RTI CAN MultiMessage Blockset

Tutorial

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2007 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Tutorial 9

Introduction to the RTI CAN MultiMessage Blockset 13

Basics on the RTI CAN MultiMessage Blockset..... 14
How to Access the RTI CAN MultiMessage Blockset..... 21
Basics on the RTICANMM MainBlock..... 22
Introduction to a Typical Use Scenario..... 24

Introduction to this Tutorial 27

Introduction to the Example Used in this Tutorial..... 28
Working With this Tutorial..... 30
Working with the Prepared Demo Results..... 32
Overview of Lessons..... 35

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset 39

Overview of Lesson 1..... 40
Step 1: How to Create a New Model and Add RTICANMM Setup Blocks to It..... 41
Step 2: How to Add RTICANMM MainBlocks and Configure CAN Communication..... 48
Step 3: How to Build an Application and Download It to Your Real-Time Hardware..... 56
Result of Lesson 1..... 59

Lesson 2: Displaying and Running the Simulation in ControlDesk 61

Overview of Lesson 2..... 61
Step 1: How to Create CAN Layouts in ControlDesk..... 63
Step 2: How to Trigger Message Transmission With CAN Layouts..... 68
Step 3: How to Monitor CAN Messages..... 72
Result of Lesson 2..... 79

Lesson 3: Triggering Message Transmission via the Model and ControlDesk 81

Overview of Lesson 3 81
Step 1: How to Enable or Disable Message Transmission 83
Step 2: How to Specify Cyclic Message Transmission or Transmission via Kickout 89
Step 3: How to Check Message Transmission Settings in ControlDesk 94
Result of Lesson 3 101

Lesson 4: Manipulating Signal Values via the Model and ControlDesk 103

Overview of Lesson 4 103
Step 1: How to Specify Signals to be Manipulated via the Model 105
Step 2: How to Enable Manipulation of TX Model Signals via ControlDesk 109
Result of Lesson 4 114

Lesson 5: Working with Model-Specific Bus Hierarchies 117

Overview of Lesson 5 118
Step 1: How to Add a Simulink Bus to Your Model 120
Step 2: How to Create an Inport for the Triggering to RTICANMM Block 122
Step 3: How to Create an Inport for the Mapping to RTICANMM Block 127
Step 4: How to Connect the Simulink Bus 130
Result of Lesson 5 132

Lesson 6: Performing a CAN Restbus Simulation 135

Overview of Lesson 6 136
Step 1: How to Simulate a "Real" ECU 137
Step 2: How to Simulate a Restbus 141
Step 3: How to Specify Counter Signals 145
Step 4: How to Specify Checksums 150
Step 5: How to Check the Restbus Simulation in ControlDesk 161
Result of Lesson 6 167

Lesson 7: Customizing Messages 169

Overview of Lesson 7 169
Step 1: How to Enable ID and Length Manipulation of TX Messages 171
Step 2: How to Enable Dynamic Message Manipulation Options 174
Step 3: How to Prepare the Detection of Checksum and Cycle Time Misbehavior 176
Step 4: How to Work with Customized Messages in ControlDesk 181
Result of Lesson 7 188

Lesson 8: Customizing Signals 191

Overview of Lesson 8 192
Step 1: How to Enable Dynamic Signal Values 193
Step 2: How to Enable Static Signal Error Values 195
Step 3: How to Specify the Default Signal Manipulation 197
Step 4: How to Prepare the Detection of Counter Signal Misbehavior 199
Step 5: How to Work with Customized Signals in ControlDesk 202
Result of Lesson 8 212

Lesson 9: Updating the Database 215

Overview of Lesson 9 215
Step 1: How to Update the Database and Check Automatic Effects on the Model 217
Step 2: How to Adapt the Model to the New DBC File Manually 220
Result of Lesson 9 223

Lesson 10: Working with Variants of a CAN Controller 225

Overview of Lesson 10 225
Step 1: How to Specify Variants of a CAN Controller 227
Step 2: How to Work with Variants of a CAN Controller in ControlDesk 235
Result of Lesson 10 239

Lesson 11: Working Without a Database 241

Overview of Lesson 11 241
Step 1: How to Use Free Raw Messages Instead of a Database 244
Step 2: How to Use Capture Messages 250
Result of Lesson 11 256

Lesson 12 (Advanced): Gatewaying Messages 259

Overview of Lesson 12 260
Step 1: How to Build a Model for Gatewaying 263
Step 2: How to Simulate "Real" ECUs for Gatewaying 270
Step 3: How to Build a Unidirectional Gateway Using Two RTICANMM MainBlocks 273
Step 4: How to Build a Bidirectional Gateway Using Two RTICANMM MainBlocks 287
Step 5: How to Gateway Messages Using the RTICANMM Gateway Block 296
Step 6: How to Check the Gateway Behavior via ControlDesk 304
Result of Lesson 12 311

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File 315

Overview of Lesson 13 316
Step 1: How to Build a Model for Using a J1939-Compliant DBC File 320
Step 2: How to Configure the CAN-J1939 Communication of Your Model 329
Step 3: How to Observe the Network Management in ControlDesk 349
Step 4: How to Observe the CAN-J1939 Communication in ControlDesk 357
Step 5: How to Gateway J1939 Messages Using the RTICANMM Gateway Block 363
Result of Lesson 13 369

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database 373

Overview of Lesson 14 373
Step 1: How to Build a Model for Using an M File as the Database 377
Step 2: How to Define CAN Communication in an M File 381
Step 3: How to Use an M File as the Database 388
Result of Lesson 14 395

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters 399

Overview of Lesson 15 400

Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters 401
Step 2: How to Check the Behavior of E2E Checksums in ControlDesk 417
Step 3: How to Add a Checksum Case to the E2E Protection Header File 421
Result of Lesson 15 426

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles 429

Overview of Lesson 16 430
Step 1: How to Work with Checksum Algorithms Based on E2E Protection According to AUTOSAR E2E Profiles 434
Step 2: How to Check the Behavior of Checksums Based on E2E Protection According to AUTOSAR E2E Profiles in ControlDesk 446
Step 3: How to Use Debug Messages in the Checksum Header File 451
Result of Lesson 16 454

Summary 457

Your Working Results 457
Further Examples 458

Glossary 459

Appendix 483

How to Create an Experiment in ControlDesk 483
Connecting CAN Channels for Working With this Tutorial 485

Index 487

About This Tutorial

Contents

This tutorial guides you through your first steps with the RTI CAN MultiMessage Blockset. It is divided into short lessons in which you learn typical procedures step-by-step. You should work through these lessons before using the more detailed RTI CAN MultiMessage Blockset Reference 🕮.

Target groups

This tutorial is primarily targeted at engineers who develop Simulink models with a large number of CAN messages.

Required knowledge

Knowledge in handling the PC, the Microsoft Windows operating system, MATLAB/Simulink, and CAN is assumed.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
⍰	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.

Symbol	Description
🕮	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Examples:

- Where you find terms such as `rti<XXXX>` replace them by the RTI platform support you are using, for example, `rti1007`.
- Where you find terms such as `<model>` or `<submodel>` in this document, replace them by the actual name of your model or submodel. For example, if the name of your Simulink model is `smd_1007_sl.slx` and you are asked to edit the `<model>_usr.c` file, you actually have to edit the `smd_1007_sl_usr.c` file.

RTI block name conventions All I/O blocks have default names based on dSPACE's board naming conventions:

- Most RTI block names start with the board name.
- A short description of functionality is added.
- Most RTI block names also have a suffix.

Suffix	Meaning
B	Board number (for PHS-bus-based systems)
M	Module number (for MicroAutoBox II)
C	Channel number
G	Group number
CON	Converter number
BL	Block number
P	Port number
I	Interrupt number

A suffix is followed by the appropriate number. For example, DS2201IN_B2_C14 represents a digital input block located on a DS2201 board. The suffix indicates board number 2 and channel number 14 of the block. For more general block naming, the numbers are replaced by variables (for example, DS2201IN_Bx_Cy).

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Introduction to the RTI CAN MultiMessage Blockset

Introduction

The RTI CAN MultiMessage Blockset allows you to handle a large number of CAN messages. You can control and manipulate all the CAN messages by a single Simulink block. Before you start working with the RTI CAN MultiMessage Blockset, you should be familiar with its basic principles.

Note

It is assumed that you are familiar with the contents of this section before you start working through the tutorial.

Where to go from here

Information in this section

Basics on the RTI CAN MultiMessage Blockset.....14
Provides an overview of the features of the RTI CAN MultiMessage Blockset.

How to Access the RTI CAN MultiMessage Blockset.....21
To access the blocks of the RTI CAN MultiMessage Blockset, you must open the block library via the MATLAB Command Window.

Basics on the RTICANMM MainBlock.....22
The RTICANMM MainBlock lets you initialize a CAN controller and specify its entire CAN communication.

Introduction to a Typical Use Scenario.....24
In a vehicle, you usually work with different bus systems. Each bus system consists of several ECUs which communicate with each other. Mostly, the ECUs of different bus systems communicate with each other, too.

Basics on the RTI CAN MultiMessage Blockset

Introduction

The RTI CAN MultiMessage Blockset is a Simulink blockset for efficient and dynamic handling of complex CAN setups in application scenarios for rapid control prototyping (RCP) and hardware-in-the-loop (HIL) simulation. Using the RTI CAN MultiMessage Blockset, you can easily handle a large number of CAN messages (more than 200). You can control all the incoming RX messages and outgoing TX messages of an entire CAN controller by a single Simulink block. This reduces the size of model files, and the time required for code generation and the build process.

Precondition for using the RTI CAN MultiMessage Blockset

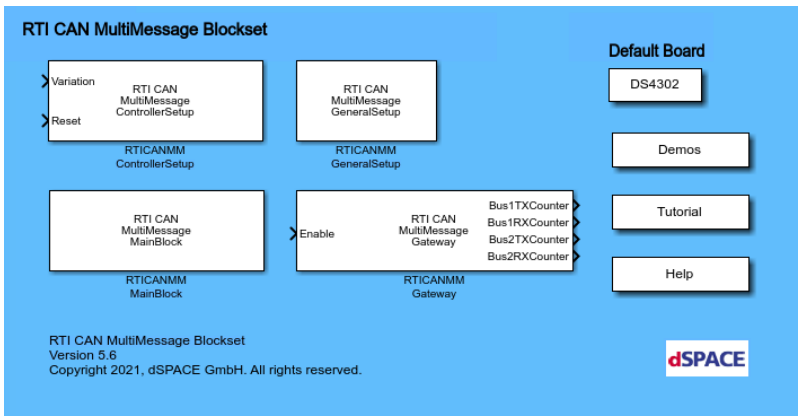
To work with the RTI CAN MultiMessage Blockset, you need a MEX compiler.

Only the MinGW and Microsoft Visual Studio 2017 Professional compilers are supported as MEX compilers. For details and limitations, refer to Required C and C++ Compilers (Installing dSPACE Software 🕮).

In MATLAB, the compiler has to be initialized via the `mex -setup` command.

Blocks of the RTI CAN MultiMessage Blockset

The RTI CAN MultiMessage Blockset consists of four blocks:



RTICANMM GeneralSetup block Using the RTICANMM GeneralSetup block, you can specify general settings for your model, for example, the destination folder for generated files. The settings of the RTICANMM GeneralSetup block apply to all other RTICANMM blocks. For this reason, there must always be one and only one RTICANMM GeneralSetup block in your model.

RTICANMM ControllerSetup block Using the RTICANMM ControllerSetup block, you initialize a CAN controller for your dSPACE board. There must be at

least one RTICANMM ControllerSetup block in each model. You have to use one RTICANMM ControllerSetup block for each CAN controller you want to specify.

RTICANMM MainBlock Using the RTICANMM MainBlock, you can set up and specify the entire CAN communication of your CAN controller. You can specify the settings of your CAN communication on various dialog pages.

RTICANMM Gateway block Using the RTICANMM Gateway block, you can exchange messages between two CAN buses. You can specify messages to be excluded from gatewaying and change the specified settings during run time. If you want to modify signals before gatewaying, you cannot use the RTICANMM Gateway block. Instead you must specify a gateway using at least two RTICANMM MainBlocks.

Further components of the RTI CAN MultiMessage library

Besides the RTI blocks, the RTI CAN MultiMessage library contains further components that let you easily change your default working board, access dSPACE Help, etc.

Default Board Lets you select the board type to be used as the default board. When you open a demo model from the library, it is automatically configured for your current default board. You can change the selected board type by double-clicking the button. If you then copy a demo, you are asked whether to automatically reconfigure the blocks copied from the library according to the new default board.

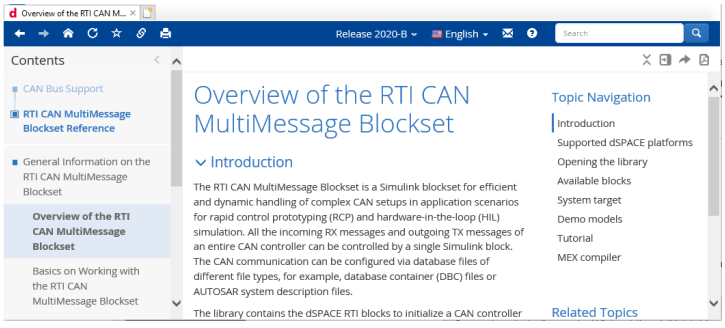
Demos Opens a sublibrary with the available Simulink demo models, which introduce you to some typical use scenarios of the RTI CAN MultiMessage Blockset.

Tip

For an easier understanding, you should first work through this tutorial before you use the demo models.

Tutorial Opens the rticanmm_tutorial_lib sublibrary, which provides access to the database files used in this tutorial and lets you open the tutorial demo results (refer to Working with the Prepared Demo Results on page 32), for example.

Help Provides access to dSPACE Help. dSPACE Help provides all the information needed to work with the RTI CAN MultiMessage Blockset.



For tips and tricks on working with dSPACE Help, click the button in dSPACE Help.

Supported dSPACE platforms

The RTI CAN MultiMessage Blockset is supported by the following dSPACE platforms:

- SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board
- MicroAutoBox II
- MicroLabBox
- PHS-bus-based systems (DS1006 or DS1007 modular systems) containing one of the following I/O boards:
  - DS2202 HIL I/O Board
  - DS2210 HIL I/O Board
  - DS2211 HIL I/O Board
  - DS4302 CAN Interface Board
  - DS4505 Interface Board, if the DS4505 is equipped with DS4342 CAN FD Interface Modules

The dSPACE platforms provide 1 ... 4 CAN controllers (exception: DS6342 provides 1 ... 8 CAN controllers). A CAN controller performs serial communication according to the CAN protocol. To use a dSPACE board with CAN bus interface, you must configure the CAN controller – called the CAN channel – according to the application.

Note

The RTI CAN MultiMessage Blockset is not supported by the MicroAutoBox III. If you work with the MicroAutoBox III, you must use the Bus Manager to implement CAN and CAN FD bus simulation.

Defining CAN communication

To define the CAN communication of a CAN controller, you can specify a DBC, MAT, FIBEX, or AUTOSAR system description file as the database file on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮). You can also define CAN communication without using a database file.

DBC file as the database The Data Base Container (DBC) file format was developed by Vector Informatik GmbH, Stuttgart, Germany. For the RTI CAN MultiMessage Blockset, you can use all the DBC files that pass the consistency check of Vector Informatik's CANdb++ Editor.

FIBEX file as the database The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. A FIBEX file usually describes

more than one bus system. You therefore have to select one of the available bus systems if you work with a FIBEX file as the database.

AUTOSAR system description file as the database You can use an AUTOSAR system description file as the database for CAN communication. AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership that aims to develop and establish an open standard for automotive electric/electronic (E/E) architectures.

AUTOSAR system description files are files of AUTOSAR XML file type that describe a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template.

An AUTOSAR system description file usually describes more than one bus system. You therefore have to select one of the available bus systems if you work with an AUTOSAR XML file as the database.

MAT file as the database You can also use the MAT file format as the database for CAN communication, or specify other database file formats as the database. You must convert your specific database files into the MAT file format for this purpose. Because the MAT file requires a particular structure, it must be generated by M-script.

Working without a database file If you want to work without a database file, you can use free raw messages and/or capture messages. These messages are independent of DBC and MAT files.

Changing database defaults When you work with a database file, you can change its default settings via the following dialog pages of the RTICANMM MainBlock:

- Cycle Time Defaults Page (RTICANMM MainBlock)
- Base/Update Time Page (RTICANMM MainBlock)
- TX Message Length Page (RTICANMM MainBlock)
- Message Defaults Page (RTICANMM MainBlock)
- Signal Defaults Page (RTICANMM MainBlock)
- Signal Ranges Page (RTICANMM MainBlock)
- Signal Errors Page (RTICANMM MainBlock)

Updating a model

The RTI CAN MultiMessage Blockset creates an S-function for the specified database file. You can easily update the CAN configuration of a model by replacing the database file and updating the S-function.

Tip

You do not have to recreate the S-function for the RTI CAN MultiMessage Blockset if you switch from one processor board to another, e.g., from a DS1006 to a DS1007, and vice versa.
When you switch to or from a MicroAutoBox II or MicroLabBox, you only have to recreate the ControllerSetup block. You also have to recreate the ControllerSetup block if you change the controller name.

Modifying model parameters during run time

Model parameters such as messages or signal values can be modified during run time either via model input or via the **Bus Navigator** in ControlDesk. For modifying model parameters via ControlDesk, a variable description file (TRC) is automatically generated each time you create an S-function for the RTICANMM MainBlock. The entries of the TRC file let you analyze received signals, change the values of signals to be transmitted, etc. In ControlDesk, you can access the settings specified in the TRC file via the model's system description file (SDF). The SDF file bundles all the TRC files and additional information for the application.

(Relevant only for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) For information on where to find the signals of the CAN bus in the TRC file, refer to Available TRC File Variable Entries and Their Locations in the TRC File (ConfigurationDesk Real-Time Implementation Guide 🕮).

Support of CAN FD protocol

The RTI CAN MultiMessage Blockset supports the classic CAN protocol, the non-ISO CAN FD protocol (which is the original CAN FD protocol from Bosch) and the ISO CAN FD protocol (according to the ISO 11898-1:2015 standard). The CAN FD protocols allow data rates higher than 1 Mbit/s and payloads of up to 64 bytes per message.

Keep in mind that the CAN FD protocols are supported only by dSPACE platforms equipped with a CAN FD-capable CAN controller.

For more information, refer to Basics on Working with CAN FD (RTI CAN MultiMessage Blockset Reference 🕮).

Support of AUTOSAR features

The RTI CAN MultiMessage Blockset supports miscellaneous AUTOSAR features, such as secure onboard communication and global time synchronization. For more information, refer to Aspects of Miscellaneous Supported AUTOSAR Features (RTI CAN MultiMessage Blockset Reference 🕮).

Triggering messages

You can specify each message defined in the database file as a TX message and/or RX message. You can specify triggering conditions for TX messages. The specified trigger conditions can be changed during run time.

Triggering TX message transmission You can specify different conditions to trigger the transmission of TX messages. For example, message transmission can be triggered cyclically or by an event.

You can also work with dynamic message triggering, for example, modify the cycle times and initiate a spontaneous transmission (interactive or model-based) during run time.

Triggering reactions to the reception of RX messages You can specify the reactions to receiving a specific RX message. One example of a reaction is the immediate transmission of a TX message.

Manipulating signals to be transmitted

The RTI CAN MultiMessage Blockset provides several options to manipulate the values of signals before they are transmitted. You can switch between them with the Bus Navigator in ControlDesk via entries in the generated TRC file. In addition, you can calculate checksum, parity, toggle, counter, and mode counter values.

User-defined variables

You can include user-defined variables in a TRC file in addition to the parameters of the RTI CAN MultiMessage Blockset.

Working with variants of CAN communication

You can work with different CAN communication variants on one CAN controller, and switch between them during run time. Only one variant for each CAN controller can be active at a time. In the Bus Navigator, the active variant is labeled when an application is running on the real-time hardware. An inactive variant is labeled . If you open layouts of an inactive variant, the headers of all the RX and TX layouts are red.

Defining free raw messages

You can define free raw messages as additional messages that are independent of the database file. Once they are defined, you can use them like standard database messages and specify various options, for example:

- Trigger options
- Ports and displays
- Message ID and length adjustable during run time

The following features are not supported:

- Checksum generation
- Custom signal manipulation

Capturing messages

You can process the raw data of messages whose IDs match a given filter via the capture messages feature. This can be a specific ID, a range of IDs, or even all IDs. Optionally, you can exclude the messages contained in the database file.

The captured messages can be made available as outports of the MainBlock or in the TRC file.

CAN partial networking

With CAN partial networking, you can set selected ECUs in a network to sleep mode or shut them down if they do not have to run continuously. Wake-up messages then activate specific ECUs as and when required, and for as long as required.

Note

Partial networking is possible for the following dSPACE real-time hardware:
- MicroAutoBox II equipped with the DS1513 I/O Board
- MicroLabBox
- dSPACE hardware that supports working with CAN FD messages and that is equipped with DS4342 CAN FD Interface Modules, such as:
  - PHS-bus-based systems (DS1006 or DS1007 modular systems) with DS4505 Interface Board
  - MicroAutoBox II variants with DS1507
  - MicroAutoBox II variants with DS1514

The RTI CAN MultiMessage Blockset lets you specify the CAN partial networking wake-up messages by filtering message IDs and message data.

The CAN transceiver of the dSPACE real-time hardware is switched to sleep mode via the real-time application. You can use partial networking messages to wake up dSPACE real-time hardware after its CAN transceiver is switched to sleep mode.

(Relevant for MicroAutoBox II only) If a MicroAutoBox II's CAN transceiver is woken up via a partial networking message, the MicroAutoBox II behaves as if it was powered up manually. Depending on where the real-time application is loaded (flash memory or RAM), the MicroAutoBox II starts the application or waits for further input.

For more information, refer to Partial Networking Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

Visualization with the Bus Navigator

The RTI CAN MultiMessage Blockset supports visualization with the Bus Navigator. Layouts/instruments are generated on demand and provide access to all CAN signals and all switches required to configure CAN communication during run time. You do not have to preconfigure layouts by hand.

CAN bus monitoring with the Bus Navigator

The RTI CAN MultiMessage Blockset supports CAN bus monitoring with the Bus Navigator. CAN bus monitoring means observing CAN messages that are transmitted on a CAN bus. CAN bus monitoring always uses a certain host service, depending on the dSPACE platform used. Refer to Main Page (RTICANMM GeneralSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

RTI CAN Blockset and RTI CAN MultiMessage Blockset

(Not relevant for SCALEXIO systems with a DS2671 Bus Board, DS2672 Bus Module, DS6301 CAN/LIN Board, DS6341 CAN Board, and/or DS6342 CAN Board) You can use the RTI CAN Blockset and the RTI CAN MultiMessage

Blockset in parallel for different CAN controllers. However, you cannot use the RTI CAN MultiMessage Blockset and the RTI CAN Blockset for the same CAN controller.

Related topics

Basics

Basics on the RTICANMM MainBlock.....22
Introduction to a Typical Use Scenario.....24

HowTos

How to Access the RTI CAN MultiMessage Blockset.....21

References

General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

How to Access the RTI CAN MultiMessage Blockset

Preconditions

- The RTI CAN MultiMessage Blockset is installed.
- MATLAB is open.

Method

To access the RTI CAN MultiMessage Blockset

1 Type `rticanmm` in your MATLAB Command Window.

Result

The RTI CAN MultiMessage Blockset opens.

Related topics

Basics

Basics on the RTI CAN MultiMessage Blockset.....14
Basics on the RTICANMM MainBlock.....22
Introduction to a Typical Use Scenario.....24

Basics on the RTICANMM MainBlock

Introduction

Using the RTICANMM MainBlock, you can initialize a CAN controller and specify the entire CAN communication for it. The RTICANMM MainBlock provides numerous dialog pages for this.

Main features of the RTICANMM MainBlock

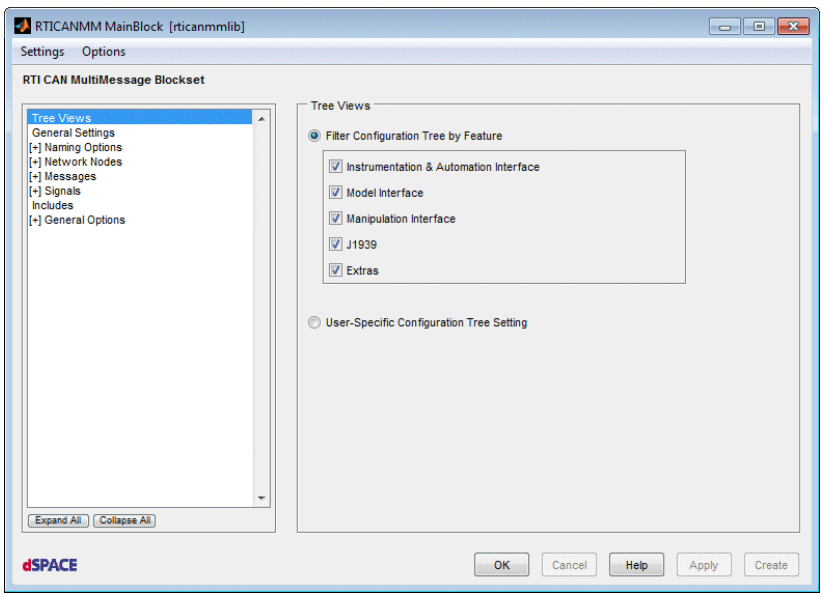
The main features of the RTICANMM MainBlock are:

- You can specify the database for your CAN communication.
- You can specify variants of a CAN controller. You can switch between the CAN controller variants during run time.
- You can specify TX and RX messages.
- You can specify the trigger conditions for TX messages.
- You can specify the manipulation options for TX messages.
- You can specify TX and RX signals.
- You can specify the manipulation options for TX signals.

Dialog tree and message view

The RTICANMM MainBlock has two views: The dialog tree and the message view.

Dialog tree The dialog tree is the default view when you double-click the RTICANMM MainBlock. All the dialog pages of the block are arranged in the dialog tree.

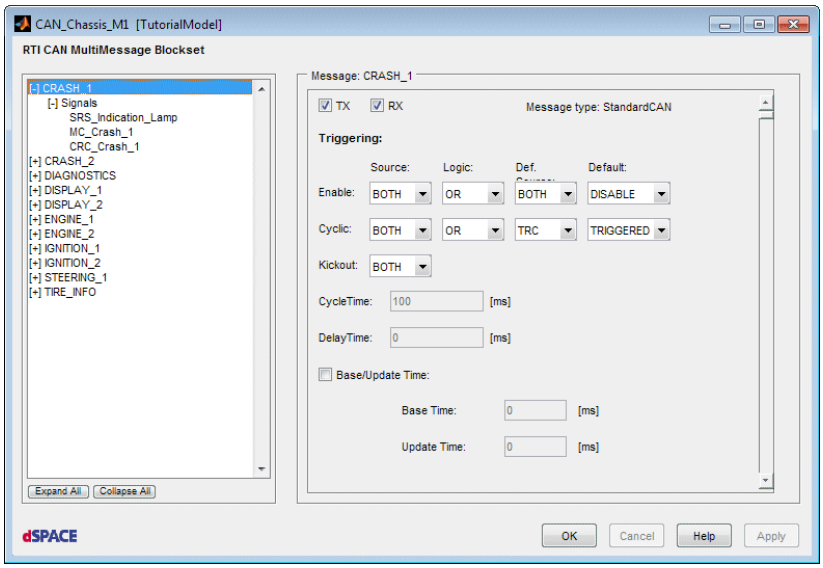


By default, all the dialog pages are collapsed. You can expand the dialog pages by clicking **Expand All**. You can customize the dialog tree on the Tree Views page. You can show or hide dialog pages by activating predefined filters or specify a user-defined dialog tree.

Note

If you change the filter settings, the parameters of the hidden pages are set to their default values.

Message view The message view is available only if you specify a database file for the RTICANMM MainBlock. The message view displays all the settings of a message or signal sent by an ECU. You can access the message view by clicking Settings - Open Settings Message View in the RTICANMM MainBlock dialog.



Unlike the dialog tree, to improve handling the message view lets you specify all the settings belonging to one message or signal on one page each. General settings of the RTICANMM MainBlock cannot be specified in the message view but only in the dialog tree.

Note

If you specify settings in the message view while the corresponding pages in the dialog tree are hidden, all the settings that relate to the hidden pages are reset to their defaults. For this reason, all the pages of the dialog tree must be displayed when you specify settings in the message view.

To change from the message view to the dialog tree, you have to click OK.

Code optimization and default settings

You can customize the available settings via the dialog pages and/or via the message view of the RTICANMM MainBlock. In most cases, customized settings enlarge the code to be built for the model. To guarantee best performance, you should change as few default settings as possible.

Saving settings of an RTICANMM MainBlock

Via the Settings menu, you can save all the settings of an RTICANMM MainBlock or just the settings of single property pages to a MAT file. You can

load the MAT file to another MainBlock to configure the block. The MAT file is independent of your model, for example, you can use it for a MainBlock of another model. If you use the MainBlock you saved the MAT file from and the MainBlock you loaded the MAT file to in the same model, you have to change the CAN name and the selected ControllerSetup block or the variation index (these settings have to be unique within a model).

Tip

Before you configure an RTICANMM MainBlock via a MAT file, select **User-Specific Configuration Tree Settings** on the Tree Views Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮). The settings specified on the Tree Views page are not loaded via the MAT file. By selecting the user-specific configuration, you avoid that settings of dialog pages not displayed in the minimum set are set to the default values.

Related topics

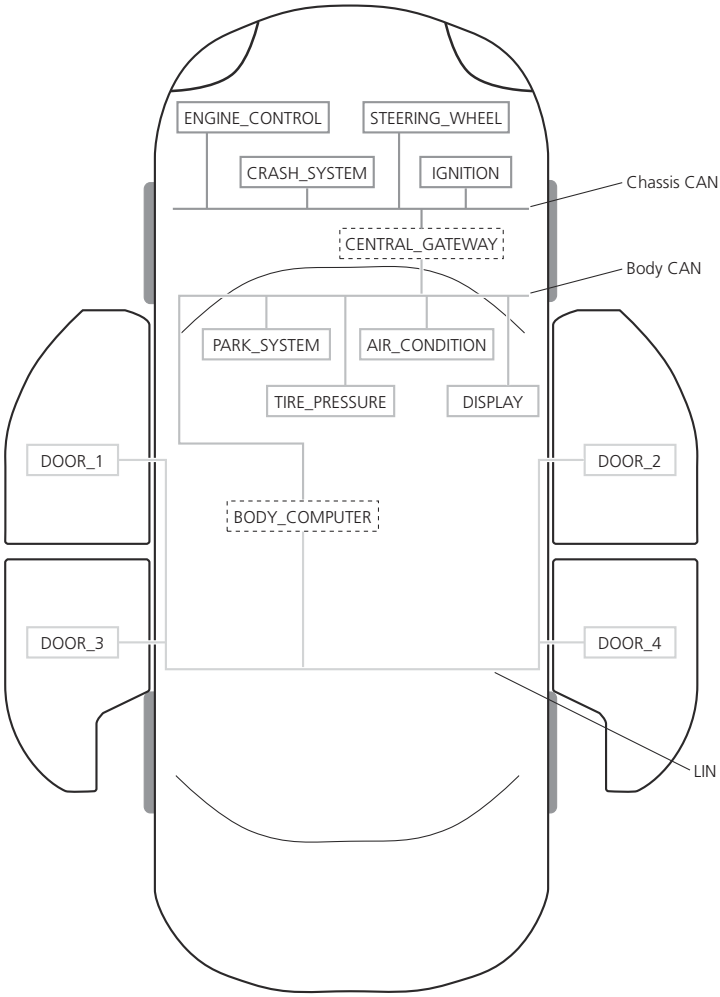
Basics

Basics on the RTI CAN MultiMessage Blockset 14

Introduction to a Typical Use Scenario

Overview

In a vehicle, you usually work with different bus systems, for example, CAN and LIN. Each bus system consists of several ECUs which communicate with each other. Mostly, the ECUs of different bus systems communicate with each other, too. The following illustration shows a simple example of different bus systems in a vehicle.



ECU communication Communication between ECUs is message-oriented. Each ECU transmits and/or receives messages. This often results in hundreds of messages to be transmitted via the bus.

ECU communication between different buses For exchanging messages between different bus systems, the messages must be gatewayed between the buses. Often, there are ECUs dedicated to gatewaying. In the example above, the CENTRAL_GATEWAY and BODY_COMPUTER ECUs gateway the messages between the Chassis CAN bus and Body CAN bus, and between the Body CAN bus and LIN bus.

Development of bus communication

To develop and test communication between different bus systems and between the ECUs of one bus, hardware-in-the-loop (HIL) applications are used. According to the development stage, one or more ECUs and their communication with other ECUs are simulated while real ECU(s) are connected to the HIL simulator. During simulation, the effects of the simulated ECU(s) and their communication with the real ECU(s) can be observed. Additionally, errors and misbehavior in

ECU communication can be simulated. The messages and signals of the simulated ECU(s) must be manipulated before transmission for this purpose.

Related topics

Basics

Basics on the RTI CAN MultiMessage Blockset..... 14
Basics on the RTICANMM MainBlock..... 22
Introduction to this Tutorial..... 27

HowTos

How to Access the RTI CAN MultiMessage Blockset..... 21

Introduction to this Tutorial

Introduction

So that you can work with this tutorial efficiently, you are introduced to the example used and to the structure of the tutorial. Additionally, you will find tips on how to work with the tutorial.

Note

It is assumed that you are familiar with the contents of this section before you start working through the tutorial.

Where to go from here

Information in this section

Introduction to the Example Used in this Tutorial.....28
To learn typical procedures using the RTI CAN MultiMessage Blockset, you will work with simple made-up CAN communications in this tutorial.

Working With this Tutorial.....30
This tutorial helps you to learn the basic steps with the RTI CAN MultiMessage Blockset. The tutorial is divided into several lessons to guide you. The lessons explain the individual tasks involved in working with the tutorial example.

Working with the Prepared Demo Results.....32
This tutorial provides demo results for each lesson. You can use these demo results as the starting point for working through a lesson or compare your working results with the demo result of a lesson.

Overview of Lessons.....35
Gives you an overview of the lessons contained in the RTI CAN MultiMessage Blockset Tutorial.

Introduction to the Example Used in this Tutorial

Overview

To learn typical procedures using the RTI CAN MultiMessage Blockset, you will work with simple made-up CAN communications in this tutorial. To simplify matters, the following conventions apply to the example:

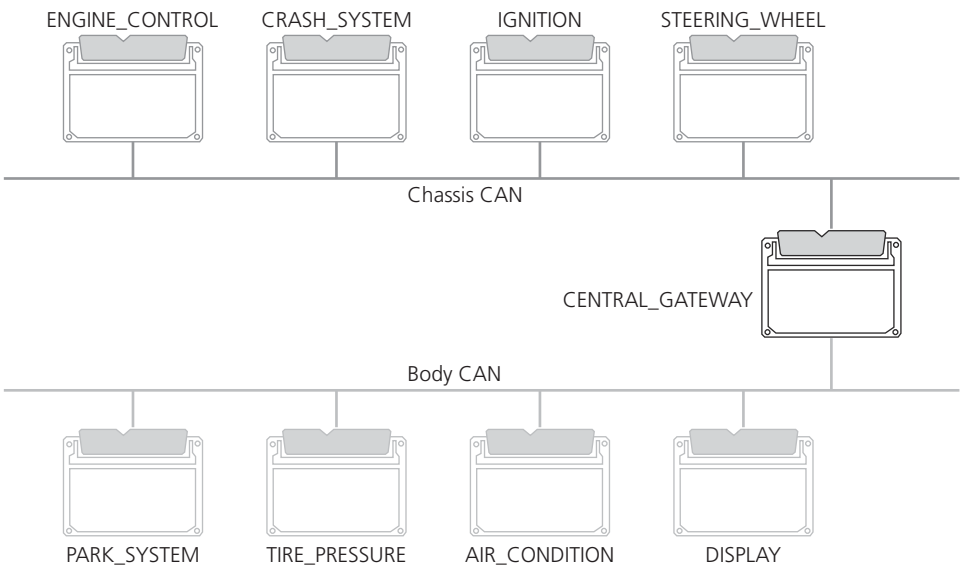
- You will work with only two CAN bus systems and nine ECUs.
- Real applications often consist of hundreds of messages, but in this tutorial you will work with only about 10 messages.

Note

The CAN communication defined for the CAN bus systems and their ECUs is only for training purposes. It does not represent real CAN communication defined for a vehicle. It is therefore recommended not to use the database files of this tutorial in real applications.

CAN bus systems and their ECUs

The illustration below shows the bus systems and their ECUs that you will work with in this tutorial.



Chassis CAN bus The Chassis CAN bus consists of four ECUs and the CENTRAL_GATEWAY ECU. The ECUs receive and transmit the following messages:

ECU	RX Messages	TX Messages
ENGINE_CONTROL	CRASH_2	ENGINE_1
	DIAGNOSTICS	ENGINE_2
	DISPLAY_1	
	DISPLAY_2	
	IGNITION_1	
	TIRE_INFO	
CRASH_SYSTEM	DIAGNOSTICS	CRASH_1
	DISPLAY_1	CRASH_2
	IGNITION_1	
	IGNITION_2	
	STEERING_1	
IGNITION	CRASH_2	IGNITION_1
	DIAGNOSTICS	IGNITION_2
	DISPLAY_1	
STEERING_WHEEL	DIAGNOSTICS	STEERING_1
	DISPLAY_1	
	IGNITION_1	
CENTRAL_GATEWAY	CRASH_1	DIAGNOSTICS
	CRASH_2	DISPLAY_1
	ENGINE_1	DISPLAY_2
	ENGINE_2	TIRE_INFO
	IGNITION_1	
	STEERING_1	

Body CAN bus The Body CAN bus consists of four ECUs and the CENTRAL_GATEWAY ECU. The ECUs receive and transmit the following messages:

ECU	RX Messages	TX Messages
PARK_SYSTEM	DIAGNOSTICS	PARK_1
TIRE_PRESSURE	DIAGNOSTICS	TIRE_INFO
	IGNITION_1	
AIR_CONDITION	DIAGNOSTICS	AIR_CONDITION_1
	IGNITION_1	AIR_CONDITION_2

ECU	RX Messages	TX Messages
DISPLAY	AIR_CONDITION_1	DISPLAY_1
	AIR_CONDITION_2	DISPLAY_2
	CRASH_1	
	DIAGNOSTICS	
	PARK_1	
	STEERING_1	
	TIRE_INFO	
CENTRAL_GATEWAY	DIAGNOSTICS	CRASH_1
	DISPLAY_1	DIAGNOSTICS
	DISPLAY_2	IGNITION_1
	TIRE_INFO	STEERING_1

Related topics

Basics

Overview of Lessons.....35
Working with the Prepared Demo Results.....32
Working With this Tutorial.....30

Working With this Tutorial

Introduction

This tutorial helps you to learn the basic steps with the RTI CAN MultiMessage Blockset.

Limitations

Your hardware must provide at least two CAN controllers for working through this tutorial. For lesson 12 and lesson 13, you will need four CAN controllers.

Preconditions

The following preconditions must be met for working through this tutorial:

Required software

- The RTI CAN MultiMessage Blockset is installed on your host PC.
- The J1939 support for the RTI CAN MultiMessage Blockset requires a separate license. This license must be available and activated if you want to work through Lesson 13 (Advanced): Working with a J1939-Compliant DBC File.
- ControlDesk is installed on your host PC.

Required knowledge

- You know the basics of
  - CAN communication (for details, refer to Basics on CAN).
  - DBC file handling (for details, refer to the documentation provided by Vector GmbH, Stuttgart, Germany).
  - TRC file handling (for details, refer to TRC file).
- You are familiar with
  - MATLAB/Simulink (for details, refer to the MATLAB/Simulink documentation).
  - Using the Real-Time Interface (RTI) for dSPACE systems (for details, refer to Working with RTI and RTI-MP (RTI and RTI-MP Implementation Guide)).
  - The graphical user interface of ControlDesk (for details, refer to User Interface of ControlDesk (ControlDesk User Interface Handling)).

Required hardware connections / CAN connections

- Your hardware is connected to your PC and registered (for details, refer to How to Register a Platform (ControlDesk Platform Management)).
- The CAN channels of your dSPACE real-time hardware are physically connected as follows:
  - Channel 1 is connected to channel 2
  - Channel 3 is connected to channel 4 (needed only for lessons 12 and 13)

  To establish the connection, you need information on the mapping of CAN controller numbers specified in RTI/RTLib to CAN signals and pins of dSPACE boards. Refer to Connecting CAN Channels for Working With this Tutorial on page 485.
- For the MicroAutoBox II, no termination resistance is installed. If you work with a MicroAutoBox II, you have to terminate it physically using a termination resistor (for details, refer to Setup Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference)).

Working with this tutorial

This tutorial is based on the example you were introduced to in the previous topic.

Lessons and steps The tutorial is divided into several lessons to guide you. The lessons explain the individual tasks involved in working with the tutorial example. To guide you through the tasks part by part, each lesson is divided into several steps.

Lesson results At the end of each lesson, you will find a lesson result. It summarizes the training contents of the lesson and gives further information. If you only want information on the basic principles of a lesson's training contents, you can use the lesson result as a reference.

Modular tutorial This tutorial is modular, i.e., you can work through a lesson without having completed the previous lessons. The result of each lesson is stored in the ready-made initial demo model, which you can use as the starting point for the next lesson.

Prepared demo

The results of each lesson are available as demo results. You can use these results in two ways:

- You can use the demo results as a starting point. Since most lessons build on previous lessons, you can work through a lesson using the result of the previous lesson as a starting point.
- You can compare your working results with the demo result of the lesson.

For details on how to work with the prepared demo results, refer to Working with the Prepared Demo Results on page 32.

Summary

After the lessons, you will find a summary of the entire tutorial. This contains working results and tips how to continue working with the RTI CAN MultiMessage Blockset. Refer to Summary on page 457.

Glossary

At the end of the tutorial you will find a glossary with brief explanations on the most important expressions used. Refer to Glossary on page 459. The individual lessons contain cross-references to the expressions, for example, TRC file.

Duration

Working through the entire tutorial will take you about 17 hours.

Related topics

Basics

Connecting CAN Channels for Working With this Tutorial 485
Introduction to the Example Used in this Tutorial 28
Overview of Lessons 35
Working with the Prepared Demo Results 32

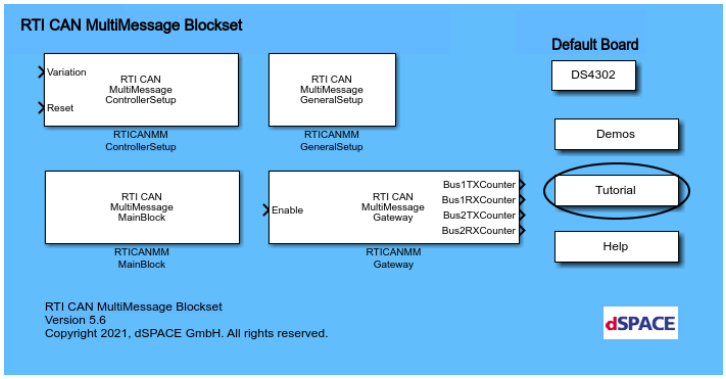
Working with the Prepared Demo Results

Introduction

This tutorial provides demo results for each lesson. You can use these demo results as the starting point for working through a lesson or compare your working results with the demo result of a lesson.

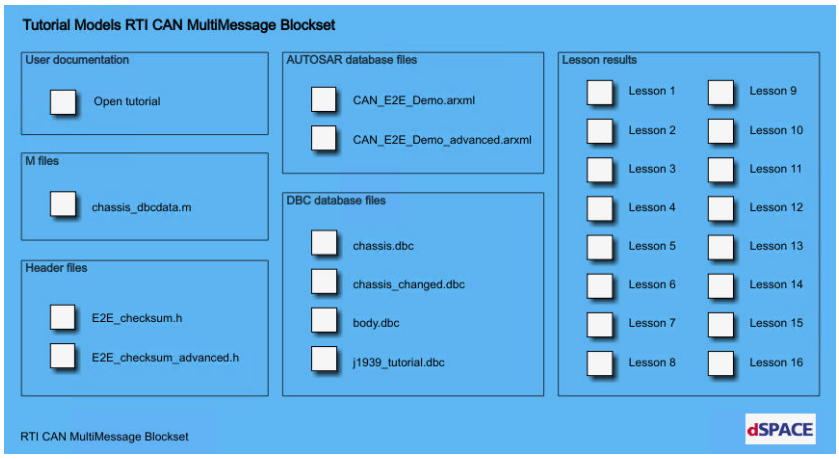
Accessing the demo results

In the RTI CAN MultiMessage Blockset, double-click Tutorial to open the rticanmm_tutorial_lib library.



Tip

You can access the rticanmm_tutorial_lib library directly by typing `rticanmm_tutorial` in the MATLAB Command Window.



Getting demo results for your specific real-time hardware

The settings of the RTICANMM blocks and the Simulink model depend on the real-time hardware used. For this reason, all the files generated during the create process of a MainBlock and/or during the build process are not available via the result folders. You have to create these files according to the real-time hardware you use.

Using the demo results as starting point

If you want to start a lesson using the demo model, perform the following steps:

1. If not already done, activate the RTI support of your dSPACE platform. To do so, type `rti<xxxx>` in the MATLAB Command Window. Replace `<xxxx>` with the platform you want to use, for example, `rti1007`.
2. In MATLAB, change the Current Directory to your working folder.
3. Open the rticanmm_tutorial_lib library.

4. Double-click the result folder of the previous lesson. For example, if you want to work through lesson 4, double-click **Lesson 3 Results**.
5. In the **Get tutorial** dialog, click **Copy tutorial**.

   The entire tutorial folder is copied to your working folder specified in the Current Directory.
6. Open the model.
7. On the **Simulation** ribbon, click **Prepare – Model settings** to open the **Configuration Parameters** dialog..

   Tip

   If you work with a DS1007 PPC Processor Board and a DS4302 CAN Interface Board, you can skip the following step. Continue with step 9.

8. On the **Code Generation** page, update the **System target file** according to your needs:

Platform	System Target File
DS1006	rti1006.tlc
DS1007	rti1007.tlc
MicroAutoBox II	rti1401.tlc
MicroLabBox	rti1202.tlc
SCALEXIO system	dsrt.tlc

9. On the **Solver** page, ensure that **Type** is set to 'Fixed-step' and **Solver** is set to 'ode1 (Euler)'.
10. In the model, update the real-time hardware specified in the RTICANMM ControllerSetup blocks according to your needs.
11. Double-click the RTICANMM GeneralSetup block in the model and select **Options - Create S-function for All CAN Blocks**.

    The S-function is created and you can start working with the model.

Comparing your working results with the demo result

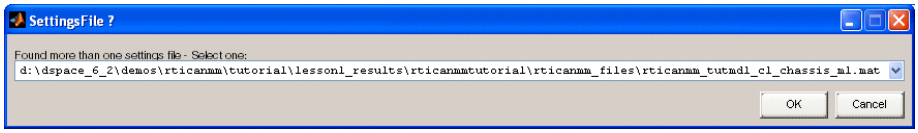
If you want to compare your results with the demo results, you can open the demo model in parallel with your model. Since the models have different names, they can be open at the same time without causing conflicts.

Note

Do not build an application for your model or for the demo model while the other one is open. Since the RTICANMM ControllerSetup blocks have identical names, the build process will fail.

To compare your results with the demo results, double-click **Lesson <number> Results** in the **rticanmm_tutorial_lib** library. In the **Get Tutorial** dialog, click **Only open model**.

The demo model opens and you can compare your model with the demo model. When you open one of the MainBlocks of the demo model, the **SettingsFile? RTICANMM** dialog opens.



Since your model and the demo model are open at the same time, MATLAB finds both settings files. Select the settings file located in your dSPACE root folder.

Related topics

Basics

Introduction to the Example Used in this Tutorial 28
Overview of Lessons 35
Working With this Tutorial 30

Overview of Lessons

Overview of lessons

This tutorial contains the following lessons:

- Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39

  You will build a Simulink model, set up CAN controllers, and define CAN communication using the RTI CAN MultiMessage Blockset. You will also build an application for your model and download it to the real-time hardware.
- Lesson 2: Displaying and Running the Simulation in ControlDesk on page 61

  You will create message layouts in ControlDesk using the **Bus Navigator**. You will then transmit, receive, and monitor CAN messages configured with the RTI CAN MultiMessage Blockset.
- Lesson 3: Triggering Message Transmission via the Model and ControlDesk on page 81

  You will trigger messages from within the model and/or via ControlDesk. You will use different triggering conditions and observe message behavior during run time.
- Lesson 4: Manipulating Signal Values via the Model and ControlDesk on page 103

  You will specify signals to be manipulated via the Simulink model only. Additionally, you will specify signals to be manipulated via the model and via ControlDesk.
- Lesson 5: Working with Model-Specific Bus Hierarchies on page 117

  You will connect a Simulink bus to your RTICANMM MainBlock.
- Lesson 6: Performing a CAN Restbus Simulation on page 135

  You will carry out a restbus simulation. You will simulate ECUs and specify checksum algorithms and counter signals for some messages. You will then check the behavior of the restbus simulation in ControlDesk.

- Lesson 7: Customizing Messages on page 169

  You will customize messages, for example, to simulate message errors. You will enable message ID and length manipulation, prepare the detection of checksum and cycle time misbehavior, and observe message behavior in ControlDesk.
- Lesson 8: Customizing Signals on page 191

  You will customize signals, for example, to simulate signal errors. You will work with static signal errors, prepare the detection of counter signal misbehavior, and observe signal behavior in ControlDesk.
- Lesson 9: Updating the Database on page 215

  You will update the database for the overall model, check automatic effects on the model, and adapt some model settings to the new database manually.
- Lesson 10: Working with Variants of a CAN Controller on page 225

  You will build a Simulink model and specify two database files for one CAN controller. These database files are variants of one CAN controller. You will switch between the variants in ControlDesk.
- Lesson 11: Working Without a Database on page 241

  You will work without a database but only with free raw messages and capture messages.
- Lesson 12 (Advanced): Gatewaying Messages on page 259

  You will gateway messages in two ways: via two RTICANMM MainBlocks and via an RTICANMM Gateway block, depending on whether or not the messages are to be manipulated before being gatewayed. Additionally, you will build a unidirectional and a bidirectional gateway using two RTICANMM MainBlocks.

  Tip

  If you do not need to gateway messages, you can skip this lesson.

- Lesson 13 (Advanced): Working with a J1939-Compliant DBC File on page 315

  You will work with J1939 messages and build a gateway using the RTICANMM Gateway block. You will observe the CAN-J1939 communication of your model via ControlDesk.

  Tip

  If you do not need to work with a J1939-compliant DBC file, you can skip this lesson.

- Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database on page 373

  You will use an M file to define the CAN communication of your model. Using the M file, you will generate a MAT file and specify it as the database for the RTICANMM MainBlocks of your model.

  Tip

  If you do not need to define CAN communication in an M file, you can skip this lesson.

- Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters on page 399

  You will implement checksum algorithms based on end-to-end communication (E2E protection) parameters. You will apply the checksum algorithms to signals and observe the behavior of the checksums in ControlDesk.

  Tip

  If you do not need to work with checksum algorithms based on E2E protection parameters, you can skip this lesson.

- Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles on page 429

  You will work with checksum algorithms based on E2E communication protection mechanisms according to AUTOSAR E2E profiles. You will apply the checksum algorithms to signals and observe the behavior of the checksums in ControlDesk using the example of profile 05 and profile 11 (including the profile variants 11A, 11C).

  Tip

  If you do not need to work with checksums based on E2E protection data according to AUTOSAR E2E profiles, you can skip this lesson.

Where to go from here

You can now start to build your first model using the RTI CAN MultiMessage blocks. Refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39.

Related topics

Basics

Introduction to the Example Used in this Tutorial 28
Working with the Prepared Demo Results 32
Working With this Tutorial 30

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset

Where to go from here

Information in this section

Overview of Lesson 1 40
In this lesson, you will create a model with blocks from the RTI CAN MultiMessage Blockset, and build the application for the model.

Step 1: How to Create a New Model and Add RTICANMM Setup Blocks to It 41
In this step, you will create a new model, and add one RTICANMM GeneralSetup block and two RTICANMM ControllerSetup blocks to the model.

Step 2: How to Add RTICANMM MainBlocks and Configure CAN Communication 48
In this step, you will add RTICANMM MainBlocks to your model, specify a database for them and configure CAN communication.

Step 3: How to Build an Application and Download It to Your Real-Time Hardware 56
In this step, you will learn how to build an application for your model and download it to your real-time hardware.

Result of Lesson 1 59
In this lesson, you created a model with blocks from the RTI CAN MultiMessage Blockset and built the application for the model.

Overview of Lesson 1

Introduction

Creating a model with blocks from the RTI CAN MultiMessage Blockset, and building the application for the model, differ from how models without these blocks are handled.

- You should save your model before you add RTICANMM blocks to it.
- You have to add the RTICANMM blocks to your model in a certain sequence.
- You have to let S-functions be created for the RTICANMM blocks before you can build an application for the model.

What you will learn

In this lesson you will learn how to create an RTICANMM model with a small CAN communication. Additionally, you will learn how to build an application for the model.

- You will create a new model and save it.
- You will add RTICANMM setup blocks to your model.
- You will add RTICANMM MainBlocks to your model.
- You will specify two CAN controllers: one transmitting and receiving messages, one only receiving messages. Both CAN controllers will be connected to the Chassis CAN bus of the tutorial example (refer to Introduction to the Example Used in this Tutorial on page 28).
- You will let S-functions be created for the RTICANMM ControllerSetup blocks and for the RTICANMM MainBlocks.
- You will build an application and download it to your real-time hardware.

Before you begin

You should be familiar with the basic principles of the RTI CAN MultiMessage Blockset. For details, refer to Introduction to the RTI CAN MultiMessage Blockset on page 13.

You should be familiar with the example used in this tutorial and with the limitations and preconditions that must be met to work through this tutorial. For details refer to Introduction to this Tutorial on page 27.

Before you start this lesson, open MATLAB. If you work with more than one dSPACE platform, specify which platform should be activated (for details, refer to How to Activate a Specific Platform Support (RTI and RTI-MP Implementation Guide 🕮)).

Steps

This lesson contains the following steps:

- Step 1: How to Create a New Model and Add RTICANMM Setup Blocks to It on page 41
- Step 2: How to Add RTICANMM MainBlocks and Configure CAN Communication on page 48
- Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 1 on page 59.

Duration

Working through this lesson will take you about 40 minutes.

What's next

Now you can start to create your first RTICANMM model, see Step 1: How to Create a New Model and Add RTICANMM Setup Blocks to It on page 41.

Related topics

Basics

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset..... 39

Step 1: How to Create a New Model and Add RTICANMM Setup Blocks to It

Objective

Creating a model using RTICANMM blocks differs slightly from creating a model using common Simulink blocks. Using the RTICANMM blockset, you have to run through the following sequence to create a model:

- You have to save your model before you add the first RTICANMM block to it. This is important because all the files that are created by the RTICANMM blockset are stored relative to the model path by default.
- The first RTICANMM block you add to your model must be an RTICANMM GeneralSetup block. If you saved your model already, you do not have to open this block.
- After adding the RTICANMM GeneralSetup block to your model, you have to add one or more RTICANMM ControllerSetup blocks. You have to specify the CAN controller settings and let an S-function be created.

In this step you will learn how to perform this sequence.

Part 1

To create a new model and open the RTI CAN MultiMessage Blockset

1 Create a new folder named `RTICANMMTutorial` in your working folder.

Tip

Save all the files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all the files to one folder also means that you have to change the path of the Current Directory in MATLAB.

2 In MATLAB, change the Current Directory to your RTICANMMTutorial folder.

3 On the Home ribbon, click Simulink – Simulink. The Simulink Start Page window opens.

4 If the real-time hardware you want to generate code for is listed under My Templates, click the corresponding icon to create an untitled model that is configured with the platform's parameters.

If your real-time hardware is not listed, click Blank Model to create an unconfigured untitled model.

5 On the Simulation ribbon, click Prepare – Model settings to open the Configuration Parameters dialog..

6 On the Solver page, ensure that Type is set to 'Fixed-step' and Solver is set to 'ode1 (Euler)'.

Tip

If you selected your platform from My Templates you can skip the next step.

7 On the Code Generation page, update the System target file according to your needs:

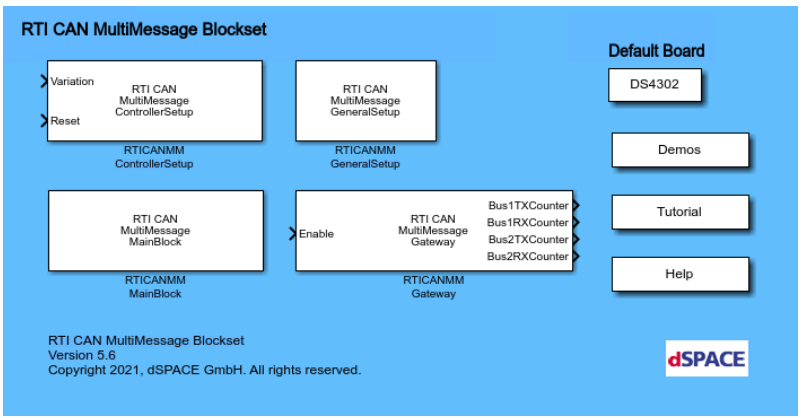
Platform	System Target File
DS1006	rti1006.tlc
DS1007	rti1007.tlc
MicroAutoBox II	rti1401.tlc
MicroLabBox	rti1202.tlc
SCALEXIO system	dsrt.tlc

8 Click OK to apply your changes and to close the dialog.

9 On the Simulation ribbon, click File – Save – Save As. In the Save As dialog, name the model `TutorialModel` and save it to your RTICANMMTutorial folder

10 In the MATLAB Command Window, type `rticanmm`.

The rticanmmlib library opens.



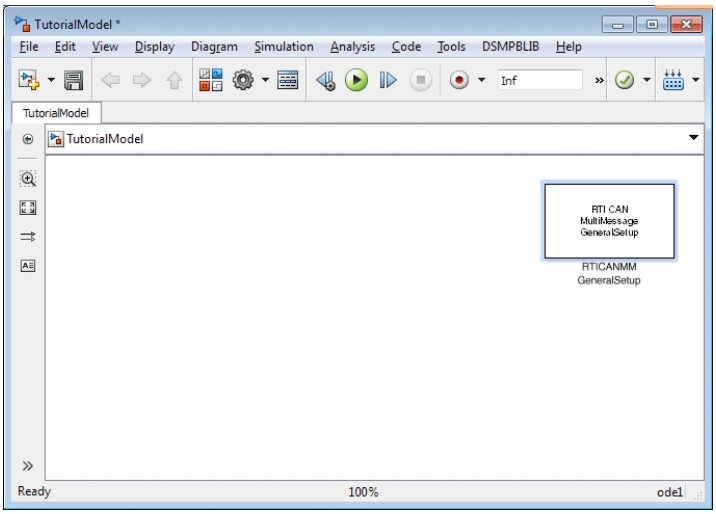
Interim result

You created a new model and opened the rticanmmlib library. You can now add an RTICANMM GeneralSetup block to your model. Continue with part 2.

Part 2

To add an RTICANMM GeneralSetup block to your model

1 In the rticanmmlib library, select the RTICANMM GeneralSetup block and drag it to your model.



The RTICANMM GeneralSetup block lets you specify general settings for your model. For example, you can specify where the files created for your RTICANMM model should be saved.

Note

The settings specified in the RTICANMM GeneralSetup block apply to all the RTICANMM blocks of your model. For this reason, you must always add one and only one RTICANMM GeneralSetup block to your model.

Since you saved your model already and specified your RTICANMMTutorial folder as the Current Directory, you do not have to make any changes in the RTICANMM GeneralSetup block.

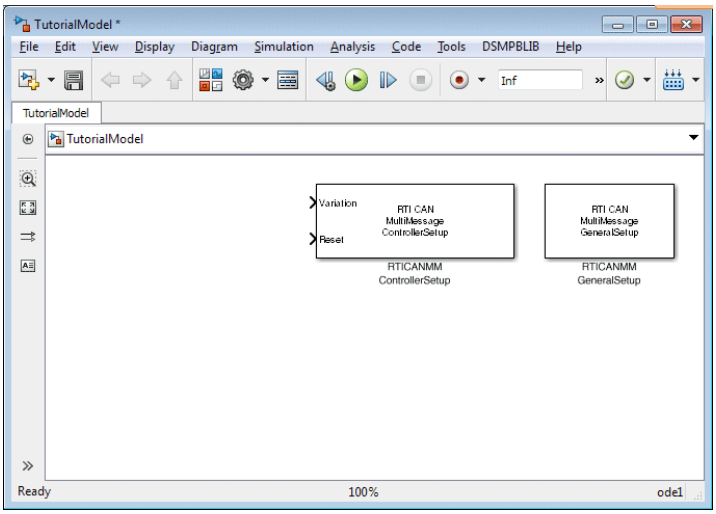
Interim result

You added an RTICANMM GeneralSetup block to your model. You can now add two RTICANMM ControllerSetup blocks to your model, one block for each CAN controller. Continue with part 3.

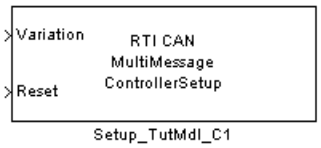
Part 3

To add RTICANMM ControllerSetup blocks to your model

1 In the rticanmmlib library, select the RTICANMM ControllerSetup block and drag it to your model.



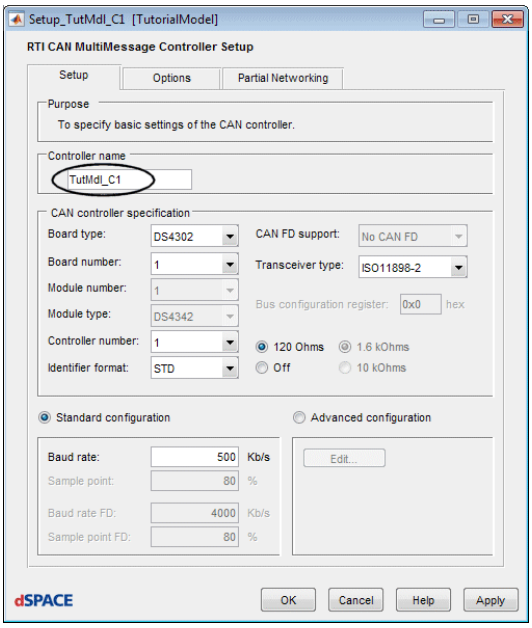
2 In your model, name the RTICANMM ControllerSetup block **`Setup_TutMdl_C1`**.



3 Double-click the Setup_TutMdl_C1 block to open it.

The Setup_TutMdl_C1 [TutorialModel] dialog opens.

4 In the **Controller name** frame, change the controller name to **`TutMdl_C1`**.



Note

The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.

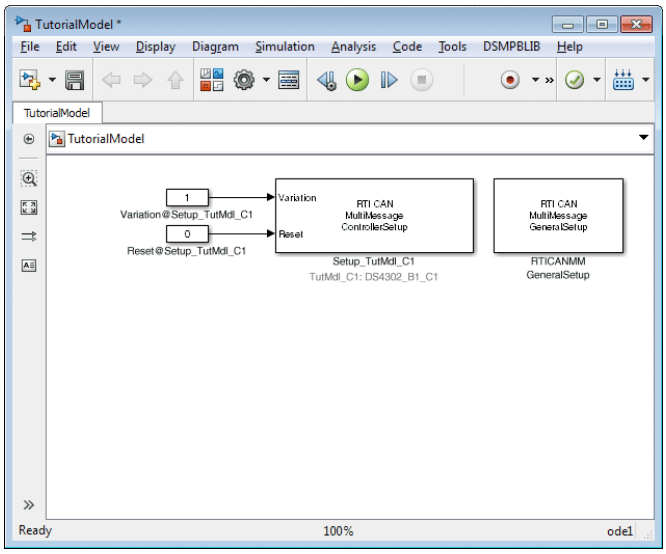
5 From the **Board type** drop-down list, select the board you are working with.

6 Click **OK**.

The RTICANMM ControllerSetup block dialog is closed, with the following effects:

- The block is parameterized and the CAN controller can be used for CAN communication with other RTICANMM blocks.
- An S-function is automatically created for the block.
- Two Simulink Constant blocks are automatically connected to the **Variation** and the **Reset** inports of the block.

- The controller name, board type, board number and controller number are displayed in your model.



7 From the rticanmmlib library, add a second RTICANMM ControllerSetup block to your model.

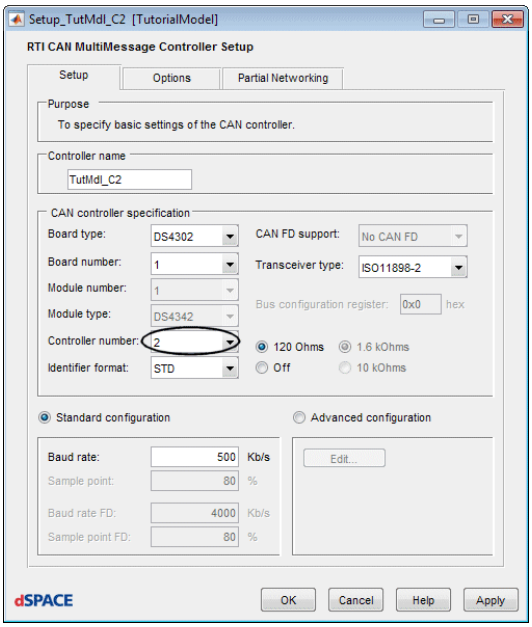
> **Note**
>
> Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

8 In your model, name the new RTICANMM ControllerSetup block **`Setup_TutMdl_C2`**.

9 Double-click the Setup_TutMdl_C2 block and rename the controller **`TutMdl_C2`**.

10 From the Board type drop-down list, select the board you are working with.

11 In the CAN controller specification frame, select Controller number **2**.

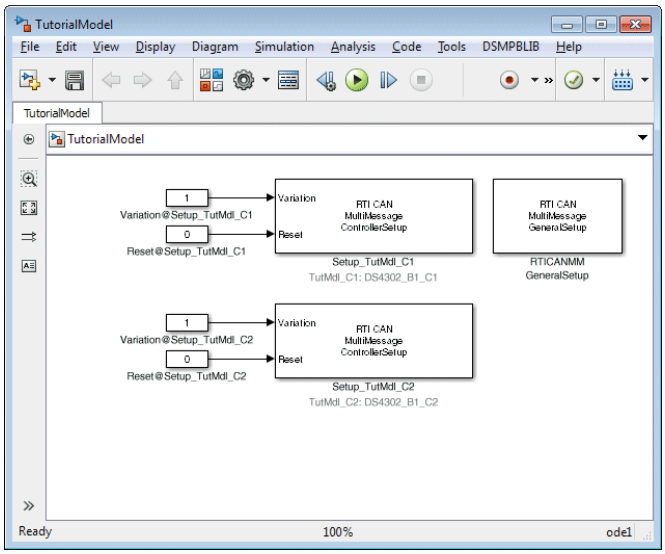


12 Click OK.

The Setup_TutMdl_C2 block dialog is closed and the Setup_TutMdl_C2 block is parameterized.

13 Click to save your model.

Result



You have created a new model and added one RTICANMM GeneralSetup block and two RTICANMM ControllerSetup blocks to it. In the RTICANMM GeneralSetup block, you defined general settings for your model. In the RTICANMM ControllerSetup blocks, you initialized two CAN controllers and

specified settings for them. S-functions were automatically created for both RTICANMM ControllerSetup blocks after you closed the block dialogs. Additionally, Simulink Constant blocks were connected to the blocks' inports. Your model displays the names of the RTICANMM ControllerSetup blocks and general information on the specified settings.

What's next

In the next step, you can add RTICANMM MainBlocks to your model, specify a database for them and configure CAN communication.

Related topics

Basics

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset..... 39
Overview of Lesson 1..... 40

Step 2: How to Add RTICANMM MainBlocks and Configure CAN Communication

Objective

The RTICANMM MainBlocks are the most important blocks for your daily work. You can configure the entire CAN communication of one CAN controller with one RTICANMM MainBlock.

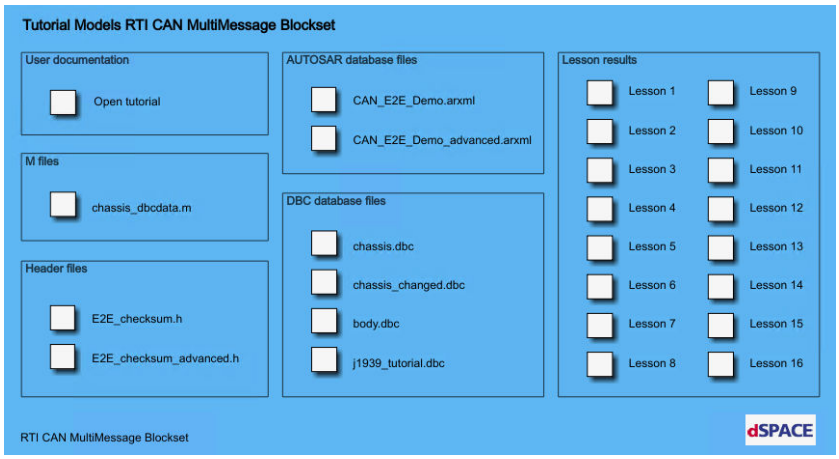
In this step you will add two RTICANMM MainBlocks to your model, one block for each CAN controller.

Part 1

To add an RTICANMM MainBlock to your model for the TutMdl_C1 CAN controller

1 In your MATLAB Command Window, type `rticanmm_tutorial`.

The rticanmm_tutorial_lib library opens.

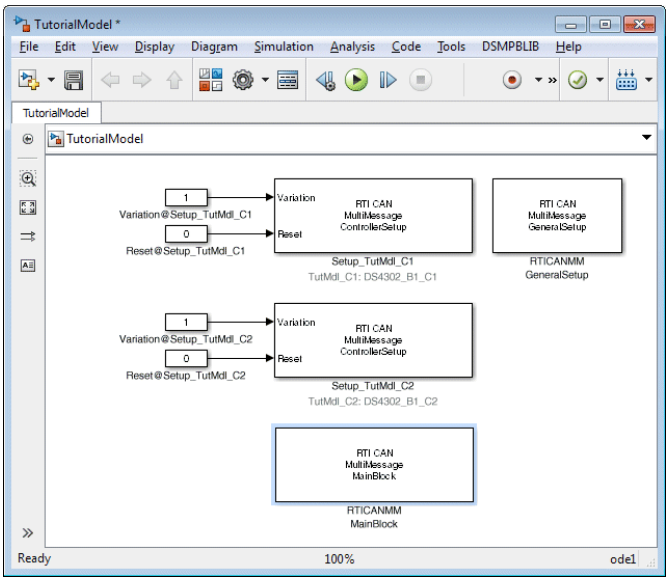


2 In the rticanmm_tutorial_lib library, double-click chassis.dbc.

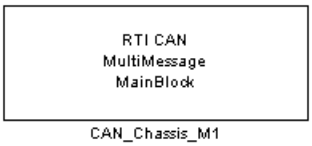
The chassis.dbc file is copied to your working folder. This file will be used as a database.

3 Close the rticanmm_tutorial_lib library.

4 From the rticanmmlib library, select the RTICANMM MainBlock and drag it to your model.



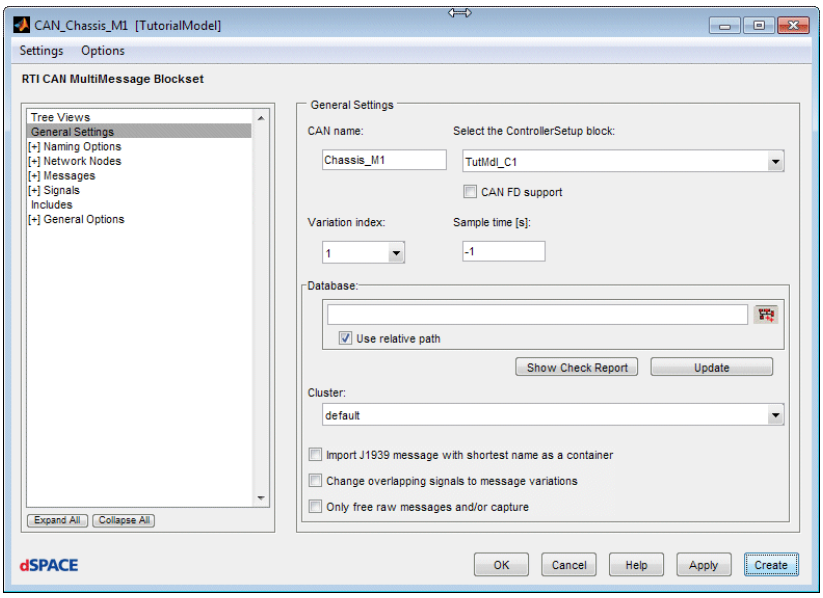
5 In your model, name the RTICANMM MainBlock **`CAN_Chassis_M1`**.



6 Double-click the CAN_Chassis_M1 MainBlock.

The CAN_Chassis_M1 [TutorialModel] dialog opens. The General Settings page is displayed.

7 In the CAN name edit field, type **`Chassis_M1`**.

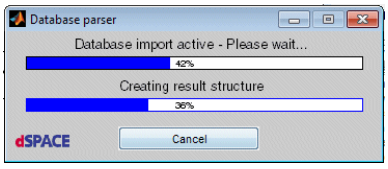


Tip

In the Select the ControllerSetup block drop-down list, the TutMdl_C1 controller is preselected. Since this MainBlock will configure the CAN communication for the TutMdl_C1 controller, you do not have to change the preselected controller.

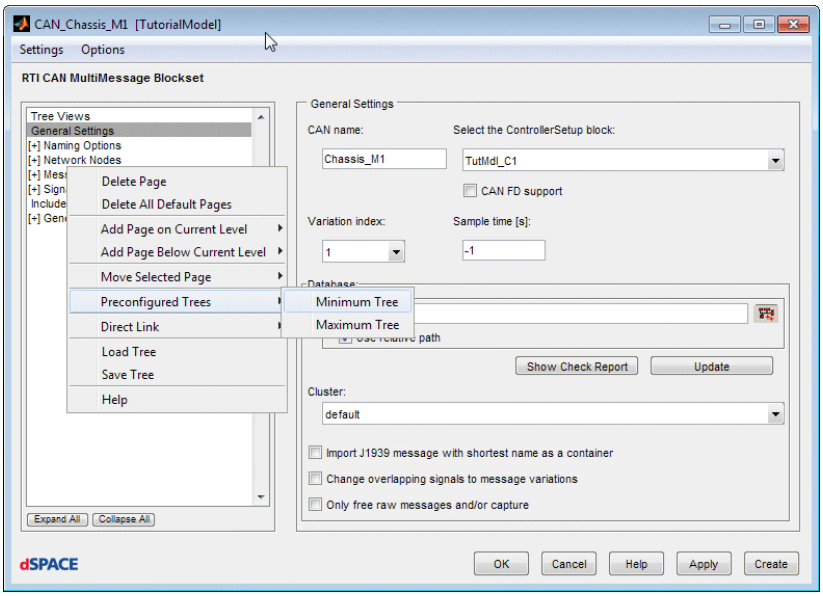
8 In the Database frame, click and select the chassis.dbc file from your RTICANMMTutorial folder.

The Database Parser dialog opens and parses the DBC file automatically.

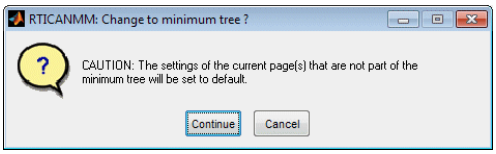


When the dialog closes, the chassis.dbc file has been specified as the database for the CAN_Chassis_M1 MainBlock.

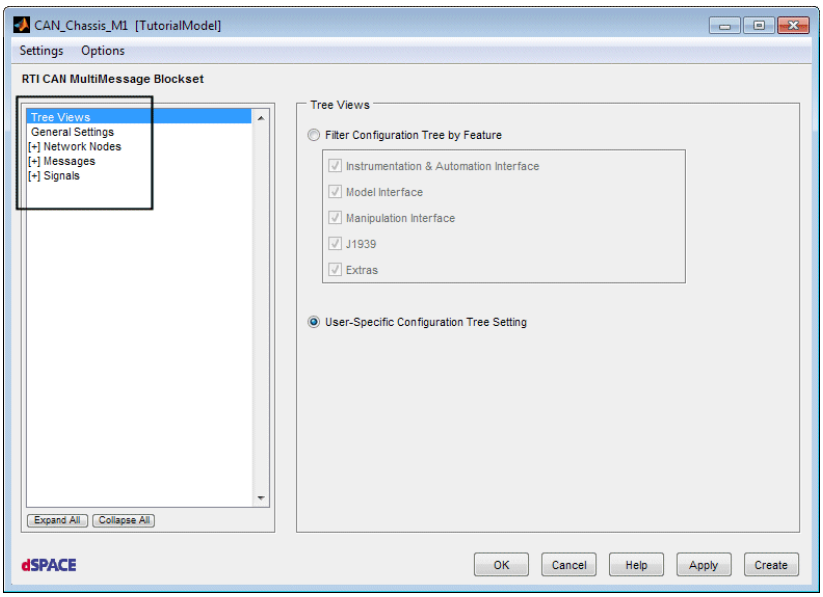
9 From the context menu of the dialog tree, select **Preconfigured Trees – Minimum Tree** to reduce the dialog tree to the minimum number of dialog pages.



A caution message similar to the following might be displayed, informing you that settings of pages that are not part of the minimum tree are set to their default values. Click **Continue**.

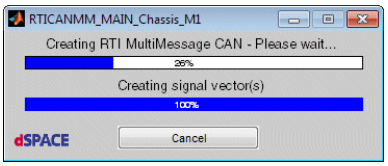


The **Tree Views** page opens, displaying the reduced dialog tree.



10 Click Create.

The RTICANMM_MAIN_Chassis_M1 dialog opens.



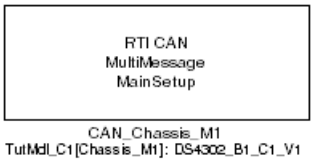
All the files needed for RTICANMM communication, such as an S-function, are created automatically.

Note

You always have to create all the RTICANMM MainBlocks of your model before you can build an application for it.

11 Click OK.

The CAN_Chassis_M1 [TutorialModel] dialog is closed. Your model displays the block's name, the specified CAN controller, the CAN bus name, and the board type, board number, and controller number.



Interim result

You added an RTICANMM MainBlock to your model and specified a database file for it. Since the TutMdl_C1 CAN controller is the default, this MainBlock configures the CAN communication on the TutMdl_C1 CAN controller. RTICANMM MainBlocks transmit and receive messages by default. Since you have not changed these default settings, the TutMdl_C1 CAN controller transmits and receives messages. You can now add the second RTICANMM MainBlock to your model. This block will configure the CAN communication on the TutMdl_C2 CAN controller, which will only receive messages. Continue with part 2.

Part 2

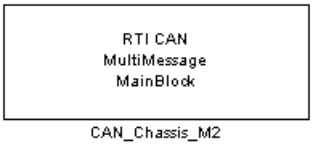
To add an RTICANMM MainBlock to your model for the TutMdl_C2 CAN controller

1 From the rticanmmlib library, add a second RTICANMM MainBlock to your model.

Note

Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

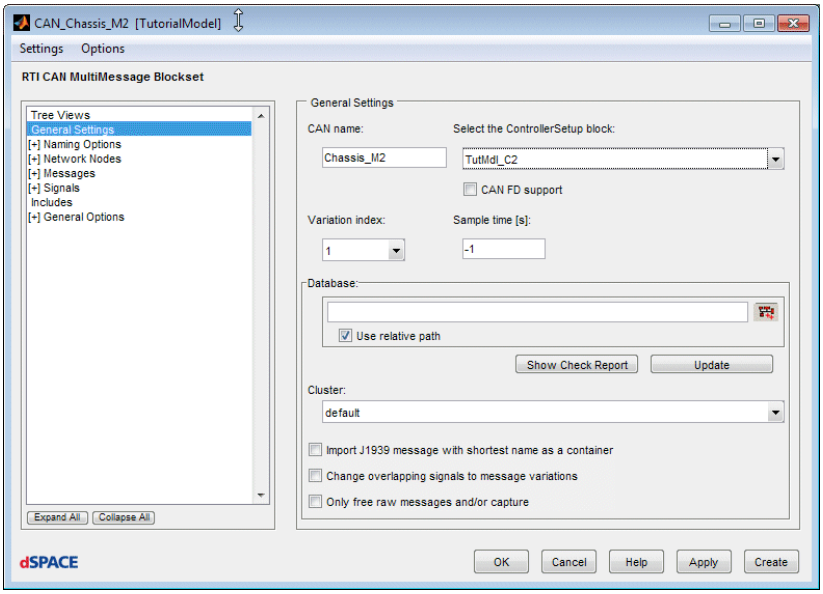
2 In your model, name the RTICANMM MainBlock **`CAN_Chassis_M2`**.



3 Open the CAN_Chassis_M2 MainBlock.

4 On the General Settings page, name the CAN bus **`Chassis_M2`**.

5 From the Select the ControllerSetup block drop-down list, select the **`TutMdl_C2`** controller.

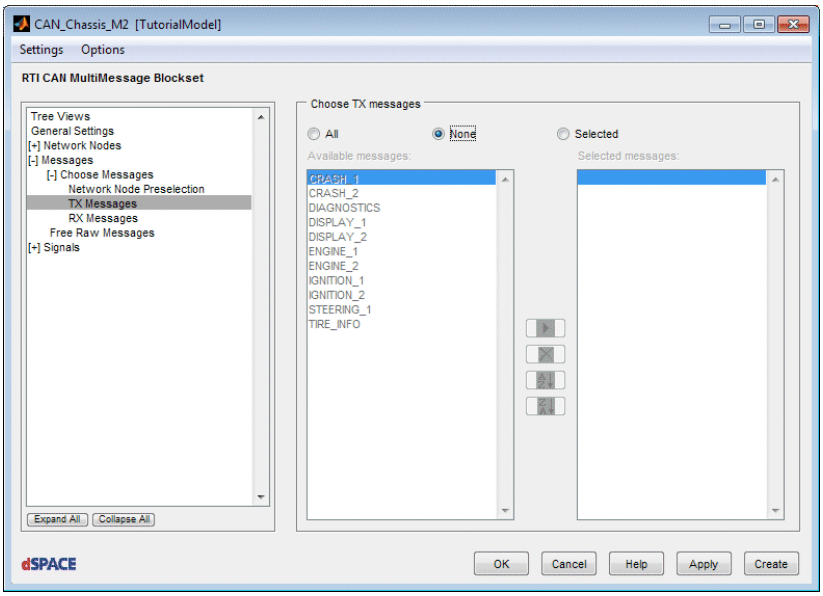


6 In the Database frame, specify the chassis.dbc file as a database for this MainBlock.

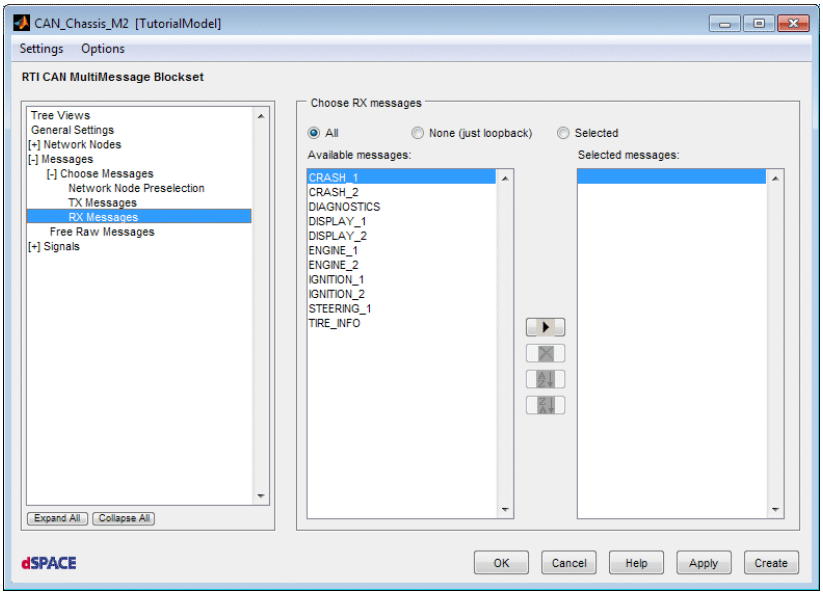
7 From the context menu of the dialog tree, select Preconfigured Trees – Minimum Tree to reduce the dialog tree to the minimum number of dialog pages.

8 In the dialog tree, select Messages – Choose Messages – TX Messages.

9 On the **TX Messages** page, select **None** to specify that no messages are to be transmitted by this CAN controller.



10 Select the **RX Messages** page.



On the RX Messages page, you can select the messages to be received by this RTICANMM MainBlock. All the messages specified in the chassis.dbc database file are received by default. You do not have to change the default settings for this MainBlock.

11 Click **Create**.

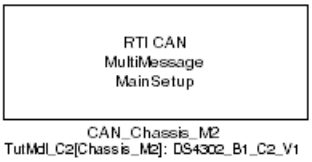
The **RTICANMM_MAIN_C2_Chassis_M2** dialog opens. All the files needed for your RTICANMM communication are created automatically, such as an S-function. The dialog closes automatically after all the necessary files are created.

Note

You always have to create all the RTICANMM MainBlocks of your model before you can build an application for it.

12 In the CAN_Chassis_M2 [TutorialModel] dialog, click OK.

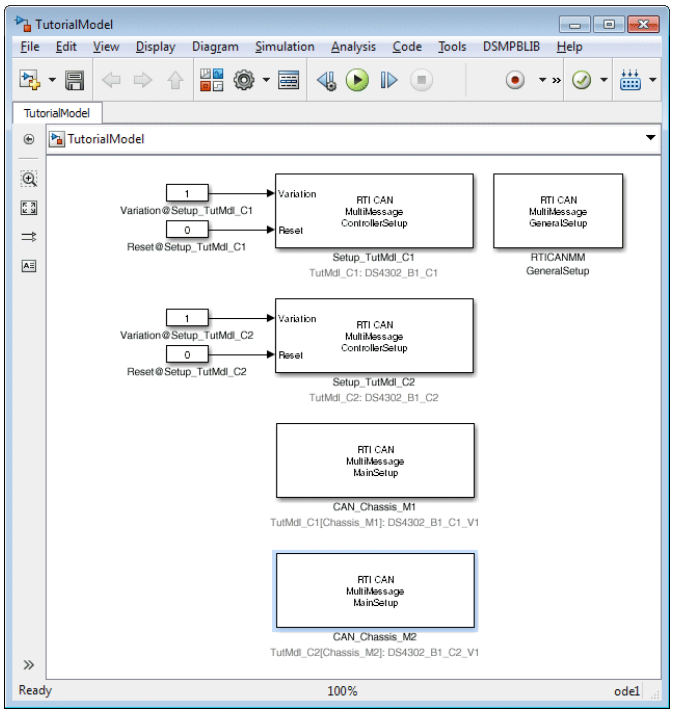
The dialog is closed. Your model displays the block's name, the specified CAN controller, the CAN bus name, and the board type, board number and controller number.



13 Save your model.

Result

You have added two RTICANMM MainBlocks to your model.



CAN_Chassis_M1 MainBlock The CAN_Chassis_M1 MainBlock specifies the CAN communication for the TutMdl_C1 CAN controller. The database for this block is the chassis.dbc file. By default, the block receives and transmits messages.

CAN_Chassis_M2 MainBlock The CAN_Chassis_M2 MainBlock specifies the CAN communication for the TutMdl_C2 CAN controller. The database for this block is the chassis.dbc file. Since you disabled the transmission of messages, this block only receives messages.

What's next

In the next step, you can build an application for your model and download it to your real-time hardware.

Related topics

Basics

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset..... 39
Overview of Lesson 1..... 40

Step 3: How to Build an Application and Download It to Your Real-Time Hardware

Objective

In the previous two steps, you built an S-function for your RTICANMM ControllerSetup and RTICANMM MainBlocks.

In this step, you will build an application for your model and download the application to your real-time hardware. In ControlDesk, you will check the application running on your real-time hardware.

Note

Your real-time hardware must be connected to your host PC and registered. For detailed information on registering, refer to How to Register a Platform (ControlDesk Platform Management).

Method

To build an application for your model and download it to your hardware

1 On the C Code ribbon, select Build – Build. If the C Code ribbon is hidden, go to the APPS ribbon and select Simulink Coder to display it.

Tip

You can also start the build process by pressing `Ctrl` + `B`.

The build process starts. Take a look at your MATLAB Command Window to observe the build process. MATLAB builds an application and downloads it directly to your real-time hardware. The download process can take some seconds. When the build and download process is finished, `*** Finished RTI build procedure for model TutorialModel` is displayed in the MATLAB Command Window.

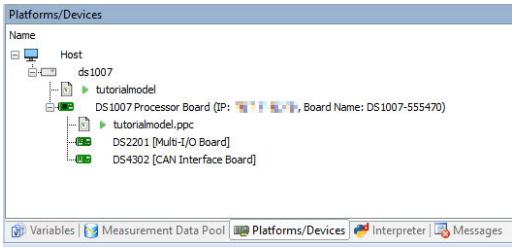
Tip

If MATLAB does not download the application directly to the real-time hardware, downloading is disabled. To enable it, do the following:

1. Open the Configuration Parameters dialog via the Simulation ribbon. On the ribbon, click Prepare – Model settings.
2. In the Configuration Parameters dialog, select Code Generation – RTI load options.
3. Select Load application after build.

2 Start ControlDesk.

3 In the Platforms/Devices controlbar, select your real-time hardware.

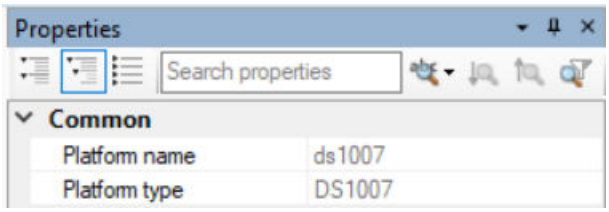


Tip

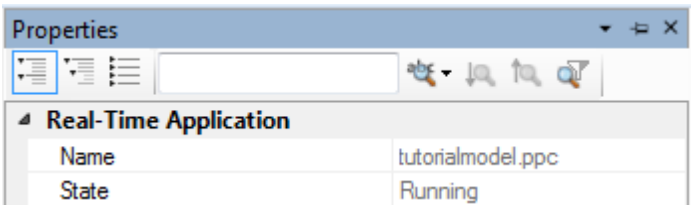
If the Platforms/Devices controlbar is hidden, go to the View ribbon and select Controlbar – Switch Controlbars – Platforms/Devices to display it.

4 In the Properties controlbar, check the following platform properties:

- Platform name
- Platform type



- Real-Time Application – Name
- Real-Time Application – State



Tip

- If the Properties controlbar is hidden, go to the View ribbon and select Controlbar – Switch Controlbars – Properties to display it.
- The location of the individual properties in the Properties controlbar depends on the hardware you are working with.

Result

You have started the build process for your model. MATLAB has built an application for it and downloaded the application to your real-time hardware. In ControlDesk, you have checked the platform properties and the loaded application.

What's next

With this step, you finished building your first model using RTI CAN MultiMessage blocks. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

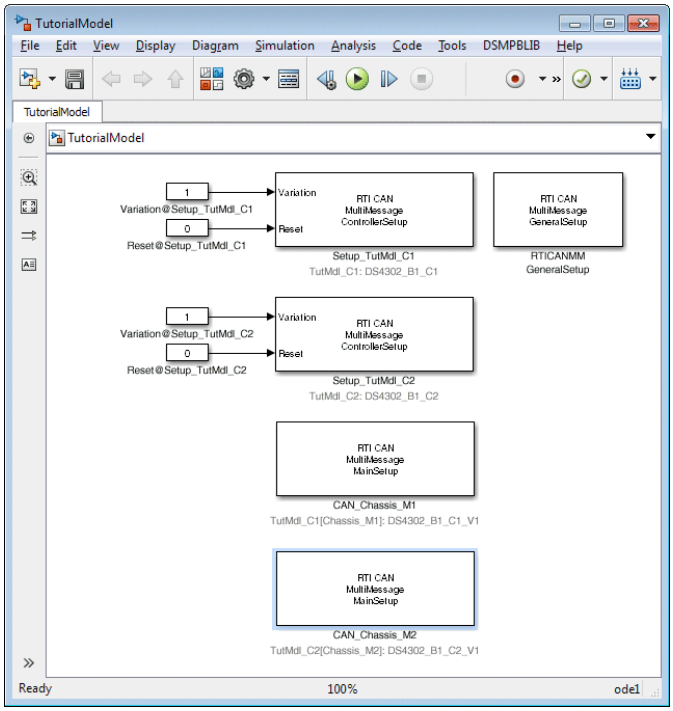
Basics

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset..... 39
Overview of Lesson 1..... 40

Result of Lesson 1

Result

In this lesson, you built your first model using RTICANMM blocks.



You learned the basic steps and principles of working with the RTI CAN MultiMessage Blockset:

- You should save your model before you add RTICANMM blocks to it.
- The first RTICANMM block you add to your model must be an RTICANMM GeneralSetup block.
- There must be one and only one RTICANMM GeneralSetup block in your model.
- Since you saved your model, you do not have to open the RTICANMM GeneralSetup block. General settings for your model are specified automatically. You only have to open this block if you want to change the default settings. You can, for example, specify a different folder to store the files created by the RTICANMM blocks.
- For each CAN controller you want to use with the RTICANMM blocks, you have to add one RTICANMM ControllerSetup block to your model and specify the CAN controller settings in the block.
- Since the name of the CAN controller must be unique, you should always rename it.
- You cannot copy RTICANMM blocks.
- The most important RTICANMM block for your daily work is the RTICANMM MainBlock. You can specify the entire CAN communication of one CAN controller with one RTICANMM MainBlock.

- You have to let S-functions be created for the RTICANMM ControllerSetup and RTICANMM MainBlocks of your model, before you can build an application for it.
- Each time you build an application for your model, the application is automatically downloaded to your real-time hardware, and you can control the application from within ControlDesk.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson1_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- General Information on the RTI CAN MultiMessage Blockset (RTI CAN MultiMessage Blockset Reference 🕮)
- CAN Setup Blocks (RTI CAN MultiMessage Blockset Reference 🕮)
- General Information on RTICANMM MainBlock (RTI CAN MultiMessage Blockset Reference 🕮)
- Commands, Menus and Dialogs of RTICANMM MainBlock (RTI CAN MultiMessage Blockset Reference 🕮)
- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to learn the basic principles of handling RTICANMM applications in ControlDesk.

Related topics

Basics

Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset..... 39
Overview of Lesson 1..... 40

Lesson 2: Displaying and Running the Simulation in ControlDesk

Where to go from here

Information in this section

Overview of Lesson 2 61
In this lesson, you will learn the basic principles of handling RTICANMM applications in ControlDesk.

Step 1: How to Create CAN Layouts in ControlDesk 63
In this step, you will open an RX and TX layout for one of the messages. The layouts display settings specified in the database file and/or in the RTICANMM MainBlock. Via TX layouts, you can manipulate message settings for which manipulation via ControlDesk is enabled.

Step 2: How to Trigger Message Transmission With CAN Layouts 68
In this step, you will learn how to trigger message transmission and manipulate message- and signal-specific settings via the TX layout. You will observe the effects of the manipulation in the RX layout.

Step 3: How to Monitor CAN Messages 72
In this step, you will learn how to monitor the messages that are received by a CAN controller.

Result of Lesson 2 79
In this lesson, you learned the basic steps and principles of displaying and manipulating CAN communication via ControlDesk.

Overview of Lesson 2

Introduction

When you work with the RTI CAN MultiMessage Blockset, you can use ControlDesk to trigger message transmission during run time and to monitor the CAN communication.

CAN bus configuration referenced by the SDF file When you build the real-time application for a model containing blocks of the RTI CAN MultiMessage Blockset, an SDF file and CAN bus configuration files (referenced by that SDF file) are generated. The CAN bus configuration contains information on all the controllers, databases, and messages that belong to the application. Each time you rebuild the application for your model, a new SDF file is generated.

When you load the SDF file to an experiment in ControlDesk, the CAN bus configuration referenced by that SDF file is displayed in the **Bus Navigator** tree.

From the CAN bus configuration displayed in the **Bus Navigator**, you can generate RX and TX message layouts. A message layout is a layout containing a Bus Instrument that lets you either observe (Bus Instrument - RX type) or manipulate (Bus Instrument - TX Type) CAN communication. All the message layouts are created automatically, and you do not have to preconfigure them manually. The layouts are tailor-made, i.e., they contain only the information for the selected message. RX and TX layouts differ from each other.

- RX layouts are view-only and display the settings of the selected message that was received by a CAN controller.
- TX layouts let you manipulate global, message-specific and/or signal-specific settings for the selected message during run time. You can specify settings to be manipulated via TX layouts on several pages of the RTICANMM MainBlock.

CAN bus monitoring ControlDesk's CAN bus monitoring feature allows you to observe all the messages that are received by a CAN controller.

What you will learn

In this lesson you will learn how to control and manipulate your CAN communication via ControlDesk.

- You will create layouts for TX and RX messages.
- You will be introduced to the basic elements of the message layouts.
- You will start the simulation and manipulate message parameters.
- You will observe the messages received by a CAN controller.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 1.

Prepared demo

If you did not create your own model in lesson 1, you can use the prepared demo **Lesson1_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder, and the **TutorialModel** model must be open.
- The application for the model must be built and downloaded to your real-time hardware. For details, refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39.
- ControlDesk must be running.

Steps

This lesson contains the following steps:

- Step 1: How to Create CAN Layouts in ControlDesk on page 63
- Step 2: How to Trigger Message Transmission With CAN Layouts on page 68
- Step 3: How to Monitor CAN Messages on page 72

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 2 on page 79.

Duration

Working through this lesson will take you about 50 minutes.

What's next

Now you can start to create your first layouts. Refer to Step 1: How to Create CAN Layouts in ControlDesk on page 63.

Related topics

Basics

Lesson 2: Displaying and Running the Simulation in ControlDesk.....61

Step 1: How to Create CAN Layouts in ControlDesk

Objective

To manipulate CAN communication via ControlDesk, you can open RX and/or TX layouts for messages. The layouts display settings specified in the database file and/or in the RTICANMM MainBlock. Via TX layouts, you can manipulate message settings for which manipulation via ControlDesk is enabled.

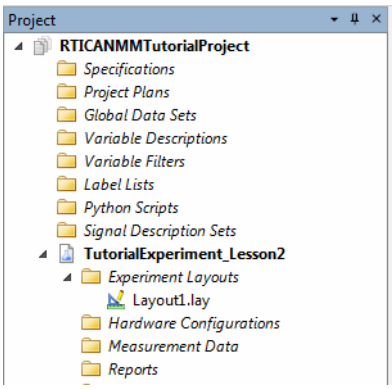
In this step, you will open an RX and TX layout for one of the messages.

Method

To create CAN layouts in ControlDesk

1. Create a project named `RTICANMMTutorialProject` and an experiment named `TutorialExperiment_Lesson2` in ControlDesk. For instructions, refer to method 1 in How to Create an Experiment in ControlDesk on page 483.

The illustration below displays the Project controlbar after creating `RTICANMMTutorialProject` and `TutorialExperiment_Lesson2`.



2 From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.

Tip

If Add to active Experiment is not available, the platform is already added to the experiment and displayed in the Hardware Configurations folder.

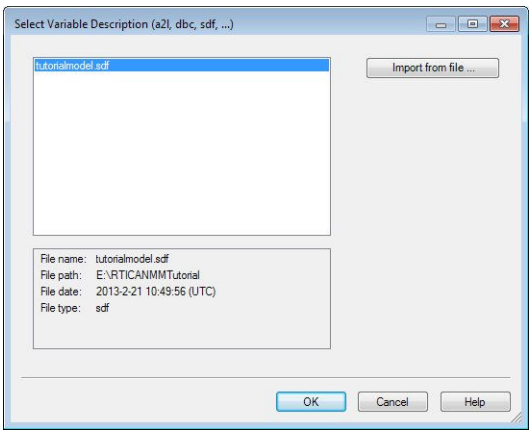
3 Click OK to confirm the default platform name.

The registered platform is added to the experiment.

4 From the platform's context menu in the Project controlbar, select Add Variable Description.

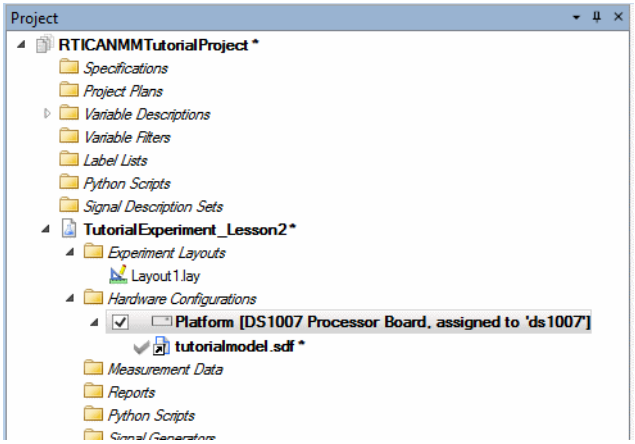
ControlDesk opens the Select Variable Description dialog.

5 In the Select Variable Description dialog, click Import from file and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

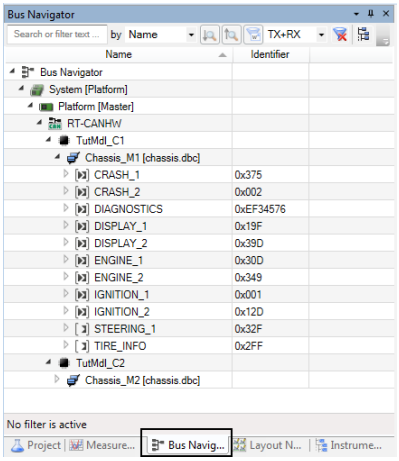


6 In the Select Variable Description dialog, click OK.

ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

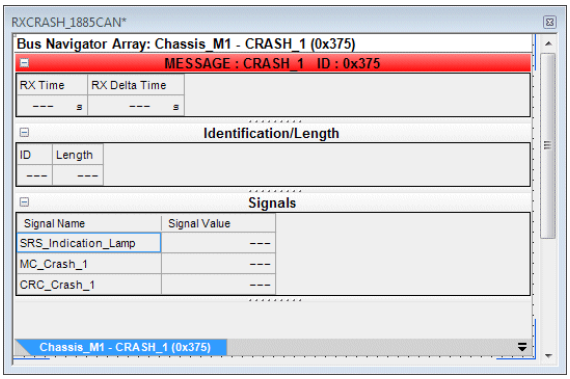


7 Select the Bus Navigator controlbar.



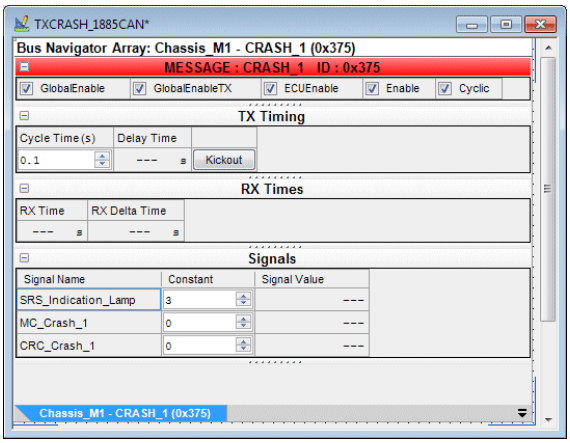
8 In the Bus Navigator tree, right-click the CRASH_1 node below the Chassis_M1 [chassis.dbc] node and select Generate RX Layout.

The RX layout for the CRASH_1 message opens.



9 In the Bus Navigator tree, right-click CRASH_1 and select Generate TX Layout.

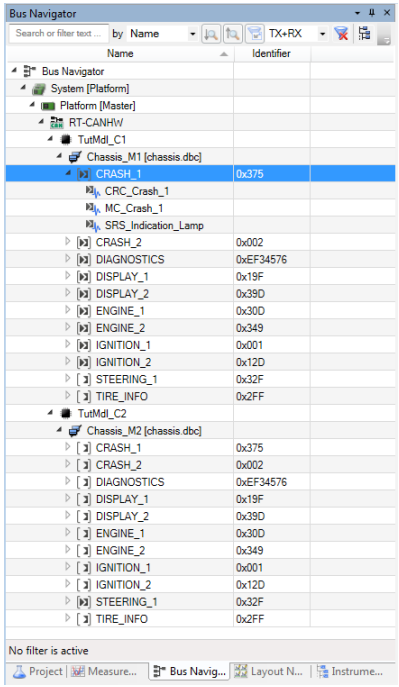
The TX layout for the CRASH_1 message opens.



