attributes in Matlab M-Files** を選択してこのオプションを有効（チェックマークあり）にしておくと、パラメータをエクスポートする際にパラメータの属性（"Min."、"Max."、"Modification State" など）もすべて一緒に書き込まれ、無効（チェックマークなし）にするとパラメータの値だけが書き込まれます。

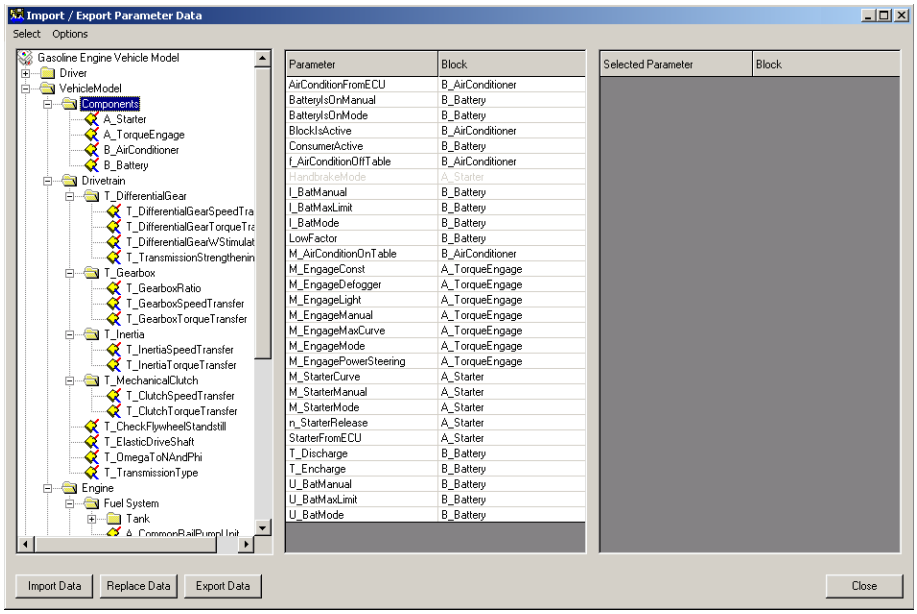
4.7.10 パラメータをファイルからインポートする

エクスポートと同様の方法で、パラメータをファイルからインポートすることができます。ただし、インポートしたいパラメータが指定のファイル内に存在しない場合はエラーメッセージが表示されます。

インポートするパラメータを選択する：

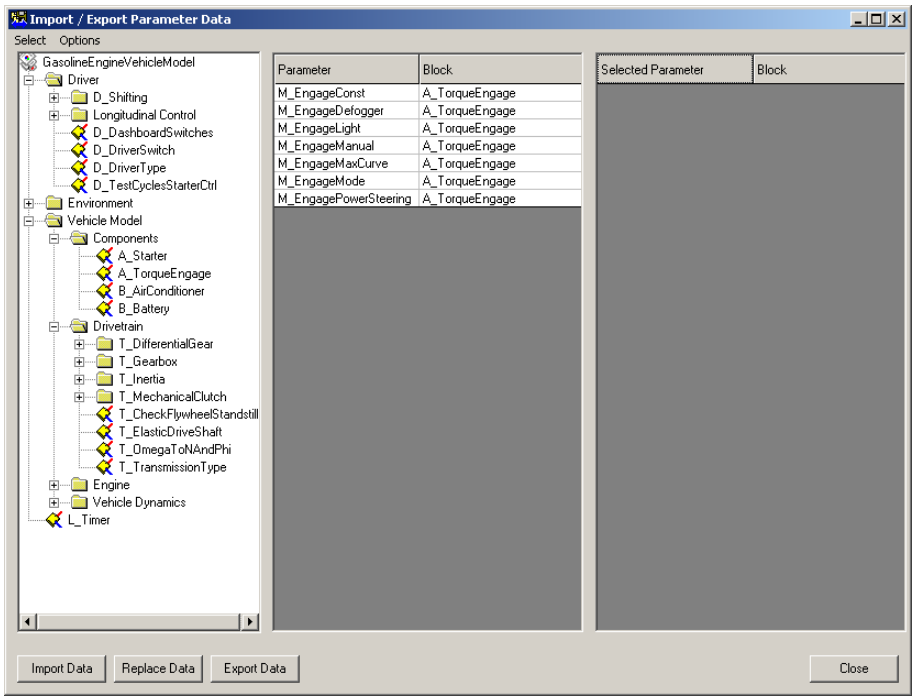
- パラメータを M ファイルまたは DCM ファイルからインポートするには、**Import/Export → Import/Export Parameter Data** を選択します。

  "Import/Export Parameter Data" ダイアログボックスが開きます。



- エクスポートの場合と同様、メニューの **Options → Im-/Export Attributes in Matlab M-File** オプションを切り替えて、インポート／エクスポートのオプションを変更することができます（319 ページの「パラメータのインポートとエクスポートに関するオプション」を参照してください）。