Result

You created a project and an experiment in ControlDesk. You selected the real-time hardware (platform) and specified the application's SDF file for that experiment. Finally you generated a TX and an RX message layout.

Bus Navigator tree



The Bus Navigator tree contains the following nodes:

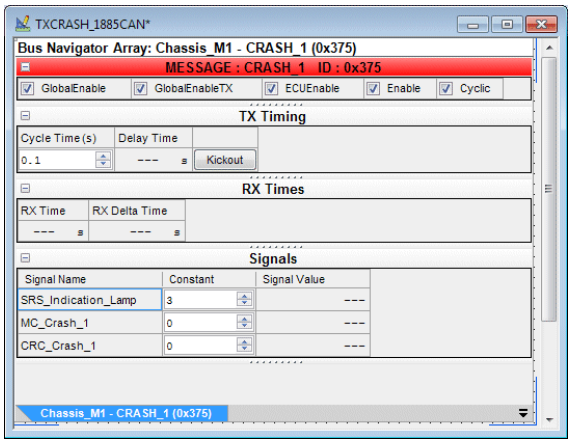
- **Bus Navigator**: This is the root node of a bus configuration in the Bus Navigator tree and displays the system(s) used.
- **System [Platform]**: This node contains the processor node(s) to which a bus configuration is loaded.

- **Platform [Master]**: This node displays the platform you are working with. It displays the bus configuration that is loaded to the platform. It contains the bus system node(s).
- **RT-CAN HW**: This node contains all the CAN controllers configured using RTICANMM blocks.
- **TutMdl_C1**, **TutMdl_C2**: These nodes display the controllers configured with the RTICANMM ControllerSetup blocks.
- **Chassis_M1 [chassis.dbc]**, **Chassis_M2 [chassis.dbc]**: These nodes display the CAN bus names and database files specified in the corresponding RTICANMM MainBlocks and contain the message configurations.
- **CRASH_1**, **CRASH_2**, ...: These and all the other nodes of this level display the messages specified in the database file. The message IDs are displayed in brackets. The icons indicate whether the messages are RX, TX, or RX and TX messages:

Icon	Message Type
[▶]	TX
[ ]	RX
[▶]	TX/RX

- **CRC_Crash_1**, **MC_Crash_1**, ...: These and all the other nodes of this level display the signals of the messages.

RX and TX message layouts



The message layouts display all the message-specific information for CRASH_1 specified in the DBC file and/or in the CAN_Chassis_M1 MainBlock. Via the TX layout, you can manipulate message settings for which manipulation via ControlDesk is enabled:

- You can enable or disable global message transmission via **GlobalEnable** and/or **GlobalEnableTX**. Disabling global message transmission disables not only the CRASH_1 message but the entire CAN communication (GlobalEnable) or all the TX messages (GlobalEnableTX) defined for this CAN controller.

- You can enable or disable the transmitting ECU of this message via **ECUEnable**.

> **Tip**
>
> You will learn more on enabling and disabling ECUs in Lesson 6: Performing a CAN Restbus Simulation.

- You can enable or disable the transmission of this message via **Enable**.
- You can transmit this message via **Cyclic** and/or **Kickout**.
- You can manipulate the message's **Cycle Time** but not its **Delay Time**.
- You can manipulate the message's signal values.

Since you opened both layouts for the first time and did not start online calibration yet, the headers of both layouts are red.

What's next

In the next step, you can start online calibration and manipulate message- and signal-specific settings via the TX layout. You can observe the effects of the manipulation in the RX layout.

Related topics

Basics

Lesson 2: Displaying and Running the Simulation in ControlDesk.....61
Overview of Lesson 2.....61

Step 2: How to Trigger Message Transmission With CAN Layouts

Objective

Using the TX layout of a message, you can trigger message transmission. You can, for example, transmit a message cyclically or once immediately via kickout. To trigger message transmission, you have to start online calibration.

In this step, you will trigger the transmission of the CRASH_1 message via its TX layout and observe the effects on the TX layouts and RX layouts.

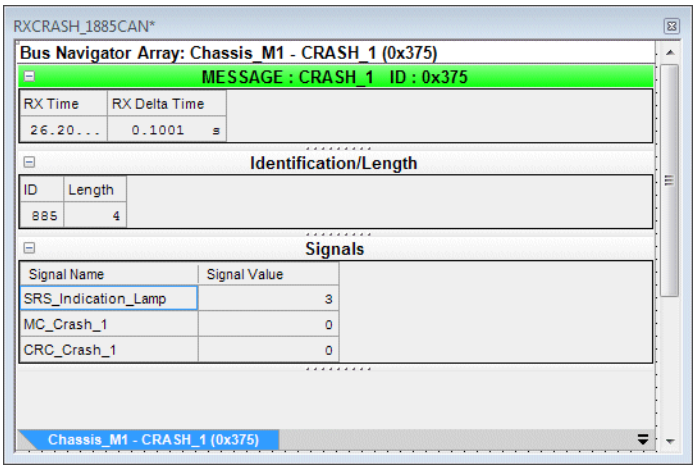
Method

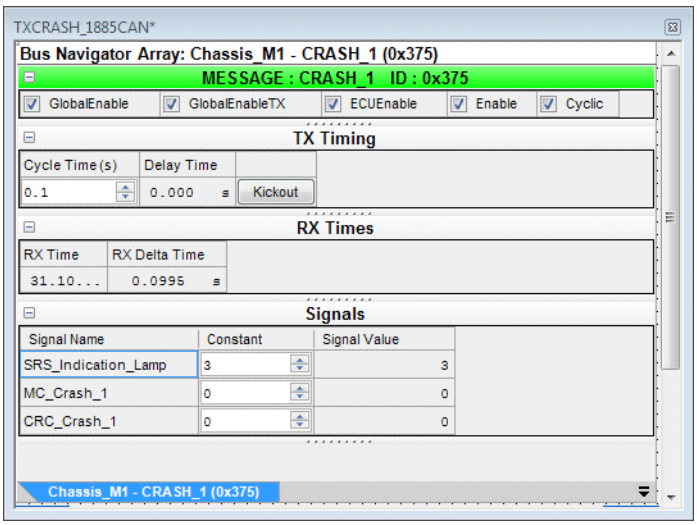
To trigger message transmission during run time

1 On ControlDesk's **View** ribbon, click **Window – Tile Horizontally** to arrange the TX and RX layouts, so that you can see both layouts at the same time.

> **Tip**
>
> You can reset the display by maximizing one of the layouts.

2 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.





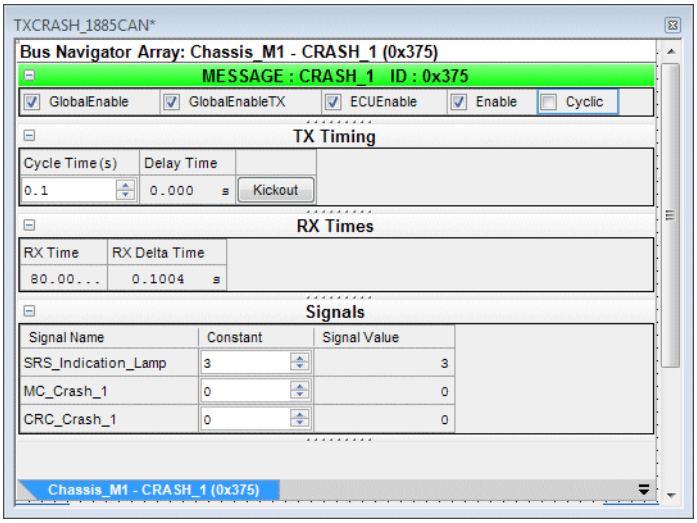
- The header colors of both layouts change from red to green. A green header indicates that CRASH_1 *can* be transmitted/received. However, green headers do not indicate whether or not the message is actually transmitted/received. The transmission/reception of the message by the CAN controller is indicated by a running RX time.
- The **RX Time** is running. This indicates that CRASH_1 is being transmitted and received by the CAN controller. Since an application starts running immediately when you download it to your real-time hardware, the RX time does not start from 0 when you start online calibration.

Note

If the RX time is not running, make sure that CAN channels 1 and 2 of your real-time hardware are physically connected.

- The **RX Delta Time** value changes from **0** to **0.100**. This is the time between the reception of two CRASH_1 messages.

3 In the TX layout, clear the **Cyclic** checkbox.

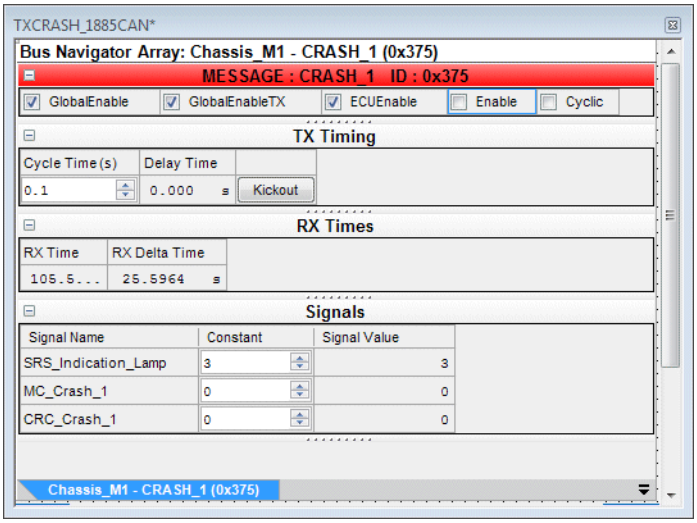


The cyclic transmission of the CRASH_1 message is disabled. The **RX Time** stops. The header remains green because transmission via kickout is still possible.

4 Click **Kickout**.

The message is transmitted once via kickout. The values of the **RX Time** and **RX Delta Time** change.

5 Clear the **Enable** checkbox.



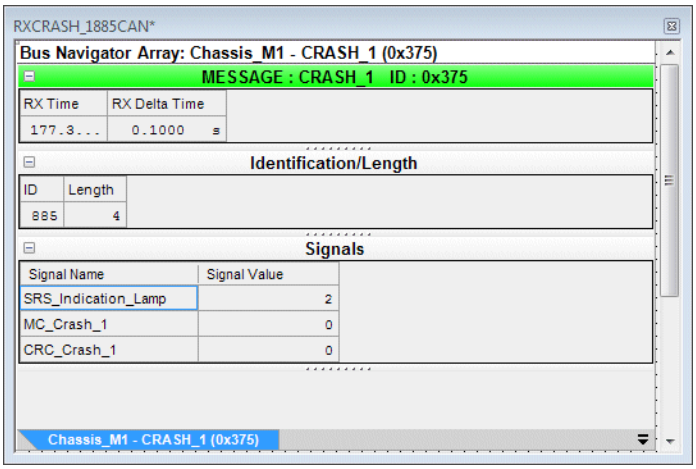
The header color of the TX layout changes from green to red. The message can no longer be transmitted, whether via cyclic transmission or via kickout.

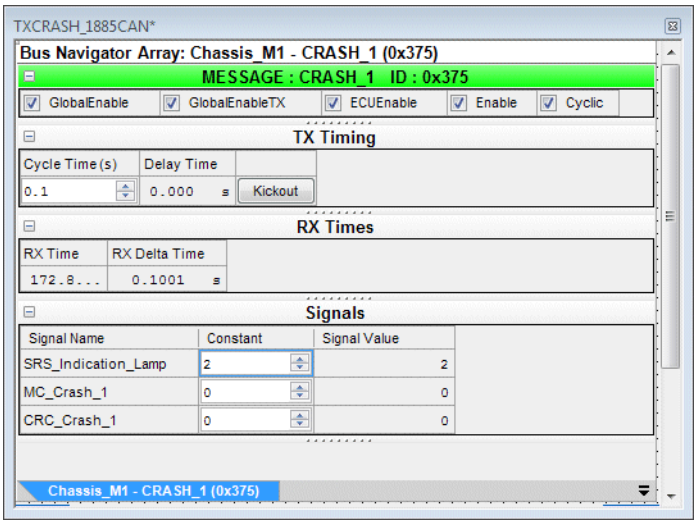
6 Select **Enable** and **Cyclic**.

The message is transmitted via cyclic transmission. The TX header color changes from red to green and the **RX Time** is running.

7 Change the **Constant** value of the **SRS_Indication_Lamp** to **2**.

The **SRS_Indication_Lamp** signal is transmitted with a value of 2. In both layouts, the **Signal Value** is changed to 2.





Tip

You manipulated CRASH_1 for the TutMdl_C1 CAN controller. Since you observed the effects of the manipulation on the same CAN controller, you observed loopback behavior in CRASH_1: The message is transmitted and received by the TutMdl_C1 CAN controller, the displayed signal values are referenced to the same variable in the model's TRC file.

8 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

9 On the **File** ribbon, click **Save Project** to save the project and experiment.

When you save a project, all the generated TX and RX layouts that are currently open in an experiment are also saved as part of the project and experiment.

Tip

You can also save the project and experiment by clicking the icon in the Quick Access Toolbar.

Result

You have manipulated the transmission settings for the CRASH_1 message:

- You have enabled and disabled message transmission.
- You have transmitted the message cyclically and via kickout.
- You have changed the value of the SRS_Indication_Lamp signal.

You observed the effects of your manipulations on the TX and RX layouts:

- A red header indicates that the message cannot be transmitted/received.
- A green header indicates that the message *can* be transmitted/received.
- A running RX time indicates that the message is being transmitted/received. The RX delta time indicates the time between two message receptions.
- The signal value changes on both layouts if you specify another signal value.

What's next

In the next step, you can monitor all the messages that are received by a CAN controller.

Related topics

Basics

Lesson 2: Displaying and Running the Simulation in ControlDesk.....61
Overview of Lesson 2.....61

Step 3: How to Monitor CAN Messages

Objective

ControlDesk lets you monitor all the messages that are received by a CAN controller. You can open a monitoring list for each CAN controller of the application. The monitoring list displays information on all the messages and their signals received by a CAN controller.

Tip

Every message transmitted by a CAN controller is received as a loopback message by the same CAN controller. These loopback messages are displayed in the monitoring list.

In this step, you will monitor messages for the TutMdl_C1 CAN controller.

For a better understanding of CAN bus monitoring, you will first modify the CAN communication of the TutMdl_C2 CAN controller using the CAN_Chassis_M1 MainBlock.

Part 1

To modify the CAN communication of the TutMdl_C2 CAN controller

1 In your model, open the CAN_Chassis_M1 MainBlock.

Tip

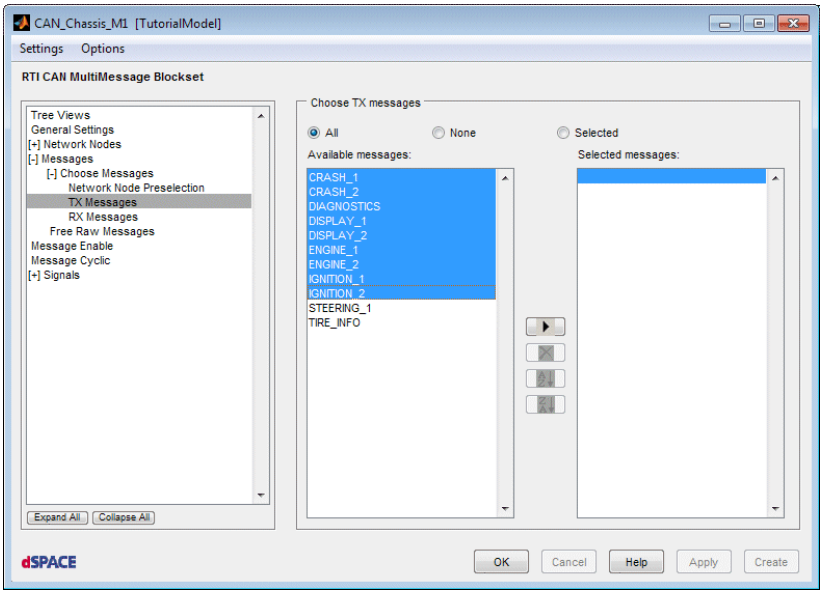
Since you already specified all the general settings for the CAN_Chassis_M1 MainBlock, the Tree Views page is now displayed by default.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

3 In the Available messages list, select all the messages except for STEERING_1 and TIRE_INFO.

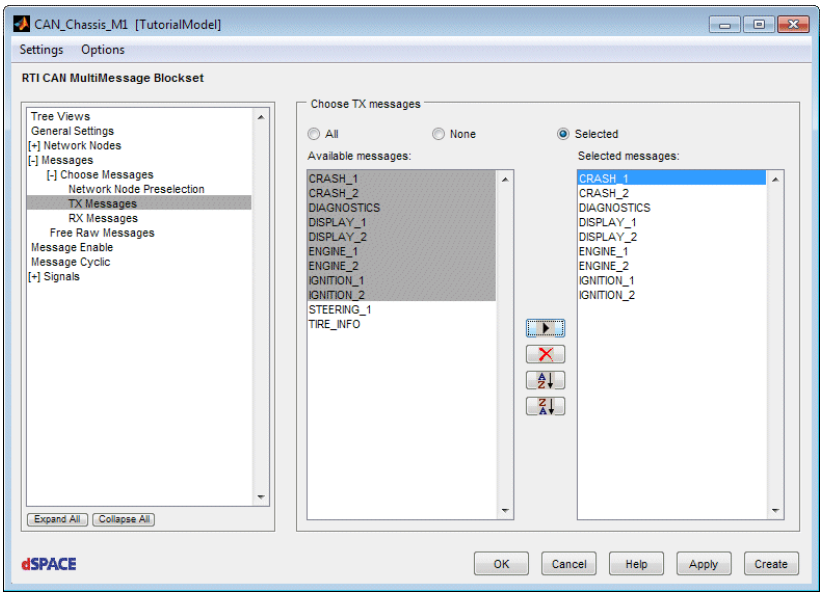
Tip

You can use `Shift` and `Ctrl` for multiselection.



4 Click .

All the selected messages are added to the **Selected messages** list.



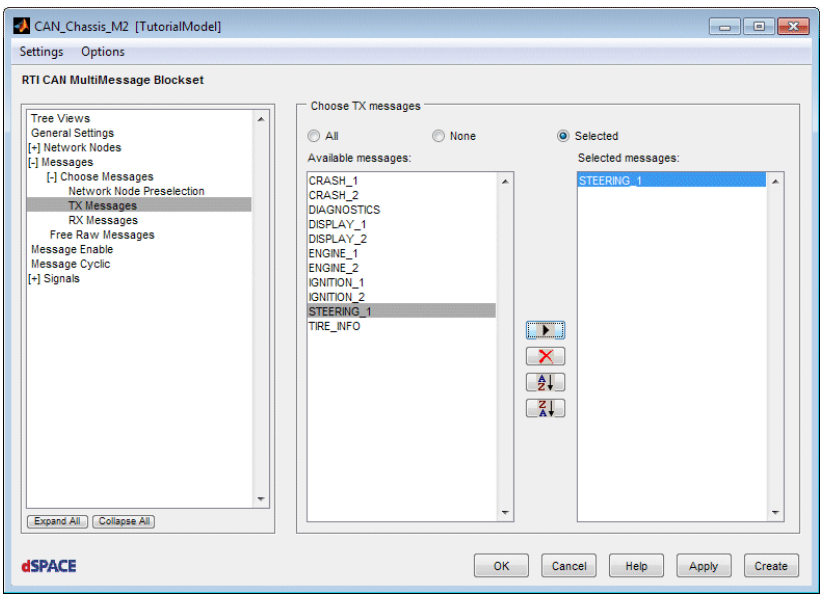
5 Click **Create** to let the necessary files be created automatically and to apply the specified settings to the model.

6 Click **OK** to close the **CAN_Chassis_M1 [TutorialModel]** dialog.

7 Open the **CAN_Chassis_M2** MainBlock.

8 On the **TX Messages** page, select the **Selected** radio button.

9 In the **Available messages** list, select **STEERING_1** and click .



10 Click **Create** and close the dialog.

11 Build an application for your model.

Interim result

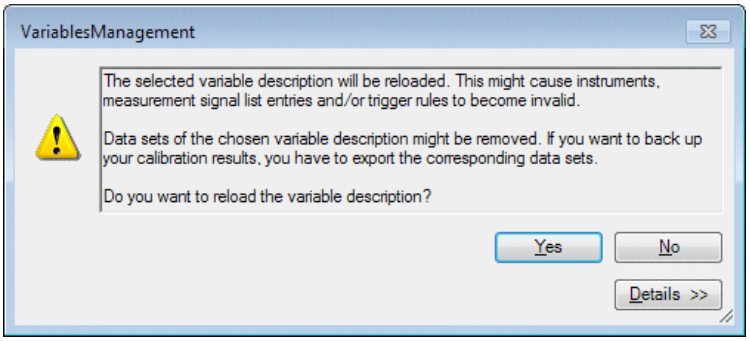
You modified your CAN communication: You disabled the message transmission of TIRE_INFO. STEERING_1 is only transmitted by the CAN_Chassis_M2 MainBlock. You can now monitor messages in ControlDesk. Continue with part 2.

Part 2

To monitor messages received by the TutMdl_C1 CAN controller

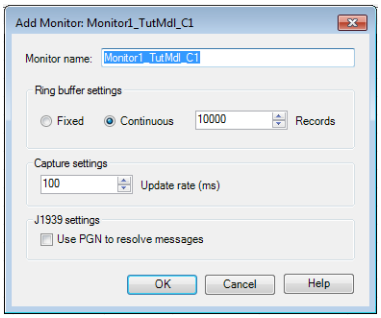
1 From the context menu of the `tutorialmodel.sdf` file in the active experiment, select Reload Variable Description.

The following warning is displayed, informing you that reloading a variable description might cause instruments and other items to become invalid. Click Yes.



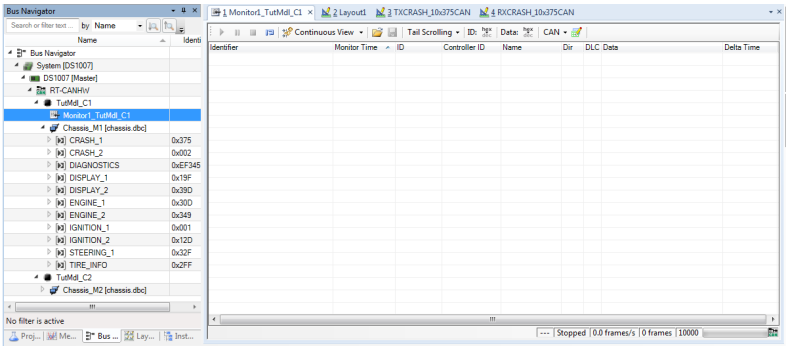
2 In the Bus Navigator tree, right-click the TutMdl_C1 node and select Add Unfiltered Monitor.

The Add Monitor: Monitor1_TutMdl_C1 dialog opens.



3 Click OK to close the dialog.

The Monitor1_TutMdl_C1 node is added to the Bus Navigator tree and the Monitor1_TutMdl_C1 monitoring list opens. The list does not display any messages because CAN bus monitoring was not started yet.

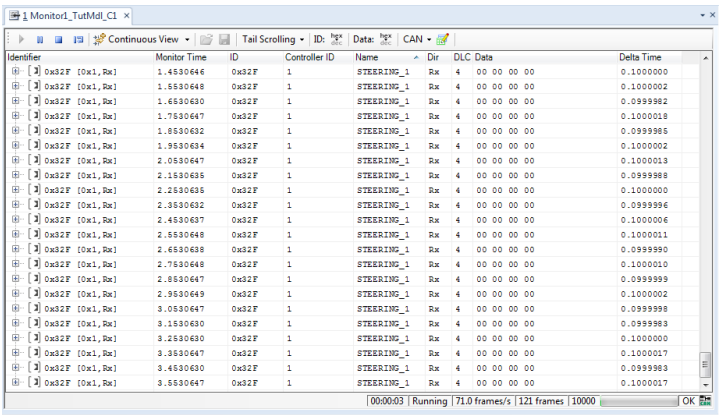


4 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.

5 In the Bus Navigator tree, right-click the **Monitor1_TutMdl_C1** node and select **Start**.

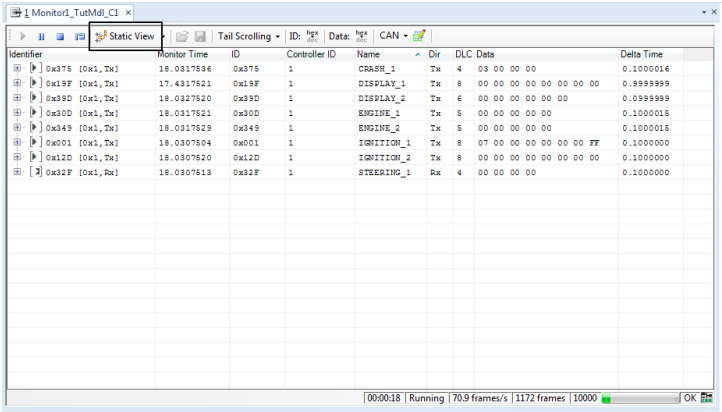
> **Tip**
>
> To start monitoring, you can also click ▶ in the monitoring list's toolbar.

CAN bus monitoring starts.



All the messages received by the TutMdl_C1 CAN controller are displayed. The changing **Delta Time** and **Monitor Time** values indicate that bus communication is active.

6 From the monitoring list's toolbar, select **Static View**.



In static view mode, the monitoring list contains one row for each message, and the row is updated with the latest received data.

7 Click the **Name** column header to sort the monitoring list alphabetically by message name.

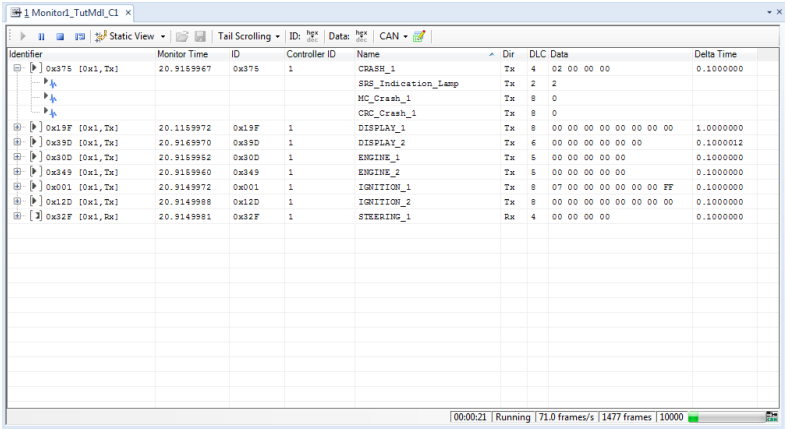
- *CRASH_1*: The **Dir** entry for CRASH_1 is **TX**. This indicates that CRASH_1 is transmitted by the TutMdl_C1 CAN controller and received as a loopback message.
- *STEERING_1*: The **Dir** entry for STEERING_1 is **RX**. This indicates that STEERING_1 is transmitted by another CAN controller and received by the TutMdl_C1 CAN controller.

> **Tip**
>
> Monitor the CAN communication for the TutMdl_C2 CAN controller. Since STEERING_1 is the only message transmitted by this CAN controller, it is the only message with a TX direction in the monitoring list.

- *TIRE_INFO*: TIRE_INFO is not displayed because it is transmitted neither by the TutMdl_C1 CAN controller nor by the TutMdl_C2 CAN controller.

8 Click + next to the identifier of the **CRASH_1** message.

The list expands, the signals and their data of the CRASH_1 message are displayed. The **Data** entry for the **SRS_Indication_Lamp** signal is 2, i.e. the value you entered in step 2 of this lesson.



9 Close the Monitor1_TutMdl_C1 monitoring list. ControlDesk asks you if you really want to delete Monitor1_TutMdl_C1. Click 'Yes'. Monitoring is stopped automatically. The Monitor1_TutMdl_C1 node is removed from the Bus Navigator tree.

10 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

11 Save the project.

Result

You modified your CAN communication: You disabled message transmission for the TIRE_INFO message and specified the STEERING_1 message to be only transmitted by the TutMdl_C2 CAN controller.

In ControlDesk, you reloaded the variable description, opened the monitoring list and started CAN bus monitoring. For a better overview, you sorted the monitoring list by message name. Since the TIRE_INFO message is not transmitted by any CAN controller, it is not displayed in the list. Since the STEERING_1 message is only received but not transmitted by the TutMdl_C1 CAN controller, its direction is RX. For the CRASH_1 message, you ascertained that the data value of the SRS_Indication_Lamp signal is the one you entered in step 2 of this lesson.

What's next

With this step, you finished displaying and running the simulation in ControlDesk. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 2: Displaying and Running the Simulation in ControlDesk.....61
Overview of Lesson 2.....61

Result of Lesson 2

Result

In this lesson, you created your first RX layout and TX layout and manipulated the message CRASH_1 during run time. Additionally, you monitored your CAN communication.

You learned the basic steps and principles of displaying and manipulating your CAN communication via ControlDesk:

CAN bus configuration, SDF file and Bus Navigator tree

- The CAN bus configuration contains information on all the controllers, databases, and messages that belong to a real-time application. When you build the application for a model containing blocks for handling a CAN bus communication, the CAN bus configuration is created together with an SDF file referencing that configuration.
- To display and manipulate CAN communication in ControlDesk, you have to create an experiment with a real-time platform that references the SDF file for the RTICANMM model.

  All the information on your CAN communication is available via the Bus Navigator tree displayed in the **Bus Navigator**. You can easily identify RX, TX, or TX/RX messages by their icons in the Bus Navigator tree.
- In the Bus Navigator tree, you have to right-click a message's node to generate an RX and/or TX layout. All the layouts are tailor-made and contain only the settings for the selected message. The settings specified in the model's DBC file are used for layout creation by default. You can change these settings via your RTICANMM model.
- The CAN bus configuration is automatically updated when you make any changes in your model and build a new application for it. To display the updated CAN bus configuration in ControlDesk, the variable description has to be reloaded.

RX and TX layouts

- The RX layout is view-only, the TX layout lets you manipulate the message settings during run time. To manipulate message settings, you have to start online calibration.
- A green header in TX layouts indicates that a message *can* be transmitted. Green headers do not indicate that the message is already being transmitted. The transmission/reception of a message is indicated by a running RX time in TX and RX layouts.
- Message layouts can be recreated as required. When you save a project, all the layouts of an experiment that have been opened are automatically saved.

CAN bus monitoring

- You can monitor all the messages that are received by a CAN controller.
- Every message transmitted by a CAN controller is received as a loopback message by the same CAN controller. These loopback messages are displayed in the monitoring list, their direction is TX.
- In the Bus Navigator tree, you can open a monitoring list via context menu of a CAN controller node, for example.

- You have to start monitoring for messages to be displayed in the list.
- You can sort the monitoring list by its entries.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson2_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Introduction to the Bus Navigator (ControlDesk Bus Navigator 🕮)
- Basics on Working with the Bus Navigator (ControlDesk Bus Navigator 🕮)
- Working with Bus Instruments (ControlDesk Bus Navigator 🕮)
- Monitoring and Logging Bus Communication (ControlDesk Bus Navigator 🕮)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to learn the basic principles of triggering TX messages via your model and/or via ControlDesk.

Related topics

Basics

Lesson 2: Displaying and Running the Simulation in ControlDesk.....61
Overview of Lesson 2.....61

Lesson 3: Triggering Message Transmission via the Model and ControlDesk

Where to go from here

Information in this section

Overview of Lesson 3 81
In this lesson, you will learn the basic principles of triggering TX messages via model and/or via ControlDesk. You will learn how to manipulate message transmission.

Step 1: How to Enable or Disable Message Transmission 83
In this step, you will learn how to enable and disable message transmission by default.

Step 2: How to Specify Cyclic Message Transmission or Transmission via Kickout 89
In this step, you will learn how to specify cyclic message transmission or transmission via kickout.

Step 3: How to Check Message Transmission Settings in ControlDesk 94
In this step, you will check the effects of the message transmission settings you specified in the model on the layouts.

Result of Lesson 3 101
In this lesson, you learned the basic principles of triggering and manipulating message transmission during run time.

Overview of Lesson 3

Introduction

All the TX messages of a CAN communication are transmitted according to the settings specified in the database file and the default settings of the RTICANMM

MainBlocks. You can change these settings and specify new default settings. You can also modify the default settings during run time.

What you will learn

In this lesson you will learn how to manipulate message transmission.

- You will specify default settings for message transmissions.
- During run time, you will enable and disable transmission and transmit messages cyclically or via kickout.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 2.

Prepared demo

If you did not create your own model in lesson 2, you can use the prepared demo **Lesson2_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Enable or Disable Message Transmission on page 83
- Step 2: How to Specify Cyclic Message Transmission or Transmission via Kickout on page 89
- Step 3: How to Check Message Transmission Settings in ControlDesk on page 94

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 3 on page 101.

Duration

Working through this lesson will take you about 60 minutes.

What's next

Now you can start to enable and disable message transmission by default, see Step 1: How to Enable or Disable Message Transmission on page 83.

Related topics

Basics

Lesson 3: Triggering Message Transmission via the Model and ControlDesk..... 81

Step 1: How to Enable or Disable Message Transmission

Objective

You can enable or disable transmission for all the messages of your model by default. These default settings can be modified via your model and/or via ControlDesk during run time.

In this step, you will both enable and disable message transmission by default. Additionally, you will allow modification of these settings via your model and via ControlDesk during run time.

Part 1

To enable or disable message transmission via model and/or ControlDesk

1. In your Simulink model, open the CAN_Chassis_M1 MainBlock.

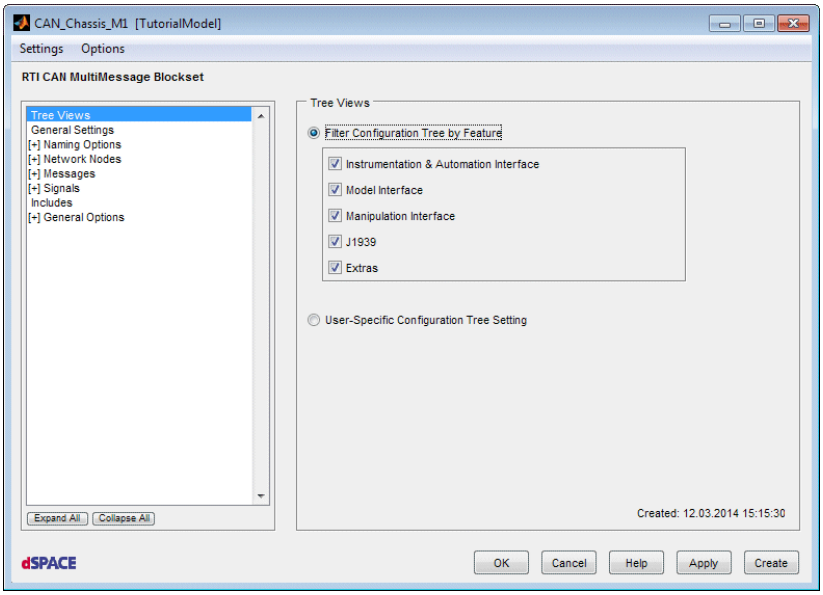
   The Tree Views page is displayed for you to display or hide dialog pages. You can use predefined filters or specify your own filters for this.

2. On the Tree Views page, select Filter Configuration Tree by Feature.

   The predefined filters are enabled and the dialog tree is extended according to the selected filters.

   Tip

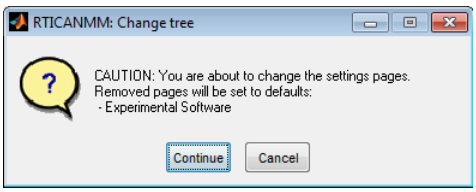
   Focus one of the predefined filters, for example, Model Interface, with your mouse. A tooltip lists all the pages that are enabled if you select the filter.



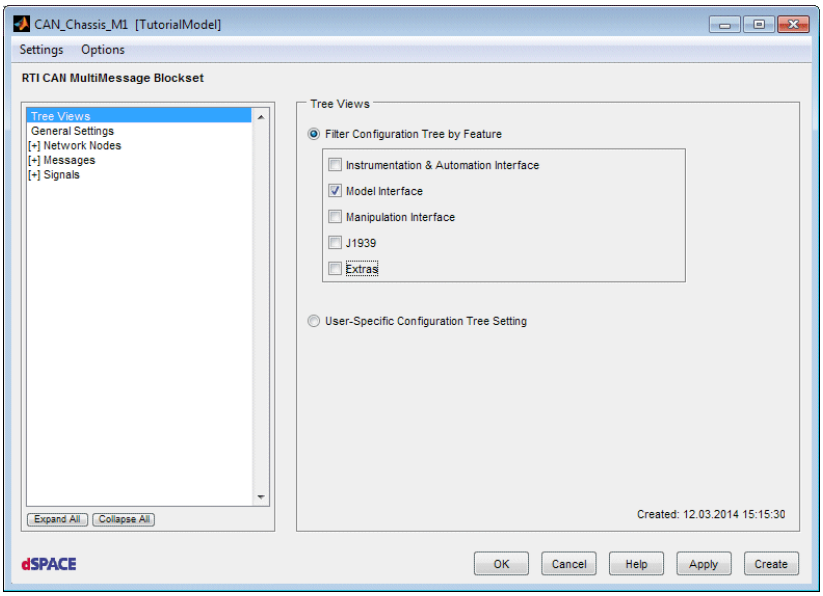
For a better overview, you can clear all the filters you do not need. This reduces the number of pages displayed in the dialog tree.

3 Clear all filters except for **Model Interface**.

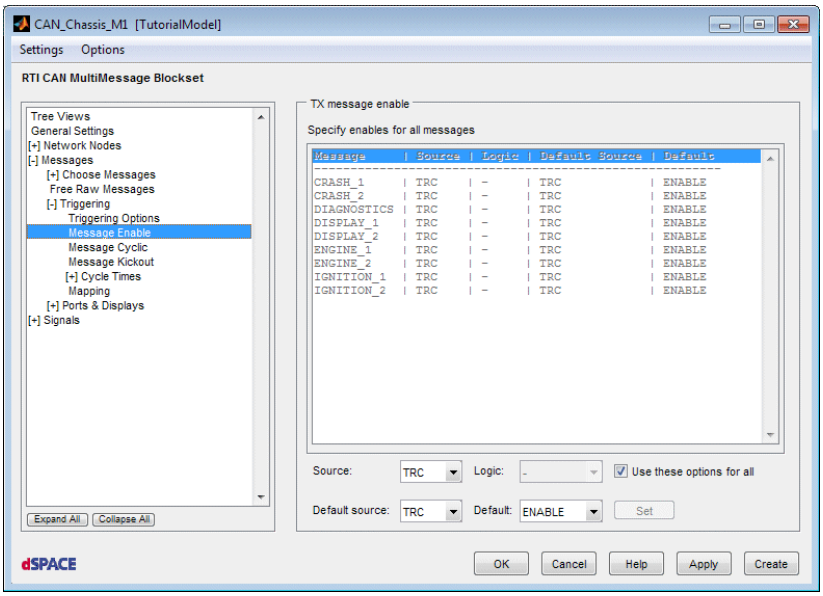
If a dialog similar to the following one is displayed, click **Continue**.



The dialog tree is reduced to the pages available for the Model Interface filter.



4 In the dialog tree, select Messages – Triggering – Message Enable.

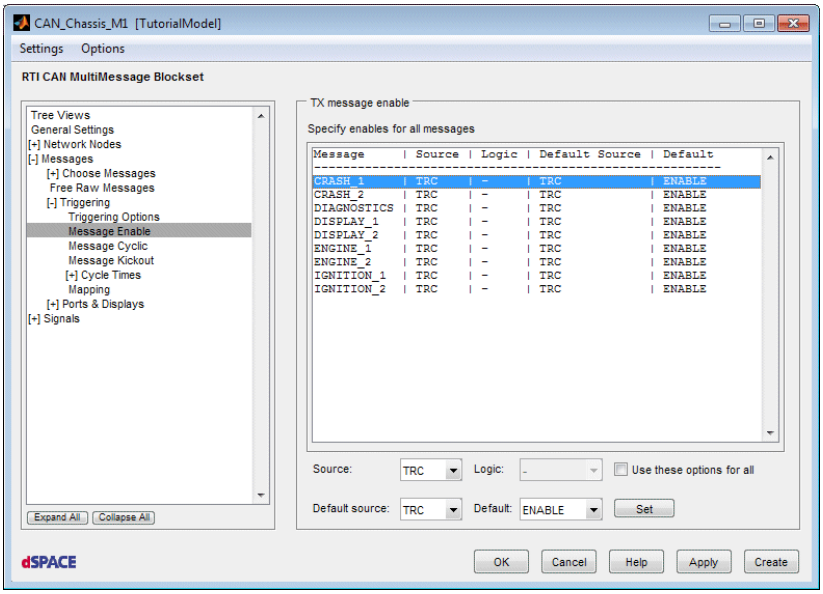


On the Message Enable page, you can enable or disable message transmission by default. You can specify whether the default settings can be modified via your model and/or via ControlDesk during run time. You can specify these settings for all the messages displayed in the Specify enables for all messages list or just for some of them.

5 On the Message Enable page, clear the Use these options for all checkbox.

The Specify enables for all messages list is enabled.

6 In the Specify enables for all messages list, select CRASH_1.



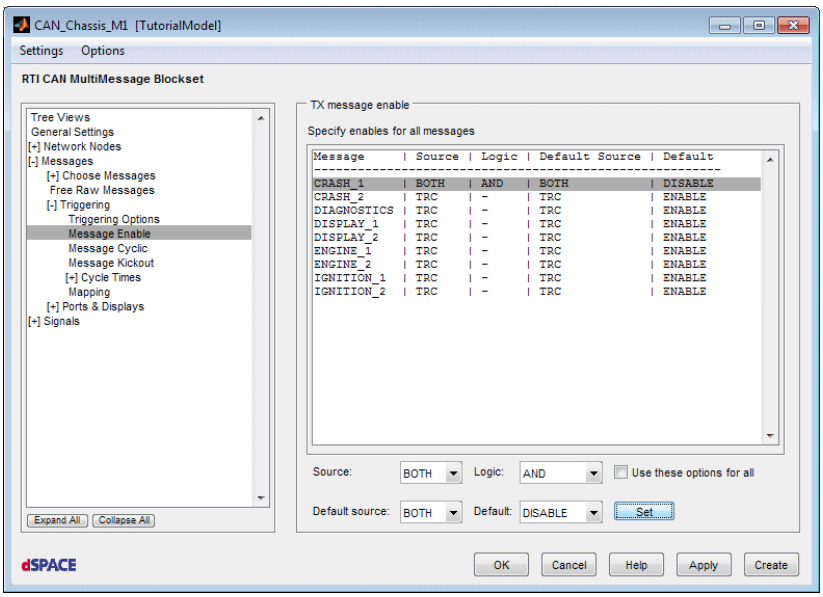
7 Specify the following settings for CRASH_1:

Setting	Selection
Source	BOTH
Default Source	BOTH
Logic	AND
Default	DISABLE

8 Click Set.

Note

You always have to click Set to apply your settings.

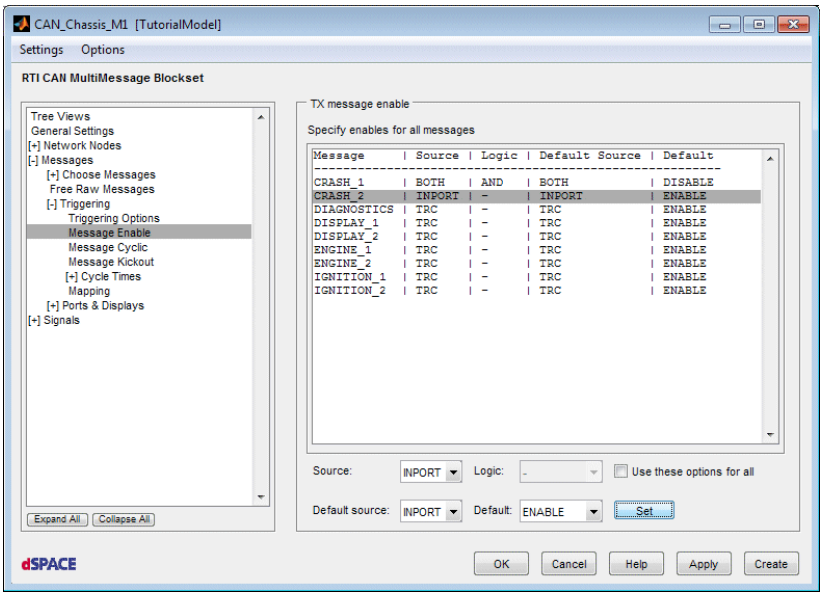


Message transmission for CRASH_1 is disabled by default. During run time, message transmission can be enabled via the model *and* ControlDesk.

9 In the Specify enables for all messages list, select CRASH_2.

10 From the Source drop-down list, select Inport.

11 Click Set.



Inport is the source for CRASH_2. During run time, the transmission of this message can be enabled or disabled only via the model.

Interim result

You specified the default transmission behavior of CRASH_1 and CRASH_2. You can now observe the effects of the specifications on your model. Continue with part 2.

Part 2

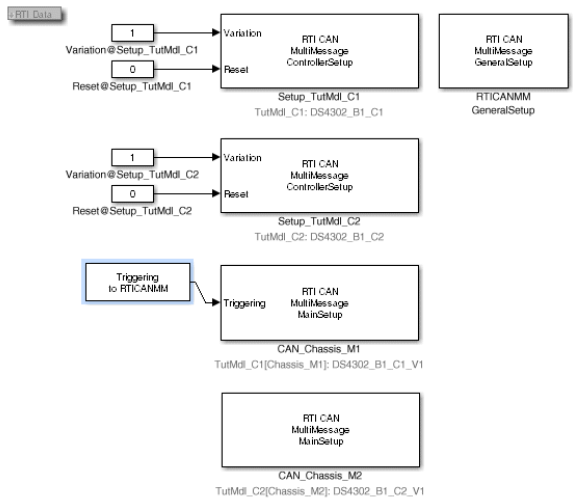
To check the effects of the specified settings on your model

1 In the CAN_Chassis_M1 [TutorialModel] dialog, click Create.

The CAN_Chassis_M1 MainBlock is created, and the specified settings apply to your model.

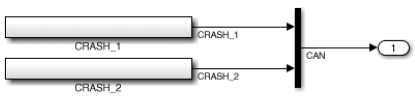
2 Close the CAN_Chassis_M1 [TutorialModel] dialog.

In your model, a Triggering inport is added to the CAN_Chassis_M1 MainBlock. A Triggering to RTICANMM block is automatically connected to the Triggering inport.



3 Open the **Triggering to RTICANMM** block.

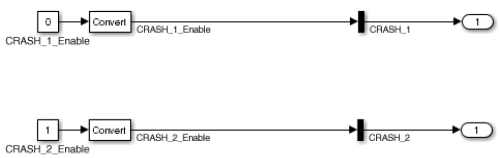
The **TutorialModel/Triggering2CAN_Chassis_M1** subsystem opens.



Since you specified CRASH_1 and CRASH_2 to be enabled or disabled via your model, these messages are the only elements of this subsystem.

4 From the **TutorialModel/Triggering2CAN_Chassis_M1** subsystem, open both messages.

The **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_1** and **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_2** subsystems open.



The Constant blocks in both subsystems indicate the specified default settings. For CRASH_1, you disabled message transmission by default. As a consequence, the Constant block's value is 0. For CRASH_2, message transmission is enabled by default and the Constant block's value is 1.

The subsystems do not indicate whether message transmission can be manipulated only via the model or also via ControlDesk. You also cannot see the logic (AND or OR) if both model and ControlDesk manipulation are activated. You can check these settings only via the **Message Enable** page in the MainBlock.

5 Close all the subsystems.

6 Save your model.

Result

You specified some default settings for message transmission for CRASH_1 and CRASH_2. You disabled the transmission of CRASH_1 by default. During run time, you can enable the message transmission via the model *and* ControlDesk (TRC file). For CRASH_2, you enabled message transmission by default. During run time, this setting can be changed only via the model.

During creation of the MainBlock's S-function, a Triggering inport was added to the block and automatically connected to a Triggering to RTICANMM block. The subsystems of the Triggering to RTICANMM block represent the settings you specified in the MainBlock. Via the Triggering to RTICANMM block, you can easily add your own signals to the inports of the MainBlock without knowing the internal bus structure of the MainBlock.

What's next

In the next step, you can specify cyclic message transmission or transmission via kickout.

Related topics

Basics

Lesson 3: Triggering Message Transmission via the Model and ControlDesk 81
Overview of Lesson 3 81

Step 2: How to Specify Cyclic Message Transmission or Transmission via Kickout

Objective

Messages can be transmitted cyclically with a certain cycle time or via kickout. A kickout lets you transmit the message once immediately. Usually, you want either cyclic message transmission or transmission via kickout. You can therefore disable cyclic message transmission if a message is to be transmitted via kickout.

In this step, you will disable cyclic message transmission for some messages. Additionally, you will let this default setting be modified via your model or ControlDesk during run time.

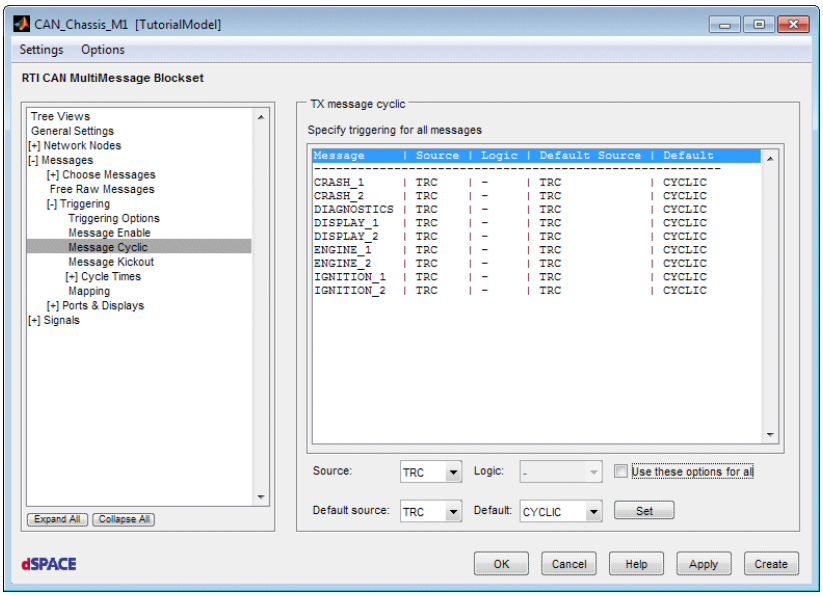
Part 1

To enable and disable cyclic message transmission

1. Open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Messages – Triggering – Message Cyclic.

   On the Message Cyclic page, you can enable or disable cyclic transmission for all the messages displayed in the Specify triggering for all messages list or just for some of them. Additionally, you can specify whether these default settings can be manipulated via your model and/or via ControlDesk.
3. On the Message Cyclic page, clear the Use these options for all checkbox.

The Specify triggering for all messages list is enabled.



All the messages are transmitted cyclically by default. During run time, these settings can be modified only via ControlDesk.

4 In the Specify triggering for all messages list, select CRASH_1.

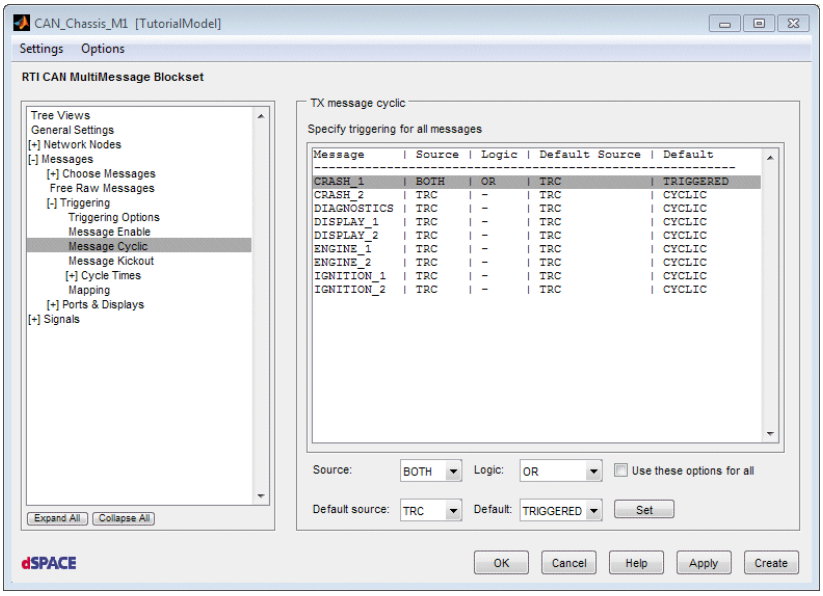
5 Specify the following settings for CRASH_1 via the drop-down lists:

Setting	Selection
Source	BOTH
Default Source	TRC
Logic	OR
Default	TRIGGERED

6 Click Set.

Note

You always have to click Set to apply your settings.



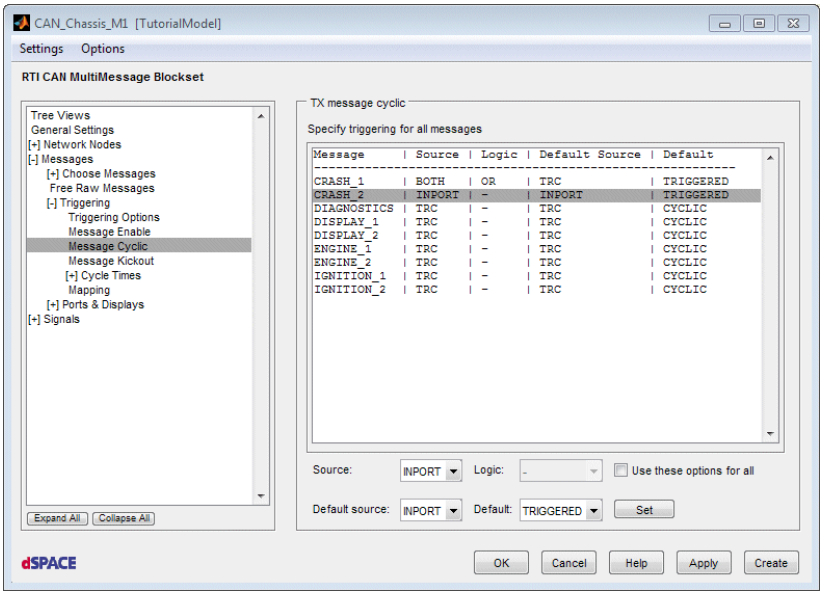
Cyclic message transmission is disabled for CRASH_1 by default. During run time, cyclic transmission can be enabled via the model or ControlDesk.

7 In the Specify triggering for all messages list, select CRASH_2.

8 From the Source drop-down list, select INPORT.

9 From the Default drop-down list, select TRIGGERED.

10 Click Set.



Cyclic message transmission is disabled for CRASH_2 by default. During run time, you can enable or disable cyclic transmission only via the model.

Interim result

You changed the default settings for cyclic message transmission for CRASH_1 and CRASH_2. You can now modify the default settings for transmission via kickout. Continue with part 2.

Part 2

To specify message transmission via kickout

1 Select the Message Kickout page.

On the Message Kickout page, you can specify whether you can kickout messages via your model or via ControlDesk. You can specify these settings for all the messages displayed in the Specify kickout for all messages list or just for some of them.

2 On the Message Kickout page, clear the Use these options for all checkbox.

The Specify kickout for all messages list is enabled.

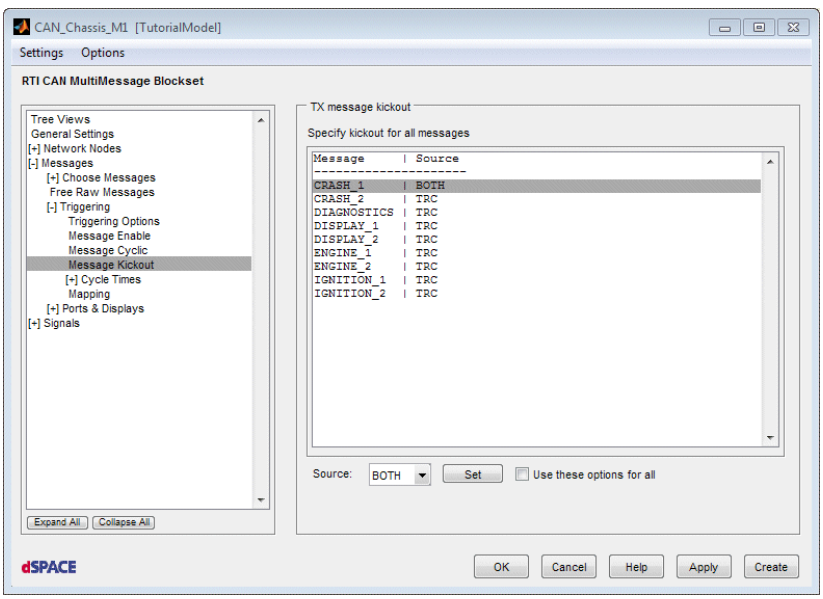
3 In the Specify kickout for all messages list, select CRASH_1.

4 From the Source drop-down list, select BOTH.

5 Click Set.

Note

You always have to click Set to apply your settings.



The trigger for the kickout of CRASH_1 can be set via the model or via ControlDesk.

6 In the Specify kickout for all messages list, select CRASH_2.

7 From the Source drop-down list, select INPORT.

8 Click Set.

The trigger for the kickout of CRASH_2 can be set only via the model.

9 Click **Create**.

The **CAN_Chassis_M1** MainBlock is created, and the specified settings apply to your model. During creation, the **Overwrite?** dialog opens. You are asked whether you want to overwrite the existing **Triggering to RTICANMM** block or add a new one to your model.

10 In the **Overwrite?** dialog, click **Overwrite**.

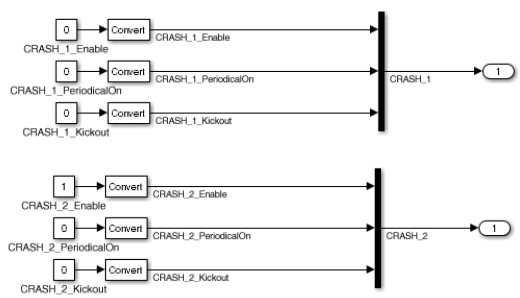
The dialog closes and the settings of the existing Triggering to RTICANMM block are overwritten.

> **Tip**
>
> Unless you change the **Triggering to RTICANMM** block manually, you can always select **Overwrite** in the **Overwrite?** dialog. To save time, you can also select the checkbox **Do not ask again**.

11 In the **CAN_Chassis_M1 [TutorialModel]** dialog, click **OK** to close the dialog.

12 Open the **Triggering to RTICANMM** block and its subsystems to check the modified structure.

For CRASH_1 and CRASH_2, cyclic message transmission and transmission via kickout can be modified via the **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_1** or via the **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_2** subsystem, respectively.



Cyclic message transmission can be modified via the **CRASH_1_PeriodicalOn** and **CRASH_2_PeriodicalOn** Constant blocks. Transmission via kickout can be activated via the **CRASH_1_Kickout** and **CRASH_2_Kickout** Constant blocks.

13 Close all the subsystems.

14 Save your model.

Result

You changed the default settings specified for cyclic message transmission and transmission via kickout for CRASH_1 and CRASH_2. You disabled cyclic message transmission for both messages by default. During run time, you can enable cyclic transmission or transmit CRASH_1 via kickout via your model or ControlDesk. You can enable cyclic transmission or transmit CRASH_2 via kickout only via your model.

Tip

A kickout via the model is triggered only by a positive signal edge.

During creation of the MainBlock's S-function, the subsystems of the Triggering to RTICANMM block were modified according to the settings you specified in the MainBlock.

What's next

In the next step, you can check the effects of the settings you specified in ControlDesk.

Related topics

Basics

Lesson 3: Triggering Message Transmission via the Model and ControlDesk..... 81
Overview of Lesson 3..... 81

Step 3: How to Check Message Transmission Settings in ControlDesk

Objective

All the settings you specified in the previous two steps affect the message layouts in ControlDesk. In this step, you will open the TX and RX layouts of CRASH_1 and CRASH_2 to check the effects of your specifications and to manipulate the default settings.

Part 1

To check message transmission settings of CRASH_1 via ControlDesk

1. Build an application for your model.
2. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
3. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
4. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.
5. From the platform's context menu in the Project controlbar, select Add Variable Description.

   ControlDesk opens the Select Variable Description dialog.

6 In the **Select Variable Description** dialog, click **Import from file** and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

> **Note**
>
> To make sure that you add the latest version of the SDF file to the experiment, always use **Import from file** to import the file. If the **Select Variable Description** dialog displays the `TutorialModel.sdf` file and you select this file directly, you might load an obsolete version. This might cause problems while you work through this tutorial.

7 In the **Select Variable Description** dialog, click **OK**.

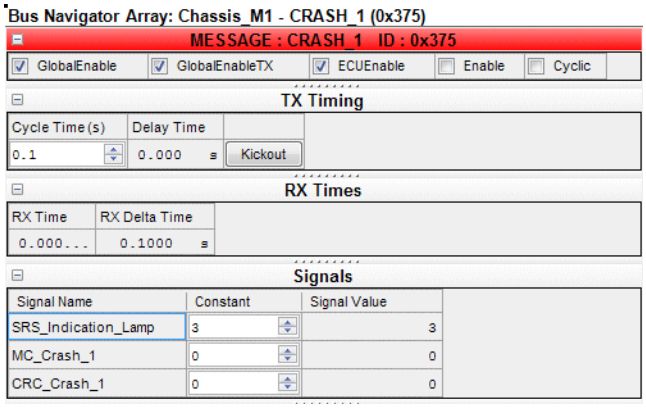
ControlDesk closes the **Select Variable Description** dialog and adds the variable description (SDF file) to the platform.

8 In the Bus Navigator tree, right-click the **CRASH_1** message below the **Chassis_M1 [chassis.dbc]** node and open the RX and TX layouts.

9 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.

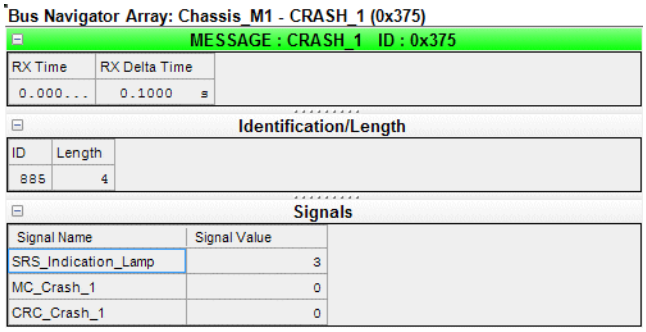
The layouts change according to your specified settings:

- *TX layout*: Since you disabled message transmission by default, **Enable** is cleared and the message cannot be transmitted. The header of the layout therefore remains red and the **RX Time** is not running. Since you specified a triggered message transmission, **Cyclic** is cleared by default.



> **Tip**
>
> In your model, open the Chassis_M1 MainBlock and check the settings you specified on the Message Enable, Message Cyclic, and Message Kickout pages.

- *RX layout*: Since the message can be received, the header of the RX layout changed from red to green. Since the **RX Time** is not running, the message is not being received at the moment.



Interim result

You checked the effects of your specifications on the layouts of CRASH_1. You can now modify the default settings for CRASH_1 during run time. Continue with part 2.

Part 2

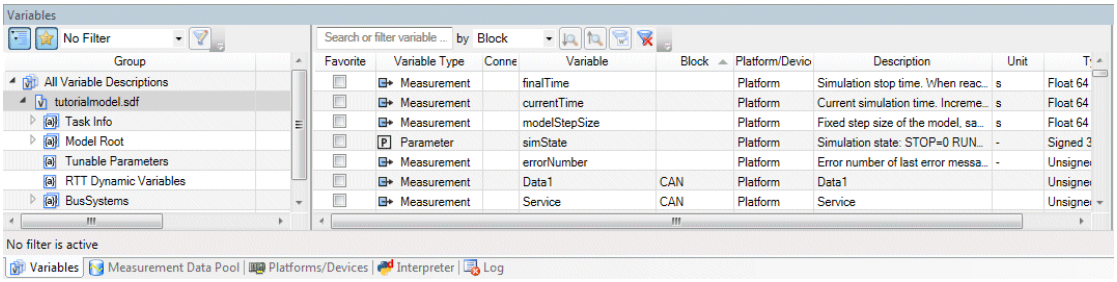
To enable message transmission for CRASH_1 during run time

1 In the TX layout, select Enable.

The header remains red and the message cannot be transmitted.

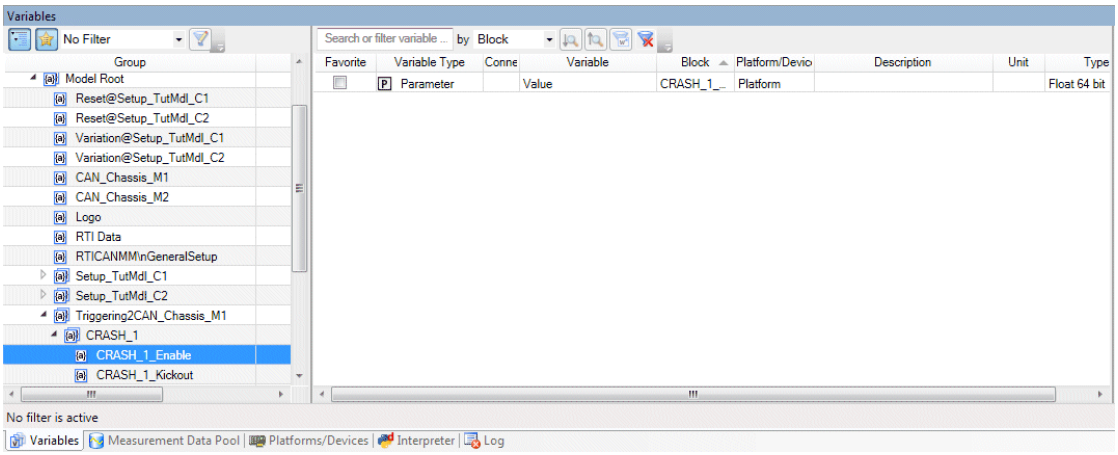
In your model, you specified that message transmission must be enabled via ControlDesk *and* via the model. Since message transmission is not enabled via the model yet, the message cannot be transmitted.

2 Select the Variables controlbar.



The tree view on the left displays the structure of the TRC file referenced by the model's SDF file, the variable list on the right displays the variables of the TRC file.

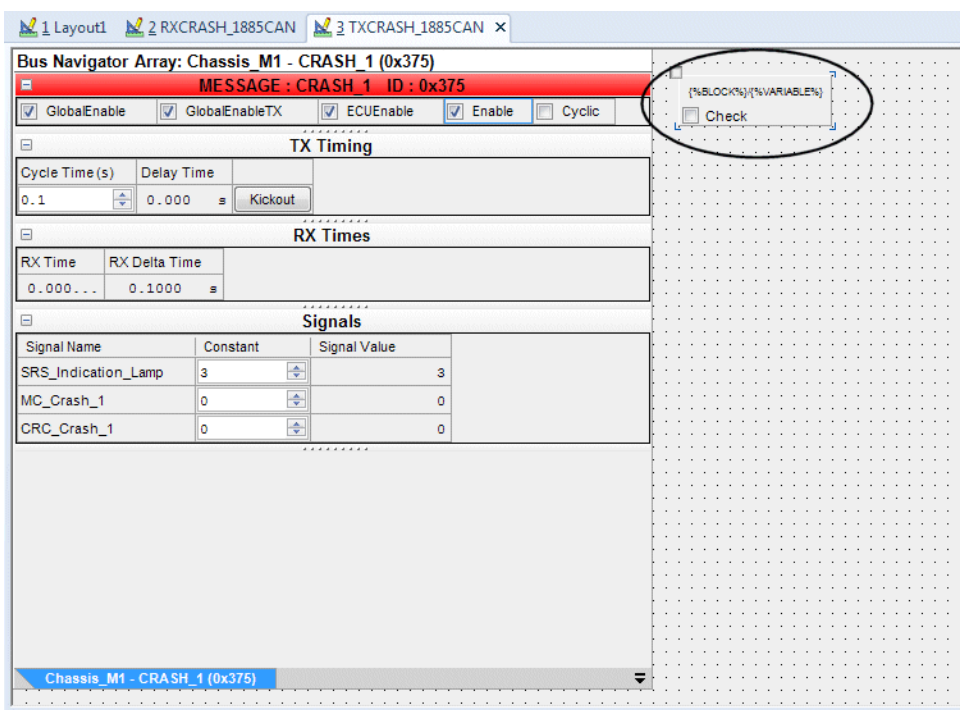
3 In the tree view, select tutorialmodel.sdf – Model Root – Triggering2CAN_Chassis_M1 – CRASH_1 – CRASH_1_Enable.



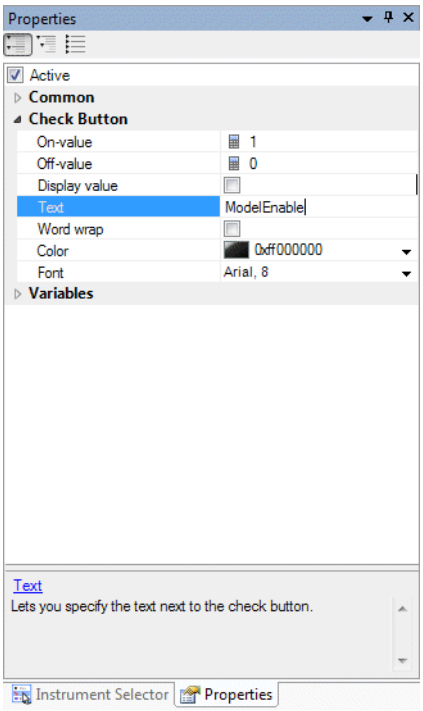
The variable list displays the variable value.

4 From the Instrument Selector, add a Check Button to the TX layout of CRASH_1.

> **Tip**
>
> If the Instrument Selector controlbar is not displayed, go to the View ribbon and select Controlbar – Switch Controlbars – Instrument Selector.

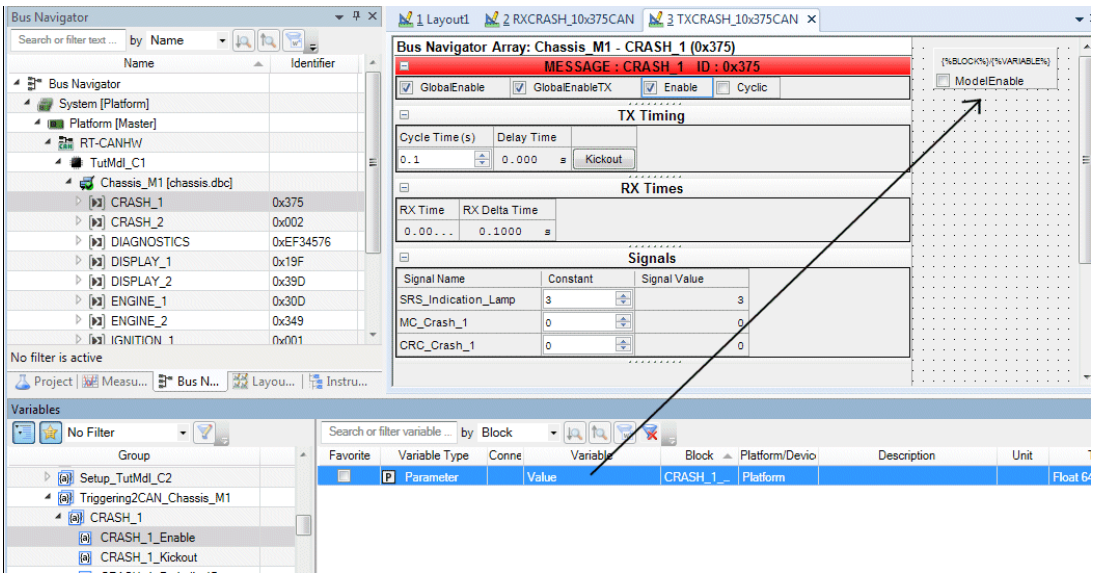


5 Select the Check Button and change the text in its properties to **ModelEnable**.



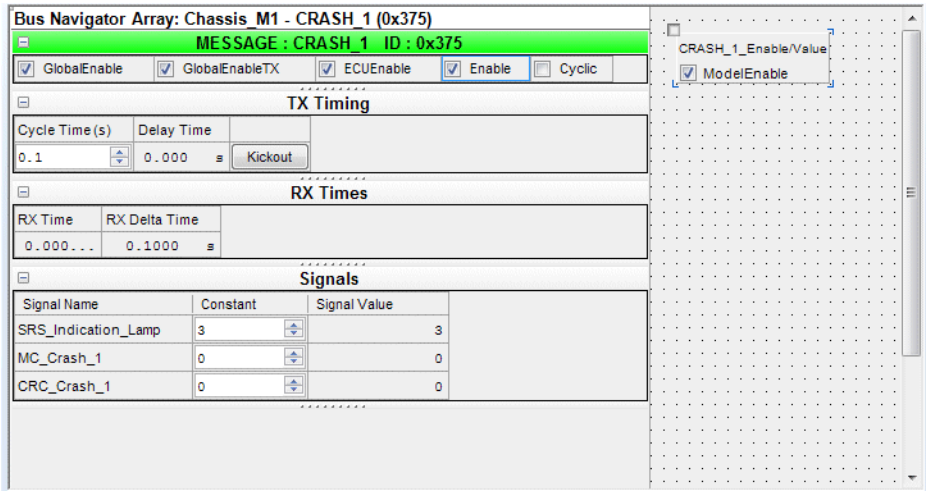
6 Select the Check Button again. The Check Button is renamed ModelEnable.

7 From the variable list of the Variables controlbar, drag the Value parameter to the ModelEnable button.



A data connection is created between the parameter and the button, and the main caption of the instrument is changed to CRASH_1_Enable/Value.

8 In the TX layout, select ModelEnable.



The header of the layout changes from red to green. Now the message can be transmitted. Since cyclic transmission is still disabled, the message is not transmitted yet and the RX time is not running.

9 Select Cyclic.

The message is transmitted via cyclic transmission, the RX time is running.

In your model, you specified to enable cyclic message transmission via the model *or* via ControlDesk. Enabling cyclic message transmission via ControlDesk therefore lets you transmit the message cyclically.

10 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

11 Save your project. The TX layout you changed in this step is also saved with the experiment.

12 Close both layouts.

Interim result

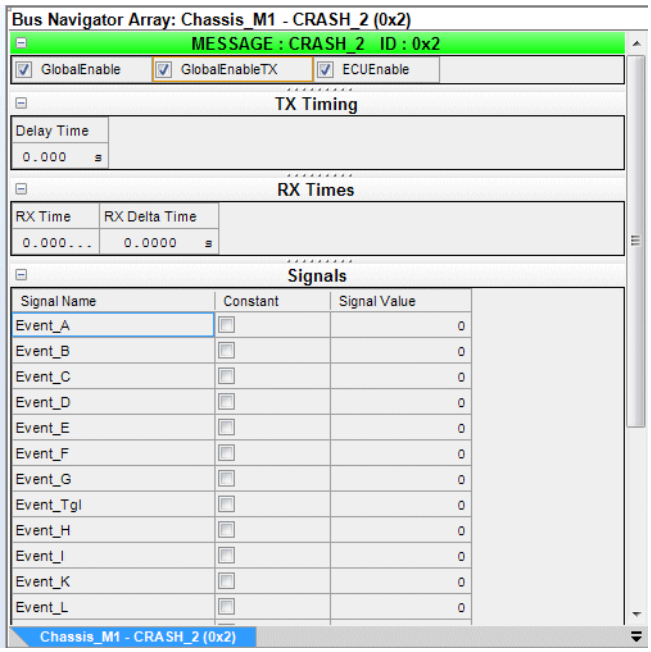
You modified the default settings for CRASH_1 during run time. You can now check the effects of your specifications on the layouts of CRASH_2. Continue with part 3.

Part 3

To check message transmission settings of CRASH_2 via ControlDesk

1 In the Bus Navigator tree, right-click the CRASH_2 message below the Chassis_M1 [chassis.dbc] node and generate the TX layout.

2 On the Home ribbon, click Status Control – Go Online to start online calibration.



In your model, you specified that all the message triggering options can be manipulated only via the model, so there are no Enable, Cyclic, and Kickout buttons. Since you enabled message transmission by default, the message can be transmitted. Since you disabled cyclic transmission by default, the message is not being transmitted yet.

3 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

4 Save the project.

Result

You checked the effects of the message transmission settings you specified in the model on the layouts:

- All the message transmission settings you disabled by default are disabled in TX layouts.
- Settings you specified to be manipulated via ControlDesk during run time can be enabled and disabled via TX layouts.
- Settings you specified to be manipulated via the model only are not represented in TX layouts.
- Settings you specified to be manipulated via the model *and* via ControlDesk must always be manipulated via both sources to change the default setting.

What's next

With this step, you finished the triggering of message transmission via the model and via ControlDesk. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 3: Triggering Message Transmission via the Model and ControlDesk..... 81
Overview of Lesson 3..... 81

Result of Lesson 3

Result

In this lesson, you learned the basic principles of manipulating message transmission during run time:

Changing default message transmission options You can change the default message transmission options of the RTICANMM MainBlocks via the Message Enable, Message Cyclic, and Message Kickout pages. For example, you can disable message transmission by default or specify transmission via kickout instead of cyclic transmission.

Manipulating message transmission via your model and/or ControlDesk During run time, you can change the default settings specified for message transmission. You can select your model and/or ControlDesk as the manipulation source for this.

- *INPORT*: Lets you manipulate message transmission via your model. To manipulate message transmission during run time, an inport is added to the RTICANMM MainBlock and automatically connected to a Triggering to RTICANMM block.
- *TRC*: Lets you manipulate message transmission via ControlDesk. You can manipulate message transmission during run time via the TX layout of the selected message.
- *BOTH*: Lets you manipulate message transmission via your model and/or via ControlDesk. You have to specify the logical combination of these two sources (AND or OR).

Triggering to RTICANMM block The Triggering to RTICANMM block is created automatically when you select INPORT or BOTH as the manipulation source. The block is structured according to the specifications you make on the Message Enable, Message Cyclic and/or Message Kickout pages. You can specify settings for the Triggering to RTICANMM block, for example, disable the Overwrite? dialog, on the Peripheral Options page. You can customize the internal structure of this block to connect signals of your model to the RTICANMM MainBlock.

Note

If you customize the internal structure of the Triggering to RTICANMM block, you cannot use the automatic update mechanism available during the create process of the RTICANMM MainBlock. This might lead to data inconsistencies. To avoid data inconsistencies and to facilitate MainBlock updates, you can use model-specific bus hierarchies instead (refer to Lesson 5: Working with Model-Specific Bus Hierarchies on page 117).

Effects of message transmission settings on layouts In ControlDesk, you can check the effects of your settings on the message layouts. The automatically generated layouts let you only manipulate settings for which you specified TRC or BOTH as the manipulation source. If you specified BOTH and AND as the logical combination for a certain setting, you have to manipulate it via your model *and* ControlDesk to change it.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson3_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Dialog Tree (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Tree Views Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Cyclic Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Kickout Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Peripheral Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Triggering to RTICANMM Block (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to learn the basic principles of manipulating signal values via your model and/or via ControlDesk.

Related topics

Basics

Lesson 3: Triggering Message Transmission via the Model and ControlDesk..... 81
Overview of Lesson 3..... 81

Lesson 4: Manipulating Signal Values via the Model and ControlDesk

Where to go from here

Information in this section

Overview of Lesson 4 103
In this lesson, you will learn the basic principles of manipulating signal values via the model and/or via ControlDesk.

Step 1: How to Specify Signals to be Manipulated via the Model 105
In this step, you will specify signals to be manipulated via your model. You must specify them as TX model signals.

Step 2: How to Enable Manipulation of TX Model Signals via ControlDesk 109
In this step, you will enable signal manipulation via ControlDesk for some of the TX model signals.

Result of Lesson 4 114
In this lesson, you learned the basic principles of manipulating signal values via the model and/or via ControlDesk.

Overview of Lesson 4

Introduction

By default, all the signals of all TX messages (TX signals) can be manipulated via ControlDesk. However, you can specify TX signals to be manipulated via the model only. You have to specify them as TX model signals for this purpose. If you want to manipulate TX model signals via your model *and* via ControlDesk, you have to enable this option separately.

What you will learn	In this lesson you will learn how to specify TX model signals and let them be manipulated via ControlDesk.
Starting point	The starting point of this lesson is the **TutorialModel** model you created in lesson 3.
Prepared demo	If you did not create your own model in lesson 3, you can use the prepared demo **Lesson3_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.
Before you begin	Before you begin this lesson, the following preconditions must be met: ▪ The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open. ▪ You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56). ▪ The RTICANMMTutorialProject must be open and you have to know how to create an experiment in ControlDesk. Refer to How to Create an Experiment in ControlDesk on page 483).
Steps	This lesson contains the following steps: ▪ Step 1: How to Specify Signals to be Manipulated via the Model on page 105 ▪ Step 2: How to Enable Manipulation of TX Model Signals via ControlDesk on page 109
Result and further information	To check the correctness of the executed steps and for more information, refer to Result of Lesson 4 on page 114.
Duration	Working through this lesson will take you about 30 minutes.
What's next	Now you can start to specify signals to be manipulated via your model, see Step 1: How to Specify Signals to be Manipulated via the Model on page 105.
Related topics	Basics Lesson 4: Manipulating Signal Values via the Model and ControlDesk..... 103

Step 1: How to Specify Signals to be Manipulated via the Model

Objective

By default, all the TX signals of your model can be manipulated via ControlDesk. If you want to manipulate TX signals from within the model, you have to specify them as TX model signals.

In this step, you will select the TX model signals.

Part 1

To specify signals to be manipulated via the model

1. In your Simulink model, open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Signals – TX – Model Signals (TX).

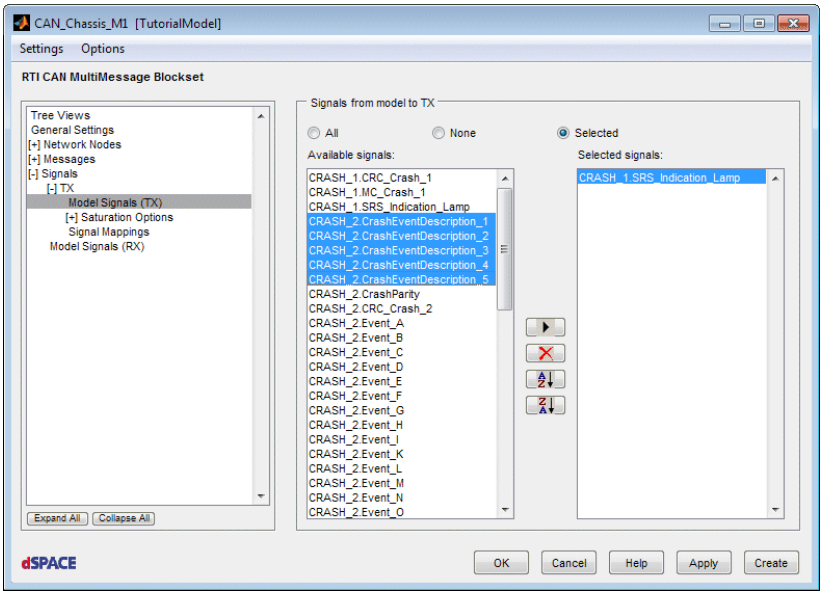
   On the Model Signals (TX) page, you can specify signals to be manipulated via your model. Since TX model signals can reduce performance, you should always specify as few TX model signals as possible. By default, TX model signals cannot be modified via ControlDesk but only via the model.
3. On the Model Signals (TX) page, select Selected.

   The Available signals list is enabled.
4. In the Available signals list, select CRASH_1.SRS_Indication_Lamp.
5. Click .

   CRASH_1.SRS_Indication_Lamp is added to the Selected signals list.
6. In the Available signals list, select all the signals from CRASH_2.CrashEventDescription_1 to CRASH_2.CrashEventDescription_5.

   Tip

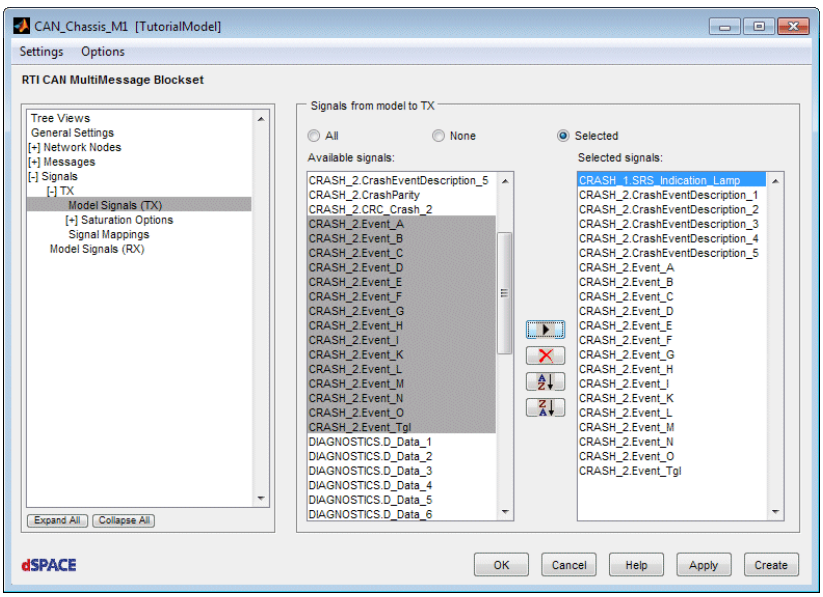
   You can use **Shift** and **Ctrl** for multiselection.



7 Click to add the signals to the Selected signals list.

8 In the Available signals list, select all the signals from CRASH_2.Event_A to CRASH_2.Event_Tgl.

9 Click to add the signals to the Selected signals list.



Interim result

You specified some TX signals as TX model signals. These signals can be manipulated via your model. You can now check the effects of these settings on your model. Continue with part 2.

Part 2

To check the effects of the specified settings on your model

1 In the CAN_Chassis_M1 [TutorialModel] dialog, click Create.

The CAN_Chassis_M1 MainBlock is created, and the specified settings apply to your model. During creation, the Overwrite? dialog opens. You are asked whether you want to overwrite the existing Triggering to RTICANMM block or add a new one to your model.

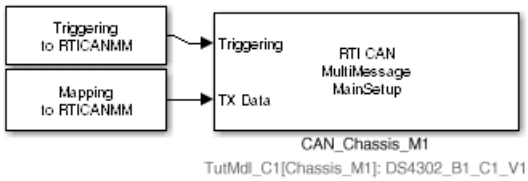
2 In the Overwrite? dialog, click Overwrite.

The dialog closes and the settings of the existing Triggering to RTICANMM block are overwritten.

> **Tip**
>
> Unless you change the Triggering to RTICANMM block manually, you can always select Overwrite in the Overwrite? dialog. To save time, you can also select the checkbox Do not ask again.

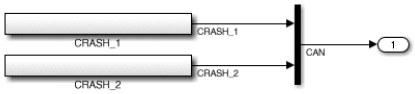
3 In the CAN_Chassis_M1 [TutorialModel] dialog, click OK to close the dialog.

In your model, a TX Data inport is added to the CAN_Chassis_M1 MainBlock. A Mapping to RTICANMM block is automatically connected to the TX Data inport.



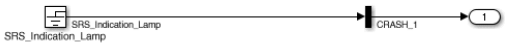
4 In your model, open the Mapping to RTICANMM block.

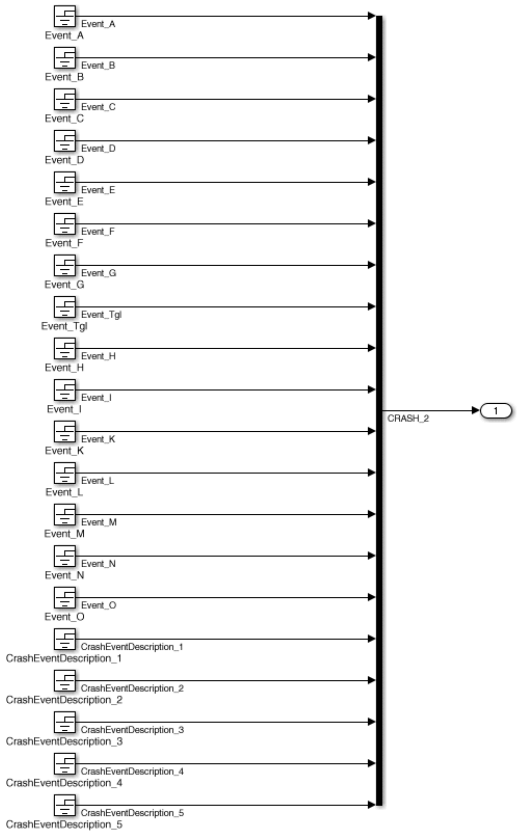
The TutorialModel/User2CAN_Chassis_M1 subsystem opens. Since you specified signals of CRASH_1 and CRASH_2 to be manipulated via your model, these messages are the only elements of this subsystem.



5 From the TutorialModel/User2CAN_Chassis_M1 subsystem, open both messages.

The TutorialModel/User2CAN_Chassis_M1/CRASH_1 and TutorialModel/User2CAN_Chassis_M1/CRASH_2 subsystems open.





A ground block has been created for each specified TX model signal and the signals are connected to a Bus Creator.

6 Close all the subsystems without saving.

7 Save your model.

Result

You selected some TX signals to be manipulated via your model. These TX model signals cannot be manipulated via ControlDesk.

During creation of the MainBlock's S-function, a TX Data inport was added to the block and automatically connected to a Mapping to RTICANMM block. The subsystems of the Mapping to RTICANMM block represent the settings you specified in the MainBlock.

What's next

In the next step, you can select TX model signals to be manipulated via the model *and* via ControlDesk.

Related topics

Basics

Lesson 4: Manipulating Signal Values via the Model and ControlDesk..... 103
Overview of Lesson 4..... 103

Step 2: How to Enable Manipulation of TX Model Signals via ControlDesk

Objective

By default, TX model signals cannot be manipulated via ControlDesk. Manipulation of TX model signals via the model *and* via ControlDesk must be enabled separately.

In this step, you will enable signal manipulation via ControlDesk for some of the TX model signals.

Part 1

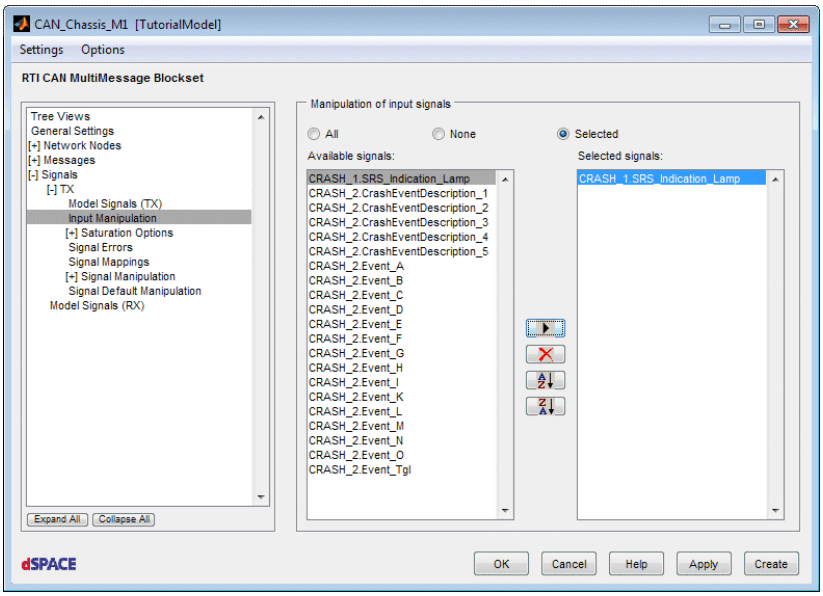
To enable manipulation of TX model signals via ControlDesk

1. Open the CAN_Chassis_M1 MainBlock.
2. On the Tree Views page, select Manipulation Interface.
3. In the dialog tree, select Signals – TX – Input Manipulation.

   On the Input Manipulation page, you can enable manipulation of TX model signals via ControlDesk.
4. On the Input Manipulation page, select Selected.

   The Available signals list is enabled. All the signals you selected on the Model Signals (TX) page are displayed.
5. In the Available signals list, select CRASH_1.SRS_Indication_Lamp.

6 Click .



CRASH_1.SRS_Indication_Lamp is added to the **Selected signals** list. This signal can be modified via ControlDesk.

7 In the **CAN_Chassis_M1 [TutorialModel]** dialog, click **Create**.

The **CAN_Chassis_M1** MainBlock is created, and the specified settings apply to your model. During creation, the **Overwrite?** dialog opens twice. You are asked whether you want to overwrite the existing **Mapping to RTICANMM** and **Triggering to RTICANMM** blocks or add new ones to your model.

8 In both **Overwrite?** dialogs, click **Overwrite**.

The dialogs are closed and the settings of the existing Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten.

9 In the **CAN_Chassis_M1 [TutorialModel]** dialog, click **OK** to close the dialog.

10 Save your model.

Interim result

You have enabled signal manipulation via ControlDesk for the CRASH_1.SRS_Indication_Lamp signal. You can now check the effects of the specified settings in ControlDesk. Continue with part 2.

Part 2

To check the specified settings for TX model signals in ControlDesk

1 Build an application for your model.

2 Create an experiment in the **RTICANMMTutorialProject** project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.

3 From the context menu of the registered real-time hardware in the **Platforms/Devices** controlbar, select **Add to active Experiment** to add the platform to the new experiment.

   The **Define Platform Name** dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.

4 Click **OK** to confirm the default platform name.

   The registered platform is added to the experiment.

5 From the platform's context menu in the **Project** controlbar, select **Add Variable Description**.

   ControlDesk opens the **Select Variable Description** dialog.

6 In the **Select Variable Description** dialog, click **Import from file** and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

   Note

   To make sure that you add the latest version of the SDF file to the experiment, always use **Import from file** to import the file. If the **Select Variable Description** dialog displays the `TutorialModel.sdf` file and you select this file directly, you might load an obsolete version. This might cause problems while you work through this tutorial.

7 In the **Select Variable Description** dialog, click **OK**.

   ControlDesk closes the **Select Variable Description** dialog and adds the variable description (SDF file) to the platform.

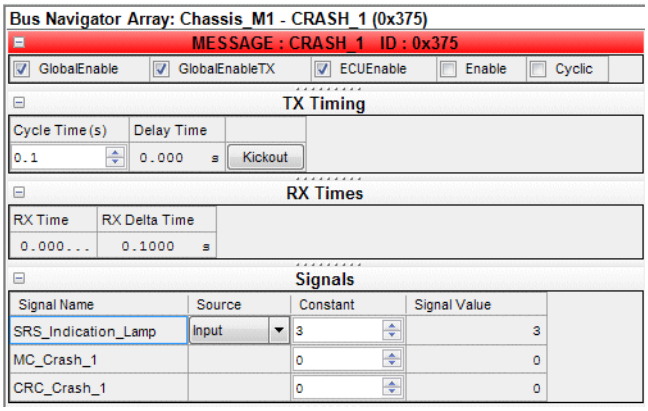
8 In the Bus Navigator tree, right-click the **CRASH_1** message below the **Chassis_M1 [chassis.dbc]** node and open the TX layout.

   Tip

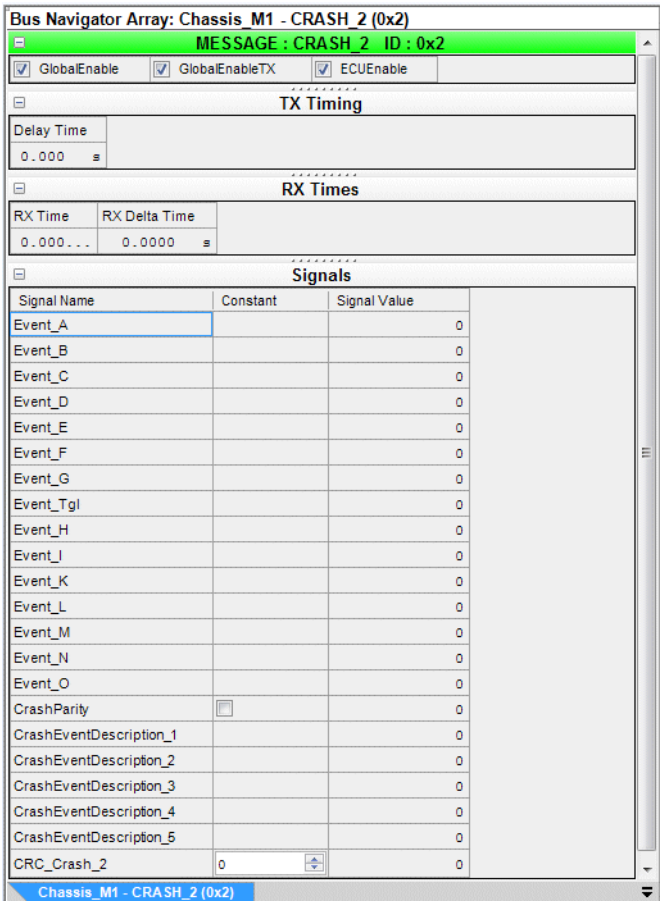
   In the previous lesson, you added a CheckButton named ModelEnable to the TX layout. Since you created a new experiment and generated a new TX layout, the new TX layout is generated according to the model specifications, and all the manual changes are lost.

9 Right-click the **CRASH_2** message below the **Chassis_M1 [chassis.dbc]** node and open the TX layout.

10 On the Home ribbon, click Status Control – Go Online to start online calibration.



CRASH_1: All the signals displayed in the TX layout of CRASH_1 can be modified. Since you selected SRS_Indication_Lamp on the Signal Manipulation page in your model, you can modify this signal via ControlDesk even though it is a TX model signal. The Source drop-down list lets you select whether the signal should be manipulated via the model or via ControlDesk. By default, the signal can be manipulated via the model. Since the MC_Crash_1 and CRC_Crash_1 signals are no TX model signals, you cannot select a source for them. They can be manipulated via ControlDesk only.



CRASH_2: In the TX layout of CRASH_2, a Constant value that can be modified is displayed only for CrashParity and CRC_Crash_2. In the model, you specified all the other signals as TX model signals. Since you selected none of these signals on the Input Manipulation page, you can modify them only via the model but not via ControlDesk.

11 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

12 Save the project.

Result

You selected one TX model signal to be manipulated via your model *and* via ControlDesk. In ControlDesk, you can select the signal manipulation source via Source. All the other signals you specified as TX model signals cannot be modified via ControlDesk.

What's next

With this step, you finished the manipulation of signal values via the model and via ControlDesk. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 4: Manipulating Signal Values via the Model and ControlDesk..... 103
Overview of Lesson 4..... 103

Result of Lesson 4

Result

In this lesson, you learned the basic principles of manipulating signals via your model:

TX model signals Only TX model signals can be manipulated via your model. Since they can reduce performance, you should specify as few TX model signals as possible. If you want to manipulate a TX model signal via ControlDesk as well, you have to select it on the Input Manipulation page.

TX Data inport When you specify TX model signals, a TX Data inport is added to your MainBlock during the create process. This inport is automatically connected to a Mapping to RTICANMM block.

Mapping to RTICANMM block The structure of the Mapping to RTICANMM block is adapted to the specifications you make on the Model Signals (TX) page. You can customize the internal structure of this block to connect signals of your model to the specified TX model signals. To specify a TX model signal from within the model, you have to replace the Simulink Ground block in the substructure of the Mapping to RTICANMM. You can replace the Simulink Ground block by another suitable Simulink block (for example, a Simulink Constant block or a Simulink Data Type Conversion block) or another subsystem. You can specify settings for the Mapping to RTICANMM block, for example, disable the Overwrite? dialog, on the Peripheral Options page.

Note

If you customize the internal structure of the Mapping to RTICANMM block, you cannot use the automatic update mechanism available during the create process of the RTICANMM MainBlock. This might lead to data inconsistencies. To avoid data inconsistencies and to facilitate MainBlock updates, you can use model-specific bus hierarchies instead (refer to Lesson 5: Working with Model-Specific Bus Hierarchies on page 117).

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson4_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Model Signals (TX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Input Manipulation Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Peripheral Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Mapping to RTICANMM Block (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to learn the basic principles of working with model-specific bus hierarchies.

Related topics

Basics

Lesson 4: Manipulating Signal Values via the Model and ControlDesk..... 103
Overview of Lesson 4..... 103

Lesson 5: Working with Model-Specific Bus Hierarchies

Where to go from here

Information in this section

Overview of Lesson 5 118
In this lesson, you will learn the basic principles of working with model-specific bus hierarchies.

Step 1: How to Add a Simulink Bus to Your Model 120
In this step, you will create a new Simulink bus in your model.

Step 2: How to Create an Inport for the Triggering to RTICANMM Block 122
In this step, you will select messages to be manipulated by the Simulink bus and let an inport be created for the Triggering to RTICANMM block.

Step 3: How to Create an Inport for the Mapping to RTICANMM Block 127
In this step, you will select signals to be manipulated by the Simulink bus and let an inport be created for the Mapping to RTICANMM block.

Step 4: How to Connect the Simulink Bus 130
In this step, you will connect the Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks.

Result of Lesson 5 132
In this lesson, you learned the basic principles of working with model-specific bus hierarchies.

Overview of Lesson 5

Introduction

Messages and/or signals specified in an RTICANMM MainBlock can be manipulated from within the model. You can, for example, connect signals of your model to the MainBlock to trigger the transmission of messages. This is done by connecting model-specific bus hierarchies to the MainBlock.

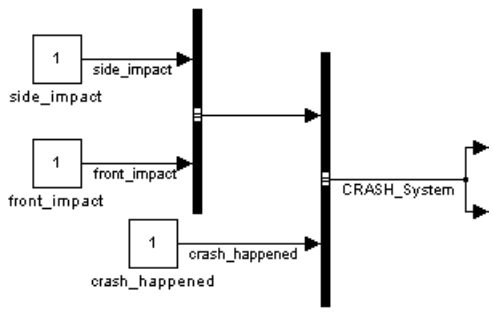
Connecting signals to RTICANMM MainBlocks Whenever you select messages and signals specified in an RTICANMM MainBlock to be manipulated via your model, inports are created for the MainBlock and automatically connected to a Triggering to RTICANMM and/or Mapping to RTICANMM block (for details, refer to Lesson 3: Triggering Message Transmission via the Model and ControlDesk on page 81 and Lesson 4: Manipulating Signal Values via the Model and ControlDesk on page 103).

Using these two blocks, you can manipulate messages and signals specified in the RTICANMM MainBlock in two ways:

- You can customize the internal structures of the Triggering to RTICANMM and Mapping to RTICANMM blocks to connect signals of your model to them. This makes it easy to add your own signals to the inports of the MainBlock without knowing its internal bus structure. If you customize the internal structures of the Triggering to RTICANMM and Mapping to RTICANMM blocks, you cannot use the automatic update mechanism available during the create process of the MainBlock. This might lead to data inconsistencies.
- You can connect model-specific bus hierarchies to the Triggering to RTICANMM and Mapping to RTICANMM blocks. When using model-specific bus hierarchies, you can also use the automatic update mechanisms available during the create process of the MainBlock. This prevents data inconsistencies. To use model-specific bus hierarchies, you can let inports be created for the Triggering to RTICANMM and Mapping to RTICANMM blocks. Via these blocks, you can connect the model-specific bus hierarchies to your MainBlock and directly map them to messages and signals specified in the MainBlock.

Behavior of the Simulink bus in this lesson

In this lesson, you will create a simple Simulink bus which triggers the transmission of the messages CRASH_1 and CRASH_2 and their signals.



- *Triggering CRASH_1 and CRASH_2*: To simulate a crash, the crash_happened Constant block triggers the transmission of CRASH_1 and CRASH_2 via kickout.

- *Determining signal values of CRASH_2*: To simulate a side impact, the side_impact Constant block determines the value of Event_A. To simulate a front impact, the front_impact Constant block determines the value of Event_B.

What you will learn

In this lesson, you will learn how to work with model-specific bus hierarchies.

- You will select messages to be manipulated by a Simulink bus and let an inport be created for the Triggering to RTICANMM block.
- You will select signals to be manipulated by a Simulink bus and let an inport be created for the Mapping to RTICANMM block.
- You will add a Simulink bus to your model.
- You will connect the Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 4.

Prepared demo

If you did not create your own model in lesson 4, you can use the prepared demo **Lesson4_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).

Steps

This lesson contains the following steps:

- Step 1: How to Add a Simulink Bus to Your Model on page 120
- Step 2: How to Create an Inport for the Triggering to RTICANMM Block on page 122
- Step 3: How to Create an Inport for the Mapping to RTICANMM Block on page 127
- Step 4: How to Connect the Simulink Bus on page 130

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 5 on page 132.