- たとえば、"A_TorqueEngage" ブロックのパラメータをインポートするには、まずブロック構成リストでこのブロックアイテムをクリックします。

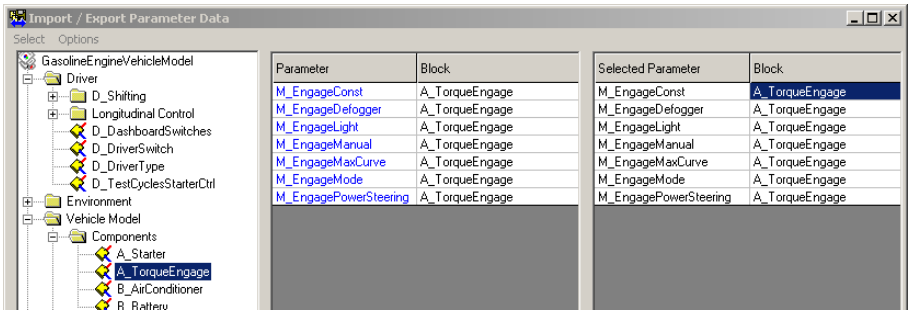


このブロックに含まれるパラメータのリストが、ダイアログボックス中央部に表形式で表示されます。

ここでは、スコープが "imported" のパラメータはグレイアウトされた状態で表示されます。

- **Select → Select All** を選択します。

選択されたパラメータ（ここでは "A_TorqueEngage" ブロックのすべてのパラメータ）が、"Selected Parameters" 列に表示されます。このリストに追加されたパラメータは、青色で表示されます。



ファイルを選択する：

- **Import Data** ボタンをクリックします。

ファイル選択ダイアログボックスが開きます。

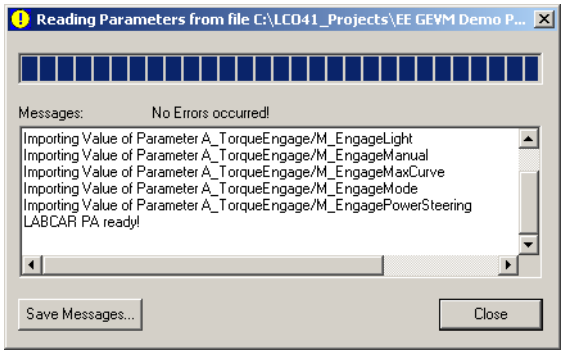
- ファイルのタイプを選択します。
- ファイル名を入力、または選択します。

- **Open** をクリックします。
  選択されたパラメータがインポートされます。

インポートが実行されると、選択されているパラメータの値は、インポート元のファイルの値に置き換えられますが、この際、以下の 2 とおりの状況が考えられます。

- 元の値とインポートされた新しい値とが等しい
- 元の値とインポートされた新しい値とが異なっている

上記の識別情報も "Import Parameters" ステータスウィンドウに出力されます。

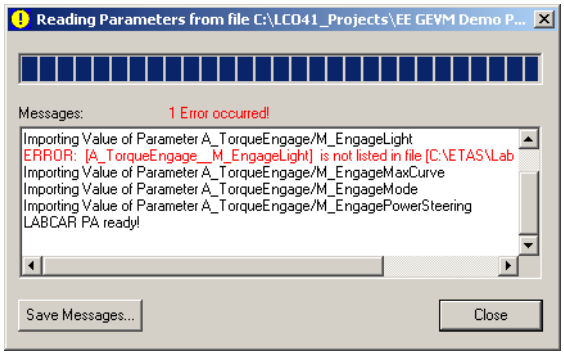


注記

これらのメッセージはエラーメッセージではないので、オプション設定（317 ページの「ステータスメッセージウィンドウの表示モードを指定する：」を参照してください）が有効になっていない場合は、インポートが完了した時点でこのステータスメッセージウィンドウは自動的に閉じます。

エラーメッセージ

選択されたパラメータがインポート元ファイルからインポートできない場合や、読み取り時に他のエラーが発生した場合は、エラーメッセージが表示されます。



これらのメッセージを ASCII ファイルまたは RTF ファイルに保存することもできます。

エラーメッセージを保存する

- **Save Errors** ボタンをクリックします。
  ファイル選択ダイアログボックスが開きます。
- ファイルのタイプ（ASCII または RTF）を選択します。
- ファイル名を入力します。

- **Save** をクリックします。

注記

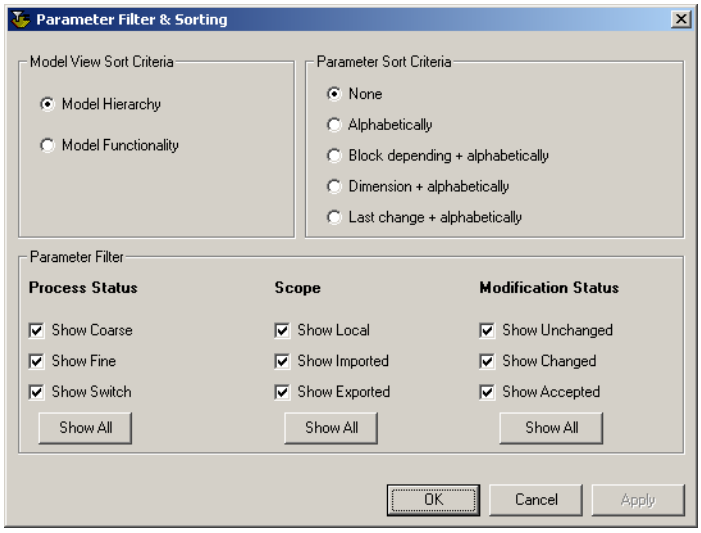
RTF(Rich Text Format）ファイルの場合、エラーメッセージは赤色で出力されます（ASCII ファイルでは不可能です)。