Duration

Working through this lesson will take you about 40 minutes.

What's next

Now you can start to add a model-specific bus hierarchy to your model, see Step 1: How to Add a Simulink Bus to Your Model on page 120.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117

Step 1: How to Add a Simulink Bus to Your Model

Objective

There are several ways to add a Simulink bus to your model, for example, you can copy an existing Simulink bus to your model or create a new bus. In this step, you will create a new Simulink bus in your model.

Tip

If you do not want to create a new Simulink bus, copy the CRASH_System subsystem from the demo model Lesson5_TutorialModel. In this case, skip this step and continue with Step 2: How to Create an Inport for the Triggering to RTICANMM Block on page 122.

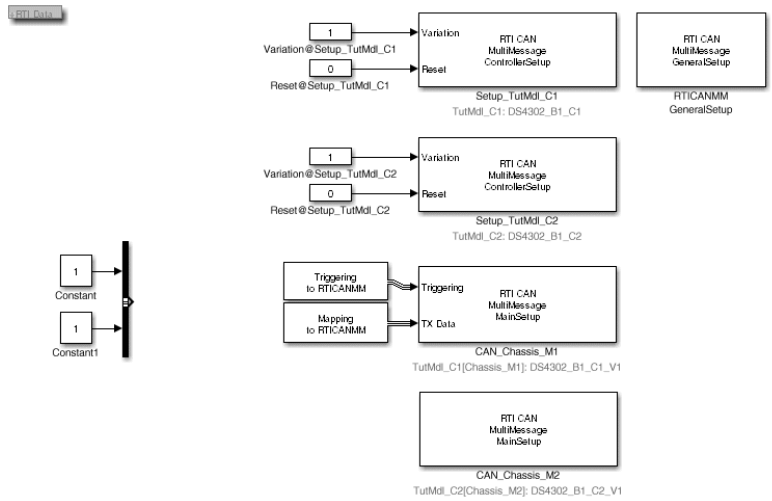
Method

To add a Simulink bus to your model

1. On the Simulation ribbon, click Library – Library Browser.

   The Simulink Library Browser opens.
2. Select the Commonly Used Blocks and drag the following blocks to your model:
   - Two Constant blocks.
   - One Bus Creator block.

3 Connect the two Constant blocks with the inports of the Bus Creator.

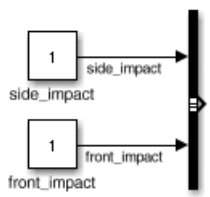


4 Rename the upper Constant block **`side_impact`**.

5 Name the connection between the side_impact Constant block and the Bus Creator **`side_impact`**.

6 Rename the lower Constant block **`front_impact`**.

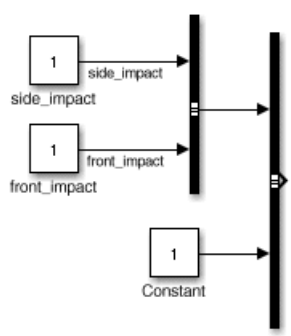
7 Name the connection between the front_impact Constant block and the Bus Creator **`front_impact`**.



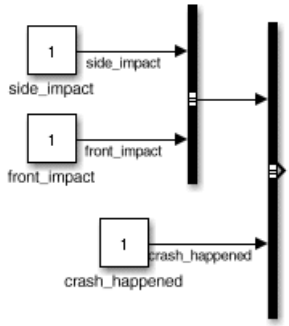
8 From the Commonly Used Blocks, drag another Bus Creator and another Constant block to your model.

9 Connect the outport of the first Bus Creator with the upper inport of the second Bus Creator.

10 Connect the new Constant block with the lower inport of the second Bus Creator.



11 Name the new Constant block and its connection to the Bus Creator `crash_happened`.



12 Save your model.

Result

You added a Simulink bus to your model.

What's next

In the next step, you can select messages to be manipulated by the Simulink bus and let an inport be created for the Triggering to RTICANMM block.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117
Overview of Lesson 5..... 118

Step 2: How to Create an Inport for the Triggering to RTICANMM Block

Objective

To connect a Simulink bus to your RTICANMM blocks, you have to let inports be created.

You can let inports be created for the Triggering to RTICANMM block by specifying the triggering conditions of messages. You can select the enable, cyclic, or kickout triggering condition of a message to be manipulated by the bus. To select a triggering condition, you have to enable manipulation via the model on the Message Enable, Message Cyclic, or Message Kickout page for it.

In this step, you will let CRASH_1 and CRASH_2 be transmitted via kickout by the crash_happened Constant block.

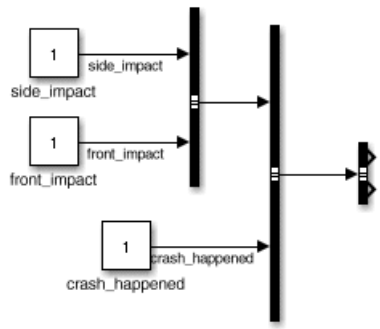
Part 1

To create an inport for the Triggering to RTICANMM block

1 Open the Simulink Library Browser.

2 From the Commonly Used Blocks, drag a Bus Selector to your model.

3 Connect the outport of your Bus Creator with the inport of the Bus Selector. If you work with the CRASH_System subsystem, connect the subsystem with the Bus Selector.



4 Name the connection between the Bus Creator and the Bus Selector `CRASH_System`.

5 Open the CAN_Chassis_M1 MainBlock.

6 In the dialog tree, select Messages – Triggering – Mapping.

On the Mapping page, you can specify a mapping structure for your TX messages. This allows you, for example, to connect signals from the model to the TX messages to trigger message transmission.

7 On the Mapping page, select User-defined mapping.

The User-defined mapping list is enabled

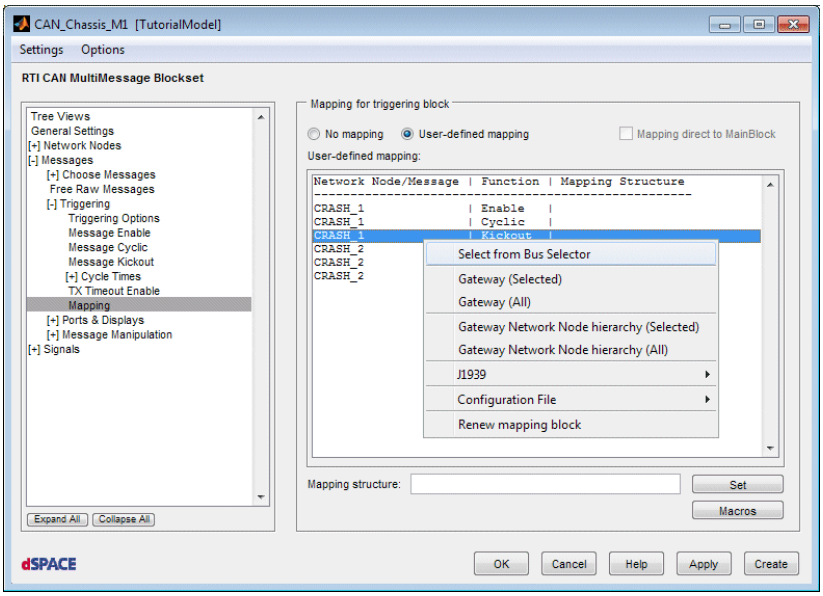
8 In your model, select the Bus Selector.

Note

To select the Bus Selector, you have to click it directly. Do not select the Bus Selector via a selection frame.

9 In the User-defined mapping list of the Mapping page, select CRASH_1 – Kickout.

10 Right-click CRASH_1 – Kickout and select Select from Bus Selector from the context menu.

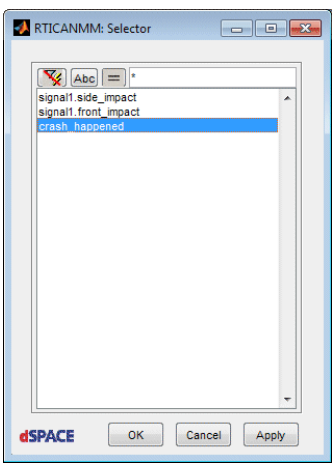


Note

If the error message The current block is not a bus selector occurs, you lost the focus on the Bus Selector. Click the Bus Selector to select it and repeat the steps 9 and 10.

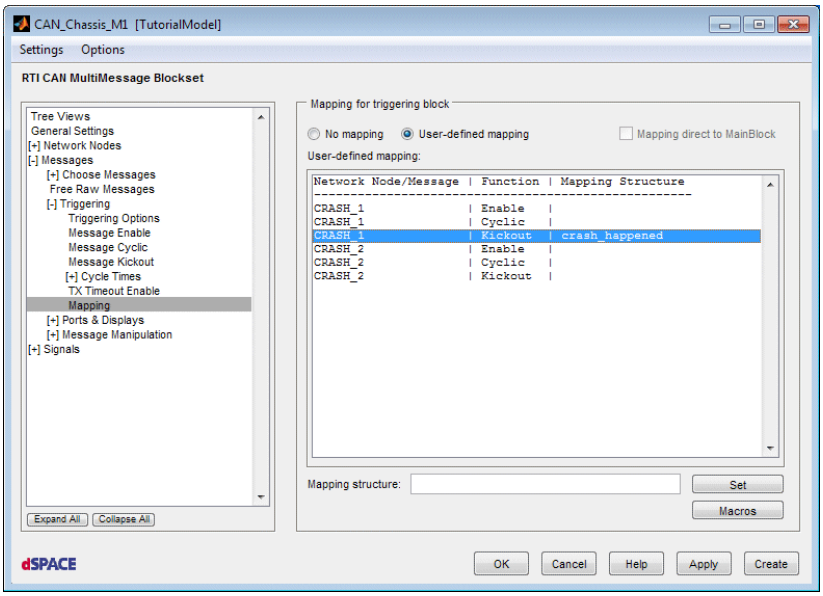
The RTICANMM:Selector dialog opens.

11 In the RTICANMM:Selector dialog, select crash_happened.



12 Click OK.

The RTICANMM:Selector dialog closes. The mapping structure crash_happened applies to CRASH_1 – Kickout.



13 In the User-defined mapping list, select CRASH_2 – Kickout.

14 Right-click CRASH_2 – Kickout and select Select from Bus Selector from the context menu.

15 In the RTICANMM:Selector dialog, select crash_happened and click OK.

The crash_happened mapping structure applies to CRASH_2 Kickout.

Interim result

You defined the CRASH_1 – Kickout and CRASH_2 – Kickout messages to be triggered by the crash_happened Constant block. You can now check the effects of these settings on your model. Continue with part 2.

Part 2

To check the effects of the specified settings on your model

1 In the CAN_Chassis_M1 [TutorialModel] dialog, click Create.

The CAN_Chassis_M1 MainBlock is created, and the specified settings apply to your model. During creation, the Overwrite? dialog opens twice. You are asked whether you want to overwrite the existing Mapping to RTICANMM and Triggering to RTICANMM blocks or add new ones to your model.

2 In both **Overwrite?** dialogs, click **Overwrite**.

The dialogs are closed and the settings of the existing Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten.

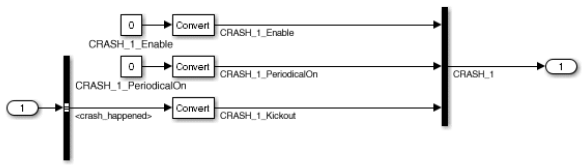
> **Tip**
>
> Unless you change the **Mapping to RTICANMM** and/or **Triggering to RTICANMM** block manually, you can always select **Overwrite** in the **Overwrite?** dialog. To save time, you can also select the checkbox **Do not ask again**.

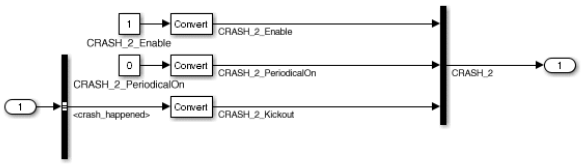
3 In the **CAN_Chassis_M1 [TutorialModel]** dialog, click **OK** to close the dialog.

In your model, an inport is added to your **Triggering to RTICANMM** block.

4 Open the **Triggering to RTICANMM** block and its subsystems.

In the **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_1** and **TutorialModel/Triggering2CAN_Chassis_M1/CRASH_2** subsystems, the inport of the Triggering to RTICANMM block is connected to **CRASH_1_Kickout** and **CRASH_2_Kickout** via **crash_happened**.





5 Close all the subsystems.

6 Save your model.

Result

You selected CRASH_1 and CRASH_2 to be triggered via kickout by the crash_happened Constant block. During creation of the CAN_Chassis_M1 MainBlock, an inport was added to the Triggering to RTICANMM block and its subsystems were updated according to the mapping structure you defined in the MainBlock.

What's next

In the next step, you can select signals to be manipulated by the Simulink bus and let an inport be created for the Mapping to RTICANMM block.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117
Overview of Lesson 5..... 118

Step 3: How to Create an Inport for the Mapping to RTICANMM Block

Objective

You can let inports be created by selecting TX model signals for the Mapping to RTICANMM block. These TX model signals will be manipulated by the bus.

In this step, you will select Event_A and Event_B signals to be manipulated by the side_impact and front_impact Constant blocks.

Part 1

To create an inport for the Mapping to RTICANMM block

1. Open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Signals – TX – Signal Mappings.

   On the Signal Mappings page, you can specify a mapping structure for your TX model signals.
3. On the Signal Mappings page, select User-defined mapping.

   The User-defined mappings list is enabled.
4. In your model, select the Bus Selector.

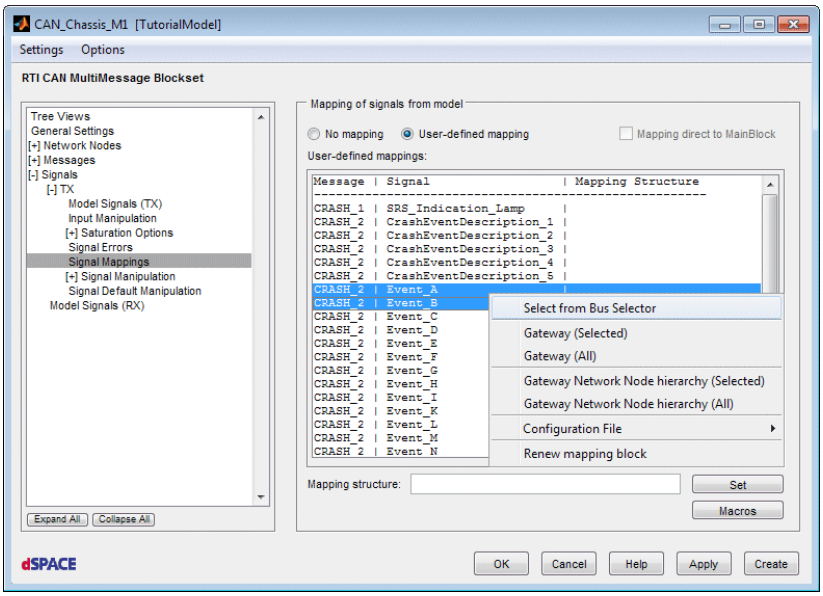
   Note

   To select the Bus Selector, you have to click it directly. Do not select the Bus Selector via a selection frame.
5. In the User-defined mappings list of the Signal Mappings page, select CRASH_2 – Event_A and CRASH_2 – Event_B.

   Tip

   You can use **`Shift`** and **`Ctrl`** for multiselection.

6 Right-click CRASH_2 – Event_A and CRASH_2 – Event_B and select Select from Bus Selector from the context menu.

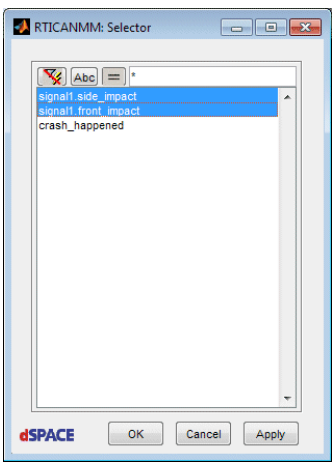


Note

If the error message The current block is not a bus selector occurs, you lost the focus on the Bus Selector. Click the Bus Selector to select it and repeat the steps 5 and 6.

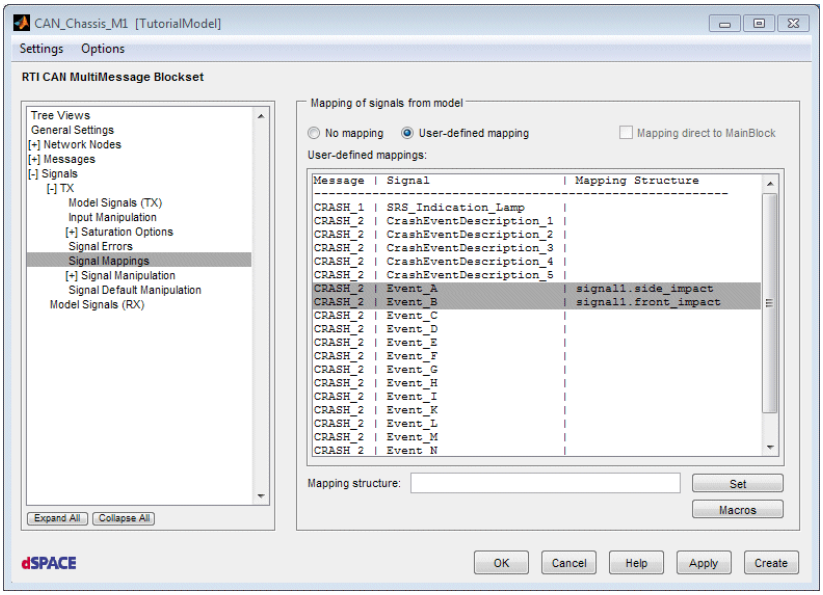
The RTICANMM:Selector dialog opens.

7 In the RTICANMM:Selector dialog, select signal1.side_impact and signal1.front_impact.



8 In the RTICANMM:Selector dialog, click OK.

The RTICANMM:Selector dialog closes. The mapping structure signal1.side_impact applies to CRASH_2 – Event_A and the mapping structure signal1.front_impact applies to CRASH_2 – Event_B.



Interim result

You defined the CRASH_2 – Event_A and CRASH_2 – Event_B signals to be manipulated by the side_impact and front_impact Constant blocks. You can now check the effects of these settings on your model. Continue with part 2.

Part 2

To check the effects of the specified settings on your model

1. In the CAN_Chassis_M1 [TutorialModel] dialog, click Create.

   The CAN_Chassis_M1 MainBlock is created, and the specified settings apply to your model. During creation, the Overwrite? dialog opens twice. You are asked whether you want to overwrite the existing Mapping to RTICANMM and Triggering to RTICANMM blocks or add new ones to your model.

2. In both Overwrite? dialogs, click Overwrite.

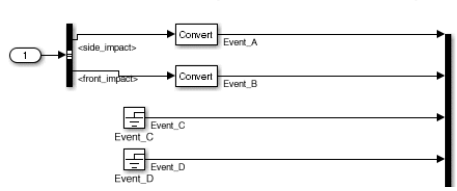
   The dialogs are closed and the settings of the existing Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten.

3. In the CAN_Chassis_M1 [TutorialModel] dialog, click OK to close the dialog.

   In your model, an inport is added to your Mapping to RTICANMM block.

4. Open the Mapping to RTICANMM block and its subsystems.

   In the TutorialModel/User2CAN_Chassis_M1/CRASH_2 subsystem, the inport of the Mapping to RTICANMM block is connected to Event_A and Event_B via side_impact and front_impact.



5 Close all the subsystems.

6 Save your model.

Result

You selected TX model signals to be manipulated by the Simulink bus. During creation of the CAN_Chassis_M1 MainBlock, an inport was added to the Mapping to RTICANMM block and its subsystems were updated according to the mapping structure you defined in the MainBlock.

What's next

In the next step, you can connect the Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117
Overview of Lesson 5..... 118

Step 4: How to Connect the Simulink Bus

Objective

After inports were created for the Triggering to RTICANMM and Mapping to RTICANMM blocks, the Simulink bus can be connected to them.

In this step, you will connect the Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks.

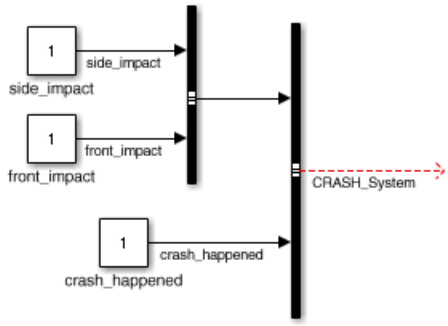
Method

To connect the Simulink bus to the Triggering to RTICANMM block and Mapping to RTICANMM block

1 In your model, delete the Bus Selector.

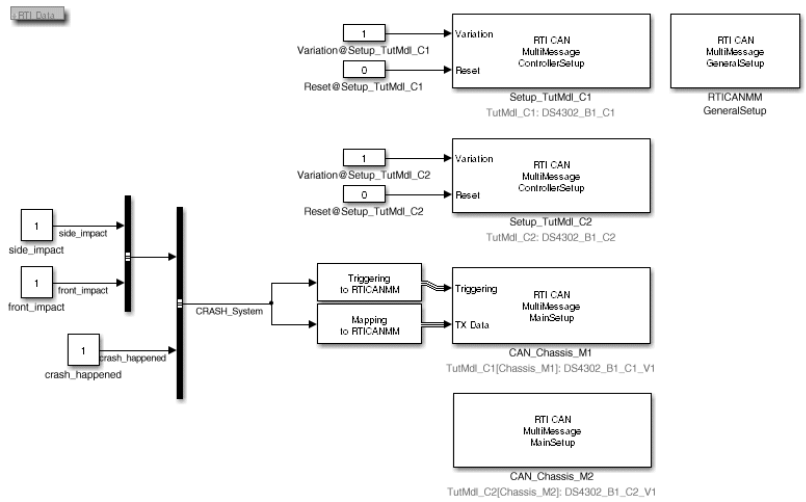
Note

Make sure that the CRASH_System connection and its label are not lost. If the label is lost, insert it again.



2 Connect CRASH_System to the Triggering to RTICANMM block.

3 Connect the Mapping to RTICANMM block to CRASH_System.



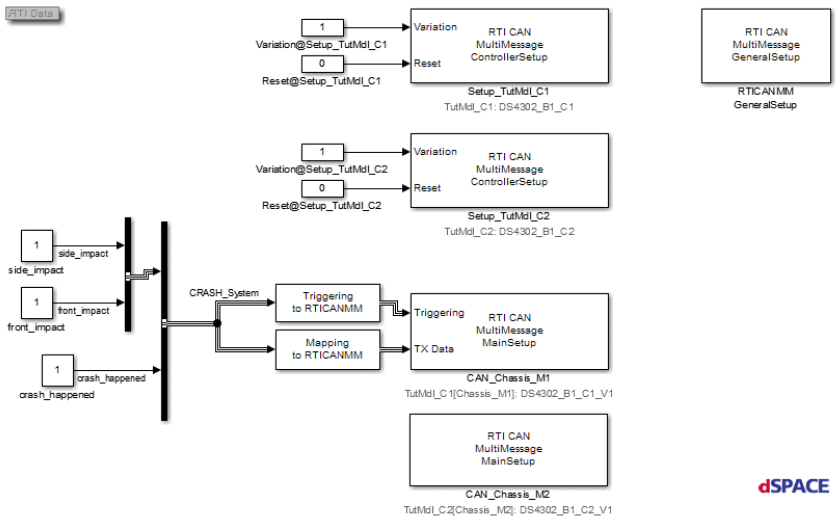
4 Save your model.

5 Build an application for your model.

Tip

Open the Triggering to RTICANMM and Mapping to RTICANMM blocks and their substructures to check the connections of the bus hierarchy.

Result



You connected the Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks and built an application for your model. To check the connections of the bus, you can open the Triggering to RTICANMM and Mapping to RTICANMM blocks and their substructures.

What's next

With this step, you finished adding a model-specific bus hierarchy to your model. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117
Overview of Lesson 5..... 118

Result of Lesson 5

Result

In this lesson, you learned the basic principles of working with model-specific bus hierarchies:

- You can use signals of your model, for example, to trigger messages and signals specified in an RTICANMM MainBlock. This is useful, for example, for simulating crash variants like front or side impacts.

- To connect signals of your model to an RTICANMM MainBlock, you can either customize the internal structure of a Triggering to RTICANMM and/or Mapping to RTICANMM block or use a Simulink bus.
- Using a Simulink bus prevents data inconsistencies, since the automatic update mechanisms during the create process of an RTICANMM MainBlock can be used.
- You can create a Simulink bus directly in your model or add a subsystem containing a bus to your model.
- You can connect a Simulink bus to an RTICANMM MainBlock via a Triggering to RTICANMM and/or Mapping to RTICANMM block.
- To create the necessary inports for the Triggering to RTICANMM and Mapping to RTICANMM block, you have to specify messages and TX model signals to be manipulated via the Simulink bus. You can connect a Simulink Bus Selector block to the bus for this purpose. You can define the mapping structure of the selected messages and TX model signals using this block. During creation of the MainBlock, inports are created and the subsystems of the Triggering to RTICANMM and Mapping to RTICANMM blocks are updated according to your specifications.
- Since the Simulink Bus Selector block is only needed for specifying mapping structures, you can delete it after you specified them. When you do so, you have to make sure to delete neither the connection between the Simulink bus and the Bus Selector nor the connection's label. You have to use this connection to connect the bus manually to the Triggering to RTICANMM and Mapping to RTICANMM blocks.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson5_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Mapping Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Signal Mappings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Message Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Message Cyclic Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Message Kickout Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Model Signals (TX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Triggering to RTICANMM Block (RTI CAN MultiMessage Blockset Reference)
- Mapping to RTICANMM Block (RTI CAN MultiMessage Blockset Reference)

Where to go from here

Now you can continue with the next lesson to learn the basic principles of performing a restbus simulation.

Related topics

Basics

Lesson 5: Working with Model-Specific Bus Hierarchies..... 117
Overview of Lesson 5..... 118

Lesson 6: Performing a CAN Restbus Simulation

Where to go from here

Information in this section

Overview of Lesson 6 136
In this lesson, you will learn learn the basic principles of performing a restbus simulation.

Step 1: How to Simulate a "Real" ECU 137
In this step, you will simulate a "real" ECU using one RTICANMM MainBlock.

Step 2: How to Simulate a Restbus 141
In this step, you will simulate the restbus using a separate RTICANMM MainBlock.

Step 3: How to Specify Counter Signals 145
In this step, you will specify counter signals to check the correct transmission of messages and to trigger the transmission of signals in a message.

Step 4: How to Specify Checksums 150
In this step, you will specify checksum algorithms and apply them to messages to check the transmission of TX messages.

Step 5: How to Check the Restbus Simulation in ControlDesk 161
In this step, you will check the representation of the "real" and the simulated ECUs in ControlDesk. Additionally, you will check the behavior of counter and checksum signals.

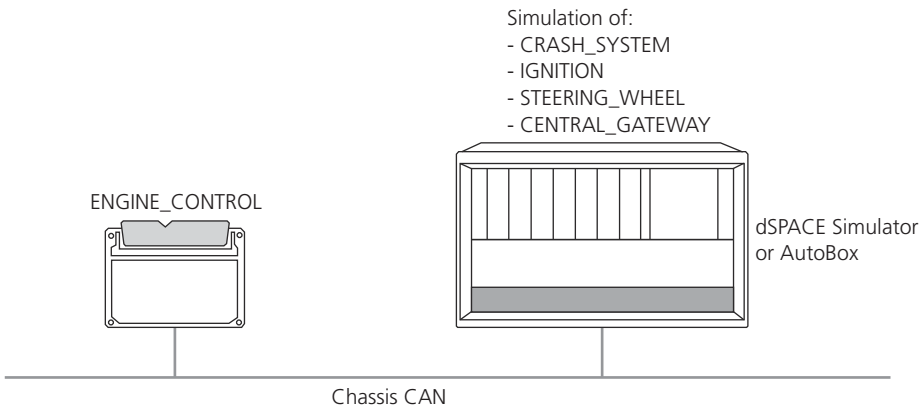
Result of Lesson 6 167
In this lesson, you learned the basic principles of performing a restbus simulation.

Overview of Lesson 6

Introduction

A typical use scenario for the RTI CAN MultiMessage Blockset is restbus simulation. In this, one or more real ECUs are connected via a bus to a simulator that simulates all the other ECUs of the bus. All the messages and signals of the simulated ECUs can be manipulated and the effects on the real ECU(s) observed via the simulator. To check the correct transmission of messages, checksum messages and counter signals can be defined for the simulated ECUs. Additionally, you can use counter signals to trigger the transmission of signals in a message.

Since you are working without physical ECUs in this tutorial, ENGINE_CONTROL will be simulated as a "real" ECU.



What you will learn

In this lesson, you will learn how to perform a typical restbus simulation.

- You will simulate a "real" ECU using one RTICANMM MainBlock.
- You will simulate all the other ECUs using a second RTICANMM MainBlock.
- You will define the communication between these two RTICANMM MainBlocks.
- You will define counter signals to check the correct transmission of messages and to trigger the transmission of signals.
- You will define checksum signals to check the correct transmission of messages.
- You will monitor the restbus simulation in ControlDesk.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 5.

Prepared demo

If you did not create your own model in lesson 5, you can use the prepared demo **Lesson5_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Simulate a "Real" ECU on page 137
- Step 2: How to Simulate a Restbus on page 141
- Step 3: How to Specify Counter Signals on page 145
- Step 4: How to Specify Checksums on page 150
- Step 5: How to Check the Restbus Simulation in ControlDesk on page 161

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 6 on page 167.

Duration

Working through this lesson will take you about 90 minutes.

What's next

Now you can start to simulate a restbus for the Chassis CAN bus, see Step 1: How to Simulate a "Real" ECU on page 137.

Related topics

Basics

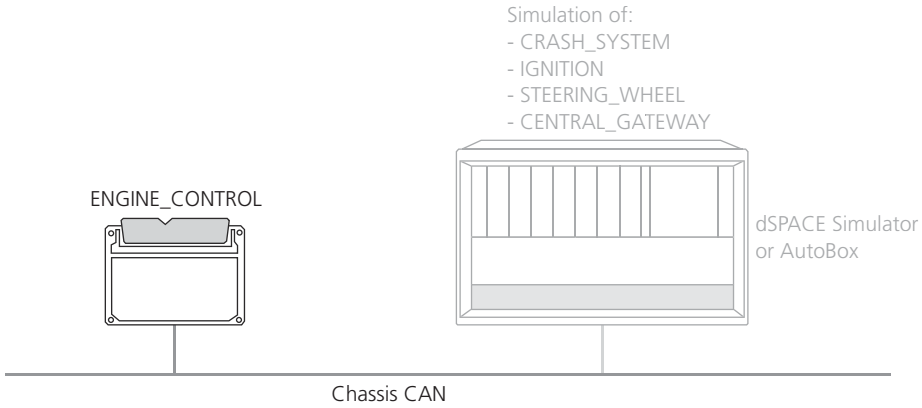
Lesson 6: Performing a CAN Restbus Simulation..... 135

Step 1: How to Simulate a "Real" ECU

Objective

For a restbus simulation, one or more real ECUs can be connected via a bus to a simulator which simulates all the other ECUs of the bus. Since you are working without physical ECUs in this tutorial, you will simulate a "real" ECU by a separate RTICANMM MainBlock.

In this step, you will simulate ENGINE_CONTROL as a "real" ECU.



Method

To simulate a real ECU

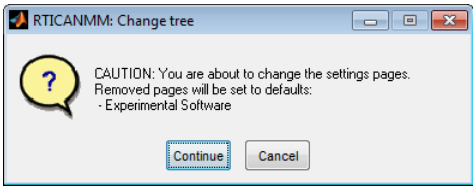
1 Open the CAN_Chassis_M2 MainBlock.

2 On the Tree Views page, select Filter Configuration Tree by Feature.

The predefined filters are enabled and the dialog tree is extended according to the selected filters.

3 Clear all filters except for Model Interface.

If a dialog similar to the following one is displayed, click Continue.



The dialog tree is reduced to the pages available for the Model Interface filter.

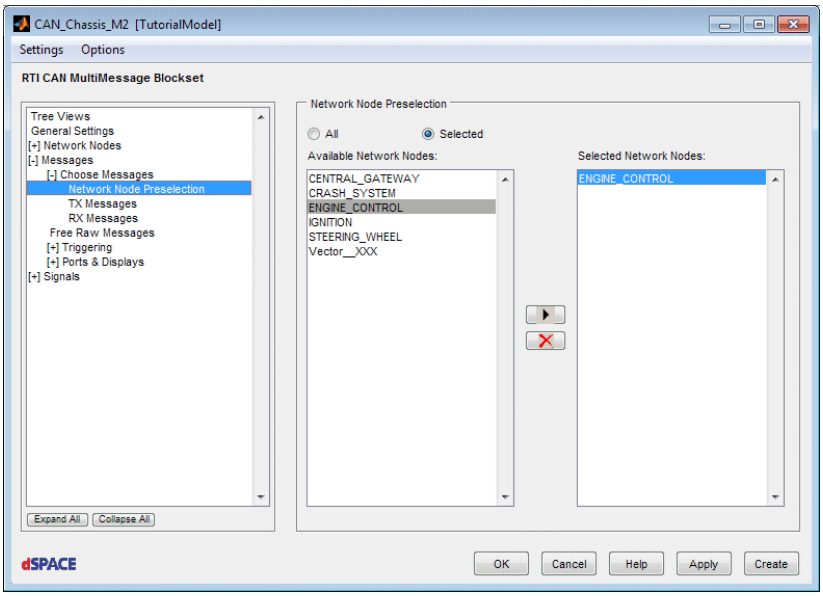
4 In the dialog tree, select Messages – Choose Messages – Network Node Preselection.

On the Network Node Preselection page, you can select ECUs. You can specify message settings for the selected ECUs on various dialog pages.

5 In the Available Network Nodes list, select ENGINE_CONTROL.

6 Click .

ENGINE_CONTROL is added to the **Selected Network Nodes** list.



7 In the dialog tree, select **Messages – Choose Messages – TX Messages**.

8 In the **Selected messages** list, select STEERING_1.

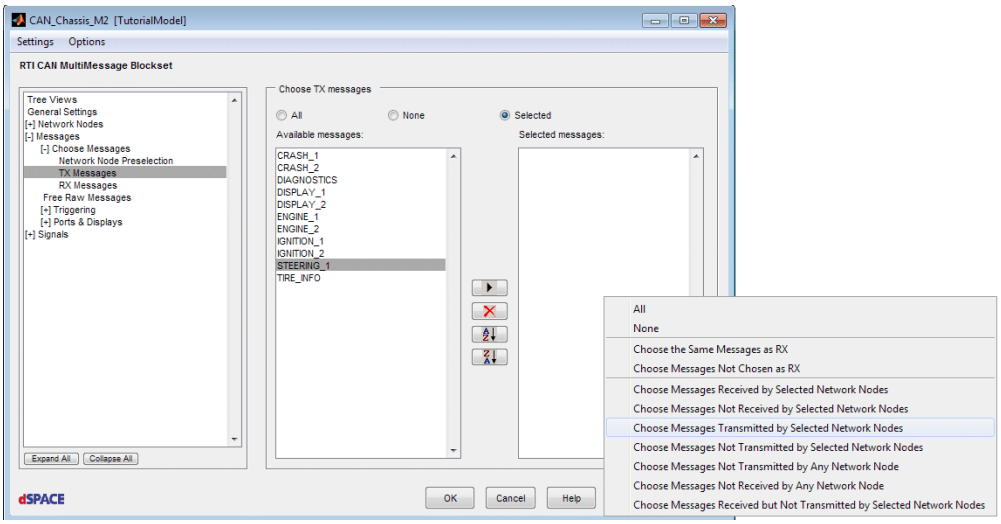
9 Click .

STEERING_1 is deleted from the **Selected messages** list. You can now select new messages.

Note

The **Selected messages** list is never overwritten. If you want to select new messages without adding them to already selected messages, you have to delete all the messages in the **Selected messages** list first.

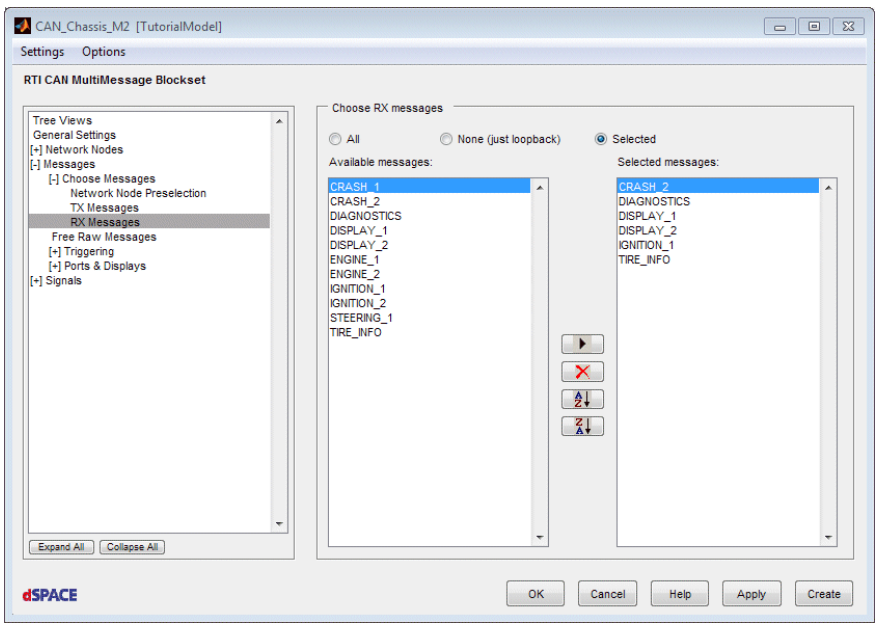
10 In the Selected messages list, right-click and select Choose Messages Transmitted by Selected Network Nodes from the context menu.



Since ENGINE_1 and ENGINE_2 are transmitted by ENGINE_CONTROL, they are added to the Selected messages list. Only these messages will be affected by the settings specified in the CAN_Chassis_M2 MainBlock.

11 Select the RX Messages page.

12 In the Selected messages list, right-click and select Choose Messages Received by Selected Network Nodes from the context menu.



The displayed messages are received by ENGINE_CONTROL.

13 Click Create.

The CAN_Chassis_M2 MainBlock is created, and the specified settings apply to your model.

14 Click OK to close the CAN_Chassis_M2 [TutorialModel] dialog.

Result

You specified the CAN_Chassis_M2 MainBlock as a "real" ECU. This MainBlock transmits and receives only messages that are transmitted and received by ENGINE_CONTROL. You selected ENGINE_CONTROL on the Network Node Preselection page. You also selected all the TX messages transmitted and all the RX messages received by this ECU.

What's next

In the next step, you can simulate all the other ECUs of the Chassis CAN bus.

Related topics

Basics

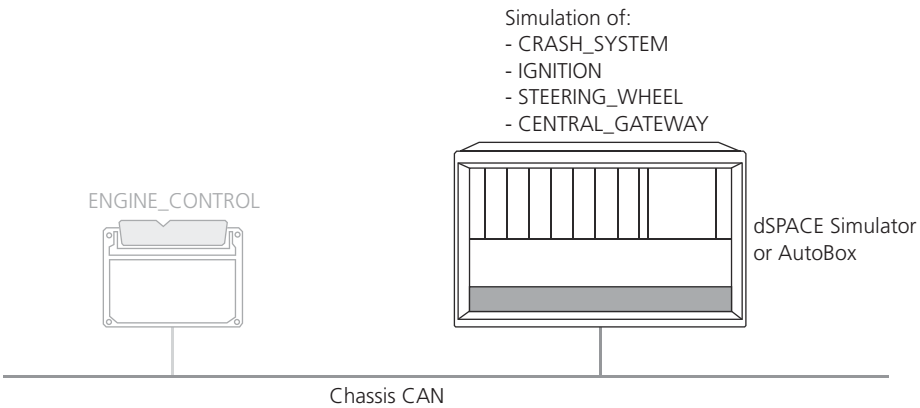
Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Step 2: How to Simulate a Restbus

Objective

To simulate a restbus, you can use a separate MainBlock and disable all the TX messages of the real ECU for it.

In this step, you will simulate all the ECUs of the Chassis CAN bus except for ENGINE_CONTROL. You will simulate the ECUs by using the CAN_Chassis_M1 MainBlock. You will disable all the TX messages of ENGINE_CONTROL for this block.



Method

To simulate the Chassis CAN bus

1. Open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Messages – Choose Messages – Network Node Preselection.

3 Specify ENGINE_CONTROL as selected network node.

4 In the dialog tree, select Messages – Choose Messages – TX Messages.

5 Select all the messages in the Selected messages list.

Tip

You can use **Shift** and **Ctrl** for multiselection.

6 Click .

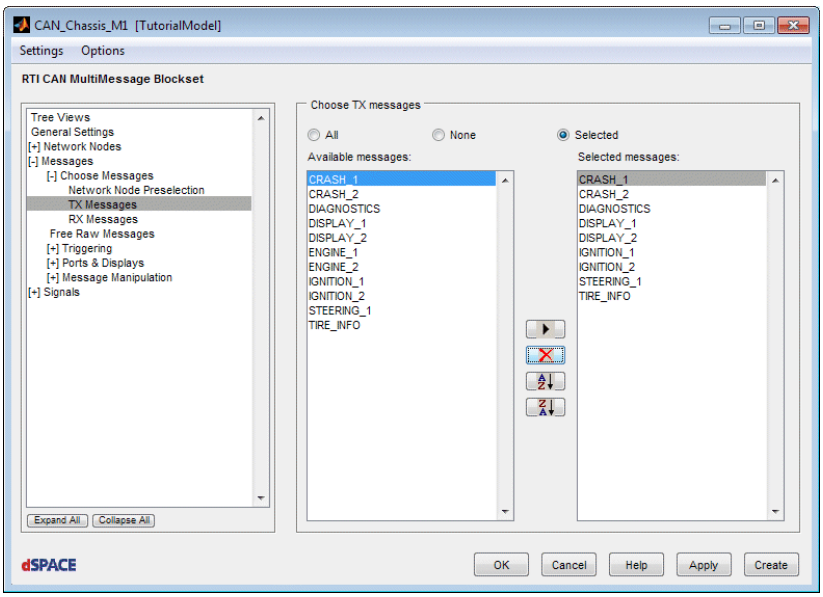
All the messages in the Selected messages list are deleted. You can now select new messages.

Note

The Selected messages list is never overwritten. If you want to select new messages without adding them to already selected messages, you have to delete all the messages in the Selected messages list first.

7 In the Selected messages list, right-click and select Choose Messages Not Transmitted by Selected Network Nodes.

All the messages except those transmitted by ENGINE_CONTROL are selected.



The selected messages will be affected by the settings specified in the CAN_Chassis_M1 MainBlock. The messages transmitted by ENGINE_CONTROL are not affected by the block's settings.

8 Select Network Nodes – Network Node Interfaces – Network Node Enable.

On the Network Node Enable page, you can specify whether you want to enable ECUs via your model and/or via ControlDesk. You can enable or disable ECUs by default and change the default settings during run time. If you disable an ECU, its TX messages are also disabled. You can specify these settings for all the ECUs displayed in the **Specify enables for all network nodes** list or just for some of them.

Tip

Disabling an ECU during run time allows you, for example, to exclude the ECU from restbus simulation when connecting it to the simulator as a physical ECU via the bus.

Since the TX messages of ENGINE_CONTROL are not affected by this block, you can disable ENGINE_CONTROL.

9 Clear the **Use these options for all** checkbox.

10 Specify **NONE** as the **Source** for ENGINE_CONTROL.

11 Click **Set**.

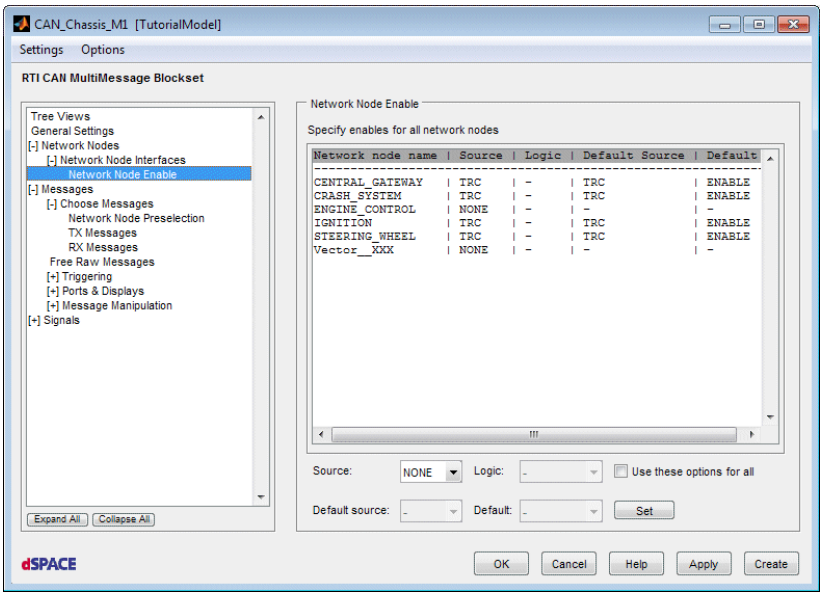
Note

You always have to click **Set** to apply your settings.

12 Specify the following settings for all the other ECUs:

Setting	Selection
Source	TRC
Default Source	TRC
Default	ENABLE

13 Click **Set**.



You disabled ENGINE_CONTROL. Since you selected **NONE** as the **Source**, you cannot change this setting during run time.

You enabled CENTRAL_GATEWAY, CRASH_SYSTEM, IGNITION and STEERING_WHEEL ECUs by default. This default value can be changed via ControlDesk during run time.

14 Click **Create**.

The **CAN_Chassis_M1** MainBlock is created, and the specified settings apply to your model. During creation, the **Overwrite?** dialog opens twice. You are asked whether you want to overwrite the existing **Mapping to RTICANMM** and **Triggering to RTICANMM** blocks or add new ones to your model.

15 In both **Overwrite?** dialogs, click **Overwrite**.

The dialogs are closed and the settings of the existing Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten.

Tip

Unless you change the **Mapping to RTICANMM** and/or **Triggering to RTICANMM** block manually, you can always select **Overwrite** in the **Overwrite?** dialog. To save time, you can also select the checkbox **Do not ask again**.

16 Click **OK** to close the **CAN_Chassis_M1 [TutorialModel]** dialog.

17 Save your model.

Result

You configured the CAN communication for the restbus simulation of the Chassis CAN bus. The CAN_Chassis_M1 MainBlock simulates all the ECUs of the Chassis CAN bus except ENGINE_CONTROL.

You selected ENGINE_CONTROL on the Network Node Preselection page. Since you selected all messages *not* transmitted by the selected network node on the TX Messages page, this block manipulates only TX messages not transmitted by ENGINE_CONTROL.

The ECUs of the Chassis CAN bus communicate with each other, so you did not have to specify RX messages. All the available RX messages can be used as RX messages.

All the ECUs simulated by this MainBlock can be enabled or disabled via ControlDesk during run time. This allows you, for example, to exclude an ECU from restbus simulation but to connect it to the simulator as a physical ECU via the bus.

What's next

In the next step, you can specify counter signals to check the correct transmission of messages and to trigger the transmission of signals.

Related topics

Basics

Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Step 3: How to Specify Counter Signals

Objective

The values of counter signals change with every message transmission. Counter signals can be used for two purposes: To check the correct transmission of a message or trigger the transmission of signals in a message.

- *Checking correct message transmission*: The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Using counter signals in this way is often called *alive counter*.

  Tip

  In lesson 8, you will learn more about counter signals and how to use them to check correct message transmission.

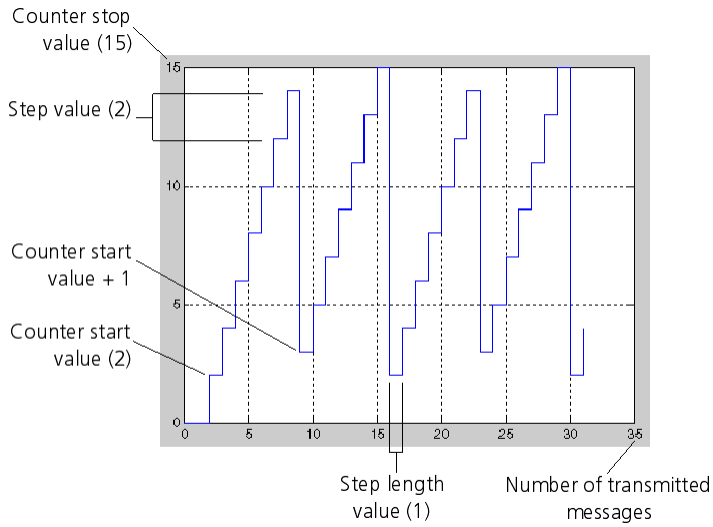
- *Triggering the transmission of signals*: CAN messages can consist of a mode signal and mode-dependent signals. The signal value of the mode signal triggers the transmission of the mode-dependent signals. For example, a message has a length of two bytes but consists of nine signals, one of which is a mode signal. The mode signal is transmitted via byte 1, and all the mode-dependent signals are transmitted via byte 2. In this case, you can define the mode signal as a counter signal. Since the counter signal value changes with every message transmission, this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can transmit all the signals even though the message has a length of only two bytes.

In this step, you will specify counter signals for both purposes.

Behavior of counter signals

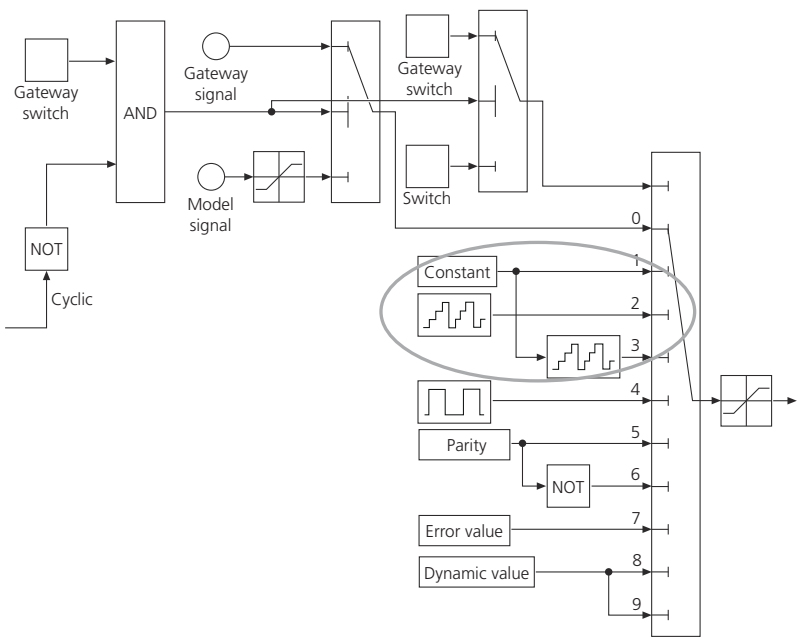
You can specify a start, stop, step, and step length value for a counter signal. After the counter reaches the stop value, it turns around to the counter start value.

Suppose you specify a counter start value of 2, a counter stop value of 15, and a step value of 2. The counter then turns around to (counter start value + 1) when it reaches 14 (first period). For the second period, the counter reaches the counter stop value and turns around to the counter start value. The cycle starts again with the first period.



Manipulation of counter signals during run time

During run time, you can switch between the counter signal, the signal's constant value, and an increment counter. In this, the counter step value is incremented by the signal's constant value. The illustration below visualizes the switch.



Part 1

To specify counter signals

1 Open the CAN_Chassis_M1 MainBlock.

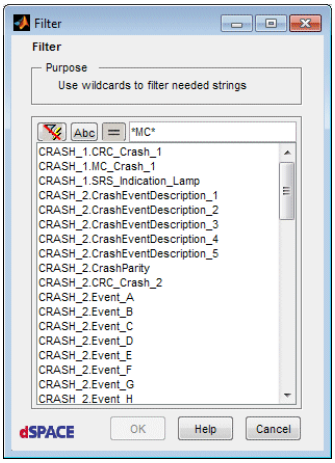
2 In the dialog tree, select Signals - TX - Signal Manipulation - Counter.

3 On the **Counter** page, click **Filter**.

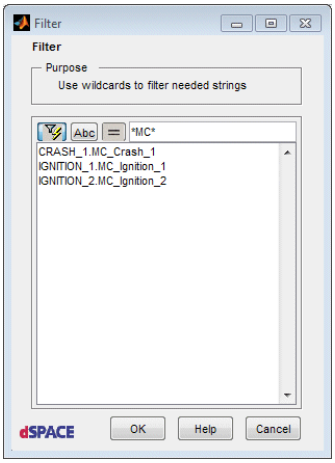
The **Filter** dialog opens for you to filter signals by their names.

> **Tip**
>
> In the chassis.dbc file, all the signals that are intended to be used as counter signals contain **MC** in their signal names, so you can filter for them via the Filter dialog.

4 In the edit field of the **Filter** dialog, type **`*MC*`**.



5 Click to activate the filter.



All the signals with MC in their signal names are displayed.

6 Select all the signals displayed in the Filter dialog.

7 Click **OK**.

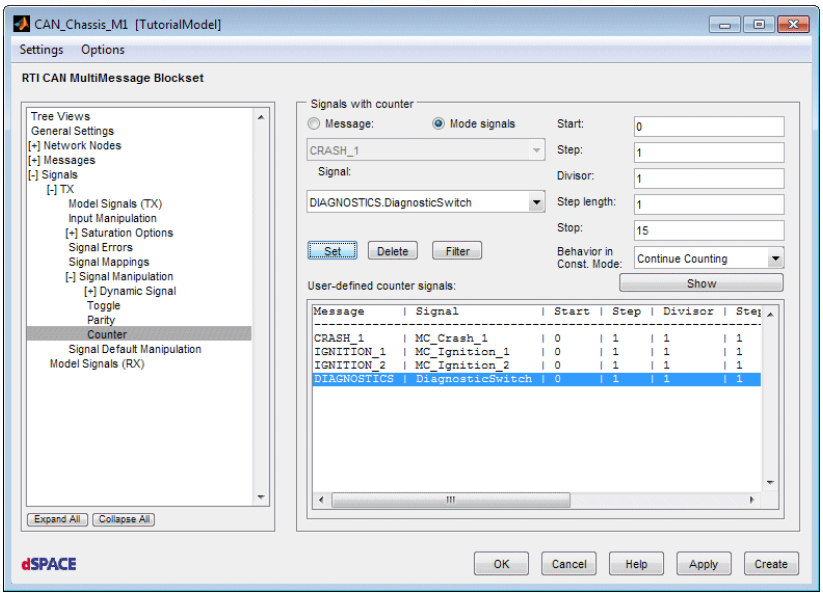
The Filter dialog is closed and the signals are added to the **User-defined counter signals** list on the Counter page.

8 On the **Counter** page, select **Mode signals**.

The message drop-down list is disabled and **DIAGNOSTICS.DiagnosticSwitch** is displayed in the **Signal** drop-down list.

9 Click Set.

DIAGNOSTICS.DiagnosticSwitch is added to the User-defined counter signals list.



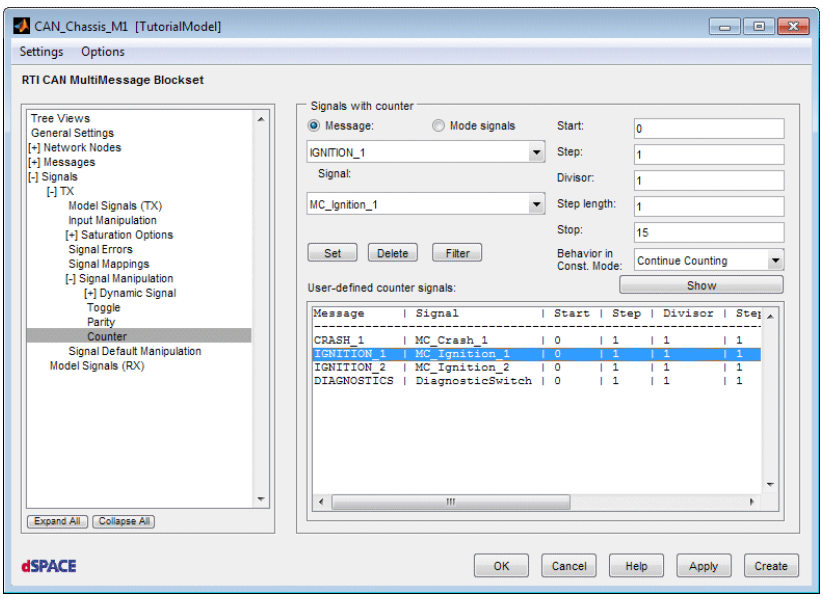
Interim result

You specified MC_Crash_1, MC_Ignition_1, MC_Ignition_2, and DiagnosticSwitch as counter signals. You can now change some of the default settings. Continue with part 2.

Part 2

To configure counter signals

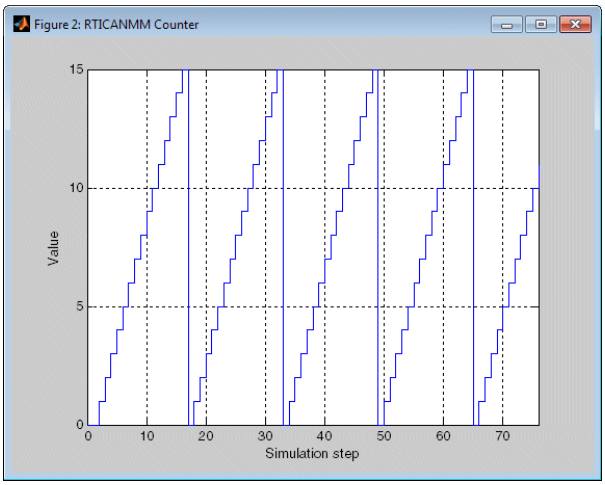
1 In the User-defined counter signals list, select IGNITION_1.



IGNITION_1 is displayed in the message drop-down list and MC_Ignition_1 is displayed in the signal drop-down list. The settings specified for this counter signal are displayed above the Show button.

2 Click Show.

The Figure2:RTICANMM Counter graph opens and displays the specified counter signal.



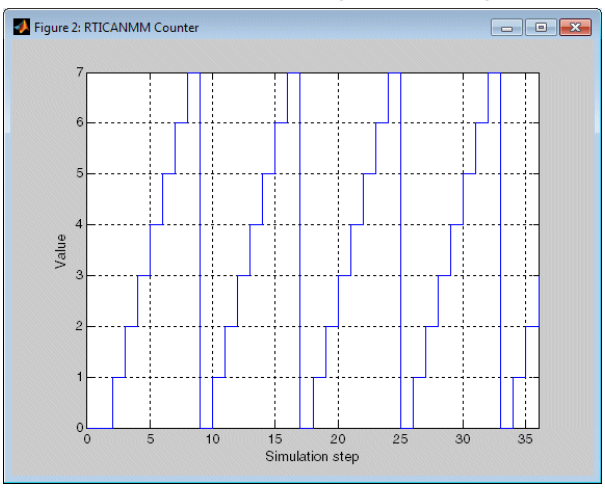
3 Close the graph.

4 In the Stop edit field, type **7**.

5 Click Set.

In the User-defined counter signals list, the stop value changes from 15 to 7.

6 Click Show to visualize the changed counter signal.



7 Close the graph.

8 In the User-defined counter signals list, select DIAGNOSTICS.

9 Specify a Start value of **1** and a Stop value of **8**.

10 Click **Set**.

In the **User-defined counter signals** list, the start and stop values change according to your specified values.

11 Click **Create**.

The **CAN_Chassis_M1** MainBlock is created. The **Overwrite?** dialog opens twice and asks you whether you want to overwrite the Triggering to RTICANMM and Mapping to RTICANMM block.

12 In the **Overwrite?** dialogs, click **Overwrite**.

The Triggering to RTICANMM and Mapping to RTICANMM blocks are overwritten, and the settings specified in the CAN_Chassis_M1 MainBlock apply to your model.

13 Click **OK** to close the **CAN_Chassis_M1 [TutorialModel]** dialog.

14 Save your model.

Result

You specified four counter signals and manipulated the default values of two counter signals.

What's next

In the next step, you can specify checksum algorithms and apply them to messages to check the transmission of TX messages.

Related topics

Basics

Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Step 4: How to Specify Checksums

Objective

To check the transmission of messages, you can specify checksum algorithms in a C-coded header file and apply them to messages. Checksum algorithms are calculations that ensure data integrity for the transmission and reception of a message. Checksums apply to TX and RX messages:

- *Checksum calculation for TX messages*: A checksum is calculated according to the applied algorithm, and assigned to the TX message. Then the message is transmitted together with the calculated checksum.
- *Checksum check for RX messages*: A checksum is calculated for the message and compared to the checksum in the received message. If they differ, an error occurs.

In this step, you will define checksum algorithms, specify the bytes used for the checksum calculation, and apply the checksum algorithms to TX messages.

Part 1

To create a checksum header file and specify checksum case identifiers

1 Open the CAN_Chassis_M1 MainBlock.

2 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

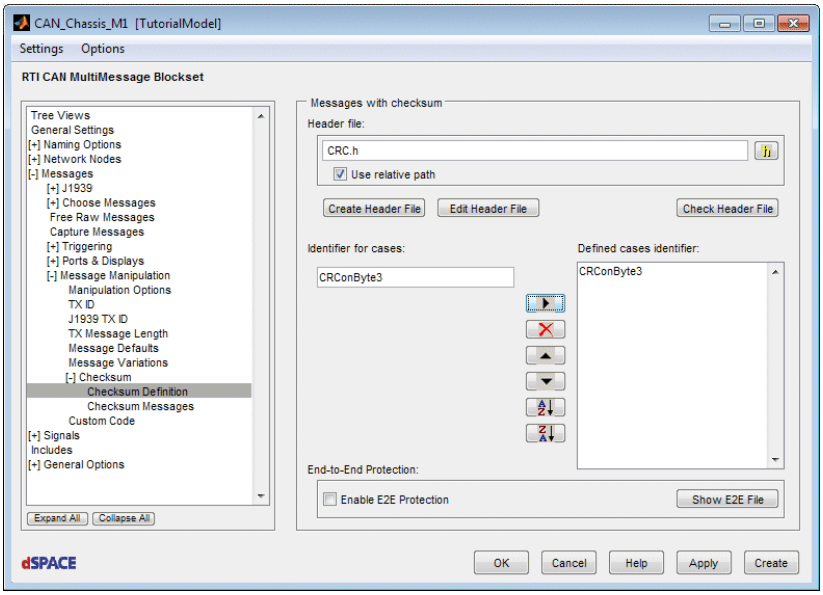
On the Checksum Definition page, you can define a checksum header file and specify checksum case identifiers.

3 In the Header file edit field, type **`CRC.h`**.

4 In the Identifier for cases edit field, overwrite CRCCASE and type **`CRConByte3`**.

5 Click .

CRConByte3 is added to the Defined cases identifier list.

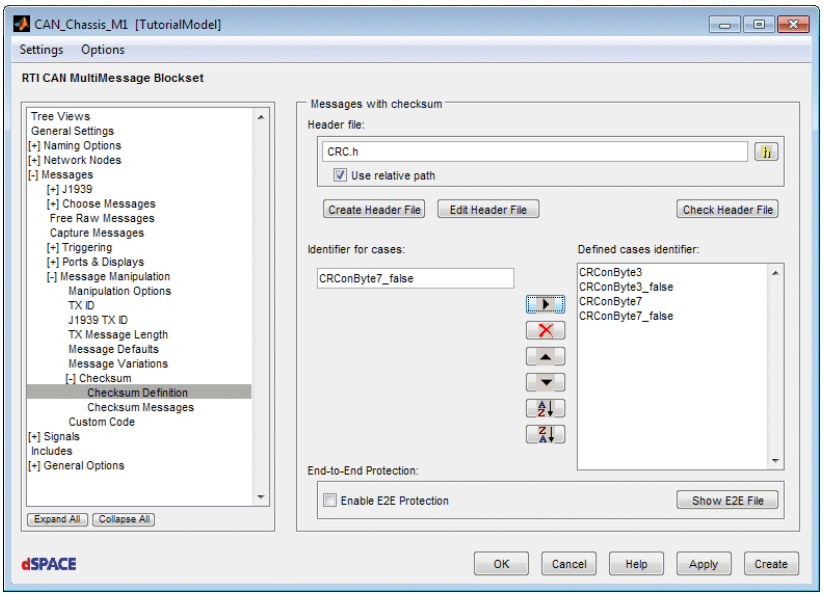


6 In the Identifier for cases edit field, type **`CRConByte3_false`**.

7 Click .

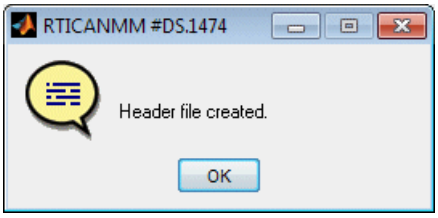
CRConByte3_false is added to the Defined cases identifier list.

8 Define two more case identifier named **`CRConByte7`** and **`CRConByte7_false`**.



9 Click Create Header File.

You are informed that the header file was created.



10 Click OK.

Interim result

You created a checksum header file and specified checksum case identifiers. You can now edit the header file and define checksum algorithms. Continue with part 2.

Part 2

To edit the checksum header file and define checksum algorithms

1 On the Checksum Definition page, click Edit Header File.

The `CRC.h` header file opens in an editor.

```
CRC.h
1  /*****************************************************************************************
2  /*   Automatic generated File from RTI CAN MultiMessage Blockset:
3  //      CRC.h
4  /* INPUT PARAMETER(S)
5  /*          (1) option : crc / crccheck   0: TX  (calculate)  1: RX (check)
6  /*          (2) Pointer to MsgData
7  /*          (3) Index of CRC see array in rticanmm_customer
8  /*          (4) StartBit of ChecksumSignal
9  /*          (5) Length of ChecksumSignal
10 /*
11 /* OUTPUT PARAMETER(S)
12 /*          (1) crccheck  (RX)
13 /*   DESCRIPTION:
14 /*     defines customer-specific CRCs
15 /*   AUTHOR(S):
16 /*     generated by dSPACE(R) RTICANMM
17 /* *****************************************************************************************
18
19 static UInt8 rticanmmcrc(int crcoption, RTICANMMMsgStruct* Msg, int crctype, int CsBitPos, int CsLe
20 {
21
22  UInt8 crc=0xFF;
23  int i=0;
24  int BytePos=0;
25
26  // Msg_Length = Msg->len;
27  // Id         = Msg->identifier;
28
29 switch(crctype) {
30   case 1:{     //CRConByte3
31
32    // Your CRC-Calculation here: Example:
33
```

The four checksum cases defined in this file are related to the case identifiers as follows:

Checksum Case	Case Identifier
Case 1	CRConByte3
Case 2	CRConByte3_false
Case 3	CRConByte7
Case 4	CRConByte7_false

2 In the header file, change the data type and the default value of the `crc` variable as follows:

```
UInt32 crc = 0;
```

3 Define a new variable named `CRCResult` as follows:

```
UInt8 CRCResult = 0;
```

The modified code now looks as follows:

```
static UInt8 rticanmmcrc(int crcoption,RTICANMMMsgStruct* Msg, int crctype,int CsBitPos,int CsLength)
{

 UInt32 crc = 0;
 UInt8  CRCResult = 0;
 int i=0;
 int BytePos=0;
```

4 Go to `// Id = Msg->identifier;` .

5 In a new line, insert a new `switch (crctype)` code block as follows:

> **Note**
>
> Do not modify the already existing `switch (crctype)` code block yet. You need that code block later on.

```
switch (crctype)
  {
   case 1:      // CRConByte3
   case 2:     // CRConByte3_false
           BytePos=3;
           break;
   case 3:     // CRConByte7
   case 4:    // CRConByte7_false
           BytePos=7;
           break;
  }
```

You defined byte 3 as the checksum byte for the checksum cases 1 and 2, and byte 7 as the checksum byte for the cases 3 and 4. The results of the checksum calculation are written to these bytes.

6 In a new line, insert the following code block:

```
/* CRC calculation */
 for (i = 0; i<Msg->len; i++)
  {
    if(i != BytePos) /* exclude crc byte */
      crc = 0xFF & (crc + Msg->RAW_DATA[i]);
  }
```

You defined the basic checksum calculation and the new code block now looks as follows:

```
// Id          = Msg->identifier;

switch (crctype)
  {
   case 1:      // CRConByte3
   case 2:     // CRConByte3_false
           BytePos=3;
           break;
   case 3:     // CRConByte7
   case 4:    // CRConByte7_false
           BytePos=7;
           break;
  }

/* CRC calculation */
 for (i = 0; i<Msg->len; i++)
 {
   if(i != BytePos) /* exclude crc byte */
     crc = 0xFF & (crc + Msg->RAW_DATA[i]);
 }
```

The checksum calculation defined in the for-loop applies to all checksum algorithms. The raw data of all the bytes of a message except for the checksum byte are summarized. For correct checksum calculation, the checksum byte is excluded. For example, CRConByte3 summarizes the raw

data of all the bytes of a message except for byte 3. To avoid overflow, you limited the byte values to 0xFF.

7 Go to `switch(crctype)` located below the code block you added.

```
/* CRC calculation */
 for (i = 0; i<Msg->len; i++)
 {
   if(i != BytePos) /* exclude crc byte */
     crc = 0xFF & (crc + Msg->RAW_DATA[i]);
 }

 switch(crctype) {
 case 1:{      //CRConByte3

  // Your CRC-Calculation here: Example:

   BytePos=7;

   crc=0xFF;


 }; break;
```

8 For `case 1`, delete `BytePos=7;`.

9 Change the calculation for the `crc` variable of this case as follows:

```
crc = 0xFF & crc;
```

10 For `case 2`, `case 3`, and `case 4`, delete `BytePos=7;`.

11 Change the calculation for the `crc` variable of the cases 2 to 4 as follows:

Checksum Case	CRC Calculation

Case 2	`crc = 0xFF & (crc + 1);`
	`if(crc > 254)`
	`{`
	`  crc = 0;`
	`}`

Case 3	`crc = 0xFF & crc;`

Case 4	`crc = 0xFF & (crc + 1);`
	`if(crc > 254)`
	`{`
	`  crc = 0;`
	`}`

You changed the calculation of the four checksum algorithms. The source code for the four modified cases is now:

```
switch(crctype) {
  case 1:{      //CRConByte3
    // Your CRC-Calculation here: Example:
   crc = 0xFF & crc;
  }; break;


  case 2:{      //CRConByte3_false
    // Your CRC-Calculation here: Example:
    crc = 0xFF & (crc + 1);
    if(crc >  254)
     {
      crc = 0;
     }
  }; break;


  case 3:{      //CRConByte7
     // Your CRC-Calculation here: Example:
    crc = 0xFF & crc;
  }; break;


  case 4:{      //CRConByte7_false
     // Your CRC-Calculation here: Example:
    crc = 0xFF & (crc + 1);
    if(crc >  254)
     {
      crc = 0;
     }
  }; break;
```

For CRConByte3 and CRConByte7 you used the checksum calculation defined in the for-loop. For CRConByte3_false and CRConByte7_false you modified the checksum calculation and incremented the calculated checksum value by 1.

Tip

You will use the CRConByte3_false and CRConByte7_false checksum algorithms in *Lesson 7: Customizing Messages* to simulate misbehavior of checksum calculation.

12 Go to the end of the `switch (crctype)` block.

13 In a new line, transfer the `crc` variable to the `CRCResult` variable as follows:

```
CRCResult = (UInt8)(crc);
```

```
    case 4:{      //CRConByte7_false
       // Your CRC-Calculation here: Example:
      crc = 0xFF & (crc + 1);
      if(crc >  254)
       {
        crc = 0;
       }
    }; break;

   default: {return(1);}
  }

  CRCResult = (UInt8)(crc);

   if(crcoption==0){
      // Return of CRC (TX)
      Msg->RAW_DATA[BytePos]=crc;
      return(0);
   }
```

14 In the `if (ccroption==0)` condition, replace `crc` by `CRCResult`.

```
  if(crcoption==0){
     // Return of CRC (TX)
     Msg->RAW_DATA[BytePos]=CRCResult;
     return(0);
  }
  else{
     // Return of CRC-Check (RX)
     return (Msg->RAW_DATA[BytePos]!=CRCResult);
  }
 }
```

The if-condition configures the different behavior of checksum algorithms for TX and RX messages.

15 Save and close the `CRC.h` file.

Interim result

You modified the checksum algorithms specified in the `CRC.h` header file. You can now assign the checksum algorithms to messages. Continue with part 3.

Part 3

To assign checksum algorithms to messages

1 Select the Checksum Messages page.

On the Checksum Messages page, you can assign checksum algorithms to messages.

2 Click Filter.

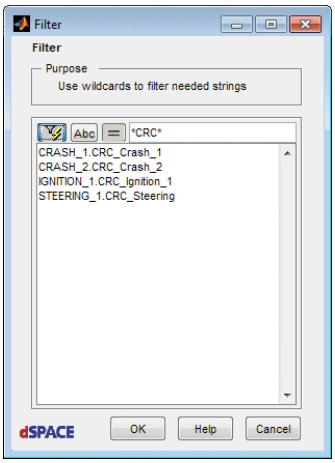
The Filter dialog opens for you to filter signals by their names.

Tip

In the `chassis.dbc` file, all the signals that are intended to be used as checksum signals contain CRC in their signal names, so you can filter for them via the Filter dialog.

3 In the edit field of the Filter dialog, enter ***CRC***.

4 Click .

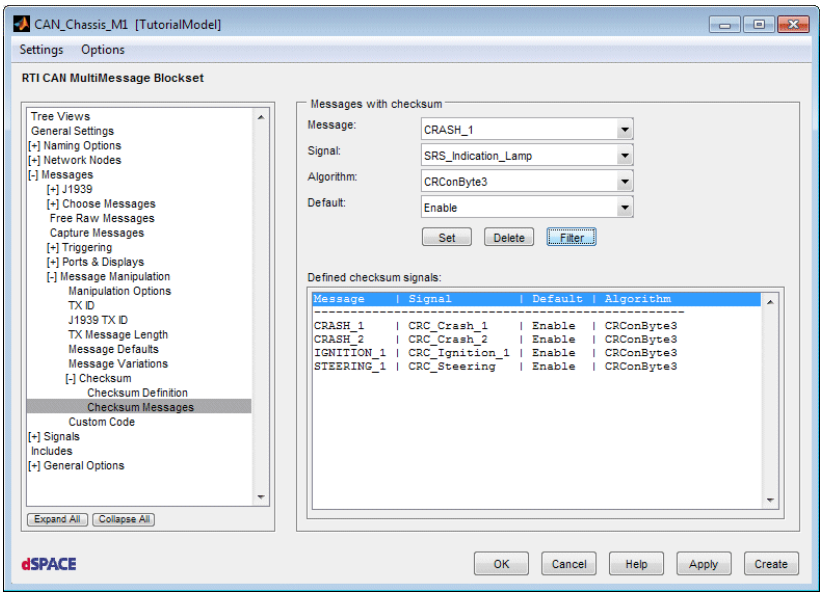


All the signals with CRC in their signal names are displayed.

5 Select all the signals displayed in the Filter dialog.

6 Click OK.

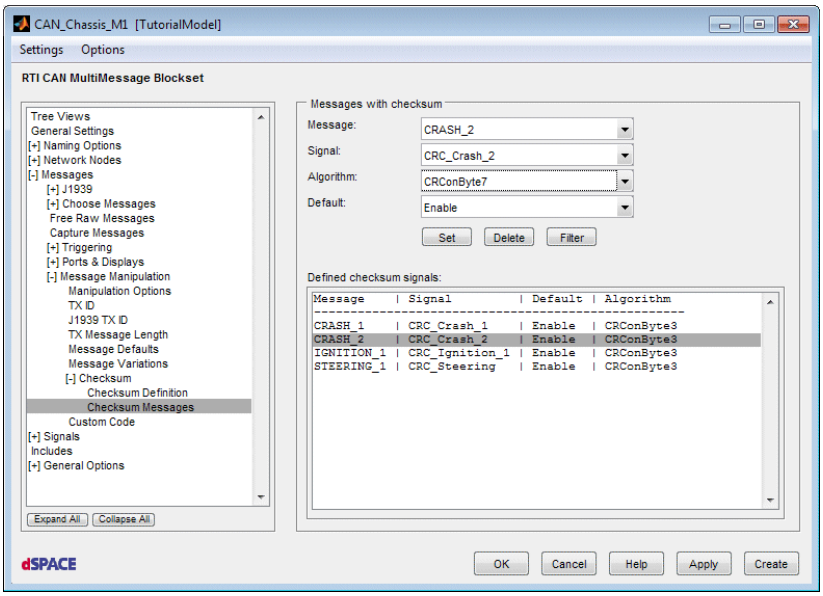
The Filter dialog is closed and the signals are added to the Defined checksum signals list.



7 In the Defined checksum signals list, select CRASH_2.

The settings specified for CRASH_2 are displayed in the drop-down lists.

8 From the Algorithm drop-down list, select CRConByte7.



9 Click Set.

CRConByte7 is specified as the checksum algorithm for CRASH_2.

10 Select IGNITION_1 and specify CRConByte7 as the checksum Algorithm.

11 Click Set.

CRConByte7 is specified as the checksum algorithm for IGNITION_1.

Interim result

You assigned checksum algorithms to four messages. For a better understanding of checksum behavior, you can now display the raw data of the messages' bytes. Continue with part 4.

Part 4

To display TX raw data in ControlDesk

1 Select the Tree Views page and select Instrumentation & Automation Interface.

2 In the dialog tree, select Messages – Ports & Displays – Raw Data – TX Raw Data Display.

On the TX Raw Data Display page, you can display the raw data of all the bytes of a TX message in ControlDesk. You can specify whether the raw data of all the available TX messages is displayed or just of a selection of messages.

3 Select All.

The raw data of all the TX messages will be displayed in ControlDesk. Since you selected all the messages via the All radio button, you do not have to make any manual selections from the lists.

4 Click **Create**.

The **CAN_Chassis_M1** MainBlock is created. The **Overwrite?** dialog opens twice and asks you whether you want to overwrite the Triggering to RTICANMM and Mapping to RTICANMM block.

5 In the **Overwrite?** dialogs, click **Overwrite**.

The Triggering to RTICANMM and Mapping to RTICANMM blocks are overwritten, and the settings specified in the CAN_Chassis_M1 MainBlock apply to your model.

6 Click **OK** to close the **CAN_Chassis_M1 [TutorialModel]** dialog.

7 Save your model.

Result

You created a checksum header file named `CRC.h` and specified the checksum algorithms CRConByte3, CRConByte3_false, CRConByte7, and CRConByte7_false.

CRC byte In the header file, you defined which byte is used for the checksum calculation:

- *CRConByte3*, *CRConByte3_false*: The calculated checksum is written to byte 3 of the message.
- *CRConByte7*, *CRConByte7_false*: The calculated checksum is written to byte 7 of the message.

CRC algorithm You defined a basic checksum calculation that applies to all checksum algorithms via a for-loop. The checksum calculation summarizes the raw data of all the bytes except for the checksum byte. For example, CRConByte3 summarizes the raw data of all the bytes of a message except for byte 3. To avoid overflow, you limited the byte values to 0xFF. To prepare the simulation of checksum misbehavior, you defined the CRConByte3_false and CRConByte7_false checksum algorithms. You modified the checksum calculation for both checksum algorithms and incremented the calculated checksum value by 1.

Tip

You will work with the CRConByte3_false and CRConByte7_false checksum algorithms in *Lesson 7: Customizing Messages* to simulate misbehavior of checksum calculation.

Messages with CRC algorithm You applied the checksum algorithms to the following messages:

Message	Signal	Checksum Algorithm
CRASH_1	CRC_CRASH_1	CRConByte3
CRASH_2	CRC_CRASH_2	CRConByte7
IGNITION_1	CRC_IGNITION_1	CRConByte7
STEERING_1	CRC_STEERING_1	CRConByte3

For a better understanding of checksum behavior, you activated the display of TX raw data. This helps you to identify which message bytes are in use by which signals.

What's next

In the next step, you can check the restbus simulation in ControlDesk and observe the effects of your specified settings on the message layouts.

Related topics

Basics

Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Step 5: How to Check the Restbus Simulation in ControlDesk

Objective

In ControlDesk, you can check the representation of the "real" and the simulated ECUs in the **Bus Navigator**. Additionally, you can check the behavior of counter and checksum signals.

In this step, you will check the representation of the "real" and the simulated ECUs. Additionally, you will check the behavior of counter and checksum signals.

Tip

You can simulate misbehavior of checksum and counter signals in lesson 7 and lesson 8 of this tutorial.

Part 1

To check the "real" and the simulated ECUs

1. Build an application for your model.
2. Create an experiment in the **RTICANMMTutorialProject** project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
3. From the context menu of the registered real-time hardware in the **Platforms/Devices** controlbar, select **Add to active Experiment** to add the platform to the new experiment.

   The **Define Platform Name** dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
4. Click **OK** to confirm the default platform name.

   The registered platform is added to the experiment.

5 From the platform's context menu in the Project controlbar, select Add Variable Description.

ControlDesk opens the Select Variable Description dialog.

6 In the Select Variable Description dialog, click Import from file and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

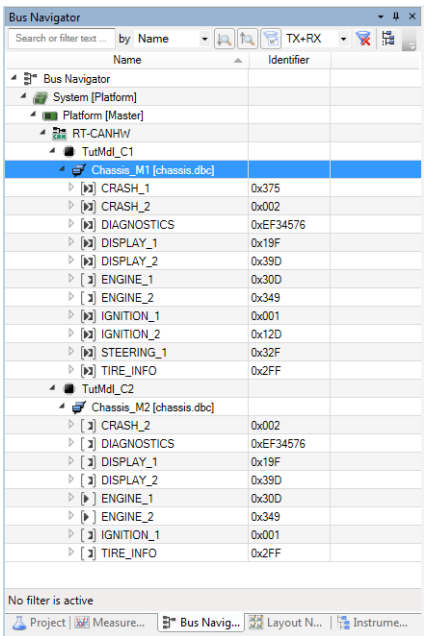
Note

To make sure that you add the latest version of the SDF file to the experiment, always use Import from file to import the file. If the Select Variable Description dialog displays the `TutorialModel.sdf` file and you select this file directly, you might load an obsolete version. This might cause problems while you work through this tutorial.

7 In the Select Variable Description dialog, click OK.

ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

8 Open the Bus Navigator.



Chassis_M2 [chassis.dbc]: All the messages below this node are messages of the "real" ECU. Since you are working without a physical ECU in this tutorial, ENGINE_CONTROL is simulated by the CAN_Chassis_M2 MainBlock to act like a real ECU.

Chassis_M1 [chassis.dbc]: All the messages below this node are messages of all the other ECUs specified in the chassis.dbc file. These ECUs are the simulated ECUs in this restbus simulation. Since ENGINE_1 and ENGINE_2 are transmitted by ENGINE_CONTROL, these messages have only RX status.

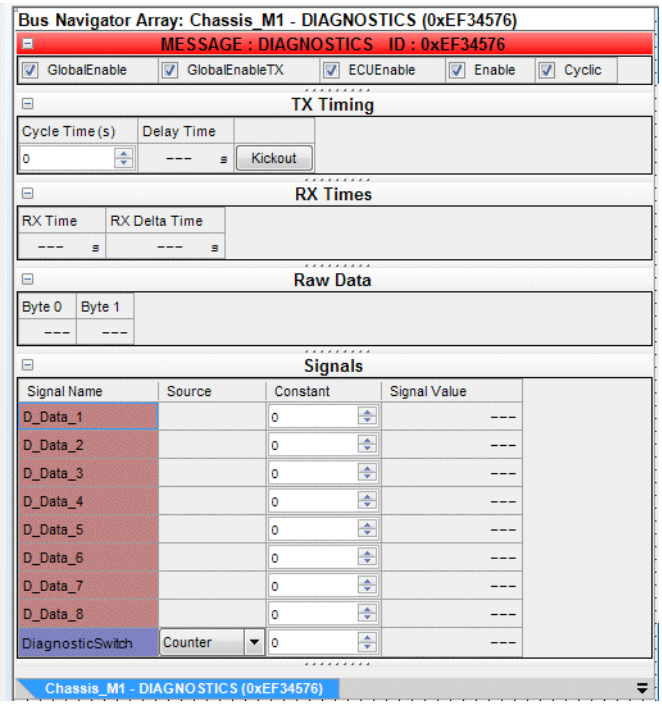
Interim result

You checked the presentation of the "real" ECU and the simulated ECUs. You can now check the behavior of counter signals. Continue with part 2.

Part 2

To check the behavior of counter signals

1 Right-click the DIAGNOSTICS message below the Chassis_M1 [chassis.dbc] node and open the TX layout.



DIAGNOSTIC is a message with only two bytes but nine signals. The DiagnosticSwitch signal acts as a mode signal: Its value determines the transmission of all the other signals. The signals are transmitted according to a mode value specified in the database file. In this message, the numbers in the signal names represent the mode values specified in the chassis.dbc file. For example, if DiagnosticSwitch has a signal value of 2, then D_Data_2 is transmitted. The colors of the signals indicate their status: The mode signal DiagnosticSwitch is violet, it is always transmitted. All the other signals are red because they are not transmitted yet.

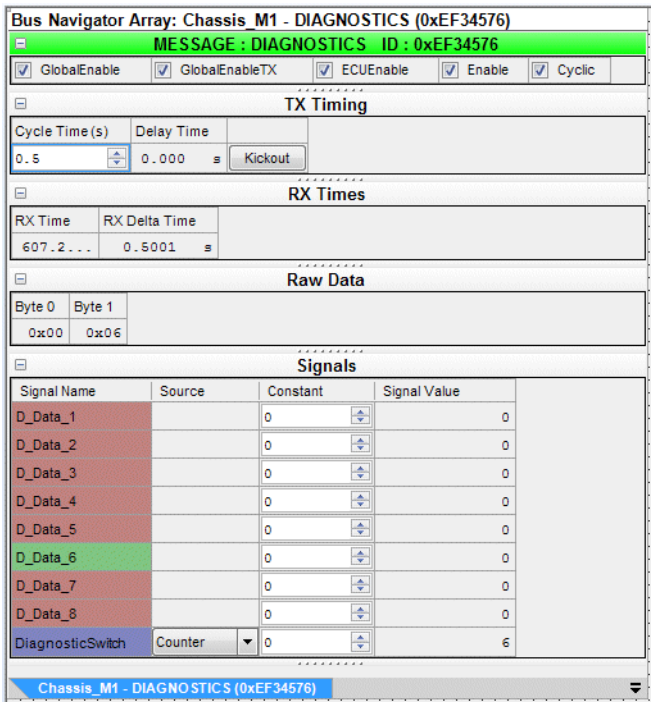
Tip

For an overview of the mode signals specified in the chassis.dbc file, click Show Check Report on the General Settings page of the CAN_Chassis_M1 MainBlock in your model. For the DIAGNOSTICS message, the mode signal, the mode-dependent signals, and the mode values of the mode-dependent signals are listed in a report (HTML file).

2 On the Home ribbon, click Status Control – Go Online to start online calibration.

The message is ready for transmission. Since no cycle time is specified, the message cannot be transmitted cyclically.

3 In the Cycle time field, type **0.500** and press **Enter**.



The message is transmitted. Since you specified DiagnosticSwitch as a counter signal, its value changes with every message transmission. According to the counter settings specified in your model, the DiagnosticSwitch value changes constantly from 1 to 8. Every time it changes, another signal is transmitted, while all the other signals are inactive.

4 From the Source drop-down list, select Increment Counter.

5 For the DiagnosticSwitch signal, specify a Constant value of **2**.

The counter value has an increment of 2. Since DiagnosticSwitch is a mode signal, every second signal is transmitted.

6 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

7 Close the layout.

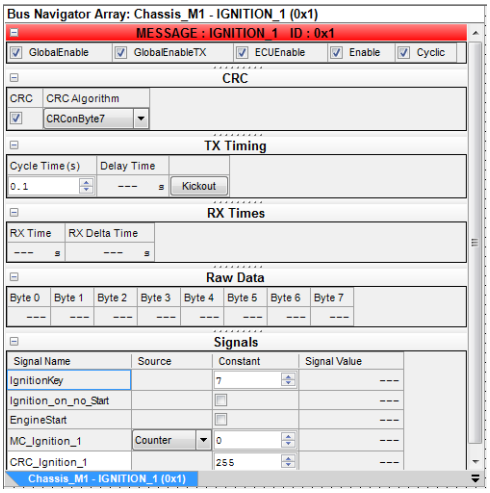
Interim result

You checked the behavior of the DiagnosticSwitch counter signal. Since this signal is also a mode signal, the counter value determines the transmission of all the other signals of DIAGNOSTICS. You can now check the behavior of checksums. Continue with part 3.

Part 3

To check the behavior of checksums

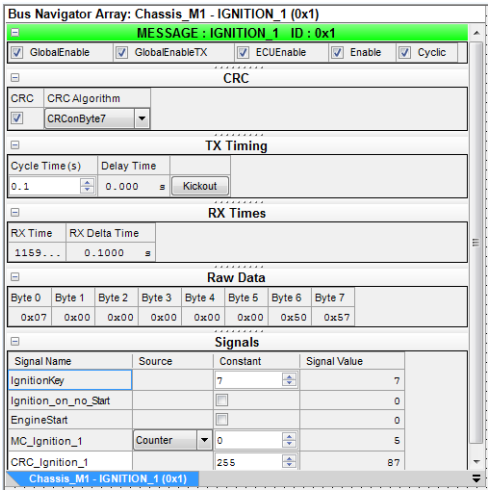
1 Right-click the IGNITION_1 message below the Chassis_M1 [chassis.dbc] node and open the TX layout.



The white color of the CRC_Ignition_1 signal indicates that a checksum algorithm has been specified for it.

2 On the Home ribbon, click Status Control – Go Online to start online calibration.

The message is transmitted and checksum calculation is enabled.



Since CRConByte7 is specified as the default checksum algorithm, the checksum algorithm summarizes the raw data values from byte 0 to byte 6 and transmits the calculated checksum value via byte 7. The signal value of CRC_Ignition_1 is the calculated checksum value in decimal format.

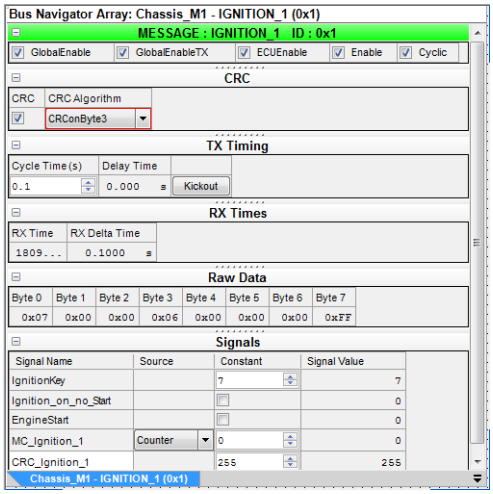
3 Clear CRC.

The checksum is disabled and CRC_Ignition_1 transmits its constant value via byte 7.

4 Select **CRC**.

The checksum is enabled and the checksum signal is transmitted.

5 From the **CRC Algorithm** drop-down list, select **CRConByte3**.



The selected checksum algorithm transmits a checksum signal via byte 3. Since the checksum signal of this message is defined to be transmitted via byte 7, the value transmitted via byte 3 is not recognized as a checksum signal. However, there is no active checksum algorithm which calculates a checksum signal for byte 7. For this reason, CRC_Ignition_1 transmits its constant value via byte 7.

In the checksum header file, you limited the checksum byte values to 0xFF to avoid an overflow. Because of the value of byte 7, rounding effects occur during checksum calculation and the calculated checksum signal value is 0x76.

Tip

For a better overview, specify a cycle time of 0.500, for example.

6 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

7 Save the project.

Result

You checked the effects of your specifications in ControlDesk.

- In the **Bus Navigator**, Chassis_M2 represents the "real" ENGINE_CONTROL ECU, and Chassis_M1 simulates all the other ECUs of the Chassis CAN bus.
- The DiagnosticSwitch counter signal changes its value constantly with every message transmission. DiagnosticSwitch is also a mode signal, so its value determines the transmission of all the other signals of the DIAGNOSTICS message.

- The CRC_Ignition_1 checksum signal transmits the checksum signal via byte 7 if **CRConByte7** is selected as the checksum algorithm. If **CRConByte3** is selected as the checksum algorithm or if checksum calculation is disabled, CRC_Ignition_1 transmits its constant value via byte 7.

What's next

With this step, you finished performing a restbus simulation. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Result of Lesson 6

Result

In this lesson, you learned the basic principles of performing a restbus simulation:

Creating a restbus Usually, you connect one or more real ECUs to a simulator via a bus. The simulator simulates all the other ECUs specified for the bus system. Since you are working without physical ECUs in this tutorial, you simulated ENGINE_CONTROL as a "real" ECU using the CAN_Chassis_M2 MainBlock. You simulated all the other ECUs of the Chassis CAN bus using the CAN_Chassis_M1 MainBlock. To check the correct transmission of messages, you specified counter signals and checksums.

Counter signals Counter signals change their signal value with every message transmission. They are defined by a counter start, step, step length, and counter stop value. Each time the counter stop value is reached, the counter turns around and starts again.

During run time, you can specify whether to transmit the constant, counter, or increment counter value of a signal. If the transmitted value does not match the value expected by the message receiver, an error occurs.

For the DIAGNOSTICS message, the counter signal is not only used to check the correct transmission of the message, but also to trigger the transmission of signals by one mode signal.

Checksums To specify checksums, you have to create a checksum header file and specify checksum algorithms. You can apply these algorithms to messages.

In this lesson, you specified four checksum algorithms in the CRC.h file: CRConByte3, CRConByte3_false, CRConByte7, and CRConByte7_false. According to your specifications, the CRConByte3 and CRConByte3_false checksum algorithms are transmitted via byte 3 of a message, the CRConByte7 and CRConByte7_false algorithms via byte 7.

Tip

The CRConByte3_false and CRConByte7_false checksum algorithms are used to simulate misbehavior of checksum calculations. You will work with these two checksum algorithms in Lesson 7: Customizing Messages on page 169.

In ControlDesk, you checked the behavior of the CRC_Ignition_1 checksum signal. Since this signal is transmitted via byte 7 of the message, it can only transmit the checksum value for the CRConByte7 and CRConByte7_false algorithms. If CRConByte3 or CRConByte3_false are selected or checksum calculation is disabled, CRC_Ignition_1 transmits its constant value. This can cause an error.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson6_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Checksum Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to customize messages, for example, to simulate message errors.

If you need to know how to work with checksum algorithms based on end-to-end communication (E2E protection) parameters, you can also continue with Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters on page 399.

Related topics

Basics

Lesson 6: Performing a CAN Restbus Simulation..... 135
Overview of Lesson 6..... 136

Lesson 7: Customizing Messages

Where to go from here

Information in this section

Overview of Lesson 7 169
In this lesson, you will learn the basic principles of customizing messages, for example, to simulate message errors.

Step 1: How to Enable ID and Length Manipulation of TX Messages 171
In this step, you will enable the manipulation of the message ID and message length during run time.

Step 2: How to Enable Dynamic Message Manipulation Options 174
In this step, you will enable dynamic message manipulation options. These options let you manipulate message settings for a defined number of message transmissions.

Step 3: How to Prepare the Detection of Checksum and Cycle Time Misbehavior 176
In this step, you will prepare the detection of checksum and cycle time misbehavior. You will also learn how to enable error display in ControlDesk.

Step 4: How to Work with Customized Messages in ControlDesk 181
In this step, you will create an additional layout to display error values. Then you will use customized messages to simulate message errors.

Result of Lesson 7 188
In this lesson, you learned the basic principles of customizing messages.

Overview of Lesson 7

Introduction

Using the RTICANMM blockset, you can customize messages, for example, to simulate message errors like faulty message IDs, message lengths, checksum

algorithms, or transmission states (message manipulation). You can customize messages in your model or enable manipulation options to customize messages during run time. Additionally, you can enable error display in ControlDesk. The displayed error values indicate whether or not a message is transmitted with its correct settings.

What you will learn

In this lesson, you will learn how to customize basic message settings.

- You will use customized messages to simulate signal errors.
- You will manipulate a message ID and length during run time.
- You will change the checksum algorithm of a message for a defined number of message transmissions.
- You will suppress the transmission of a message for a defined number of message transmissions.
- You will display the error values resulting from the customized settings in ControlDesk.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 6.

Prepared demo

If you did not create your own model in lesson 6, you can use the prepared demo **Lesson6_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.
- You have to be familiar with the basic principles of creating a new layout in ControlDesk.

Steps

This lesson contains the following steps:

- Step 1: How to Enable ID and Length Manipulation of TX Messages on page 171
- Step 2: How to Enable Dynamic Message Manipulation Options on page 174

- Step 3: How to Prepare the Detection of Checksum and Cycle Time Misbehavior on page 176
- Step 4: How to Work with Customized Messages in ControlDesk on page 181

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 7 on page 188.

Duration

Working through this lesson will take you about 60 minutes.

What's next

Now you can start to enable the manipulation of the message ID and length during run time, see Step 1: How to Enable ID and Length Manipulation of TX Messages on page 171.

Related topics

Basics

Lesson 7: Customizing Messages..... 169

Step 1: How to Enable ID and Length Manipulation of TX Messages

Objective

During run time, you can manipulate the ID and length of a message via your model and/or via ControlDesk.

In this step, you will enable the manipulation of the message ID and length during run time.

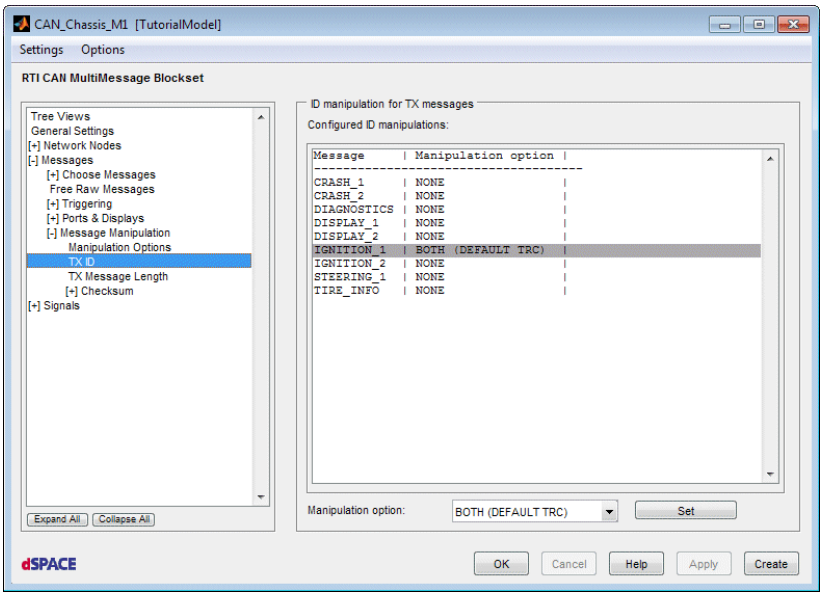
Part 1

To enable TX message ID manipulation

1. Open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Messages – Message Manipulation – TX ID page.

   On the TX ID page, you can enable ID manipulation and specify whether the ID should be manipulated via your model and/or via ControlDesk.
3. In the Configured ID manipulations list, select IGNITION_1.
4. From the Manipulation option drop-down list, select BOTH (DEFAULT TRC).

5 Click Set.



Interim result

For IGNITION_1, you enabled ID manipulation. Since you selected BOTH (DEFAULT TRC), the message ID can be manipulated via ControlDesk by default. You can change this default setting during run time. You can now enable length manipulation. Continue with part 2.

Part 2

To enable TX message length manipulation

1 Select the TX Message Length page.

On the TX Message Length page, you can enable TX message length manipulation. You can specify whether the message length should be manipulated via your model or via ControlDesk. You can also specify a default message length.

Tip

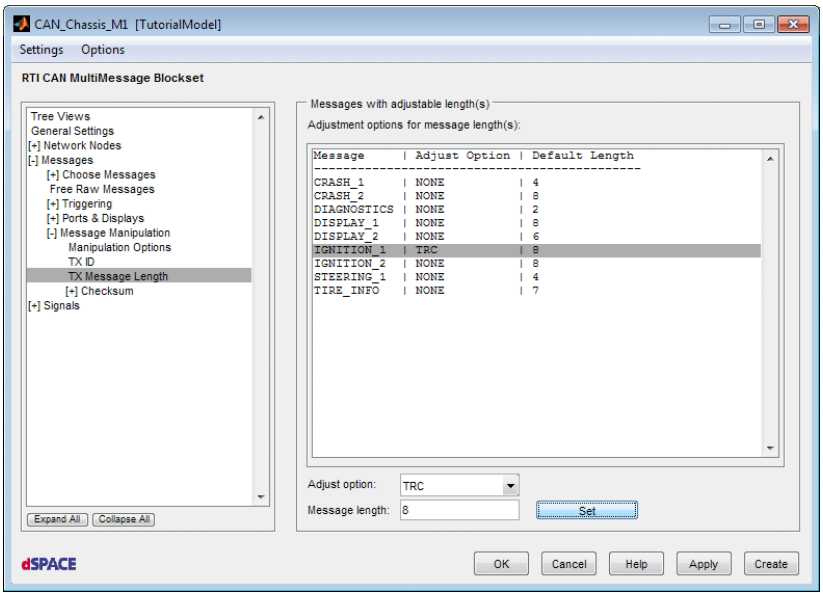
During run time, you can change the specified default message length in the full CAN message range (0 ... 8 byte) even if you specified a smaller length as the default.

2 In the Adjustment options for message length(s) list, select IGNITION_1.

3 From the Adjust option drop-down list, select TRC.

4 Click Set.

You enabled message length manipulation for IGNITION_1. The message length can be manipulated via ControlDesk.



Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Result

You enabled ID and length manipulation via ControlDesk for IGNITION_1. Since you selected BOTH (DEFAULT TRC) as the manipulation option for the message ID, you can manipulate the ID via ControlDesk by default. You can also change this default setting during run time and enable ID manipulation via your model. BOTH (DEFAULT TRC) is a switch, i. e., the message ID can be manipulated via ControlDesk *or* via the model inports.

What's next

In the next step, you can enable dynamic message manipulation options. These options let you manipulate message settings for a defined number of message transmissions.

Related topics

Basics

Lesson 7: Customizing Messages..... 169
Overview of Lesson 7..... 169

Step 2: How to Enable Dynamic Message Manipulation Options

Objective

Dynamic message manipulation options let you change message settings for a defined number of message transmissions during run time.

- *Dynamic checksum*: During run time, you can change the checksum algorithm of a message for a defined number of message transmissions. After that the previous checksum algorithm applies again.
- *Dynamic message length*: During run time, you can change the message length for a defined number of message transmissions. After that the previous message length applies again.
- *Timeout*: During run time, you can disable the transmission of a message for a defined number of possible message transmissions. The message is not transmitted even if transmission is triggered by the other trigger conditions.

In this step, you will enable these three dynamic message manipulation options.

Part 1

To activate dynamic message lengths and dynamic checksums

1. In the dialog tree of the CAN_Chassis_M1 [TutorialModel] dialog, select Messages – Message Manipulation – Manipulation Options.

   On the Manipulation Options page, you can enable dynamic message length and dynamic checksum.

2. Select Activate dynamic MessageLength and Activate dynamic CRC.

Interim result

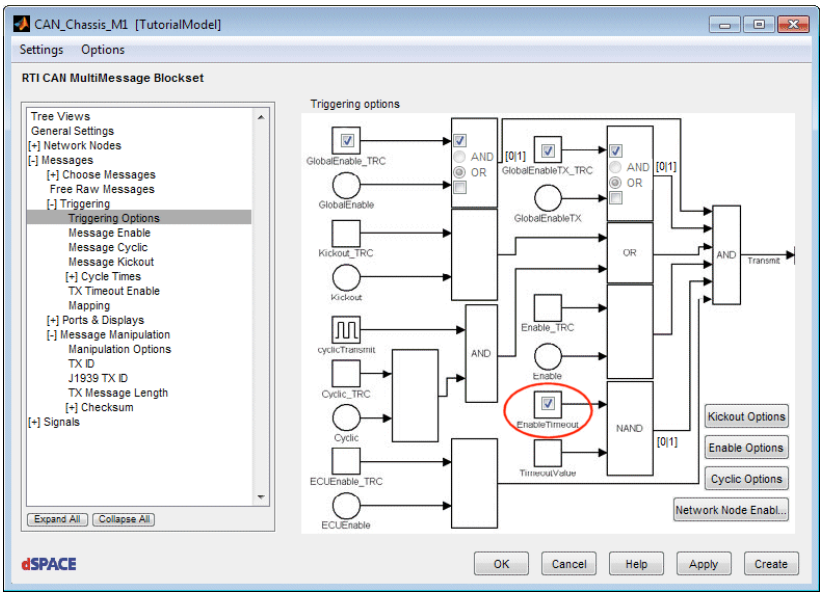
Dynamic message length and dynamic checksums are activated. You can now enable timeout. Continue with part 2.