4.7.11 フィルタとソートの機能

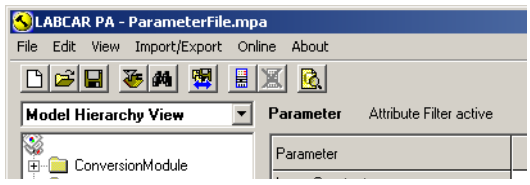
フィルタを用いて、パラメータテーブルに表示されるパラメータの数を制限することができます。たとえば、ステータスが "changed" のパラメータのみ、またスコープが "exported" のパラメータのみ表示されるように設定できます。

また、モデルのブロック構成リストやパラメータテーブル内のパラメータのソートに関するルールも指定できます。

この機能を使用するには、メインウィンドウで **Edit → Filter Settings** を選択して以下のダイアログボックスを開きます。



フィルタが有効になっている時は、メインウィンドウのパラメータテーブルの上に "Attribute Filter Active" と表示されます。



フィルタの種類とその意味について、以下に説明します。

Model View Sort Criteria

モデルに含まれる各モデルブロックをメインウィンドウ左側のフィールドにどのような形式で表示するかを指定します。

- **Model Hierarchy**
  モデルを階層構造に従って表示します。
- **Model Functionality**
  モデルを、機能単位（Driver、Engine など）で分類して表示します。

この形式は、メインウィンドウ左上のツールバーの下にあるリストボックスでも選択できます。

Parameter Sort Criteria

メインウィンドウ右側のパラメータテーブルに表示されるパラメータのソート基準を指定します。

- **None**
  ソートを行いません。パラメータはモデル内の各ブロック内での発生順序に従って表示されます（mpa ファイル内に格納されている順序と同じです）。
- **Alphabetically**
  パラメータ名を基準にして、アルファベット順にソートされます。
- **Block depending and alphabetically**
  まずブロック名でソートされ、同じブロック内のパラメータはパラメータ名でソートされます。
- **Dimension and alphabetically**
  まずパラメータのディメンション（2D テーブル、1D テーブル、スカラ、配列）でソートされ、同じディメンションを持つパラメータはパラメータ名でソートされます。
- **Last change and alphabetically**
  まず最終変更日でソートされ、同じ変更日のパラメータはパラメータ名でソートされます。

Parameter Filters

特定の条件に合うパラメータのみを表示することができます。

Process Status

- Show Coarse
  パラメータタイプ（"Proc. State"）が "coarse" であるパラメータを表示します。
- Show Fine
  パラメータタイプ（"Proc. State"）が "fine" であるパラメータを表示します。
- Show Switch
  パラメータタイプ（"Proc. State"）が "switch" であるパラメータを表示します。

Show All ボタンを用いると、上記の 3 つのフィルタオプションがすべて有効になり、実質的にはフィルタが非アクティブ状態となります。

Scope

- Show Local
  スコープが "local" であるパラメータを表示します。
- Show Imported
  スコープが "imported" であるパラメータを表示します。
- Show Exported
  スコープが "exported" であるパラメータを表示します。

Show All ボタンを用いると、上記の 3 つのフィルタオプションがすべて有効になり、実質的にはフィルタが非アクティブ状態となります。

Modification Status

- Show Unchanged

  変更ステータス（"Mod. Status"）が "unchanged" であるパラメータを表示します。
- Show Changed

  変更ステータス（"Mod. Status"）が "changed" であるパラメータを表示します。
- Show Accepted

  変更ステータス（"Mod. Status"）が "accepted" であるパラメータを表示します。

Show All ボタンを用いると、上記の 3 つのフィルタオプションがすべて有効になり、実質的にはフィルタが非アクティブ状態となります。

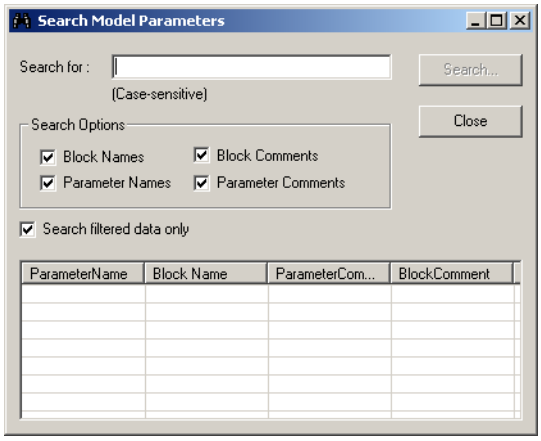
注記

1 つのグループ内の 3 つのフィルタオプションをすべて無効にしてしまうと、テーブルには何も表示されなくなることになり、他のグループのフィルタも無意味になっていまいます。
そのため、3 つのフィルタオプションをすべて無効にすると、実際にはすべてのパラメータが表示されるようになっています。

4.7.12 モデルパラメータ検索機能

検索機能を用いてモデル内のパラメータを検索することができます。

メニューコマンド **Edit → Search Model Parameters** または **<CTRL+F>** で、以下のパラメータ検索ダイアログボックスが開きます。