Part 2

To enable timeout

1. In the dialog tree, select Messages – Triggering – Triggering Options.

2 In the Triggering options diagram, select EnableTimeout.



You enabled timeout for all the TX messages of the CAN_Chassis_M1 MainBlock.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Result

You enabled dynamic message length, dynamic checksum, and timeout for all the messages of your CAN_Chassis_M1 MainBlock.

What's next

In the next step, you can prepare the detection of checksum and cycle time misbehavior.

Related topics

Basics

Lesson 7: Customizing Messages..... 169
Overview of Lesson 7..... 169

Step 3: How to Prepare the Detection of Checksum and Cycle Time Misbehavior

Objective

To provoke misbehavior for checksum messages and/or cycle times, the receiver of a message must expect a certain checksum and/or cycle time value.

- *Checksums*: You can copy the checksum settings specified for the sending MainBlock to the receiving MainBlock. When you do this, the receiving MainBlock expects the default settings specified for checksums. If you change the default settings at the sender side during run time, an error will occur at the receiver side.
- *Cycle times*: Often, cycle time defaults are specified in the database file. You can use these defaults or specify different cycle time defaults. To provoke cycle time errors for received messages, you can define error values or specify a tolerance range for the default cycle times. In this case, an error will occur if the cycle time of a received message is out of the specified tolerance range.

To display message errors in ControlDesk, you have to enable error display.

In this step, you will learn how to prepare the detection of checksum and cycle time misbehavior. You will also learn how to enable error display.

Part 1

To check the settings for cycle time errors

1. In the dialog tree of the CAN_Chassis_M1 [TutorialModel] dialog, select the Tree Views page.
2. On the Tree Views page, select Extras.
3. In the dialog tree, select Messages – Triggering – Cycle Times – Cycle Time Defaults.

   The cycle times specified in the database file are the default as message cycle times. You do not have to change the default settings for this lesson.
4. In the dialog tree, select Messages – Ports & Displays – Errors – Cycle Time Error.

   The default error tolerance is 50 % of the cycle time. For example, the specified cycle time of a message is 100 ms. With an error tolerance of 50 %, an error will occur for cycle times lower than 50 ms or greater than 150 ms.

   You do not have to change the default settings for this lesson.

Interim result

You checked the default settings specified for the cycle times and the cycle time errors. You can now copy the settings specified for checksums in the CAN_Chassis_M1 MainBlock. Continue with part 2.

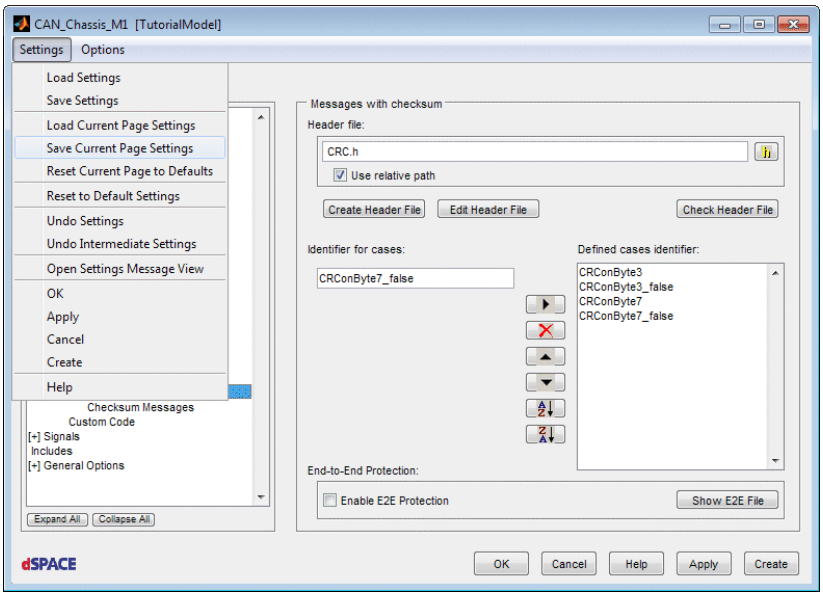
Part 2

To save settings of dialog pages

1 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

> **Note**
>
> The page settings need some time to be loaded. To prevent misbehavior, do not continue with the next step before the page is loaded completely.

2 From the menu bar of the CAN_Chassis_M1 [TutorialModel] dialog, select Settings – Save Current Page Settings.



The Settings MAT-file dialog opens.

3 Name the MAT file **`CRCDefinition`** and save it to your RTICANMMTutorial folder.

4 Select the Checksum Messages page.

5 Save the settings of the Checksum Messages page. Name the MAT file **`CRCMessages`**.

6 Click Create.

The CAN_Chassis_M1 MainBlock is created. The Overwrite? dialog opens twice and asks you whether you want to overwrite the Mapping to RTICANMM and Triggering to RTICANMM block.

7 In the Overwrite? dialogs, click Overwrite.

The Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten, and the settings specified in the CAN_Chassis_M1 MainBlock apply to your model.

8 Click OK to close the CAN_Chassis_M1 [TutorialModel] dialog.

9 Save your model.

Interim result

You saved the settings specified on the Checksum Definition and Checksum Messages pages of the CAN_Chassis_M1 MainBlock. You can now assign these settings to the CAN_Chassis_M2 MainBlock. Continue with part 3.

Part 3

To load settings of dialog pages

1 Open the CAN_Chassis_M2 MainBlock.

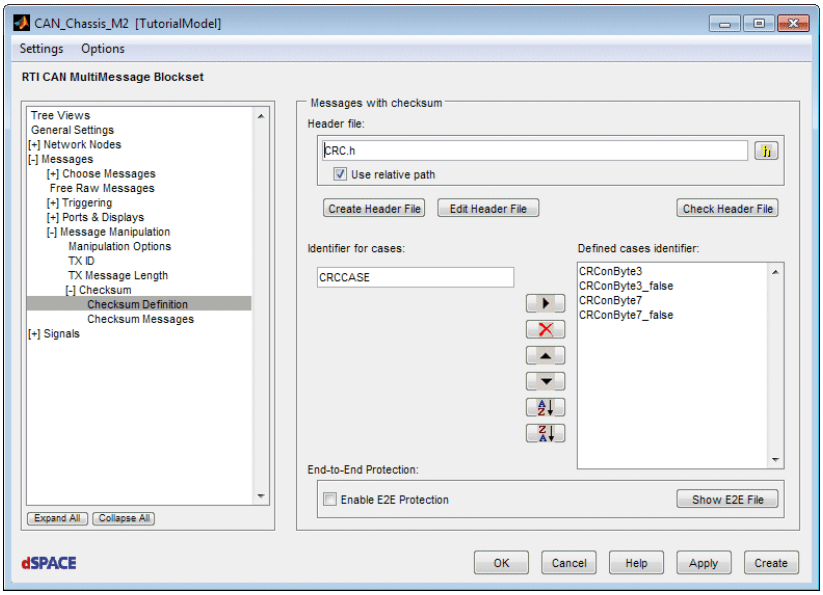
2 On the Tree Views page, select Manipulation Interface.

3 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

4 From the menu bar of the CAN_Chassis_M2 [TutorialModel] dialog, select Settings – Load Current Page Settings.

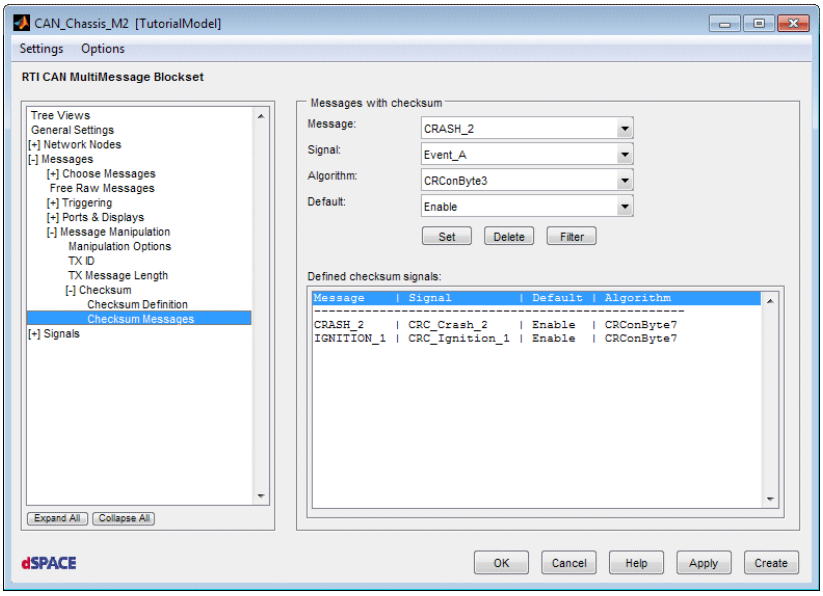
5 In the Settings MAT-file dialog, select CRCDefinition.mat and click Open.

The dialog closes and the settings specified in the CRCDefinition file are loaded to the Checksum Definition page.



6 Select the Checksum Messages page and load the settings specified in the CRCMessages.mat file to it.

This MainBlock receives and transmits only messages of ENGINE_CONTROL. Since CRASH_1 and STEERING_1 are not messages of ENGINE_CONTROL, their signals are not added to the list.



Interim result

You copied the settings of the Checksum Definition and Checksum Messages pages specified for the CAN_Chassis_M1 MainBlock to the CAN_Chassis_M2 MainBlock. You can now enable error display. Continue with part 4.

Part 4

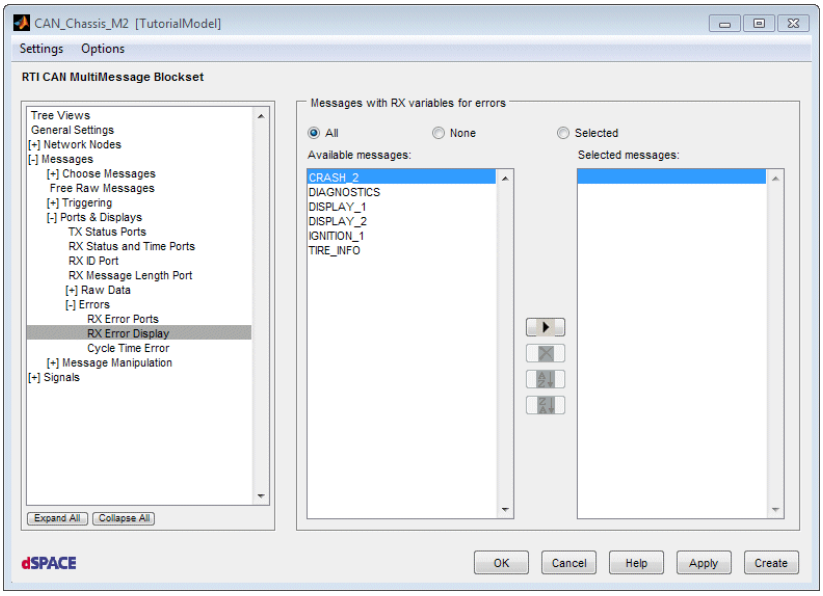
To enable error display

1 In the dialog tree, select the Messages – Ports & Displays – Errors – RX Error Display page.

   On the RX Error Display page, you can display message errors in ControlDesk. You can specify messages to display message errors for.

2 Select All.

   The Available messages list is enabled and displays all the RX messages of this MainBlock. Message errors for these messages can be displayed in ControlDesk. Since you selected all messages via the radio button, you do not have to make any manual selections in the lists.



3 Click **Create** to let the necessary files be created automatically and to apply the specified settings to the model.

4 Click **OK** to close the **CAN_Chassis_M2 [TutorialModel]** dialog.

5 Save your model.

6 Build an application for your model.

Result

You prepared the detection of checksum and cycle time misbehavior. Additionally, you enabled error display in ControlDesk and built an application for your model.

What's next

In the next step, you can work with customized messages in ControlDesk.

Related topics

Basics

Lesson 7: Customizing Messages..... 169
Overview of Lesson 7..... 169

Step 4: How to Work with Customized Messages in ControlDesk

Objective

To work with customized messages in ControlDesk, three layouts are usually needed:

- The message's TX layout of the sending CAN controller.
- The message's RX layout of the receiving CAN controller.
- An additional layout to display the error values.

If you change message settings via the TX layout or via the model, you can observe the effects in the RX layout and/or in the layout that displays error values.

In this step, you will create an additional layout to display error values. Then you will use customized messages to simulate message errors.

Part 1

To display error values

1. In ControlDesk, make sure that the tutorialmodel.ppc application is running on your real-time hardware.
2. Create an experiment in the **RTICANMMTutorialProject** project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
3. From the context menu of the registered real-time hardware in the **Platforms/Devices** controlbar, select **Add to active Experiment** to add the platform to the new experiment.

   The **Define Platform Name** dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
4. Click **OK** to confirm the default platform name.

   The registered platform is added to the experiment.
5. From the platform's context menu in the **Project** controlbar, select **Add Variable Description**.

   ControlDesk opens the **Select Variable Description** dialog.
6. In the **Select Variable Description** dialog, click **Import from file** and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

   Note

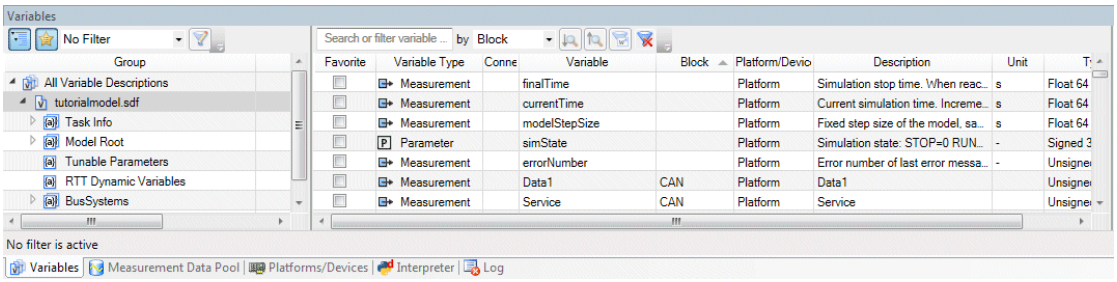
   To make sure that you add the latest version of the SDF file to the experiment, always use **Import from file** to import the file. If the **Select Variable Description** dialog displays the `TutorialModel.sdf` file and you select this file directly, you might load an obsolete version. This might cause problems while you work through this tutorial.
7. In the **Select Variable Description** dialog, click **OK**.

   ControlDesk closes the **Select Variable Description** dialog and adds the variable description (SDF file) to the platform.

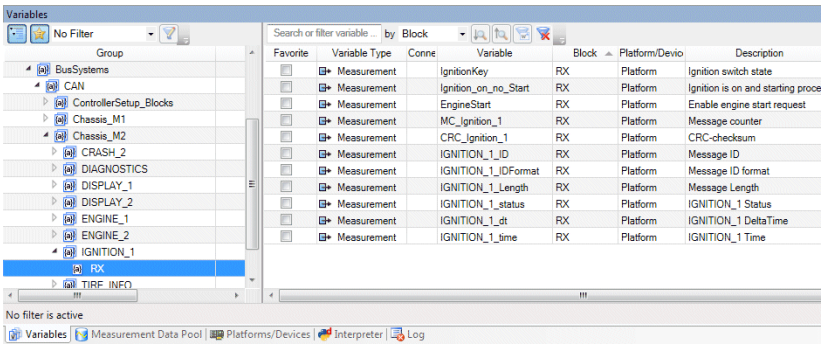
8 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

9 To rename the layout, select Rename from the layout's context menu in the Project controlbar and enter **errorplotter**. Click OK.

10 Select the Variables controlbar.



11 In the tree view, select tutorialmodel – BusSystems – CAN – Chassis_M2 – IGNITION_1 – RX.



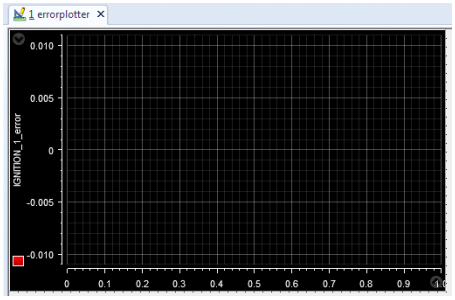
The window on the right displays all the available variables for the IGNITION_1 RX message.

12 From the variable list, drag the IGNITION_1_error variable onto an empty space in the errorplotter layout.

A drop-down list opens for you to select an instrument to assign to the variable.

13 From the list, select Time Plotter.

A Time Plotter is added to the errorplotter layout and the y-axis of the Time Plotter displays the name of the connected signal.



14 Save the project.

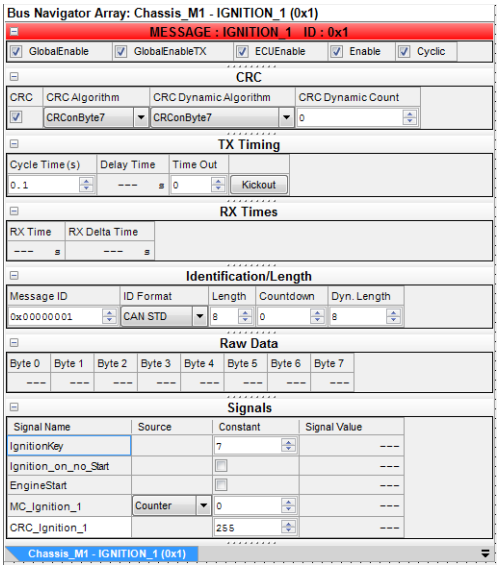
Interim result

You created a layout to display message errors of IGNITION_1. You can now open the message's RX and TX layouts, simulate message errors, and observe the effects on the RX and errorplotter layouts. Continue with part 2.

Part 2

To simulate message errors for IGNITION_1

1 In the Bus Navigator tree, right-click the IGNITION_1 message below the Chassis_M1 [chassis.dbc] node and open the TX layout.

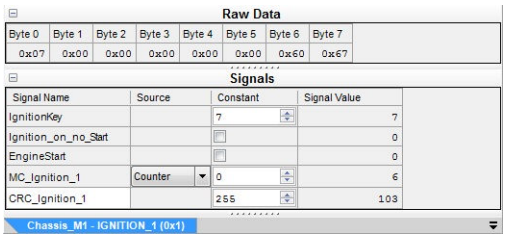


2 On the Home ribbon, click Status Control – Start Measuring to start measurement.

Note

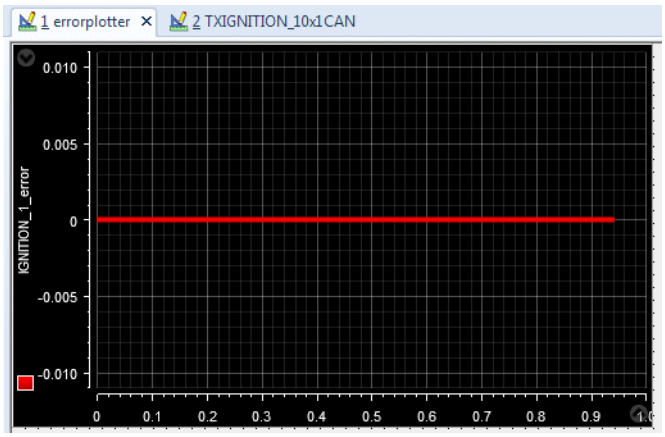
Data displayed in a Time Plotter is updated only after you start measurement. If online calibration is not yet started, it is started automatically when you start a measurement.

The message is transmitted. Since you enabled raw data display, the byte values of IGNITION_1 are displayed.



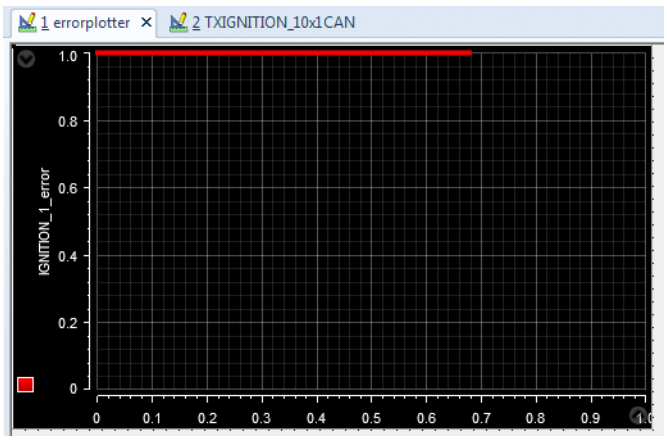
Byte 0 is in use by IgnitionKey, Byte 6 is in use by MC_Ignition_1, and Byte 7 is in use by CRC_Ignition_1.

The **Time Plotter** displays the errors transmitted by IGNITION_1. Since no error was simulated so far, the **Time Plotter** on the errorplotter layout displays an error value of 0.



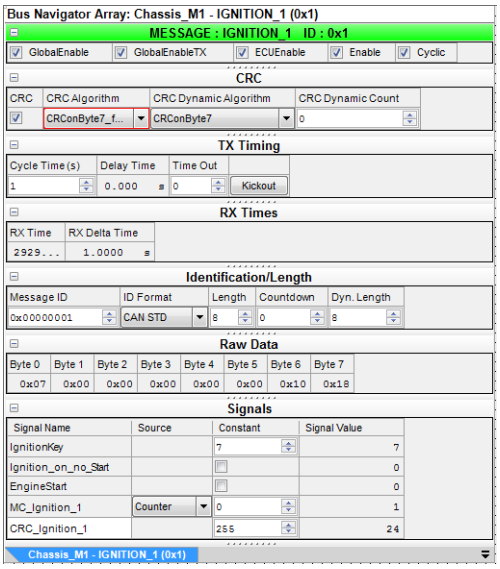
3 In the **Cycle Time(s)** edit field of the TX layout, type **`1.000`** and press **`Enter`**.

You changed the cycle time from 100 ms to 1 s. Since the error tolerance is 50 % of the default cycle time, an error occurs. The **Time Plotter** on the errorplotter layout displays an error value of 1.

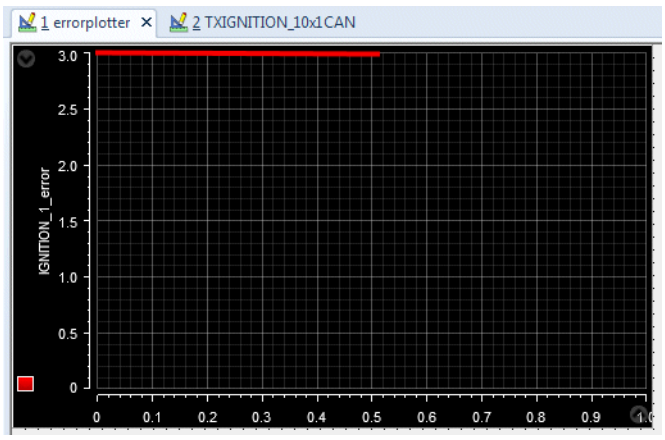


4 From the **CRC Algorithm** drop-down list, select **CRConByte7_false**.

According to the specifications in your CRC.h file, the CRConByte7_false checksum algorithm increments the CRConByte7 checksum algorithm by 1.



Since the value of the CRConByte7 checksum algorithm is expected for Byte 7, an error occurs. The error values for the checksum algorithm and the cycle time are added together in the **Time Plotter** on the `errorplotter` layout, which therefore displays an error value of 3.



5 From the **CRC Algorithm** drop-down list, select **CRConByte7**.

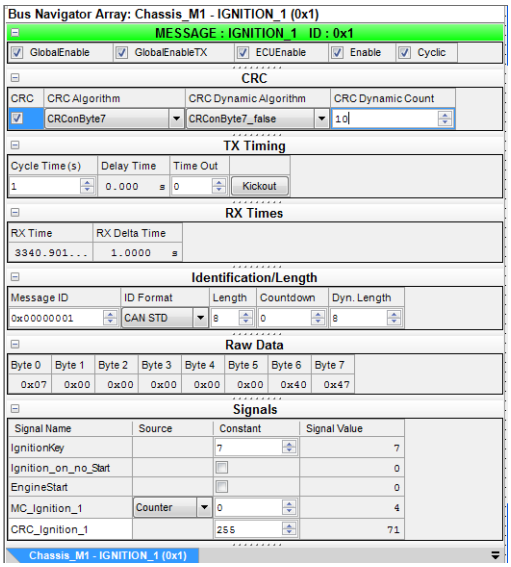
The checksum signal value is calculated according to the CRConByte7 checksum algorithm. Since this is the expected checksum signal behavior, no error occurs and the error value displayed in the Time Plotter on the errorplotter layout changes from 3 to 1.

Tip

For a detailed description on the specified checksum algorithm behavior, refer to *Step 4: How to Specify Checksums* of lesson 6.

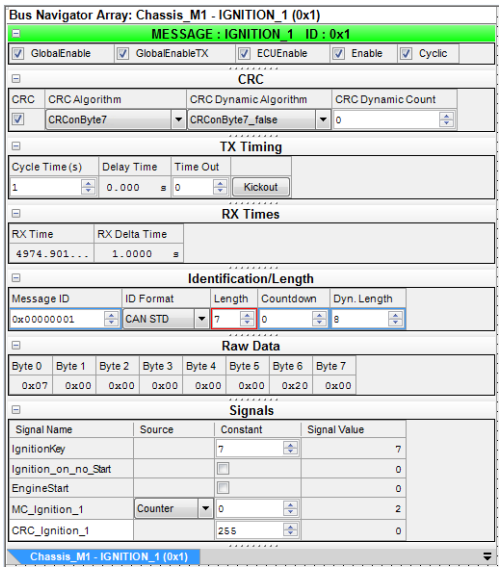
6 From the CRC Dynamic Algorithm drop-down list, select CRConByte7_false.

7 In the CRC Dynamic Count edit field, type **10** and press **Enter**.



For 10 message transmissions, the CRConByte7_false checksum algorithm is transmitted. Since a signal value calculated according to the CRConByte7 checksum algorithm is expected, an error occurs for checksum calculation. The error value displayed in the Time Plotter on the errorplotter layout changes from 1 to 3. Then CRConByte7 is used as the checksum algorithm again and no error occurs for checksum calculation.

8 In the Length edit field, specify a message length of **7**.

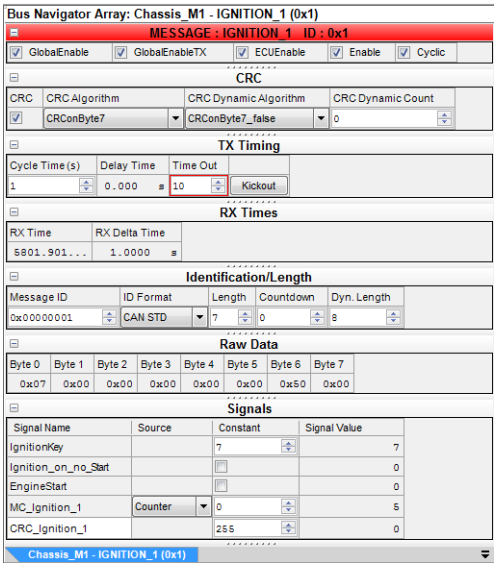


The message length changes from 8 to 7 bytes. Byte 7 is not transmitted any more and its value changes to 0. Since byte 7 is in use by CRC_Ignition_1, this signal cannot be transmitted and an error occurs for checksum calculation.

9 In the Countdown edit field, specify a countdown value of **10**.

For 10 message transmissions, the message is transmitted with the Dyn. Length value (8). Since byte 7 is transmitted, no error occurs for checksum calculation. Then the message is transmitted with a length of 7 and an error occurs.

10 In the Timeout edit field, type **10** and press **Enter**.



The header changes from green to red. The transmission of IGNITION_1 is disabled for 10 possible message transmissions. Then IGNITION_1 is transmitted again and the header changes from red to green.

11 In the Message ID edit field, specify an ID of **2**.

Changing the message ID has different effects on the controllers of your model:

- *TutMdl_C2*: In the RX layout, the RX time stops. This RX layout relates to a message with an ID of 0x1. Since the TutMdl_C2 controller does not receive a message with this ID, the RX time stops.
- *TutMdl_C1*: In the TX layout, the RX time is still running and the TutMdl_C1 controller receives the message as a loopback message.

> **Tip**
>
> Open the RX layout for IGNITION_1 on the TutMdl_C1 controller. The RX time is running and IGNITION_1 is received as a loopback message.

12 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

13 Save your project.

Result

In ControlDesk, you generated a new layout by creating an experiment and added a Time Plotter to it to visualize error values. Then you used the

customized IGNITION_1 message to simulate errors and observe the effects on the errorplotter and RX layouts.

What's next

With this step, you finished customizing messages and working with them. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 7: Customizing Messages..... 169
Overview of Lesson 7..... 169

Result of Lesson 7

Result

In this lesson, you learned the basic principles of customizing messages:

- You can customize messages, for example, to simulate message errors.
- Most manipulation options to customize messages must be enabled in your model. Depending on the manipulation option, you have to specify settings for the sender or for both the sender and the receiver.
- Messages can be customized via your model and/or via ControlDesk during run time.
- Displaying message errors must be enabled explicitly.
- To display errors in ControlDesk, you can assign a message's error variable to a Time Plotter.
- When online calibration is started in ControlDesk, you can observe the behavior of customized messages via the Time Plotter and via the message's RX layout.
- Displaying the raw data of the message bytes can help to identify errors.

Simulated message errors In this lesson, you simulated errors for the following message settings:

- ID
- Length (static and dynamic length manipulation)
- Checksum (static and dynamic checksum manipulation)
- Message transmission (timeout)
- Cycle times

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson7_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- TX ID Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Message Length Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Manipulation Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Triggering Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Cycle Time Defaults Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Cycle Time Error Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Settings Menu (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Error Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to customize signals, for example, to simulate signal errors.

Related topics

Basics

Lesson 7: Customizing Messages..... 169
Overview of Lesson 7..... 169

Lesson 8: Customizing Signals

Where to go from here

Information in this section

Overview of Lesson 8 192
In this lesson, you will customize signals, for example, to simulate signal errors.

Step 1: How to Enable Dynamic Signal Values 193
In this step, you will enable dynamic signal value options. These options let you manipulate signal values for a defined number of message transmissions.

Step 2: How to Enable Static Signal Error Values 195
In this step, you will enable static signal error values.

Step 3: How to Specify the Default Signal Manipulation 197
In this step, you will learn how to specify the default signal manipulation option for each signal to be transmitted.

Step 4: How to Prepare the Detection of Counter Signal Misbehavior 199
In this step, you will learn how to prepare the detection of counter signal misbehavior.

Step 5: How to Work with Customized Signals in ControlDesk 202
In this step, you will create a new layout in ControlDesk to display signal errors. Then you will use customized signals to simulate signal errors.

Result of Lesson 8 212
In this lesson, you learned how to customize signals and work with them.

Overview of Lesson 8

Introduction

Using the RTICANMM blockset, you can customize signals, for example, to simulate different signal errors (Signal manipulation). You can customize signals in your model or enable manipulation options to customize signals during run time. You can also enable error display in ControlDesk. The displayed error values indicate whether or not a signal is transmitted with its default settings.

What you will learn

In this lesson, you will learn how to customize basic signal settings.

- You will use customized signals to simulate signal errors.
- You will change the value of signals for a defined number of message transmissions.
- You will work with error values specified in the database file.
- You will specify a signal's default manipulation option.
- You will simulate counter signal misbehavior.
- You will display the error values resulting from the customized settings in ControlDesk.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 7.

Prepared demo

If you did not create your own model in lesson 7, you can use the prepared demo **Lesson7_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.
- You have to be familiar with the basic principles of creating a new layout in ControlDesk.

Steps

This lesson contains the following steps:

- Step 1: How to Enable Dynamic Signal Values on page 193
- Step 2: How to Enable Static Signal Error Values on page 195

- Step 3: How to Specify the Default Signal Manipulation on page 197
- Step 4: How to Prepare the Detection of Counter Signal Misbehavior on page 199
- Step 5: How to Work with Customized Signals in ControlDesk on page 202

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 8 on page 212.

Duration

Working through this lesson will take you about 70 minutes.

What's next

Now you can start to enable dynamic signal value options. These options let you manipulate signal values for a defined number of message transmissions, see Step 1: How to Enable Dynamic Signal Values on page 193.

Related topics

Basics

Lesson 8: Customizing Signals..... 191

Step 1: How to Enable Dynamic Signal Values

Objective

Dynamic signal values let you change the value of a signal for a defined number of message transmissions. During run time, you can specify the dynamic signal value and a countdown value. The countdown value determines the number of message transmissions the dynamic signal value will be transmitted for. After that the signal value is reset to the previous signal value.

In this step, you will enable dynamic signal values.

Method

To enable dynamic signal values

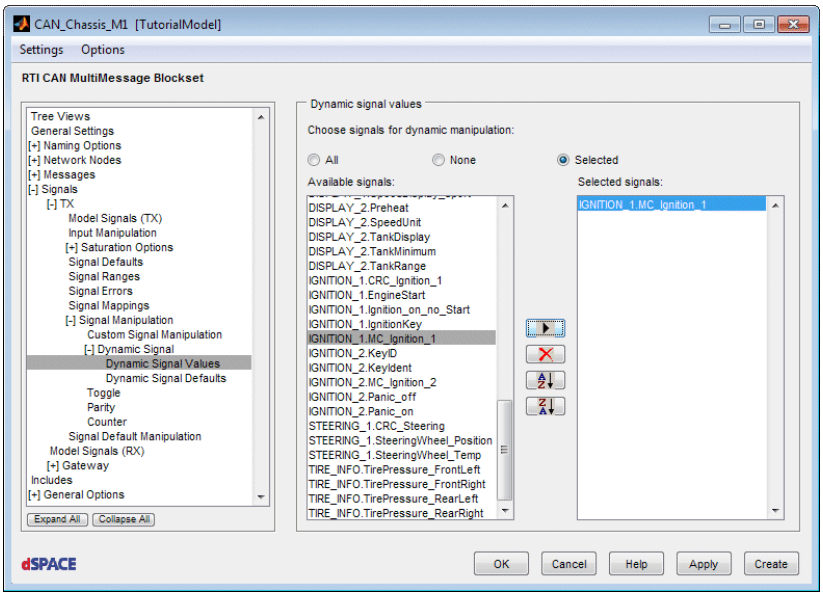
1. Open the CAN_Chassis_M1 MainBlock.
2. In the dialog tree, select Signals – TX – Signal Manipulation – Dynamic Signal – Dynamic Signal Values.

   On the Dynamic Signal Values page, you can select signals for dynamic signal value manipulation.
3. Select Selected.

   The Available signals list is enabled.
4. In the Available signals list, select IGNITION_1.MC_Ignition_1.

5 Click .

The signal IGNITION_1.MC_Ignition_1 is added to the **Selected signals** list.



6 In the Available signals list, select **DISPLAY_1.AirTempOut** and **DISPLAY_1.AirTempOut_sport**.

7 Add both signals to the **Selected signals** list.

Tip

Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Result

You enabled dynamic signal values for the IGNITION_1.MC_Ignition_1, DISPLAY_1.AirTempOut, and DISPLAY_1.AirTempOut_sport signals.

What's next

In the next step, you can enable static signal error values.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Step 2: How to Enable Static Signal Error Values

Objective

You can define error values to be transmitted for each signal. You can specify user-defined error values or use error values from the database file. During run time, you can decide whether the error value is to be transmitted. However, you cannot change the error value during run time. A red status LED in your layout indicates the transmission of an error value.

In this step, you will enable static signal error values.

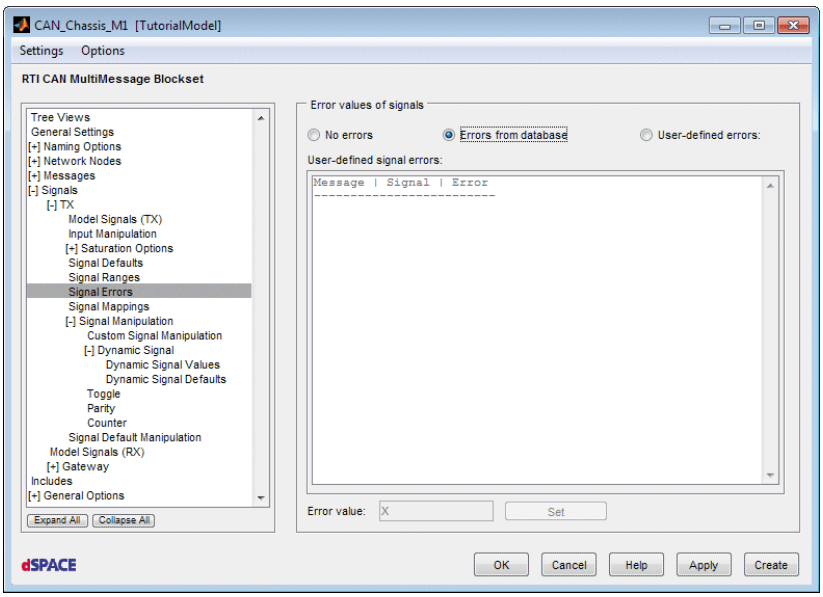
Method

To enable static signal error values

1 In the dialog tree, select Signals – TX – Signal Errors.

On the Signal Errors page, you can enable signal errors. You can either use signal errors from the database file or specify user-defined signal errors.

2 Select Errors from database.



The signal errors specified in the chassis.dbc file are taken for your model. Since the User-defined signal errors list is disabled, it is not updated and does not display any signals.

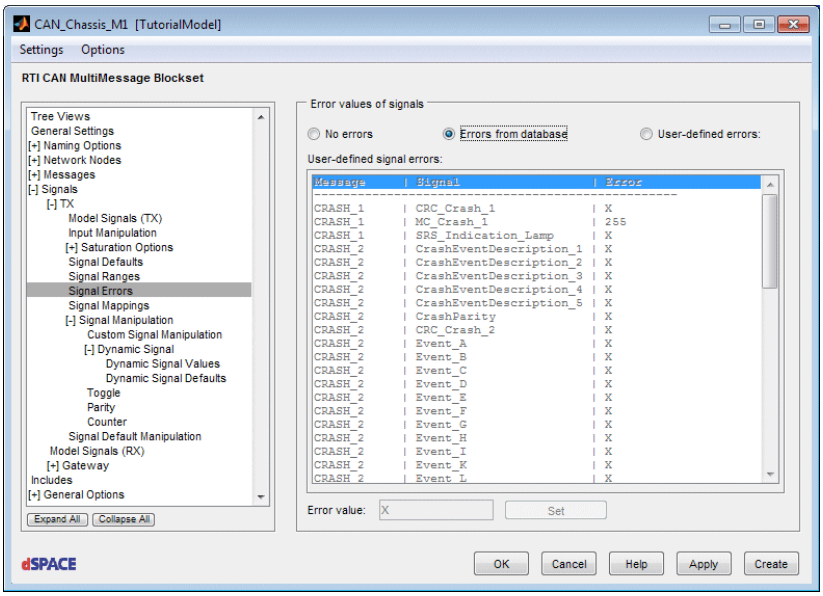
3 To display the signals specified in the database file, select User-defined errors.

The User-defined signal errors list is enabled and displays all the signals specified in the database file.

4 Scroll down the **User-defined signal errors** list to check the signal errors specified in the DBC file.

> **Tip**
>
> In the **User-defined signal errors** list, x indicates that no error value is specified for the specific signal.
> The signal errors are represented in physical units.

5 Select **Errors from database** again.



The **User-defined signal errors** list is disabled but still displays all the signals specified in the DBC file.

> **Tip**
>
> Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Result

You enabled static signal error values. You specified to use the error values defined in the chassis.dbc file.

What's next

In the next step, you can specify the default signal manipulation. For example, you can specify the error value of a signal to be its default signal manipulation.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Step 3: How to Specify the Default Signal Manipulation

Objective

You can specify several manipulation options, for example, counter, constant, or error for each signal to be transmitted. Each signal has a default manipulation option which is given according to an RTICANMM-internal hierarchy. You can change the default signal manipulation option individually for each signal.

In this step, you will change the default signal manipulation of two signals.

Method

To specify default signal manipulation

1 In the dialog tree, select Signals – TX – Signal Default Manipulation.

> **Note**
>
> There are two Signal Default Manipulation pages in the RTICANMM MainBlock. One is on the TX page and one on the Gateway page. Make sure you use the Signal Default Manipulation page on the TX page.

On the Signal Default Manipulation page, you can define a default signal manipulation option for each signal. For example, you can define the counter or error option of a signal to be its default option.

2 Select User-defined defaults.

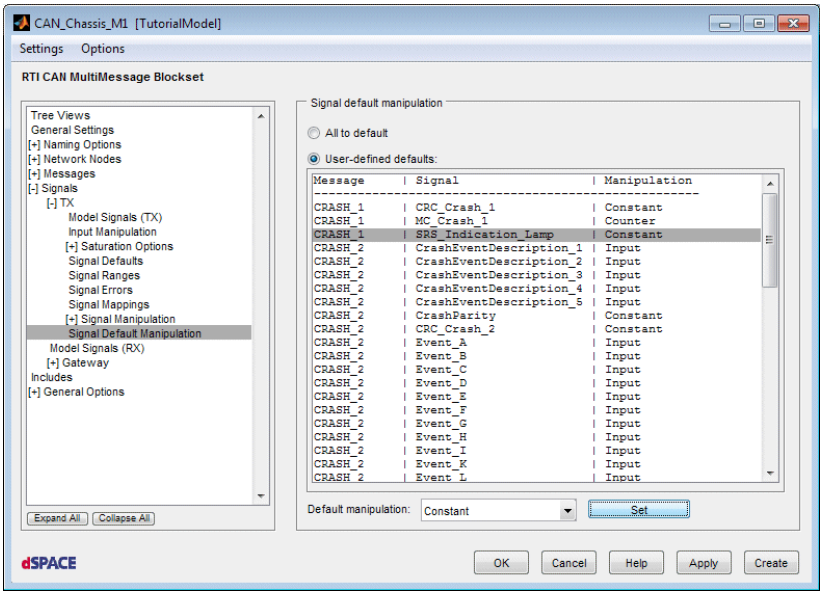
The User-defined defaults list is enabled and displays all the signals of the TX messages specified for this MainBlock.

3 In the User-defined defaults list, select CRASH_1 – SRS_Indication_Lamp.

4 From the Default manipulation drop-down list, select Constant.

5 Click Set.

> **Note**
>
> You always have to click Set to apply your settings.



The default manipulation option for the SRS_Indication_Lamp signal changes from Input to Constant.

6 In the **User-defined defaults** list, select **DIAGNOSTICS – DiagnosticSwitch**.

7 Specify **Constant** as the **Default manipulation** option.

8 Click **Set**.

Tip

Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Result

You changed the default signal manipulation for CRASH_1 – SRS_Indication_Lamp and DIAGNOSTICS – DiagnosticSwitch to Constant.

What's next

In the next step, you can prepare the detection of counter signal misbehavior.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Step 4: How to Prepare the Detection of Counter Signal Misbehavior

Objective

To provoke misbehavior for counter signals, the receiver of a message must expect a certain value for the counter signal.

- *Copy counter settings*: You can copy the counter settings specified for the sending MainBlock to the receiving MainBlock. When you do this, the receiving MainBlock expects the default settings specified for counter signals. If you change the default settings during run time, an error will occur.
- *Counter behavior during timeout*: You can specify whether the counter stops or continues counting during the timeout of a message. If the counter continues counting, the transmitted counter value and the expected counter value might not match.

  For example, a message has been transmitted 3 times. Then you specify a timeout of 5. The message is not transmitted for 5 possible message transmissions. When transmission of the message starts again, the message receiver expects a counter value of 4. If the counter continued counting, it has a value of 8. Since the transmitted and the expected counter values do not match, an error will occur.

In this step, you will learn how to prepare the detection of counter signal misbehavior.

Tip

To display errors and misbehavior of signals and/or messages, you have to enable error display. Since you already enabled error display in step 3 of lesson 7, you do not have to enable it in this lesson.

Part 1

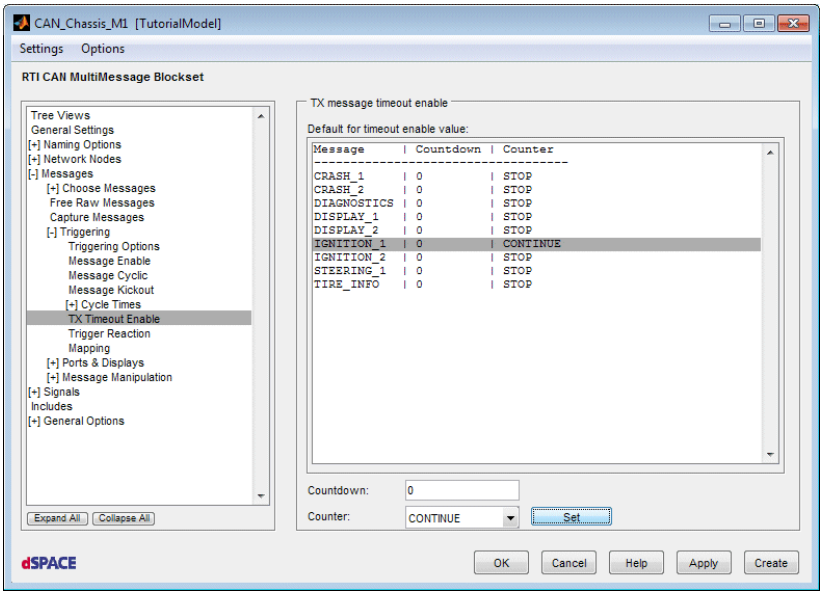
To specify the counter behavior during timeout

1. In the dialog tree, select Messages – Triggering – TX Timeout Enable.

   On the TX Timeout Enable page, you can specify default timeout values for messages and specify the counter behavior during timeout.
2. In the Default for timeout enable value list, select IGNITION_1.
3. From the Counter drop-down list, select CONTINUE.
4. Click Set.

Note

You always have to click Set to apply your settings.



The counter behavior for IGNITION_1 changes from STOP to CONTINUE.

5 In the Default for timeout enable value list, select IGNITION_2.

6 For IGNITION_2, specify CONTINUE as the Counter state.

Interim result

You changed the counter behavior during timeout for IGNITION_1 and IGNITION_2 from STOP to CONTINUE. You can now copy the settings specified for counter signals in the CAN_Chassis_M1 MainBlock. Continue with part 2.

Part 2

To copy settings of dialog pages

1 In the dialog tree, select Signals – TX – Signal Manipulation – Counter.

> **Note**
>
> The page settings need some time to be loaded. To prevent misbehavior, do not continue with the next step before the page is loaded completely.

2 From the menu bar of the CAN_Chassis_M1 [TutorialModel] dialog, select Settings – Save Current Page Settings.

3 In the Settings MAT-file dialog, type **`Counter`** in the File name field.

4 Click Save.

The dialog closes and the settings of the Counter page are saved.

5 Click Create.

The CAN_Chassis_M1 MainBlock is created.

6 Click OK to close the CAN_Chassis_M1 [TutorialModel] dialog.

Interim result

You saved the settings specified on the Counter page of the CAN_Chassis_M1 MainBlock. You can now apply these settings to the CAN_Chassis_M2 MainBlock. Continue with part 3.

Part 3

To load settings of dialog pages

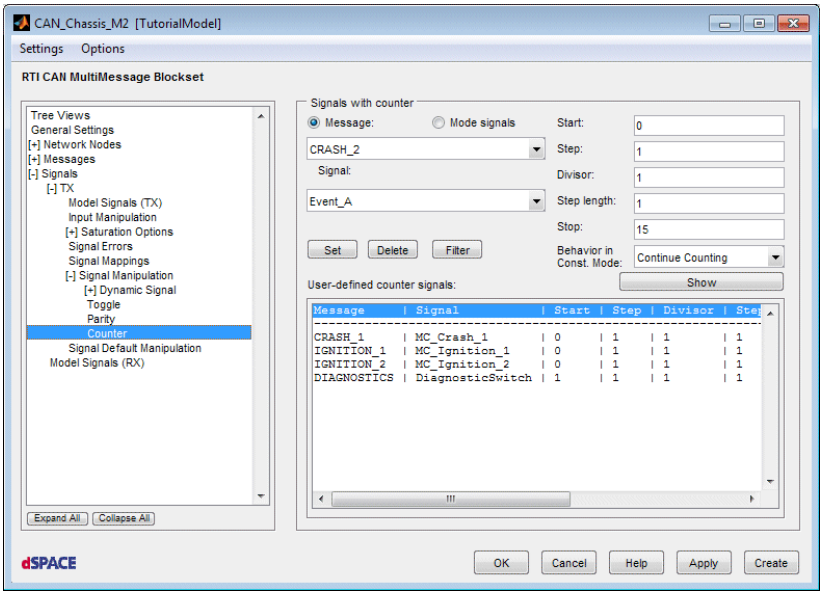
1 Open the CAN_Chassis_M2 MainBlock.

2 In the dialog tree, select Signals – TX – Signal Manipulation – Counter.

3 From the menu bar of the CAN_Chassis_M2 [TutorialModel] dialog, select Settings – Load Current Page Settings.

4 In the Settings MAT-file dialog, select Counter.mat and click Open.

The dialog closes and the settings specified in the Counter MAT file are loaded to the Counter page.



5 Click Create.

The CAN_Chassis_M2 MainBlock is created, and the specified settings apply to your model.

6 Click OK to close the CAN_Chassis_M2 [TutorialModel] dialog.

7 Save your model.

8 Build an application for your model.

Result

You prepared the detection of counter signal misbehavior.

What's next

In the next step, you can work with customized signals in ControlDesk.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Step 5: How to Work with Customized Signals in ControlDesk

Objective

To work with customized signals in ControlDesk, three layouts are usually needed:

- The message's TX layout of the sending CAN controller.
- The message's RX layout of the receiving CAN controller.
- An additional layout to display the error values.

If you change signal settings via the TX layout or via the model, you can observe the effects in the RX layout and/or in the layout that displays the error values.

In this step, you will create a new errorplotter layout to display signal errors. Then you will use customized signals to simulate signal errors.

Part 1

To display errors

1. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
2. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
3. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.
4. From the platform's context menu in the Project controlbar, select Add Variable Description.

   ControlDesk opens the Select Variable Description dialog.

5 In the Select Variable Description dialog, click Import from file and select the `TutorialModel.sdf` file from your `RTICANMMTutorial` folder.

Note

To make sure that you add the latest version of the SDF file to the experiment, always use Import from file to import the file. If the Select Variable Description dialog displays the `TutorialModel.sdf` file and you select this file directly, you might load an obsolete version. This might cause problems while you work through this tutorial.

6 In the Select Variable Description dialog, click OK.

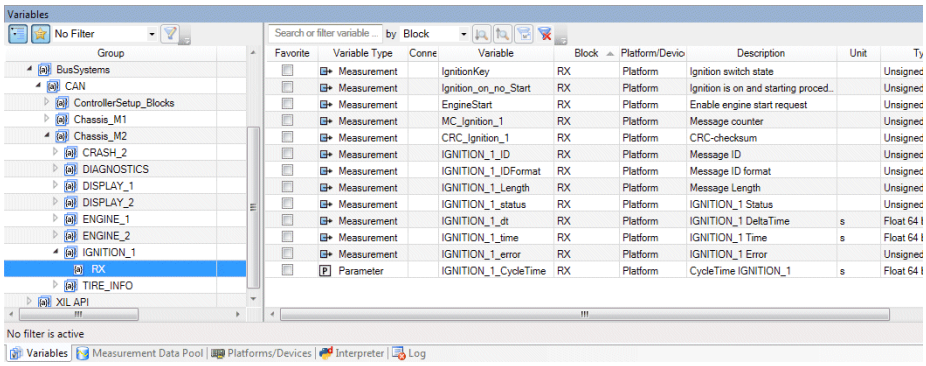
ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

7 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

8 To rename the layout, select Rename from the layout's context menu in the Project controlbar and enter **`errorplotter`**. Click OK.

9 Select the Variables controlbar.

10 In the tree view, select tutorialmodel – BusSystems – CAN – Chassis_M2 – IGNITION_1 – RX.



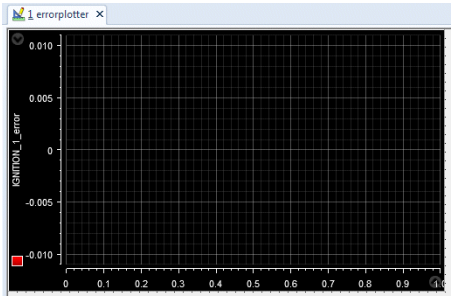
The window on the right displays all the available variables for the IGNITION_1 RX message.

11 From the variable list, drag the IGNITION_1_error variable onto an empty space in the errorplotter layout.

A drop-down list opens for you to select an instrument to assign to the variable.

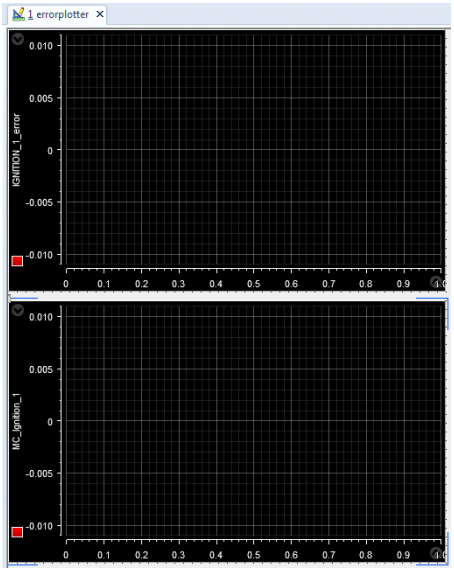
12 From the list, select **Time Plotter**.

A **Time Plotter** is added to the **errorplotter** layout and the y-axis of the **Time Plotter** displays the name of the connected signal.



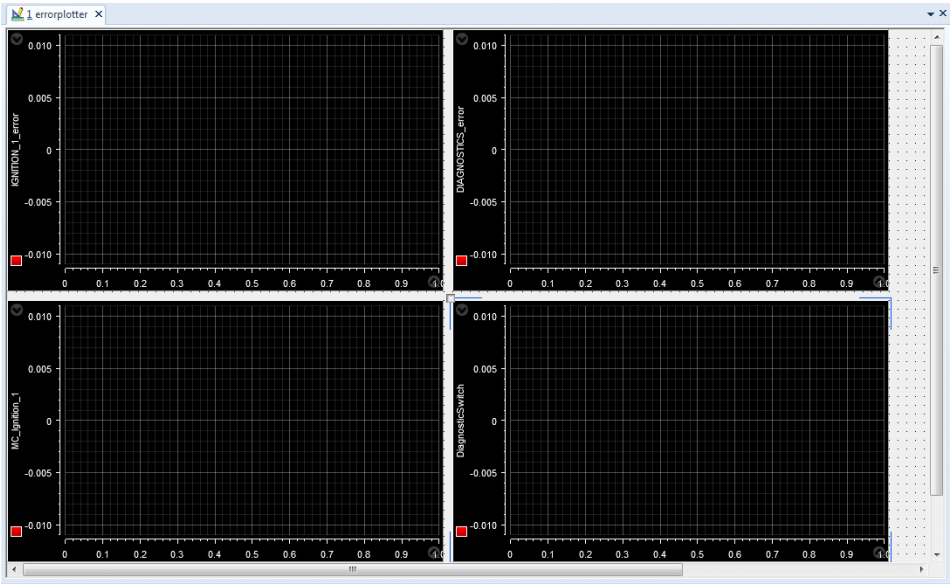
13 Repeat steps 11…12 for the **MC_Ignition_1** variable.

The y-axis of the Time Plotter displays the name of the connected signal.



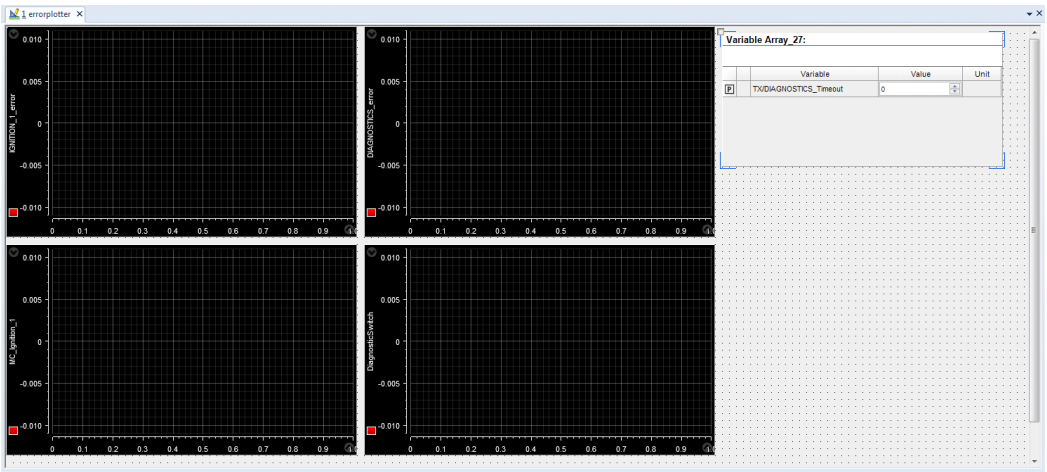
14 In the tree view, select **tutorialmodel – BusSystems – CAN – Chassis_M2 – DIAGNOSTICS – RX**.

15 Repeat steps 11…12 for the DIAGNOSTICS_error and DiagnosticSwitch variables.



16 In the tree view of the Variables controlbar, select tutorialmodel – BusSystems – CAN – Chassis_M1 – DIAGNOSTICS – TX.

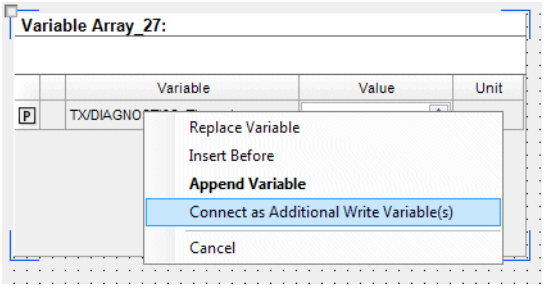
17 Repeat steps 11…12 for the DIAGNOSTICS_Timeout variable, but select Variable Array instead of Time Plotter.



18 In the tree view of the Variables controlbar, select tutorialmodel – BusSystems – CAN – Chassis_M1 – IGNITION_1 – TX.

19 From the variable list, drag the IGNITION_1_Timeout variable onto the DIAGNOSTICS_Timeout variable in the Variable Array using the right mouse button.

20 From the context menu, select Connect as Additional Write Variable(s).



21 Save the project.

Interim result

You have generated your errorplotter layout. You can display the counter and error values of IGNITION_1 and DIAGNOSTICS via this layout. Additionally, you can specify a timeout value for both messages via the Variable Array. You can now transmit static signal errors for DISPLAY_2. Continue with part 2.

Part 2

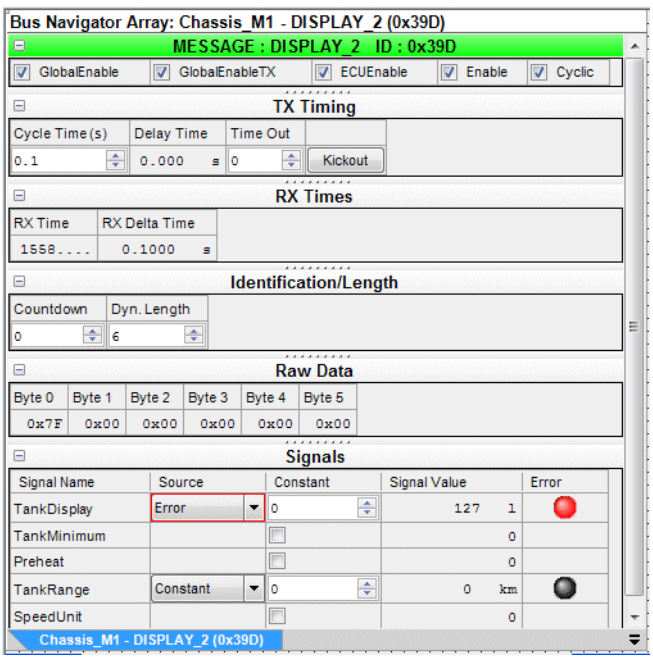
To transmit static signal errors for DISPLAY_2

1 In the Bus Navigator tree, right-click DISPLAY_2 below the Chassis_M2 [chassis.dbc] node and generate the RX layout.

2 Right-click DISPLAY_2 below the Chassis_M1 [chassis.dbc] node and generate the TX layout.

3 On the Home ribbon, click Status Control – Start Measuring to start measurement.

4 From the Source drop-down list of the TankDisplay signal, select Error.



The signal's value changes from 0 to 127. This is the signal's error value specified in the chassis.dbc file. The Error LED is active and indicates that the error value is transmitted. The TutMdl_C2 controller receives the error value.

Interim result

You transmitted the static error value of TankDisplay and observed the effects on the TX and RX layouts. You can now simulate signal errors for IGNITION_1. Continue with part 3.

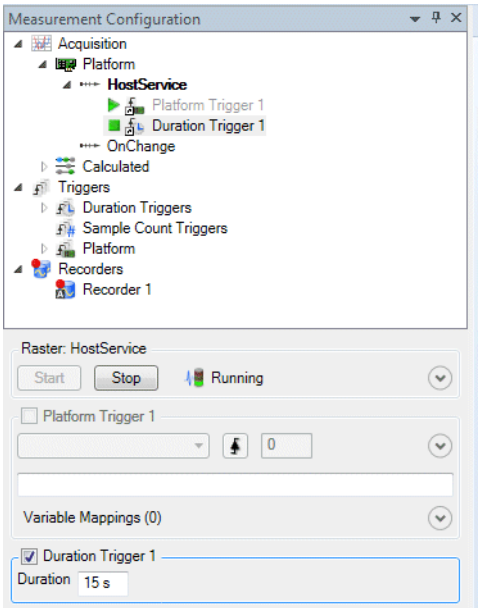
Part 3

To simulate signal errors for IGNITION_1

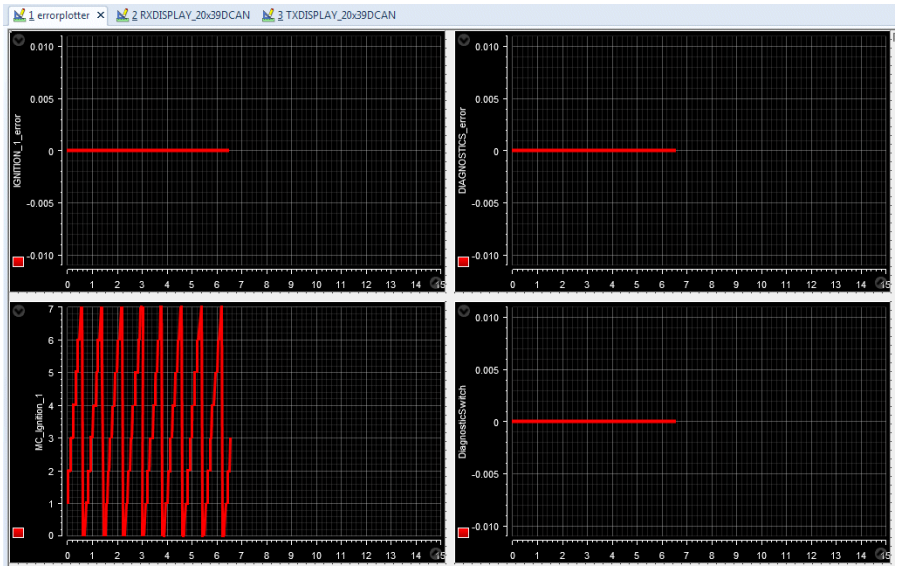
1 In the Measurement Configuration controlbar, select Acquisition – Platform – HostService – Duration Trigger 1.

> **Tip**
>
> If the Measurement Configuration controlbar is hidden, go to the View ribbon and select Controlbar – Switch Controlbars – Measurement Configuration.

2 In the Duration edit field, specify **15** seconds.



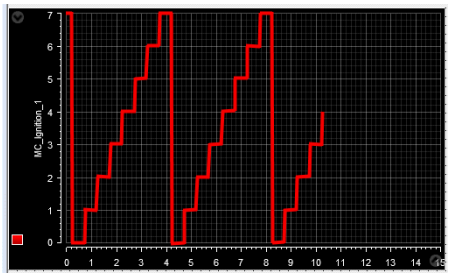
The x-axes of all the Time Plotters are scaled to 15. You can observe the behavior of the counter signal in the Time Plotter displaying the MC_IGNITION_1 signal.



3 Generate the TX layout of IGNITION_1 for Chassis_M1.

4 In the TX layout of IGNITION_1, type **0.500** in the Cycle time(s) edit field and press **Enter**.

The cycle time is reduced and you can observe every step of the counter signal in the corresponding Time Plotter on the errorplotter layout.



Tip

For a better understanding of the behavior of the counter signal, select the Counter page (which you can find on the Signals – TX – Signal Manipulation page in the dialog tree) of your CAN_Chassis_M1 MainBlock. On the Counter page, select the MC_Ignition_1 signal and click Edit. Then click Show to display the specified counter signal settings.

Since you manipulated the cycle time, the Time Plotter displaying the error signal of IGNITION_1 shows an error value of 1.

5 In the TX layout of IGNITION_1, type **5** in the Dyn. Value edit field and press **Enter**.

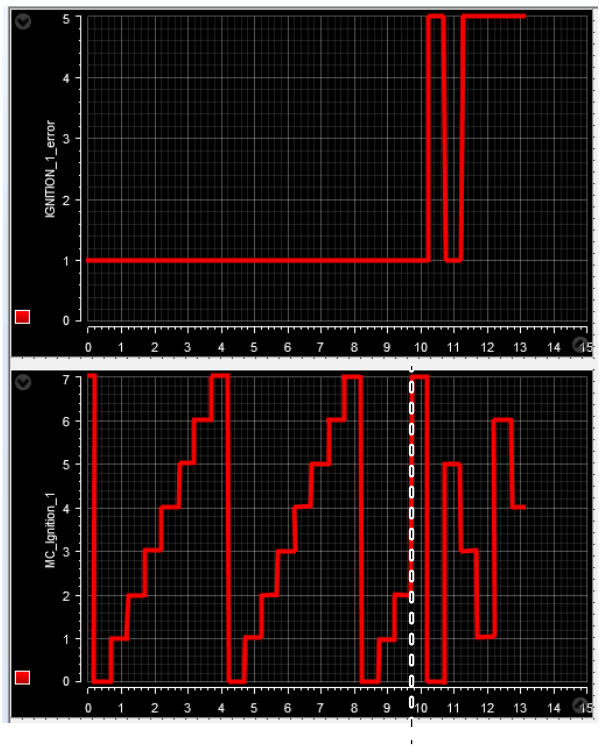
6 Specify a **Countdown** of **25**.

Signals

Signal Name	Source	Constant	Signal Value	Countdown	Dyn. Value
IgnitionKey		7	7		
Ignition_on_no_Start			0		
EngineStart			0		
MC_Ignition_1	Counter	0	3	25	5
CRC_Ignition_1		255	55		

7 From the **Source** drop-down list, select **DynValue**.

For the next 25 message transmissions, the MC_Ignition_1 counter signal is transmitted with the dynamic value. You can observe the behavior of the counter signal in your errorplotter layout. When the transmission of the dynamic value starts, the counter value is increased by 5. This generates an error.



In the illustration above, dynamic value transmission starts at a counter value of 2. The counter value is increased to 7. Since 3 was expected as the counter value, an error is generated.

Since 7 is the counter stop value, the counter turns around to the counter start value (0). This is the expected behavior and no error is generated. Then the counter value is increased to 5. Since a counter value of 1 was expected, an error occurs.

After the specified 25 message transmissions, the normal counter value is transmitted and no errors occur.

Interim result

You transmitted a dynamic signal value for MC_Ignition_1 and observed the effects displayed in the errorplotter layout. You can now display the different counter behaviors of IGNITION_1 and DIAGNOSTICS. Continue with part 4.

Part 4

To display different counter behaviors of IGNITION_1 and DIAGNOSTICS

1 Generate the TX layout of DIAGNOSTICS for Chassis_M1.

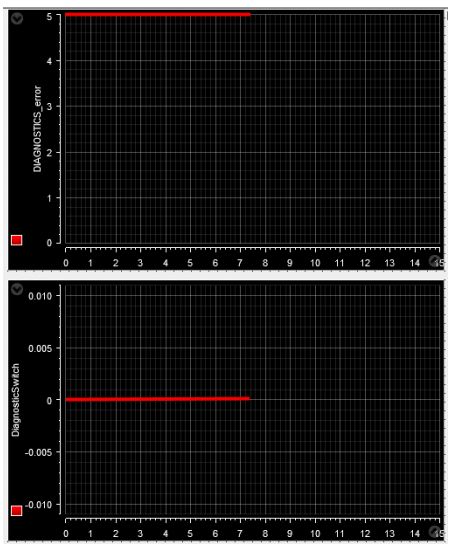
Since the cycle time is 0, the message is not transmitted yet.

2 In the Cycle Time (s) edit field, type **`0.500`** and press **`Enter`**.

The message is transmitted. In your model, you specified Constant as the signal default manipulation option for the DiagnosticSwitch signal. For this reason,

- DiagnosticSwitch is transmitted with its constant value (0) and not with the counter value.
- The DiagnosticSwitch Time Plotter in your errorplotter layout displays 0.

Since you manipulated the cycle time and the signal's receiver expects the counter value, the DIAGNOSTICS_error Time Plotter displays an error value of 5.



3 In the TX layout of DIAGNOSTICS, select Counter from the Source drop-down list.

The DiagnosticSwitch Time Plotter displays the counter value. Since the counter value is the expected signal value, no error occurs for the counter signal. However, an error value of 1 is displayed in the DIAGNOSTIC_error Time Plotter because you manipulated the cycle time.

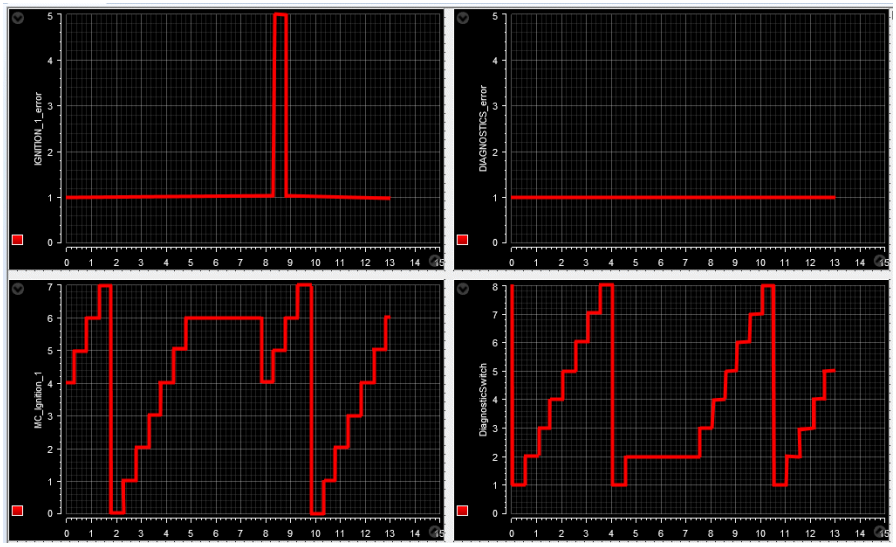
4 In the errorplotter layout, type **`5`** in the Variable Array and press **`Enter`**.

This specifies a timeout for IGNITION_1 and DIAGNOSTICS. Both messages are not transmitted for 5 possible transmissions. After that both messages are transmitted again.

In your model, you specified the following settings on the TX Timeout Enable page:

Message	Counter
IGNITION_1	CONTINUE
DIAGNOSTICS	STOP

These settings result in a different counter behavior during timeout.



- *MC_Ignition_1*: In the illustration above, the timeout started at a counter value of 6. After 5 suspended message transmissions, the message is transmitted again. Internally, the counter has not stopped counting and the first message is transmitted with a counter value of 4. Since 7 was expected as the counter value, an error occurs.
- *DiagnosticSwitch*: The timeout started at a counter value of 2. After 5 suspended message transmissions, the message is transmitted again. Internally, the counter has stopped counting and the first message is transmitted with a counter value of 3. Since this is the expected counter value, no error occurs.

5 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

6 Save the project.

Result

You added instruments to your errorplotter layout to display signal error values and values of counter signals for IGNITION_1 and DIAGNOSTICS. Then you simulated various signal errors and observed the effects on the RX and errorplotter layouts.

What's next

With this step, you finished customizing signals and working with them. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Result of Lesson 8

Result

In this lesson, you learned the basic principles of customizing signals:

- You can customize signals, for example, to simulate signal errors.
- Most manipulation options to customize signals must be enabled in your model. Depending on the manipulation option, you have to specify settings for the sender or for both the sender and the receiver.
- Using customized signals, errors can easily be simulated via ControlDesk.
- Signal error display must be enabled explicitly.
- To display errors in ControlDesk, you can assign a message's error variable to a Time Plotter.
- After starting a measurement in ControlDesk, you can observe the behavior of customized signals via the Time Plotter and via the message's RX layout.

Simulated signal errors In this lesson, you simulated errors for the following signal settings:

- Dynamic signal values
- Static signal error values
- Default signal manipulation
- Counter behavior during timeout

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson8_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Dynamic Signal Values Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Signal Errors Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Signal Default Manipulation Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Timeout Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

- Settings Menu (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Error Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to update the database specified for your model.

Related topics

Basics

Lesson 8: Customizing Signals..... 191
Overview of Lesson 8..... 192

Lesson 9: Updating the Database

Where to go from here

Information in this section

Overview of Lesson 9 215
In this lesson, you will update the database of your model. You will assign a different database file.

Step 1: How to Update the Database and Check Automatic Effects on the Model 217
In this step, you will specify a new database file for your model, detect the differences between the two files, and check the automatic effects on your model.

Step 2: How to Adapt the Model to the New DBC File Manually 220
In this step, you will adapt the user-defined settings of your model to the new database.

Result of Lesson 9 223
In this lesson, you learned how to update the database file of your model.

Overview of Lesson 9

Introduction

You can update the database file of your model in two ways: You can update the specified database file by another file version, or assign a different database file. Updating the database has the following effects on your model:

- *Default settings*: All the default settings of your model are automatically adapted to the updated/new database file.
- *User-defined settings*: Since user-defined settings are not adapted automatically, you have to adapt them manually. For this reason, you should always specify as few user-defined settings as possible.

Tip

Some dialog pages provide radio buttons to specify settings (for example, the TX Messages Page and the Signal Errors page). If you use the radio buttons to specify settings, the settings on these pages are automatically adapted to an updated/new database file.

What you will learn

In this lesson, you will learn how to update the database of your model using a new database file.

- You will update the database file of your model by specifying a different DBC file.
- You will check the differences between the new and the old DBC file.
- You will adapt your model to the new DBC file.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 8.

Prepared demo

If you did not create your own model in lesson 8, you can use the prepared demo **Lesson8_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The **Current Directory** in MATLAB must be your RTICANMMTutorial folder and the **TutorialModel** model must be open.
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).

Steps

This lesson contains the following steps:

- Step 1: How to Update the Database and Check Automatic Effects on the Model on page 217
- Step 2: How to Adapt the Model to the New DBC File Manually on page 220

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 9 on page 223.

Duration

Working through this lesson will take you about 30 minutes.

What's next

Now you can start to update the database of your model, see Step 1: How to Update the Database and Check Automatic Effects on the Model on page 217.

Related topics

Basics

Lesson 9: Updating the Database..... 215

Step 1: How to Update the Database and Check Automatic Effects on the Model

Objective

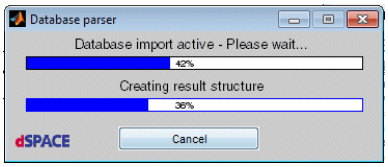
You can update the database file specified for your model or specify a new database file. When you update or replace the database file, it is parsed automatically and a log file displaying the differences between the database files is generated. The log file helps you to identify the user-defined settings that must be updated.

In this step, you will specify a new database file for your model, detect the differences between the two files, and check the automatic effects on your model.

Part 1

To update the database by assigning a different database file

1. In your MATLAB Command Window, type **`rticanmm_tutorial`**.
   The rticanmm_tutorial_lib library opens.
2. In the rticanmm_tutorial_lib library, double-click chassis_changed.dbc.
   The chassis_changed.dbc file is copied to your RTICANMMTutorial folder.
3. Close the rticanmm_tutorial_lib library.
4. In your model, double-click the CAN_Chassis_M1 block to open it.
5. Select the General Settings page.
6. Click .
7. In the File dialog, select the chassis_changed.dbc file and click Open.
   The Database Parser dialog opens and the DBC file is parsed automatically.



   After parsing, the dialog closes.

8 Select your MATLAB Command Window.

In your MATLAB Command Window, you will find a link to a log (HTML) file in your RTICANMMTutorial folder, summarizing the differences between the chassis.dbc file and the chassis_changed.dbc file.

9 Open the **compare_chassis_changed_chassis_log.htm** log file.

```
Compare CAN Datas
Log File
checked Data: chassis_changed, chassis
checked on: 15:11:44 9-8-2007

Data: chassis_changed

Message: TIRE_INFO_2
  Message not found in chassis

Data: chassis

Message: CRASH_2
  Signal "Event_B" not found in "chassis_changed"
  Signal "CrashEventDescription_5" not found in "chassis_changed"
Message: IGNITION_2
  Message not found in chassis_changed
```

The log file displays the following information:

- **Message: TIRE_INFO_2**: This message is new in the chassis_changed.dbc file.
- **Message: CRASH_2**: The **Event_B** and **CrashEventDescription_5** signals are not available in the chassis_changed.dbc file.
- **Message: IGNITION_2** This message is not available in the chassis_changed.dbc file.

10 Close the log file.

11 In the **CAN_Chassis_M1 [TutorialModel]** dialog, click **OK** to close the dialog.

> **Note**
>
> Do not create the CAN_Chassis_M1 MainBlock.

12 Open the **CAN_Chassis_M2** MainBlock.

13 Specify the chassis_changed.dbc file as the new database.

14 In the **CAN_Chassis_M2 [TutorialModel]** dialog, click **Create**.

15 Click **OK** to close the **CAN_Chassis_M2 [TutorialModel]** dialog.

Interim result

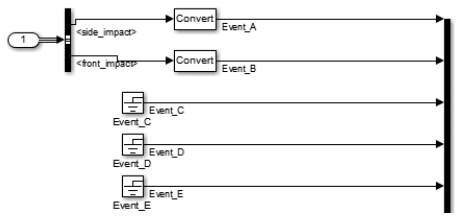
You updated the database file of your model and checked the differences between the chassis.dbc and the chassis_changed.dbc file. You can now check the effects of the new database file on your model. Continue with part 2.

Part 2

To check the effects of the new database on the model

1. In your model, open the Mapping to RTICANMM block.
2. Open the substructure CRASH_2.

   Event_B is connected to front_impact.

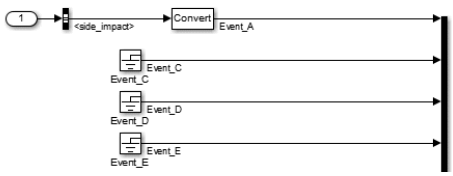


3. Close the CRASH_2 substructure and the Mapping to RTICANMM block.
4. Open the CAN_Chassis_M1 MainBlock and click Create.

   The CAN_Chassis_M1 MainBlock is created.

   The Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten, and the settings specified in the CAN_Chassis_M1 MainBlock apply to your model.
5. Click OK to close the CAN_Chassis_M1 [Tutorial] dialog.
6. Open the Mapping to RTICANMM and the substructure CRASH_2.

   Since no manual changes were made in the CRASH_2 substructure, it was automatically adapted to the new database. Event_B and the connection belonging to it are deleted.



7. Close the CRASH_2 substructure and the Mapping to RTICANMM block.
8. Save your model.

Result

You updated the database of your model and checked the differences between the two database files. Additionally, you checked the effects of the new database on your model. Since you made no manual changes to the Mapping to RTICANMM block, it was adapted automatically.

What's next

In the next step you can adapt the user-defined settings of your model to the new database.

Related topics

Basics

Lesson 9: Updating the Database..... 215
Overview of Lesson 9..... 215

Step 2: How to Adapt the Model to the New DBC File Manually

Objective

Manual settings in your model must be adapted to the database manually after the database update. Messages and signals not available with the new/updated database file are deleted automatically. However, new messages and signals are not added to manual selections automatically. User-defined settings which are affected by the updated database must be adapted manually.

In this step, you will update some of the manual settings according to the new database file.

Part 1

To adapt the CAN_Chassis_M1 MainBlock to the new DBC file

1 Open the CAN_Chassis_M1 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

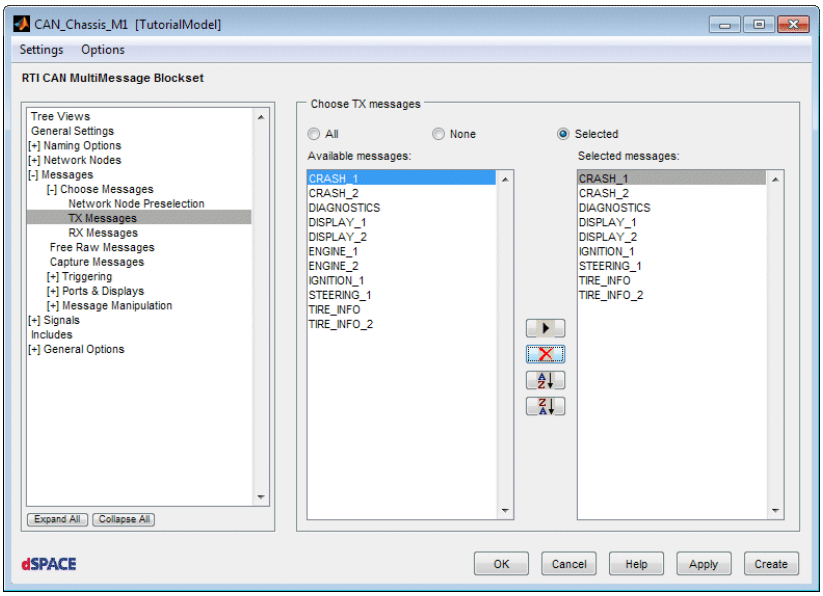
The settings of this page are affected by the updated database. Since you made manual changes on this page in previous lessons, you have to adapt it to the new database manually. In previous lessons, you selected all the messages not transmitted by the selected ECU as TX messages. Since you do not know whether the new TIRE_INFO_2 message is transmitted by the selected ECU, you have to repeat the selection of the messages not transmitted by the selected ECU.

3 Select all the messages displayed in the Selected messages list.

4 Click to delete all the messages in the list.

A selection of messages is never overwritten. If you select new messages, they are added to the already selected messages. You should therefore always delete all the selected messages before you select new messages.

5 Right-click in the **Selected messages** list and select **Choose Messages Not Transmitted by Selected Network Nodes** from the context menu.



Since the new message TIRE_INFO_2 is not transmitted by the selected network node, it is added to the Selected messages list.

Note

In previous lessons you specified, for example, counter, checksum, and TX signals. In principle, these settings must be updated manually. However, TIRE_INFO_2 is not intended to be used, for example, as a counter, checksum or TX signal, so in this case you do not have to update these settings manually.

6 Click **Create**.

The **CAN_Chassis_M1** MainBlock is created. The **Overwrite?** dialog opens twice and asks you whether you want to overwrite the Mapping to RTICANMM and Triggering to RTICANMM block.

7 In the **Overwrite?** dialogs, click **Overwrite**.

The Mapping to RTICANMM and Triggering to RTICANMM blocks are overwritten, and the settings specified in the CAN_Chassis_M1 MainBlock apply to your model.

8 Click **OK** to close the **CAN_Chassis_M1 [TutorialModel]** dialog.

Interim result

You adapted the CAN_Chassis_M1 MainBlock to the new database. You can now adapt the CAN_Chassis_M2 MainBlock. Continue with part 2.

Part 2

To adapt the CAN_Chassis_M2 MainBlock to the new DBC file

1. Open the CAN_Chassis_M2 MainBlock.
2. In the dialog tree, select Messages – Choose Messages – RX Messages.
3. Select all the messages displayed in the Selected messages list and click .
4. Right-click in the Selected messages list and select Choose Messages Received by Selected Network Nodes from the context menu.

   Note

   In previous lessons, you selected TX messages according to ENGINE_CONTROL. Since TIRE_INFO_2 is not transmitted by ENGINE_CONTROL, you do not have to update the TX Messages page.

5. Click Create.
6. Click OK to close the CAN_Chassis_M2 [TutorialModel] dialog.
7. Save your model.

Result

You adapted manual settings of both MainBlocks to the new database. You adapted only settings which are affected by the changed database.

Note

Since the changes in the new DBC file have only little effect on your model, you updated only a few dialog pages. In realistic use cases, database updates might have much more effect on the model, so updating the model manually is more complex.

What's next

With this step, you finished updating the database of your model. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 9: Updating the Database..... 215
Overview of Lesson 9..... 215

Result of Lesson 9

Result

In this lesson, you learned the basic principles of updating the database:

- You can easily update the database via the General Settings page.
- The new or updated database file is parsed automatically.
- A log file is generated which lists all the differences between the two database files.
- All the default settings of your model are adapted to the new database automatically.
- User-defined settings that are affected by the new database must be adapted manually.
- Some dialog pages provide radio buttons to specify settings (for example, the TX Messages Page and the Signal Errors page). If you use the radio buttons to specify settings, the settings on these pages are automatically adapted to an updated/new database file.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson9_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Mapping to RTICANMM Block (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to work with variants of a CAN controller.

Related topics

Basics

Lesson 9: Updating the Database 215
Overview of Lesson 9 215

Lesson 10: Working with Variants of a CAN Controller

Where to go from here

Information in this section

Overview of Lesson 10 225
In this lesson, you will learn how to work with variants of a CAN controller.

Step 1: How to Specify Variants of a CAN Controller 227
In this step, you will build a model for working with two variants of a CAN controller.

Step 2: How to Work with Variants of a CAN Controller in ControlDesk 235
In this step, you will work with different variants of a CAN controller. You will change the active variant of the controller during run time.

Result of Lesson 10 239
In this lesson, you learned how to work with variants of a CAN controller.

Overview of Lesson 10

Introduction

You can work with up to 20 variants of one CAN controller. You have to use a separate RTICANMM MainBlock and define a variation index for each variant.

You can define variants of a CAN controller in two ways by:

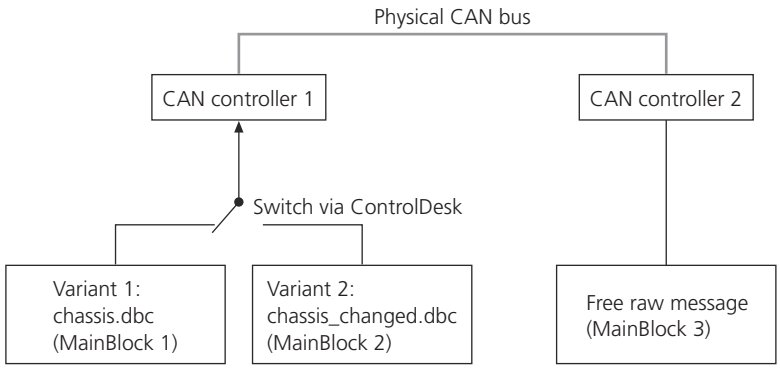
- Specifying different DBC files for one CAN controller.
- Using the same DBC file but defining a different CAN communication via the RTICANMM MainBlocks.

Only one variant can be active at a time, but you can switch between the variants during run time.

CAN controller variants in this lesson

In this lesson, you will work with two CAN controllers:

- For the first CAN controller, you will specify two variants using different DBC files. The **chassis.dbc** file will be the first variant and the **chassis_changed.dbc** file the second variant.
- For the second CAN controller, you will specify one free raw message. Using the free raw message, you will observe the different behavior of the variants of the first CAN controller in ControlDesk.



Differences between the CAN controller variants The DBC files you will specify for the first CAN controller have the following differences:

Message	Chassis.dbc	Chassis_changed.dbc
CRASH_2	Included	Included except for the Event_B and CrashEvent_Description_5 signals
IGNITION_2	Included	Not included
TIRE_INFO_2	Not included	Included

What you will learn

In this lesson, you will learn the basic principles of working with variants of a CAN controller.

- You will build a model to work with different variants of a CAN controller.
- You will work with two DBC files on one CAN controller.
- You will change the active variant during run time and observe the effects on CAN communication.

Starting point

The starting point of this lesson is a new model in MATLAB.

Prepared demo

Since you will build a new model in this lesson, no prepared demo is required.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- You have to be familiar with building a model using RTICANMM blocks (refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39).
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Specify Variants of a CAN Controller on page 227
- Step 2: How to Work with Variants of a CAN Controller in ControlDesk on page 235

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 10 on page 239.

Duration

Working through this lesson will take you about 30 minutes.

What's next

Now you can start to build a model for working with variants of a CAN controller, see Step 1: How to Specify Variants of a CAN Controller on page 227.

Related topics

Basics

Lesson 10: Working with Variants of a CAN Controller..... 225

Step 1: How to Specify Variants of a CAN Controller

Objective

Each variant of a CAN controller must be specified in a separate RTICANMM MainBlock. In the MainBlock, you have to select the CAN controller and the variation index. The variant with variation index 1 is the active variant by default. You can change the active variant during run time.

In this step, you will build a model with two CAN controllers. You will specify the chassis.dbc and chassis_changed.dbc files as variants of the first CAN controller

and define a free raw message for the second CAN controller. Using the free raw message of the second CAN controller, you will observe the different behaviors of the CRASH_2 message of both variants.

Part 1

To create a model and add RTICANMM blocks to it

1. In your working folder, create a new folder named **`RTICANMMTutorial_Lesson10`**.
2. In MATLAB, change the Current Directory to your RTICANMMTutorial_Lesson10 folder.

   Tip

   Save all the files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all files to one folder also includes changing the path of the Current Directory in MATLAB.

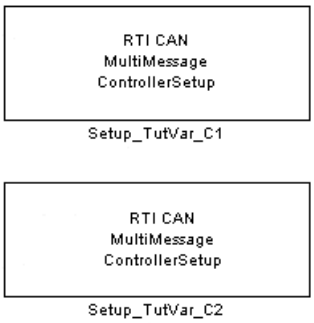
3. In the MATLAB Command Window, type **`rticanmm_tutorial`** to open the rticanmm_tutorial_lib library.
4. From the rticanmm_tutorial_lib library, double-click chassis.dbc and chassis_changed.dbc.

   Both DBC files are automatically copied to your RTICANMMTutorial_Lesson10 folder.
5. Close the rticanmm_tutorial_lib library.
6. Open a new model and save it as TutorialVariants in your RTICANMMTutorial_Lesson10 folder.
7. In the MATLAB Command Window, type **`rticanmm`** to open the rticanmmlib library.
8. From the rticanmmlib library, add
   - One RTICANMM GeneralSetup block,
   - Two RTICANMM ControllerSetup blocks, and
   - Three RTICANMM MainBlocks to your model.

   Note

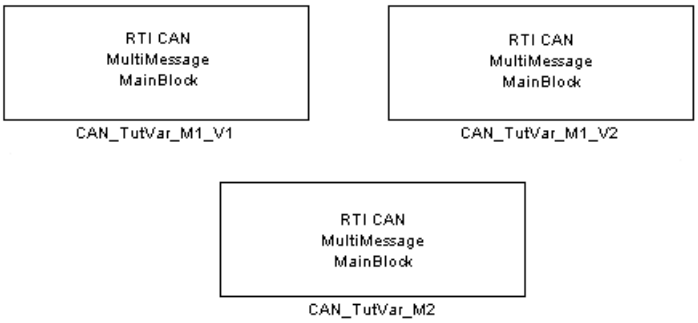
   Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

9. Rename the two RTICANMM ControllerSetup blocks as follows:
   - **`Setup_TutVar_C1`**
   - **`Setup_TutVar_C2`**



10 Rename the three RTICANMM MainBlocks as follows:

- **`CAN_TutVar_M1_V1`**
- **`CAN_TutVar_M1_V2`**
- **`CAN_TutVar_M2`**



Interim result

You built a new model and added all the required RTICANMM blocks to it. You can now parameterize the RTICANMM ControllerSetup blocks. Continue with part 2.

Part 2

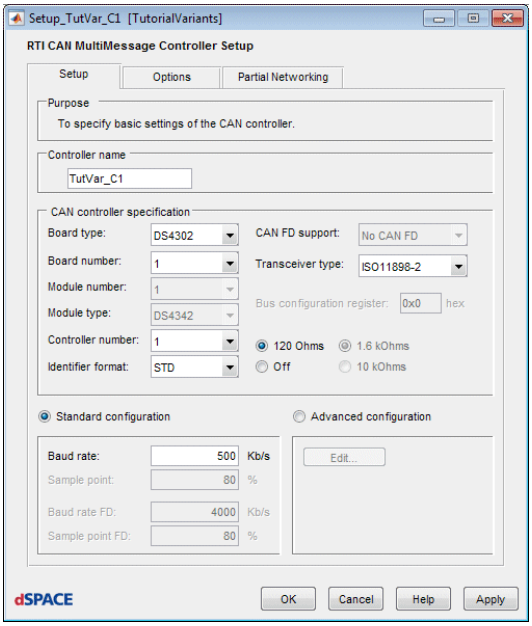
To parameterize the RTICANMM ControllerSetup blocks of your model

1 Double-click the Setup_TutVar_C1 ControllerSetup block to open it.

2 Specify **`TutVar_C1`** as Controller name.

Note

The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.



3 From the **Board type** drop-down list, select the board you are working with.

4 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.

5 Double-click the **Setup_TutVar_C2** ControllerSetup block.

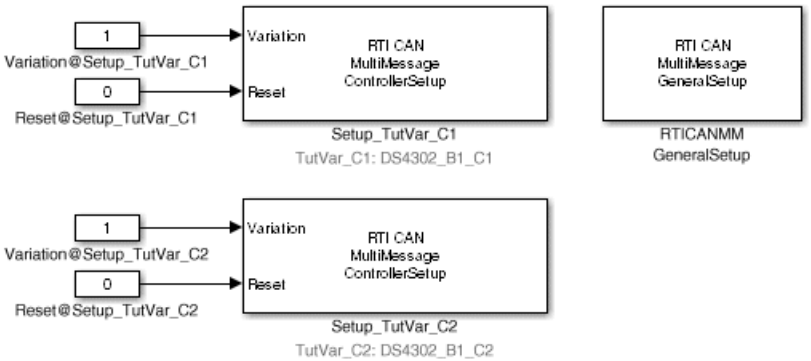
6 Specify **`TutVar_C2`** as **Controller name**.

7 From the **Board type** drop-down list, select the board you are working with.

8 From the **Controller number** drop-down list, select **2**.

9 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.



Interim result

You parameterized the RTICANMM ControllerSetup blocks. You can now specify variants of the TutVar_C1 CAN controller. Continue with part 3.

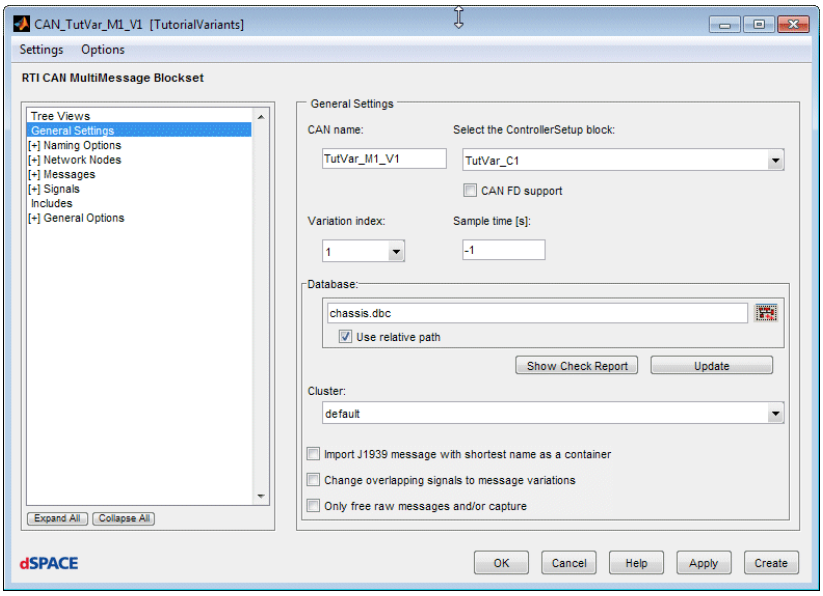
Part 3

To specify variants of the TutVar_C1 CAN controller

1 Open the CAN_TutVar_M1_V1 MainBlock.

2 On the General Settings page, specify the following settings:

Setting	Entry / Selection
CAN name	TutVar_M1_V1
Database	chassis.dbc



Since you have not changed the defaults for the ControllerSetup block and the variation index, this MainBlock configures the first variant of the TutVar_C1 CAN controller. This variant is active by default.

3 Click Create.

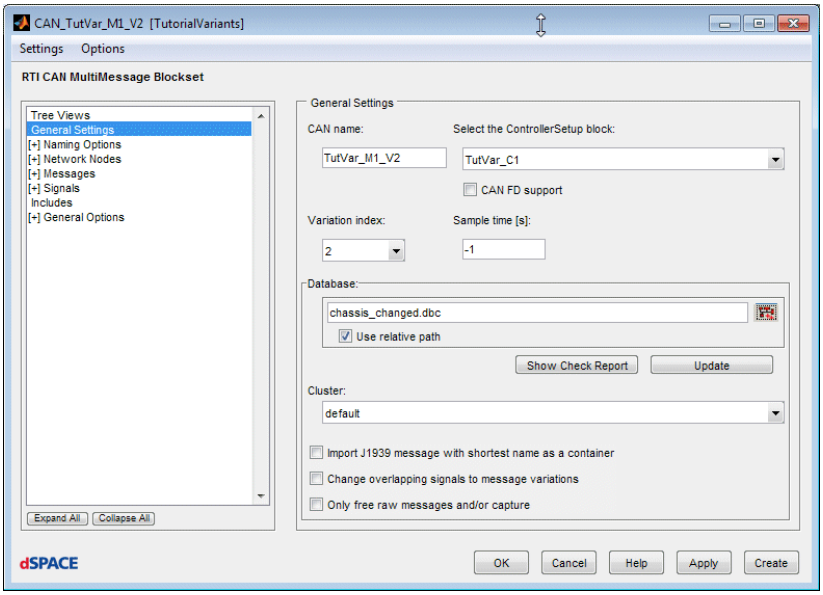
The CAN_TutVar_M1_V1 MainBlock is created. The necessary files are created automatically, and the specified settings apply to your model.

4 Click OK to close the dialog.

5 Open the CAN_TutVar_M1_V2 MainBlock.

6 Select the General Settings page, and specify the following settings:

Setting	Entry / Selection
CAN name	TutVar_M1_V2
Variation index	2
Database	chassis_changed.dbc



Since you have not changed the defaults for the ControllerSetup block but selected 2 as the variation index, this MainBlock configures the second variant of the TutVar_C1 CAN controller.

7 Click Create and close the dialog.

Interim result

You specified two variants for the TutVar_C1 CAN controller. You can now specify a free raw message for the TutVar_C2 CAN controller to observe the behavior of the CRASH_2 message. Continue with part 4.

Part 4

To configure the CAN communication of the TutVar_C2 CAN controller

1 Open the CAN_TutVar_M2 MainBlock.

2 Select the General Settings page, and specify the following settings:

Setting	Entry / Selection
CAN name	TutVar_M2
Select the ControllerSetup block	TutVar_C2

3 Select Only free raw messages and/or capture.

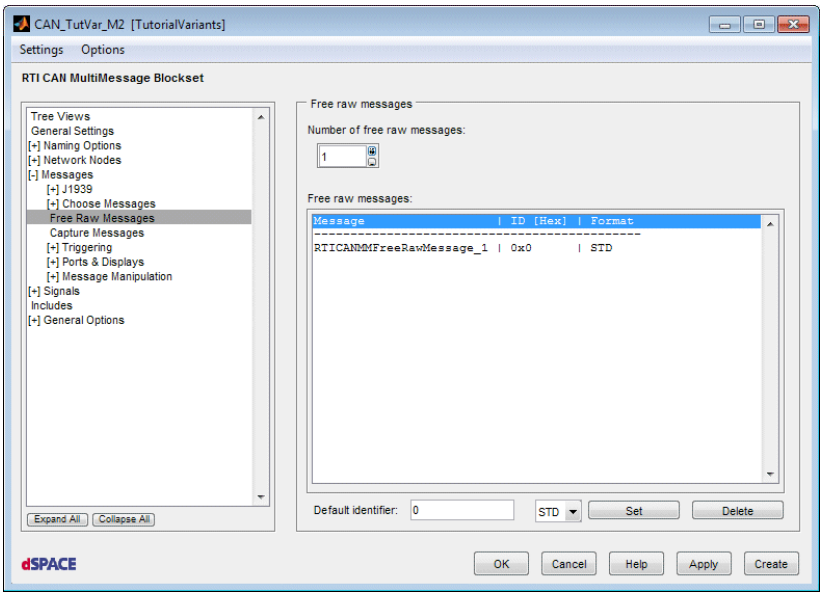
This MainBlock configures the CAN communication of the TutVar_C2 CAN controller. On this CAN controller, you will work without a database but only with free raw messages.

4 In the dialog tree, select Messages – Free Raw Messages.

On the Free Raw Messages page, you can specify messages which are independent of a database file.

5 In the Number of free raw messages edit field, type `1` and press `Enter`.

One free raw message is added to the **Free raw messages** list. The ID of the message is 0x0 and the format is STD by default.

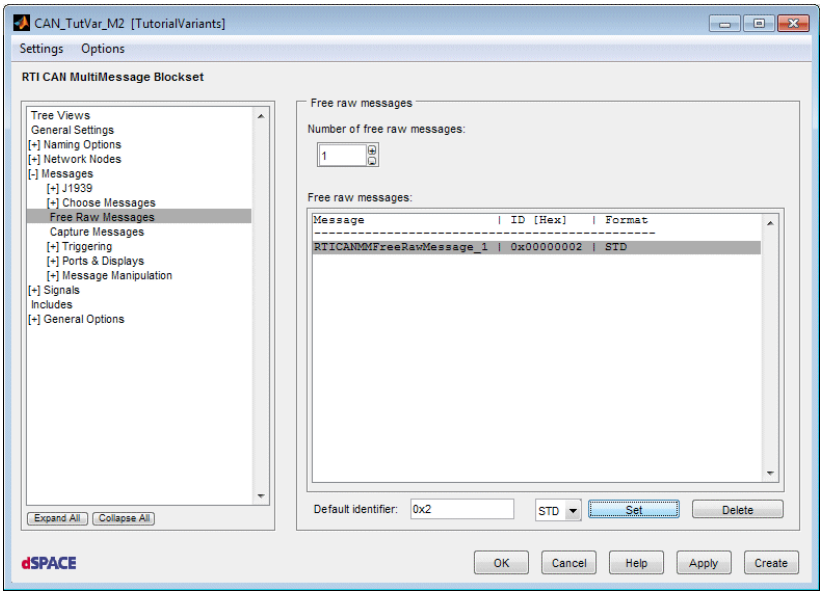


6 In the **Free raw messages** list, select the free raw message.

7 In the **Default identifier** edit field, type **0x2** and click **Set**.

Note

You always have to click **Set** to apply your settings.



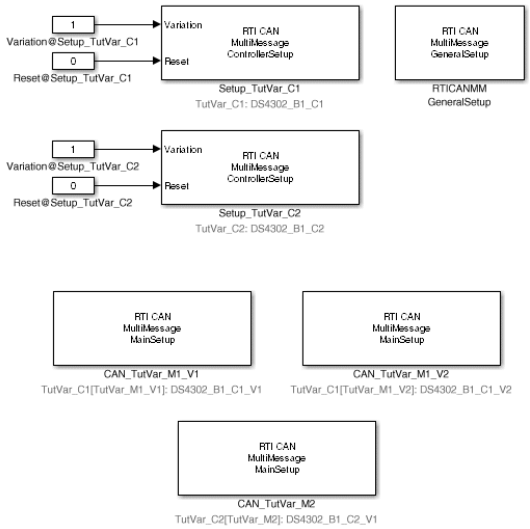
You specified a free raw message with an ID of 0x2 and the standard identifier format.

8 Click **Create** and close the dialog.

9 Save your model.

Result

You built a model with three RTICANMM MainBlocks but only two RTICANMM ControllerSetup blocks.