以下のオプションを利用して、検索対象とするパラメータを限定することができます。

Search Options

- **Block Names**

  指定した文字列が "Block Name" 属性内に含まれるパラメータを対象に検索が行われます。
- **Block Comments**

  指定した文字列が "Block Comment" 属性（マウスポインタがブロックを指したときにポップアップ表示される内容）内に含まれるパラメータを対象に検索が行われます。

- **Parameter Names**

  指定した文字列がパラメータ名の中に含まれるパラメータを対象に検索が行われます。

- **Parameter Comments**

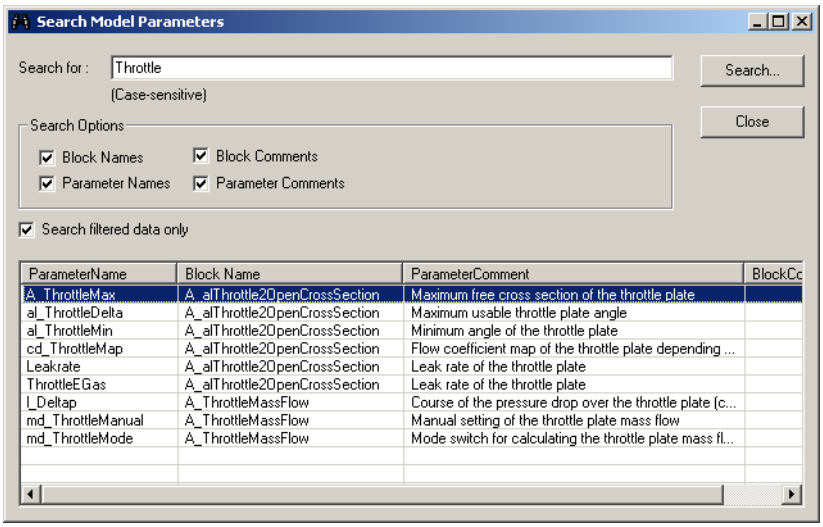
  指定した文字列が "Comment" 属性内に含まれるパラメータを対象に検索が行われます。

Search filtered data only

パラメータテーブルの表示フィルタが有効になっている場合、このオプションを指定すると、表示されているパラメータだけを対象に検索が行われます。

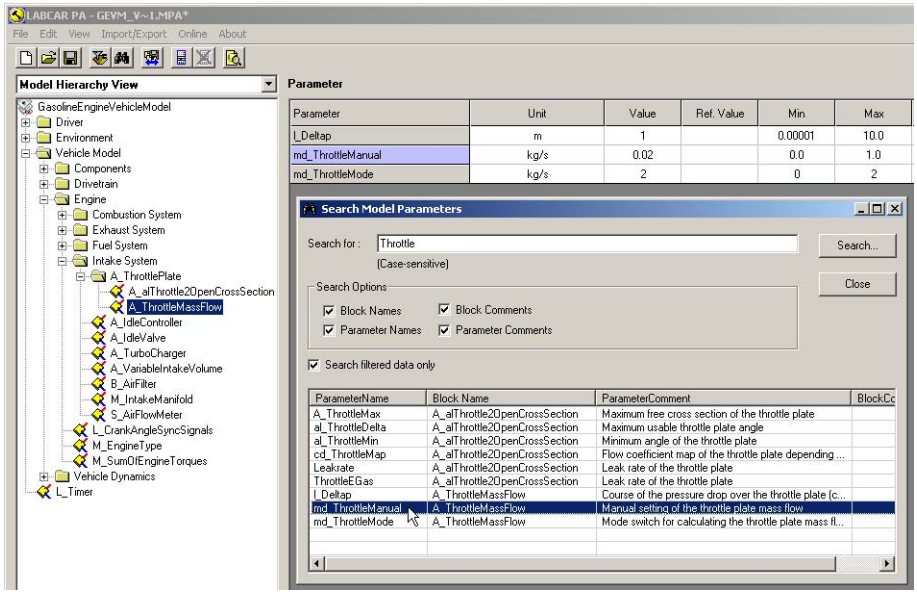
検索機能の活用例：

下の図は、"Throttle" という文字列を検索した結果です。ここではすべてのパラメータが検索対象となっています。



ここで、たとえばパラメータ名のみが検索対象となっていると、パラメータ "Leakrate" と "l_Deltap" は結果リストに現れません。

結果リスト内の任意のパラメータをクリックすると、メインウィンドウのブロック構成リストからそのパラメータが属するブロックが選択され、そのブロック内のすべてのパラメータがパラメータテーブルに表示されます。そして検索リスト内で選択されたパラメータは青色の背景で示されます。



4.7.13 パラメータスコープのバインディング

インポートされたパラメータの値は、エクスポート元のブロックでしか変更できません。このようなパラメータを変更する際は、スコープバインディングリストウィンドウを利用すると便利です。

このウィンドウを開くには、メニューコマンド **View → Parameter Scope Binding** を選択します。

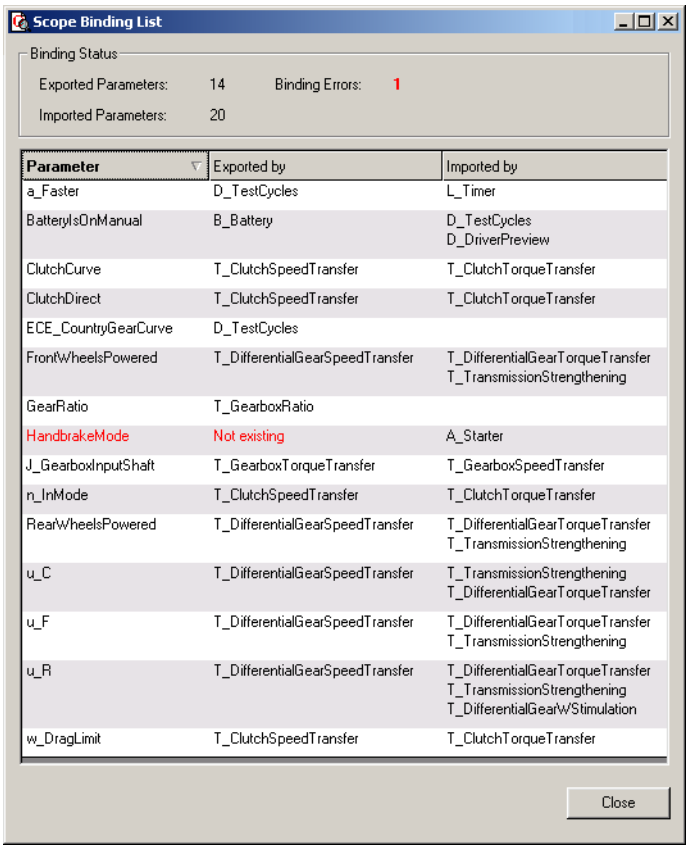


図 4-18 スコープバインディングリスト

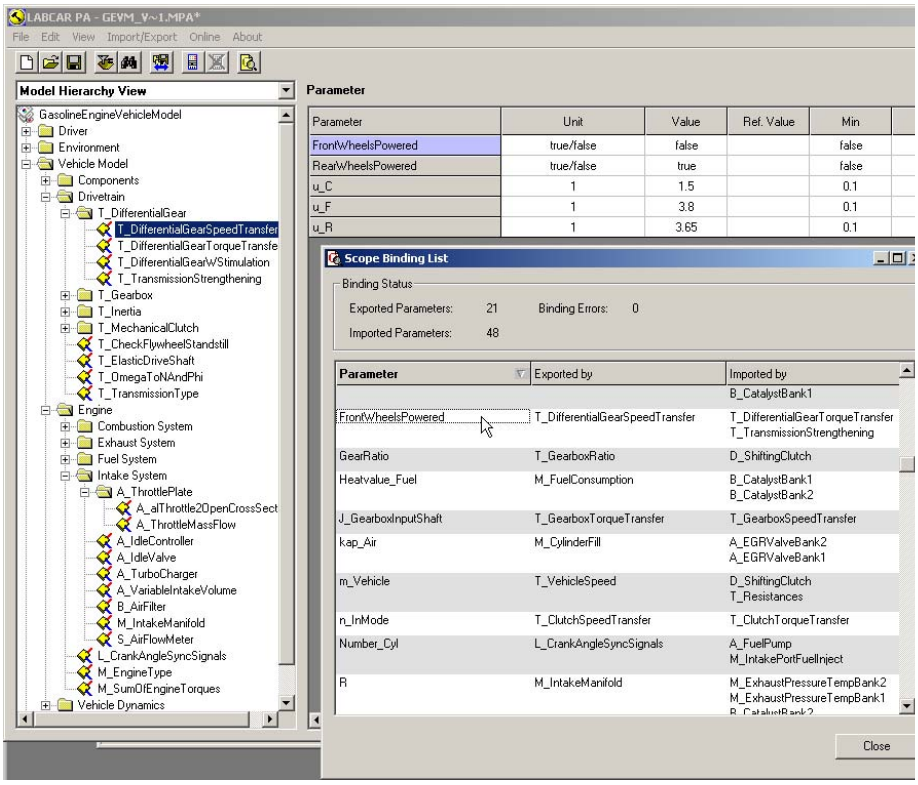
リスト内の "Parameter" 列には、1 つのブロック（"Exported by" 列のブロック）からエクスポートされて別のブロック（"Imported by" 列のブロック）にインポートされるすべてのパラメータが列挙されます。

パラメータの表示順は、パラメータ名で昇順または降順にソートできます（リストのヘッダ部分をクリックするたびに昇順／降順が切り替わります）。

337 ページの図 4-18 の例は、「スコープバインディング」に矛盾があるパラメータが表示されています。この例では、"Handbrake Mode" パラメータは "A_Starter" ブロックにインポートされていますが、どのブロックからもエクスポートされていません。そのため、このパラメータは赤の文字色で表示されています。

このような矛盾点は、実際にはファイルが開かれるときに自動的にチェックされ、ユーザーに対してすべての矛盾点が通知されます。

パラメータ名かエクスポート元のブロック名をクリックすると、メインウィンドウのブロック構成リスト内でそのブロックが選択され、このブロックに属するすべてのパラメータがパラメータテーブルに表示されます。エクスポートされたパラメータの背景は青色になります。