The CAN_TutVar_M1_V1 and the CAN_TutVar_M1_V2 MainBlocks configure the CAN communication variants of the TutVar_C1 CAN controller. You specified the CAN_TutVar_M1_V1 as the first variant and the CAN_TutVar_M1_V2 as the second variant of the controller. By default, the first variant is the active variation of the CAN controller.

Tip

You can change the default active variant via the Constant block connected to the **Variation** inport of the TutVar_C1 CAN controller.

The CAN_TutVar_M2 MainBlock configures the CAN communication of the TutVar_C2 CAN controller. This CAN controller only displays the behavior of the CRASH_2 message. You specified one free raw message with an ID of 0x2. Since this ID is in use by CRASH_2, the free raw message will receive CRASH_2 from the TutVar_C1 CAN controller.

What's next

In the next step, you can build an application for your model, change the active variant of the TutVar_C1 CAN controller, and observe the behavior of the CRASH_2 message.

Related topics

Basics

Lesson 10: Working with Variants of a CAN Controller..... 225
Overview of Lesson 10..... 225

Step 2: How to Work with Variants of a CAN Controller in ControlDesk

Objective

When you work with different variants of a CAN controller, you can change the active variant of the controller during run time.

In this step, you will build an application for your model, create an experiment in ControlDesk, change the active variant of the CAN controller, and observe the behavior of the CRASH_2 message.

Method

To work with variants of a CAN controller in ControlDesk

1. Build an application for your TutorialVariants model.
2. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
3. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

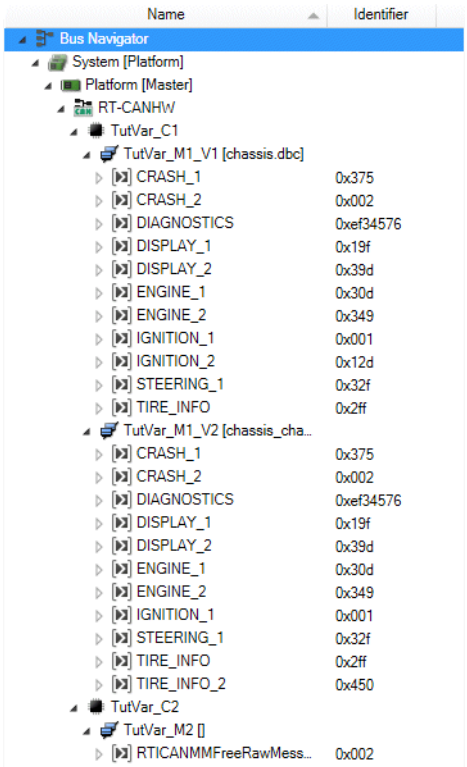
   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
4. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.
5. From the platform's context menu in the Project controlbar, select Add Variable Description.

   ControlDesk opens the Select Variable Description dialog.
6. In the Select Variable Description dialog, click Import and select the `TutorialVariants.sdf` file from your `RTICANMMTutorial_Lesson10` folder.
7. In the Select Variable Description dialog, click OK.

   ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

   In the Bus Navigator tree, TutVar_M1_V1 and TutVar_M1_V2 are displayed as the databases of the TutVar_C1 CAN controller. For the TutVar_C2 CAN controller, the specified free raw message is displayed.

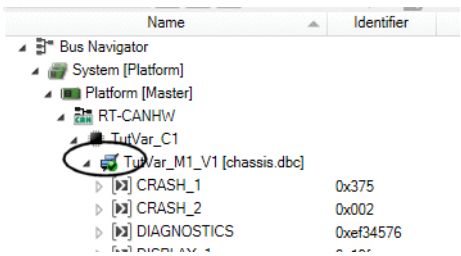


8 Generate the following layouts:

CAN Bus	Message	Layout
TutVar_M1_V1	CRASH_2	TX
TutVar_M1_V2	CRASH_2	TX
TutVar_M2	RTICANMMFreeRawMessage_1	RX

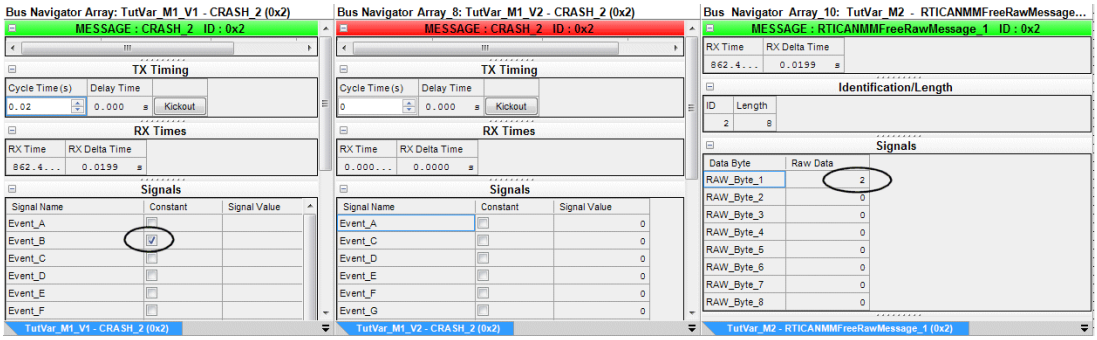
9 On the Home ribbon, click Status Control – Go Online to start online calibration.

TutVar_M1_V1 is displayed as the active database.



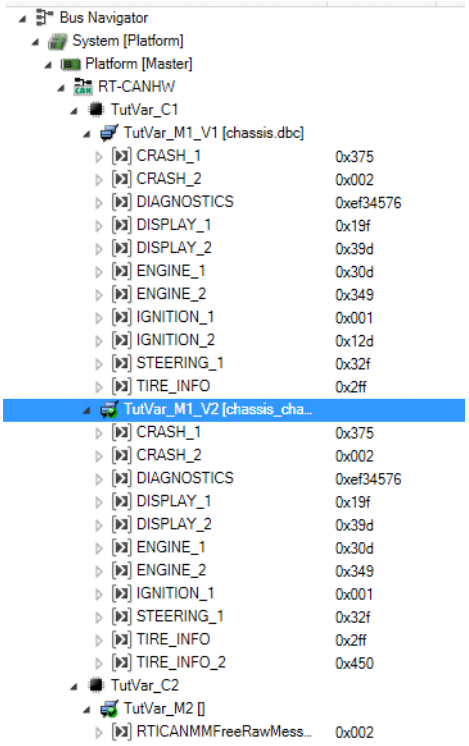
10 In the TX layout of TutVar_M1_V1, select Constant of Event_B, and in the Cycle Time(s) edit field, type 0.02 and press **Enter**.

The CAN_TutVar_M1_V1 MainBlock transmits CRASH_2 to the CAN_TutVar_M2 MainBlock. The free raw message receives Event_B on the second bit of Byte 1. The displayed raw data is the decimal equivalent of the value of Byte 1 ($2^1 = 2$). Since CAN_TutVar_M1_V2 is the inactive MainBlock, the corresponding layout is inactive.



11 Right-click the TutVar_M1_V2 node in the Bus Navigator tree and select Activate from the context menu.

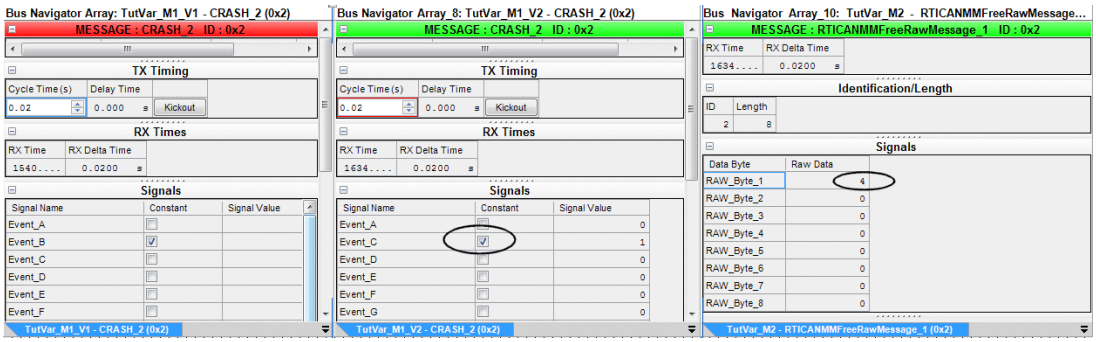
TutVar_M1_V2 is the active variant and transmits CRASH_2 via the bus. CAN_TutVar_M1_V1 is the inactive variant and no longer transmits CRASH_2.



Note

Activating a variant via the Bus Navigator deactivates the Variation inport of the corresponding RTICANMM ControllerSetup block in your model.

12 In the TX layout of TutVar_M1_V2, select Constant of Event_C, and in the Cycle Time(s) edit field, type 0.02 and press **Enter**.



Even though Event_B is not included in CRASH _2 of the chassis_changed.dbc file, the bit positions of the remaining signals have not changed. The free raw message of the CAN_TutVar_M2 MainBlock therefore receives Event_C on the third bit of Byte 1. The displayed raw data is the decimal equivalent of the value of Byte 1 ($2^2 = 4$).

13 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

14 Save the project.

Result

You worked with variants of the TutVar_C1 CAN controller in ControlDesk. You changed the active variant during run time and observed the effects on the free raw message of the TutVar_C2 CAN controller. Since you changed the variant via the Bus Navigator, you disabled the Variation inport of the TutVar_C1 block of your model.

What's next

With this step, you finished working with variants of a CAN controller. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 10: Working with Variants of a CAN Controller..... 225
Overview of Lesson 10..... 225

Result of Lesson 10

Result

In this lesson, you learned the basic principles of working with variants of a CAN controller:

- You can define up to 20 variants of one CAN controller by specifying different DBC files for the CAN controller.
- You need a separate RTICANMM MainBlock for each variant and you have to select the variation index.
- The variant with a variation index of 1 is the active variant by default. You can change the default active variant via the Constant block connected to the **Variation** inport of the CAN controller.
- You can activate and deactivate the variants of a CAN controller during run time via the model or ControlDesk. If you change the active variant via the **Bus Navigator**, you deactivate the **Variation** inport of the corresponding RTICANMM ControllerSetup block in the model.

  Tip

  If you want to change the variation of a CAN controller from within the model again, you must first reactivate the **Variation** inport of the ControllerSetup block. This is done by setting the **Variation_Switch** variable to `0`. The **Variation_Switch** variable is in the **Variables** controlbar of ControlDesk below `BusSystems/CAN/ControllerSetup_Blocks/<controller name>`.

- To observe the behavior of the variants, you can use free raw messages, etc.

  Tip

  For more information on working with free raw messages, refer to Lesson 11: Working Without a Database.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson10_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Free Raw Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to work without a database.

Related topics

Basics

Lesson 10: Working with Variants of a CAN Controller..... 225
Overview of Lesson 10..... 225

Lesson 11: Working Without a Database

Where to go from here

Information in this section

Overview of Lesson 11 241
In this lesson, you will learn how to work without a database file.

Step 1: How to Use Free Raw Messages Instead of a Database 244
In this step, you will build a model without specifying a database file. You will specify free raw messages for one of the two RTICANMM MainBlocks.

Step 2: How to Use Capture Messages 250
In this step, you will specify capture messages for one of the two RTICANMM MainBlocks.

Result of Lesson 11 256
In this lesson, you learned the basic principles of working without a database file.

Overview of Lesson 11

Introduction

When using the RTICANMM blockset, you usually work with a database file. However, you can also work without a database file. In that case, you have to specify free raw and/or capture messages to define your CAN communication. These messages are independent of a database file.

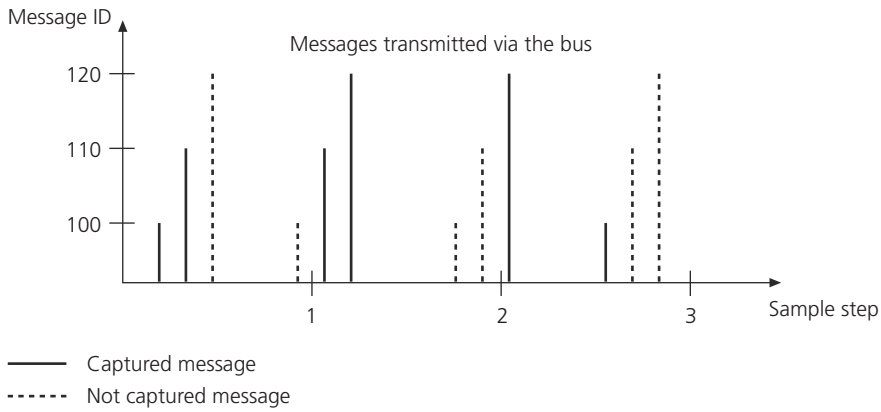
Free raw messages

Free raw messages always consist of eight 8-bit signals. You can completely modify these messages, including their identifiers and message lengths. They are TX and RX messages.

Capture messages

You can specify identifier filters for capture messages. These messages capture messages on the CAN bus whose identifiers match the specified filter values.

Behavior of capture messages Each capture message catches one message per sampling step. For example, three messages are transmitted via the bus and you specify two capture messages. Capture message 1 catches then the first message transmitted on the bus in one sampling step and capture message 2 catches the second message. Further messages transmitted in the sampling step are not captured.



Capture Message	Message Captured in ...		
	Sample Step 1	**Sample Step 2**	**Sample Step 3**
1	ID 100	ID 110	ID 120
2	ID 110	ID 120	ID 100

Since messages are captured in each sampling step, the values of the capture messages often change within milliseconds.

Tip

To ensure that all the messages of one sampling step are captured, you can use the model's baud rate and sample time to calculate the maximum number of messages available on the bus in one sampling step. This number is the number of capture messages needed to capture all the messages. For an example on calculating the maximum number of messages in one sampling step, refer to Capture Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Capture messages and network management A typical field of application for capture messages is network management, for example, as defined by the OSEK-NM standard. For example, capture messages can be used to monitor the state of the CAN bus nodes.

What you will learn	In this lesson, you will learn the basic principles of working without a database. ▪ You will build a model without specifying a database file for the RTICANMM MainBlocks. ▪ You will specify free raw messages. ▪ You will specify capture messages.
Starting point	The starting point of this lesson is a new model in MATLAB.
Prepared demo	Since you will build a new model in this lesson, no prepared demo is required.
Before you begin	Before you begin this lesson, you have to be familiar with building a model using RTICANMM blocks (refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39).
Steps	This lesson contains the following steps: ▪ Step 1: How to Use Free Raw Messages Instead of a Database on page 244 ▪ Step 2: How to Use Capture Messages on page 250
Result and further information	To check the correctness of the executed steps and for more information, refer to Result of Lesson 11 on page 256.
Duration	Working through this lesson will take you about 30 minutes.
What's next	Now you can start to build a model without specifying a database file for the RTICANMM MainBlocks, see Step 1: How to Use Free Raw Messages Instead of a Database on page 244.
Related topics	Basics Lesson 11: Working Without a Database 241

Step 1: How to Use Free Raw Messages Instead of a Database

Objective

When you work without a database, you have to specify at least one free raw message or capture message. In this step, you will build a model and specify free raw messages for one of the two RTICANMM MainBlocks.

Part 1

To create a model and add RTICANMM blocks to it

1. In your working folder, create a new folder named RTICANMMTutorial_Lesson11.
2. In MATLAB, change the Current Directory to your RTICANMMTutorial_Lesson11 folder.

> **Tip**
>
> Save all the files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all files to one folder also includes changing the path of the Current Directory in MATLAB.

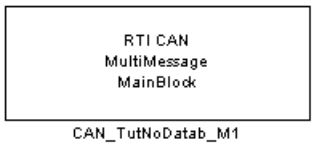
3. In MATLAB, open a new model and save it as TutorialNoDatabase in your RTICANMMTutorial_Lesson11 folder.
4. In the MATLAB Command Window, type **`rticanmm`** to open the rticanmmlib library.
5. From the rticanmmlib library, add
   - One RTICANMM GeneralSetup block,
   - Two RTICANMM ControllerSetup blocks, and
   - Two RTICANMM MainBlocks to your model.

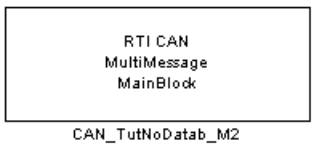
> **Note**
>
> Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

6. Rename the two RTICANMM ControllerSetup blocks as follows:
   - **`Setup_TutNoDatab_C1`**
   - **`Setup_TutNoDatab_C2`**

7 Rename the two RTICANMM MainBlocks as follows:

- **`CAN_TutNoDatab_M1`**
- **`CAN_TutNoDatab_M2`**





Interim result

You built a new model and added all the required RTICANMM blocks to it. You can now parameterize the RTICANMM ControllerSetup blocks. Continue with part 2.

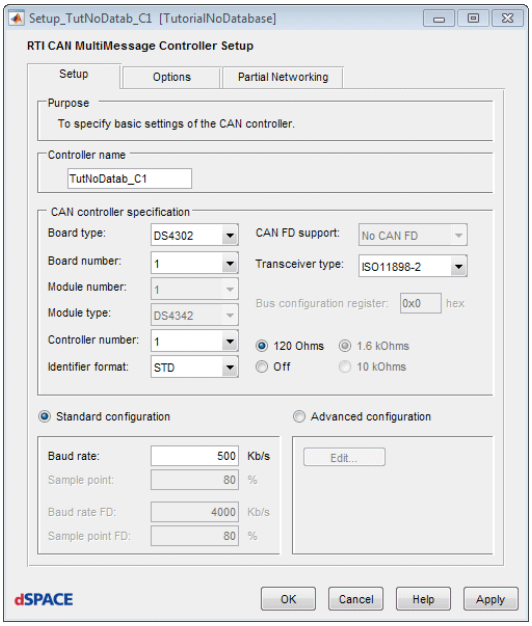
Part 2

To parameterize the RTICANMM ControllerSetup blocks of your model

1 Double-click the Setup_TutNoDatab_C1 ControllerSetup block to open it.

2 Specify **`TutNoDatab_C1`** as Controller name.

> **Note**
>
> The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.



3 From the **Board type** drop-down list, select the board you are working with.

4 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.

5 Double-click the **Setup_TutNoDatab_C2** ControllerSetup block.

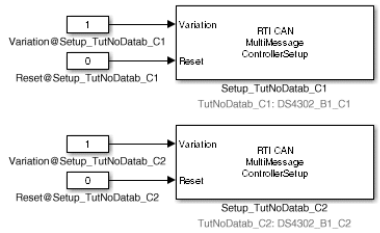
6 Specify **`TutNoDatab_C2`** as **Controller name**.

7 From the **Board type** drop-down list, select the board you are working with.

8 From the **Controller number** drop-down list, select **2**.

9 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.



Interim result

You parameterized the RTICANMM ControllerSetup blocks. You can now enable working without a database file. Continue with part 3.

Part 3

To enable working without a database file

1 Open the CAN_TutNoDatab_M1 MainBlock.

2 On the General Settings page, specify TutNoDatab_M1 as the CAN name.

3 Select Only free raw messages and/or capture.

This option enables you to work without a database. You have to specify at least one free raw message or capture message before you can create this MainBlock.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You enabled working without a database file for the CAN_TutNoDatab_M1 MainBlock. You can now specify free raw messages for it. Continue with part 4.

Part 4

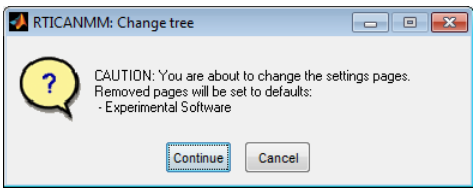
To use free raw messages instead of a database file

1 Select the Tree Views page and select Filter Configuration Tree by Feature.

The predefined filters are enabled and the dialog tree is extended according to the selected filters.

2 Clear all filters except for Model Interface and Manipulation Interface.

If a dialog similar to the following one is displayed, click Continue.

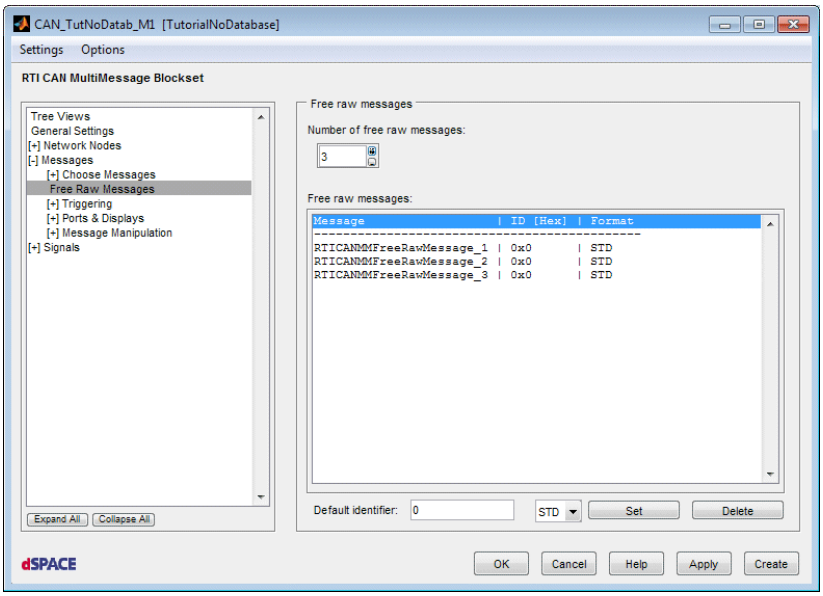


The dialog tree is reduced to the pages available for the Manipulation Interface filter.

3 In the dialog tree, select Messages – Free Raw Messages.

The Free Raw Messages page lets you define the number of free raw messages you want to use in your model. You can also specify the IDs of the messages and select the format for the messages.

4 In the Number of free raw messages edit field, type **3** and press **Enter**.



Three free raw messages are added to the list. The ID of all the messages is 0x0 by default.

5 In the Free raw messages list, select RTICANMMFreeRawMessage_1.

6 In the Default identifier edit field, type **0x100**.

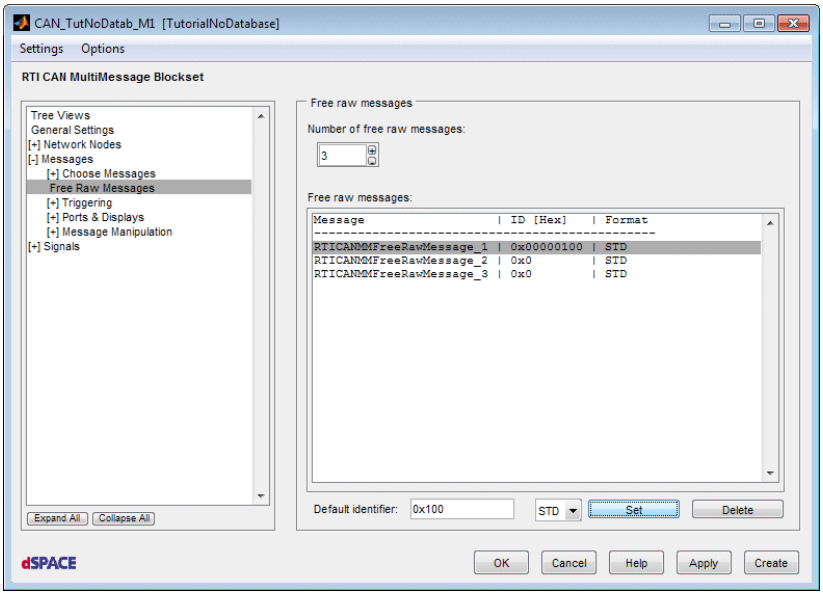
7 Ensure that STD is selected from the format drop-down list.

STD stands for the standard CAN format for classic CAN messages. 8 raw bytes will be generated for the free raw message.

8 Click Set.

Note

You always have to click Set to apply your settings.



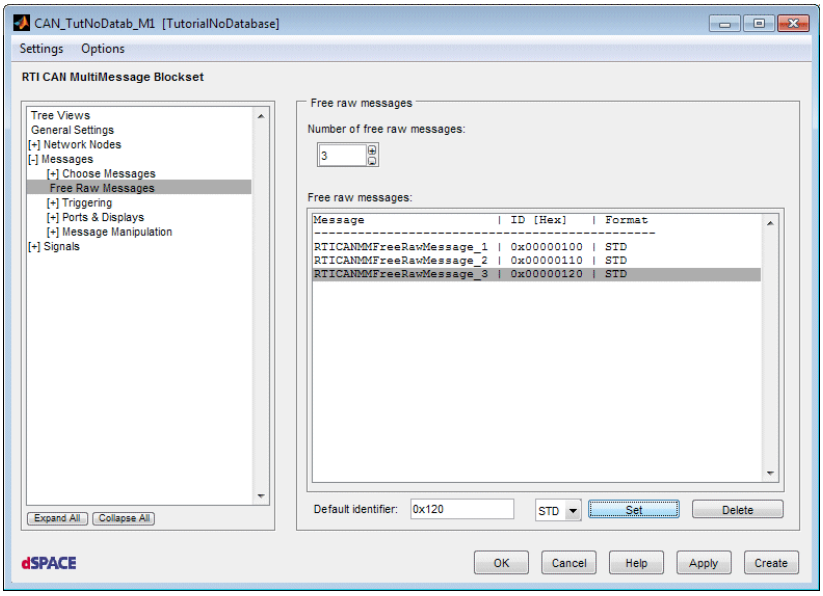
The default ID of the first free raw message changed from 0x0 to 0x100.

Note

The message's ID has to be unique for a model, so you always have to change the default ID. Make sure that no other message uses the specified ID at the same time.

9 For RTICANMMFreeRawMessage_2, specify a default ID of 0x110.

10 For RTICANMMFreeRawMessage_3, specify a default ID of 0x120.



11 In the dialog tree, select Messages – Message Manipulation – TX ID.

You enabled the TX ID page by defining free raw messages. You can manipulate the ID of the three free raw messages via ControlDesk by default. You do not have to change the default settings for this lesson.

12 In the dialog tree, select **Messages – Message Manipulation – TX Message Length**.

You enabled the TX Message Length page by defining free raw messages. The default message length of all the free raw messages is 8 byte. You can manipulate the message length via ControlDesk. You do not have to change the default settings for this lesson.

13 Click **Create**.

The necessary files are created automatically, and the specified settings apply to your model.

14 Click **OK** to close the **CAN_TutNoDatab_M1 [TutorialNoDatabase]** dialog.

Result

You built a model with two RTICANMM MainBlocks. You enabled working without a database file and created three free raw messages for the CAN_TutNoDatab_M1 MainBlock. You specified a default ID for each free raw message. Specifying free raw messages enables the TX ID and TX Message Length pages. By default, you can change the default ID and message length for each free raw message via ControlDesk.

What's next

In the next step, you can enable working without a database for the CAN_TutNoDatab_M2 MainBlock and specify capture messages.

Related topics

Basics

Lesson 11: Working Without a Database 241
Overview of Lesson 11 241

Step 2: How to Use Capture Messages

Objective

To work without a database file, you have to specify at least one free raw or capture message.

In this step, you will specify capture messages for the CAN_TutNoDatab_M2 MainBlock. Additionally, you will specify free raw messages to let both CAN controllers of your model communicate with each other.

Part 1

To enable working without a database

1 Open the CAN_TutNoDatab_M2 MainBlock.

2 On the General Settings page, specify the following settings:

Setting	Entry / Selection
CAN name	TutNoDatab_M2
Selected ControllerSetup block	TutNoDatab_C2

3 Select Only free raw messages and/or capture.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You enabled working without a database file for the CAN_TutNoDatab_M2 MainBlock. You can now specify capture messages for it. Continue with part 2.

Part 2

To use capture messages

1 Select the Tree Views page and select Filter Configuration Tree by Feature.

2 Clear all filters except for Model Interface and Extras.

3 In the dialog tree, select Messages – Capture Messages.

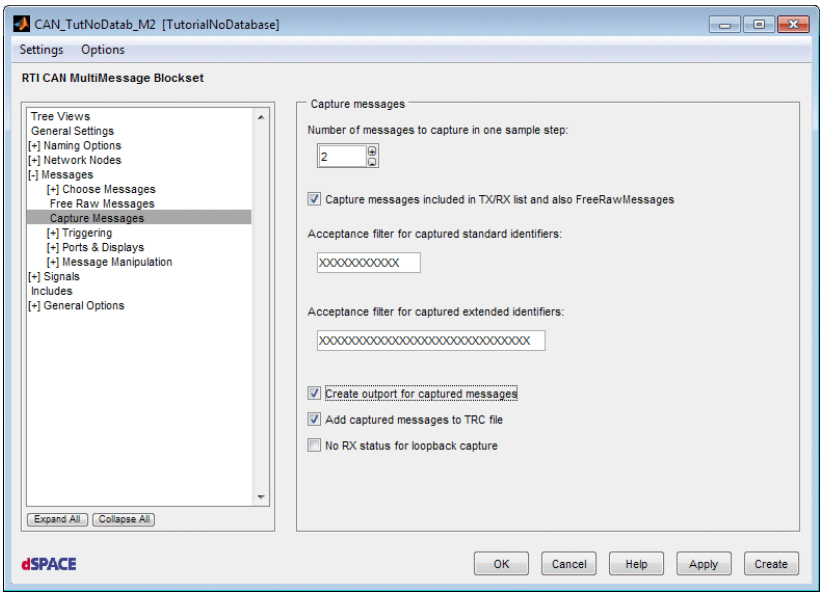
The Capture Messages page lets you define the number of messages to be captured in one sampling step. Additionally, you can specify filters for capturing only messages whose IDs match the filter settings.

4 In the Number of messages to capture in one sample step edit field, type 2.

5 Select Capture messages included in TX/RX list and also FreeRawMessages.

Selecting this option lets you capture all the RX, TX and free raw messages specified for this MainBlock.

6 Select Create outport for captured messages.



This option lets an RX Data outport be created for the MainBlock. The messages captured by the capture messages of this MainBlock are available via this outport.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

7 Select the Free Raw Messages page.

8 In the Number of free raw messages edit field, type **2** and press `Enter`.

Two free raw messages are added to the list. The default ID of both messages is 0x0.

9 In the Free raw messages list, select RTICANMMFreeRawMessage_1.

10 In the Default identifier edit field, type **`0x100`**.

11 Click Set.

The default ID of the first free raw message is changed from 0x0 to 0x100.

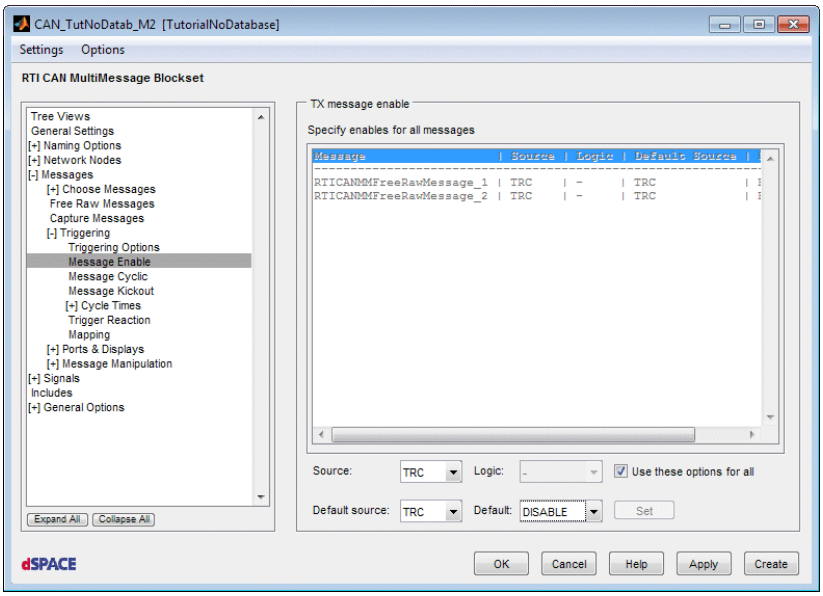
12 Select RTICANMMFreeRawMessage_2 and specify an ID of 0x110.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

13 In the dialog tree, select Messages – Triggering – Message Enable.

14 From the **Default** drop-down list, select **DISABLE**.



By default, the specified free raw messages of this MainBlock are not transmitted. Since the **Specify enables for all messages** list is disabled, it is not updated.

Note

If you encounter problems in modifying the **Default** setting because the dialog settings on the Message Enable page are grayed out, perform the following steps:

1. Select the **General Settings** page and clear the **Only free raw messages and/or capture** checkbox.
2. Change to the **Messages – Triggering – Message Enable** page and select **DISABLE** from the **Default** drop-down list.
3. Select the **General Settings** page and select the **Only free raw messages and/or capture** checkbox again.

15 Click **Create**.

The necessary files are created automatically, and the specified settings apply to your model.

16 Click **OK** to close the **CAN_TutNoDatab_M2 [TutorialNoDatabase]** dialog.

Interim result

You specified capture messages and free raw messages for the CAN_TutNoDatab_M2 MainBlock. For the capture messages, you enabled capturing the free raw messages of this MainBlock. Since the specified free raw messages have the same IDs as the messages specified for the CAN_TutNoDatab_M1 MainBlock, both CAN controllers communicate with each other. You can now check the effects of your specified settings in your model. Continue with part 3.

Part 3

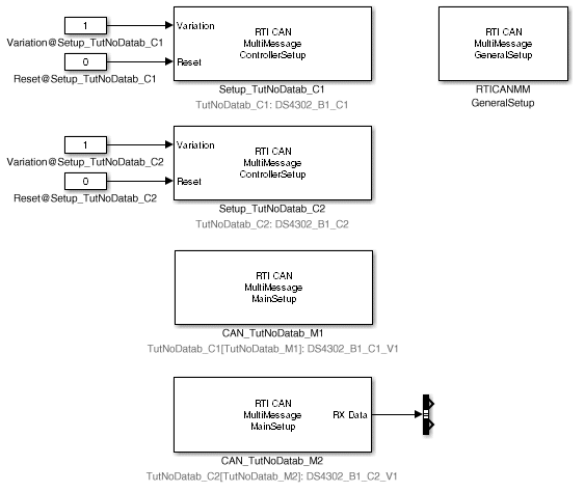
To check your specified settings

1 On the Simulation ribbon, click Library – Library Browser.
The Simulink Library Browser opens.

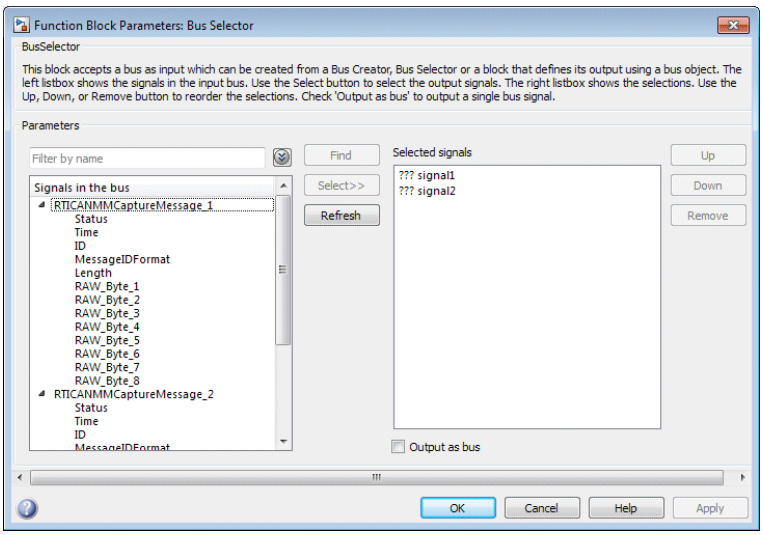
2 In the Simulink Library Browser, select the Commonly Used Blocks.

3 From the Commonly Used Blocks, drag a Bus Selector block to your model.

4 Connect the Bus Selector to the RX Data outport of the CAN_TutNoDatab_M2 MainBlock.



5 Double-click the Bus Selector to open it.



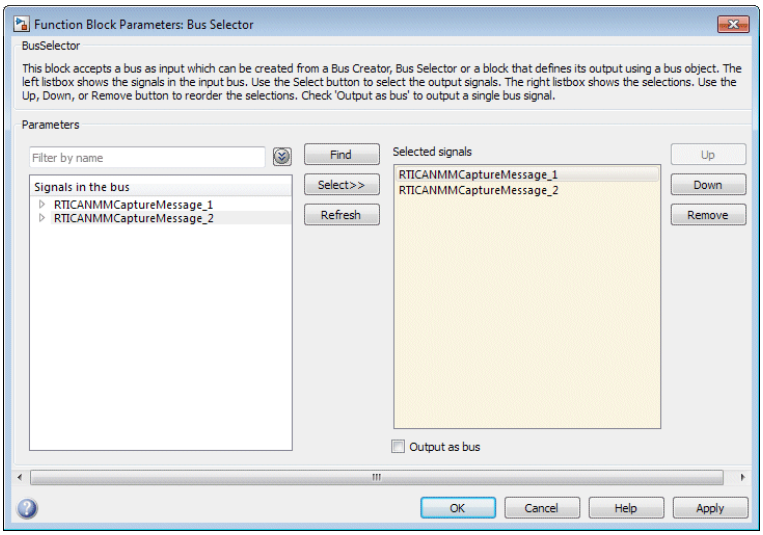
The Signals in the bus list displays the specified capture messages and their signals.

6 Delete the signals displayed in the Selected signals list.

7 Select RTICANMMCaptureMessage_1 and click Select.
RTICANMMCaptureMessage_1 is added to the Selected signals list.

8 Select **RTICANMMCaptureMessage_2** and click **Select**.

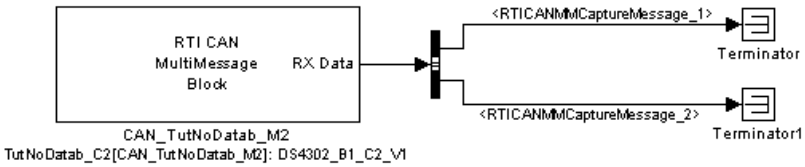
RTICANMMCaptureMessage_2 is added to the Selected signals list.



9 Click **OK** to close the Bus Selector.

10 From the **Commonly Used Blocks**, drag two **Terminator** blocks to your model.

11 Connect the Terminator blocks with the outports of the Bus Selector.



12 Save your model.

Result

You specified capture messages for the CAN_TutNoDatab_M2 MainBlock and let an RX Data outport be created for captured messages. Since you have not changed the default values of the identifier filters, all the messages available on the bus will be captured.

Additionally, you specified two free raw messages with IDs of 0x100 and 0x110 to let both CAN controllers of your model communicate with each other. To prevent this MainBlock from transmitting the specified free raw messages, you disabled their transmission on the Message Enable page.

By adding a Bus Selector to your model, you checked the effects of your settings.

What's next

With this step, you finished working without specifying a database file for your model. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 11: Working Without a Database 241
Overview of Lesson 11 241

Result of Lesson 11

Result

In this lesson, you have learned the basic principles of working without a database file:

- For the RTICANMM MainBlocks, you have to specify to work with free raw messages and/or capture messages only.
- For each RTICANMM MainBlock, you have to specify at least one free raw or capture message.
- Free raw messages always consist of eight 8-bit signals by default. They can be TX and RX messages.
- You can manipulate free raw messages in the same way you manipulate messages specified in a database file.
- Message IDs must be unique, so you always have to change the default free raw message ID.
- When you specify free raw messages, the TX ID and TX Message Length pages are enabled and default settings are specified. You do not have to change these default settings.
- Capture messages let you capture all the messages that match a specified filter. You can specify free raw messages to be captured.
- Using a Simulink Bus Selector block, you can visualize your specified settings in your model.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson11_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Free Raw Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Capture Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- TX ID Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)

- TX Message Length Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

You can now continue with the next lesson to build a model for gatewaying.

If you do not need to gateway messages, you can also continue with another lesson. Refer to Overview of Lessons on page 35.

Related topics

Basics

Lesson 11: Working Without a Database 241
Overview of Lesson 11 241

Lesson 12 (Advanced): Gatewaying Messages

Where to go from here

Information in this section

Overview of Lesson 12 260
In this lesson, you will learn the basic principles of exchanging messages between different CAN buses via gateways. You can gateway messages via the Simulink model using two RTICANMM MainBlocks or using an RTICANMM Gateway block.

Step 1: How to Build a Model for Gatewaying 263
In this step, you will build a model for gatewaying messages between two CAN bus systems.

Step 2: How to Simulate "Real" ECUs for Gatewaying 270
In this step, you will simulate the "real" ECUs of both CAN buses for gatewaying.

Step 3: How to Build a Unidirectional Gateway Using Two RTICANMM MainBlocks 273
In this step, you will build a unidirectional gateway using two RTICANMM MainBlocks. Additionally, you will manipulate the triggering conditions of the messages for one CAN bus.

Step 4: How to Build a Bidirectional Gateway Using Two RTICANMM MainBlocks 287
In this step, you will extend your model and build a bidirectional gateway.

Step 5: How to Gateway Messages Using the RTICANMM Gateway Block 296
In this step, you will learn how to gateway messages via an RTICANMM Gateway block and exclude those messages from being gatewayed by this block which need to be manipulated before gatewaying.

Step 6: How to Check the Gateway Behavior via ControlDesk 304
In this step, you will learn how to check the behavior of gatewayed messages and manipulate some of them.

Result of Lesson 12 311
In this lesson, you learned the basic principles of exchanging messages between different CAN buses via gateways.

Overview of Lesson 12

Introduction

Messages can be transmitted between different ECUs on one CAN bus. Messages can also be exchanged between different CAN buses via gateways. Using the RTICANMM blockset, you can gateway CAN messages in two ways:

Gateway via two RTICANMM MainBlocks Gatewaying messages using two RTICANMM MainBlocks allows you to manipulate the messages before gatewaying.

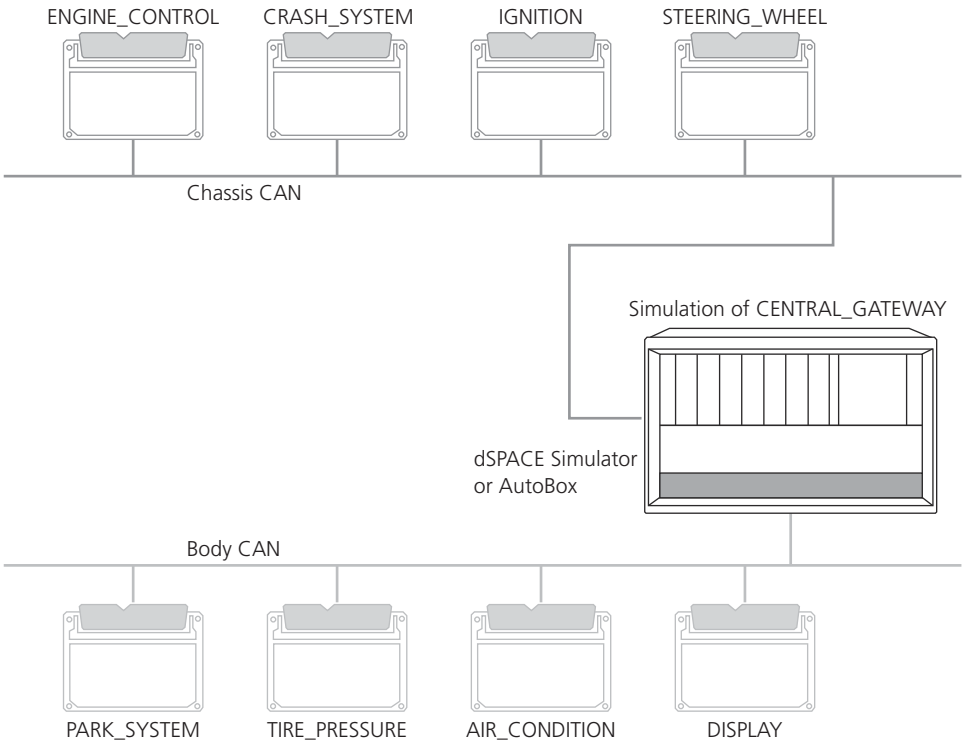
> **Note**
>
> Gatewaying messages via two RTICANMM MainBlocks reduces performance, so you should gateway as few messages as possible in this way.

Gateway via the RTICANMM Gateway block Using the RTICANMM Gateway block, you can gateway a large number of messages between two RTICANMM MainBlocks or between two RTICANMM ControllerSetup blocks. Messages gatewayed by the Gateway block cannot be manipulated before gatewaying. For optimum performance, you should gateway as many messages as possible via the Gateway block.

> **Tip**
>
> If you do not need to gateway messages, you can skip this lesson.

Gateway task in this lesson

In this lesson, you will exchange messages between the Chassis CAN and the Body CAN buses.



You will practice both gateway methods: Via two MainBlocks, you will exchange messages to be manipulated before gatewaying. All the other messages will be gatewayed via a Gateway block.

In connection with your gateway task, you will perform a restbus simulation as follows:

- *Simulating CENTRAL_GATEWAY*: You will simulate the CENTRAL_GATEWAY ECU and gateway messages between the Chassis CAN bus and the Body CAN bus. The messages of the CENTRAL_GATEWAY ECU can be manipulated before gatewaying.
- *Simulating "real" ECUs*: Since you are working without physical ECUs in this tutorial, you will simulate the "real" ECUs of both CAN bus systems. "Real" ECUs are all the ECUs displayed in the illustration above except for CENTRAL_GATEWAY. None of the messages of the "real" ECUs will be manipulated before gatewaying.

Tip

For more information on restbus simulation, see Lesson 6: Performing a CAN Restbus Simulation on page 135.

What you will learn

In this lesson, you will learn the basic principles of gatewaying messages.

- You will build a model to gateway messages between two CAN buses.
- You will gateway messages via two RTICANMM MainBlocks and via an RTICANMM Gateway block.
- You will build a unidirectional gateway: from Chassis CAN to Body CAN.
- You will build a bidirectional gateway: from Chassis CAN to Body CAN and vice versa.
- You will check the gateway behavior via ControlDesk.

Starting point

The starting point of this lesson is a new model in MATLAB.

Prepared demo

Since you will build a new model in this lesson, no prepared demo is required.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- You need four CAN controllers to work with the application in ControlDesk.
- The CAN channels of your dSPACE real-time hardware are physically connected as follows:
  - Channel 1 is connected to channel 2
  - Channel 3 is connected to channel 4
- You have to be familiar with building a model using RTICANMM blocks (refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39).
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Build a Model for Gatewaying on page 263
- Step 2: How to Simulate "Real" ECUs for Gatewaying on page 270
- Step 3: How to Build a Unidirectional Gateway Using Two RTICANMM MainBlocks on page 273
- Step 4: How to Build a Bidirectional Gateway Using Two RTICANMM MainBlocks on page 287
- Step 5: How to Gateway Messages Using the RTICANMM Gateway Block on page 296
- Step 6: How to Check the Gateway Behavior via ControlDesk on page 304

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 12 on page 311.

Duration

Working through this lesson will take you about 140 minutes.

What's next

Now you can start to build a model for gatewaying, see Step 1: How to Build a Model for Gatewaying on page 263.

Related topics

Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259

Step 1: How to Build a Model for Gatewaying

Objective

For gatewaying messages between two CAN bus systems, you have to build a model using four CAN controllers. You need a CAN controller for:

- Simulating all the "real" ECUs of the Chassis CAN bus.
- Simulating all the "real" ECUs of the Body CAN bus.
- Simulating CENTRAL_GATEWAY for the Chassis CAN bus.
- Simulating CENTRAL_GATEWAY for the Body CAN bus.

In this step, you will build a new model which uses four CAN controllers.

Part 1

To create a gateway model and add RTICANMM blocks to it

1. In your working folder, create a new folder named RTICANMMTutorial_Lesson12.
2. In MATLAB, change the Current Directory to your RTICANMMTutorial_Lesson12 folder.

   Tip

   Save all files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all files to one folder also includes changing the path of the Current Directory in MATLAB.

3. In the MATLAB Command Window, type **`rticanmm_tutorial`** to open the rticanmm_tutorial_lib library.

4 From the rticanmm_tutorial_lib library, double-click the body.dbc and chassis.dbc files.

   Both files are automatically copied to your RTICANMMTutorial_Lesson12 folder.

5 Close the rticanmm_tutorial_lib library.

6 Open a new model and save it as TutorialGateway in your RTICANMMTutorial_Lesson12 folder.

7 In the MATLAB Command Window, type **rticanmm** to open the rticanmmlib library.

8 From the rticanmmlib library, add
   - One RTICANMM GeneralSetup block,
   - Four RTICANMM ControllerSetup blocks, and
   - Four RTICANMM MainBlocks to your model.

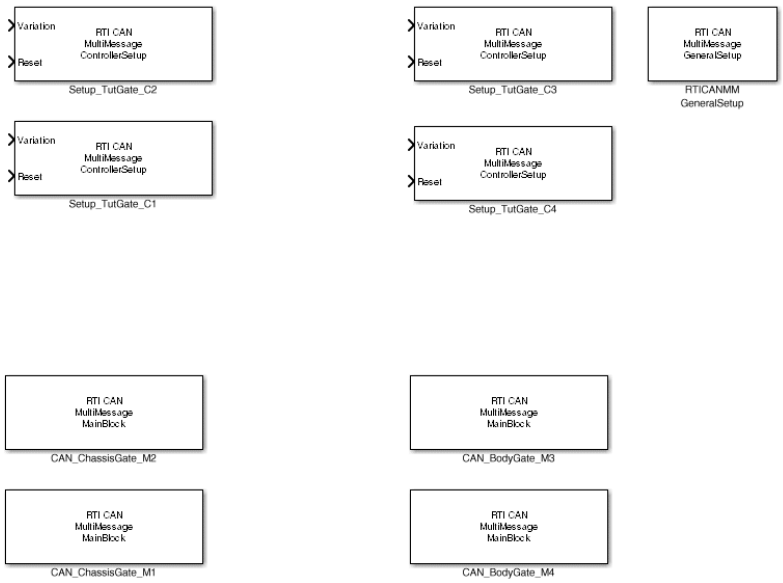
   Note

   Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

9 Rename the four RTICANMM ControllerSetup blocks as follows:
   - **Setup_TutGate_C1**
   - **Setup_TutGate_C2**
   - **Setup_TutGate_C3**
   - **Setup_TutGate_C4**

10 Rename the four RTICANMM MainBlocks as follows:
   - **CAN_ChassisGate_M1**
   - **CAN_ChassisGate_M2**
   - **CAN_BodyGate_M3**
   - **CAN_BodyGate_M4**

11 For easier model handling, arrange the RTICANMM blocks as shown in the illustration below.



Interim result

You built a new model and added one RTICANMM GeneralSetup block, four RTICANMM ControllerSetup blocks, and four RTICANMM MainBlocks to your model. Additionally, you renamed the RTICANMM ControllerSetup blocks and the RTICANMM MainBlocks. You can now parameterize the RTICANMM ControllerSetup blocks. Continue with part 2.

Part 2

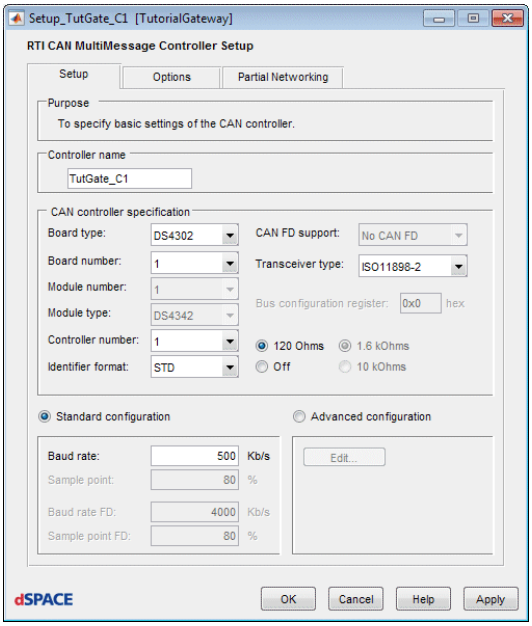
To parameterize the RTICANMM ControllerSetup blocks of your model

1 Double-click the Setup_TutGate_C1 ControllerSetup block to open it.

2 Specify **`TutGate_C1`** as the Controller name.

Note

The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.



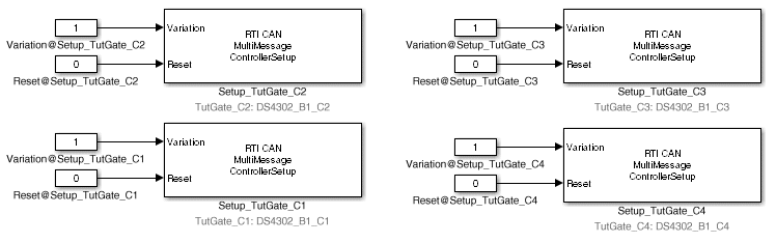
3 From the **Board type** drop-down list, select the board you are working with.

4 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup blocks are connected to Constant blocks.

5 Change the settings of the other RTICANMM ControllerSetup blocks as follows:

RTICANMM ControllerSetup Block	Controller Name	Board Type	Controller Number
Setup_TutGate_C2	TutGate_C2	Select the board type according to the hardware you are working with.	2
Setup_TutGate_C3	TutGate_C3		3
Setup_TutGate_C4	TutGate_C4		4



Interim result

You parameterized all the RTICANMM ControllerSetup blocks of your model. You can now parameterize your RTICANMM MainBlocks. Continue with part 3.

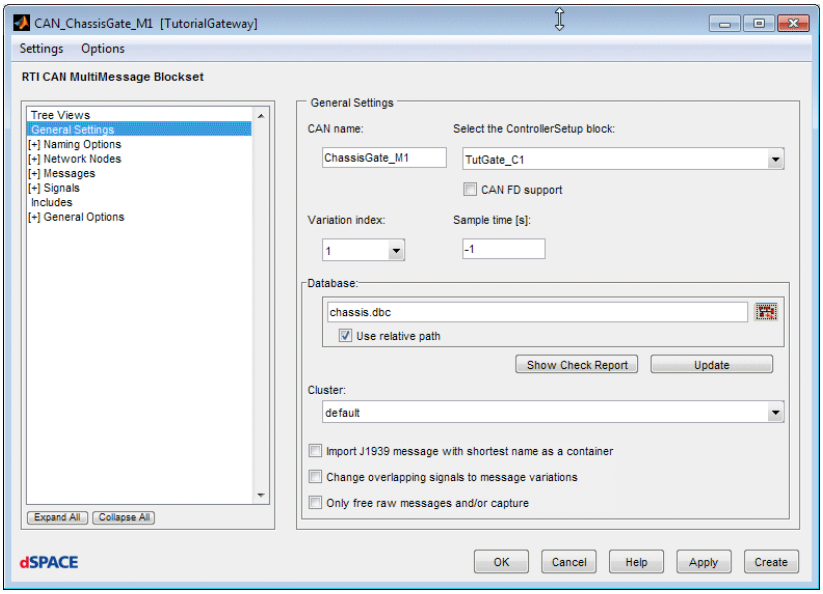
Part 3

To parameterize the RTICANMM MainBlocks of your model

1 Double-click the CAN_ChassisGate_M1 MainBlock to open it.

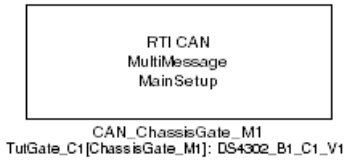
2 Select the General Settings page, and specify the following settings:

Setting	Entry / Selection
CAN name	ChassisGate_M1
Select the ControllerSetup block	TutGate_C1
Database	chassis.dbc



3 Click Create and close the dialog.

The CAN_ChassisGate_M1 MainBlock is created. All the files needed for your RTICANMM communication, such as an S-function, are created automatically.

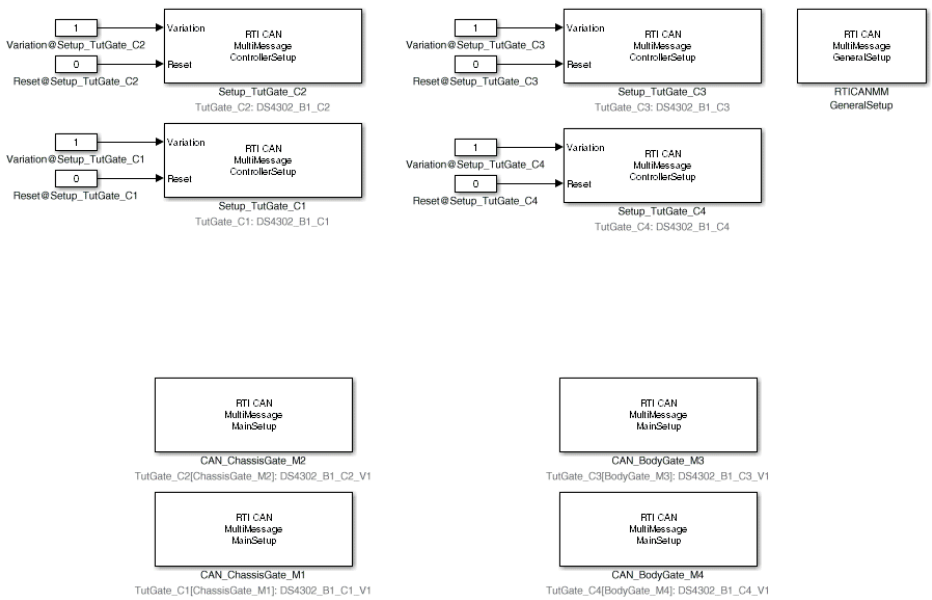


4 Specify the general settings of the other RTICANMM MainBlocks as follows:

RTICANMM MainBlock	CAN Name	Selected ControllerSetup Block	Database
CAN_ChassisGate_M2	ChassisGate_M2	TutGate_C2	chassis.dbc
CAN_BodyGate_M3	BodyGate_M3	TutGate_C3	body.dbc
CAN_BodyGate_M4	BodyGate_M4	TutGate_C4	body.dbc

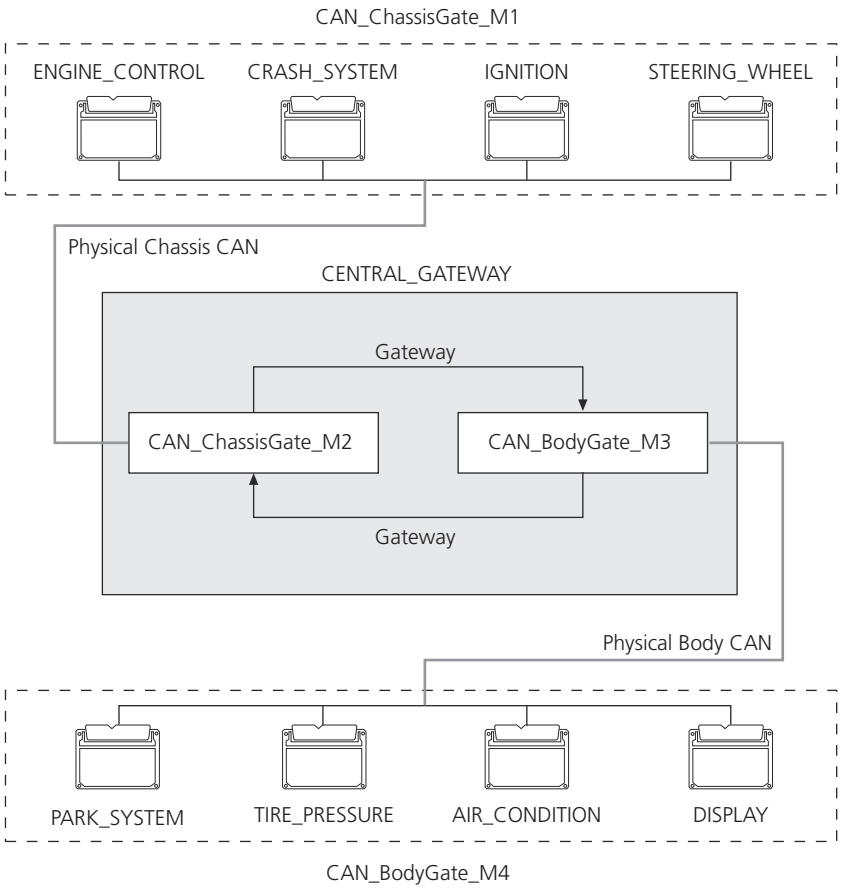
5 Save your model.

Result



You built a model with four CAN controllers to gateway messages between two CAN bus systems. Additionally, you parameterized the RTICANMM ControllerSetup blocks and RTICANMM MainBlocks of your model.

The four MainBlocks will be needed to simulate the "real" ECUs of both CAN buses and to simulate the CENTRAL_GATEWAY ECU. The CENTRAL_GATEWAY ECU will be used to gateway messages between the Chassis CAN bus and the Body CAN bus.



To provide the physical Chassis CAN bus, CAN channels 1 and 2 of your real-time hardware must be connected physically. To provide the physical Body CAN bus, CAN channels 3 and 4 must be connected physically.

What's next

In the next step, you can simulate the "real" ECUs of both CAN buses for gatewaying.

Related topics

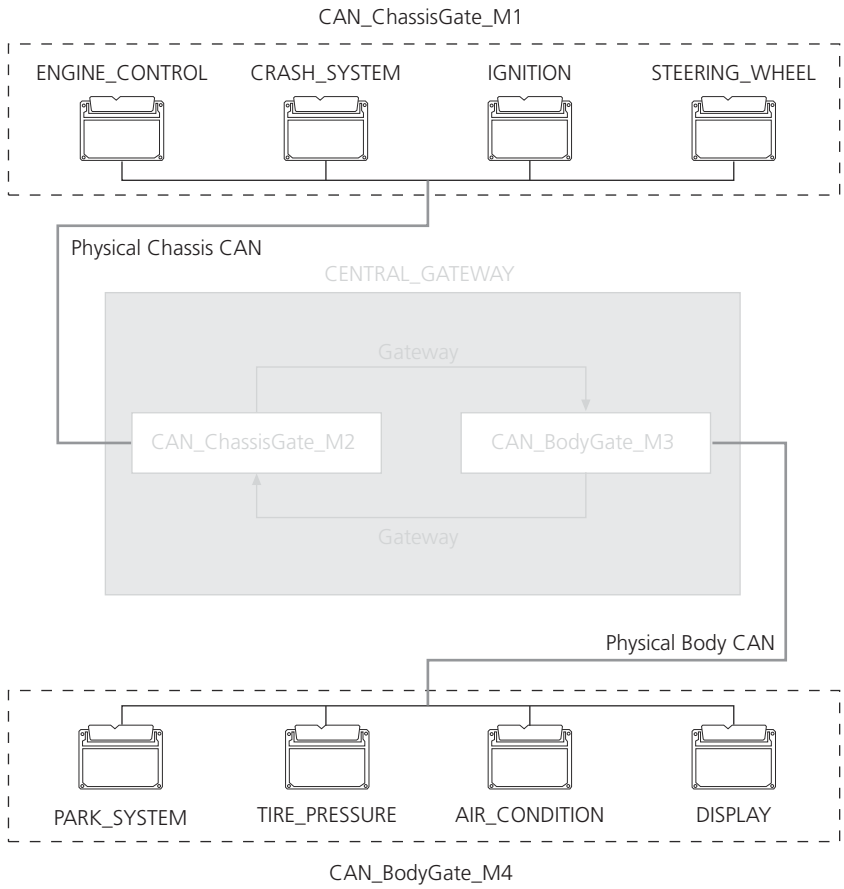
Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Step 2: How to Simulate "Real" ECUs for Gatewaying

Objective

In this tutorial, you are working without physical ECUs, so you have to simulate the "real" ECUs of the Chassis and Body CAN bus. The communication of these ECUs will be performed on physical CAN buses.



In this step, you will simulate all the ECUs of both CAN buses except for the CENTRAL_GATEWAY ECU.

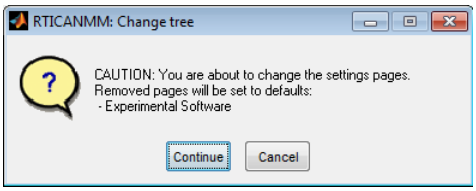
Part 1

To simulate the "real" ECUs of the Chassis CAN bus

1. Open the CAN_ChassisGate_M1 MainBlock.
2. On the Tree Views page, select Filter Configuration Tree by Feature.

3 Clear all filters except for **Model Interface**.

If a dialog similar to the following one is displayed, click **Continue**.



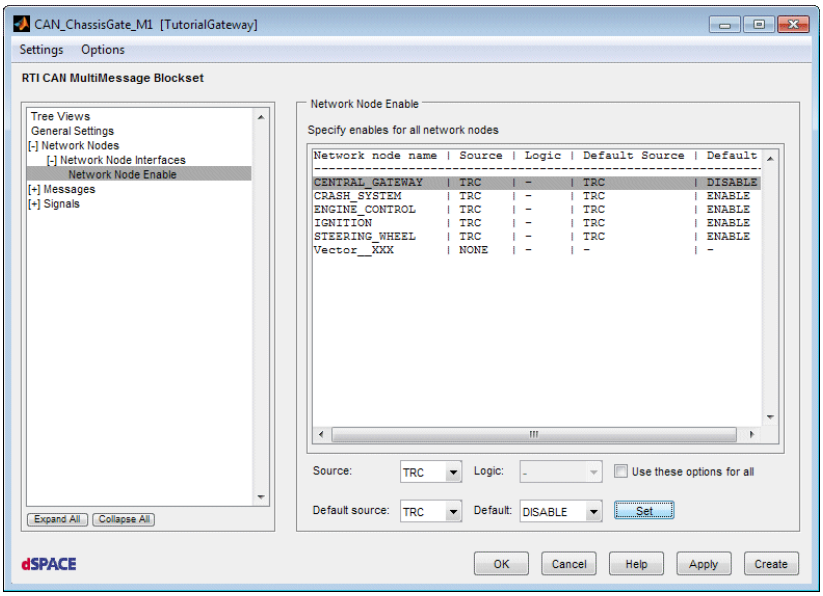
4 In the dialog tree, select **Network Nodes – Network Node Interfaces – Network Node Enable**.

5 Clear **Use these option for all**.

6 In the **Specify enables for all network nodes** list, select **CENTRAL_GATEWAY**.

7 From the **Default** drop-down list, select **Disable**.

8 Click **Set**.



CENTRAL_GATEWAY can be enabled or disabled in ControlDesk. It is disabled by default. None of the settings of the CAN_ChassisGate_M1 MainBlock apply to the CENTRAL_GATEWAY ECU.

9 Click **Create** to let the necessary files be created automatically and to apply the settings to the model.

10 Click **OK** to close the **CAN_ChassisGate_M1 [TutorialGateway]** dialog.

Interim result

Using the CAN_ChassisGate_M1 MainBlock, you simulated all the "real" ECUs of the Chassis CAN bus. The messages of these ECUs are transmitted via the physical TutGate_C1 controller. Since you disabled CENTRAL_GATEWAY by default, none of the settings specified for this block apply to CENTRAL_GATEWAY. You can now simulate all the "real" ECUs of the Body CAN bus. Continue with part 2.

Part 2

To simulate the "real" ECUs of the Body CAN bus

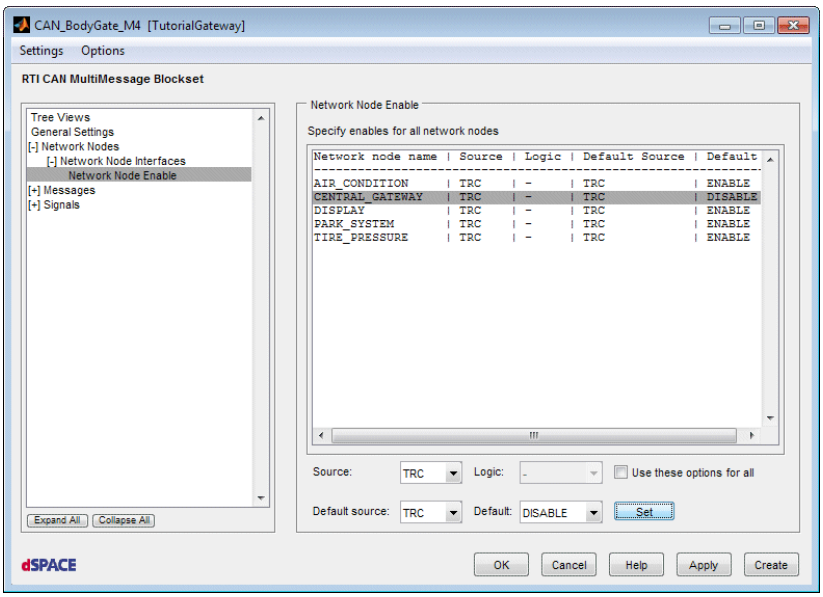
1 Open the CAN_BodyGate_M4 MainBlock.

2 On the Tree Views page, select Filter Configuration Tree by Feature.

3 Clear all filters except for Model Interface.

4 In the dialog tree, select Network Nodes – Network Node Interfaces – Network Node Enable.

5 Disable the CENTRAL_GATEWAY by default (as described in steps 5 ... 8 of part 1).



The CENTRAL_GATEWAY ECU is disabled by default.

6 Click Create.

7 Click OK to close the CAN_BodyGate_M4 [TutorialGateway] dialog.

Result

You simulated the "real" ECUs of the Chassis and Body CAN bus. Using the CAN_ChassisGate_M1 and CAN_BodyGate_M4 MainBlocks, you simulated the ECUs as follows:

RTICANMM MainBlock	Physical CAN Controller	Database	Simulated "Real" ECUs
CAN_ChassisGate_M1	TutGate_C1	chassis.dbc	ENGINE_CONTROL CRASH_SYSTEM IGNITION STEERING_WHEEL
CAN_BodyGate_M4	TutGate_C4	body.dbc	PARK_SYSTEM TIRE_PRESSURE AIR_CONDITION DISPLAY

What's next

In the next step, you can build a unidirectional gateway.

Related topics

Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Step 3: How to Build a Unidirectional Gateway Using Two RTICANMM MainBlocks

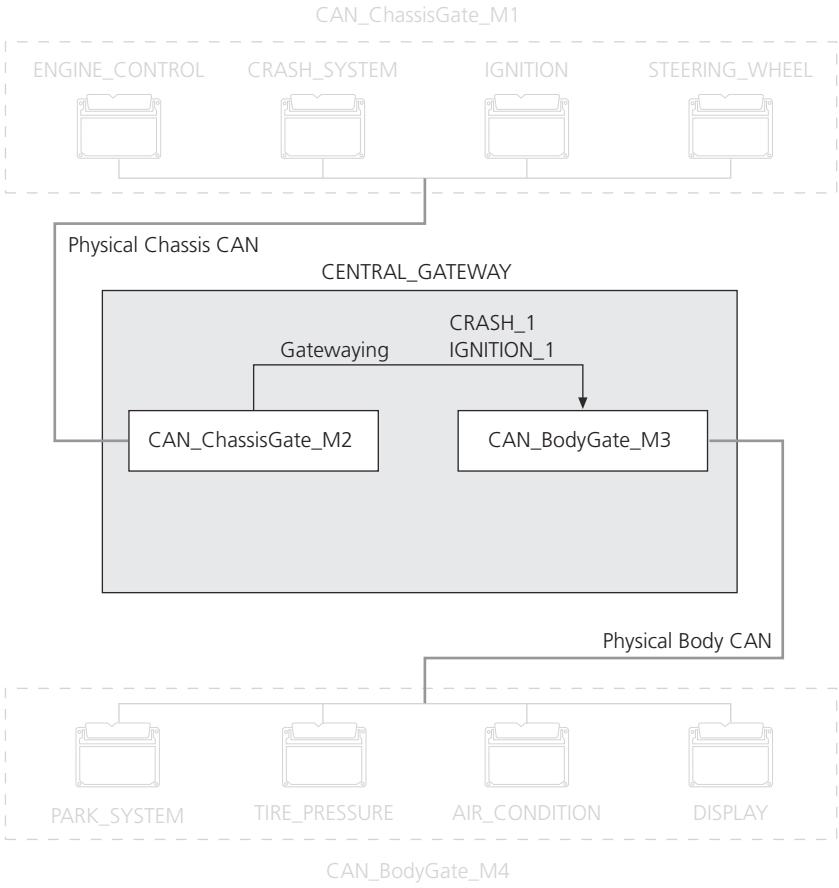
Objective

When you gateway messages between two CAN bus systems, you can have different gateway directions:

- From CAN bus 1 to CAN bus 2.
- From CAN bus 2 to CAN bus 1.
- From CAN bus 1 to CAN bus 2 and vice versa.

The first two cases are unidirectional gateways, the last is a bidirectional gateway.

In this step, you will build a unidirectional gateway between the Chassis CAN bus and the Body CAN bus. You will simulate the CENTRAL_GATEWAY ECU for both CAN buses and specify the CRASH_1 and IGNITION_1 messages and their signals to be gatewayed. Additionally, you will manipulate the triggering conditions of the messages for the Body CAN bus.

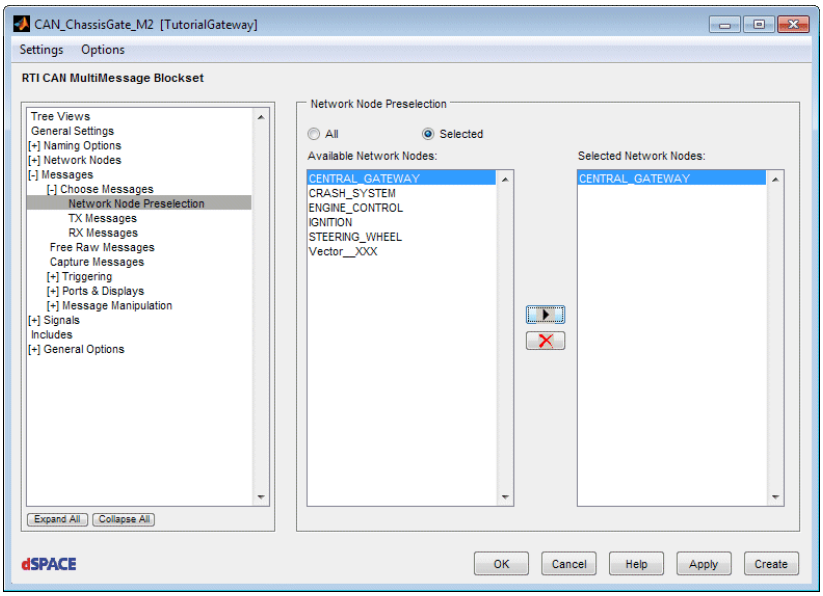


Part 1

To simulate the CENTRAL_GATEWAY ECU for the Chassis CAN bus

1. Open the CAN_ChassisGate_M2 MainBlock.
2. On the Tree Views page, select Filter Configuration Tree by Feature.
3. Clear all filters except for Model Interface and Extras.
4. In the dialog tree, select Messages – Choose Messages – Network Node Preselection.

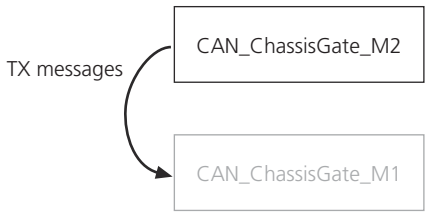
5 From the **Available network nodes** list, select **CENTRAL_GATEWAY** and click .



CENTRAL_GATEWAY is added to the **Selected network nodes** list.

6 In the dialog tree, select **Messages – Choose Messages – TX Messages**.

The TX messages of the CAN_ChassisGate_M2 MainBlock are transmitted on the Chassis CAN bus and received by the "real" ECUs of the CAN_ChassisGate_M1 MainBlock.



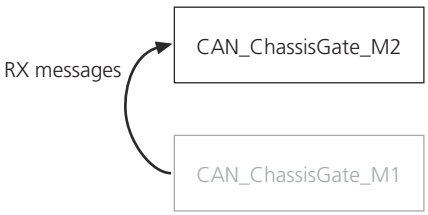
Since this MainBlock will gateway messages to the Body CAN bus but not transmit messages via the Chassis CAN bus, no TX messages are needed.

7 Select **None**.

The **Available messages** list is disabled and no messages are transmitted by this MainBlock.

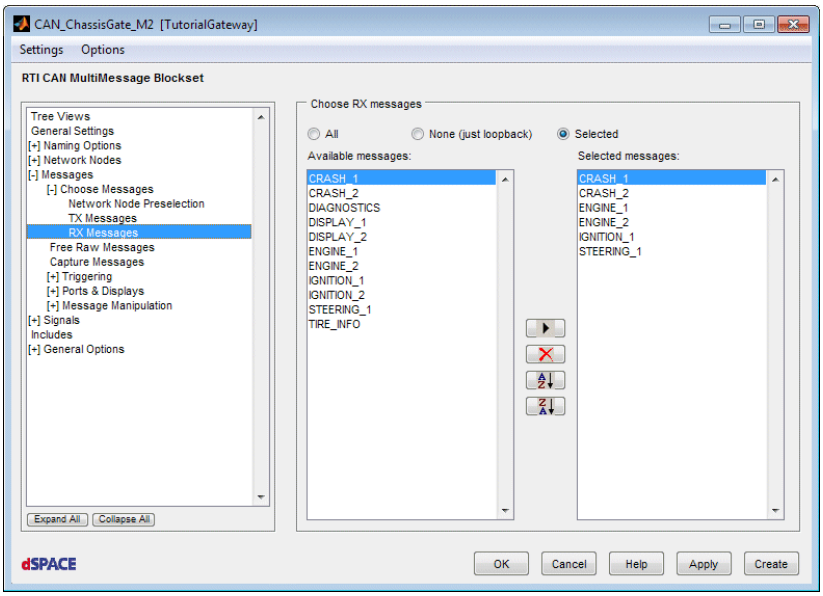
8 Select the **RX Messages** Page.

The CAN_ChassisGate_M2 MainBlock receives RX messages from the "real" ECUs of the CAN_ChassisGate_M1 MainBlock.



Some of these messages will be gatewayed to the Body CAN bus.

9 Right-click in the Selected messages list and select Choose Messages Received by Selected Network Nodes from the context menu.



All the messages received by the CENTRAL_GATEWAY ECU are added to the Selected messages list.

Interim result

You simulated CENTRAL_GATEWAY for the Chassis CAN bus. You disabled all the TX messages so that the CAN_ChassisGate_M2 block does not transmit messages via the Chassis CAN bus. The CAN_ChassisGate_M2 block is used to gateway messages from the Chassis CAN bus to the Body CAN bus. You therefore specified all the messages received by CENTRAL_GATEWAY from the Chassis CAN bus as RX messages. You can now specify some of these messages for gatewaying. Continue with part 2.

Part 2

To specify messages and signals of CENTRAL_GATEWAY for gatewaying

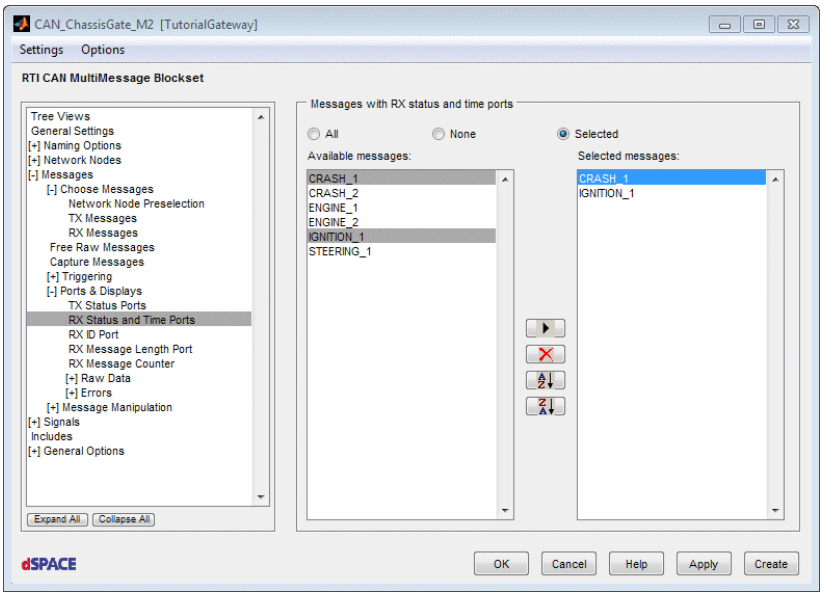
1 In the dialog tree, select Messages – Ports & Displays – RX Status and Time Ports.

2 Select Selected.

The Available messages list is enabled and displays all the available RX messages.

3 Select CRASH_1 and IGNITION_1.

4 Click .



The CRASH_1 and IGNITION_1 messages are added to the **Selected messages** list. These messages will be gatewayed by the CAN_ChassisGate_M2 MainBlock. When you create the CAN_ChassisGate_M2 MainBlock later on, an **RX Data** outport (providing status and time information for the selected RX messages) will be added to the CAN_ChassisGate_M2 block.

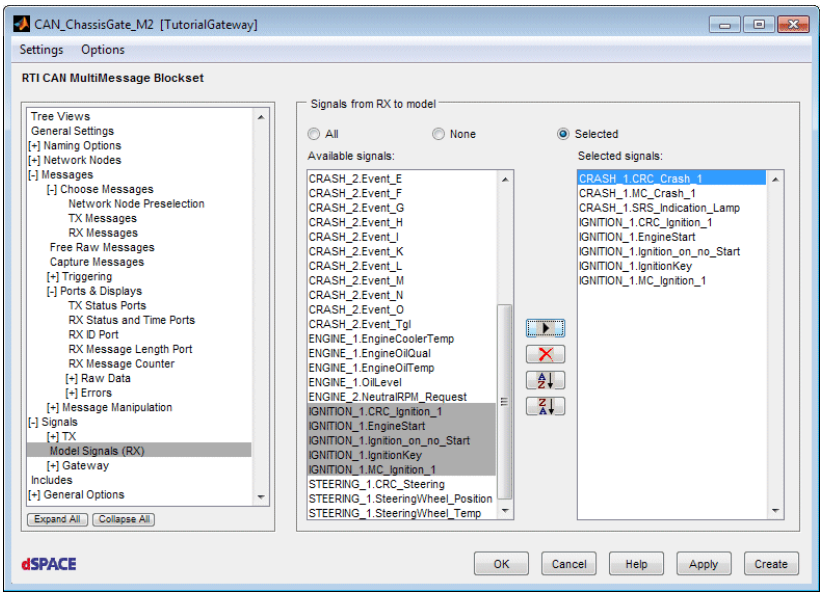
5 In the dialog tree, select **Signals – Model Signals (RX)**.

6 Select **Selected**.

The **Available signals** list is enabled and displays all the signals of the available RX messages.

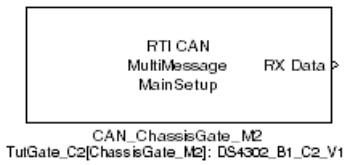
7 From the **Available signals** list, select all the signals of **CRASH_1** and **IGNITION_1**.

8 Click .



All the signals of CRASH_1 and IGNITION_1 are selected as RX outport signals. Only these signals can be manipulated before gatewaying.

9 Click **Create** to let the necessary files be created automatically and to apply the settings to the model. An **RX Data** outport is added to the CAN_ChassisGate_M2 MainBlock.



10 Click **OK** to close the **CAN_ChassisGate_M2 [TutorialGateway]** dialog.

Interim result

You specified CRASH_1 and IGNITION_1 to be gatewayed via the RX Data outport to the CAN_BodyGate_M3 MainBlock. You can now specify the triggering conditions for the transmission of CRASH_1 and IGNITION_1 via the Body CAN bus. Continue with part 3.

Part 3

To specify the triggering conditions for transmitting CRASH_1 and IGNITION_1 via the Body CAN bus

1 Open the **CAN_BodyGate_M3** MainBlock.

2 On the **Tree Views** page, select **Filter Configuration Tree by Feature**.

3 Clear all filters except for **Model Interface** and **Extras**.

4 In the dialog tree, select **Messages – Choose Messages – TX Messages**.

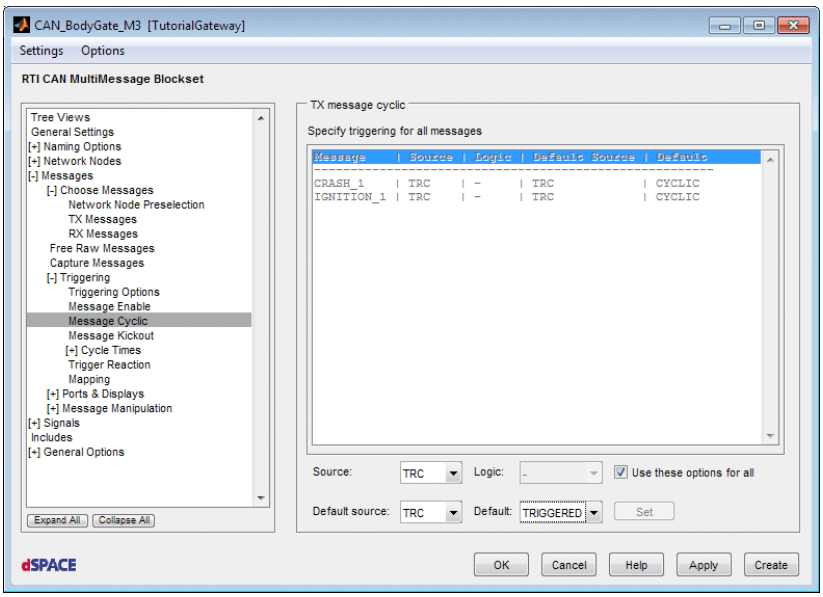
5 From the **Available messages** list, select **CRASH_1** and **IGNITION_1**.

6 Click .

CRASH_1 and IGNITION_1 are added to the **Selected messages** list. They are selected for transmission.

7 In the dialog tree, select **Messages – Triggering – Message Cyclic**.

8 From the **Default** drop-down list, select **Triggered**.



For CRASH_1 and IGNITION_1, triggered message transmission is enabled. That is, the messages are not transmitted cyclically, but message transmission will be triggered by the RX_Status of the CAN_ChassisGate_M2 MainBlock.

Tip

Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Since the **Specify triggering for all messages** list is disabled, it is not updated. The specified settings nevertheless have been applied.

9 Select the **Message Kickout** page.

CRASH_1 and IGNITION_1 are displayed in the **Specify kickout for all messages** list.

10 From the **Source** drop-down list, select **BOTH**.

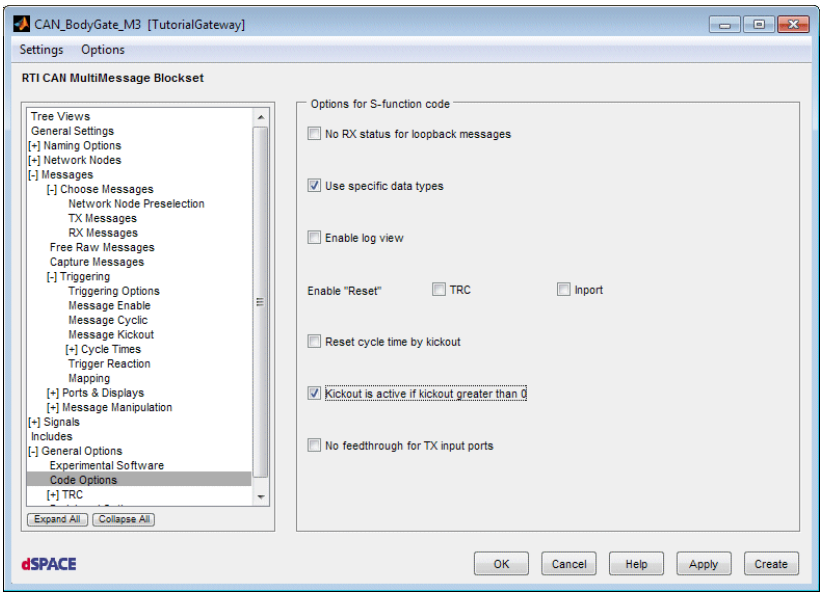
CRASH_1 and IGNITION_1 can be transmitted via kickout. The transmission can be triggered via the CAN_ChassisGate_M2 MainBlock or via ControlDesk. Since the **Specify kickout for all messages** list is disabled, it is not updated. When you create the **CAN_BodyGate_M3** MainBlock, a **Triggering** inport will be created. The Triggering inport will be connected to a **Triggering to RTICANMM** block automatically.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

11 In the dialog tree, select General Options – Code Options.

12 Select Kickout is active if kickout greater than 0.



Messages are transmitted via kickout when the RX Data outport receives a rising edge. After you selected this option, no rising edge is needed to transmit a message via kickout, only a value greater than 0. A message is transmitted each time a value greater than 0 is received.

Interim result

You specified the triggering conditions for CRASH_1 and IGNITION_1 transmission via the Body CAN bus. You can now specify the mapping structure for the transmission of both messages via the Body CAN bus. Continue with part 4.

Part 4

To specify the mapping structures of CRASH_1 and IGNITION_1 for transmission via the Body CAN bus

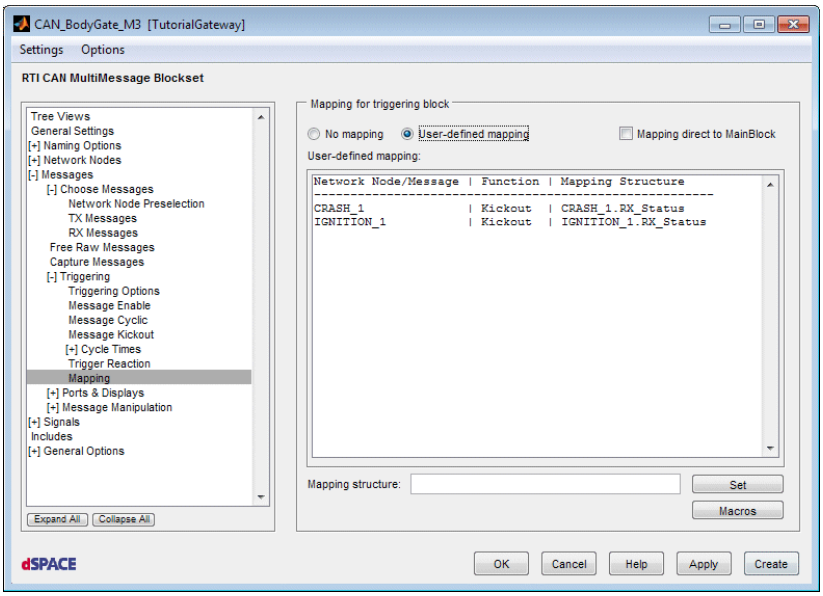
1 In the dialog tree, select Messages – Triggering – Mapping.

2 Select User-defined mapping.

The User-defined mapping list is enabled and CRASH_1 and IGNITION_1 are displayed. You can now specify your own mapping structure for triggering signals.

3 Right-click in the User-defined mapping list and select Gateway (All) from the context menu.

The mapping structures are added to the messages automatically. The RX_Status of the CAN_ChassisGate_M2 MainBlock triggers message transmission on the Body CAN bus via kickout.



Interim result

You specified the mapping structure for both messages to be transmitted via the Body CAN bus. You can now specify gateway signals for the signals of CRASH_1 and IGNITION_1. Additionally, you will enable error signals to manipulate some of the gateway signals during run time. Continue with part 5.

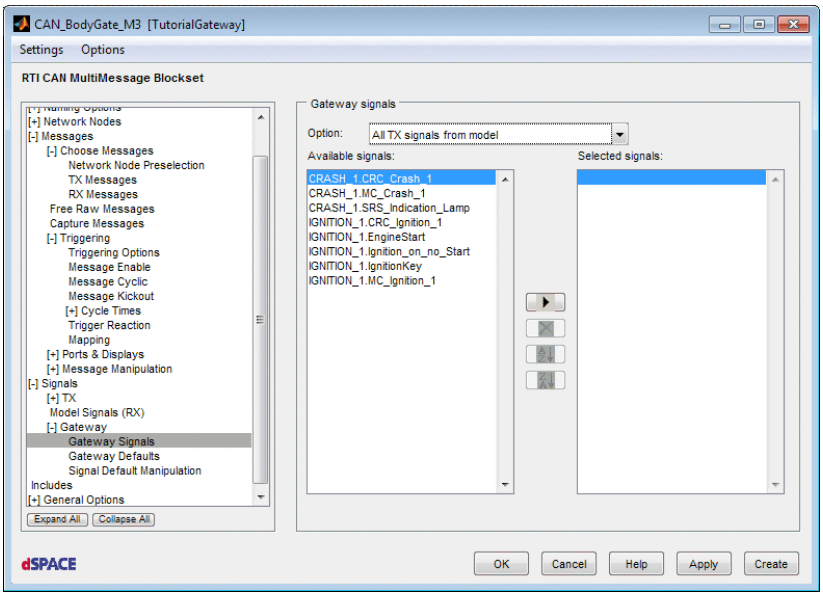
Part 5

To specify gateway signals and error signals

1 In the dialog tree, select Signals – Gateway – Gateway Signals.

On the Gateway Signals page, you can specify which signals of the TX messages should be transmitted with the values of the gateway signals.

2 From the Option drop-down list, select All TX signals from model.



The Available signals list is enabled and displays all the TX signals of this MainBlock. All these signals will be transmitted with the values of the corresponding gateway signals. When you create the CAN_BodyGate_M3 MainBlock, these settings will create a TX Data Gateway inport for the block.

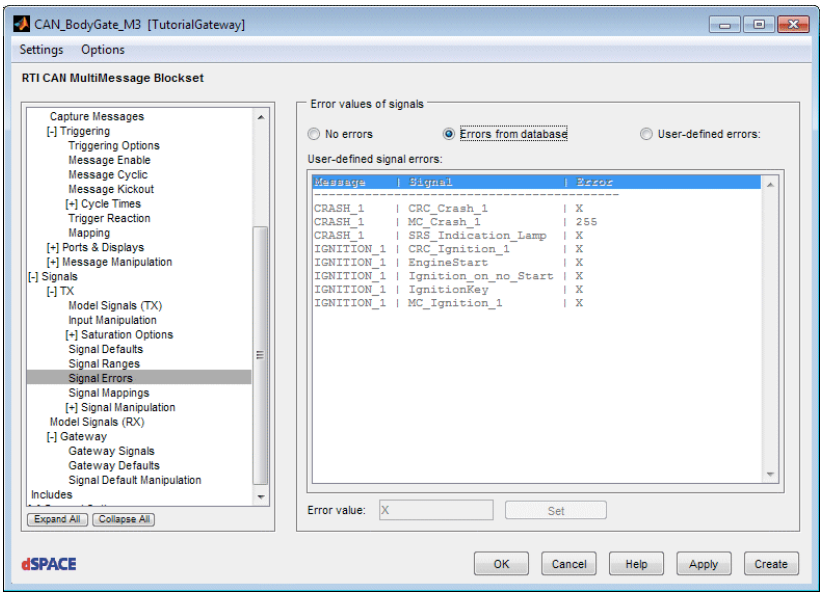
3 In the dialog tree, select Signals – TX – Signal Errors.

4 Select User-defined errors.

The User-defined signal errors list is enabled and displays the errors defined in the body.dbc file for the signals of the TX messages. An error value of 255 is defined for the MC_Crash_1 signal by default.

5 Select **Errors from database**.

The **User-defined signal errors** list is disabled but the settings defined in the database file are still displayed.



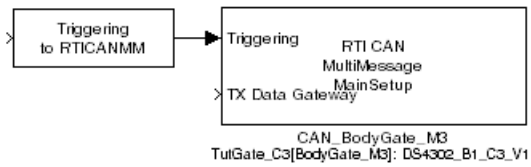
Tip

You selected **User-defined error signals** only to display the error values specified in the body.dbc file. Since you have not changed the error values specified in the database file, you selected **Errors from database**. Doing this, the settings of this page will be updated automatically: for example, if you update the database file.

6 Click **Create** to let the necessary files be created automatically and to apply the specified settings to the model.

7 Click **OK** to close the **CAN_BodyGate_M3 [TutorialGateway]** dialog.

The **CAN_BodyGate_M3** MainBlock is created. In your model, a **Triggering** inport and a **TX Data Gateway** inport are added to the CAN_BodyGate_M3 MainBlock. The Triggering block is automatically connected to a **Triggering to RTICANMM** block. Gateway signals from the CAN_ChassisGate_M2 block can be connected to the CAN_BodyGate_M3 block via the TX Data Gateway inport.



Interim result

You selected all the signals of CRASH_1 and IGNITION_1 to be transmitted with the values of the corresponding gateway signals. However, you enabled the

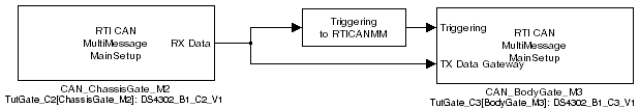
transmission of the error values defined in the database file. During run time, you can transmit the gateway value or the error value of the MC_Crash_1 signal. You can now build the gateway between the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlock. Continue with part 6.

Part 6

To build the gateway between CAN_ChassisGate_M2 and CAN_BodyGate_M3

1 Connect the RX Data outport of the CAN_ChassisGate_M2 MainBlock to the Triggering to RTICANMM block of the CAN_BodyGate_M3 MainBlock. This connects the RX_Status to the kickout function of the CRASH_1 and IGNITION_1 messages.

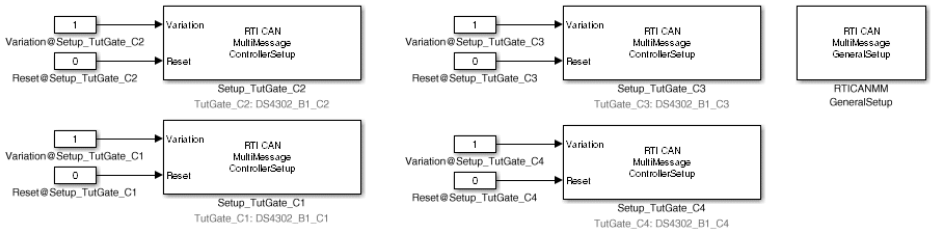
2 Connect the TX Data Gateway inport of the CAN_BodyGate_M3 MainBlock to the RX Data outport of the CAN_ChassisGate_M2 MainBlock.

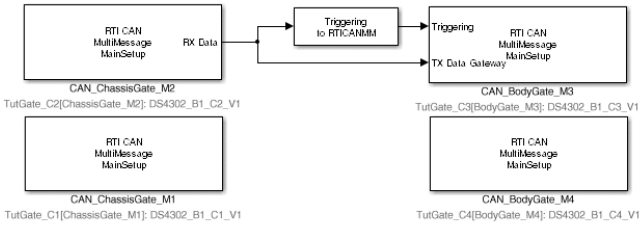


3 Save your model.

Result

You built a unidirectional gateway between the Chassis CAN bus and the Body CAN bus. Messages are transmitted from the Chassis CAN bus to the Body CAN bus.





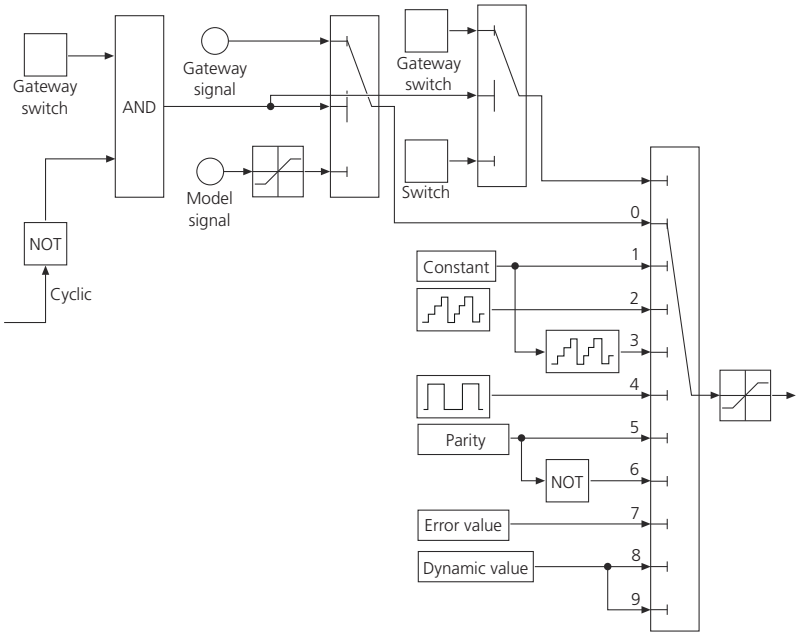
Behavior of the CAN_ChassisGate_M1 MainBlock This block simulates all the "real" ECUs of the Chassis CAN bus. The "real" ECUs are ENGINE_CONTROL, CRASH_SYSTEM, IGNITION, and STEERING_WHEEL. The TX messages of these ECUs will not be manipulated via your model.

Behavior of the CAN_BodyGate_M4 MainBlock This block simulates all the "real" ECUs of the Body CAN bus. The "real" ECUs are PARK_SYSTEM, TIRE_PRESSURE, AIR_CONDITION, and DISPLAY. The TX messages of these ECUs will not be manipulated via your model.

Behavior of the CAN_ChassisGate_M2 MainBlock This block simulates the CENTRAL_GATEWAY ECU of the Chassis CAN bus. It acts as a gateway to the CAN_BodyGate_M3 MainBlock. Since gatewaying messages via two MainBlocks reduces performance, only CRASH_1 and IGNITION_1 are gatewayed.

- *Settings specified on the TX Messages page*: You disabled all the TX messages. Since this block only gateways messages from the Chassis CAN bus to the Body CAN bus, no messages need to be transmitted via the Chassis CAN bus.
- *Settings specified on the RX Messages page*: You selected all the messages received by CENTRAL_GATEWAY as RX messages. This selection is needed to specify messages for gatewaying from the Chassis CAN bus to the Body CAN bus.
- *Settings specified on the RX Status and Time Ports page*: You selected CRASH_1 and IGNITION_1 to be gatewayed. An RX Data outport is generated and provides status information according to the reception of messages. If a message is not yet received in the current sampling step, the RX_Status is set to 0. If a message has been received in the current sampling step, the RX_Status is set to 1. These values are needed to trigger the transmission of a message via the Body CAN bus.
- *Settings specified on the Model Signals (RX) page*: You selected all the signals of CRASH_1 and IGNITION_1 to be gatewayed. The signals will be gatewayed via the RX Data outport. You can manipulate these signals before gatewaying using the CAN_BodyGate_M3 MainBlock.

Behavior of the CAN_BodyGate_M3 MainBlock This block simulates the CENTRAL_GATEWAY ECU of the Body CAN bus. It transmits CRASH_1 and IGNITION_1 to the Body CAN bus. The messages are transmitted according to the switch displayed in the following illustration.



- *Settings specified on the TX Messages page*: Since only CRASH_1 and IGNITION_1 are gatewayed, you selected only these two messages as TX messages. They will be transmitted on the Body CAN bus. You can manipulate them before transmission. Since you selected all the signals of both messages as RX signals in the CAN_ChassisGate_M2 MainBlock, you can also manipulate the signals before transmitting.

Tip

You can, for example, manipulate CRASH_1 and IGNITION_1 and their signals as described in *Lesson 7: Customizing Messages on page 169* and *Lesson 8: Customizing Signals on page 191*.

- *Settings specified on the Message Cyclic page*: You selected triggered transmission to enable the transmission of gatewayed messages.
- *Settings specified on the Message Kickout page*: You enabled the kickout of messages via your model and via ControlDesk. When you do this, message transmission can be triggered via the messages received by the CAN_ChassisGate_M2 MainBlock. You can also kickout CRASH_1 and IGNITION_1 via the corresponding layout in ControlDesk.
- *Settings specified on the Mapping page*: You selected **Gateway (All)**. When you do this, the message transmission of CRASH_1 and IGNITION_1 is triggered by the RX_Status value transmitted from the RX Data outport of CAN_ChassisGate_M2. For example, when the CAN_ChassisGate_M2 MainBlock receives CRASH_1, the RX_Status is set to 1. This triggers the transmission of CRASH_1 via the Body CAN bus.

- *Settings specified on the Gateway Signals page*: You selected **All TX signals from model** as gateway signals. This lets you transmit the signals of CRASH_1 and IGNITION_1 with the values of the signals specified as RX signals in the CAN_ChassisGate_M2 MainBlock. A TX Data Gateway inport is created. During run time, you can enable and disable the transmission of the gateway values via ControlDesk.
- *Settings specified on the Signal Errors page*: You enabled the transmission of the error values defined in the body.dbc file for the signals of the TX messages. For MC_Crash_1 an error value of 255 is specified. During run time, you can switch to transmit the gateway value or the error value of the signal.
- *Settings specified on the General Options page*: You selected **Kickout is active if kickout greater than 0**. This option is needed to ensure the kickout of messages if they are received in millisecond steps. If a message is received in millisecond steps, the RX_Status value might never be 0 but always 1. Since the RX_Status needs a rising edge to enable kickout, no messages can be transmitted via kickout if the value is always 1. When you select this option, no rising edge is needed to kickout a message but only an RX_Status value greater than 0.

What's next

In the next step, you can extend your model and build a bidirectional gateway.