インポート先ブロック名をクリックした場合にも、同様のことが行われます。

4.7.14 オンラインでのパラメータ設定

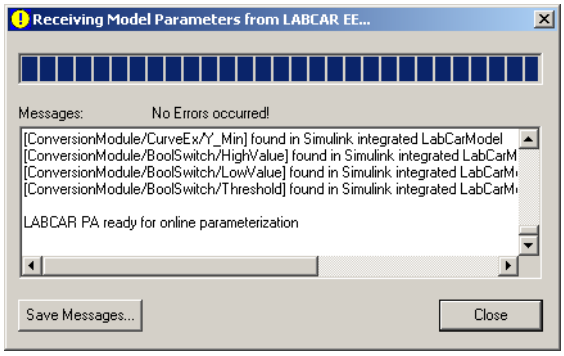
オンラインでパラメータ設定を行うと、LABCAR-PA 1.0 で変更されたパラメータの内容が LABCAR-EE で実行されている実験に直接書き込まれます。

オンラインでのパラメータ設定を開始する：

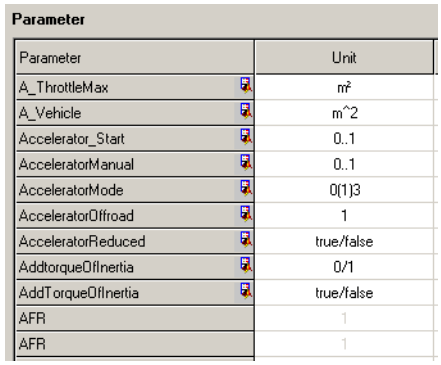
- LABCAR-EE で実験を起動します。
- LABCAR-PA でモデルのパラメータファイル（*.mpa）を開きます。

- **Online → Go Online** を選択します。

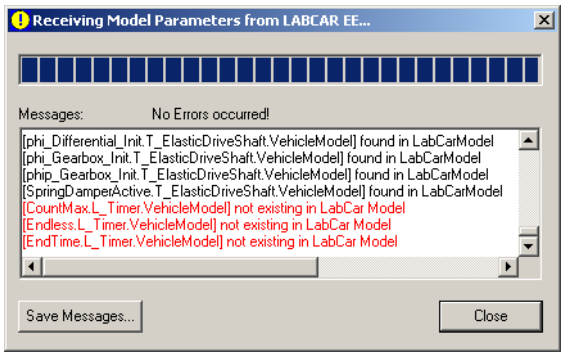
  実験環境への接続が確立され、パラメータファイルの中に含まれるパラメータが、LABCAR プロジェクト内で検索されます。



  見つかったパラメータには、パラメータテーブル内に "Online" アイコンが表示されます。



  見つからなかったパラメータがあると、エラーメッセージが表示されます。



- **Close** を選択します。

  現在実行されているプロジェクトにおけるモデルのパラメータの内容を、LABCAR-PA のパラメータテーブルで直接変更できるようになります。

パラメータが見つからない場合の処置

パラメータファイル内のパラメータが現在実行中のモデル内で見つからない場合、次の 2 つの理由が考えられます。

- ASAM-MCD-2MC ラベルが一致していない（おそらく Simulink モデルが別の名前でインポートされている、などの理由による）

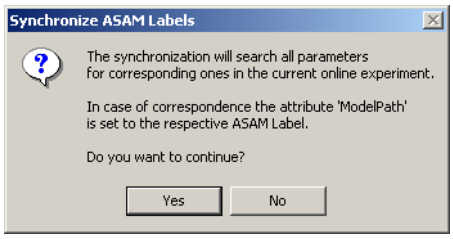
  この場合、ラベルの同期化（後述）により解決します。
- パラメータが現在実行中のモデル内に存在しない

上記の 1 番目の状況を解決するには、以下の手順でラベル名を同期化します。

ASAM-MCD-2MC ラベルを同期化させる

- **Online → Synchronize ASAM Labels** を選択します。

  現在実行中のプロジェクト内で、パラメタファイル内のパラメータに対応する ASAM-MCD-2MC ラベルの検索が行われます。

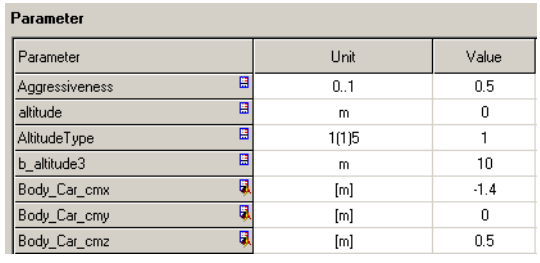


注記

同期化は、現在のモデル内の ASAM-MCD-2MC ラベルが、319 ページの「パラメータのインポートとエクスポートに関するオプション」で説明した規則に従って作成されたものである場合にのみ有効です。

対応するものが見つかると、パラメータファイル内の ASAM-MCD-2MC ラベル（"ModelPath" 属性）に、現在実行中のモデル内で使用されている名前が新たに割り当てられます。

同期化された ASAM-MCD-2MC ラベルがインポートされた Simulink モデルのものである場合、パラメータテーブル内においてこれらのパラメータには Simulink アイコンが付加された "Online" アイコンが表示されます。



4.8 LABCAR-CCI V5.4.0（"Calibration Connector for INCA" ― INCA 用適合コネクタ）

LABCAR-OPERATOR V5.4.0 のアドオンソフトウェアである LABCAR-CCI V5.4.0 は、ETK などの INCA デバイスにアクセスするための機能を提供します。つまり、INCA の一部の機能を LABCAR-OPERATOR から制御することを可能にするものです。実験環境（LABCAR-EE）の "Workspace Elementss" ウィンドウに、INCA の実験で使用する ASAM ラベルが表示されます。

下図は、LABCAR-CCI V5.4.0 が LABCAR-OPERATOR V5.4.0 の環境にどのように統合されているかを示したものです。

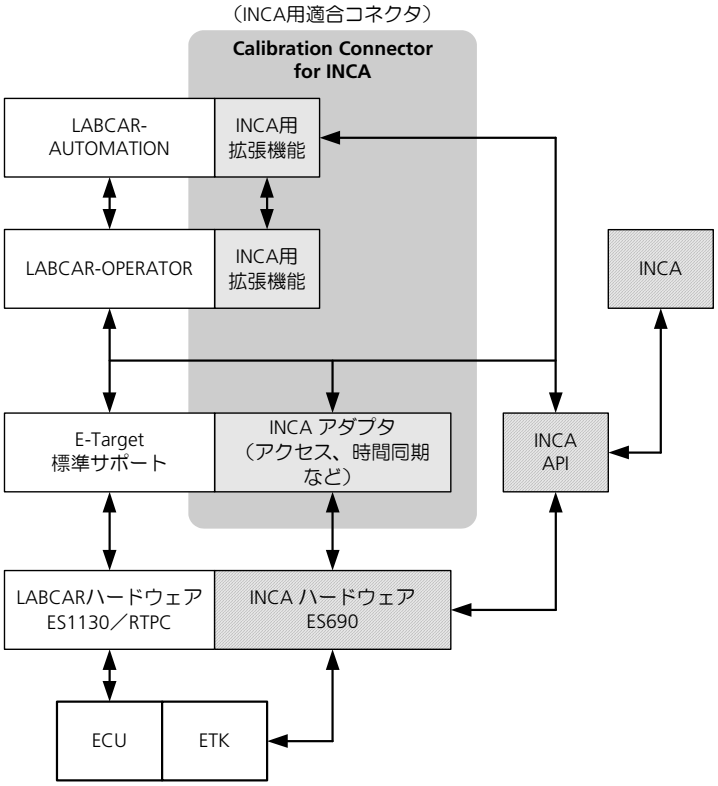


図 4-19 LABCAR-OPERATOR に統合された LABCAR-CCI V5.4.0

LABCAR-CCI V5.4.0 は、以下の 3 つの主要コンポーネントで構成されています。

- INCA ハードウェアによる測定や適合を行うために LABCAR-OPERATOR V5.4.0 内で使用される INCA アドオン機能

  INCA データベースを開いたり、INCA ハードウェアの初期化、測定の開始／停止などを行う拡張機能が含まれます。
- LABCAR-AUTOMATION 内で使用される INCA アドオン機能

  時間同期や実験ターゲット（"E-Target"、LABCAR では「シミュレーションターゲット」とも呼ばれます）などについての新しいアーキテクチャをサポートします。
- INCA ハードウェアへのアクセス管理や、実験ターゲットのタイムスタンプと INCA の測定値との同期を行うための、INCA 拡張機能

本章では、以下のトピックについて説明します。

- システム要件（342 ページ）

  LABCAR-CCI V5.4.0 を使用するためのソフトウェア要件とハードウェア要件について説明します。
- LABCAR-CCI V5.4.0 の操作方法（342 ページ）

  LABCAR-CCI V5.4.0 の操作方法について説明します。

4.8.1 システム要件

以下には、LABCAR-CCI V5.4.0 を使用するために必要なソフトウェアコンポーネントとハードウェアコンポーネントについて説明します。

INCA のバージョン

LABCAR-CCI V5.4.0 に対応する INCA のバージョンは、「LABCAR-OPERATOR 5.x.y - Software Compatibility List」というドキュメントに記載されています。このドキュメントには、**? → Help** からアクセスできます。

注記

INCA は LABCAR-CCI V5.4.0 のコンポーネントではないので、INCA ライセンスは別途ご購入いただく必要があります。INCA がインストールされていないと、LABCAR-CCI V5.4.0 を使用することはできません。

INCA は LABCAR-OPERATOR と同じ PC にインストールできます。

ハードウェア要件

ECU アクセスに関する各種設定は INCA の実験環境上で定義し、接続用デバイスとして、ES690 コンパクトモジュールの CAN、K-Line、ETK インターフェースなどを使用できます。