Related topics

Basics

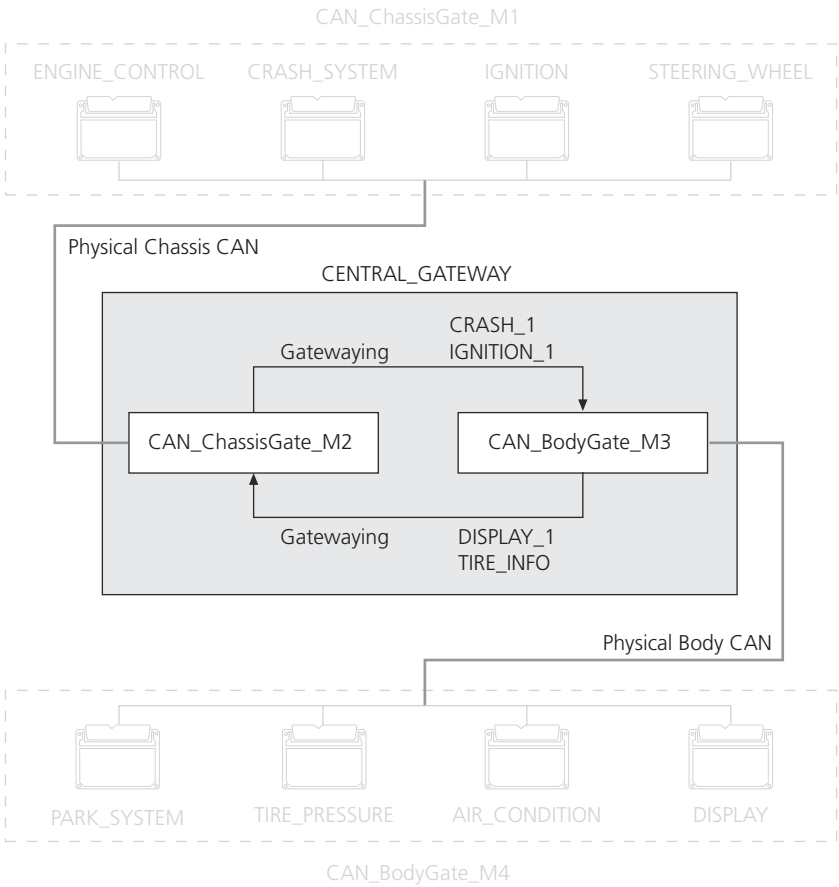
Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Step 4: How to Build a Bidirectional Gateway Using Two RTICANMM MainBlocks

Objective

A bidirectional gateway can exchange messages between two CAN buses in both directions

In this step, you will extend the gateway you built in the previous step. For the CAN_BodyGate_M3 MainBlock, you will specify the DISPLAY_1 and TIRE_INFO messages to be gatewayed from the Body CAN bus to the Chassis CAN bus. These messages will be transmitted on the Chassis CAN bus via the CAN_ChassisGate_M2 MainBlock.



Part 1

To specify messages sent by the CAN_BodyGate_M3 MainBlock

1 Open the CAN_BodyGate_M3 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – RX Messages.

3 In the Available messages list, select DISPLAY_1 and TIRE_INFO and click .

Both messages are added to the Selected messages list. They are received by the CENTRAL_GATEWAY ECU from the Body CAN bus and will be gatewayed to the Chassis CAN bus.

4 In the dialog tree, select Messages – Ports & Displays – RX Status and Time Ports.

5 Select All.

The Available messages list is enabled. The status and time ports are enabled for both RX messages. When you create the CAN_BodyGate_M3 MainBlock, an RX Data outport will be added to the block.

Tip

Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

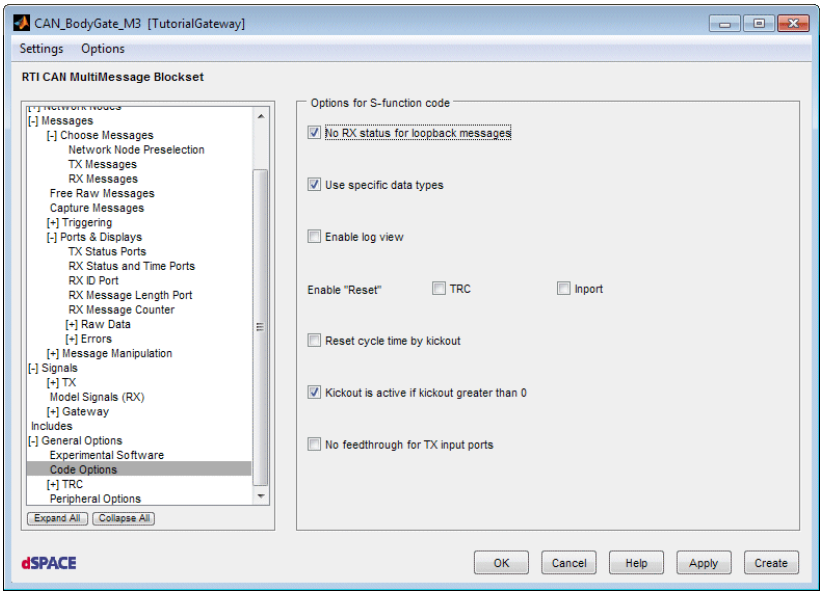
6 In the dialog tree, select **Signals – Model Signals (RX)**.

7 Select **All**.

The **Available signals** list is enabled and displays all the signals of the available RX messages. All the displayed signals are selected as RX outport signals.

8 In the dialog tree, select **General Options – Code Options**.

9 Select **No RX status for loopback messages**.



Each time a CAN controller transmits a message, the same CAN controller receives this message as a loopback message. Since loopback messages can change the RX_Status, they can affect the transmission of messages mapped to the RX_Status. By selecting this checkbox, you prevent the RX_Status from being changed when a loopback message is received.

10 Click **Create**.

The necessary files are created automatically. The **Overwrite?** dialog opens and asks you whether you want to overwrite the Triggering to RTICANMM block.

11 In the **Overwrite?** dialog, click **Overwrite**.

The Triggering to RTICANMM block is overwritten, and the settings specified in the CAN_BodyGate_M3 MainBlock apply to your model.

12 Click **OK** to close the **CAN_BodyGate_M3 [TutorialGateway]** dialog.

The CAN_BodyGate_M3 MainBlock is created. In your model, an **RX Data** outport is added to the block.

Interim result

You specified the messages to be gatewayed between the Body CAN bus and the Chassis CAN bus. You can now specify TX messages of the CAN_ChassisGate_M2 MainBlock. Continue with part 2.

Part 2

To specify TX messages of the CAN_ChassisGate_M2 MainBlock

1 Open the CAN_ChassisGate_M2 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

3 Select Selected.

4 In the Available messages list, select DISPLAY_1 and TIRE_INFO and click .

Both messages are added to the Selected messages list. These two messages are received by the CENTRAL_GATEWAY ECU from the Body CAN bus. CENTRAL_GATEWAY transmits them to the Chassis CAN bus.

Interim result

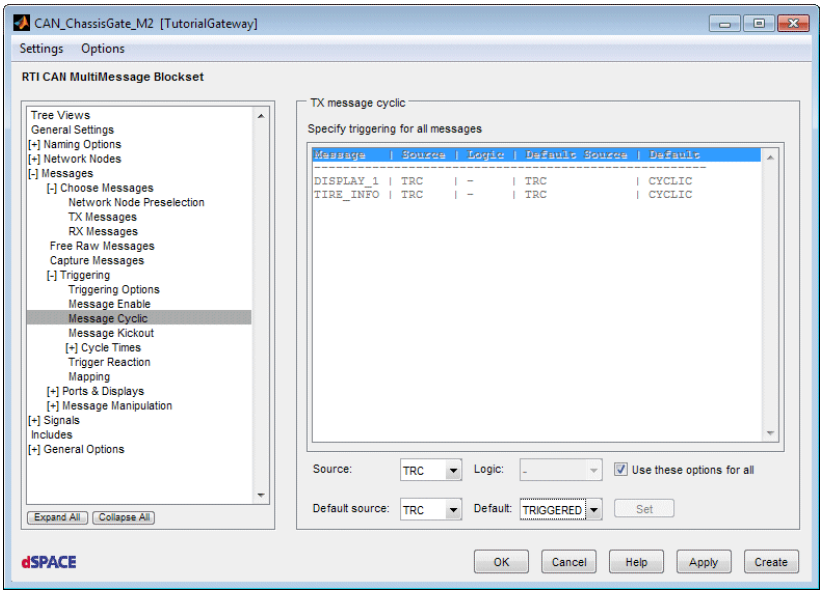
You selected DISPLAY_1 and TIRE_INFO to be transmitted from the CAN_ChassisGate_M2 MainBlock on the Chassis CAN bus. You can now specify the triggering conditions of these messages. Continue with part 3.

Part 3

To specify the triggering conditions for transmitting DISPLAY_1 and TIRE_INFO via the Chassis CAN bus

1 In the dialog tree, select Messages – Triggering – Message Cyclic.

2 From the Default drop-down list, select Triggered.



Triggered message transmission is enabled for DISPLAY_1 and TIRE_INFO. That is, the messages are not transmitted cyclically, but message transmission will be triggered by the RX_Status of the CAN_BodyGate_M3 MainBlock. Since the **Specify triggering for all messages** list is disabled, it is not updated.

> **Tip**
>
> Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

3 Select the **Message Kickout** page.

4 From the **Source** drop-down list, select **BOTH**.

DISPLAY_1 and TIRE_INFO can be transmitted via kickout. The transmission can be triggered via the CAN_BodyGate_M3 MainBlock or via ControlDesk. Since the **Specify kickout for all messages** list is disabled, it is not updated. When you create the CAN_ChassisGate_M2 MainBlock, a **Triggering** inport will be created. The Triggering inport will be automatically connected to a **Triggering to RTICANMM** block.

> **Tip**
>
> Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You specified the triggering conditions for the messages to be transmitted via the Chassis CAN bus. You can now specify their mapping structure. Continue with part 4.

Part 4

To specify the mapping structures for DISPLAY_1 and TIRE_INFO

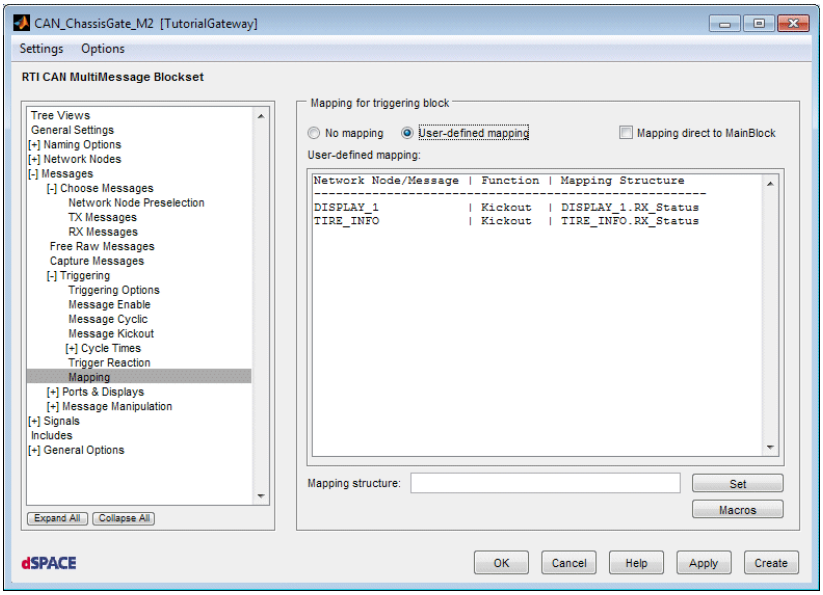
1 In the dialog tree, select **Messages – Triggering – Mapping**.

2 Select **User-defined mapping**.

The **User-defined mapping** list is enabled and DISPLAY_1 and TIRE_INFO are displayed.

3 Right-click in the **User-defined mapping** list and select **Gateway (All)** from the context menu.

The mapping structures are added to the messages automatically. The RX_Status of the CAN_BodyGate_M3 MainBlock triggers message transmission on the Chassis CAN bus via kickout.



Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

4 In the dialog tree, select General Options – Code Options.

5 Select Kickout is active if kickout greater than 0.

6 Select No RX status for loopback messages.

This prevents the RX_Status from being changed when a loopback message is received.

Interim result

You specified the mapping structures for DISPLAY_1 and TIRE_INFO to be transmitted via the Chassis CAN bus. You can now specify gateway signals for the signals of DISPLAY_1 and TIRE_INFO. Continue with part 5.

Part 5

To specify gateway signals

1 In the dialog tree, select Signals – Gateway – Gateway Signals.

2 From the Option drop-down list, select All TX signals from model.

The Available signals list is enabled and displays all the TX signals of this MainBlock. Gateway signals are defined for all these signals, that is, all these signals will be transmitted with the values of the corresponding gateway signals.

3 Click Create.

4 Click OK to close the CAN_ChassisGate_M2 [TutorialGateway] dialog.

Interim result

You selected all the signals of DISPLAY_1 and TIRE_INFO to be transmitted with the values of the corresponding gateway signals. You can now build the gateway between the CAN_BodyGate_M3 and CAN_ChassisGate_M2 MainBlock. Continue with part 6.

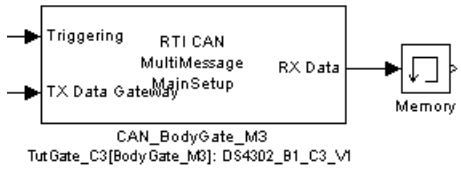
Part 6

To build the bidirectional gateway between CAN_BodyGate_M3 and CAN_ChassisGate_M2

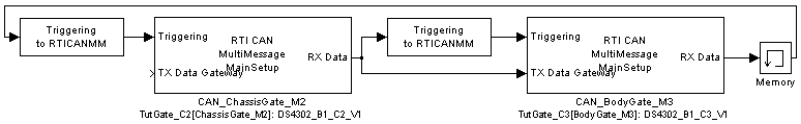
1. On the Simulation ribbon, click Library – Library Browser.

   The Simulink Library Browser opens.
2. In the Simulink Library Browser, select the Discrete library.
3. From the Discrete library, drag a Memory block to your model.

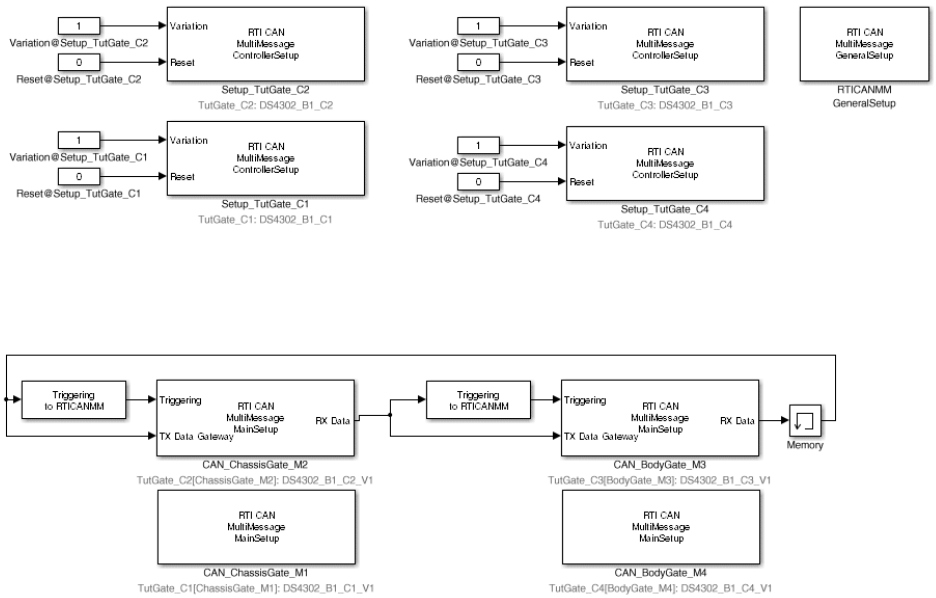
   To realize the bidirectional gateway, you will build an inport-outport loop between the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlocks. In this case, MATLAB requires the Memory block to separate the inputs and outputs of the MainBlocks and to process each MainBlock's data in the correct order.
4. Close the Simulink Library Browser.
5. Connect the RX Data outport of the CAN_BodyGate_M3 MainBlock to the inport of the Memory block.



6. Connect the outport of the Memory block to the Triggering to RTICANMM block of the CAN_ChassisGate_M2 MainBlock.



7 Connect the **TX Data Gateway** inport of the CAN_ChassisGate_M2 MainBlock to the outport of the **Memory** block.



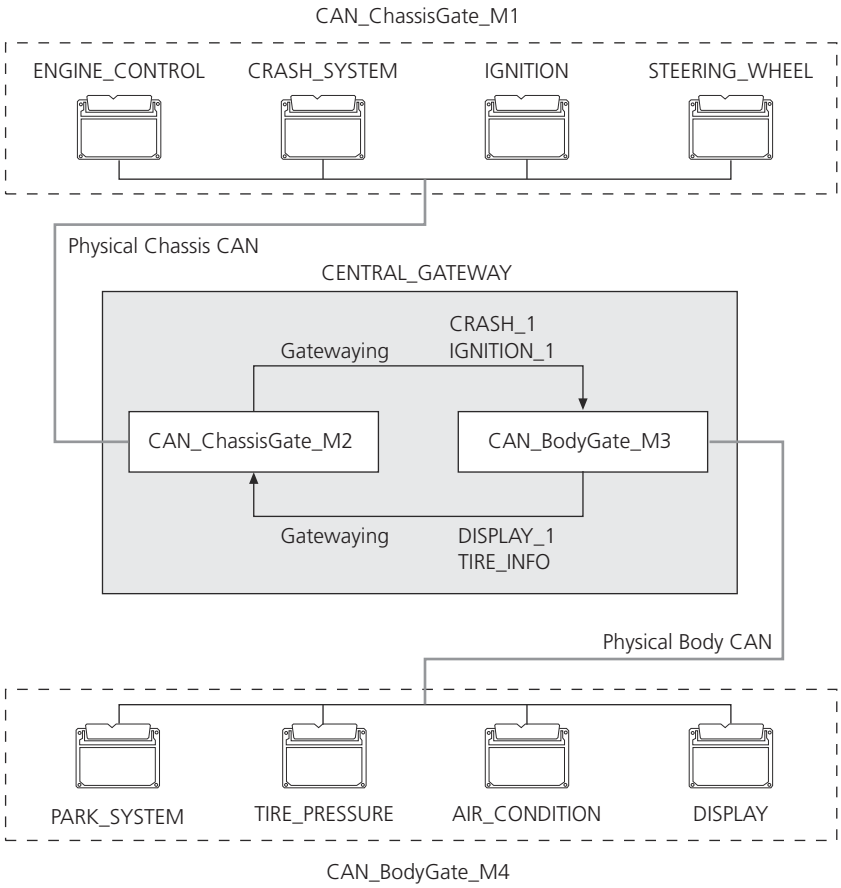
8 Save your model.

Result

You built a bidirectional gateway between the Chassis CAN bus and the Body CAN bus.

- The DISPLAY_1 and TIRE_INFO messages and their signals are gatewayed from the Body CAN bus to the Chassis CAN bus.

- The CRASH_1 and IGNITION_1 messages and their signals are gatewayed from the Chassis CAN bus to the Body CAN bus.



Building a bidirectional gateway hardly differs from building a unidirectional gateway. In principle, you repeated all the steps that were required to build the unidirectional gateway, but in reverse direction. You only switched the MainBlocks: All the steps you performed for CAN_ChassisGate_M2 in step 2 of this lesson were performed for CAN_BodyGate_M3 in this step, and vice versa.

Since you built an inport-outport loop between the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlocks to realize the bidirectional gateway, you added a Memory block to your model. The Memory block enables MATLAB to separate the inputs and outputs of the MainBlocks and to process each MainBlock's data in the correct order.

In addition, you prevented the RX_Status from being changed when a loopback message is received.

What's next

In the next step, you can gateway all the other messages of the two bus systems using an RTICANMM Gateway block.

Related topics

Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Step 5: How to Gateway Messages Using the RTICANMM Gateway Block

Objective

For the best performance, all the messages which do not need to be manipulated before gatewaying should be gatewayed via the RTICANMM Gateway block. Using the RTICANMM Gateway block allows you to specify the gateway direction or exclude single messages from gatewaying, for example.

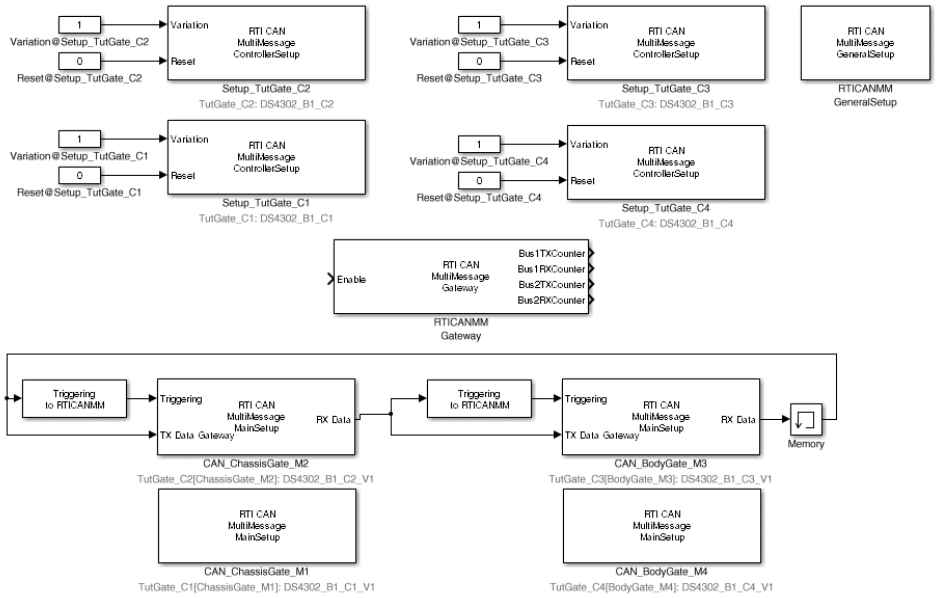
In this step, you will gateway messages via the RTICANMM Gateway block and exclude those messages from being gatewayed by this block which need to be manipulated before gatewaying.

Part 1

To add an RTICANMM Gateway block to your model

1. In the MATLAB Command Window, type **`rticanmm`** to open the rticanmmlib library.

2 From the rticanmmlib library, drag an RTICANMM Gateway block to your model.



3 Close the rticanmmlib library.

Interim result

You added an RTICANMM Gateway block to your model. You can now select the MainBlocks between which the gateway will be performed. Continue with part 2.

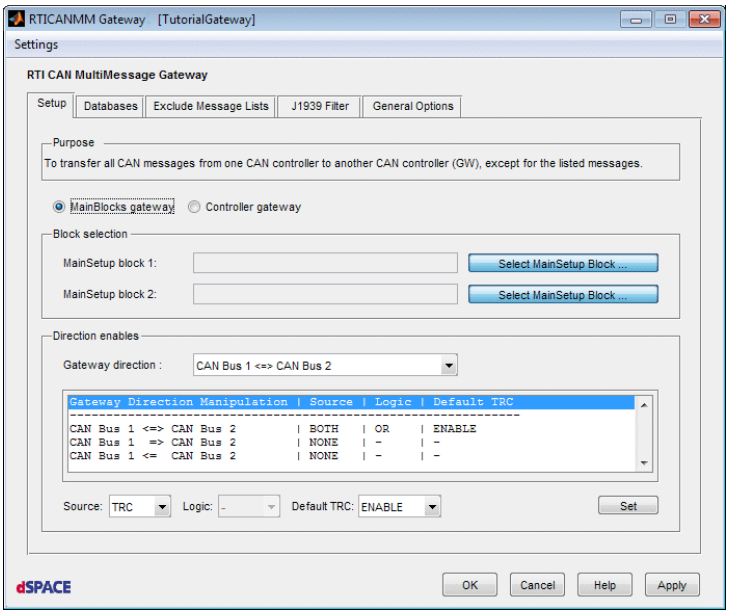
Part 2

To select the relevant MainBlocks for gatewaying

1 Double-click the RTICANMM Gateway block to open it.

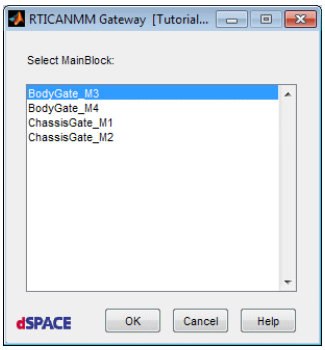
The Setup page is displayed.

2 Select MainBlocks gateway.



3 For the MainSetup block 1, click Select MainSetup Block.

A dialog opens which lists all the MainBlocks available in your model.



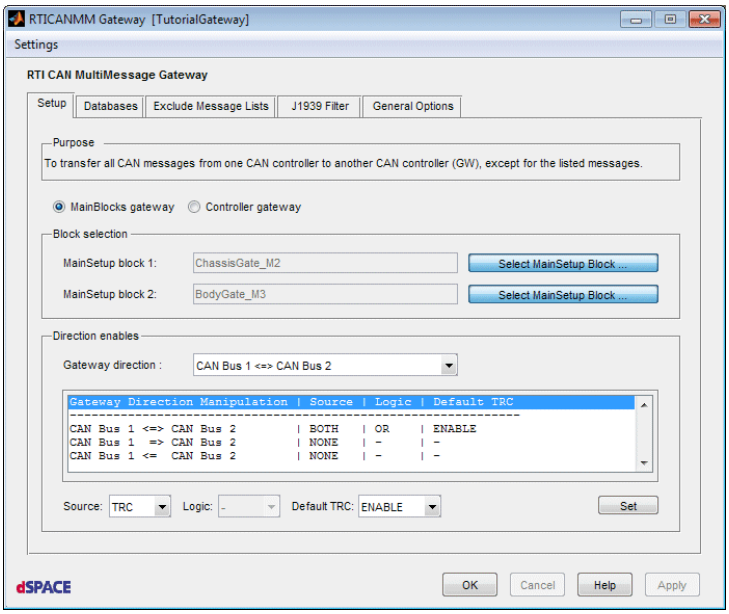
4 Select ChassisGate_M2 and click OK.

The dialog closes and the ChassisGate_M2 MainBlock is assigned to MainSetup block 1.

5 For the MainSetup block 2, click Select MainSetup Block.

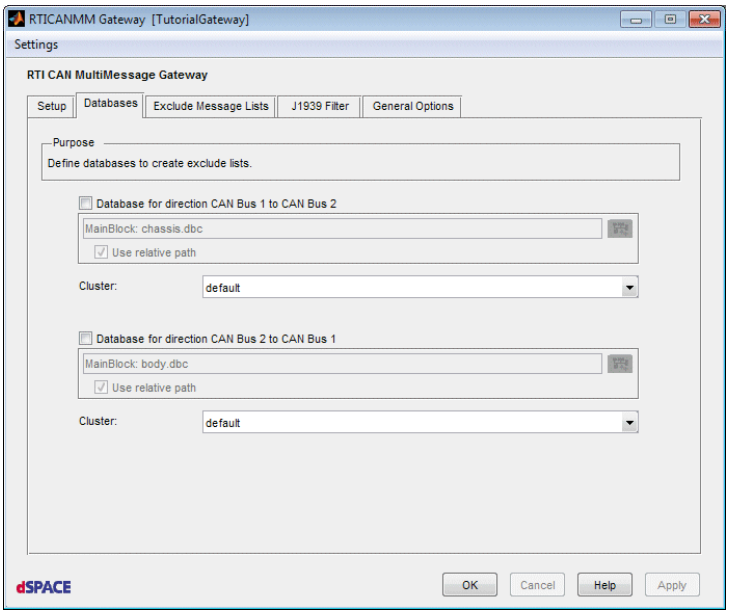
6 From the dialog, select **BodyGate_M3** and click **OK**.

The dialog closes and the BodyGate_M3 MainBlock is assigned to the MainSetup block 2.



7 Click **Apply** to apply your settings to the block.

8 Select the **Databases** page.



Since you selected **MainBlocks gateway** on the Setup page, the database files specified for the chosen MainBlocks are selected automatically on the Databases page. You do not have to specify the database files manually.

Interim result

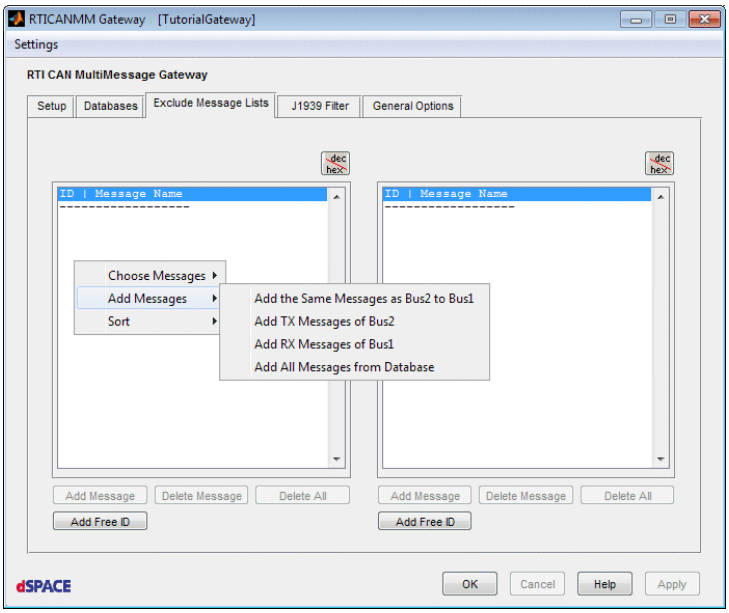
You selected the MainBlocks between which messages will be gatewayed. All the messages available in the DBC files specified for the MainBlocks will be gatewayed. You can now exclude messages which are already gatewayed by the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlocks from being gatewayed by the RTICANMM Gateway block. Continue with part 3.

Part 3

To exclude messages from being gatewayed by the RTICANMM Gateway block

1 Select the Exclude Message Lists page.

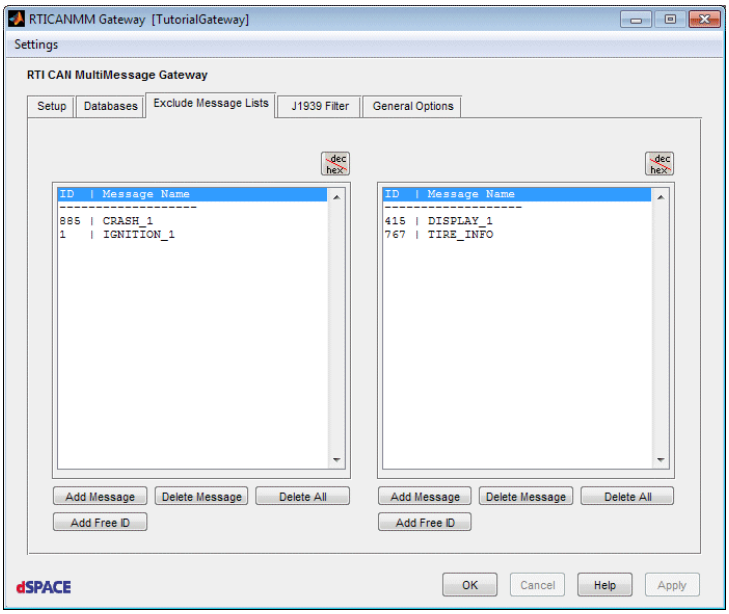
2 Right-click in the left exclude message list and select Add Messages – Add TX Messages of Bus2.



CRASH_1 and IGNITION_1 are added to the exclude message list. These messages are gatewayed by the CAN_ChassisGate_M2 MainBlock (CAN bus 1) to the CAN_BodyGate_M3 MainBlock. The CAN_BodyGate_M3 MainBlock transmits these messages via the Body CAN bus (bus 2). Since these messages are gatewayed via the MainBlocks, they do not need to be gatewayed by the RTICANMM Gateway block.

3 Right-click in the right exclude message list and select Add Messages – Add TX Messages of Bus1.

DISPLAY_1 and TIRE_INFO are added to the right exclude message list.



These messages are gatewayed by the CAN_BodyGate_M3 MainBlock and transmitted by the CAN_ChassisGate_M2 MainBlock via the Chassis CAN bus (bus 1).

Interim result

You specified CRASH_1, IGNITION_1, DISPLAY_1, and TIRE_INFO to be gatewayed via the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlock. You therefore disabled them from being gatewayed by the RTICANMM Gateway block. You can now specify general options for the RTICANMM Gateway block, for example, the gateway name. Continue with part 4.

Part 4

To specify general options for the RTICANMM Gateway block

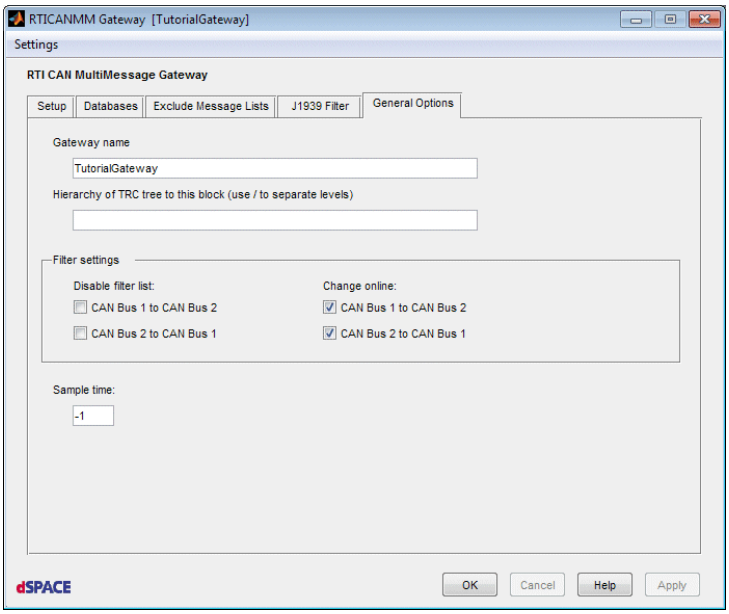
1 Select the General Options page.

2 Change the Gateway name to `TutorialGateway`.

Note

The gateway name must be unique within the model and even within a whole multiprocessor system, so you should always rename the gateway.

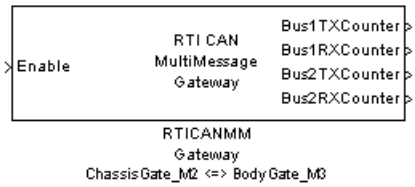
3 In the Filter settings frame, select CAN Bus 1 to CAN Bus 2 and CAN Bus 2 to CAN Bus 1 for Change online.



This allows you to include or exclude messages for both gateway directions during run time, for example, to simulate message failure.

4 Click OK.

The dialog closes and an S-function is created automatically. Your model displays the gateway direction.



Interim result

You specified a gateway name and enabled changing the specified filter settings during run time. This allows you, for example, to change the gateway behavior during run time or to simulate messages failures. You can now connect the inport and outports of the RTICANMM Gateway block to Simulink blocks. Continue with part 5.

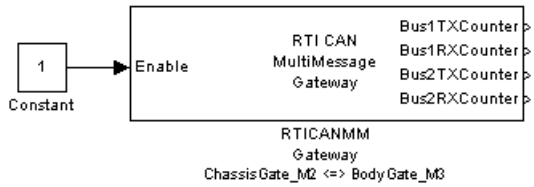
Part 5

To connect the inport and outports of the RTICANMM Gateway block

1 Open the Simulink Library Browser.

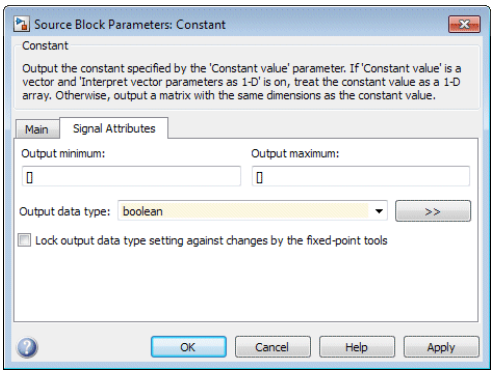
2 Select Commonly Used Blocks.

3 Drag a **Constant** block to your model and connect it with the **Enable** inport of the RTICANMM Gateway block.



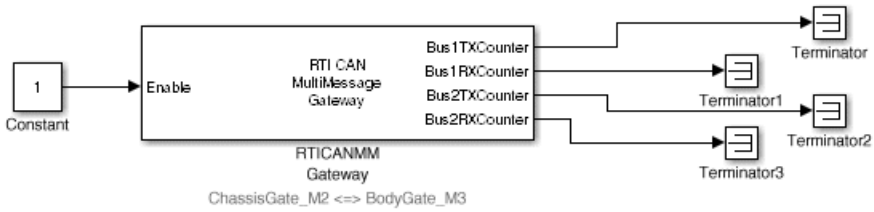
4 Double-click the Constant block to open its property dialog and select the **Signal Attributes** page.

5 From the **Output data type** drop-down list, select **boolean**.



6 Click **OK** to close the dialog.

7 From the **Commonly Used Blocks**, drag four **Terminator** blocks to your model and connect them to the outports of the RTICANMM Gateway block.



8 Close the **Simulink Library Browser**.

9 Save your model.

Result

All the messages of your model except for the CRASH_1, IGNITION_1, DISPLAY_1, and TIRE_INFO messages are gatewayed via the RTICANMM Gateway block. CRASH_1, IGNITION_1, DISPLAY_1, and TIRE_INFO are excluded from gatewaying via this block because they are already gatewayed via the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlock.

What's next

In the next step, you can check the gateway behavior via ControlDesk.

Related topics

Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Step 6: How to Check the Gateway Behavior via ControlDesk

Objective

In ControlDesk, you can check the behavior of gatewayed messages. Additionally, you can manipulate all the messages you gatewayed using the RTICANMM MainBlocks.

In this step, you will check the behavior of gateway messages and manipulate some of them.

Part 1

To check the behavior of the CRASH_1 message

1. Build an application for your TutorialGateway model.
2. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
3. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

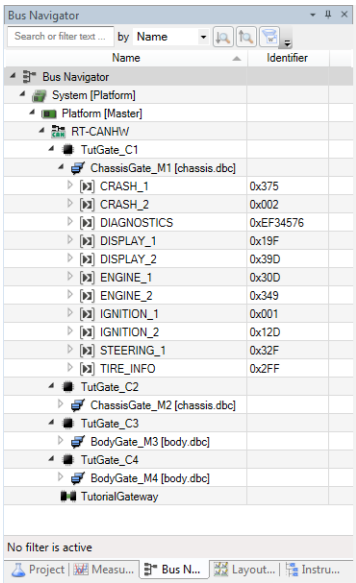
   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
4. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.
5. From the platform's context menu in the Project controlbar, select Add Variable Description.

   ControlDesk opens the Select Variable Description dialog.
6. In the Select Variable Description dialog, click Import and select the `TutorialGateway.sdf` file from your `RTICANMMTutorial_Lesson12` folder.
7. In the Select Variable Description dialog, click OK.

   ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

In addition to the four CAN controllers, the CAN bus configuration now also displays the TutorialGateway node.



8 Generate the following layouts for the CRASH_1 message:

CAN Bus	CRASH_1 Layout	Related MainBlock
ChassisGate_M1	TX	CAN_ChassisGate_M1
ChassisGate_M2	RX	CAN_ChassisGate_M2
BodyGate_M3	TX	CAN_BodyGate_M3
BodyGate_M4	RX	CAN_BodyGate_M4

9 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.

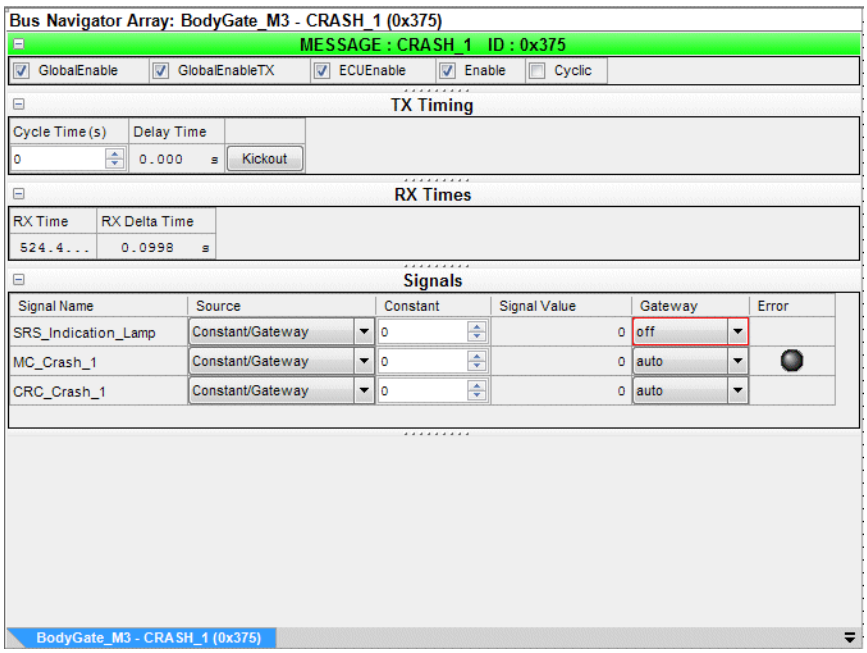
- The CAN_ChassisGate_M1 MainBlock transmits CRASH_1 cyclically with a cycle time of 100 ms.
- The CAN_ChassisGate_M2 MainBlock receives CRASH_1 from the CAN_ChassisGate_M1 MainBlock every 100 ms. Each time CAN_ChassisGate_M2 receives the message, the RX_Status is set to 1.
- The CAN_BodyGate_M3 MainBlock gets the RX_Status from the CAN_ChassisGate_M2 MainBlock and uses it as the trigger for transmitting the CRASH_1 message. Each time the RX_Status is set to 1, the CAN_BodyGate_M3 MainBlock sends CRASH_1 to the Body CAN bus.
- The CAN_BodyGate_M3 MainBlock then transmits CRASH_1 to the CAN_BodyGate_M4 MainBlock.

10 In the TX layout of ChassisGate_M1, specify a **Constant** value of **1** for **SRS_Indication_Lamp**.

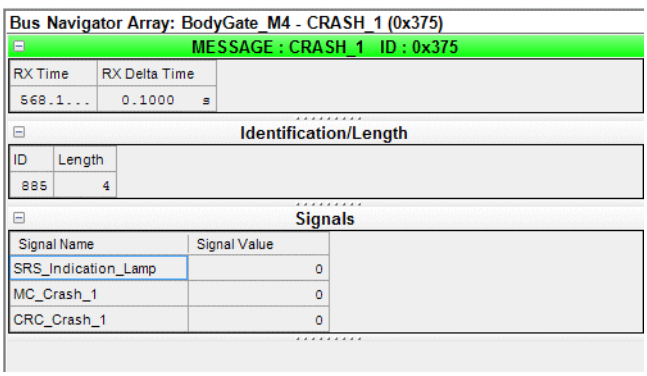
- CAN_ChassisGate_M1 transmits the new constant value of SRS_Indication_Lamp on the physical Chassis CAN bus.
- CAN_ChassisGate_M2 receives this constant value and gateways it to CAN_BodyGate_M3 via the model.

- CAN_BodyGate_M3 transmits the constant value on the physical Body CAN bus. This constant value is received by CAN_BodyGate_M4.

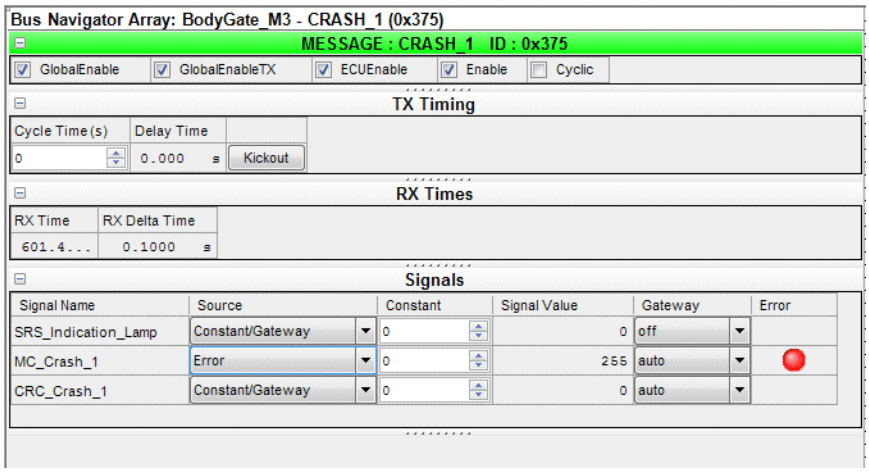
11 In the TX layout of BodyGate_M3, specify Off for the Gateway of SRS_Indication_Lamp.



The transmission of the gateway value is disabled. On the Body CAN bus, the CAN_BodyGate_M3 MainBlock transmits SRS_Indication_Lamp with the constant value specified in the TX layout of BodyGate_M3. Since this constant value is received by the CAN_BodyGate_M4 MainBlock, the corresponding RX layout displays an SRS_Indication_Lamp constant value of 0.



12 In the TX layout of BodyGate_M3, select Error from the Source drop-down list for MC_Crash_1.



- CAN_ChassisGate_M1 still transmits MC_Crash_1 with a constant value of 2 on the physical Chassis CAN bus.
- CAN_ChassisGate_M2 receives this constant value and gateways it to CAN_BodyGate_M3 via the model.
- You manipulated MC_Crash_1 before it is gatewayed to the Body CAN bus: CAN_BodyGate_M3 does not transmit the gateway value but the error value of the signal. A red LED indicates the transmission of the error value. This error value is received by CAN_BodyGate_M4.

13 On the Home ribbon, click Status Control – Go Offline to stop online calibration. Close the layouts.

Interim result

You gatewayed CRASH_1 between the Chassis and the Body CAN bus. You manipulated SRS_Indication_Lamp before it was transmitted via the Chassis CAN bus and disabled the transmission of its gateway value via the Body CAN bus. Additionally, you manipulated MC_Crash_1 before it was transmitted via the Body CAN bus and transmitted it with its error value via this bus. You can now check the behavior of STEERING_1. Continue with part 2.

Part 2

To check the behavior of the STEERING_1 message

1 Generate the following layouts for the STEERING_1 message:

CAN Bus	STEERING_1 Layout	Related MainBlock
ChassisGate_M1	TX	CAN_ChassisGate_M1
ChassisGate_M2	RX	CAN_ChassisGate_M2
BodyGate_M4	RX	CAN_BodyGate_M4

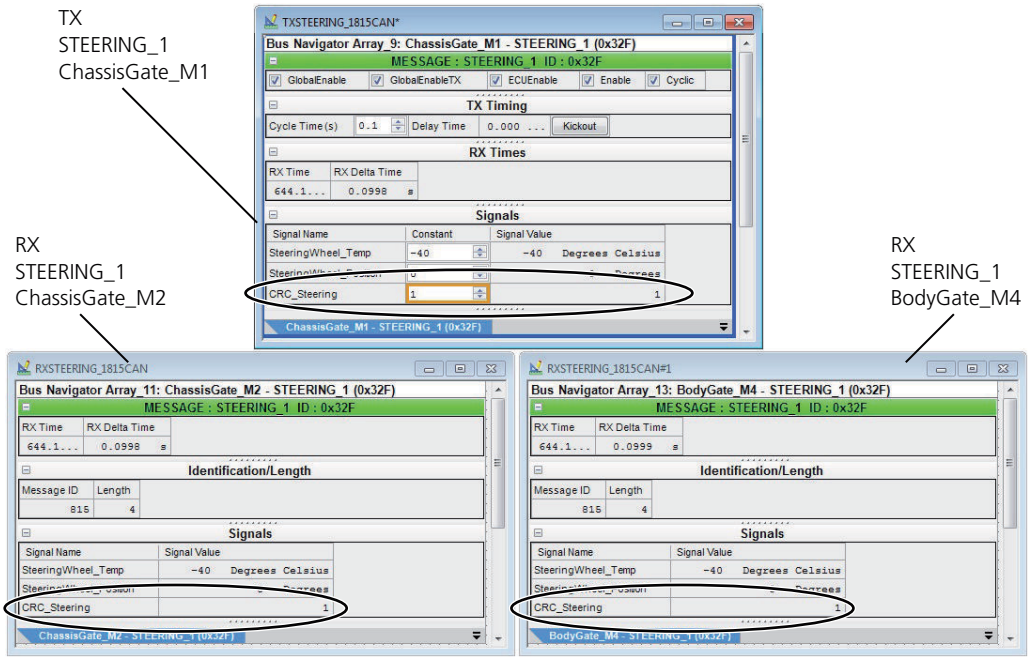
2 On the Home ribbon, click Status Control – Go Online to start online calibration.

- The CAN_ChassisGate_M1 MainBlock transmits STEERING_1 cyclically via the Chassis CAN bus.

- The CAN_ChassisGate_M2 MainBlock receives STEERING_1 but does not gateway it to the CAN_BodyGate_M3 MainBlock. Since you specified an RTICANMM Gateway block gateway between the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlocks, STEERING_1 is gatewayed via the RTICANMM Gateway block.
- The CAN_BodyGate_M4 MainBlock receives STEERING_1 from the RTICANMM Gateway block.

3 In the TX layout of ChassisGate_M1, specify a **Constant** value of **1** for **CRC_Steering**.

CAN_ChassisGate_M1 transmits CRC_Steering with a constant value of 1. Both the CAN_ChassisGate_M2 and CAN_BodyGate_M4 MainBlocks receive the manipulated signal value.

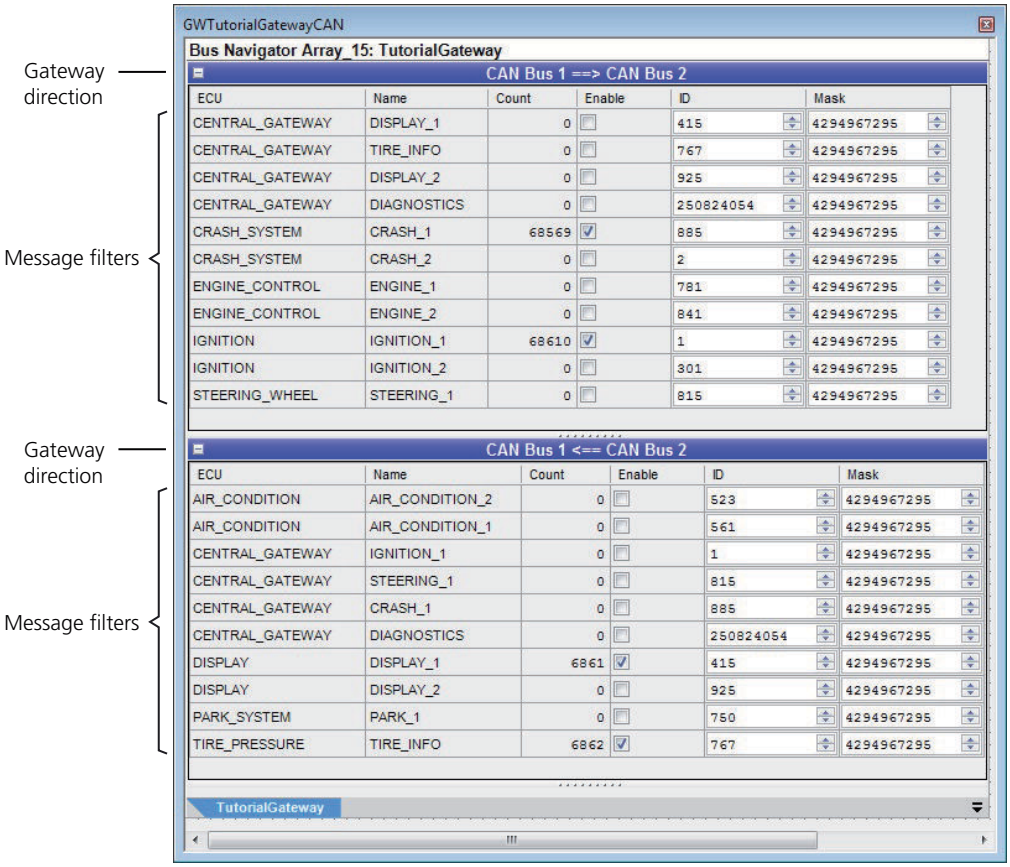


Since CAN_ChassisGate_M1 transmits STEERING_1 to CAN_ChassisGate_M2 via the physical Chassis CAN bus, you can manipulate the message before transmitting. After STEERING_1 was received by CAN_ChassisGate_M2 it is gatewayed by the RTICANMM Gateway block. Since you cannot manipulate messages when they are gatewayed by the Gateway block, you cannot manipulate STEERING_1 after it was received by CAN_ChassisGate_M2.

4 In the Bus Navigator tree, right-click the **TutorialGateway** node and select **Generate Layout** from the context menu.

The **TutorialGateway** layout opens.

The layout displays all the ECUs of the Chassis and Body CAN buses according to the gateway direction of their messages together with all their TX messages specified in the DBC files. Each row serves as a message filter for gatewaying.

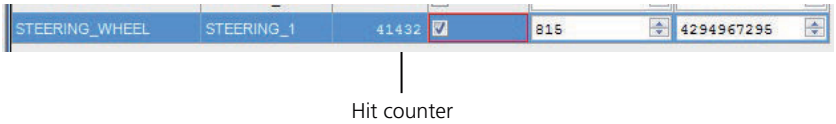


An enabled message filter in the layout indicates that the message is excluded from gatewaying via the RTICANMM Gateway block.

Since messages of the CENTRAL_GATEWAY ECU are gatewayed in both directions, the ECU is displayed twice: Once for the gateway direction from the Chassis CAN bus to the Body CAN bus and once for the opposite direction.

5 In the TutorialGateway layout, select the Enable checkbox for the STEERING_1 message of the STEERING_WHEEL ECU.

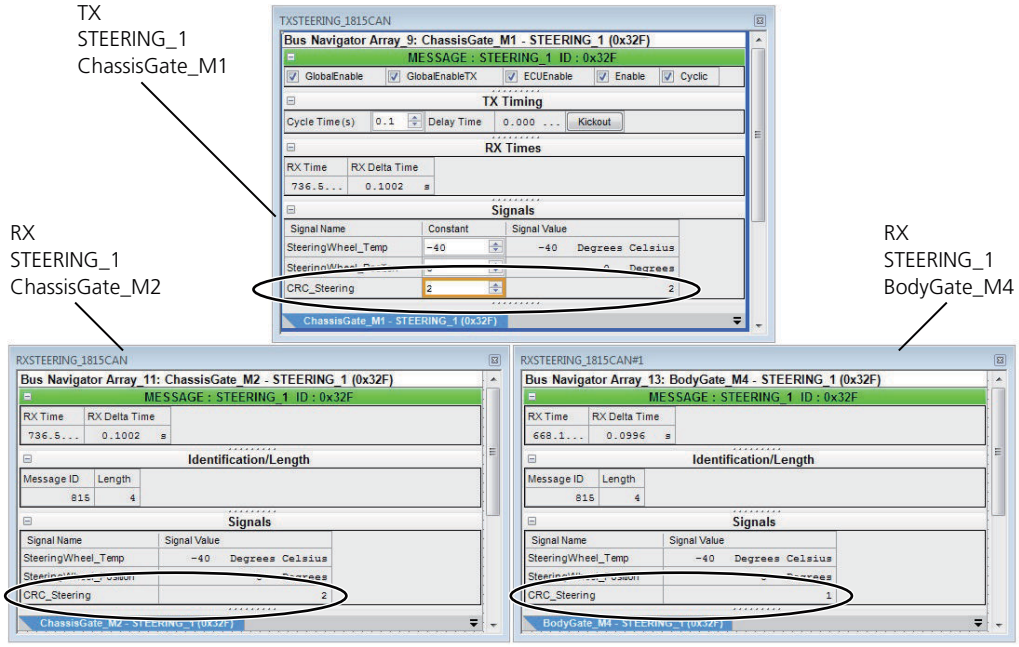
STEERING_1 is excluded from gatewaying. A hit counter counts the number of message transmissions for which the message is excluded from gatewaying. The running hit counter indicates that the message filter is active and working accurately.



STEERING_1 is not received any more by the CAN_BodyGate_M4 MainBlock. In the RX layout of BodyGate_M4, the RX time stops.

6 In the TX layout of ChassisGate_M1, specify a Constant value of **2** for CRC_Steering.

CAN_ChassisGate_M1 transmits CRC_Steering with a constant value of 2. This signal value is received by the CAN_ChassisGate_M2 MainBlock. However, it is not received by the CAN_BodyGate_M4 MainBlock, because you excluded STEERING_1 from gatewaying.



7 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

8 Save the project.

Result

You checked the gateway behavior for two messages in ControlDesk:

- CRASH_1 is gatewayed by the CAN_ChassisGate_M2 and CAN_BodyGate_M3 MainBlocks. You can manipulate this message before gatewaying and disable gatewaying via the appropriate layouts.
- STEERING_1 is transmitted from the CAN_ChassisGate_M1 MainBlock to the CAN_ChassisGate_M2 MainBlock. Between CAN_ChassisGate_M2 and CAN_BodyGate_M3, STEERING_1 is gatewayed via the RTICANMM Gateway block. Since you cannot manipulate messages which are gatewayed by the RTICANMM Gateway block, you cannot manipulate STEERING_1 before gatewaying. You can disable gatewaying for STEERING_1 via the TutorialGateway layout.

What's next

With this step, you finished the gatewaying of messages in two different ways. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

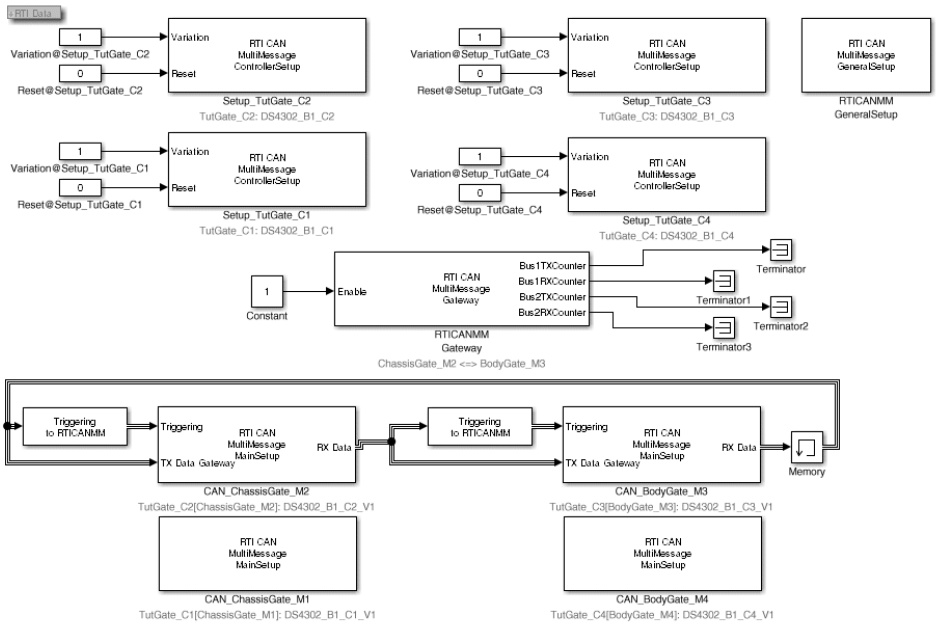
Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Result of Lesson 12

Result

You built a model to gateway messages.



In this lesson, you learned the basic principles of gatewaying messages:

General basics

- You can gateway messages between two CAN bus systems in two ways: You can gateway messages via the Simulink model using two RTICANMM MainBlocks or using an RTICANMM Gateway block.
- Using two RTICANMM MainBlocks allows you to manipulate messages and their signals before gatewaying. If you use an RTICANMM Gateway block, you cannot manipulate messages before gatewaying.
- Since additional code has to be generated, gatewaying messages using two RTICANMM MainBlocks reduces performance, so you should gateway as few messages as possible in this way. Gatewaying messages using an RTICANMM Gateway block saves resources.

Gatewaying messages using two RTICANMM MainBlocks

- To enable messages and their signals for gatewaying, you have to select them on the **RX Status and Time Ports** page and on the **Model Signals (RX)** page.
- You should enable the kickout of messages not only triggered by a rising edge of the RX_Status but by every value greater than 0. This guarantees that messages are transmitted via kickout even if they are received within milliseconds and the RX_Status is always 1.
- For the RTICANMM MainBlock that receives gateway messages, you have to select the gateway messages as TX messages and specify triggered message transmission to transmit these messages via the bus. You also have to enable message kickout for them.
- You can manipulate messages and/or manipulate signals before they are gatewayed (for example, enable error values). To manipulate messages and/or signals, you have to specify message and signal manipulation settings in the MainBlock that receives the gateway messages/signals.
- If you build a bidirectional gateway, you should prevent changes of the RX_Status by loopback messages.

Gatewaying messages using an RTICANMM Gateway block

- You have to select two RTICANMM MainBlocks or two RTICANMM ControllerSetup blocks between which messages will be gatewayed.
- Since the gateway name must be unique within the model and even within a whole multiprocessor system, you should always rename the gateway.
- You can exclude messages from gatewaying by adding them to an exclude messages list. This is useful, for example, if you want to gateway messages via two RTICANMM MainBlocks. If you want to manipulate the exclude messages list during run time, you have to enable this option on the General Options page explicitly (see also Gateway layout).

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson12_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- How to Create a Bus Instrument (Gateway Type) for a CAN Controller Gateway (ControlDesk Bus Navigator)
- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Network Node Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Network Node Preselection Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- RX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)

- RX Status and Time Ports Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Model Signals (RX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Cyclic Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Kickout Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Mapping Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Gateway Signals Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Code Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- CAN Bus Gateway Block (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to work with a J1939-compliant DBC file.

If you do not need to work with a J1939-compliant DBC file, you can also continue with another lesson. Refer to Overview of Lessons on page 35.

Related topics

Basics

Lesson 12 (Advanced): Gatewaying Messages..... 259
Overview of Lesson 12..... 260

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File

Where to go from here

Information in this section

Overview of Lesson 13 316
In this lesson, you will work with a J1939-compliant DBC file. J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles.

Step 1: How to Build a Model for Using a J1939-Compliant DBC File 320
In this step, you will build a model for working with J1939 messages.

Step 2: How to Configure the CAN-J1939 Communication of Your Model 329
In this step, you will configure the CAN-J1939 communication of your model. You will define the simulation types for the network management, instances of a broadcast and a peer-to-peer message, and the CAN communication of all MainBlocks of your model.

Step 3: How to Observe the Network Management in ControlDesk 349
In this step, you will observe the network management of your model in ControlDesk.

Step 4: How to Observe the CAN-J1939 Communication in ControlDesk 357
In this step, you will observe the CAN-J1939 communication of your model in ControlDesk. You will create a new layout in your experiment, modify the CAN communication, and observe the effects on the layouts.

Step 5: How to Gateway J1939 Messages Using the RTICANMM Gateway Block 363
In this step, you will gateway J1939 messages via an RTICANMM Gateway block.

Result of Lesson 13 369
In this lesson, you learned the basic principles of working with J1939 messages in RTICANMM.

Overview of Lesson 13

Introduction

J1939 is a vehicle CAN bus standard defined by the Society of Automotive Engineers (SAE). It is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

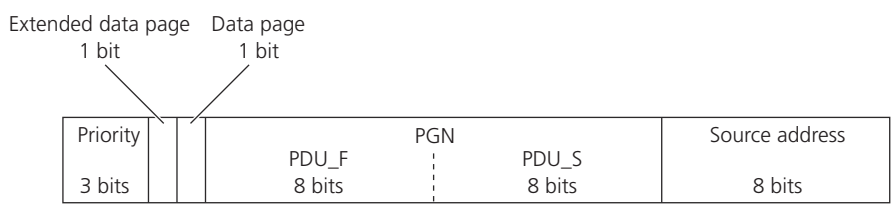
Tip

If you do not need to work with a J1939-compliant DBC file, you can skip this lesson.

Basics on J1939

CAN message identifier for J1939 Standard CAN messages use an 11-bit message identifier (CAN 2.0 A). J1939 messages use an extended 29-bit message identifier (CAN 2.0 B).

The 29-bit message identifier is split into different parts (according to SAE J1939/21 Data Link Layer):



29-bit CAN message ID

Parameter group number (PGN) A 16-bit number in the 29-bit message identifier of a CAN message defined in a J1939-compliant DBC file. Each PGN references a *parameter group* that groups parameters and assigns them to the 8-byte data field of the message. A parameter group can be the engine temperature including the engine coolant temperature, the fuel temperature, etc. PGNs and parameter groups are defined by the SAE (see SAE J1939/71 Vehicle Application Layer).

The first 8 bits of the PGN represent the *PDU_F* (Protocol Data Unit format). The PDU_F value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU_S* value (PDU-specific) depends on the PDU_F value. For messages with a PDU_F < 240 (peer-to-peer communication, also called PDU1 messages), PDU_S is not relevant for the PGN, but contains the destination address of the network node that receives the message. For

messages with a PDU_F ≥ 240 (broadcast messages, also called PDU2 messages), PDU_S specifies the second 8 bits of the PGN and represents the group extension. A group extension is used to increase the number of messages that can be broadcast in the network.

PDU_F (first 8 bits)	PDU_S (second 8 bits)	Communication Mode
< 240	Destination address	Peer-to-peer (message is transmitted to one destination network node)
≥ 240	Group extension	Broadcast (message is transmitted to any network node connected to the network)

CAN-J1939 communication of the RTI CAN MultiMessage Blockset

Container and instance messages The RTI CAN MultiMessage Blockset distinguishes between container and instance messages:

- A container message is a J1939 message defined by its PGN. A container can receive all the messages with its PGN in general. If several messages are received in one sampling step, only the last received message is held in the container.
- An instance message is a J1939 message defined by its PGN, for which the source network node and the destination network node (only for peer-to-peer communication) are defined.

There exists one container for each PGN (except for proprietary PGNs).

Source/destination mapping For each message defined in a J1939-compliant DBC file, the source/destination mapping specifies the source (transmitting) network node and the destination (receiving) network node (destination only for peer-to-peer messages).

Without source/destination mapping, if two or more network nodes sent the same PGN in the same sampling step, these messages would be placed in a common container message. As a result only the J1939 message received last in that sampling step is accessible via model and/or TRC file. With source/destination mapping, the two J1939 messages can be treated individually as separate instances of the message. Additionally, you can define instances to be shown in the container. In this case, the container contains the last received J1939 instances.

Messages in a J1939-compliant DBC file that are described only by the PGN have no source/destination mapping. Messages that are described by the CAN message ID (consisting of the priority, PGN, and source address) have source/destination mapping. The RTI CAN MultiMessage Blockset lets you specify user-defined source/destination mapping for messages that are described only by the PGN.

CAN-J1939 communication in this lesson

In contrast to the other lessons of this tutorial, this lesson is not based on a use scenario. It would take too long to build a CAN-J1939 communication that covers a use scenario, and only a few elements would deal with J1939-relevant settings in RTICANMM.

This lesson focuses on handling J1939 messages in RTICANMM. You will work with a simple DBC file that defines five network nodes, and a few broadcast and peer-to-peer messages. J1939 messages are named according to a specific schema: PG_<PGN>_<Length>[_<Source>[_<Destination>]].

Message	Source Network Node	Destination Network Node
PG_0000_08_Node1_Node4	Node1	Node4
PG_1100_22	–	–
PG_1100_22_Node4_Node1	Node4	Node1
PG_1100_22_Node4_Node3	Node4	Node3
PG_F000_08_Node4	Node4	All
PG_F001_08	–	–
PG_F001_08_Node2	Node2	All
PG_FEFF_22_Node1	Node1	All

What you will learn

In this lesson, you will learn the basic principles of working with a J1939-compliant DBC file in RTICANMM.

- You will build a simple model and use a J1939-compliant DBC file for it.
- You will define instances of a broadcast and a peer-to-peer message.
- You will configure the J1939 network management and observe the effects in ControlDesk.
- You will configure the CAN-J1939 communication and observe the effects in ControlDesk.
- You will gateway J1939 messages and observe the gateway behavior in ControlDesk.

Starting point

The starting point of this lesson is a new model in MATLAB.

Prepared demo

Since you will build a new model in this lesson, no prepared demo is required.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- The J1939 support for the RTI CAN MultiMessage Blockset requires a separate license. This license must be available and activated.
- You are familiar with the basic principles of the J1939 CAN bus standard.
- You need four CAN controllers to work with the application in ControlDesk.
- The CAN channels of your dSPACE real-time hardware are physically connected as follows:
  - Channel 1 is connected to channel 2.
  - Channel 3 is connected to channel 4.

- You have to be familiar with building a model using RTICANMM blocks (refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39).
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.
- You have to be familiar with the basic principles of creating a new layout in ControlDesk.

Steps

This lesson contains the following steps:

- Step 1: How to Build a Model for Using a J1939-Compliant DBC File on page 320
- Step 2: How to Configure the CAN-J1939 Communication of Your Model on page 329
- Step 3: How to Observe the Network Management in ControlDesk on page 349
- Step 4: How to Observe the CAN-J1939 Communication in ControlDesk on page 357
- Step 5: How to Gateway J1939 Messages Using the RTICANMM Gateway Block on page 363

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 13 on page 369.

Duration

Working through this lesson will take you about 120 minutes.

What's next

Now you can start to build a model for working with J1939 messages, see Step 1: How to Build a Model for Using a J1939-Compliant DBC File on page 320.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315

Step 1: How to Build a Model for Using a J1939-Compliant DBC File

Objective

Using a J1939-compliant DBC file does not affect the general procedure for building a model. Since you will gateway messages in this lesson, you need four CAN controllers and one RTICANMM Gateway block:

- One CAN controller transmits and receives messages via a physical CAN bus.
- One CAN controller transmits and receives messages via a second physical CAN bus.
- One RTICANMM MainBlock represents the network nodes Node1 ... Node3 connected to the first physical CAN bus.
- One RTICANMM MainBlock represents the network nodes Node4 ... Node5 connected to the second physical CAN bus.
- Two CAN controllers gateway the messages between the two physical CAN buses. These CAN controllers will be configured by the RTICANMM Gateway block.

  One of these CAN controllers is additionally used to send a message 'in the name of' a really existent network node.
- One RTICANMM MainBlock simulates the "real" network node.

In this step, you will build a new model with four CAN controllers, three RTICANMM MainBlocks and one RTICANMM Gateway block.

Part 1

To create a model and add RTICANMM blocks to it

1. In your working folder, create a new folder named **`RTICANMMTutorial_Lesson13`**.
2. In MATLAB, change the Current Directory to your RTICANMMTutorial_Lesson13 folder.

   Tip

   Save all files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all files to one folder also includes changing the path of the Current Directory in MATLAB.

3. In the MATLAB Command Window, type **`rticanmm_tutorial`** to open the rticanmm_tutorial_lib library.
4. From the rticanmm_tutorial_lib library, double-click the j1939_tutorial.dbc file.

   The j1939_tutorial.dbc file is automatically copied to your RTICANMMTutorial_Lesson13 folder.
5. Close the rticanmm_tutorial_lib library.
6. Open a new model and save it as **`TutorialJ1939`** in your RTICANMMTutorial_Lesson13 folder.

7 In the MATLAB Command Window, type **`rticanmm`** to open the rticanmmlib library.

8 From the rticanmmlib library, add
- One RTICANMM GeneralSetup block,
- Four RTICANMM ControllerSetup blocks,
- Three RTICANMM MainBlocks, and
- One RTICANMM Gateway block to your model.

Note

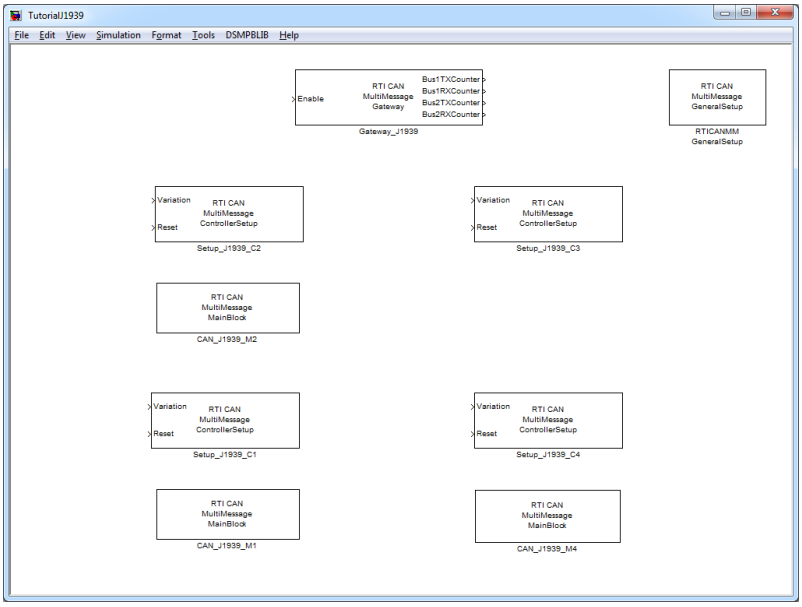
Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

9 Rename the four RTICANMM ControllerSetup blocks as follows:
- **`Setup_J1939_C1`**
- **`Setup_J1939_C2`**
- **`Setup_J1939_C3`**
- **`Setup_J1939_C4`**

10 Rename the RTICANMM MainBlocks as follows:
- **`CAN_J1939_M1`**
- **`CAN_J1939_M2`**
- **`CAN_J1939_M4`**

11 Rename the RTICANMM Gateway block **`Gateway_J1939`**.

12 For easier model handling, arrange the RTICANMM blocks as shown in the illustration below.



Interim result

You built a new model and added RTICANMM blocks to it. Additionally, you renamed the RTICANMM ControllerSetup blocks, the RTICANMM MainBlocks and the RTICANMM Gateway block. You can now parameterize the RTICANMM ControllerSetup blocks. Continue with part 2.

Part 2

To parameterize the RTICANMM ControllerSetup blocks of your model

1 Double-click the Setup_J1939_C1 ControllerSetup block to open it.

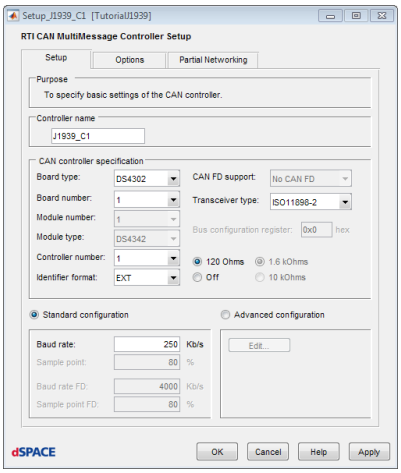
2 Specify **J1939_C1** as the Controller name.

> **Note**
>
> The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.

3 From the Board type drop-down list, select the board you are working with.

4 From the Identifier format drop-down list, select EXT.

5 In the Standard configuration frame, specify a Baud rate of **250** Kb/s.

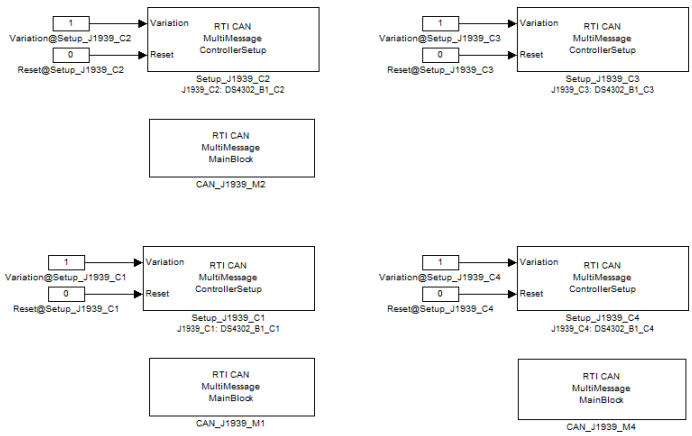


6 Click OK to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.

7 Change the settings of the other RTICANMM ControllerSetup blocks as follows:

RTICANMM ControllerSetup Block	Controller Name	Board Type	Controller Number	Identifier Format	Baud Rate [kbit/s]
Setup_J1939_C2	J1939_C2	Select the board type according to the hardware you are working with.	2	EXT	250
Setup_J1939_C3	J1939_C3		3	EXT	250
Setup_J1939_C4	J1939_C4		4	EXT	250



Interim result

You parameterized the RTICANMM ControllerSetup blocks of your model. Since J1939 requires the 29-bit identifier, you specified the extended CAN identifier format for the CAN controllers. You also specified a baud rate of 250 Kb/s which is usually used in connection with J1939. You can now parameterize the RTICANMM MainBlocks. Continue with part 3.

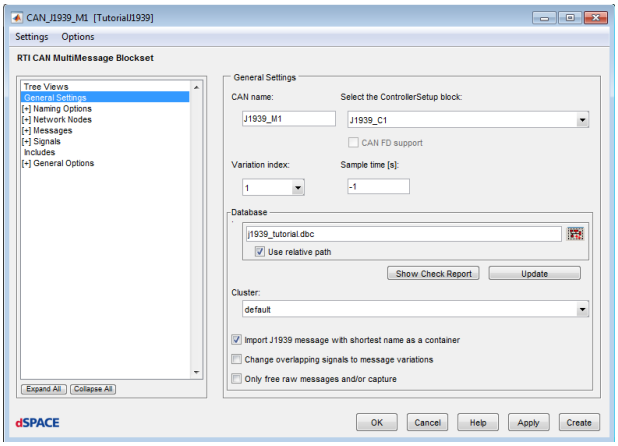
Part 3

To parameterize the RTICANMM MainBlocks of your model

1 Double-click the CAN_J1939_M1 MainBlock to open it.

2 Select the General Settings page, and specify the following settings:

Setting	Entry / Selection
CAN name	J1939_M1
Select the ControllerSetup block	J1939_C1
Database	j1939_tutorial.dbc

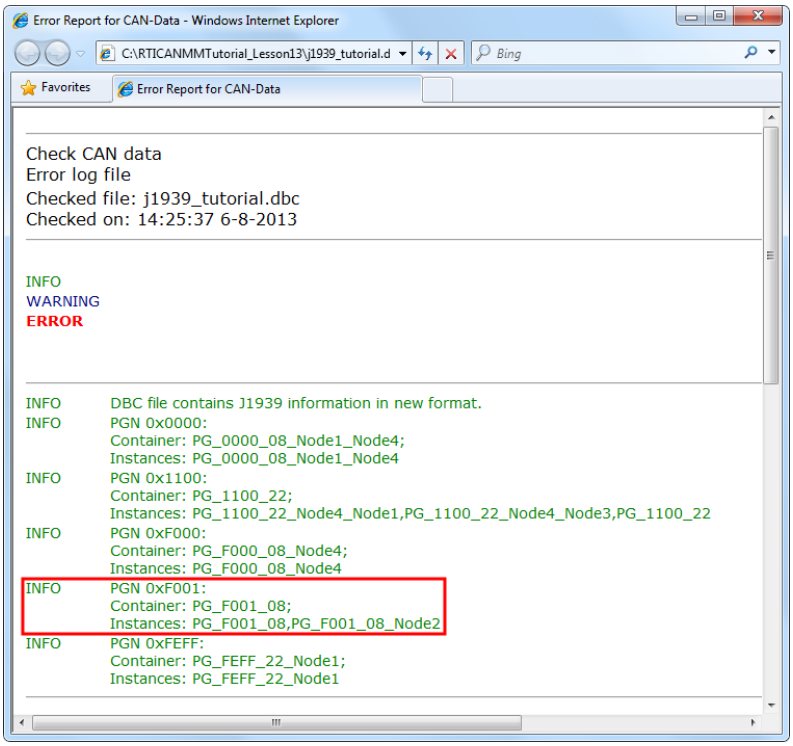


After the database is loaded, some container and instance messages are well prepared.

Example: The following messages are available for the PGN "0xF001":

- Container message: PG_F001_08
- Instances: PG_F001_08 and PG_F001_08_Node2

The configuration is extended during the setup of the communication for the CAN_J1939_M1 block.

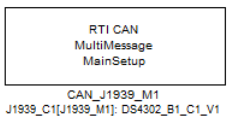


Tip

Click Show Check Report to open the check report.

3 Click Create.

The CAN_J1939_M1 MainBlock is created. All the files needed for your RTICANMM communication, such as an S-function, are created automatically.



4 Click OK to close the CAN_J1939_M1 [TutorialJ1939] dialog.

5 Specify the general settings of the CAN_J1939_M4 MainBlock as follows:

Setting	Entry / Selection
CAN name	J1939_M4
Select the ControllerSetup block	J1939_C4
Database	j1939_tutorial.dbc

6 Click **Create** to create the necessary files.

7 Close the dialog.

8 Specify the general settings of the **CAN_J1939_M2** MainBlock as follows:

Setting	Entry / Selection
CAN name	J1939_M2
Select the ControllerSetup block	J1939_C2
Database	j1939_tutorial.dbc

9 Click **Create** to create the necessary files.

10 Close the dialog.

Interim result

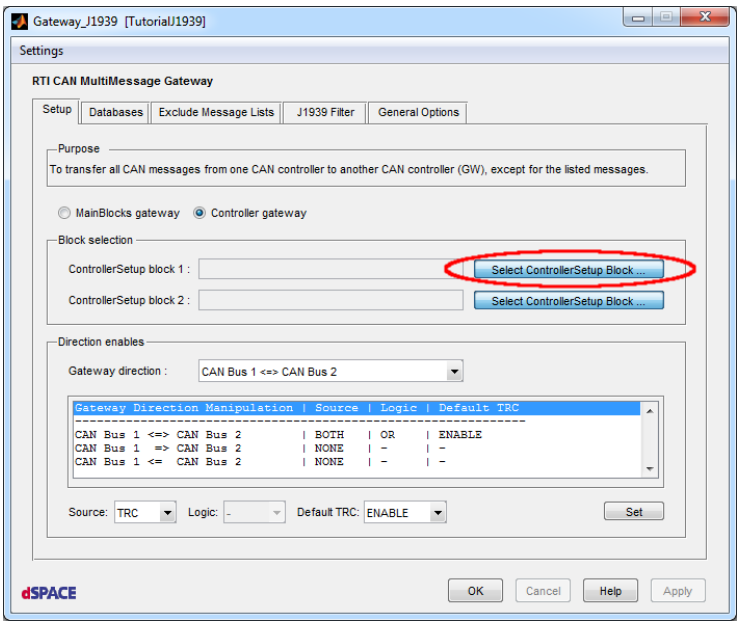
You parameterized the RTICANMM MainBlocks of your model. You can now parameterize general settings of the RTICANMM Gateway block. Continue with part 4.

Part 4

To parameterize general settings of the RTICANMM Gateway block

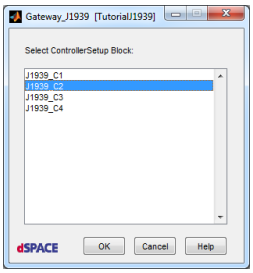
1 Double-click the RTICANMM Gateway block to open it.

2 On the **Setup** page, click **Select ControllerSetup Block** in the **ControllerSetup block 1** frame.



A dialog opens showing all the ControllerSetup blocks available in your model.

3 Select J1939_C2 and click OK.

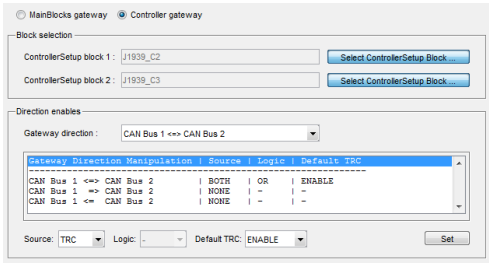


The dialog closes and the J1939_C2 ControllerSetup block is displayed in the ControllerSetup block 1 field.

4 In the ControllerSetup block 2 frame, click Select ControllerSetup Block.

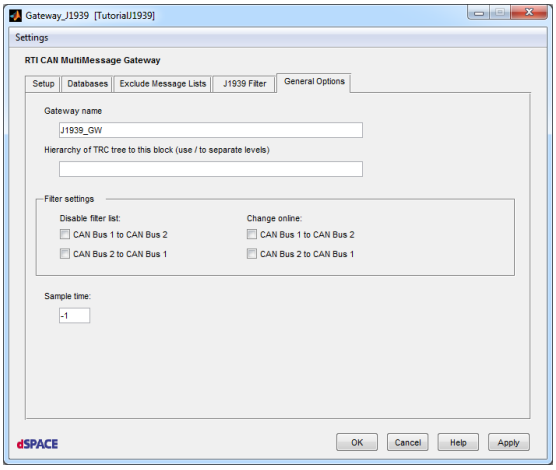
5 From the dialog, select J1939_C3 and click OK.

The dialog closes and the J1939_C3 ControllerSetup block is displayed in the ControllerSetup block 2 field.



6 Select the General Options page.

7 Change the Gateway name to `J1939_GW`.

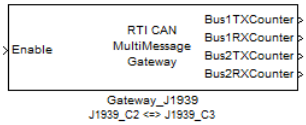


Note

The gateway name must be unique within the model and even within a whole multiprocessor system, so you should always rename the gateway.

8 Click OK.

The dialog closes and an S-function is created automatically. The RTICANMM Gateway block displays the gateway direction.



Interim result

You parameterized general settings of the RTICANMM Gateway block. You can now connect its inport and outports to Simulink blocks. Continue with part 5.

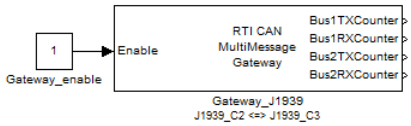
Part 5

To connect the inport and outports of the RTICANMM Gateway block

1 Open the Simulink Library Browser.

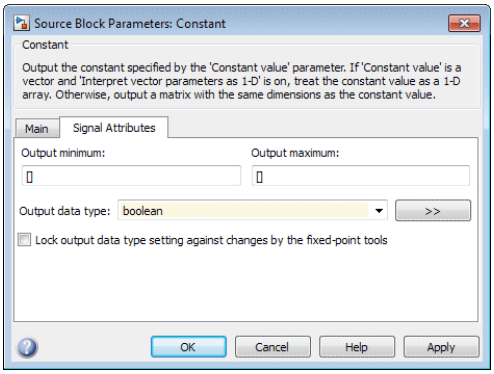
2 Select Commonly Used Blocks.

3 Drag a Constant block to your model, rename it **`Gateway_enable`** and connect it with the Enable inport of the RTICANMM Gateway block.



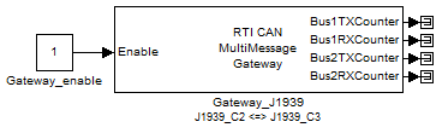
4 Double-click the Constant block to open its property dialog and select the Signal Attributes page.

5 From the Output data type drop-down list, select boolean.



6 Click OK to close the dialog.

7 From the Commonly Used Blocks, drag four Terminator blocks to your model and connect them to the outports of the RTICANMM Gateway block.



Tip

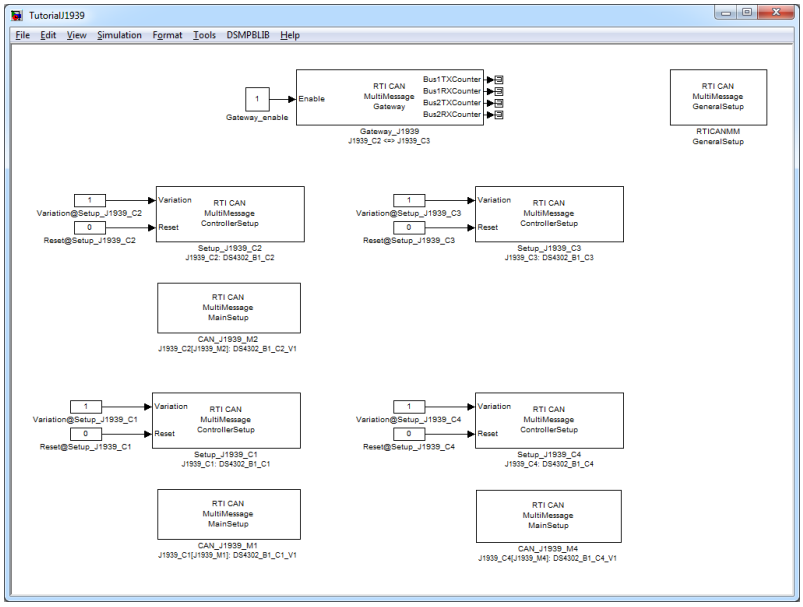
Alternatively, you can type `addterms(gcs)` in the MATLAB Command Window. This automatically adds terminators to all the unconnected block outputs in the model.

8 Close the Simulink Library Browser.

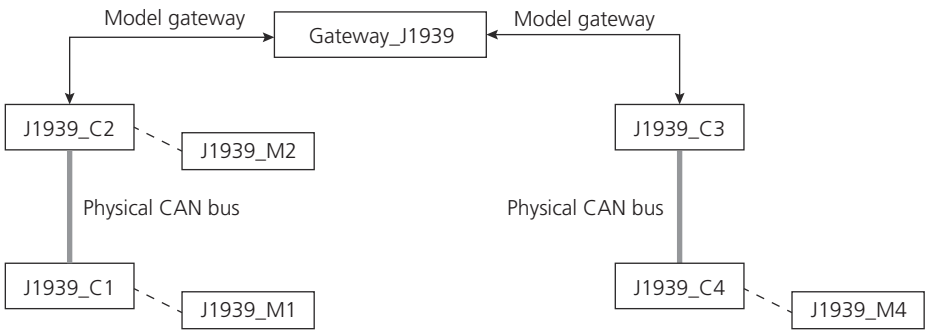
9 Save your model.

Result

The illustration below displays the result of step 1.



You built a model with four CAN controllers and prepared it for gatewaying messages. Since you will only gateway messages via the RTICANMM Gateway block in this lesson, no further RTICANMM MainBlocks are required for the gateway. The CAN_J1939_M1 MainBlock configures the CAN communication on the J1939_C1 CAN controller (network nodes Node1, Node2, Node3). The CAN_J1939_M4 MainBlock configures the CAN communication on the J1939_C4 CAN controller (network nodes Node4, Node5). The J1939_C2 and J1939_C3 CAN controllers realize gatewaying for all network nodes. The CAN_J1939_M2 MainBlock configures the simulated CAN communication on the J1939_C2 CAN controller. The CAN_J1939_M2 MainBlock is used to send a message 'in the name of' Node4 without activated network management.



What's next

In the next step, you can configure the CAN-J1939 communication of your model.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Step 2: How to Configure the CAN-J1939 Communication of Your Model

Objective

An RTICANMM MainBlock provides specific property pages for configuring J1939 messages. Via an RTICANMM MainBlock, you can configure the network management, define instances of a message and enable to show instances in a container message.

The *simulation type* of a network node specifies whether the network node is to be completely simulated by the RTI CAN MultiMessage Blockset, or is an external network node whose network management you want to observe and you want to communicate with, or is not relevant.

- A *simulated* network node is completely simulated by the RTI CAN MultiMessage Blockset. RTICANMM performs network management for this network node.
- An *external* network node is a real network node that is externally connected to the J1939 network. The network node is not simulated by the RTI CAN MultiMessage Blockset, but RTICANMM detects the network address of the network node.
- A network node with *non-relevant* simulation type is neither really existent in the J1939 network nor simulated by the RTI CAN MultiMessage Blockset.

In this step, you will define the simulation types for the network management, instances of a broadcast and a peer-to-peer message, and the CAN communication of all MainBlocks of your model.

Part 1

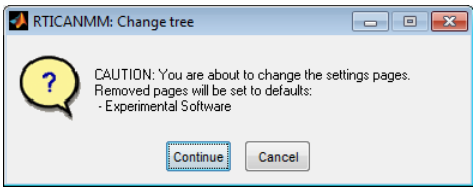
To configure the network management for the CAN_J1939_M1 MainBlock

1 Double-click the CAN_J1939_M1 MainBlock to open it.

2 On the Tree Views page, select Filter Configuration Tree by Feature.

3 Clear all filters except for the Instrumentation & Automation Interface, Manipulation Interface, and J1939 filters.

If a dialog similar to the following one is displayed, click Continue.



4 In the dialog tree, select Network Nodes – J1939 Network Node Configuration – Network Node Selection.

On the Network Node Selection page, you can specify the simulation types of the imported network nodes.

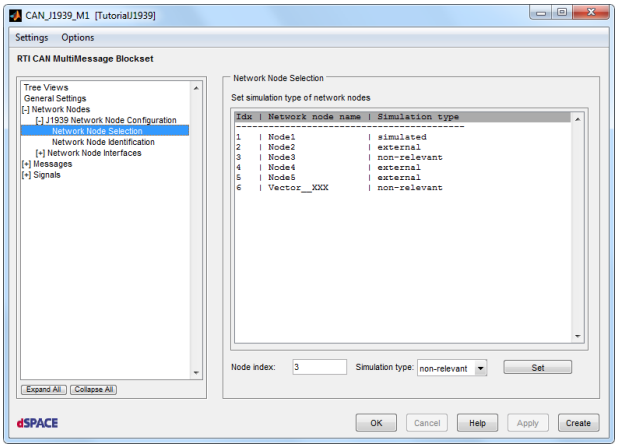
5 Specify the following settings:

Network Node	Simulation Type
Node1	Simulated
Node2	External
Node3	Non-relevant
Node4	External
Node5	External
Vector_XXX	Non-relevant

Note

You always have to click Set to apply your settings.

The CAN_J1939_M1 MainBlock represents network nodes Node1, Node2 and Node3, so these three network nodes are active. Node1 simulates the network management. Node4 and Node5 are not used for communication in this block, they are configured externally. The Vector_XXX network node is for unmapped messages only and will not be used.

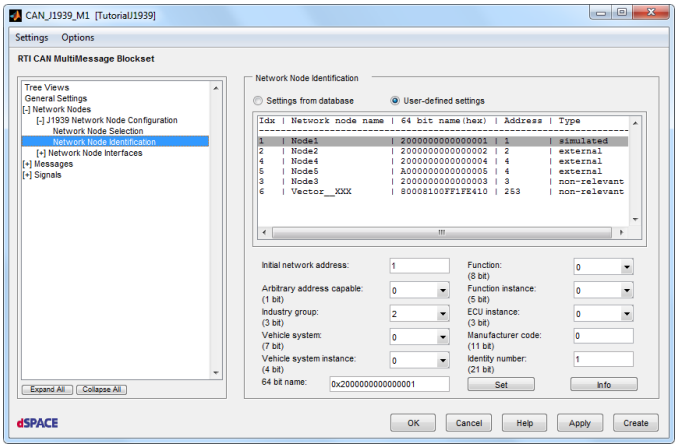


6 In the dialog tree, select Network Nodes – J1939 Network Node Configuration – Network Node Identification.

On the Network Node Identification page, you can specify node addresses and 64-bit names of network nodes for identification purposes. The 64-bit name is used to identify the network nodes and must therefore be unique for each network node.

7 Select User-defined settings.

The list containing the network node identification values is enabled. You can see the network node identification values for each network node.



Tip

Since all the network node identification values are suitably defined in the database, you do not have to change any settings on this page.

8 Select Settings from database again.

9 In the dialog tree, select Network Nodes – Network Node Interfaces – Network Node Enable.

10 Clear the Use these options for all checkbox.

The Specify enables for all network nodes list is enabled.

11 Specify the following settings for Node1:

Setting	Selection
Source	TRC
Logic	–
Default source	TRC
Default	ENABLE

12 Click Set.

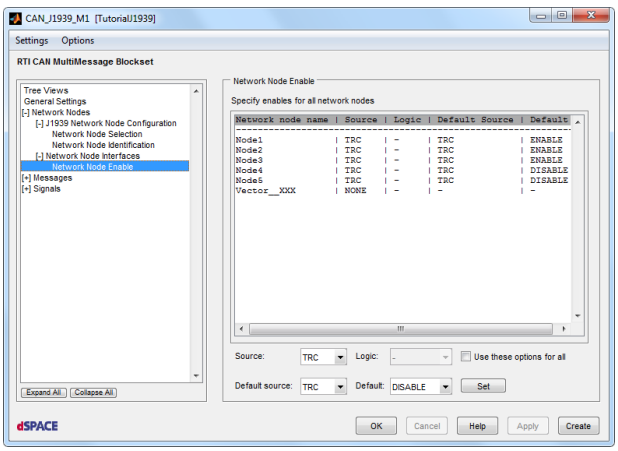
> **Note**
>
> You always have to click Set to apply your settings.

You specified to enable or disable the Node1 network node via ControlDesk during run time. You enabled Node1 by default.

13 Specify the following settings for the other network nodes:

Network Node	Source	Logic	Default Source	Default
Node2	TRC	–	TRC	ENABLE
Node3	TRC	–	TRC	ENABLE
Node4	TRC	–	TRC	DISABLE
Node5	TRC	–	TRC	DISABLE
Vector_XXX	None	–	–	–

You enabled Node2 and Node3 by default and disabled Node4 and Node5 by default. These default values can be changed via ControlDesk during run time. Since Vector_XXX is not relevant in this tutorial, it is disabled and cannot be enabled during run time.



Interim result

You configured the network management for the CAN_J1939_M1 MainBlock. You specified Node1 to be completely simulated by the RTI CAN MultiMessage Blockset and Node2 to be externally connected to the J1939 network. You specified the 'non-relevant' simulation type for Node3, which means that it is neither really existent in the J1939 network nor simulated by the RTI CAN

MultiMessage Blockset. You further enabled network nodes **Node1**, **Node2** and **Node3** by default and specified that they can be enabled or disabled via ControlDesk during run time.

You can now create instances of J1939 messages for the block. Continue with part 2.

Part 2

To create instances of J1939 messages for the CAN_J1939_M1 MainBlock

1 In the dialog tree, select **Messages – J1939 – Source Destination Mapping**.

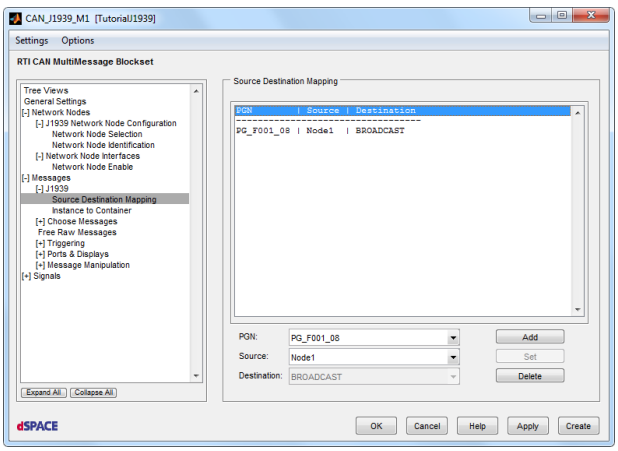
On the Source Destination Mapping page, you can specify user-defined mappings of the source network node and destination network node for J1939 messages. The name of a user-defined instance is generated automatically and has the syntax `<container name>_<source node>_<destination node>` (destination node only for peer-to-peer messages).

2 Select **PG_F001_08** from the **PGN** list and **Node1** from the **Source** list.

> **Tip**
>
> Since PG_F001_08 is specified as the PGN for a broadcast message in the DBC file and broadcast messages are sent to all the network nodes in a network, the **Destination** list is disabled.

3 Click **Add**.

You created a user-defined source/destination mapping for the PG_F001_08 message with Node1 as the source network node.



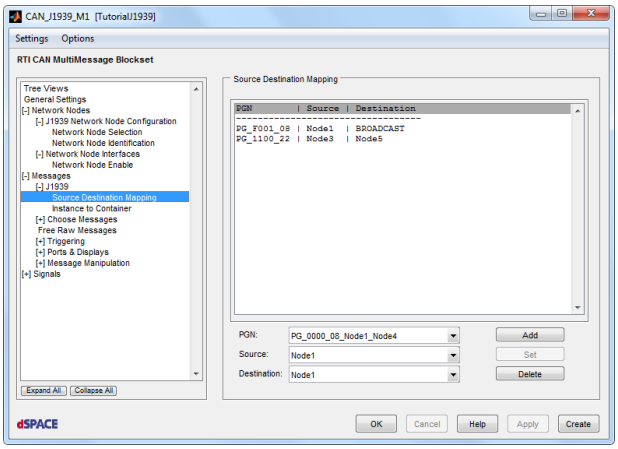
4 From the **PGN** drop-down list, select **PG_1100_22**.

Since the PG_1100_22 PGN belongs to a peer-to-peer message, the **Destination** network node list is enabled.

5 Specify **Node3** as the **Source** and **Node5** as the **Destination**.

6 Click Add.

You created an instance of PG_1100_22 with Node3 as the source and Node5 as the destination network node.



Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You specified instances of the PG_F001_08 and PG_1100_22 messages as follows:

Instance Message	Source Network Node	Destination Network Node
PG_F001_08_Node1	Node1	All
PG_1100_22_Node3_Node5	Node3	Node5

You can now configure the CAN communication of this MainBlock. Continue with part 3.

Part 3

To configure the messages of the CAN_J1939_M1 MainBlock

1 In the dialog tree, select Messages – Choose Messages – Network Node Preselection.

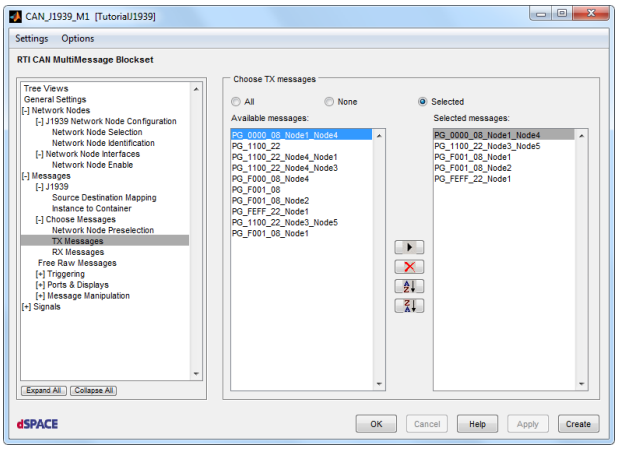
2 In the Available network nodes list, select Node1, Node2 and Node3 and click .

Node1, Node2 and Node3 are displayed in the Selected Network Nodes list. This selection can be used for further message setup.

3 In the dialog tree, select Messages – Choose Messages – TX Messages.

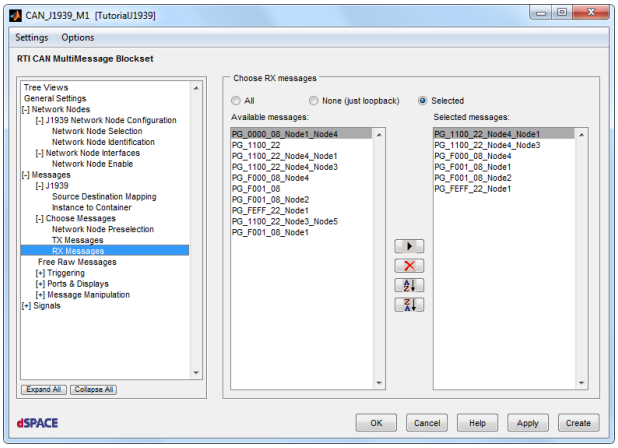
4 In the Selected messages list, right-click and select Choose Messages Transmitted by Selected Network Nodes from the context menu.

All the messages transmitted by **Node1**, **Node2** and **Node3** are added to the **Selected messages** list. Only these messages will be transmitted by this MainBlock.



5 In the dialog tree, select the **Messages – Choose Messages – RX Messages** page.

6 In the **Selected messages** list, right-click and select **Choose Messages Received by Selected Network Nodes** from the context menu.



The displayed messages are received by **Node1**, **Node2** and **Node3**.

7 In the dialog tree, select **Messages – Triggering – Message Enable**.

8 Specify the following settings:

Setting	Selection
Source	TRC
Default	DISABLE

Disabling all the messages by default helps you to understand the behavior of instance and container messages in ControlDesk. You can enable the messages during run time via ControlDesk. Since the **Specify enables for all messages** list is disabled, it is not updated.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

9 In the dialog tree, select Messages – Message Manipulation – J1939 TX ID.

All the J1939 messages selected on the TX Messages page are displayed.

On the J1939 TX ID page, you can enable the manipulation of the source node and destination node of J1939 TX messages and the message priorities via the model and/or via ControlDesk. Additionally, you can specify whether to address the source and destination nodes of each individual J1939 message by their network addresses or their node indices.

10 In the Configured J1939 ID manipulations list, select PG_F001_08_Node1 and specify the following settings for it:

Setting	Selection
Priority	TRC
Source/Destination	TRC
Node select via	ADDRESS

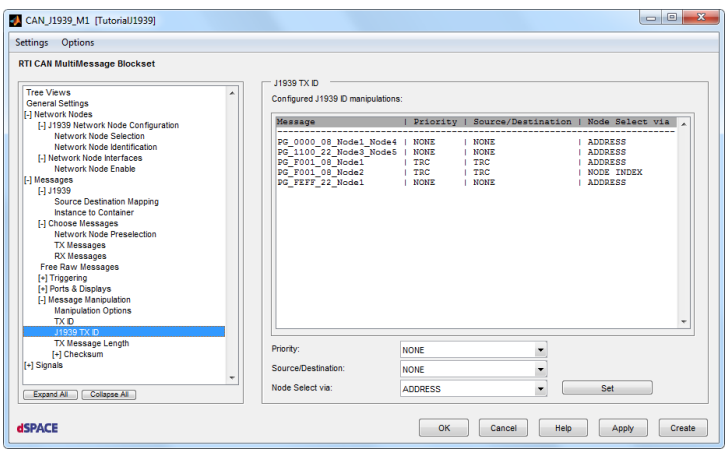
11 Click Set.