4.8.2 LABCAR-CCI V5.4.0 の操作方法

実験環境（LABCAR-EE）から、任意の INCA の実験にアクセスすることができます。

このためには、以下の手順を実行してください。

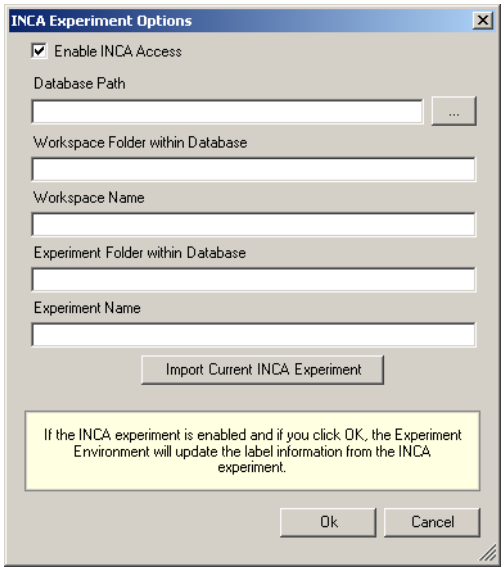
INCA の実験を指定する：

- LABCAR-EE を起動します。
- 必要に応じて、LABCAR-OPERATOR の実験をターゲットにダウンロードしてシミュレーションを起動します。この操作は後からでも行えます。
- **Experiment □INCA Options** コマンドを選択します。

  "INCA Experiment Options" ダイアログボックスが開きます。

- "Eneble INCA Access" オプションをオンにします。

  他の情報を指定するための複数のフィールドが有効になります。



- 以下の情報をマニュアル入力します。
  - Database Path：INCA データベースのパスを入力
  - Workspace Folder within Database：データベース内のワークスペースフォルダ
  - Workspace Name：ワークスペースの名前
  - Experiment Folder within Database：データベース内の実験フォルダ
  - Experiment Name：実験の名前

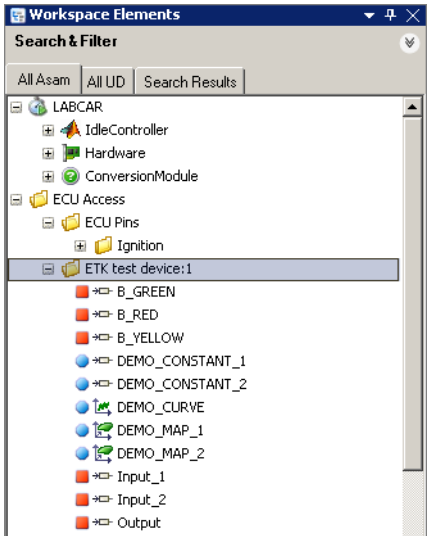
または

- INCA で実験を開き、"INCA Experiment Options" ダイアログボックスの **Import Current INCA Experiment** ボタンをクリックします。

  INCA の実験に関する情報が上記の各フィールドにインポートされます。

- **OK** をクリックします。

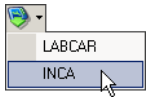
  INCA の実験への「接続」が自動的に確立され、実験で使用する測定変数とパラメータ（適合変数）を選択することが可能になります。



LABCAR-EE を保存すると（**File → Save Experiment**）、INCA の実験への接続設定も保存されます。

INCA の実験への再接続は、以下のように行います。

INCA の実験に接続する：



- **Experiment □Download □INCA** コマンドを選択します。

または

- ツールバーの左記のアイコンをクリックします。

  INCA 上で実験が開き、INCA の実験で使用されている ASAM ラベルに LABCAR-EE からアクセスすることが可能になります。

プロジェクトのダウンロード後、ハードウェアや ECU の電源がオンになっていなかったなどの理由により INCA のハードウェアがまだ初期化されていない場合、以下のようにしてハードウェアを初期化できます。

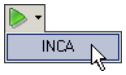
INCA ハードウェアを初期化する：

- **Experiment □Initialize INCA Hardware** コマンドを実行して、INCA ハードウェアを初期化します。

測定を開始する：

- **Experiment □Start Experiment □INCA** コマンドを実行します。

または



- ツールバーの左記のアイコンをクリックします。

  測定が開始されます。

測定を終了する：

- **Experiment □Stop Experiment □INCA** コマンドを実行します。

または

- ツールバーの左記のアイコンをクリックします。

  測定が開始されます。

実験への接続を切断する：

- INCA の実験への接続を切るには、**Experiment □ Disconnect □INCA** コマンドを実行します。

または

- ツールバーの左記のアイコンをクリックします。

  接続が切断されます。

5 付録

LABCAR-NIF V5.4.0 および LABCAR-LCX V5.4.0 は、以下のライセンスに基づいて StringTemplate と ANTLR ライブラリを使用しています。

StringTemplate のソフトウェアライセンス

The BSD License]

ANTLR のライセンス

6 お問い合わせ先

製品に関するご質問等は、各地域の ETAS 支社までお問い合わせください。

ETAS 本社

ETAS GmbH

Borsigstrasse 14
70469 Stuttgart
Germany

Phone: +49 711 3423-0
Fax: +49 711 3423-2106
WWW: www.etas.com/

日本支社

イータス株式会社

〒 220-6217
神奈川県横浜市西区
みなとみらい 2-3-5
クイーンズタワー C 17F

Phone: (045) 222-0900
Fax: (045) 222-0956
WWW: www.etas.com/

その他の支社

上記以外の各国支社および技術サポート窓口につきましては、ETAS ホームページをご覧ください。

各国支社 WWW: www.etas.com/ja/contact.php
技術サポート WWW: www.etas.com/ja/hotlines.php

図

索引

ま

め

り