Note

You always have to click Set to apply your settings.

12 Select PG_F001_08_Node2, and specify the following settings for it:

Setting	Selection
Priority	TRC
Source/Destination	TRC
Node select via	NODE INDEX



You enabled to manipulate the priority, the source and the destination of two J1939 message instances during run time via ControlDesk. The source and destination nodes of one message instance are addressed via their network addresses, those of another message instance via their node indices.

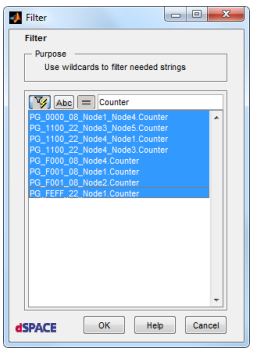
13 In the dialog tree, select Signals – TX – Signal Manipulation – Counter.

14 Click Filter.

The Filter dialog opens for you to filter signals by their names.

15 Enter **Counter** in the dialog's edit field and click to activate the wildcard filter.

16 Select all the displayed messages.



17 Click OK.

The Filter dialog is closed and the signals are added to the User-defined counter signals list on the Counter page.

18 Click Create.

The necessary files are created automatically, and the specified settings apply to your model.

19 Click OK to close the CAN_J1939_M1 [TutorialJ1939] dialog.

Interim result

You configured the CAN communication of the CAN_J1939_M1 MainBlock. You can now configure the network management of the CAN_J1939_M2 MainBlock. Continue with part 4.

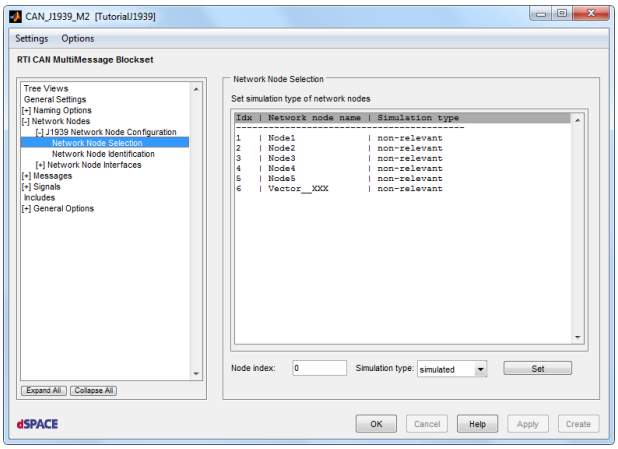
Part 4

To configure the network management for the CAN_J1939_M2 MainBlock

1 Double-click the CAN_J1939_M2 MainBlock to open it.

2 In the dialog tree, select Network Nodes – J1939 Network Node Configuration – Network Node Selection.

3 Make sure that the simulation type is set to non-relevant for all network nodes.



4 In the dialog tree, select Network Nodes – Network Node Interfaces – Network Node Enable.

5 Clear the Use these options for all checkbox.

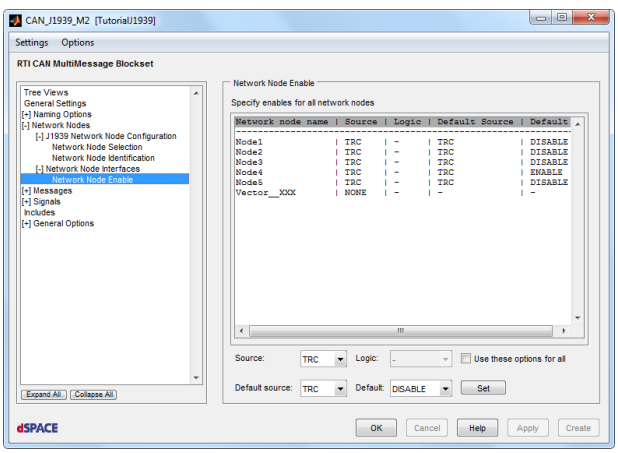
The Specify enables for all network nodes list is enabled.

6 Specify the following settings for the network nodes:

Network Node	Source	Logic	Default Source	Default
Node1	TRC	–	TRC	DISABLE
Node2	TRC	–	TRC	DISABLE
Node3	TRC	–	TRC	DISABLE
Node4	TRC	–	TRC	ENABLE
Node5	TRC	–	TRC	DISABLE
Vector_XXX	None	–	–	–

Note

You always have to click Set to apply your settings.



Interim result

You configured the network management for the CAN_J1939_M2 MainBlock. You can now configure the CAN communication of the CAN_J1939_M2 MainBlock. Continue with part 5.

Part 5

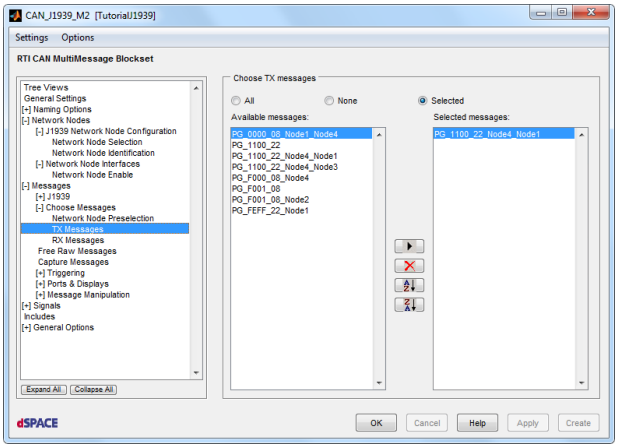
To configure the messages of the CAN_J1939_M2 MainBlock

1 In the dialog tree, select Messages – Choose Messages – TX Messages.

2 Select Selected.

3 In the Available messages list, select the PG_1100_22_Node4_Node1 message and click .

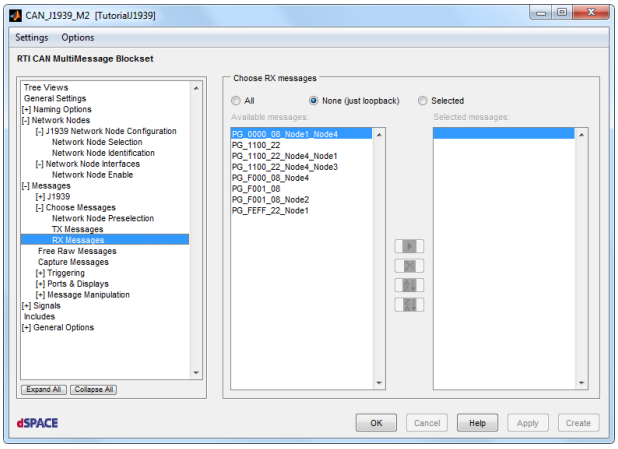
The selected message is added to the Selected messages list.



4 In the dialog tree, select the Messages – Choose Messages – RX Messages page.

5 Select None (just loopback).

This MainBlock is only used to transmit messages. You can therefore disable the reception of messages. However, all the messages transmitted by a MainBlock are always received by the same MainBlock as loopback messages. You cannot disable the reception of loopback messages.



6 In the dialog tree, select Messages – Triggering – Message Enable.

7 Specify the following settings:

Setting	Selection
Source	TRC
Default	DISABLE

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Since the Use these options for all checkbox is selected, the specified settings apply to all messages.

8 In the dialog tree, select Messages – Triggering – Message Cyclic.

9 Specify the following settings:

Setting	Selection
Source	TRC
Default	TRIGGERED

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

You specified triggered message transmission by default for the messages, and you specified to include entries in the TRC file to modify these default settings via ControlDesk.

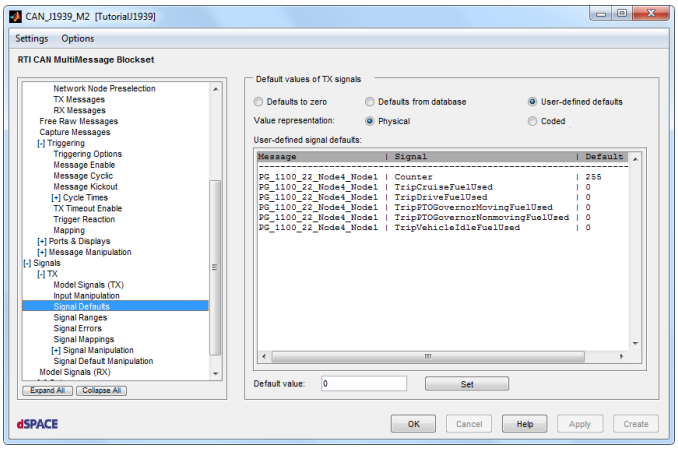
10 In the dialog tree, select Signals – TX – Signal Defaults.

On the Signal Defaults page, you can specify default values for the signals to be transmitted.

11 Select User-defined defaults.

This allows you to set default signal values individually for each TX signal.

12 Select the Counter signal of the PG_1100_22_Node4_Node1 message, and enter **255** as its default value.



Note

You always have to click Set to apply your settings.

13 In the dialog tree, select Signals – TX – Signal Manipulation – Counter.

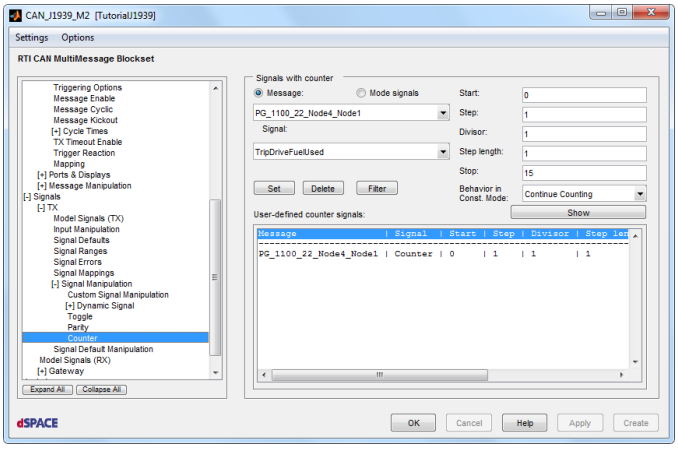
14 On the Counter page, click Filter.

The Filter dialog opens for you to filter signals by their names.

15 Enter **`Counter`** in the dialog's edit field and click to activate the wildcard filter.

16 Select all the displayed messages and click OK.

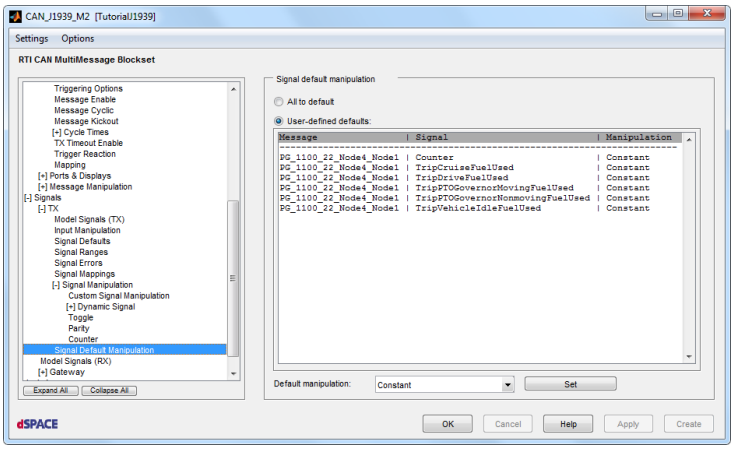
The Filter dialog is closed and the signals are added to the User-defined counter signals list on the Counter page.



17 In the dialog tree, select Signals – TX – Signal Default Manipulation.

18 Select User-defined defaults.

19 Select the Counter signal of the PG_1100_22_Node4_Node1 message and select Constant as the default signal manipulation.



20 Click Create.

The necessary files are created automatically, and the specified settings apply to your model.

21 Click OK to close the CAN_J1939_M2 [TutorialJ1939] dialog.

Interim result

You configured the CAN communication of the CAN_J1939_M2 MainBlock. You can now configure the network management for the CAN_J1939_M4 MainBlock. Continue with part 6.

Part 6

To configure the network management for the CAN_J1939_M4 MainBlock

1 Double-click the CAN_J1939_M4 MainBlock to open it.

2 On the Tree Views page, select Filter Configuration Tree by Feature.

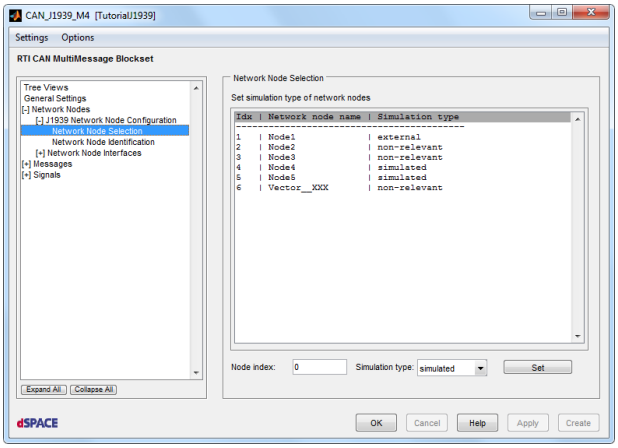
3 Clear all filters except for the Instrumentation & Automation Interface, Manipulation Interface, and J1939 filters.

4 In the dialog tree, select Network Nodes – J1939 Network Node Configuration – Network Node Selection.

5 Specify the following settings:

Network Node	Simulation Type
Node1	External
Node2	Non-relevant
Node3	Non-relevant
Node4	Simulated
Node5	Simulated
Vector_XXX	Non-relevant

The CAN_J1939_M4 MainBlock represents the network nodes Node4 and Node5, so these two network nodes are active. Node1, Node2 and Node3 are not used for communication in this block. They are configured externally or not relevant. The Vector_XXX network node is for unmapped messages only and will not be used.



6 In the dialog tree, select Network Nodes – Network Node Interfaces – Network Node Enable.

7 Clear the Use these options for all checkbox.

The Specify enables for all network nodes list is enabled.

8 Specify the following settings for Node4:

Setting	Selection
Source	TRC
Logic	–
Default source	TRC
Default	ENABLE

9 Click Set.

Note

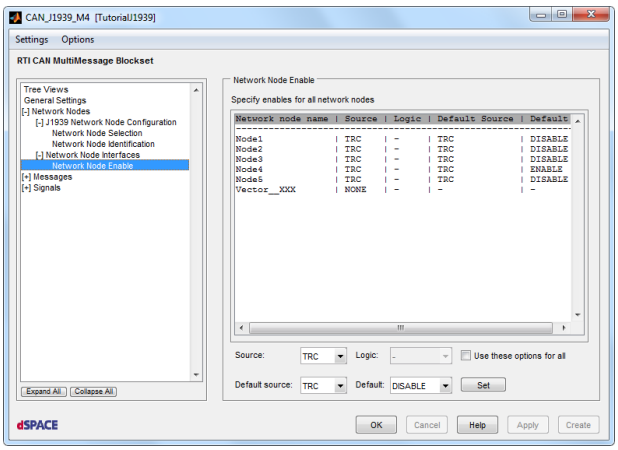
You always have to click Set to apply your settings.

You specified to enable or disable the Node4 network node via ControlDesk during run time. You enabled Node4 by default.

10 Specify the following settings for the other network nodes:

Network Node	Source	Logic	Default Source	Default
Node1	TRC	–	TRC	DISABLE
Node2	TRC	–	TRC	DISABLE
Node3	TRC	–	TRC	DISABLE
Node5	TRC	–	TRC	DISABLE
Vector_XXX	None	–	–	–

You enabled Node4 and disabled the other network nodes by default. Their default values can be changed via ControlDesk during run time. Since Vector_XXX is not relevant in this tutorial, it is disabled and cannot be enabled during run time.



Interim result

You configured the network management for the CAN_J1939_M4 MainBlock. You can now create instances of J1939 messages for the block. Continue with part 7.

Part 7

To create instances of J1939 messages for the CAN_J1939_M4 MainBlock

1. In the dialog tree, select Messages – J1939 – Source Destination Mapping.
2. Select PG_F001_08 from the PGN list and Node1 from the Source list.

   Tip

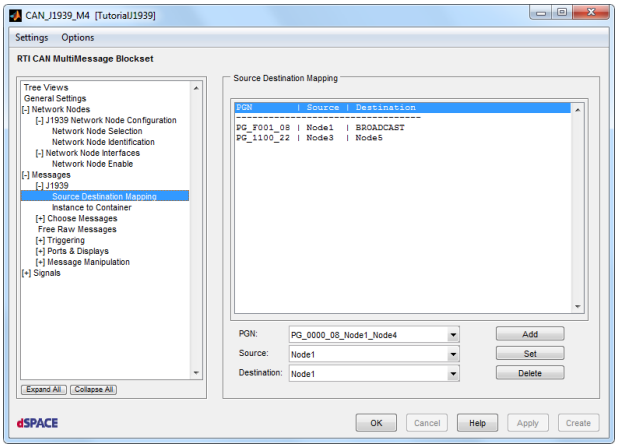
   Since PG_F001_08 is specified as the PGN for a broadcast message in the DBC file and broadcast messages are sent to all the network nodes in a network, the Destination list is disabled.

3. Click Add.

   You created a user-defined source/destination mapping for the PG_F001_08 message with Node1 as the source network node.
4. Select PG_1100_22 from the PGN list.

   Since the PG_1100_22 PGN belongs to a peer-to-peer message, the Destination network node list is enabled.
5. Select Node3 from the Source list and Node5 from the Destination list.
6. Click Add.

   You created a user-defined source/destination mapping for the PG_1100_22 message with Node3 as the source network node and Node5 as the destination network node.



Interim result

You specified instances of the PG_F001_08 and PG_1100_22 messages as follows:

Message	Source Network Node	Destination Network Node
PG_F001_08	Node1	All
PG_1100_22	Node3	Node5

You can now configure the CAN communication of the CAN_J1939_M4 MainBlock. Continue with part 8.

Part 8

To configure the CAN-J1939 communication of the CAN_J1939_M4 MainBlock

1 In the dialog tree, select Messages – J1939 – Instance to Container.

On the Instance to Container page, you can enable to show instances of a message in a container message. Showing instances in a container message can reduce performance. You should therefore specify as few instances as possible to be shown in a container message.

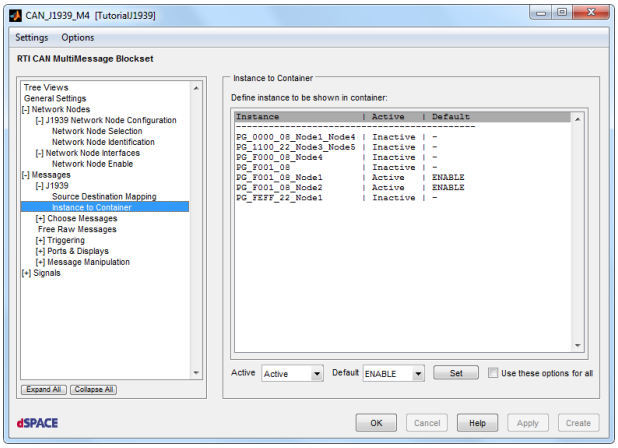
2 Select the PG_F001_08_Node1 and PG_F001_08_Node2 messages from the list and specify the following settings:

Setting	Selection
Active	Active
Default	ENABLE

3 Click Set.

Note

You always have to click Set to apply your settings.



You enabled to show the selected message instances in the container by default. You can disable this option during run time.

4 In the dialog tree, select Messages – Choose Messages – Network Node Preselection.

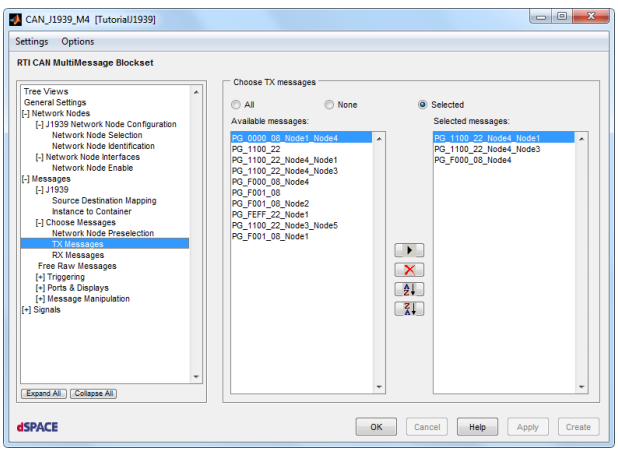
5 In the Available network nodes list, select Node4 and Node5 and click .

Node4 and Node5 are added to the Selected Network Nodes list. This network node selection can be used for further message setup.

6 In the dialog tree, select Messages – Choose Messages – TX Messages.

7 In the Selected messages list, right-click and select Choose Messages Transmitted by Selected Network Nodes from the context menu.

All the messages transmitted by Node4 and Node5 are added to the Selected messages list. Only these messages will be transmitted by this MainBlock.

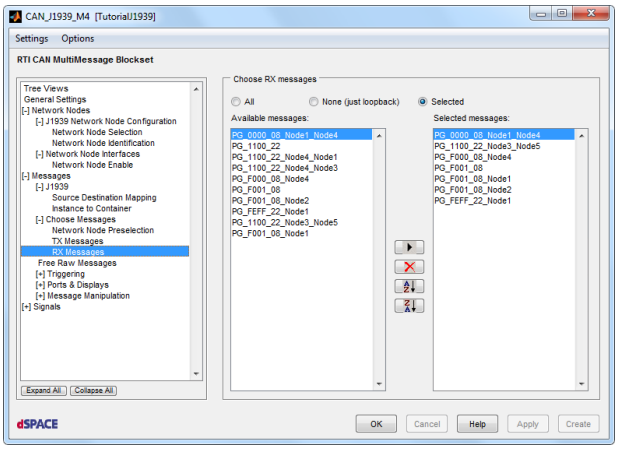


8 In the dialog tree, select Messages – Choose Messages – RX Messages.

9 In the Selected messages list, right-click and select Choose Messages Received by Selected Network Nodes from the context menu.

10 In the Available messages list, select PG_F001_08 and click .

The messages displayed in the **Selected messages** list are received by **Node4** and **Node5**.



11 In the dialog tree, select **Messages – Triggering – Message Enable**.

12 Specify the following settings:

Setting	Selection
Source	TRC
Default	DISABLE

You disabled all the messages by default, but you can enable them during run time via ControlDesk. Since the **Specify enables for all messages** list is disabled, it is not updated.

Tip

Click **Apply** to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

13 In the dialog tree, select **Signals – TX – Signal Manipulation – Counter**.

14 Click **Filter**.

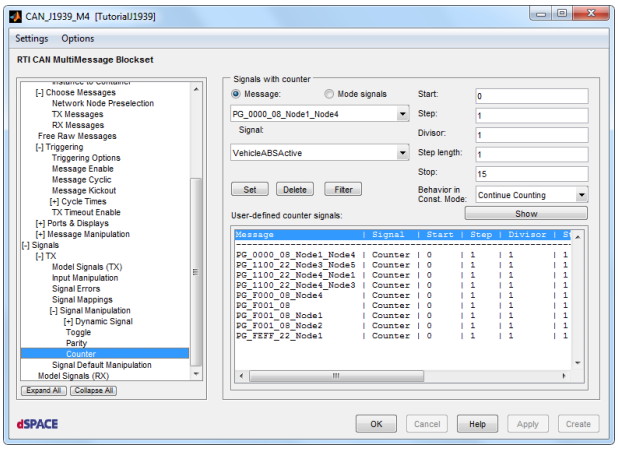
The **Filter** dialog opens for you to filter signals by their names.

15 Enter **`Counter`** in the dialog's edit field and click to activate the wildcard filter.

16 Select all the displayed messages.

17 Click **OK**.

The Filter dialog is closed and the signals are added to the **User-defined counter signals** list on the Counter page.



18 Click **Create**.

The necessary files are created automatically, and the specified settings apply to your model.

19 Click **OK** to close the **CAN_J1939_M4 [TutorialJ1939]** dialog.

20 Save your model.

21 Build an application for your model.

Result

You configured the CAN-J1939 communication of your model. You set up the network management and defined instances of a broadcast and a peer-to-peer message. You configured communication via gateway between network nodes with different simulation types (simulated, external and non-relevant), where different J1939 message types (broadcast and peer-to-peer messages > 8 bytes, CAN messages with standard length (0 ... 8 bytes)) are used. You specified counter signals for all the instance messages and enabled to show two instances in their container message.

Tip

The configuration of CAN communication performed in this step is only one example. You can configure all the instances like standard messages.

What's next

In the next step, you can observe the network management of your model in ControlDesk.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Step 3: How to Observe the Network Management in ControlDesk

Objective

In ControlDesk, you can observe the behavior of the network management and the available status variables. You can activate nodes to trigger the network management.

In this step, you will create an experiment in ControlDesk, trigger the network management by activating the conflicting Node5 node (which has the same initial node address as Node4, but with enabled `Arbitrary Address Capable` flag), and observe the effects on the layouts.

Part 1

To add the relevant network node variables of Node5 to the layout

1. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.
2. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

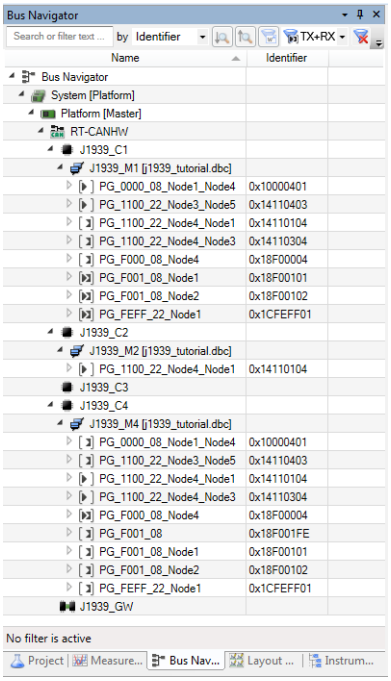
   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.
3. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.
4. From the platform's context menu in the Project controlbar, select Add Variable Description.

   ControlDesk opens the Select Variable Description dialog.
5. In the Select Variable Description dialog, click Import and select the `tutorialj1939.sdf` file from your `RTICANMMTutorial_Lesson13` folder.
6. In the Select Variable Description dialog, click OK.

   ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

The Bus Navigator displays the four CAN controllers, the J1939_GW gateway node and the configured messages.

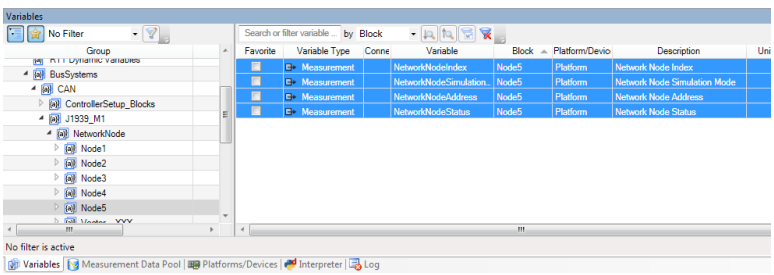


7 Rename the Layout1 default layout. To do so, select Rename from the layout's context menu in the Project controlbar and enter **`Network Management`**. Click OK.

8 Select the Variables controlbar.

9 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M1 – Network Node – Node5.

10 Select all variables from the variable list.



11 Drag the selected variables onto an empty space in the Network Management layout and assign them to a Variable Array.

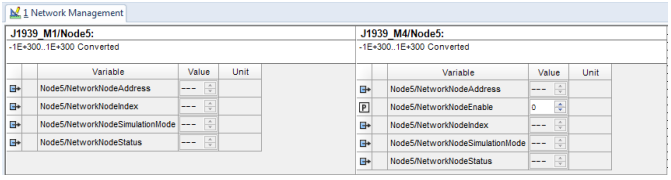
12 Change the name of the Variable Array to **`J1939_M1/Node5`**.

13 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – Network Node – Node5.

14 Select all variables from the variable list, drag them onto an empty space in the Network Management layout and assign them to a new Variable Array.

15 Rename the new Variable Array to **`J1939_M4/Node5`**.

You have now added the relevant variables of Node5 to the layout.



16 Drag the Node5/NetworkNodeStatus variable from the J1939_M1/Node5 Variable Array to a new Time Plotter.

17 Drag the Node5/NetworkNodeStatus variable from the J1939_M4/Node5 Variable Array to the same Time Plotter.

This allows you to observe the real-time behavior of Node5 during node activation.

18 Right-click the Time Plotter's y-axis area and select Y-Axes View – Horizontal Stacked.

The Y-axes are arranged one below the other.

Interim result

You added the variables of Node5 that are relevant for network management and the status variables of Node5 to a layout. You can now add the network node variables of the simulated network node Node4 to the layout. Continue with part 2.

Part 2

To add the network node variables of the simulated Node4 to the layout

1 In the tree view of the Variables controlbar, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – Network Node – Node4.

2 Select all variables from the variable list, drag them onto an empty space in the Network Management layout and assign them to a new Variable Array.

3 Rename the new Variable Array to **`J1939_M4/Node4`**.

4 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – Network Node – Node4 – J1939Identification.

5 Select the ArbitraryAddressCapable variable from the variable list and drag it to the J1939_M4/Node4 Variable Array.

6 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – Network Node – Node5 – J1939Identification.

7 Select the ArbitraryAddressCapable variable from the variable list and drag it to the J1939_M4/Node5 Variable Array.

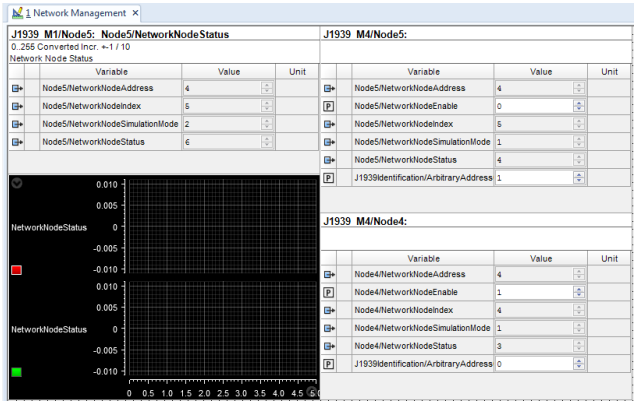
Interim result

You added the variables of Node4 that are relevant for network management and the ArbitraryAddressCapable variables of Node4 and Node5 to the layout. You can now observe the node address and node index of Node5 on the simulated and the external side of the network in ControlDesk. Continue with part 3.

Part 3

To observe the network management in ControlDesk

1 On the Home ribbon, click Status Control – Go Online to start online calibration.



The Node<x>/NetworkNodeSimulationMode variables display the simulation type selected for the nodes. They indicate the network's simulated and external sides.

Variable Value	Description
1	Simulated
2	External
3	Non-relevant

The Node<x>/NetworkNodeStatus variables display status information on the network nodes.

Variable Value	Meaning	Description	Possible Status for Node with Simulation Type ...			Remarks
			Simulated	External	Non-Relevant	
1	Network management initialization	Internal state directly after network management has been started.	✓	–	–	–
2	Address claiming	The address claiming process for an enabled node with enabled network management is in progress.	✓	–	–	A node with status value 2 executed on a dSPACE system transmits or receives only network management messages.
3	Address claimed	The network node has claimed a J1939 address. ▪ Simulated node: The address was claimed successfully. ▪ External node: The address has been verified by a Request Message for Address Claimed.	✓	✓	✓	A node with status value 3 executed on a dSPACE system receives and transmits messages.

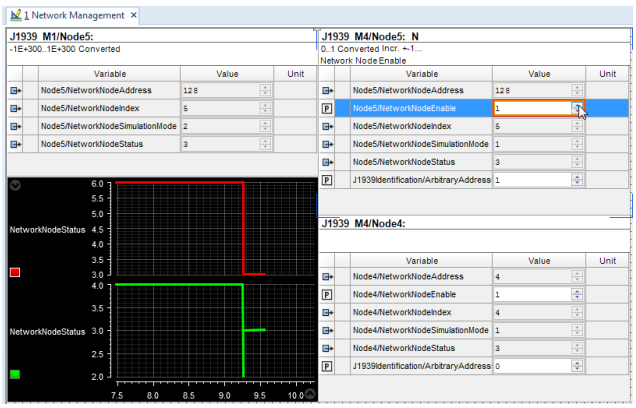
Variable Value	Meaning	Description	Possible Status for Node with Simulation Type ...			Remarks
			Simulated	External	Non-Relevant	
		▪ Non-relevant nodes are always handled as 'address claimed'.				
4	Address claiming disabled	The node or its network management is disabled. No network management is done on the bus. **Note** Network management is disabled if the 64-bit identifier is not unique.	✓	–	–	A node with status value 4 executed on a dSPACE system can transmit or receive messages on the J1939 bus.
5	Address claiming failed	▪ Simulated node: The node could not claim an address. ▪ External node: The address of the node could not be verified.	✓	✓	–	A node with status value 5 executed on a dSPACE system does not transmit or receive messages.
6	Address unverified	The address of the node is unverified. The address is requested using a Request Message for Address Claimed. The answer is not yet received.	–	✓	–	–

The network node addresses of Node4 and Node5 are identical, which means that there is an address conflict. However, Node5 is still disabled on the simulated side of the network, the node statuses are 4 and 6.

2 On the Home ribbon, click Status Control – Start Measuring to start measurement.

3 In the J1939_M4/Node5 Variable Array, change the value of the Node5/NetworkNodeEnable variable to **1**, and then immediately stop measurement by clicking Status Control – Stop Measuring.

You now have time to check the results.



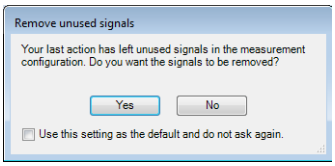
You enabled the simulated Node5 network node. The activated Node5 successfully performed dynamic network node addressing (address claiming) to get a unique address. This is indicated by the Node5/NetworkNodeStatus variable, which briefly had a value of 2 and then changed to 3. The new address has been passed to the externally configured network node. The address of Node4 has remained unchanged.

4 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

5 Save the project.

6 For a better overview, close the layout.

The Remove unused signals dialog opens.



7 In the Remove unused signals dialog, click Yes.

The unused signals are removed from the measurement signal list and the dialog closes.

Interim result

You activated a simulated network node and observed J1939 network management. Due to an address conflict, Node5 (which is simulated by the RTI CAN MultiMessage Blockset) performed address claiming to change its address. The new address was passed to the external network node.

You can now send messages from a simulated node to a simulated node on a different CAN controller and from a node without network management. Continue with part 4.

Part 4

To send a J1939 message from a node that is simulated on another CAN controller or that really exists

1 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

2 Rename the new layout `Network Management 2`.

3 Go to the Bus Navigator and open the following layouts:

CAN Bus	Message	Layout
J1939_M4	PG_1100_22_Node4_Node1	TX
J1939_M2	PG_1100_22_Node4_Node1	TX
J1939_M1	PG_1100_22_Node4_Node1	RX

4 Select the Variables controlbar.

5 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – Network Node – Node4.

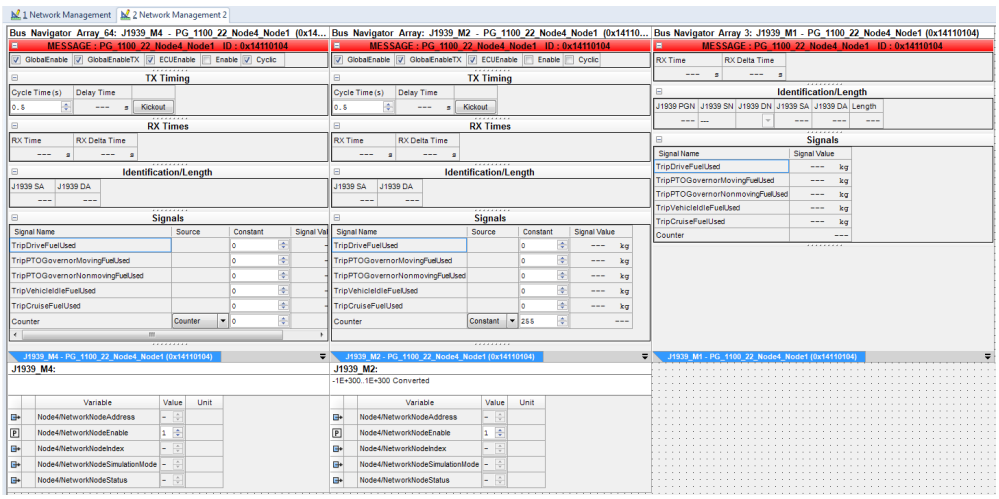
6 Select all variables from the variable list, drag them onto an empty space in the Network Management 2 layout and assign them to a Variable Array.

7 Change the name of the Variable Array to **J1939_M4**.

8 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M2 – Network Node – Node4.

9 Select all variables from the variable list, drag them onto an empty space in the Network Management 2 layout and assign them to a new Variable Array.

10 Change the name of the second Variable Array to **J1939_M2**.



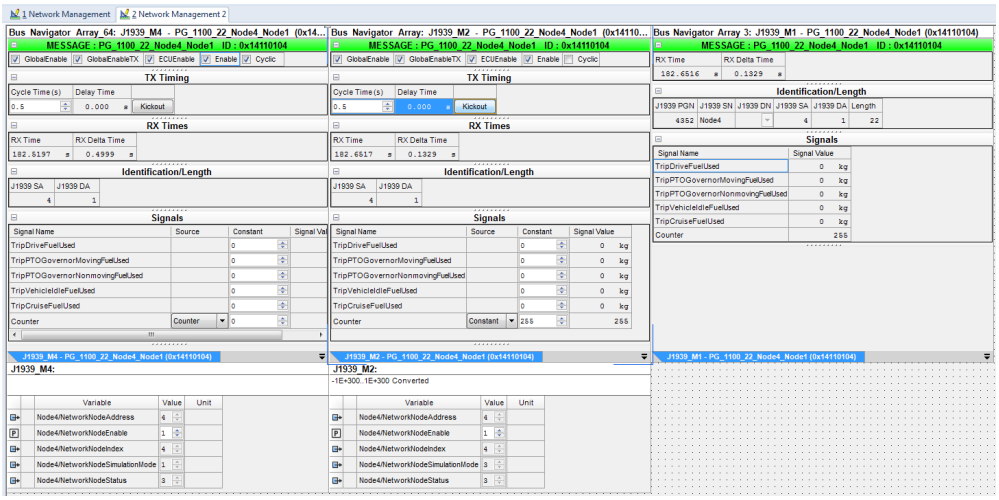
11 On the Home ribbon, click Status Control – Go Online to start online calibration.

12 In both TX layouts of PG_1100_22_Node4_Node1, select Enable.

Since you specified triggered message transmission for the PG_1100_22_Node4_Node1 message for J1939_M2, only the message from J1939_M4 is periodically transmitted on the bus. This is the normal use scenario in our lesson as the J1939 peer-to-peer message is sent from Node4 (simulated on the J1939_M4 CAN controller) to Node1 (simulated on the J1939_M1 CAN controller).

13 You will now send the PG_1100_22_Node4_Node1 message from J1939_M2 without network management. In the J1939_M2 - PG_1100_22_Node4_Node1 TX layout, click Kickout.

The message is transmitted once via kickout. For a better observation, you can click Kickout for several times. In the RX layout, the value of the RX Delta Time value changes, and the Counter variable briefly displays the value 255.



Tip

Observe the behavior of the network node addresses and statuses in the J1939_M4 and J1939_M2 Variable Arrays.

14 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

15 Save the project.

16 For a better overview, close the layouts.

Result

You configured the network management and observed the effects in ControlDesk. You sent a peer-to-peer message from a simulated network node to a node simulated on a different CAN controller. Additionally, you sent a message from a node without network management 'in the name of' a network node that is simulated on another CAN controller or that really exists.

What's next

In the next step, you can observe the CAN-J1939 communication of your model in ControlDesk.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Step 4: How to Observe the CAN-J1939 Communication in ControlDesk

Objective

In ControlDesk, you can observe the behavior of instance and container messages. You can modify the index of a source network node defined for an instance message and observe the effects on the corresponding RX message layouts, for example.

In this step, you will create a new layout in your experiment, modify the CAN communication, and observe the effects on the layouts.

Part 1

To observe the CAN-J1939 communication in ControlDesk

1 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

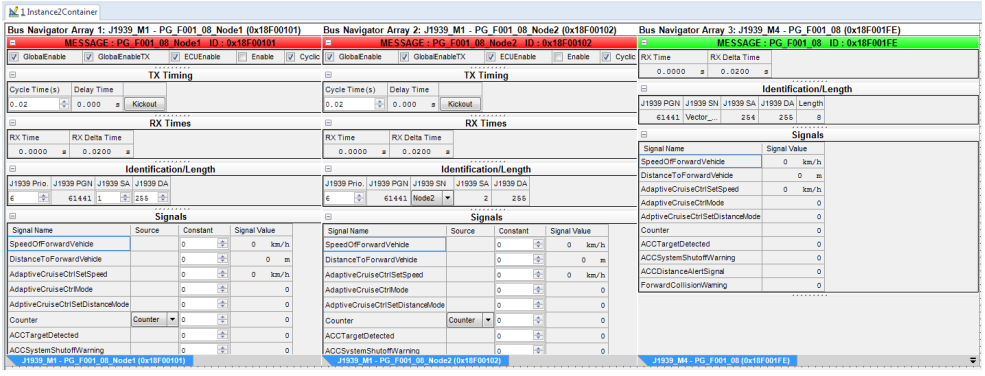
2 Rename the layout `Instance2Container`. To rename the layout, select Rename from the layout's context menu in the Project controlbar.

3 Open the following layouts:

CAN Bus	Message	Layout
J1939_M1	PG_F001_08_Node1	TX
	PG_F001_08_Node2	TX
J1939_M4	PG_F001_08	RX

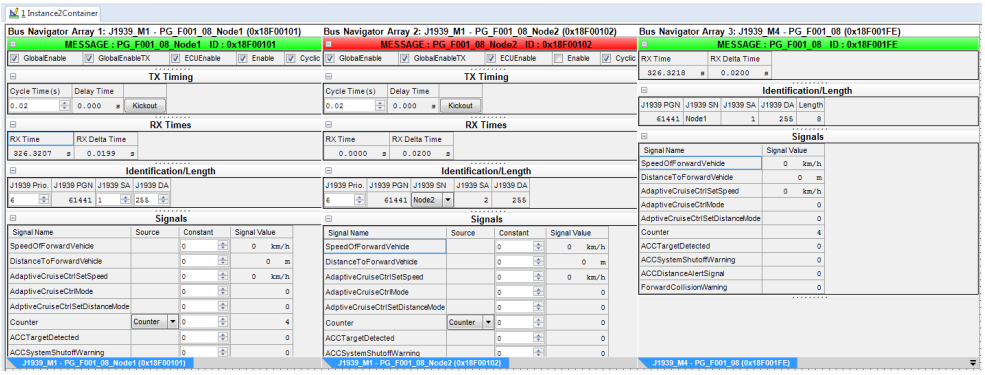
4 On the Home ribbon, click Status Control – Go Online to start online calibration.

Since you disabled all the messages by default, no message is transmitted.



5 In the TX layout of PG_F001_08_Node1, select Enable.

The PG_F001_08_Node1 message is transmitted. It is an instance of PG_F001_08 and received by the PG_F001_08 container.



6 In the TX layout of PG_F001_08_Node2, select Enable.

The PG_F001_08_Node2 instance message is transmitted and received by the PG_F001_08 container. The container now receives PG_F001_08_Node1 and PG_F001_08_Node2. Since both instance messages have identical cycle times, they are received simultaneously and only the reception of one message can be observed in the PG_F001_08 RX layout.

Interim result

You created a new layout and enabled the transmission of the PG_F001_08_Node1 and PG_F001_08_Node2 instances. For a better overview of the message transmission, you can now display the source of messages in a Time Plotter. Continue with part 2.

Part 2

To display the source node indices of messages in a Time Plotter

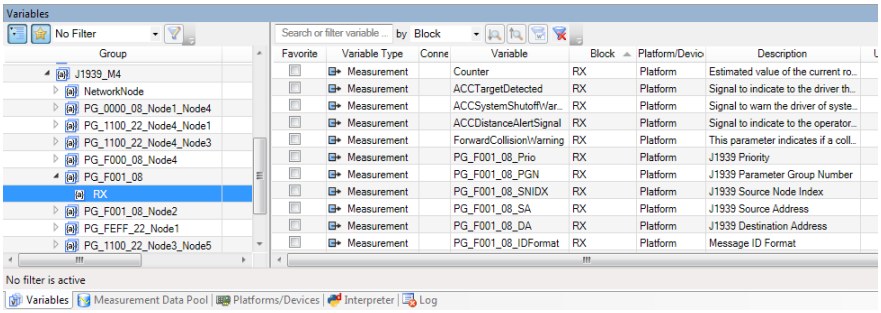
1 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

2 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

3 Rename the new layout `Instance2Container_Plotter`.

4 Select the Variables controlbar.

5 In the tree view, select tutorialj1939.sdf – BusSystems – CAN – J1939_M4 – PG_F001_08 – RX.



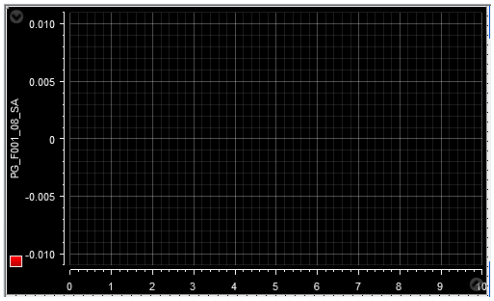
The window on the right displays all the available variables for RX PG_F001_08.

6 From the variable list, drag the PG_F001_08_SA variable onto an empty space of the layout.

A drop-down list opens for you to select an instrument to be assigned to the variable.

7 From the list, select Time Plotter.

A Time Plotter is added to the Instance2Container_Plotter layout and the y-axis of the plotter displays the name of the connected signal.



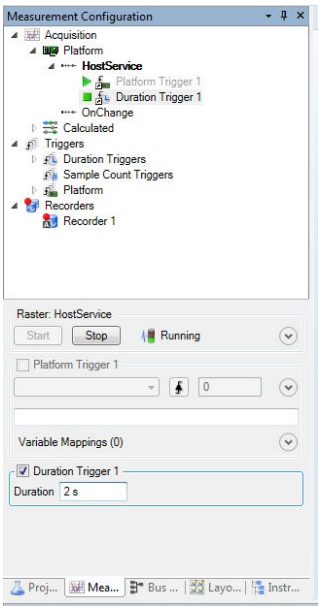
8 On the Home ribbon, click Status Control – Start Measuring to start measurement.

9 In the Measurement Configuration controlbar, select Acquisition – Platform – HostService – Duration Trigger 1.

> **Tip**
>
> If the Measurement Configuration controlbar is hidden, go to the View ribbon and select Controlbar – Switch Controlbars – Measurement Configuration.

10 In the Duration edit field, specify **2** seconds.

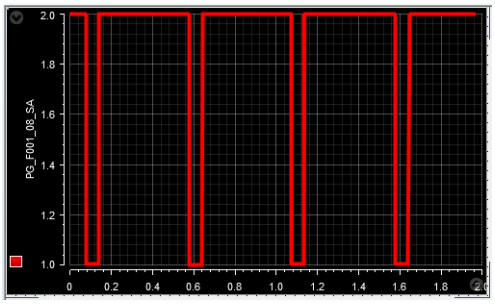
11 To activate the duration trigger, select its checkbox.



The x-axis of the Time Plotter is scaled to 2.

12 In the TX layouts of **PG_F001_08_Node1** and **PG_F001_08_Node2**, type **`0.5`** in the **Cycle time** edit fields and press **`Enter`**.

The PG_F001_08 container receives the PG_F001_08_Node1 and PG_F001_08_Node2 instance messages. The Time Plotter displays the addresses of the source nodes received by PG_F001_08.



13 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

14 Save the project.

15 For a better overview, close the layouts.

The **Remove unused signals** dialog opens.

16 In the **Remove unused signals** dialog, click **Yes**.

The unused signals are removed from the measurement signal list and the dialog closes.

Interim result

You displayed the different source network nodes received by PG_F001_08 in a plotter. You can now observe the effects on CAN communication when you manipulate settings of PG_F001_08_Node1. Continue with part 3.

Part 3

To manipulate settings of the PG_F001_08_Node1 message and observe the effects on the corresponding RX layouts

1 On the Layouting ribbon, click Layouts – Insert Layout to create a layout.

2 Rename the new layout `Instance2Container_2`.

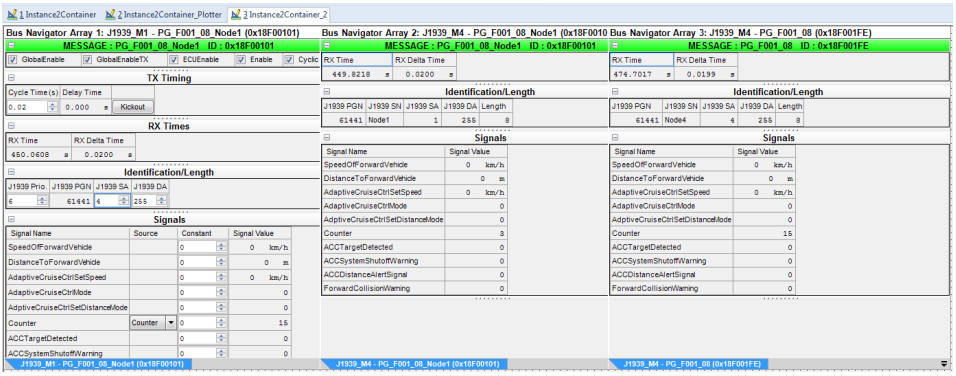
3 Open the following layouts:

CAN Bus	Message	Layout
J1939_M1	PG_F001_08_Node1	TX
J1939_M4	PG_F001_08_Node1	RX
	PG_F001_08	RX

4 On the Home ribbon, click Status Control – Go Online to start online calibration.

5 In the TX layout of PG_F001_08_Node1, select 4 as the value for J1939 SA.

You changed the source address of PG_F001_08_Node1. The PG_F001_08 container still receives PG_F001_08_Node1, but the RX layout of PG_F001_08_Node1 no longer receives the message.

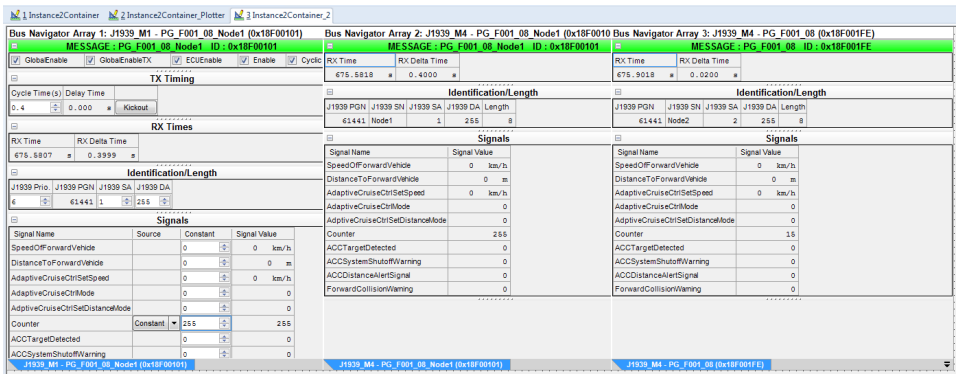


6 Specify 1 as the value for J1939 SA.

Both RX layouts receive PG_F001_08_Node1.

7 Specify a Cycle time of `0.4`.

8 For the **Counter** variable, select **Constant** from the **Source** drop-down list and specify a **Constant** value of **255**.



PG_F001_08_Node1 of the CAN_J1939_M4 MainBlock receives the message with its constant value. PG_F001_08 receives the constant value of PG_F001_08_Node1 and the counter signal of PG_F001_08_Node2 in parallel.

> **Tip**
>
> Open the TX layout of PG_F001_08_Node2 to observe the counter behavior.

9 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

10 Save the project.

11 For a better overview, close the layouts.

Result

You built an application of your model, created an experiment and modified your CAN communication.

- *RX layout of PG_F001_08*: PG_F001_08 is a container message. Since you enabled to show all the instances of a message in the container, the RX layout of this message displays PG_F001_08_Node1 and PG_F001_08_Node2. When you change the source network node or source address of one instance, the RX layout still receives the message because a container receives all the instances and not only the predefined instances.
- *TX layouts of PG_F001_08_Node1 and PG_F001_08_Node2*: You can manipulate the instances via these layouts. You changed the source address of PG_F001_08_Node1 to observe the effects on the corresponding RX layouts. Since these messages are broadcast messages, you cannot define or manipulate a destination network node.
- *RX layout of the PG_F001_08_Node1*: This layout only receives PG_F001_08_Node1. If you change the source address of PG_F001_08_Node1, for example, the message is no longer received by this RX message.

What's next

In the next step, you can gateway J1939 messages.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Step 5: How to Gateway J1939 Messages Using the RTICANMM Gateway Block

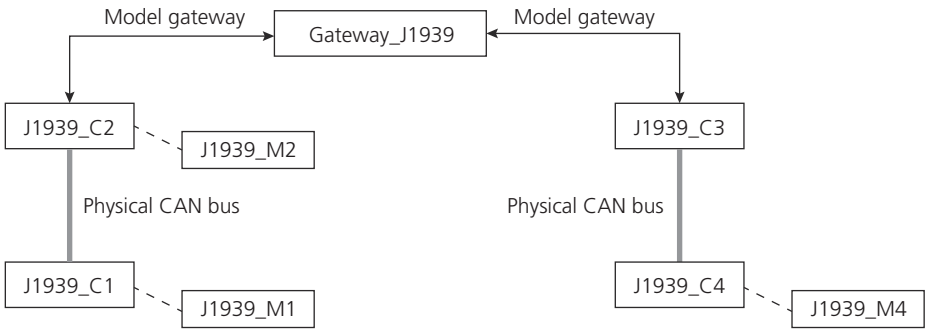
Objective

Gatewaying J1939 messages hardly differs from gatewaying standard messages. You can gateway messages via two RTICANMM MainBlocks and/or via an RTICANMM Gateway block.

Tip

For more information on gatewaying messages, refer to *Lesson 12 (Advanced): Gatewaying Messages*.

In this step, you will gateway J1939 messages via an RTICANMM Gateway block as follows:



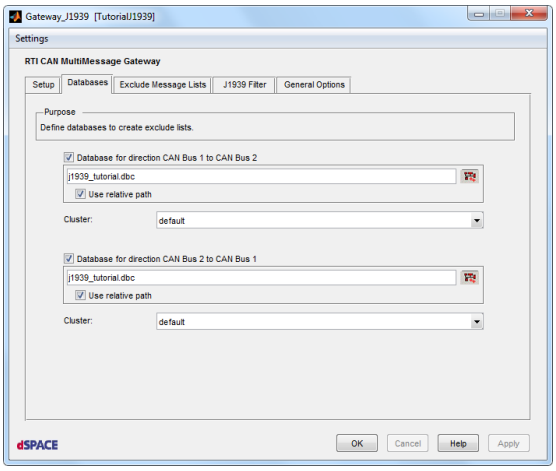
Part 1

To specify general settings of an RTICANMM Gateway block to gateway J1939 messages

1. Double-click the RTICANMM Gateway block of your model to open it.
2. Select the Databases page.
3. Select Database for direction CAN Bus 1 to CAN Bus 2.
4. Click and select the j1939_tutorial.dbc file.

5 Select **Database for direction CAN Bus 2 to CAN Bus 1**.

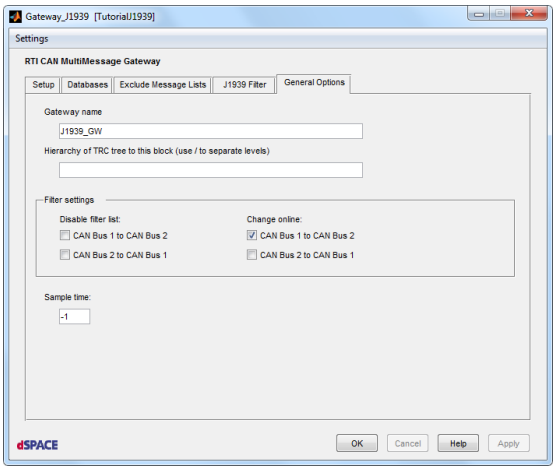
6 Click and select the j1939_tutorial.dbc file.



All the messages specified in this DBC file will be gatewayed from CAN bus 1 to CAN bus 2 and from CAN bus 2 to CAN bus 1 by default. According to the specification you made on the Setup page, all the messages will be gatewayed between the J1939_C2 CAN controller and the J1939_C3 CAN controller. You can specify filters to exclude messages from being gatewayed.

7 Select the **General Options** page.

8 In the **Filter settings** frame, select **CAN Bus 1 to CAN Bus 2** for **Change online**.



Interim result

You specified the DBC file for gatewaying messages. Additionally, you enabled to change specified filter settings for the gateway direction from CAN bus 1 to CAN bus 2 during run time. You can now specify filter settings for J1939 messages to be excluded from gatewaying. Continue with part 2.

Part 2

To specify filters for excluding J1939 messages from gatewaying

1 Select the J1939 Filter page.

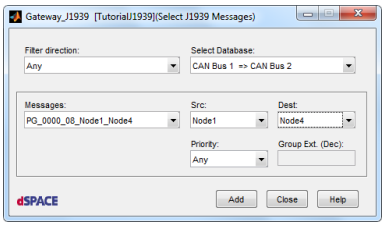
Via the J1939 Filter page, you can exclude J1939 messages from being gatewayed. In part 1, you enabled to change filter settings during run time. You can therefore change the settings defined on this page during run time.

2 Click Add.

The Gateway_J1939 [TutorialJ1939](Select J1939 Messages) dialog opens.

3 In the dialog, specify the following settings:

Setting	Selection
Messages	PG_0000_08_Node1_Node4
Src	Node1
Dest	Node4



4 Click Add.

The specified instance message is added to the list of the J1939 Filter page.

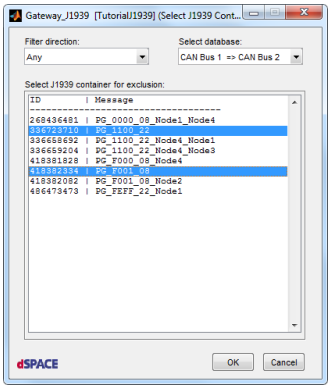
5 In the Gateway_J1939 [TutorialJ1939](Select J1939 Messages) dialog, click Close.

The dialog is closed.

6 On the J1939 Filter page, click Add J1939 container.

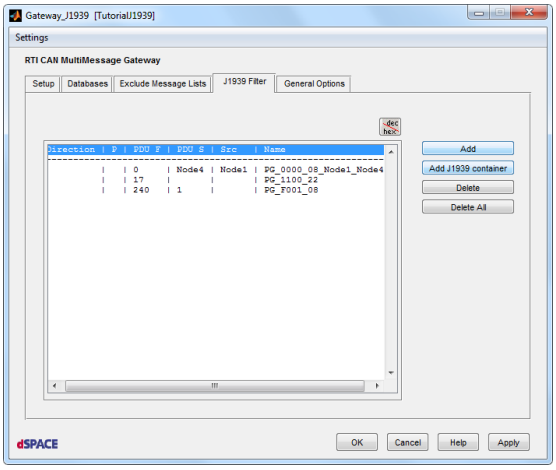
The Gateway_J1939 [TutorialJ1939](Select J1939 Container) dialog opens.

7 In the Select J1939 container for exclusion list, select PG_1100_22 and PG_F001_08.



8 Click OK.

The dialog closes and PG_1100_22 and PG_F001_08 are added to the list of the J1939 Filter page. The instance of the PG_0000_08_Node1_Node4 message and the PG_1100_22 and PG_F001_08 containers will be excluded from being gatewayed by default.



9 Click OK.

The dialog closes, an S-function is created automatically, and the specified settings apply to the block.

10 Save your model.

Interim result

You built a gateway and specified one instance of PG_0000_08_Node1_Node4 and the PG_1100_22 and PG_F001_08 containers to be excluded from being gatewayed by default. You can now observe the specified settings in ControlDesk. Continue with part 3.

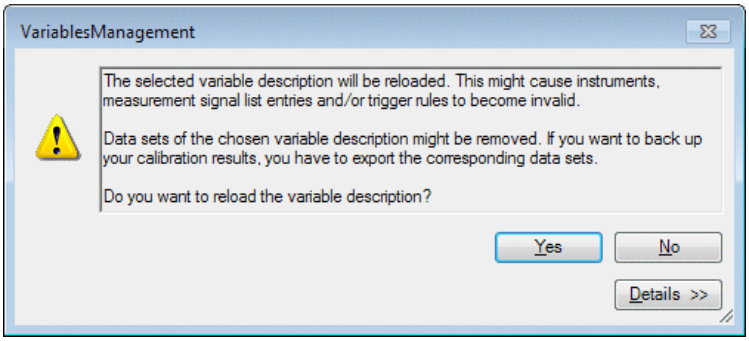
Part 3

To observe the specified CAN communication in ControlDesk

1 Build an application for your model.

2 From the context menu of the `TutorialJ1939.sdf` file in the active ControlDesk experiment, select Reload Variable Description.

The following warning is displayed, informing you that reloading a variable description might cause instruments and other items to become invalid. Click Yes.



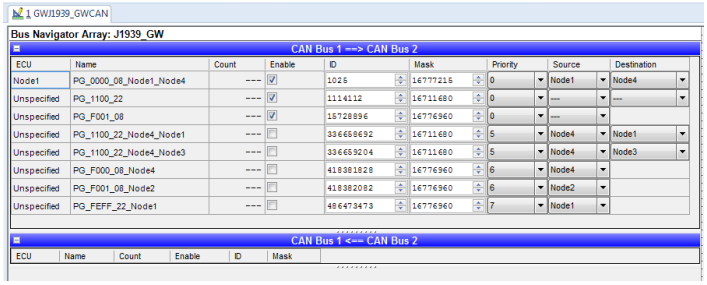
3 In the Bus Navigator tree, right-click the J1939_GW node and generate the gateway layout.

The gateway layout displays all the gatewayed messages, including those for which filters are active.

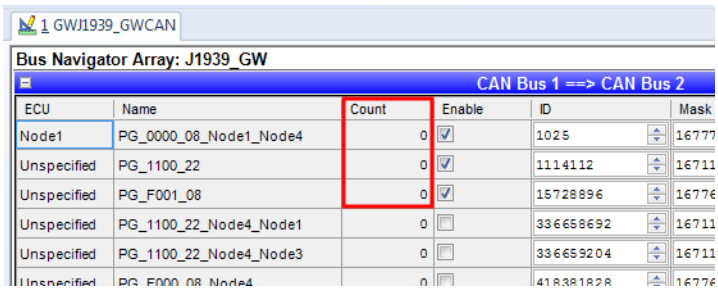
Tip

Extend the gateway layout to view additional columns which let you change the message priority, the source network node and the destination network node (only for peer-to-peer messages).

4 On the Home ribbon, click Status Control – Go Online to start online calibration.



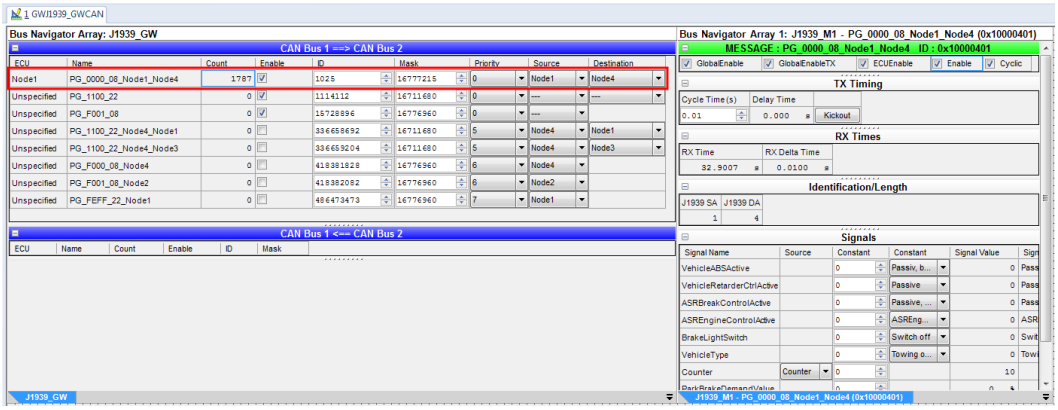
The selected checkboxes indicate that all the message filters are active and the messages are excluded from being gatewayed. Since you disabled the transmission of all the messages in step 2 of this lesson, the messages are neither transmitted nor gatewayed in any case. The constant hit counter values indicate that the message filters are not working even though they are active.



5 Open the TX layout of PG_0000_08_Node1_Node4 of the J1939_M1 CAN bus.

6 Select **Enable**.

The running RX time in the PG_0000_08_Node1_Node4 layout indicates that the message is transmitted. However, PG_0000_08_Node1_Node4 is not gatewayed. The running hit counter in the gateway layout indicates that the message filter is active and works accurately.



7 Generate the RX layout of **PG_0000_08_Node1_Node4** of the **J1939_M4** CAN bus.

Since PG_0000_08_Node1_Node4 is excluded from being gatewayed, it is not received in the corresponding RX layout.

8 In the gateway layout, clear the **Enable** checkbox for the **PG_0000_08_Node1_Node4** filter of **Node1**.

The filter is disabled, so PG_0000_08_Node1_Node4 is gatewayed and received in the corresponding RX layout.

9 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

10 Save the project.

Result

You gatewayed all the messages defined in the j1939_tutorial.dbc file and defined filters to exclude messages from being gatewayed. In ControlDesk, you observed the specified gateway behavior and changed the predefined filter settings.

What's next

With this step, you finished working with J1939 messages in RTICANMM. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Result of Lesson 13

Result

In this lesson, you have learned the basic principles of working with a J1939-compliant DBC file in RTICANMM:

- The RTI CAN MultiMessage Blockset can actively simulate the J1939 network management or observe it passively for a really existent network node or a network node that is simulated on another CAN controller. A network node can also be set to non-relevant so that the network management is disabled for it. During configuration, you can change any network node parameters whose nodes are defined in the DBC file. The network management can be triggered during run time by enabling a network node.
- The network management is not required for sending or receiving a J1939 message by RTICANMM. It is possible to send and receive J1939 messages regardless of the specified simulation type (simulated, external or non-relevant) of the network nodes. There is one limitation: J1939 messages > 8 bytes with a point-to-point connection always use the "request to send / clear to send (RTS/CTS)" transport protocol type. These messages can only be transmitted by RTICANMM if the responder network node really exists on the CAN bus, or is "simulated" in the case of a dSPACE system.
- There are two types of J1939 messages: broadcast messages are sent to all the network nodes connected to the network, peer-to-peer messages are sent to a particular network node (destination node). Depending on its PGN, a message defined in a J1939-compliant DBC file can be either a broadcast message or a peer-to-peer message.
- For each message defined in a J1939-compliant DBC file, the source/destination mapping allows you to specify the source (transmitting) network node and the destination (receiving) network node (only for peer-to-peer messages).

  With source/destination mapping, if two or more network nodes send the same PGN in the same sampling step, the two J1939 messages can be treated individually as separate instances of the message. The source/destination mapping is already specified in the DBC file or you can specify it on the Source Destination Mapping page. The name of a user-defined instance is generated automatically and has the syntax `<container name>_<source node>_<destination node>` (destination node only for peer-to-peer messages).
- The RTI CAN MultiMessage Blockset distinguishes between container and instance messages. A container message is a J1939 message defined by its PGN. A container can receive all the messages with its PGN. If several messages are received in one sampling step, only the last received message is held in the container. An instance message is a J1939 message defined by its PGN, for which the source network node and the destination network node (only for peer-to-peer communication) are defined.

- With RTICANMM, a J1939 message is received as follows: If the configuration contains an RX instance with matching PGN, source node and destination node, the message is received in that instance. If the configuration does not contain such an instance, the message is received in a matching container (if existent). The RTI CAN MultiMessage Blockset lets you specify to place an instance in its associated J1939 container.

  There exists one container for each PGN (except for proprietary PGNs).
- A message you defined instances for acts as a container message. You can enable to show all the instances of the message in the container. Container messages can display all their instance messages even if you change the source or destination network node during run time.
- Showing instances in a container message can reduce performance. You should therefore specify as few instances as possible to be shown in a container message.
- When you configure the CAN communication of your model, you can handle container messages and/or instance messages like standard messages and specify counter signals, etc., for them.
- The J1939 TX ID page of the RTICANMM MainBlock lets you prepare a J1939 message. You can enable to manipulate the message priority and/or the source and destination by addresses or node indices during run time.
- When you gateway J1939 messages, you can exclude container messages and/or instance messages from being gatewayed. You can enable to change these settings during run time.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson13_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Basics on Working with a J1939-Compliant DBC File (RTI CAN MultiMessage Blockset Reference)
- Source Destination Mapping Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- Instance to Container Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- J1939 Filter Page (RTICANMM Gateway) (RTI CAN MultiMessage Blockset Reference)
- Limitations with J1939-Support (RTI CAN MultiMessage Blockset Reference)
- Tree Views Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)

- RX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Message Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- J1939 TX ID Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- How to Create a Bus Instrument (Gateway Type) for a CAN Controller Gateway (ControlDesk Bus Navigator 🕮)

Where to go from here

Now you can continue with the next lesson to work with an M file as the database.

If you do not need to work with an M file as the database, you can continue with another lesson or the summary of this tutorial. Refer to Overview of Lessons on page 35 and Summary on page 457, respectively.

Related topics

Basics

Lesson 13 (Advanced): Working with a J1939-Compliant DBC File..... 315
Overview of Lesson 13..... 316

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database

Where to go from here

Information in this section

Overview of Lesson 14 373
In this lesson, you will work with an M file as the database.

Step 1: How to Build a Model for Using an M File as the Database 377
In this step, you will build a model for using an M file as the database.

Step 2: How to Define CAN Communication in an M File 381
In this step, you will define the CAN communication of your model in an M file. You will use an existing M file as a template.

Step 3: How to Use an M File as the Database 388
In this step, you will run the M file to generate a MAT file from it and specify the MAT file as the database for your model.

Result of Lesson 14 395
In this lesson, you worked with an M file as the database. You learned the basic principles of defining CAN communication in an M file.

Overview of Lesson 14

Introduction

With the RTI CAN MultiMessage Blockset, you can use file formats other than DBC or FIBEX to define the CAN communication of your model. If you do so, you must always use an M file. You can either define CAN communication directly in an M file or parse other file formats such as XLS or XML via the M file.

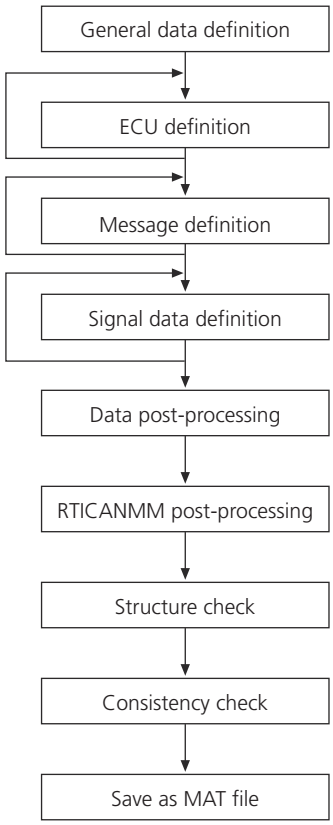
Tip

If you do not need to define CAN communication in an M file, you can skip this lesson.

RTICANMM-compliant M file structure

Since you cannot specify an M file as the database of an RTICANMM MainBlock, you have to generate a MAT file from the M file.

The M file that defines the CAN communication of your model must generate a MAT file according to a CAN structure named *dbcdata*. The basic elements to generate an RTICANMM-compliant *dbcdata* structure are:



- *General data definition*: You have to create initial parts of the database structure, for example, define the name of the user-defined database.
- *ECU definition*: You have to add all the ECUs to the structure. The ECU settings can be defined in the M file or in a parsed file format.
- *Message definition*: You have to add all the messages to the structure. The message settings can be defined in the M file or in a parsed file format.
- *Signal definition*: You have to add all the signals to the structure. The signal settings can be defined in the M file or in a parsed file format.

- *Data post-processing*: To work with a user-defined database in RTICANMM, some general information must be available for each message:
  - Number of signals
  - ID in hexadecimal format
  - Message type (standard or J1939)

  Instead of specifying the general information for each message separately, you can let MATLAB evaluate them.
- *RTICANMM post-processing*: You have to let RTICANMM-specific data be generated automatically.
- *Structure check*: You have to evaluate if all the mandatory information of the dbcdata structure is available.
- *Consistency check*: You have to check for CAN-specific errors, warnings, and information. An HTM file that displays the result of this check will be created automatically. This check is similar to the consistency check of Vector Informatik's CANdb++ Editor.
- *Save as MAT file*: The structure must be saved as a MAT file, but only if no errors occurred during the consistency check.

Tip

For more information on the *dbcdata* structure, refer to General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

M file in this lesson

In this lesson, you will work with an M file that directly defines CAN communication, so no other file formats are parsed. You will work with the chassis_dbcdata.m file template, which generates a MAT file with the same content as the chassis.dbc file. In this template, the required RTICANMM-compliant *dbcdata* structure is realized as follows:

Element to Generate DBCDATA Structure	Realized by ...
General data definition	Calling `iFCNGeneralData`
ECU definition	Calling `iFCNADD_ECU`
Message definition	Calling `iFCNADD_MESSAGEs`
Signal definition	Calling `iFCNADD_SIGNALs`
Data post-processing	Three for-loops that let MATLAB evaluate the number of signals for each message, the message ID in hexadecimal format, and the message format (standard or J1939)
RTICANMM post-processing	Calling `rtimmsu_private('can_specialdbcdata',[databasestructure])`
Structure check	Calling `rtimmsu_private('can_structcheck',[databasestructure])`
Consistency check	Calling `rtimmsu_private('can_checkdata',[databasestructure],[log file])`
Save as MAT file	An if-else condition that saves the MAT file only if no errors occurred during the consistency check.

Tip

- The `rtimmsu_private(<function>,[parameter])` functions are provided by RTICANMM so you can use them directly.
- You can use the chassis_dbcdata.m file as a template for real applications.

What you will learn

In this lesson, you will learn the basic principles of using an M file as the database.

- You will build a simple model with two CAN controllers.
- You will use the chassis_dbcdata.m file as a template. You will add one ECU, one message and four signals to the M file. The signals are one mode signal, two mode-dependent signals and one standard signal.
- You will run the M file and generate a MAT file.
- You will specify the generated MAT file as the database for both CAN controllers of your model.
- You will configure the CAN communication of your model and observe the effects via ControlDesk.

Starting point

The starting point of this lesson is a new model in MATLAB.

Prepared demo

Since you will build a new model in this lesson, no prepared demo is required.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- You are familiar with the basics of M file programming.
- You have to be familiar with building a model using RTICANMM blocks (refer to Lesson 1: Building Your First Model Using the RTI CAN MultiMessage Blockset on page 39).
- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Build a Model for Using an M File as the Database on page 377
- Step 2: How to Define CAN Communication in an M File on page 381
- Step 3: How to Use an M File as the Database on page 388

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 14 on page 395.

Duration

Working through this lesson will take you about 60 minutes.

What's next

Now you can start to build a model for using an M file as the database, see Step 1: How to Build a Model for Using an M File as the Database on page 377.

Related topics

Basics

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database..... 373

Step 1: How to Build a Model for Using an M File as the Database

Objective

Using a file format other than DBC or FIBEX as the database does not affect the general procedure for building a model.

In this step, you will build a simple model with two CAN controllers.

Part 1

To create a gateway model and add RTICANMM blocks to it

1. In your working folder, create a new folder named **`RTICANMMTutorial_Lesson14`**.
2. In MATLAB, change the Current Directory to your RTICANMMTutorial_Lesson14 folder.

   Tip

   Save all files belonging to one model in one folder. This makes it easy to move the model (for example, to another folder) because all the files are stored relative to the model path by default. Saving all files to one folder also includes changing the path of the Current Directory in MATLAB.

3. In the MATLAB Command Window, type **`rticanmm_tutorial`** to open the rticanmm_tutorial_lib library.
4. From the rticanmm_tutorial_lib library, double-click the chassis_dbcdata.m file.

   The M file is automatically copied to your RTICANMMTutorial_Lesson14 folder.
5. Close the rticanmm_tutorial_lib library.

6 Open a new model and save it as **`TutorialMFile`** in your RTICANMMTutorial_Lesson14 folder.

7 In the MATLAB Command Window, type **`rticanmm`** to open the rticanmmlib library.

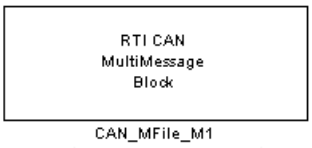
8 From the rticanmmlib library, add
- One RTICANMM GeneralSetup block,
- Two RTICANMM ControllerSetup blocks, and
- Two RTICANMM MainBlocks to your model.

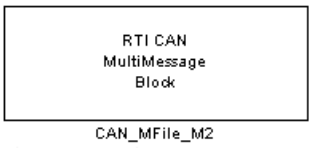
Note

Never copy an RTICANMM block. Copying RTICANMM blocks might cause problems in the model. To add a new RTICANMM block, always drag it directly from the rticanmmlib library to your model.

9 Rename the two RTICANMM ControllerSetup blocks as follows:
- **`Setup_MFile_C1`**
- **`Setup_MFile_C2`**

10 Rename the two RTICANMM MainBlocks as follows:
- **`CAN_MFile_M1`**
- **`CAN_MFile_M2`**





Interim result

You built a new model, added RTICANMM blocks to it, and renamed the RTICANMM ControllerSetup blocks and the RTICANMM MainBlocks. You can now parameterize the RTICANMM ControllerSetup blocks. Continue with part 2.

Part 2

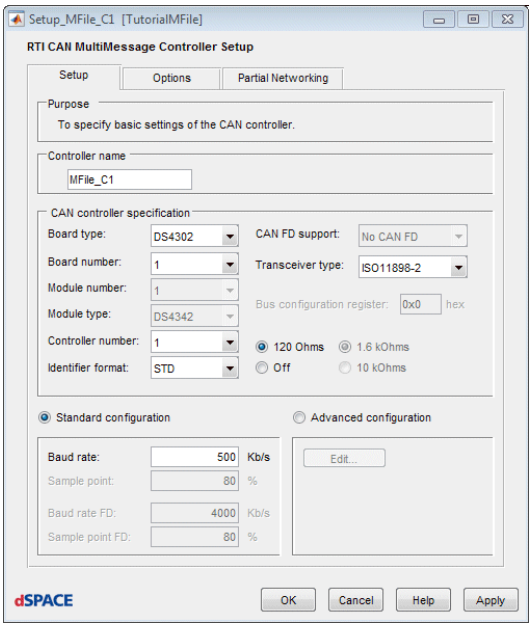
To parameterize the RTICANMM ControllerSetup blocks of your model

1 Double-click the Setup_MFile_C1 ControllerSetup block to open it.

2 Specify **`MFile_C1`** as the **Controller name**.

Note

The CAN controller name must be unique within the model and even within a whole multiprocessor system, so you should always rename the CAN controller.



3 From the **Board type** drop-down list, select the board you are working with.

4 Click **OK** to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.

5 Double-click the **Setup_MFile_C2** ControllerSetup block.

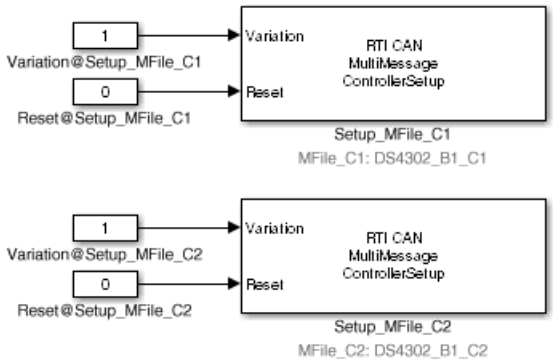
6 Specify **`MFile_C2`** as **Controller name**.

7 From the **Board type** drop-down list, select the board you are working with.

8 From the **Controller number** drop-down list, select **2**.

9 Click OK to close the dialog.

An S-function is created and the inports of the RTICANMM ControllerSetup block are connected to Constant blocks.



Interim result

You parameterized the RTICANMM ControllerSetup blocks of your model. You can now parameterize your RTICANMM MainBlocks. Continue with part 3.

Part 3

To parameterize the RTICANMM MainBlocks of your model

1 Double-click the CAN_MFile_M1 MainBlock to open it.

2 On the General Settings page, specify `MFile_M1` as the CAN name.

3 Click OK.

The dialog is closed and the specified CAN name applies to the block.

Note

Since you have not specified a database file yet, you cannot create an S-function for the MainBlock.

4 Open the CAN_MFile_M2 MainBlock.

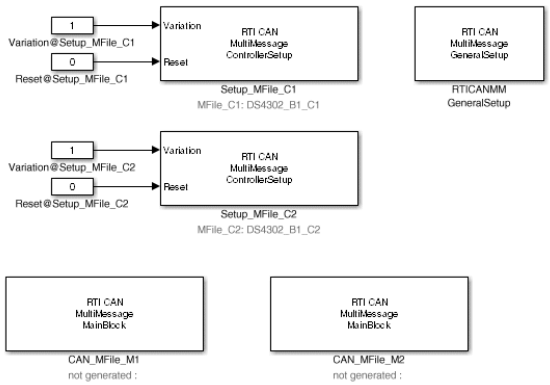
5 On the General Settings page, specify the following settings:

Setting	Entry / Selection
CAN name	MFile_M2
Selected ControllerSetup block	MFile_C2

6 Click OK to close the dialog.

7 Save your model.

Result



You built a simple model with two CAN controllers. You parameterized the RTICANMM ControllerSetup blocks, specified the CAN bus names, and selected the relevant ControllerSetup blocks for the RTICANMM MainBlocks. Since you cannot generate the MainBlocks without specifying a database file, you only applied the settings to the MainBlocks and closed the dialogs.

What's next

In the next step, you can define the CAN communication of your model in an M file.

Related topics

Basics

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database..... 373
Overview of Lesson 14..... 373

Step 2: How to Define CAN Communication in an M File

Objective

When you define CAN communication in an M file, you can create the M file from scratch, or use an existing M file as a template. Using a template allows you to adapt the template contents to your needs by modifying the ECU, message and signal entries.

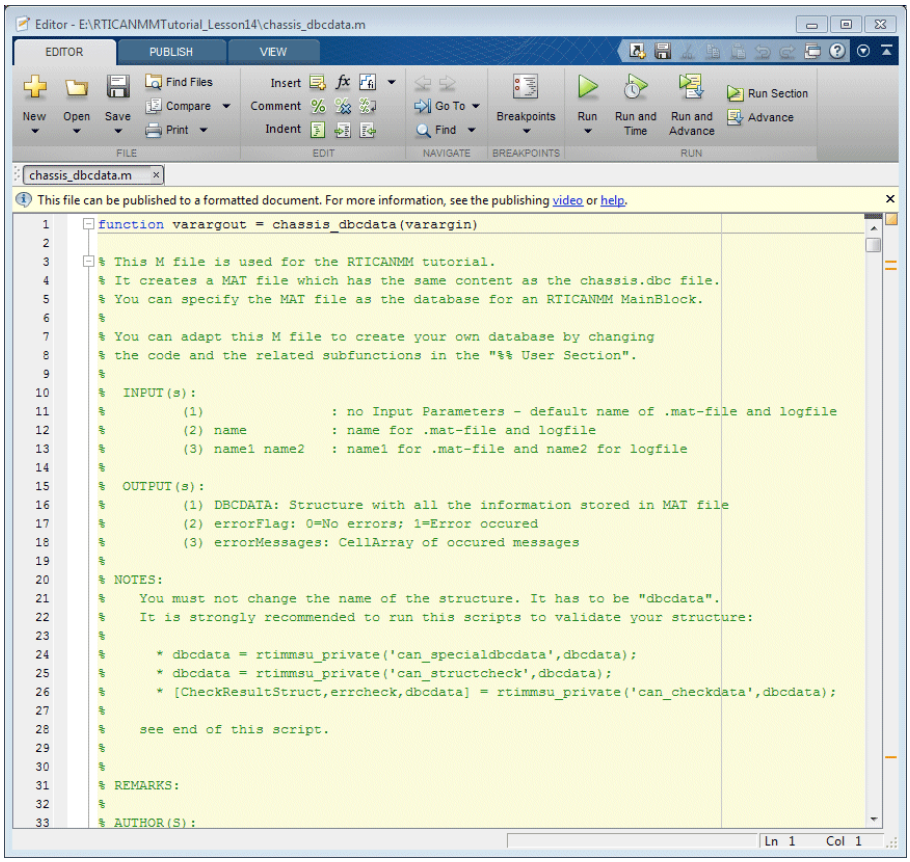
In this step, you will use the chassis_dbcdata.m file as a template. The chassis_dbcdata.m file describes the chassis.dbc and contains all the functions you need to generate an RTICANMM-compliant MAT file. You will rename the file chassis_tutorial.m and add an ECU with one message and four signals.

Part 1

To define general settings of the database file

1 In MATLAB, double-click the chassis_dbcdata.m file in the Current Folder window.

The chassis_dbcdata.m file opens in a separate editor window.



2 Save the file as **`chassis_tutorial.m`** in your RTICANMMTutorial_Lesson14 folder.

3 In line 1 of the chassis_tutorial.m file, replace **`chassis_dbcdata`** by **`chassis_tutorial`**.

```
1   function varargout = chassis_tutorial(varargin)
```

4 Go to **`% IFCNGENERALDATA`** in line 157.

```
function dbcdata = iFCNGeneralData(OPTS)
% ============================================================ %
% IFCNGENERALDATA
%
% INPUT PARAMETERS: (1) OPTS
%
%
% OUTPUT PARAMETERS: (1) DBCDATA
%
% ============================================================ %
    dbcdata.GeneralData.FileName =  [mfilename, '.m'];          % Name of this database
    [fp,fn,fe] = fileparts(OPTS.LogFilePath);
    dbcdata.GeneralData.LogFileName = [fn, fe];
    dbcdata.GeneralData.Path = fp;
    dbcdata.GeneralData.FileInfo = dir([mfilename('fullpath'), '.m']);
return % end of method :  iFCNGeneralData
```

This code block defines general settings of the database file to be generated.

5 Change the `dbcdata.GeneralData.FileName` name from `[mfilename, '.m']` to `'chassis_tutorial.m'`.

Interim result

You renamed the chassis_dbcdata.m file and adapted the file structure to the new name. You can now define the new ECU and the message. Continue with part 2.

Part 2

To define an ECU and a message in an M file

1 Go to `%% ECU Data` in line 80.

```
%% ECU Data
% Add all ECUs
    dbcdata = iFCNADD_ECU(dbcdata, 'CENTRAL_GATEWAY' , 0);
    dbcdata = iFCNADD_ECU(dbcdata, 'CRASH_SYSTEM'    , 1);
    dbcdata = iFCNADD_ECU(dbcdata, 'ENGINE_CONTROL'  , 2);
    dbcdata = iFCNADD_ECU(dbcdata, 'IGNITION'        , 3);
    dbcdata = iFCNADD_ECU(dbcdata, 'STEERING_WHEEL'  , 4);
```

This code block adds ECUs to the chassis_tutorial.m file.

2 Add the following code at the end of this code block to define a new ECU named **`TUTORIAL_ECU`**:

```
dbcdata = iFCNADD_ECU(dbcdata, 'TUTORIAL_ECU'  , 5);
```

You added an ECU named TUTORIAL_ECU and an ECU address of 5 to the chassis_tutorial.m file.

Note

- The ECU name only supports the characters 'A - Z', 'a - z', '0 - 9', and '_'.
- The ECU name must not start with a numeral.
- The ECU address has to be unique in the database.

3 Go to `% FCNADD_MESSAGEs` in line 174.

```
% ======================================================================
function dbcdata = iFCNADD_MESSAGEs(dbcdata)
% FCNADD_MESSAGEs
%
% Adds all Messages to the data base
%
% INPUT PARAMETERS: (1)  dbcdata
%
%
% OUTPUT PARAMETERS: (1) dbcdata
%
% ============================================================= %
    Message = rticanmmdefaultdatabasedata('message');
    Message.MsgName             =  'CRASH_1';
    Message.MsgId               =  885;
    Message.MsgLength           =  4;
    Message.ECU                 =  'CRASH_SYSTEM';
    Message.MsgComment          =  {'Crash information to be displayed'};
    Message.MsgCycleTime        =  100;
    Message.MsgDelayTime        =  5  ;
    Message.MsgStartValue       =  '3F373F3F3F070000h';
    Message.SendECU             =  {'CRASH_SYSTEM'};
    Message.ReceiveECU          =  {'CENTRAL_GATEWAY'};
    dbcdata = iFCNADD_MESSAGE(dbcdata, Message);
```

This code block defines all the messages of the chassis_tutorial.m file. You can add messages and define message-specific settings. For each message you want to define, you have to call the `rticanmmdefaultdatabasedata('message')` function. This RTICANMM-internal function contains all the available message settings and defines default values.

4 For an overview of the available messages-specific settings and their default values, type
`a = rticanmmdefaultdatabasedata('message')`
in your MATLAB Command Window and press **Enter**.

All the available message settings and their default values are displayed.

```
>> a=rticanmmdefaultdatabasedata('message')

a =

  struct with fields:

              MsgName: ''
                MsgId: 0
            MsgIdFormat: 0
             MsgLength: 0
                   ECU: ''
            MsgComment: {}
          MsgCycleTime: 0
          MsgDelayTime: 0
           MsgBaseTime: 0
         MsgUpdateTime: 0
          MsgUpdateNum: 1
           MsgSendType: ''
         MsgStartValue: '0000000000000000h'
           J1939active: 0
            J1939PGPrio: 0
            J1939PGDest: 0
             J1939PGSrc: 0
           J1939Mapping: ''
                SendECU: {}
             ReceiveECU: {}
                 Signal: []
    MsgCycleTimeActive: 0
            MessageType: 'StandardCAN'
              CANFD_BRS: 0
     GenMsgChkConstant: 0
    GenMsgPDUConstants: ''

>>
```

In your M File, you only need to define message settings whose default values you want to change for a message.

Note

- You must define at least an `MsgName` and `MsgId` for each message. All the other possible message settings are optional.
- The message name only supports the characters 'A - Z', 'a - z', '0 - 9', and '_'.
- The message ID has to be unique in the database.

5 In your chassis_tutorial.m file, add the following code at the end of the `% FCNADD_MESSAGEs` code block:

```
Message = rticanmmdefaultdatabasedata('message');
Message.MsgName              =  'TUTORIAL_MESSAGE';
Message.MsgId                =  777;
Message.MsgLength            =  3;
Message.ECU                  =  'TUTORIAL_ECU';
Message.MsgCycleTime         =  1000;
Message.ReceiveECU           =  {'ENGINE_CONTROL' };
dbcdata = iFCNADD_MESSAGE(dbcdata, Message);
```

This code adds a new message named **TUTORIAL_MESSAGE** to your chassis_tutorial.m file and defines message-specific settings.

6 Save your chassis_tutorial.m file.

Interim result

You added an ECU named Tutorial_ECU and a message named TUTORIAL_MESSAGE to the chassis_tutorial.m file. You specified the message ID, message length, sending ECU, cycle time, and the receiving ECU for the new message. You can now add signals to the TUTORIAL_MESSAGE message and define signal-specific settings. Continue with part 3.

Part 3

To define signals of the TUTORIAL_MESSAGE message

1 Go to `% FCNADD_SIGNALs` in line 335.

```
335 function dbcdata = iFCNADD_SIGNALSs(dbcdata)
    % FCNADD_SIGNALSs
    %
    % Adds all Signals to the data base
    %
    % INPUT PARAMETERS: (1)  dbcdata
    %
    %
    % OUTPUT PARAMETERS: (1) dbcdata
    %
    % ==============================================================
    %% CRASH_1 ID 885
        Signal = rticanmmdefaultdatabasedata('signal');
        Signal.SgnName      = 'SRS_Indication_Lamp';
        Signal.SgnStartBit  = 0;
        Signal.SgnLength    = 2;
        Signal.InitialValue = 3;
        Signal.SgnComment   = 'SRS indication lamp request';
        Signal.ReceiveECUs  = ('CENTEAL_GATEWAY');
        dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,885);
```

This code block defines all the signals of the chassis_tutorial.m file. You can add signals and define signal-specific settings. For each signal you want to define, you have to call the `rticanmmdefaultdatabasedata('signal')` function. This RTICANMM-internal function contains all the available signal settings and defines default values.

2 For an overview of the available signal-specific settings and their default values, type
`a = rticanmmdefaultdatabasedata('signal')`
in your MATLAB Command Window and press **Enter**.

All the available signal settings and their default values are displayed.

```
>> a = rticanmmdefaultdatabasedata('signal')

a =

  struct with fields:

          SgnName: ''
      SgnStartBit: 0
        SgnLength: 0
       ByteLayout: 1
           Factor: 1
           Offset: 0
       SgnComment: ''
          SgnUnit: ''
              Min: 0
              Max: 0
      ReceiveECUs: {}
    ValueNamePairs: {}
          SgnType: 0
        ModeValue: -1
         DataType: 0
     InitialValue: 0
    SgnErrorValue: ''

>>
```

In your M file, you only need to define signal settings whose default values you want to change for a signal.

Note

- You must define at least a `SgnName`, `SgnStartBit`, and `SgnLength` for each signal. All the other possible signal settings are optional.
- The signal name only supports the characters 'A - Z', 'a - z', '0 - 9', and '_'.
- The signal name has to be unique in the message the signal is defined for.

3 In your chassis_tutorial.m file, add the following code at the end of the `% FCNADD_SIGNALSs` code block to define four signals for the TUTORIAL_MESSAGE:

```
%% TUTORIAL_MESSAGE ID 777
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode';
Signal.SgnStartBit  = 0;
Signal.SgnLength    = 8;
Signal.SgnType      = 1;
Signal.SgnComment   = 'Mode signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode_1';
Signal.SgnStartBit  = 8;
Signal.SgnLength    = 8;
Signal.SgnType      = 2;
Signal.ModeValue    = 1;
Signal.InitialValue = 3;
Signal.SgnComment   = 'Mode-dependent signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode_2';
Signal.SgnStartBit  = 8;
Signal.SgnLength    = 8;
Signal.SgnType      = 2;
Signal.ModeValue    = 2;
Signal.InitialValue = 4;
Signal.SgnComment   = 'Mode-dependent signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
```

```
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Standard';
Signal.SgnStartBit  = 16;
Signal.SgnLength    = 8;
Signal.Factor       = 3.141;
Signal.Min          = 0;
Signal.Max          = 360;
Signal.SgnUnit      = 'Degrees';
Signal.SgnComment   = 'Standard signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
```

4 Save your chassis_tutorial.m file and close it.

Result

You extended the CAN communication defined in the chassis_dbcdata.m file. You used the chassis_dbcdata.m file as a template and renamed it chassis_tutorial.m. In the chassis_tutorial.m file, you defined an ECU named TUTORIAL_ECU, and the following message and signals:

- *TUTORIAL_MESSAGE*:

```
Message = rticanmmdefaultdatabasedata('message');
Message.MsgName             =  'TUTORIAL_MESSAGE';
Message.MsgId               =  777;
Message.MsgLength           =  3;
Message.ECU                 =  'TUTORIAL_ECU';
Message.MsgCycleTime        =  1000;
Message.ReceiveECU          =  ('ENGINE_CONTROL');
dbcdata = iFCNADD_MESSAGE(dbcdata, Message);
```

  This message has an ID of 777 (decimal format), a message length of 3 bytes, and a cycle time of 1000 ms. The sending ECU is TUTORIAL_ECU, the receiving ECU is ENGINE_CONTROL.

- *Tutorial_Mode*:

```
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode';
Signal.SgnStartBit  = 0;
Signal.SgnLength    = 8;
Signal.SgnComment   = 'Mode signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
```

  This signal starts at bit 0 and has a signal length of 8 bit. The signal type value is 1, i.e., this signal is a mode signal. The signal value triggers the transmission of the mode-dependent signals in the TUTORIAL_MESSAGE.

- *Tutorial_Mode_1, Tutorial_Mode_2*:

```
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode_1';
Signal.SgnStartBit  = 8;
Signal.SgnLength    = 8;
Signal.SgnType      = 2;
Signal.ModeValue    = 1;
Signal.InitialValue = 3;
Signal.SgnComment   = 'Mode-dependent signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);

Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Mode_2';
Signal.SgnStartBit  = 8;
Signal.SgnLength    = 8;
Signal.SgnType      = 2;
Signal.ModeValue    = 2;
Signal.InitialValue = 4;
Signal.SgnComment   = 'Mode-dependent signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
```

These signals both start at bit 8 and have a signal length of 8 bit each. Since they use the same bits, you defined a signal type of 2, i.e., these signals are mode-dependent.

For Tutorial_Mode_1, you defined a mode value of 1 and an initial value of 3. According to the defined mode value, this signal will be transmitted each time the Tutorial_Mode signal has a value of 1. The Tutorial_Mode_1 signal will be transmitted with its initial value.

For Tutorial_Mode_2, you defined a mode value of 2 and an initial value of 4. This signal will be transmitted with its initial value each time the Tutorial_Mode signal has a value of 2.

- *Tutorial_Standard*:

```
Signal = rticanmmdefaultdatabasedata('signal');
Signal.SgnName      = 'Tutorial_Standard';
Signal.SgnStartBit  = 16;
Signal.SgnLength    = 8;
Signal.Factor       = 3.141;
Signal.Min          = 0;
Signal.Max          = 360;
Signal.SgnUnit      = 'Degrees';
Signal.SgnComment   = 'Standard signal';
Signal.ReceiveECUs  = {'ENGINE_CONTROL'};
dbcdata = iFCNADD_SIGNAL(dbcdata,Signal,777);
```

This signal is a standard signal. It starts at bit 16 and has a length of 8 bit. You defined a minimum value (0), a maximum value (360), a scaling factor (3.141), and a unit (degrees) for it.

What's next

In the next step, you can run the M file to generate a MAT file from it and specify the MAT file as the database for your model.

Related topics

Basics

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database..... 373
Overview of Lesson 14..... 373

Step 3: How to Use an M File as the Database

Objective

M files cannot be used directly as a database for a model using RTICANMM blocks. To use an M file as the database, you have to generate a MAT file from it.

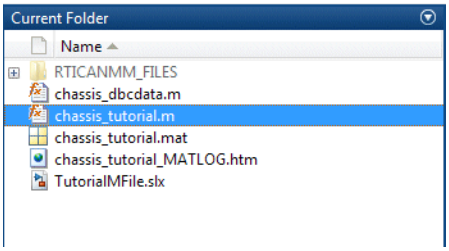
In this step, you will generate a MAT file from the chassis_tutorial.m file you edited in the previous step, and specify the MAT file as the database of your model. Additionally, you will configure the CAN communication of the RTICANMM MainBlocks of your model and observe the behavior of the TUTORIAL_MESSAGE in ControlDesk.

Part 1

To generate a MAT file and specify it as the database

1 In MATLAB, right-click the chassis_tutorial.m file in the Current Directory window and select Run from the context menu.

A MAT file named chassis_tutorial.mat is generated and displayed in the Current Directory window.



Tip

The chassis_tutorial_MATLOG.htm file is created automatically during the consistency check. Open this file to take a look at the results of the check.

2 Double-click the CAN_MFile_M1 MainBlock in the TutorialMFile model to open it.

3 On the General Settings page, specify the chassis_tutorial.mat file as the Database.

4 Click OK to close the dialog.

5 Double-click the CAN_MFile_M2 MainBlock to open it.

6 On the General Settings page, specify the chassis_tutorial.mat file as the Database.

Interim result

You generated a MAT file from the chassis_tutorial.m file and specified it as the database of the MainBlocks in your model. You can now configure the CAN communication of your model. Continue with part 2.

Part 2

To configure the CAN communication of your model

1 In the dialog tree of the CAN_MFile_M2 dialog, select Messages – Choose Messages – TX Messages.

2 On the TX Messages page, select None to specify that no messages are to be transmitted by this CAN controller.

3 Click Create.

The CAN_MFile_M2 MainBlock is created, and the specified settings apply to your model.

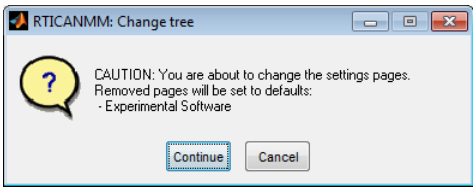
4 Click OK to close the dialog.

5 Double-click the CAN_MFile_M1 MainBlock to open it.

6 On the Tree Views page, select Filter Configuration Tree by Feature.

7 Clear all filters except for Instrumentation & Automation Interface and Manipulation Interface.

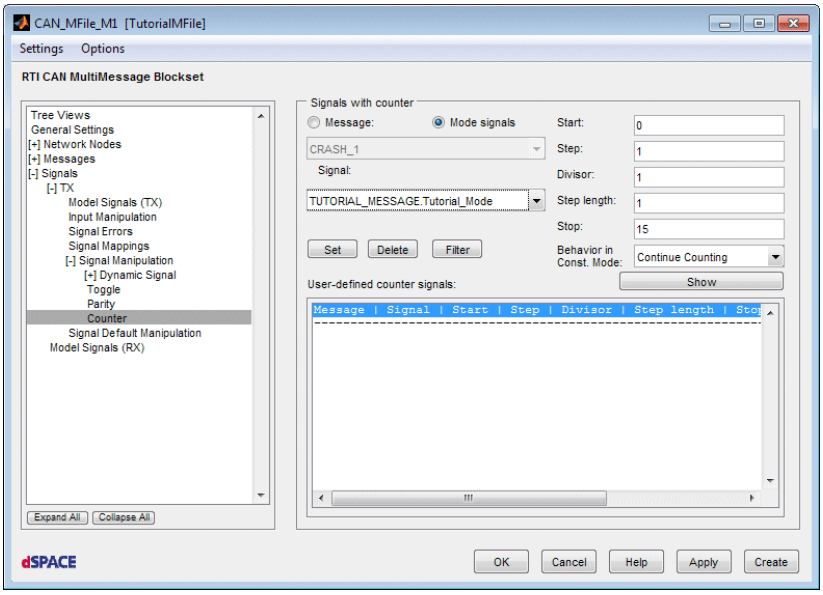
If a dialog similar to the following one is displayed, click Continue.



8 In the dialog tree, select Signals – TX – Signal Manipulation – Counter.

9 Select Mode signals.

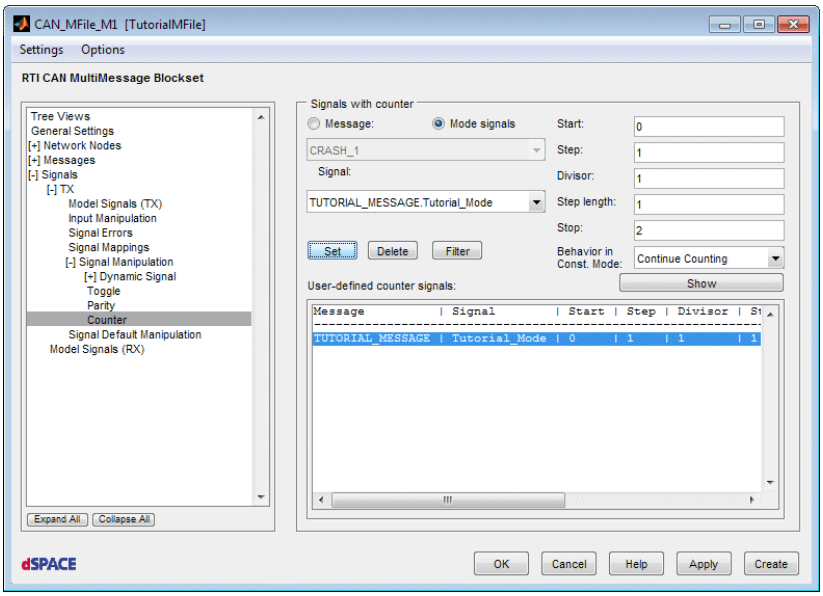
10 From the Signal drop-down list, select TUTORIAL_MESSAGE.Tutorial_Mode.



11 In the Stop edit field, type **2**.

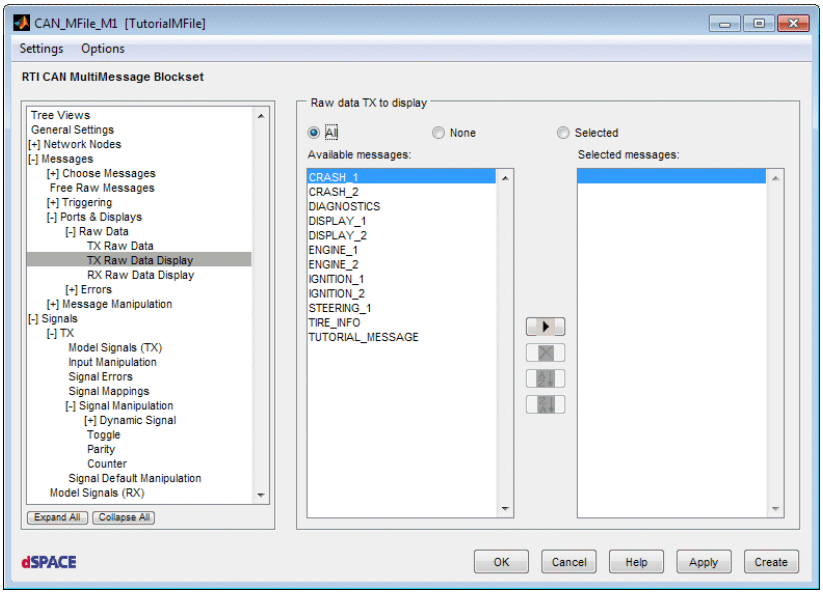
12 Click Set.

The TUTORIAL_MESSAGE.Tutorial_Mode signal has a stop value of 2 and is added to the User-defined counter signals list.



13 In the dialog tree, select Messages – Ports & Displays – Raw Data – TX Raw Data Display.

14 Select All.



The raw data of all the TX messages will be displayed in the message layouts in ControlDesk.

15 Click Create.

The CAN_MFile_M1 MainBlock is created, and the specified settings apply to your model.

16 Click OK to close the dialog.

17 Save your model.

Interim result

You specified the Tutorial_Mode signal as a counter signal. For an easier overview of the use of the TX message bytes, you activated to display the raw data of these messages in the message layouts. Since the MFile_C2 CAN controller is only needed to receive messages from the MFile_C1, you disabled all the TX messages in the MFile_M2 MainBlock. You can now observe the effects of your specifications in ControlDesk. Continue with part 3.

Part 3

To observe the effects of your specifications in ControlDesk

1 Build an application for your TutorialMFile model.

2 Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.

3 From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.

4 Click OK to confirm the default platform name.

The registered platform is added to the experiment.

5 From the platform's context menu in the Project controlbar, select Add Variable Description.

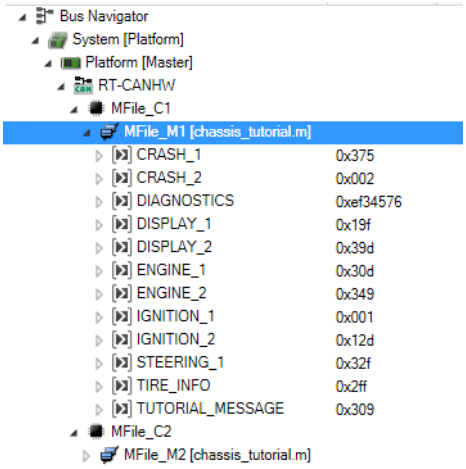
ControlDesk opens the Select Variable Description dialog.

6 In the Select Variable Description dialog, click Import and select the `TutorialMFile.sdf` file from your `RTICANMMTutorial_Lesson14` folder.

7 In the Select Variable Description dialog, click OK.

ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

8 Select the Bus Navigator tab.

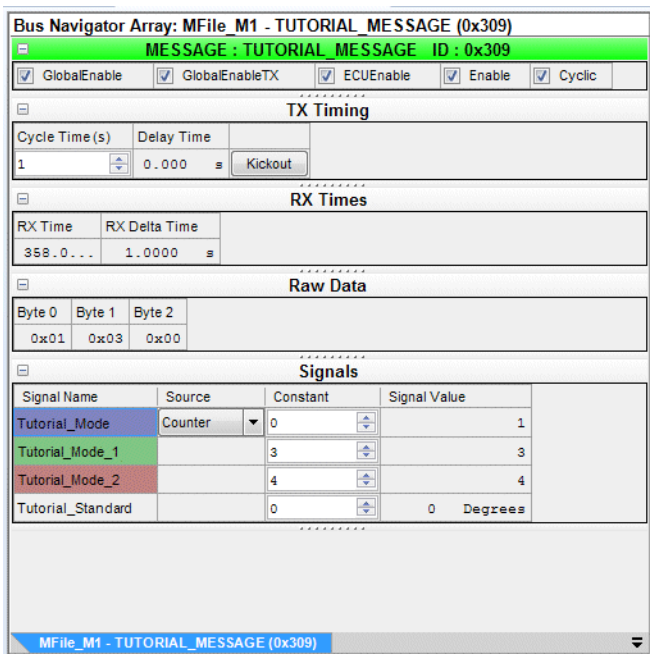


The Bus Navigator displays all the messages and the database name specified in the chassis_tutorial.m file.

9 Open the following layouts of TUTORIAL_MESSAGE:

CAN Bus	TUTORIAL_MESSAGE Layout
MFile_M1	TX
MFile_M2	RX

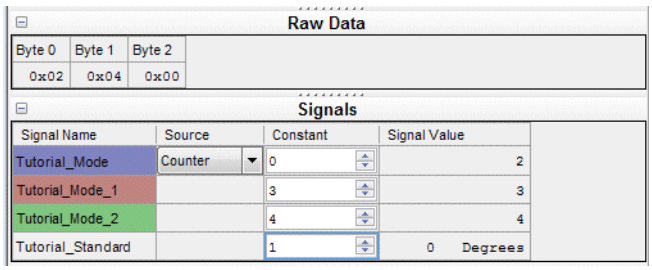
10 On the Home ribbon, click Status Control – Go Online to start online calibration.



- Both headers display the specified message ID in hexadecimal format.
- In the TX layout, Tutorial_Mode triggers the transmission of Tutorial_Mode_1 and Tutorial_Mode_2. A Tutorial_Mode value of 1 triggers the transmission of Tutorial_Mode_1, a Tutorial_Mode value of 2 triggers the transmission of Tutorial_Mode_2.
- Tutorial_Mode is transmitted via byte 0, Tutorial_Mode_1 and Tutorial_Mode_2 are transmitted via byte 1, and Tutorial_Standard is transmitted via byte 2.

11 In the Constant edit field of Tutorial_Standard, type **1** and press **Enter**.

The signal value and the raw data of byte 2 are 0.



This behavior occurs because of rounding effects caused by displaying a float variable for a 1-byte long signal. The signal value is calculated as follows:

`physical_signal_value = signal_factor * raw_data_value`

The raw data value is calculated as follows:

`raw_data_value = constant_value / signal_factor`

Since you specified a signal factor of 3.141 in the M file, the `raw_data_value` is:

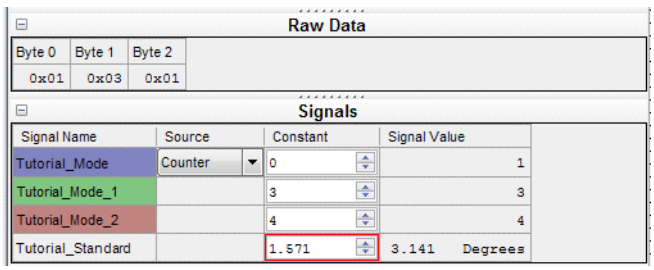
`raw_data_value = 1 / 3.141 = 0.3183`

The `raw_data_value` is rounded to 0 and the `physical_signal_value` is calculated according to the rounded raw data:

`physical_signal_value = 3.141 * 0 = 0`

12 In the Constant edit field of Tutorial_Standard, type **1.571** and press **Enter**.

The signal value is 3.141 and the raw data of byte 2 is 1.



The raw data calculation is:

`raw_data_value = 1.571/3.141 = 0.5001`

The raw data is rounded to 1 and the physical signal value is calculated according to the rounded raw data:
`physical_signal_value = 3.141 * 1 = 3.141`

Tip

- The more bytes are used for the calculation, the fewer are the rounding effects on the calculated values.
- To avoid the rounding effects, you can use the spin buttons to change the constant value.

13 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

14 Save the project.

Result

You generated a MAT file from the chassis_tutorial.m file and specified the MAT file as the database for both CAN controllers in your TutorialMFile model. You configured the CAN communication of your model in the RTICANMM MainBlocks and built an application of your model to observe the effects of your specifications in ControlDesk. Because of the specified signal length and signal factor of Tutorial_Standard, you observed rounding effects for the calculated raw data and signal values.

What's next

With this step, you finished using an M file as the database. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database..... 373
Overview of Lesson 14..... 373

Result of Lesson 14

Result

In this lesson, you learned the basic principles of defining CAN communication in an M file:

- You can define CAN communication directly in the M file or parse other file formats like XLS or XML via the M file for this purpose.
- You cannot use an M file as a database directly. You have to generate a MAT file from the M file and specify the MAT file as the database. When the M file is run in MATLAB, the MAT file is generated automatically.

- The M file must generate an RTICANMM-compliant MAT file according to a CAN structure named *dbcdata*. In the chassis_dbcdata.m file this structure is realized as follows:

Element to Generate DBCDATA Structure	Function / Loop / Condition	Called in Line ...	Defined in Line ...
General data definition	`iFCNGeneralData`	78	154 ... 169
ECU definition	`iFCNADD_ECU`	82 ... 86	82 ... 86
Message definition	`iFCNADD_MESSAGEs`	89	172 ... 320
Signal definition	`iFCNADD_SIGNALs`	92	324 ... 972
Data post-processing	for-loop	-	98 ... 115
RTICANMM post-processing	`rtimmsu_private('can_specialdbcdata',dbcdata)`	118	-
Structure check	`rtimmsu_private('can_structcheck',dbcdata)`	122	-
Consistency check	`rtimmsu_private('can_checkdata',dbcdata)`	126	-
Save as MAT file	if-else condition	-	135 ... 140

Tip

- The functions that start with `rtimmsu_private` are provided by RTICANMM so you can use them directly.
- To comply with this structure, use the chassis_dbcdata.m file as a template for your own use cases. You only need to change the ECU-, message-, and signal-specific settings.

- All the available message- and signal-specific settings and their default values are defined in the **rticanmmdefaultdatabasedata** function. You have to assign this function to each message and signal you want to add to your M file via `rticanmmdefaultdatabasedata('message')` and `rticanmmdefaultdatabasedata('signal')`, respectively. This allows you to define only message-/signal-specific settings whose default values you want to change.

Note

- For each message, you must define at least a `Message.MsgName` and `Message.MsgId`.
- For each signal, you must define at least a `Signal.SgnName`, `Signal.SgnStartBit`, and `Signal.SgnLength`.

- You can easily add and specify ECUs, messages, and signals in the M file.

Note

- The ECU, message, and signal names only support the characters 'A - Z', 'a - z', '0 - 9', and '_'.
- The ECU name must not start with a numeral.

- The ECU address and message ID have to be unique in the database, and the signal name has to be unique in a message. You should therefore check the defined settings for double definitions. In the chassis_dbcdata.m file, the check for double definitions is realized in the following lines:

Check for Double Definition of ...	Line
ECU address	1004 ... 1024
Message ID	976 ... 1000
Signal name	1035 ... 1073

You also observed rounding effects in the calculation of the raw data and the signal values of the Tutorial_Standard signal. The rounding effects were caused by the specified signal length and signal factor.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson14_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to work with end-to-end communication protection parameters in checksum algorithms.

If you do not need to work with checksums based on end-to-end communication protection parameters, you can continue with another lesson or the summary of this tutorial. Refer to Overview of Lessons on page 35 or Summary on page 457.

Related topics

Basics

Lesson 14 (Advanced): Defining CAN Communication via a User-Defined Database..... 373
Overview of Lesson 14..... 373

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters

Where to go from here

Information in this section

Overview of Lesson 15 400
In this lesson, you will work with end-to-end communication protection parameters in checksum algorithms.

Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters 401
In this step, you will work with checksum algorithms that are based on E2E protection parameters. You will specify a new database file containing defined E2E protection parameters for your model. You will enable the use of E2E protection parameters, select a checksum header file template, complete the checksum configuration, and assign the algorithms to signals.

Step 2: How to Check the Behavior of E2E Checksums in ControlDesk 417
In this step, you will observe the behavior of counter signals and checksums based on E2E protection data.

Step 3: How to Add a Checksum Case to the E2E Protection Header File 421
In this step, you will extend your E2E protection header file by adding a further checksum algorithm to it.

Result of Lesson 15 426
In this lesson, you worked with checksums based on E2E protection data.

Overview of Lesson 15

Introduction

The RTI CAN MultiMessage Blockset supports run-time access to end-to-end communication protection (E2E protection) parameters from AUTOSAR communication matrices and DBC files. The parameters can be used to implement checksum algorithms based on E2E protection parameters in the real-time application.

Tip

If you do not need to work with checksums based on E2E protection data, you can skip this lesson.

Using E2E protection parameters in user-defined checksum algorithms

You must enable the use of E2E protection parameters. If enabled, the E2E protection data defined in the database file is imported into an additionally generated E2E protection header file, which is included automatically in the S-function of the RTICANMM MainBlock. The imported E2E protection data is available for checksum algorithm implementation. The header file also contains the functions to call the E2E protection data in checksum algorithms.

What you will learn

In this lesson, you will learn the basic principles of using end-to-end communication protection parameters in user-defined checksum calculations.

- You will specify a database file containing defined E2E protection parameters for your model.
- You will use the E2E_checksum.h file as a template. The file contains some calculation algorithms (switch-case directives) based on E2E protection data.
- You will view search functions for specific E2E cases and functions to fetch E2E data.
- You will implement a further checksum algorithm based on E2E protection data.
- You will observe the behavior of checksums in ControlDesk.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 6.

Prepared demo

If you did not create your own model in lesson 6, you can use the prepared demo **Lesson6_TutorialModel.slx** as the starting point, refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- You have to know how to build an application for your model and how to check the application loaded on your real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters on page 401
- Step 2: How to Check the Behavior of E2E Checksums in ControlDesk on page 417
- Step 3: How to Add a Checksum Case to the E2E Protection Header File on page 421

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 15 on page 426.

Duration

Working through this lesson will take you about 45 minutes.

What's next

Now you can start to specify checksum algorithms that are based on E2E protection parameters, see Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters on page 401.

Related topics

Basics

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters..... 399

Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters

Objective

You can implement your own checksum algorithms based on E2E protection parameters in the real-time application.

In this step, you will specify a new database file containing defined E2E protection parameters for your model. You will enable the use of E2E protection

parameters, select a checksum header file template, complete the checksum configuration, and assign the algorithms to signals.

Part 1

To create the model and specify a new database containing E2E protection data

1. In your working folder, create a new folder named **`RTICANMMTutorial_Lesson15`**.
2. In MATLAB, change the Current Directory to the new RTICANMMTutorial_Lesson15 folder.
3. In your MATLAB Command Window, type **`rticanmm_tutorial`**.

   The rticanmm_tutorial_lib library opens.
4. Double-click Lesson 6 Results.
5. In the Get Tutorial dialog, click Only open model.

   The Lesson6_TutorialModel demo model opens
6. On the Simulation ribbon, click Save – Save as.
7. In the Save As dialog, name the model **`Lesson15_TutorialModel`** and save it to your RTICANMMTutorial_Lesson15 working folder.

   Tip

   If you created your own model in lesson 6, you can alternatively copy your TutorialModel model from your RTICANMMTutorial folder to the RTICANMMTutorial_Lesson15 folder and rename it.
   It is recommended to work with a new working folder and rename the model in this lesson. The model you created in lesson 6 is also used as the starting point of lesson 7, where it is required in unchanged form.

8. In the rticanmm_tutorial_lib library, double-click CAN_E2E_Demo.arxml.

   The CAN_E2E_Demo.arxml file is copied to your RTICANMMTutorial folder.
9. In the rticanmm_tutorial_lib library, double-click E2E_checksum.h.

   The E2E_checksum.h file is copied to your RTICANMMTutorial folder.
10. Close the rticanmm_tutorial_lib library.
11. If necessary, update the real-time hardware specified in the RTICANMM ControllerSetup blocks and the system target file according to your needs. For details, refer to Working with the Prepared Demo Results on page 32.
12. Double-click the RTICANMM GeneralSetup block in the model and select Options - Create S-function for All CAN Blocks.

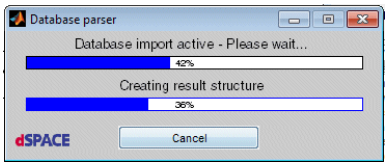
    The S-function is created and you can start working with the model.
13. Click OK to close the dialog.
14. In your model, double-click the CAN_Chassis_M1 block to open it.

    A dialog informs you that the database import failed.
15. Click OK to close the dialog.
16. Select the General Settings page.

17 Click .

18 In the File dialog, select the CAN_E2E_Demo.arxml file and click Open.

The Database Parser dialog opens and the selected database file is parsed automatically.



After parsing, the dialog closes.

19 In the CAN_Chassis_M1 [Lesson15_TutorialModel] dialog, click OK to close the dialog.

20 Open the CAN_Chassis_M2 MainBlock.

A dialog informs you that the database import failed.

21 Click OK to close the dialog.

22 Select the General Settings page.

23 Specify the CAN_E2E_Demo.arxml file as the new database.

24 Click OK to close the CAN_Chassis_M2 [Lesson15_TutorialModel] dialog.

25 Save your model.

Interim result

You created a model on the basis of the result of lesson 6, copied the required files to your working folder, and specified the CAN_E2E_Demo.arxml file as the new database for your model. CAN_E2E_Demo.arxml contains E2E protection data. You can now configure the CAN communication of the CAN_Chassis_M1 MainBlock, which is only used to transmit messages, specify counter signals to check correct message transmission, and display the raw data of TX messages. Continue with part 2.

Part 2

To configure CAN communication of the CAN_Chassis_M1 MainBlock, specify counter signals, and display TX raw data

1 Open the CAN_Chassis_M1 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

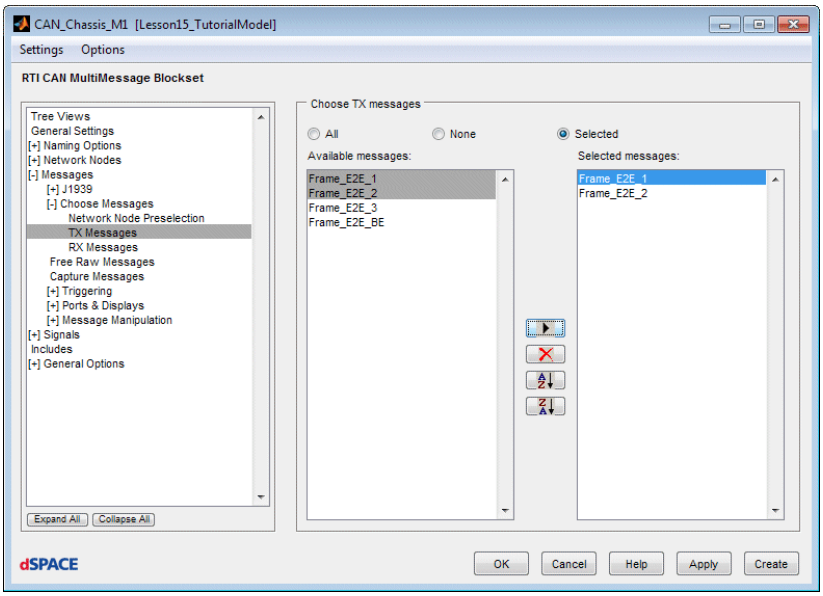
3 In the Available messages list, select the Frame_E2E_1 and Frame_E2E_2 messages.

Tip

You can use `Shift` and `Ctrl` for multiselection.

4 Click .

The selected messages are added to the **Selected messages** list.



5 In the dialog tree, select the **Messages – Choose Messages – RX Messages** page.

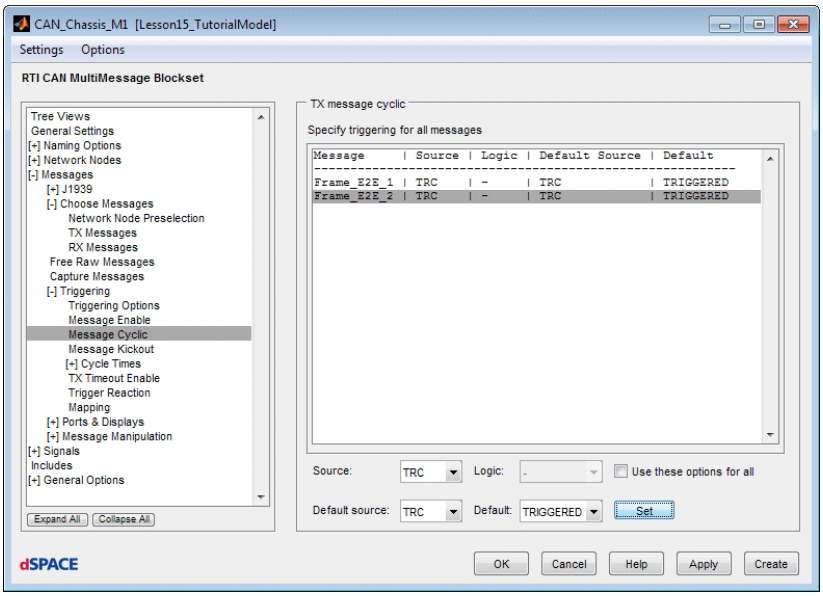
6 Select **None (just loopback)**.

This MainBlock is only used to transmit messages. You can therefore disable the reception of messages. However, all the messages transmitted by a MainBlock are always received by the same MainBlock as loopback messages.

7 In the dialog tree, select **Messages – Triggering – Message Cyclic**.

On the Message Cyclic page, you can enable or disable cyclic transmission for all the messages displayed in the **Specify triggering for all messages** list or just for some of them.

8 From the **Default** drop-down list, select **Triggered** for both messages.



You disabled cyclic message transmission for the **Frame_E2E_1** and **Frame_E2E_2** messages by default. During run time, cyclic transmission can be enabled via ControlDesk.

9 In the dialog tree, select **Signals – TX – Signal Manipulation – Counter**.

This page lets you specify counter signals. The values of counter signals change with every message transmission. TX messages are counted when they are transmitted. RX messages are counted when they are received. For each message, you can specify one or more counter signals that provide the number of message transmissions/receptions. You can specify the counter start value, the increment, and the counter stop value. Each time the counter reaches the stop value, it turns around to the counter start value.

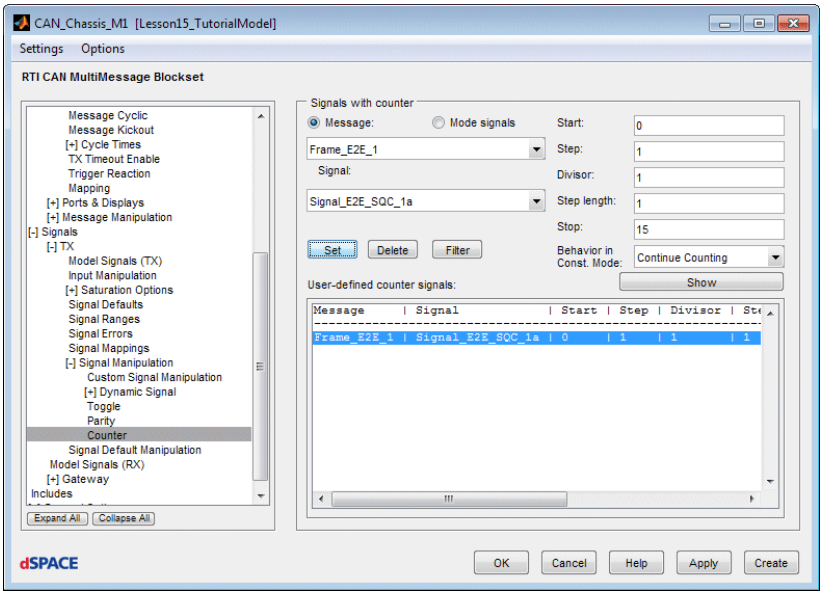
10 Specify the following settings for a counter signal for the **Frame_E2E_1** message:

Setting	Selection/Value
Message	Frame_E2E_1
Signal	Signal_E2E_SQC_1a
Start	0
Step	1
Divisor	1
Step length	1
Stop	15
Behavior in Const. mode	Continue Counting

11 Click **Set**.

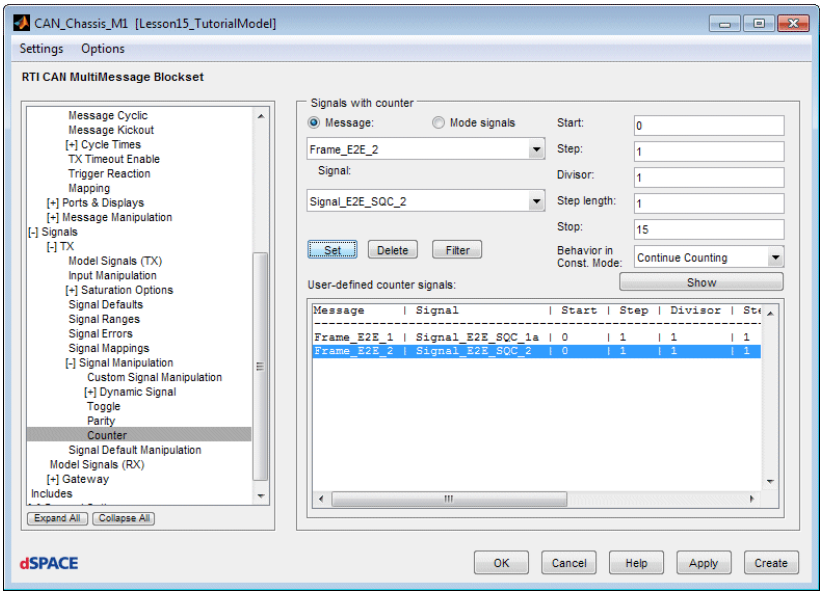
Note

You always have to click **Set** to apply your settings.



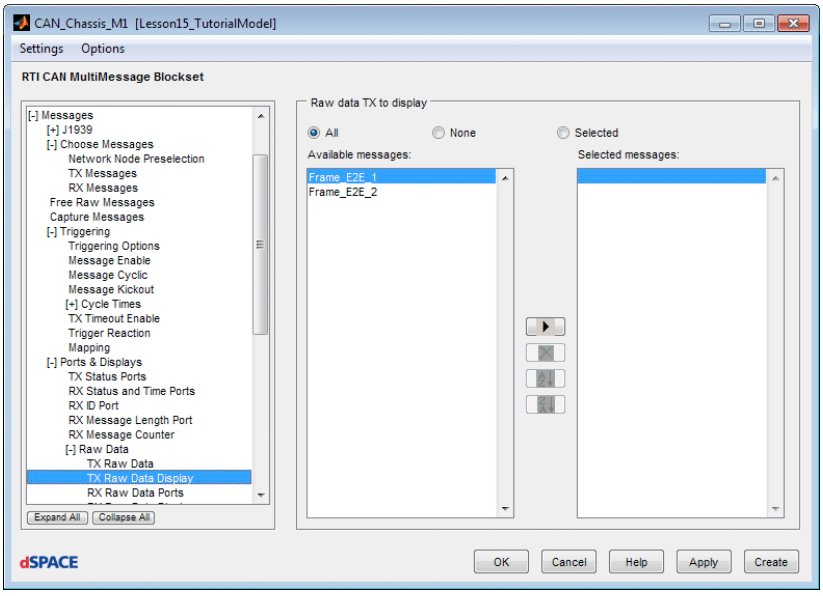
12 Specify another counter signal using the following settings:

Setting	Selection/Value
Message	Frame_E2E_2
Signal	Signal_E2E_SQC_2
Start	0
Step	1
Divisor	1
Step length	1
Stop	15
Behavior in Const. mode	Continue Counting



13 In the dialog tree, select Messages – Ports & Displays – Raw Data – TX Raw Data Display.

14 Select All.



The raw data of all bytes of all the TX messages will be displayed in the message layouts in ControlDesk.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You selected Frame_E2E_1 and Frame_E2E_2 as TX messages and disabled cyclic message transmission for them by default. You specified a counter signal for each TX message. For a better understanding of checksum behavior, you activated the display of TX raw data. You can now select the checksum header file template to be used and specify checksum case identifiers. Continue with part 3.

Part 3

To select a header file and specify checksum case identifiers

1. In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

   On the Checksum Definition page, you can define a checksum header file and specify checksum case identifiers.

2. Select the Enable E2E Protection checkbox.

   You enabled the use of E2E protection parameters. Enabling the use of E2E protection parameters generates an additional header file (named `RTICANMM_MAIN_Chassis_M1_E2EP.h`) and includes it into the S-function of the RTICANMM MainBlock. This file contains the E2E protection description data defined in the database and the functions to call this data in checksum algorithms.

3. Click .

4. In the File dialog, select the E2E_checksum.h file from your working folder and click Open.

   You selected the E2E_checksum.h file as the checksum header file to be used. The E2E_checksum.h file is a template containing two checksum cases (switch-case directives) using E2E data provided by the `CAN_E2E_Demo.arxml` database.

5 Click **Edit Header File**.

The `E2E_checksum.h` header file opens in an editor.

```
E2E_checksum.h
1  /*****************************************************************************************************/
2  /*   Automatic generated File from RTI CAN MultiMessage Blockset:                                    */
3  /*      CRC.h                                                                                        */
4  /* INPUT PARAMETER(S)                                                                                */
5  /*          (1) option : crc / crccheck   0: TX  (calculate)  1: RX (check)                          */
6  /*          (2) Pointer to MsgData                                                                   */
7  /*          (3) Index of CRC see array in rticanmm_customer                                          */
8  /*          (4) StartBit of ChecksumSignal                                                           */
9  /*          (5) Length of ChecksumSignal                                                             */
10 /*                                                                                                   */
11 /* OUTPUT PARAMETER(S)                                                                               */
12 /*          (1) crccheck  (RX)                                                                       */
13 /*   DESCRIPTION:                                                                                    */
14 /*      defines customer-specific CRCs                                                               */
15 /*   AUTHOR(S):                                                                                      */
16 /*      generated by dSPACE(R) RTICANMM                                                               */
17 /* *****************************************************************************************************/
18 static UInt8 rticanmmcrc(int crcoption, RTICANMMMsgStruct* Msg, int crctype, int CsBitPos, int CsLength)
19 {
20     UInt8  checksum = 0;
21     int i;
22
23     // calculate bit offset and byte position
24     // since all checksum positions for this database are aligned with start of the byte -> BitOffset is not used
25     UInt32 BitOffset  = (CsBitPos & 0x07);
26     UInt32 BytePos = CsBitPos >> 3;
27
28
29     // Get message ID from message structure
30     UInt32 msgId = Msg->identifier;
31     // Get message ID Format from message structure
32     UInt32 msgIdFormat = Msg->idformat;
33
34     // Initialize E2E Index list, this array is given to search function by reference
35     UInt32 E2EIndexListForMessage[MaxNumOfE2EPerMsg];
36     // Initialize new variable for number of E2E cases for specific message, this variable is given to search functic
37     UInt32 numOfE2EForMessage;
38
39     // Initialize counter and counter position
40     UInt32 counter;
```

6 For a better understanding of the code, you can open a preview of the E2E protection header file, which will be generated and included into the S-function of the **CAN_Chassis_M1** MainBlock. The file contains all the E2E protection data defined in the database and the functions to call this data in checksum algorithms. To get the preview, click **Show E2E File** on the Checksum Definition page.

The preview of the `RTICANMM_MAIN_Chassis_M1_E2EP.h` file opens in an editor. In the `USER RELEVANT FUNCTIONS` block at the beginning of the file, you find information on the functions used to fetch the E2E data.

E2E_checksum.h | RTICANMM_MAIN_Chassis_M1_E2EP.h

```
1  /*****************************************************************************************************/
2  /*                        Automatic generated File from RTI CAN MultiMessage Blockset:                 */
3  /*    INPUT PARAMETER(S)                                                                               */
4  /*        ....                                                                                         */
5  /*                                                                                                     */
6  /*    OUTPUT PARAMETER(S)                                                                              */
7  /*        ....                                                                                         */
8  /*                                                                                                     */
9  /*    DESCRIPTION:                                                                                     */
10 /*        End-to-End Protection Header File                                                            */
11 /*    AUTHOR(S):                                                                                       */
12 /*        Generated by dSPACE(R) RTICANMM                                                              */
13 /*****************************************************************************************************/
14
15 #include "string.h"
16
17 /*****************************************************************************************************/
18 /************************************* USER RELEVANT FUNCTIONS ***************************************/
19 /*****************************************************************************************************/
20
21 #define MaxNumOfE2EPerMsg 2 // Number of maximum E2E Protection cases per message for current database.
22 #define MaxSizeOfDataIDs 16 // Maximum number of DataIDs per E2E Protection Case for current database.
23 #define NumOfMsgWithE2E 4 // Number of messages with at least one E2E Protection Case for current database.
24 #define NumOfE2ECases 6 // Number of all E2E Protection Cases for current database.
25 #define MaxProfileNameLength 11 // Size of largest ProfileName string. String terminator is included in the length.
26
27 typedef struct
28 {
29     // Data values for each E2E Data:
30     char ProfileName[MaxProfileNameLength];
31     UInt8 DataIDs[MaxSizeOfDataIDs];
32     UInt32 DataOffset;
33     UInt32 CounterPosition;
34     UInt32 CRCPosition;
35     UInt32 DataIDMode;
36     UInt32 DataLength;
37     UInt32 MaxDeltaCounterInit;
38     UInt32 MaxNoNewOrRepeatedData;
39     UInt32 SyncCounterInit;
40     // Actual number of DataIDs read from database:
```

7 Have a look at the code lines and comments of the `E2E_checksum.h` file. Some of the functions are described below:

- `getE2EIndexesForMessage(E2EIndexListForMessage, &numOfE2EForMessage, msgId, msgIdFormat)`

  The `msgId` and `msgIdFormat` attributes are used to return the number of E2E protection data defined in the database for the message and to return an array of indices used to access the E2E data available in the generated data structure in the E2E protection header file. The return value of this function is 1 if E2E data is found. If no data is found, the function returns 0.

- `getE2EDataOffset(&DataOffset, currentE2EIndex);`

  The function uses `currentE2EIndex` to fetch the `DataOffset` for the given E2E case. The `DataOffset` attribute represents the start bit of a PDU. Because each E2E case is linked to the PDU, the E2E case can be identified within a CAN message by the `DataOffset`. RTICANMM only supports PDUs that are equal to complete message data, which means that their `DataOffset` is 0.

- `getE2ECounterPosition(&counterPosition, currentE2EIndex);`

  The function gets the counter position from the E2E case in the database. The counter position then can be used to determine the counter value (see code line
  `counter = Msg->RAW_DATA[counterPosition] & 0x0F;`).

When defining the counter signals on the **Counter** page, you specified the counter settings according to the counter positions. In the checksum algorithms, the specified counter is used as index for the DataIDs array.

- `getE2EDataIDs(&DataID, currentE2EIndex, 0);`

  The function gets the first DataID found for the current E2E case for the message in the database ('first DataID' means DataID with index = 0.) Typical use cases have one or 16 DataIDs per E2E case.

  To find out how many DataIDs are defined in the database, you can use the following function:

  `getE2EDataIDs_size(UInt32 *Data, UInt32 E2EIndex);`

  The number of DataIDs read from the database is stored in the first parameter.

- `getE2EAllData(DataStruct, currentE2EIndex);`

  The function returns all the data found for the current E2E case for the message in the database. For information on the used data structure, see the preview of the `RTICANMM_MAIN_Chassis_M1_E2EP.h` file.

  However, fetching all data in a single function call can have disadvantages. The function might be called for data that is not required or does not even exist. As a consequence, using this function can be more time-consuming for the real-time system than using several functions to fetch specific data.

8 Close the `E2E_checksum.h` and the preview of the `RTICANMM_MAIN_Chassis_M1_E2EP.h` files without saving.

9 On the Checksum Definition page, select all entries in the **Defined cases identifier** list and click .

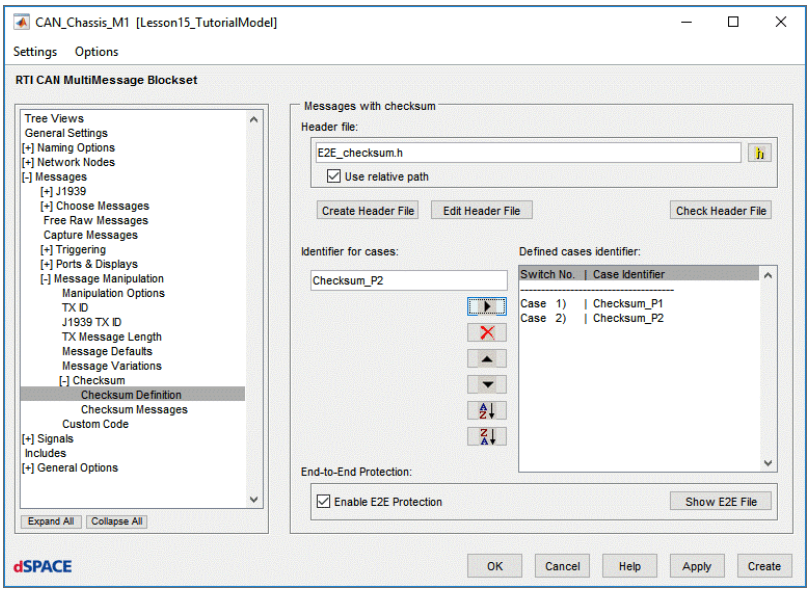
The **Defined cases identifier** list is cleared. You can now specify new case identifiers.

10 In the **Identifier for cases** edit field, type **`Checksum_P1`**.

11 Click .

Checksum_P1 is added to the **Defined cases identifier** list as case 1.

12 Define another case identifier named **`Checksum_P2`**.



The two checksum cases defined in the `E2E_checksum.h` header file are related to these case identifiers.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You selected the checksum header file template to be used and specified checksum case identifiers for the checksum cases defined in the checksum header file. You can now assign the checksum algorithms to messages. Continue with part 4.

Part 4

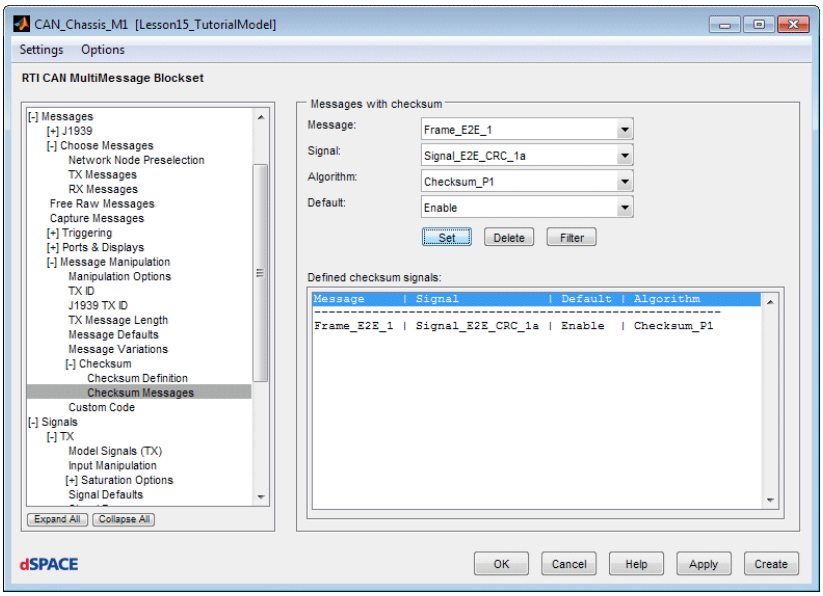
To assign checksum algorithms to messages

1. In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Messages.

   On the Checksum Messages page, you can assign checksum algorithms to messages.
2. From the Message drop-down list, select Frame_E2E_1.
3. From the Signal drop-down list, select Signal_E2E_CRC_1a as the checksum signal.
4. From the Algorithm drop-down list, select Checksum_P1.
5. From the Default drop-down list, select Enable.

6 Click Set.

Checksum_P1 is specified as the default checksum algorithm for Frame_E2E_1. Checksum calculation for the message is enabled by default.

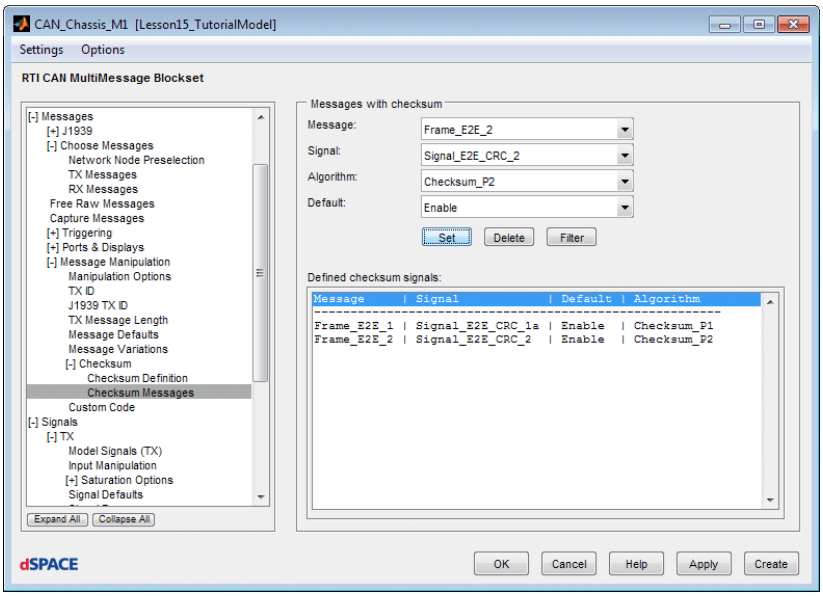


7 Specify another assignment using the following settings:

Setting	Selection
Message	Frame_E2E_2
Signal	Signal_E2E_CRC_2
Algorithm	Checksum_P2
Default	Enable

8 Click Set.

Checksum_P2 is specified as the default checksum algorithm for Frame_E2E_2. Checksum calculation for the message is enabled by default.



9 Click Create.

The CAN_Chassis_M1 MainBlock is created.

10 Click OK to close the CAN_Chassis_M1 [Lesson15_TutorialModel] dialog.

11 Delete the CRASH_System, Triggering to RTICANMM, and Mapping to RTICANMM blocks and their connections from the model.

12 Save your model.

Interim result

You assigned checksum algorithms to the TX messages of the CAN_Chassis_M1 MainBlock and completed checksum configuration for the CAN_Chassis_M1 MainBlock. You can now specify the CAN communication and the checksum settings for the CAN_Chassis_M2 MainBlock. Continue with part 5.

Part 5

To configure the CAN communication of the CAN_Chassis_M2 MainBlock and specify checksum settings for the RX messages

1 Open the CAN_Chassis_M2 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

3 Select None.

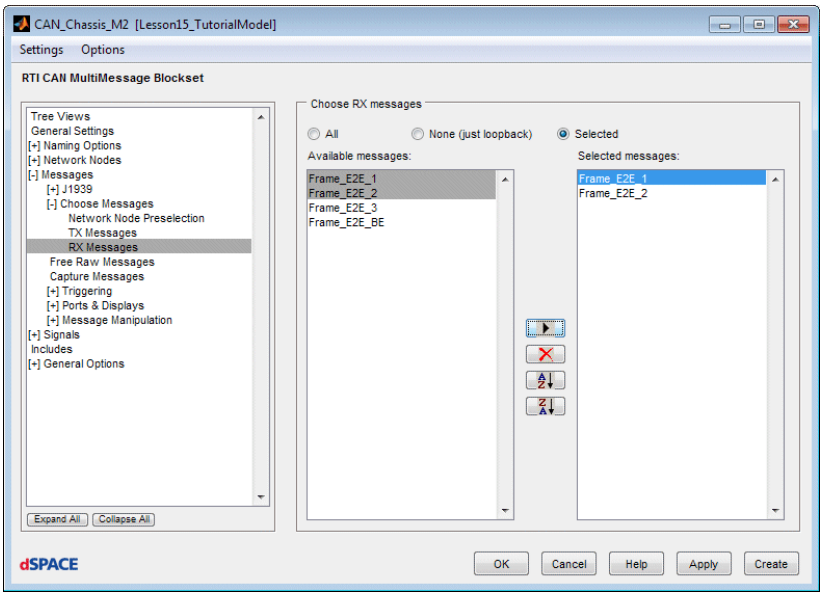
The Available messages list is disabled and no messages are transmitted by this MainBlock.

4 In the dialog tree, select Messages – Choose Messages – RX Messages.

5 In the Available messages list, select the Frame_E2E_1 and Frame_E2E_2 messages.

6 Click .

The messages are added to the Selected messages list. They will be received by the CAN_Chassis_M2 MainBlock.



7 In the dialog tree, select Messages – Ports & Displays – Raw Data – RX Raw Data Display.

8 Select All.

The raw data of all bytes of all the RX messages will be displayed in the message layouts in ControlDesk.

9 In the dialog tree, select Messages – Ports & Displays – Errors – RX Error Display.

10 Select All.

Error display is enabled for all the RX messages of this MainBlock, which means that message errors for these messages can be displayed in ControlDesk.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

11 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

12 Select the Enable E2E Protection checkbox.

13 Click .

14 In the File dialog, select the E2E_checksum.h file and click Open.

15 If the Defined cases identifier list is not empty, select all the entries and click to clear the list.

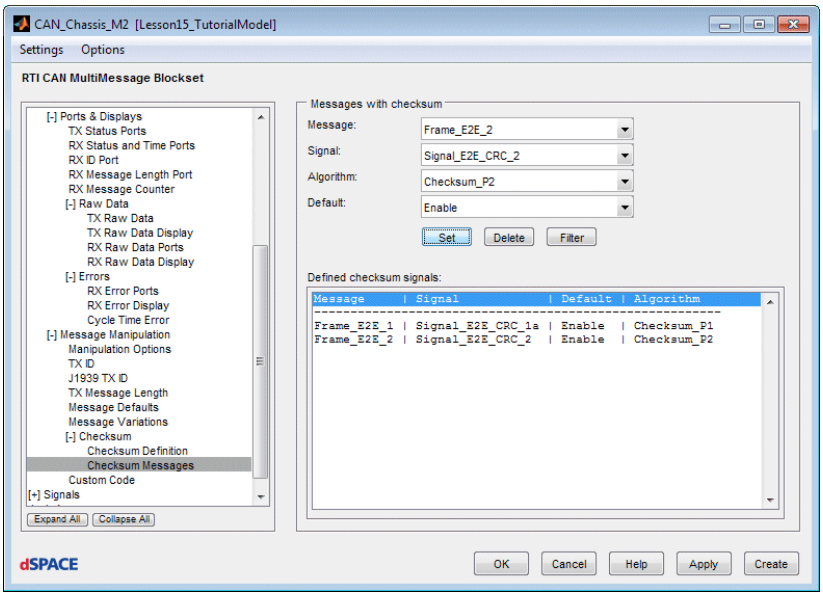
16 Define two new case identifiers named **`Checksum_P1`** and **`Checksum_P2`**.

> **Tip**
>
> Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

17 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Messages.

18 Specify checksum algorithm assignments according to the following settings:

Message	Signal	Algorithm	Default
Frame_E2E_1	Signal_E2E_CRC_1a	Checksum_P1	Enable
Frame_E2E_2	Signal_E2E_CRC_2	Checksum_P2	Enable



19 Click Create.

The CAN_Chassis_M2 MainBlock is created.

20 Click OK to close the CAN_Chassis_M2 [Lesson15_TutorialModel] dialog.

21 Save your model.

Result

You created a model and selected the `E2E_checksum.h` file as the checksum header file to be used. The `E2E_checksum.h` file contains two checksum algorithms using E2E protection data provided by the `CAN_E2E_Demo.arxml` database. You applied the checksum algorithms to TX and RX messages, and activated the display of raw data for TX and RX messages in ControlDesk for a better understanding of checksum behavior. You also enabled error display for the RX messages in ControlDesk.

What's next

In the next step, you can observe the behavior of the E2E checksum algorithms in ControlDesk.

Related topics

Basics

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters..... 399
Overview of Lesson 15..... 400

Step 2: How to Check the Behavior of E2E Checksums in ControlDesk

Objective

After you specified E2E protection checksum settings for your model, you can check the checksum behavior in ControlDesk.

In this step, you will observe the behavior of counter signals and checksums based on E2E protection data.

Method

To check the behavior of counter signals and checksums based on E2E protection parameters

1. Build an application for your model.

   During code generation, the RTI CAN MultiMessage Blockset generates data structures containing the E2E protection data from the database. This data is located in the additional header files `RTICANMM_MAIN_Chassis_M1_E2EP.h` and `RTICANMM_MAIN_Chassis_M2_E2EP.h`, which are included into the S-function of the CAN_Chassis_M1 MainBlock or CAN_Chassis_M2 MainBlock, respectively. You saw a preview of one of these files in the previous step. The header files contain the E2E protection description data defined in the database and the functions to call this data in checksum algorithms.

   The E2E protection header files are also saved to the destination folder.

2. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.

3. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.

4. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.

5 From the platform's context menu in the **Project** controlbar, select **Add Variable Description**.

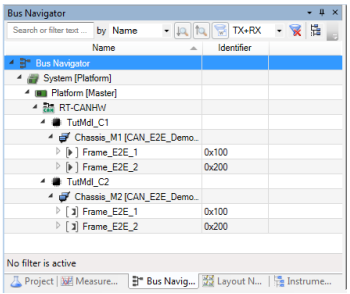
ControlDesk opens the **Select Variable Description** dialog.

6 In the **Select Variable Description** dialog, click **Import from file** and select the `Lesson15_TutorialModel.sdf` file from your `RTICANMMTutorial_Lesson15` folder.

7 In the **Select Variable Description** dialog, click **OK**.

ControlDesk closes the **Select Variable Description** dialog and adds the variable description (SDF file) to the platform.

8 Open the **Bus Navigator**.



9 Open the following layouts:

CAN Bus	Message	Layout
Chassis_M1 [CAN_E2E_Demo.arxml]	Frame_E2E_1	TX
Chassis_M2 [CAN_E2E_Demo.arxml]	Frame_E2E_1	RX
Chassis_M1 [CAN_E2E_Demo.arxml]	Frame_E2E_2	TX
Chassis_M2 [CAN_E2E_Demo.arxml]	Frame_E2E_2	RX

10 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.

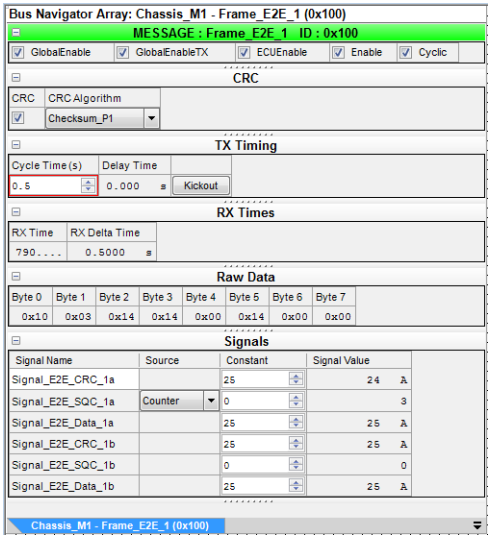
Because you disabled cyclic message transmission by default for the **Frame_E2E_1** and **Frame_E2E_2** messages, no message is transmitted.

11 In the TX layout of **Frame_E2E_1**, select the **Cyclic** checkbox.

The message is transmitted via cyclic transmission, the RX time is running.

12 In the Cycle time (s) field, type **`0.500`** and press **`Enter`**.

This improves the visibility of value changes.



Frame_E2E_1 is a message with eight bytes and six signals. Because you specified Signal_E2E_SQC_1a as a counter signal, its value changes with every message transmission. According to the counter settings specified in your model, the Signal_E2E_SQC_1a value changes constantly from 0 to 15. The white color of the Signal_E2E_CRC_1a signal indicates that a checksum algorithm was specified for it.

Checksum_P1 was specified as the default checksum algorithm for Frame_E2E_1, so it is selected from the CRC Algorithm list. As defined in the Checksum_P1 algorithm, the value of the calculated checksum (represented by Byte 0) depends on the counter value (represented by Byte 1) and the values of Byte 2 and Byte 3. The checksum value with E2E protection is calculated by performing a bitwise XOR operation on the pairs (checksum, upper/lower 4 bits of the relevant DataID). Whether to use the upper or lower 4 bits for the logical operation depends on whether the current counter value is even or odd. See the following code lines from the `E2E_checksum.h` file:

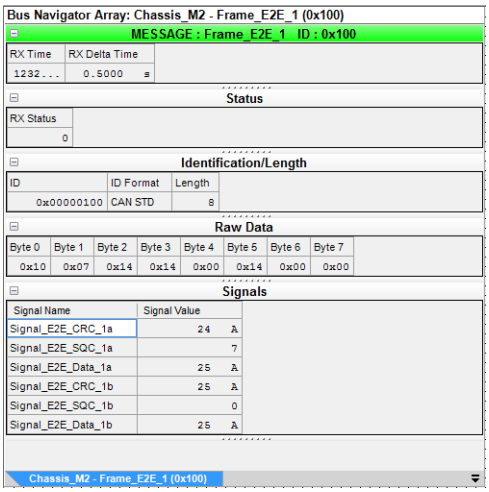
```
// Do simple checksum calculation - XORs 3rd and 4th byte
checksum = Msg->RAW_DATA[2] ^ Msg->RAW_DATA[3];

// XOR data ID on previously calculated checksum
// If counter is even -> take lower 4bits of DataID
if ((counter % 2) == 0)
{
    checksum ^= (DataID & 0x0F);
}
// If counter is odd -> take upper 4bits of DataID
else
{
    checksum ^= (DataID & 0xF0);
}
```

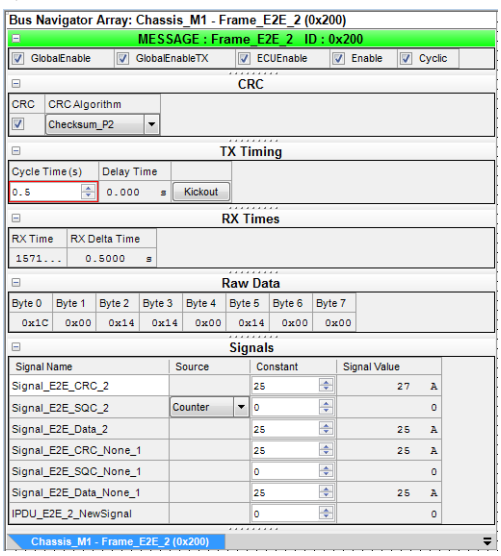
The first element in the DataID list found for the E2E case for the Frame_E2E_1 message is the relevant DataID. For comparison, see the `DataIDs` array generated in the `RTICANMM_MAIN_Chassis_M1_E2EP.h` file (see the following illustration). For the Frame_E2E_1 message, the first DataID list is relevant.

```
static UInt8 DataIDs[NumOfE2ECases][MaxSizeOfDataIDs] =
{
    {31, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {243, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {28, 8, 73, 140, 234, 86, 71, 199, 14, 46, 36, 103, 158, 122, 129, 242},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {253, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

The RX layout of Frame_E2E_1 displays the checksum value calculated for the received message.



13 In the TX layout of Frame_E2E_2, select the Cyclic checkbox and change the cycle time to **`0.500`**.



Because Checksum_P2 was specified as the default checksum algorithm for the Frame_E2E_2 message, it is selected from the CRC Algorithm list. As specified in the Checksum_P2 algorithm, the CRC byte (represented by Byte 0) consecutively displays the E2E checksum values calculated by the logical operations (checksum XOR DataID). The current counter value selects the relevant element in the DataIDs list, i.e., the counter determines which of the 16 DataIDs is used for checksum calculation.

For comparison, see the DataIDs array generated in the `RTICANMM_MAIN_Chassis_M1_E2EP.h` file. The following illustration shows the DataIDs array. For the Frame_E2E_2 message, 16 DataIDs are found in the database for the relevant E2E case.

```
static UInt8 DataIDs[NumOfE2ECases][MaxSizeOfDataIDs] =
{
    {31, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {243, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {28, 8, 73, 140, 234, 86, 71, 199, 14, 46, 36, 103, 158, 122, 129, 242},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {253, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

14 From the **CRC Algorithm** list, select **Checksum_P1**.

The checksum is now calculated according to the **Checksum_P1** algorithm.

15 Look at the **Byte 0** field to check the different behavior of the checksum. **Byte 0** now alternately displays the results of the (checksum XOR upper 4 bits of first DataID element) operation and the (checksum XOR lower 4 bits of first DataID element) operation.

16 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

17 Save the project.

Result

You checked the E2E protection checksums in ControlDesk. You changed the CRC algorithms during run time and observed the different checksum behaviors.

What's next

In the next step, you can extend your E2E protection header file by adding a further checksum algorithm to it.

Related topics

Basics

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters..... 399
Overview of Lesson 15..... 400

Step 3: How to Add a Checksum Case to the E2E Protection Header File

Objective

Checksum algorithms for messages are implemented via the checksum header file. Every algorithm must have its own C-coded switch-case directive in the header file.

In this step, you will extend the E2E protection header file by adding a further checksum case. In ControlDesk, you will observe the checksum behavior.

Part 1

To add a checksum algorithm based on E2E protection data to the header file

1. In MATLAB, double-click the `E2E_checksum.h` file in your RTICANMMTutorial_Lesson15 working folder.

   The `E2E_checksum.h` header file opens in an editor.

2. In the header file, go to the `switch(crctype)` code block.

3. Below the `case 2` code block, insert a new line.

   You will now add a third checksum case. Since the new case resembles case 1, you will copy the code block of case 1 and then make changes to it. The difference to case 1 is that the checksum will always match the used DataID.

4. Copy the whole `case 1` code block and paste it into the line you added below the code block of `case 2`.

5. Rename the newly added case to `case 3`.

```
E2E_checksum.h*
134
135                         // XOR data ID on previously calculated checksum
136                         checksum ^= (DataID & 0xFF);
137                     }
138                 }
139             }
140     }; break;
141
142     case 3: {
143             // Get all E2E indexes for current message
144             // Return value gives us information whether any E2E case was found
145             // We compare it to 0, and if any E2E case was found we proceed further
146             if (getE2EIndexesForMessage(E2EIndexListForMessage, &numOfE2EForMessage, msgId, msgIdFormat) != 0)
147             {
148             UInt32 currentE2EIndex;
149             // Initialize DataOffset:
150             // DataOffset represents start bit of PDU, since each E2E case is linked to PDU we identify E2E case inside CAN me
151             UInt32 DataOffset;
152                 // Loop through all E2E cases for this message and find the case where DataOffset is equal to 0
153                 for (i = 0; i < numOfE2EForMessage; i++)
154                 {
155                     // Set current E2E case index
156                     currentE2EIndex = E2EIndexListForMessage[i];
157
158                     // Fetch DataOffset value for current E2E case
159                     getE2EDataOffset(&DataOffset, currentE2EIndex);
160
161                     // We are searching for E2E case in current message that is applied on PDU with DataOffset == 0
162                     // Compare DataOffset to 0, if it is equal to 0 then perform E2E calculation
163                     if (DataOffset == 0)
164                     {
```

6. In the code block you added, go to `// Get DataID for current E2E case from nult position`.

7. Delete this and the following code lines up to and including

```
else
{
   checksum ^= (DataID & 0xF0);
};
```

8. Instead, add the following lines:

```
// Get DataID for current E2E case from position counter is
pointing to
getE2EDataIDs(&DataID, currentE2EIndex, counter);
// simple change for checksum
checksum = DataID & 0xFF;
```

The code block for case 3 now looks as follows:

```
E2E_checksum.h*

141
142     case 3: {
143             // Get all E2E indexes for current message
144             // Return value gives us information whether any E2E case was found
145             // We compare it to 0, and if any E2E case was found we proceed further
146             if (getE2EIndexesForMessage(E2EIndexListForMessage, &numOfE2EForMessage, msgId, msgIdFormat) != 0)
147             {
148             UInt32 currentE2EIndex;
149             // Initialize DataOffset:
150             // DataOffset represents start bit of PDU, since each E2E case is linked to PDU we identify E2E case inside CAN me
151             UInt32 DataOffset;
152                 // Loop through all E2E cases for this message and find the case where DataOffset is equal to 0
153                 for (i = 0; i < numOfE2EForMessage; i++)
154                 {
155                     // Set current E2E case index
156                     currentE2EIndex = E2EIndexListForMessage[i];
157
158                     // Fetch DataOffset value for current E2E case
159                     getE2EDataOffset(&DataOffset, currentE2EIndex);
160
161                     // We are searching for E2E case in current message that is applied on PDU with DataOffset == 0
162                     // Compare DataOffset to 0, if it is equal to 0 then perform E2E calculation
163                     if (DataOffset == 0)
164                     {
165                         // Get E2E counter position from E2E header file
166                         getE2ECounterPosition(&counterPosition, currentE2EIndex);
167
168                         // Convert bit position to byte position (since counters signals in this database take full byte, no o
169                         counterPosition = counterPosition >> 3;
170
171                         // Get current counter value
172                         counter = Msg->RAW_DATA[counterPosition] & 0x0F;
173
174                         // Get DataID for current E2E case from position counter is pointing to
175                         getE2EDataIDs(&DataID, currentE2EIndex, counter);
176
177                         // simple change for checksum
178                         checksum = DataID & 0xFF;
179                     }
180                 }
181             }
182
183         }; break;
184
185     default: {
186         return(1);
```

Checksum calculation applied to checksum case 3 works as follows: For the current E2E case, the DataIDs that are found for the message in the database are fetched consecutively for each counter position. For each counter position, the checksum value is the corresponding DataID value.

Unlike the checksum algorithm applied to case 2, the checksum values according to checksum case 3 do not depend on the values of Byte 2 and Byte 3. Another difference lies in the number of function calls used to get the DataIDs. In case 2, all DataIDs are fetched in one function call. In case 3, each DataID is fetched by a separate function call.

9 Save and close the `E2E_checksum.h` file.

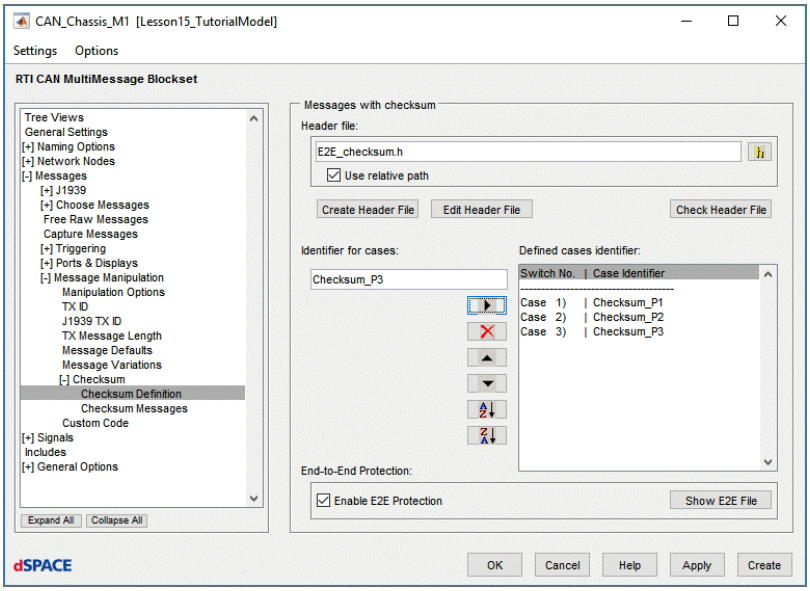
10 In your model, double-click the CAN_Chassis_M1 block to open it.

11 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

12 In the Identifier for cases edit field, type **`Checksum_P3`**.

13 Click .

Checksum_P3 is added to the Defined cases identifier list. The new case identifier must be located at the end of the list. If it is not, you can sort the identifiers via buttons.



The checksum case 3 you added to the `E2E_checksum.h` header file is related to the Checksum_P3 case identifier.

14 Click Create.

The CAN_Chassis_M1 MainBlock is created.

15 Click OK to close the CAN_Chassis_M1 [Lesson15_TutorialModel] dialog.

16 Open the CAN_Chassis_M2 block.

17 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

18 Define a new case identifier named **`Checksum_P3`**. Make sure that the new case identifier is located at the end of the list.

19 Click Create.

The CAN_Chassis_M2 MainBlock is created.

20 Click OK to close the CAN_Chassis_M2 [Lesson15_TutorialModel] dialog.

21 Save your model.

Interim result

You defined a third checksum case in the `E2E_checksum.h` header file and specified a further checksum case identifier for it. You can now check the behavior of the new checksum algorithm in ControlDesk. Continue with part 2.

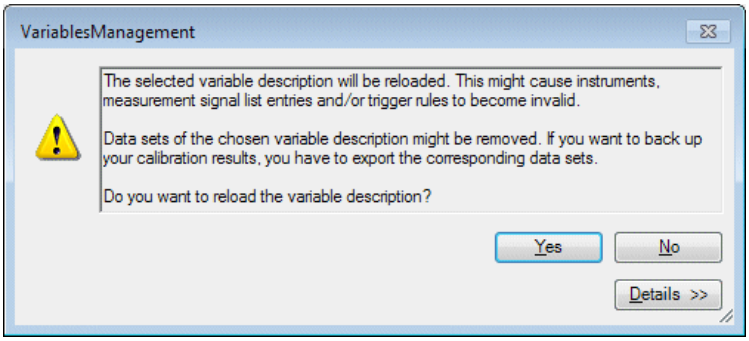
Part 2

To check the result in ControlDesk

1 Build an application for your model.

2 From the context menu of the `Lesson15_TutorialModel.sdf` file in the active ControlDesk experiment, select Reload Variable Description.

The following warning is displayed, informing you that reloading a variable description might cause instruments and other items to become invalid. Click Yes.



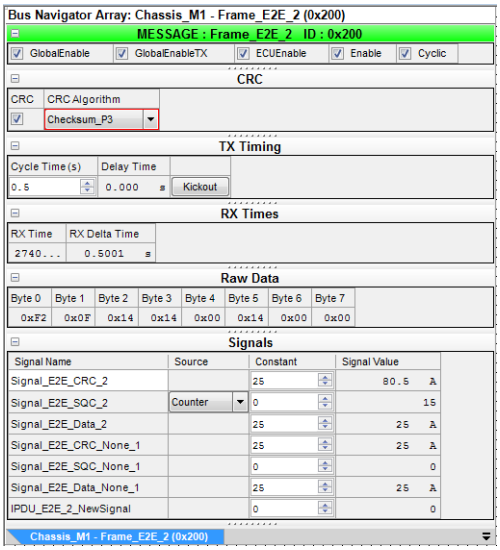
3 In the Bus Navigator tree, right-click Frame_E2E_2 below the Chassis_M1 [CAN_E2E_Demo.arxml] node and generate a TX layout.

Generating a new TX layout is necessary to have the new checksum algorithm available for selection.

4 On the Home ribbon, click Status Control – Go Online to start online calibration.

5 In the new TX layout of Frame_E2E_2, select the Cyclic checkbox and specify a Cycle Time of `0.500`.

6 From the CRC Algorithm list, select the Checksum_P3 checksum algorithm.



As specified in the Checksum_P3 algorithm, the checksum matches the used DataID. According to the current counter value, Byte 0 successively displays the 16 DataIDs that are defined in the database for the message as the checksum value.

7 To check the behavior of the Checksum_P3 algorithm for the Frame_E2E_1 message, generate a new TX layout of Frame_E2E_1 and select Checksum_P3 as the CRC algorithm. Because for the Frame_E2E_1 message only one DataID is defined in the database (see generated `DataIDs` array in the `RTICANMM_MAIN_Chassis_M1_E2EP.h` file) and this DataID is located at the first position in the list, the DataID is displayed in hexadecimal notation as the checksum value each time the current counter value is 0. For counter values ≠ 0, the checksum value is 0.

8 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

9 Save the project.

Result

You added a checksum algorithm to the `E2E_checksum.h` file and checked the effects of your specifications in ControlDesk.

What's next

With this step, you finished working with checksums based on E2E protection data. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters..... 399
Overview of Lesson 15..... 400

Result of Lesson 15

Result

In this lesson, you learned the basic principles of working with checksum algorithms based on E2E communication protection parameters provided by the database file:

- You must enable the use of E2E protection parameters. If enabled, an additional header file (named `RTICANMM_MAIN_<CAN name>_E2EP.h`) is generated and included into the S-function of the RTICANMM MainBlock. The E2E protection header file is also saved to the destination folder.
- The generated header file contains the E2E protection description data defined in the database and the functions to call this data in checksum algorithms. You can use the E2E protection data in checksum algorithm implementations.
- You can get a preview of the E2E protection header file via the **Show E2E File** button.
- E2E protection checksum algorithms are implemented in the same checksum header file as the checksum algorithms without E2E protection data.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson15_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Checksum Definition Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Checksum Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Error Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

Now you can continue with the next lesson to work with end-to-end communication protection according to AUTOSAR E2E profiles in checksum algorithms.

If you do not need to work with checksums based on E2E communication protection parameters according to AUTOSAR E2E profiles, you can continue with another lesson or the summary of this tutorial. Refer to Overview of Lessons on page 35 or Summary on page 457.

Related topics

Basics

Lesson 15 (Advanced): Using Checksums Based on E2E Protection Parameters..... 399
Overview of Lesson 15..... 400

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles

Where to go from here

Information in this section

Overview of Lesson 16 430
In this lesson, you will learn advanced principles of using end-to-end communication protection according to AUTOSAR end-to-end protection profiles in user-defined checksum calculations.

Step 1: How to Work with Checksum Algorithms Based on E2E Protection According to AUTOSAR E2E Profiles 434
In this step, you will specify a database file for your model that contains defined E2E protection parameters based on AUTOSAR requirements. You will select the checksum header file to be used, define the checksum case identifiers, enable the use of E2E protection parameters, and assign the algorithms to signals.

Step 2: How to Check the Behavior of Checksums Based on E2E Protection According to AUTOSAR E2E Profiles in ControlDesk 446
In this step, you will create an experiment in ControlDesk, generate TX and RX layouts for messages whose E2E protection behavior you want to check, and observe the behavior of the checksums. You will look at E2E profile 05 and profile 11 (variants A and C) as examples.

Step 3: How to Use Debug Messages in the Checksum Header File 451
In this step, you will enable the output of E2E debug messages in the checksum header file. The debug messages provide additional information relevant for E2E protection. You will then view the effects in the Messages controlbar in ControlDesk.

Result of Lesson 16 454
In this lesson, you learned the advanced principles of using end-to-end communication protection according to AUTOSAR in user-defined checksum calculations.

Overview of Lesson 16

Introduction

The RTI CAN MultiMessage Blockset supports E2E communication protection according to AUTOSAR E2E profiles. It accesses appropriate E2E protection parameters from AUTOSAR communication matrices at run time. The parameters can be used to implement checksum algorithms based on E2E protection parameters in the real-time application.

In AUTOSAR, E2E communication protection is related to signal groups and IPDUs. In RTICANMM, the E2E protection data is mapped to the message level for both cases.

Tip

If you do not work with checksums based on E2E protection data according to AUTOSAR E2E profiles, you can skip this lesson.

The `E2E_checksum_advanced.h` file, which is used as the checksum header file in this tutorial lesson, contains E2E protection algorithms for all AUTOSAR E2E profiles supported by the RTI CAN MultiMessage Blockset.

Using E2E protection parameters in user-defined checksum algorithms

You must enable the use of E2E protection parameters. If enabled, the E2E protection data defined in the database file is imported into an additionally generated E2E protection header file, which is automatically included in the S-function of the RTICANMM MainBlock. The imported E2E protection data is available for checksum algorithm implementation. The header file also contains the functions for calling the E2E protection data in checksum algorithms.

Basics on AUTOSAR E2E profile 05

According to AUTOSAR, when you use E2E profile 05, some control fields must be implicitly or explicitly transmitted at run time with the protected data:

Control Field	Length	Description
Counter	8 bits	Represents numbers from 0 to 255 (0xFF), incremented by 1 on each subsequent transmission. Sent explicitly. The counter value is included in CRC calculation.
CRC	16 bits	Uses a polynomial algorithm in the normal form 0x1021 provided by the AUTOSAR CRC library. Sent explicitly.

Control Field	Length	Description
Data ID	16 bits	System-wide unique number. Transmitted implicitly by adding the Data ID after the user data in CRC calculation.

E2E profile 05 requires a header of three bytes with a fixed layout: The first two bytes are the checksum (little endian order), the third byte is the counter.

The following illustration shows the header of the E2E profile 05 with its header fields:

Byte order	0								1								2							
Transmission order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Bit order	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	23	22	21	20	19	18	17	16
	E2E CRC																E2E Counter							

The length of the data to be protected must be a multiple of 1 byte. There are no other constraints for the user data layout.

AUTOSAR E2E profile 05 is available only for AUTOSAR Release 4.2.1 and later.

Basics on AUTOSAR E2E profile 11

E2E profile 11 has the variants 11A and 11C. The variants are specific configurations of E2E profile 11.

For E2E protection according to AUTOSAR E2E profile 11, the following control fields must be explicitly or implicitly transmitted at run time with the protected data:

Control Field	Length	Description
Counter	4 bits	Represents numbers from 0 to 14 (0xE), incremented by 1 on each subsequent transmission. Sent explicitly. The counter value is included in CRC calculation.
CRC	8 bits	Uses CRC-8-SAE J1850, provided by the AUTOSAR CRC library. Sent explicitly.
Data ID	16 or 12 bits	System-wide unique number. Either sent implicitly (16 bits) or partly explicitly (12 bits; 4 bits explicitly and 8 bits implicitly), by adding the Data ID after the user data in CRC calculation.

E2E profile 11 requires a header of two bytes with a fixed layout: The first byte is the checksum, the second byte is the counter, in case of profile variant 11C combined with Data ID Nibble. For profile variant 11A, Data ID Nibble is not used and you can add user data instead.

The following illustration shows the header of variant 11A of E2E profile 11 with its header fields:

Byte order	0								1							
Transmission order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit order	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
	E2E CRC												Counter			

The header of profile 11C looks like this:

Byte order	0								1							
Transmission order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit order	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
	E2E CRC								Data ID Nibble				Counter			

The length of the data to be protected must be a multiple of 1 byte. There are no othe constraints for the user data layout.

AUTOSAR E2E profile 11 is available only for AUTOSAR Release 4.3.0 and later.

E2E protection according to AUTOSAR E2E profiles in this lesson

In this lesson, you will work with end-to-end protection according to two AUTOSAR E2E profiles:

- The first example uses AUTOSAR E2E profile 05.

  The Frame_E2E_Profile05 message will be protected by AUTOSAR E2E profile 05. The following illustration shows the layout of the message:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CRC		SQC	DATA				
0		16	24				

  The message contains the following signals:

Signal	Length	Description
Signal_E2E_CRC_Profile05	2 bytes	Checksum signal
Signal_E2E_SQC_Profile05	1 byte	Counter signal
Signal_E2E_Data_Profile05_1 ... Signal_E2E_Data_Profile05_5	5 bytes	User data to be protected

  For time efficiency reasons, CRC calculation via tables is used in the tutorial.

- The second example uses E2E profile 11.

  In this example, multiple E2E cases are modeled in one message. The Frame_E2E_Profile11_A_C message will be protected by AUTOSAR E2E profile 11, using both variants 11A and 11C. The following illustration shows the database layout of the message:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CRC_A	SQC_A	DATA_A		CRC_C	SQC_C	DATA_C	
0	8	16		32	40	48	

  The message contains the following signals:

Signal	Length	Description
Signal_E2E_CRC_Profile11_A	1 byte	Checksum signal
Signal_E2E_SQC_Profile11_A	4 bits	Counter signal
Signal_E2E_Data_Profile11_A	2 bytes	User data to be protected
Signal_E2E_CRC_Profile11_C	1 byte	Checksum signal
Signal_E2E_SQC_Profile11_C	4 bits	Counter signal
Signal_E2E_Data_Profile11_C	2 bytes	User data to be protected

In this example, the checksum is also calculated using tables for reasons of time efficiency.

What you will learn

In this lesson, you will learn the advanced principles of using end-to-end communication protection parameters according to AUTOSAR in user-defined checksum calculations.

- You will specify a database file containing defined E2E protection parameters with advanced characteristics for your model.
- You will use the E2E_checksum_advanced.h file as a template. The file contains E2E protection functions based on the AUTOSAR requirements and covers all E2E profiles supported by the RTI CAN MultiMessage Blockset.
- You will observe the behavior of checksums in ControlDesk. You will look at profile 05 and profile 11 (variants A and C) as examples.
- You will enable additional debug messages in the checksum definition file. As a consequence, further information relevant for E2E protection will be displayed in ControlDesk.

Starting point

The starting point of this lesson is the **TutorialModel** model you created in lesson 6.

Prepared demo

If you did not create your own model in lesson 6, you can use the prepared demo **Lesson6_TutorialModel.slx** as the starting point. Refer to Working with the Prepared Demo Results on page 32.

Before you begin

Before you begin this lesson, the following preconditions must be met:

- You have to know how to build an application for your model and how to check the application loaded on the real-time hardware (refer to lesson 1, Step 3: How to Build an Application and Download It to Your Real-Time Hardware on page 56).
- The **RTICANMMTutorialProject** project must be open in ControlDesk and you have to know how to create an experiment. Refer to How to Create an Experiment in ControlDesk on page 483.

Steps

This lesson contains the following steps:

- Step 1: How to Work with Checksum Algorithms Based on E2E Protection According to AUTOSAR E2E Profiles on page 434
- Step 2: How to Check the Behavior of Checksums Based on E2E Protection According to AUTOSAR E2E Profiles in ControlDesk on page 446
- Step 3: How to Use Debug Messages in the Checksum Header File on page 451

Result and further information

To check the correctness of the executed steps and for more information, refer to Result of Lesson 16 on page 454.

Duration

Working through this lesson will take you about 60 minutes.

What's next

You can now start to specify checksum algorithms that are based on E2E protection parameters according to AUTOSAR E2E profiles. Refer to Step 1: How to Work with Checksum Algorithms Based on E2E Protection Parameters on page 401.

Related topics

Basics

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles..... 429

Step 1: How to Work with Checksum Algorithms Based on E2E Protection According to AUTOSAR E2E Profiles

Objective

The RTI CAN MultiMessage Blockset supports end-to-end communication protection (E2E protection) according to AUTOSAR end-to-end protection profiles.

In this step, you will specify a database file for your model that contains defined E2E protection parameters based on AUTOSAR requirements. You will select the checksum header file to be used, define the checksum case identifiers, enable the use of E2E protection parameters, and assign the algorithms to signals.

Part 1

To create the model and specify a new database containing E2E protection data for all E2E profiles supported by the RTI CAN MultiMessage Blockset

1. In the working folder, create a new folder named **`RTICANMMTutorial_Lesson16`**.
2. In MATLAB, change the Current Directory to the new RTICANMMTutorial_Lesson16 folder.
3. In the MATLAB Command Window, type **`rticanmm_tutorial`**.

   The rticanmm_tutorial_lib library opens.
4. Double-click Lesson 6 Results.

5 In the Get Tutorial dialog, click Only open model.

The Lesson6_TutorialModel demo model opens

6 On the Simulation ribbon, click Save – Save as.

7 In the Save As dialog, name the model **`Lesson16_TutorialModel`** and save it to your RTICANMMTutorial_Lesson16 working folder.

Tip

If you created your own model in lesson 6, you can alternatively copy the TutorialModel model from the RTICANMMTutorial folder to the RTICANMMTutorial_Lesson16 folder and rename it.
It is recommended to work with a new working folder and rename the model in this lesson. The model you created in lesson 6 is also used as the starting point of lesson 7, where it is required in unchanged form.

8 In the rticanmm_tutorial_lib library, double-click CAN_E2E_Demo_advanced.arxml.

The CAN_E2E_Demo_advanced.arxml file is copied to your RTICANMMTutorial folder.

9 In the rticanmm_tutorial_lib library, double-click E2E_checksum_advanced.h.

The E2E_checksum_advanced.h file is copied to your RTICANMMTutorial folder.

10 Close the rticanmm_tutorial_lib library.

11 If necessary, update the real-time hardware specified in the RTICANMM ControllerSetup blocks and the system target file according to your requirements. For details, refer to Working with the Prepared Demo Results on page 32.

12 Double-click the RTICANMM GeneralSetup block in the model and select Options - Create S-function for All CAN Blocks.

The S-function is created and you can start working with the model.

13 Click OK to close the dialog.

14 In the model, double-click the CAN_Chassis_M1 block to open it.

A dialog informs you that the database import failed.

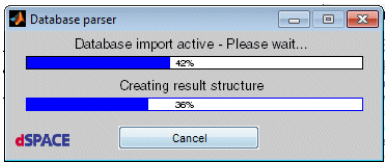
15 Click OK to close the dialog.

16 Select the General Settings page.

17 Click .

18 In the File dialog, select the CAN_E2E_Demo_advanced.arxml file and click Open.

The Database Parser dialog opens and the selected database file is parsed automatically.



After parsing, the dialog closes.

19 In the CAN_Chassis_M1 [Lesson16_TutorialModel] dialog, click OK to close the dialog.

20 Open the CAN_Chassis_M2 MainBlock.

A dialog informs you that the database import failed.

21 Click OK to close the dialog.

22 Select the General Settings page.

23 Specify the CAN_E2E_Demo_advanced.arxml file as the new database.

24 Click OK to close the CAN_Chassis_M2 [Lesson16_TutorialModel] dialog.

25 Save the model.

Interim result

You created a model on the basis of the result of lesson 6, copied the required files to the working folder, and specified the CAN_E2E_Demo_advanced.arxml file as the new database for the model. CAN_E2E_Demo_advanced.arxml contains E2E protection data for all E2E profiles supported by the RTI CAN MultiMessage Blockset. You can now configure the CAN communication of the CAN_Chassis_M1 MainBlock, which is used only to transmit messages, specify counter signals to check correct message transmission, and display the raw data of TX messages. Continue with part 2.

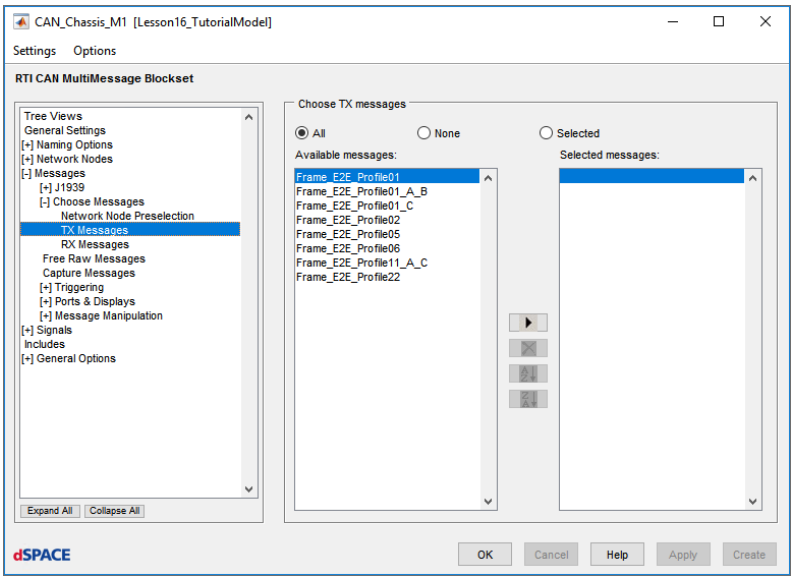
Part 2

To configure CAN communication of the CAN_Chassis_M1 MainBlock, specify counter signals, and display TX raw data

1 Open the CAN_Chassis_M1 MainBlock.

2 In the dialog tree, select Messages – Choose Messages – TX Messages.

3 Select **All**.



All messages are specified as TX messages.

4 In the dialog tree, select the **Messages – Choose Messages – RX Messages** page.

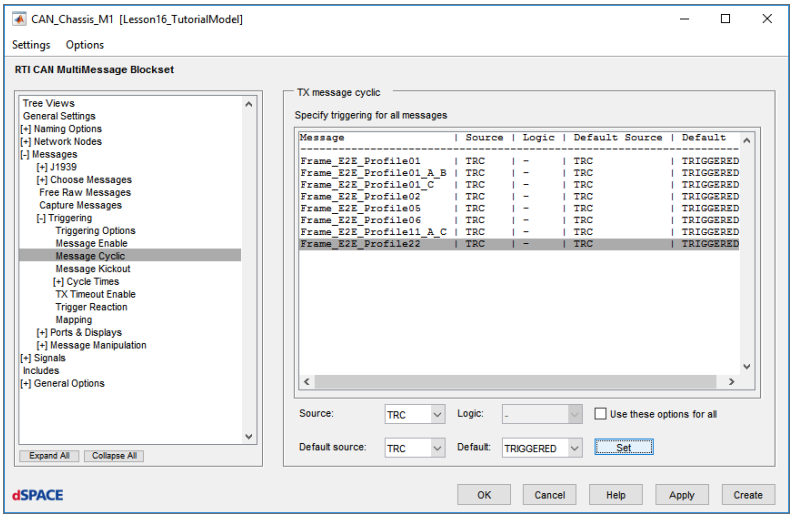
5 Select **None (just loopback)**.

This MainBlock is used only to transmit messages. You can therefore disable the reception of messages. However, all the messages transmitted by a MainBlock are always received by the same MainBlock as loopback messages.

6 In the dialog tree, select **Messages – Triggering – Message Cyclic**.

On the Message Cyclic page, you can enable or disable cyclic transmission for all or some of the messages displayed in the **Specify triggering for all messages** list.

7 From the **Default** drop-down list, select **Triggered** for all messages.



You disabled cyclic message transmission for all messages by default. During run time, cyclic transmission can be enabled via ControlDesk.

8 In the dialog tree, select **Signals – TX – Signal Manipulation – Counter**.

This page lets you specify counter signals. The values of counter signals change with each message transmission. TX messages are counted when they are transmitted. RX messages are counted when they are received. For each message, you can specify one or more counter signals that provide the number of message transmissions/receptions. You can specify the counter start value, the increment, and the counter stop value. Each time the counter reaches the stop value, it switches to the counter start value. Each E2E profile has its own requirement for the counter signal range.

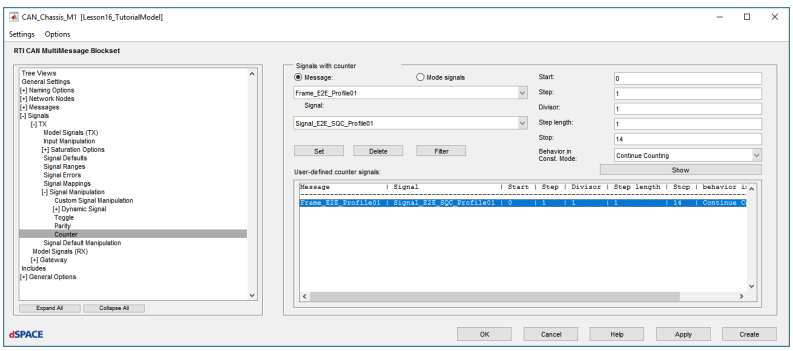
9 Specify the following settings for a counter signal for the **Frame_E2E_Profile01** message:

Message	Signal	Start	Step	Divisor	Step Length	Stop	Behavior in Const. Mode
Frame_E2E_Profile01	Signal_E2E_SQC_Profile01	0	1	1	1	14	Continue Counting

10 Click **Set**.

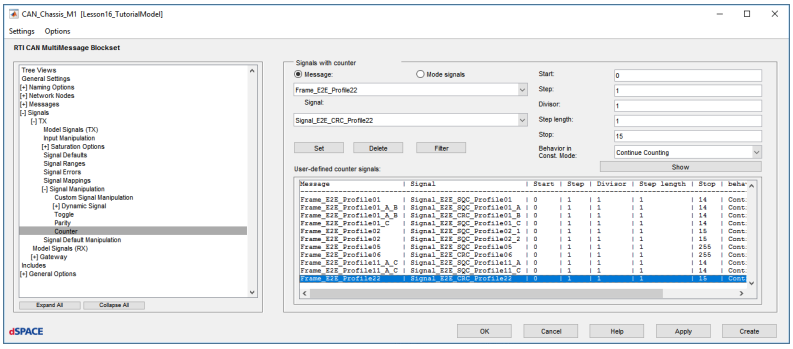
Note

You always have to click **Set** to apply your settings.



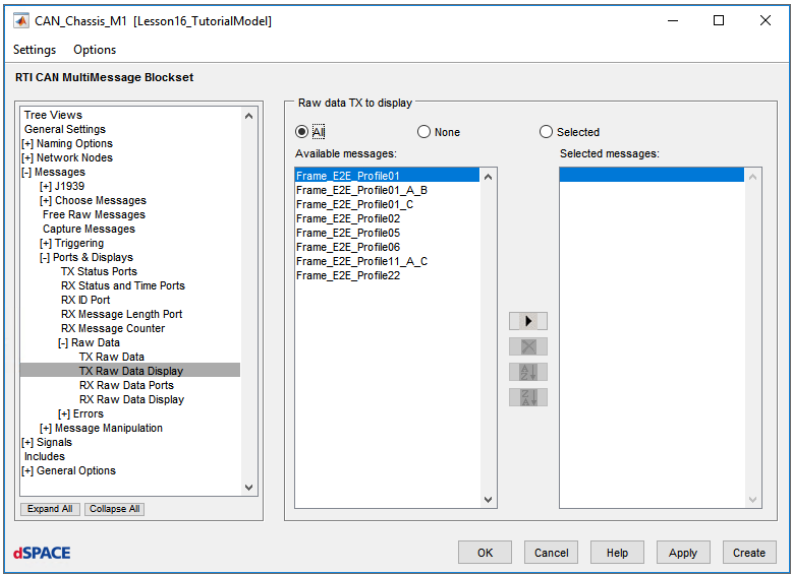
11 Specify more counter signals using the following settings:

Message	Signal	Start	Step	Divisor	Step Length	Stop	Behavior in Const. Mode
Frame_E2E_Profile01_A_B	Signal_E2E_SQC_Profile01_A	0	1	1	1	14	Continue Counting
Frame_E2E_Profile01_A_B	Signal_E2E_SQC_Profile01_B	0	1	1	1	14	Continue Counting
Frame_E2E_Profile01_C	Signal_E2E_SQC_Profile01_C	0	1	1	1	14	Continue Counting
Frame_E2E_Profile02	Signal_E2E_SQC_Profile02_1	0	1	1	1	15	Continue Counting
Frame_E2E_Profile02	Signal_E2E_SQC_Profile02_2	0	1	1	1	15	Continue Counting
Frame_E2E_Profile05	Signal_E2E_SQC_Profile05	0	1	1	1	255	Continue Counting
Frame_E2E_Profile06	Signal_E2E_SQC_Profile06	0	1	1	1	255	Continue Counting
Frame_E2E_Profile11_A_C	Signal_E2E_SQC_Profile11_A	0	1	1	1	14	Continue Counting
Frame_E2E_Profile11_A_C	Signal_E2E_SQC_Profile11_C	0	1	1	1	14	Continue Counting
Frame_E2E_Profile22	Signal_E2E_SQC_Profile22	0	1	1	1	15	Continue Counting



12 In the dialog tree, select Messages – Ports & Displays – Raw Data – TX Raw Data Display.

13 Select All.



The raw data of all bytes of all the TX messages are displayed in the message layouts in ControlDesk.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You selected all messages from the database as TX messages and disabled cyclic message transmission for them by default. You specified a counter signal for each TX message. For some messages, there are multiple counter signals because there are multiple E2E cases for them. For a better understanding of checksum behavior, you activated the display of TX raw data. You can now select the checksum header file template to be used and specify checksum case identifiers. Continue with part 3.

Part 3

To select a header file and specify checksum case identifiers

1. In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

   On the Checksum Definition page, you can define a checksum header file and specify checksum case identifiers.

2. Select the Enable E2E Protection checkbox.

   You enabled the use of E2E protection parameters. Enabling the use of E2E protection parameters generates an additional header file (named `RTICANMM_MAIN_Chassis_M1_E2EP.h`) and includes it in the S-function of the RTICANMM MainBlock. This header file contains the E2E protection description data defined in the database and the functions for calling this data in checksum algorithms.

3. Click h.

4. In the File dialog, select the E2E_checksum_advanced.h file from the working folder and click Open.

   You selected the E2E_checksum_advanced.h file as the checksum header file to be used. The E2E_checksum_advanced.h file is a template containing two checksum cases (switch-case directives) using E2E data provided by the `CAN_E2E_Demo_advanced.arxml` database. The first case directive executes E2E protection without inserting an error. The second case directive executes E2E protection and inserts an error by toggling the value of bit 0 of the calculated checksum.

5. On the Checksum Definition page, select all entries in the Defined cases identifier list and click .

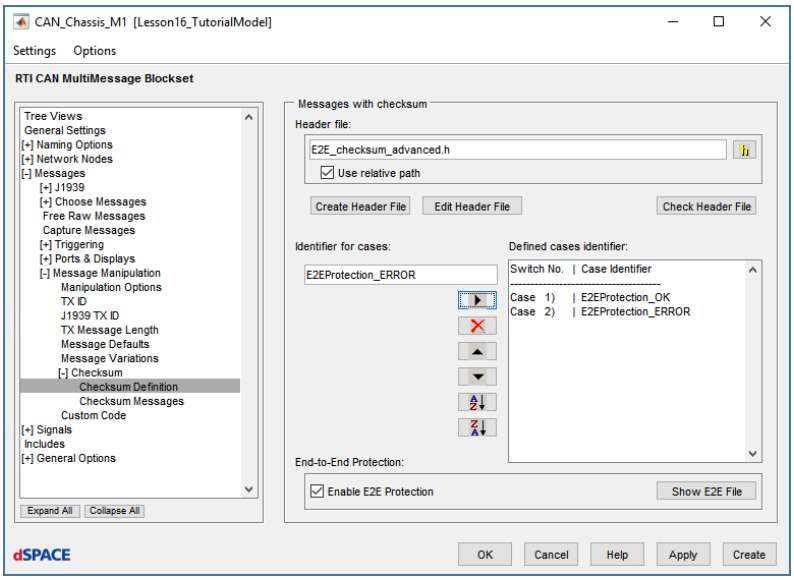
   The Defined cases identifier list is cleared. You can now specify new case identifiers.

6. In the Identifier for cases edit field, type `E2EProtection_OK`.

7. Click .

   E2EProtection_OK is added to the Defined cases identifier list.

8 Define another case identifier named **`E2EProtection_ERROR`**.



The two checksum cases defined in the `E2E_checksum_advanced.h` header file are related to these case identifiers.

Tip

Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

Interim result

You selected the checksum header file to be used and specified checksum case identifiers for the checksum cases defined in the checksum header file. You can now assign the checksum algorithms to messages. Continue with part 4.

Part 4

To assign checksum algorithms to messages

1 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Messages.

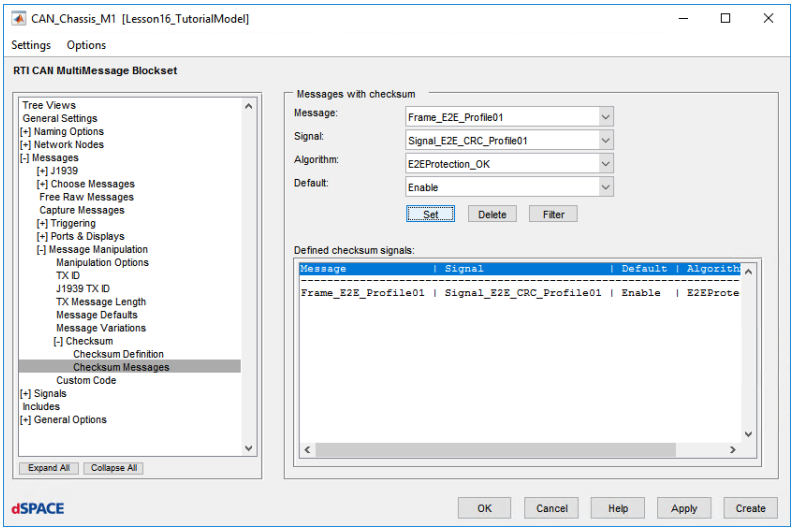
On the Checksum Messages page, you can assign checksum algorithms to messages.

2 Specify the following settings:

Message	Signal	Algorithm	Default
Frame_E2E_Profile01	Signal_E2E_CRC_Profile01	E2EProtection_OK	Enable

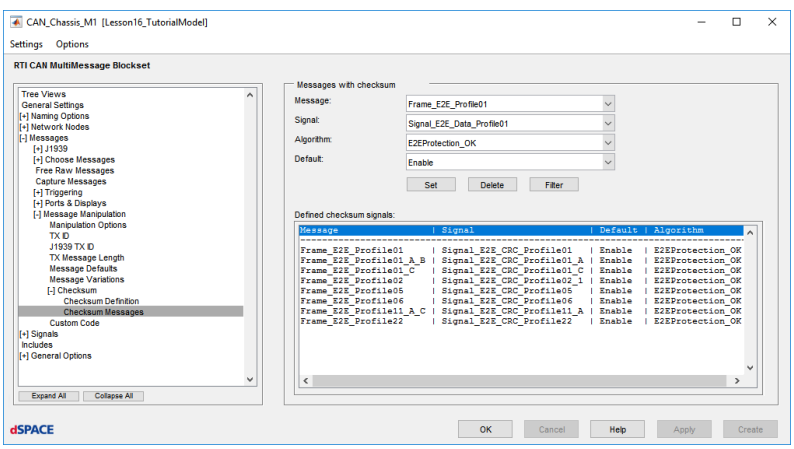
3 Click Set.

E2EProtection_OK is specified as the default checksum algorithm for Frame_E2E_Profile01. You selected Signal_E2E_CRC_ Profile01 as the checksum signal. Checksum calculation for the message is enabled by default.



4 Specify the following settings for the other checksum signals:

Message	Signal	Algorithm	Default
Frame_E2E_Profile01_A_B	Signal_E2E_CRC_Profile01_A	E2EProtection_OK	Enable
Frame_E2E_Profile01_C	Signal_E2E_CRC_Profile01_C	E2EProtection_OK	Enable
Frame_E2E_Profile02	Signal_E2E_CRC_Profile02_1	E2EProtection_OK	Enable
Frame_E2E_Profile05	Signal_E2E_CRC_Profile05	E2EProtection_OK	Enable
Frame_E2E_Profile06	Signal_E2E_CRC_Profile06	E2EProtection_OK	Enable
Frame_E2E_Profile11_A_C	Signal_E2E_CRC_Profile11_A	E2EProtection_OK	Enable
Frame_E2E_Profile22	Signal_E2E_CRC_Profile22	E2EProtection_OK	Enable



Note

Only one signal per message can be defined as the checksum signal. This will not be a problem even if there are multiple E2E cases per message.

5 Click Create.

The CAN_Chassis_M1 MainBlock is created.

6 Click OK to close the CAN_Chassis_M1 [Lesson16_TutorialModel] dialog.

7 Delete the CRASH_System, Triggering to RTICANMM, and Mapping to RTICANMM blocks and their connections from the model.

8 Save the model.

Interim result

You assigned checksum algorithms to the TX messages of the CAN_Chassis_M1 MainBlock and completed checksum configuration for the CAN_Chassis_M1 MainBlock. You can now specify the CAN communication and the checksum settings for the CAN_Chassis_M2 MainBlock. Continue with part 5.

Part 5

To configure the CAN communication of the CAN_Chassis_M2 MainBlock and specify checksum settings for the RX messages

1 Open the CAN_Chassis_M2 MainBlock.

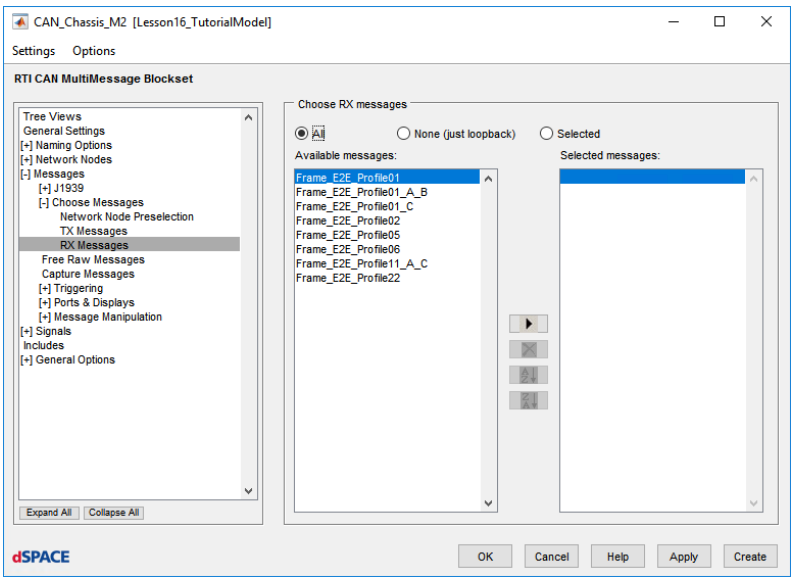
2 In the dialog tree, select Messages – Choose Messages – TX Messages.

3 Select None.

The Available messages list is disabled and no messages are transmitted by this MainBlock.

4 In the dialog tree, select Messages – Choose Messages – RX Messages.

5 Select All.



All messages are specified as RX messages. They will be received by the CAN_Chassis_M2 MainBlock

6 In the dialog tree, select Messages – Ports & Displays – Raw Data – RX Raw Data Display.

7 Select All.

The raw data of all bytes of all the RX messages will be displayed in the message layouts in ControlDesk.

8 In the dialog tree, select Messages – Ports & Displays – Errors – RX Error Display.

9 Select All.

Error display is enabled for all the RX messages of this MainBlock, which means that message errors for these messages can be displayed in ControlDesk.

> **Tip**
>
> Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

10 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Definition.

11 Select the Enable E2E Protection checkbox.

12 Click .

13 In the File dialog, select the E2E_checksum_advanced.h file and click Open.

14 If the Defined cases identifier list is not empty, select all the entries and click to clear the list.

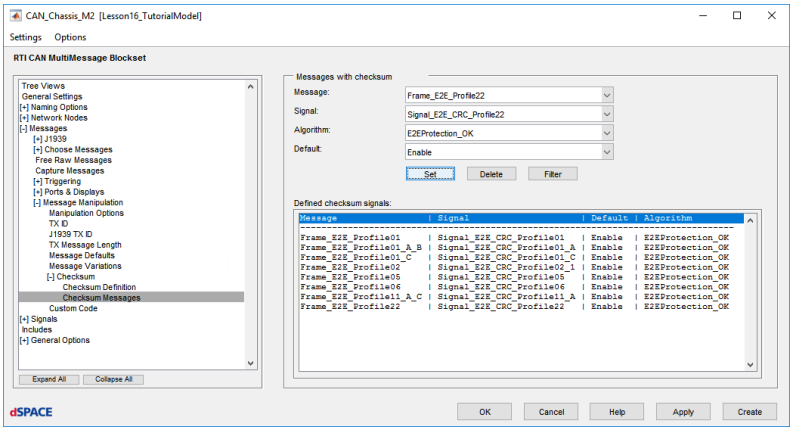
15 Define two new case identifiers named **`E2EProtection_OK`** and **`E2EProtection_ERROR`**.

> **Tip**
>
> Click Apply to apply your settings to the block. Because no files are created, applying the settings is quicker than creating the entire block.

16 In the dialog tree, select Messages – Message Manipulation – Checksum – Checksum Messages.

17 Specify checksum algorithm assignments according to the following settings:

Message	Signal	Algorithm	Default
Frame_E2E_Profile01	Signal_E2E_CRC_Profile01	E2EProtection_OK	Enable
Frame_E2E_Profile01_A_B	Signal_E2E_CRC_Profile01_A	E2EProtection_OK	Enable
Frame_E2E_Profile01_C	Signal_E2E_CRC_Profile01_C	E2EProtection_OK	Enable
Frame_E2E_Profile02	Signal_E2E_CRC_Profile02_1	E2EProtection_OK	Enable
Frame_E2E_Profile05	Signal_E2E_CRC_Profile05	E2EProtection_OK	Enable
Frame_E2E_Profile06	Signal_E2E_CRC_Profile06	E2EProtection_OK	Enable
Frame_E2E_Profile11_A_C	Signal_E2E_CRC_Profile11_A	E2EProtection_OK	Enable
Frame_E2E_Profile22	Signal_E2E_CRC_Profile22	E2EProtection_OK	Enable



Note

Only one signal per message can be defined as the checksum signal. This will not be a problem even if there are multiple E2E cases per message.

18 Click Create.

The CAN_Chassis_M2 MainBlock is created.

19 Click OK to close the CAN_Chassis_M2 [Lesson16_TutorialModel] dialog.

20 Save the model.

Result

You created a model and selected the `E2E_checksum_advanced.h` file as the checksum header file to be used. The `E2E_checksum_advanced.h` file contains two checksum algorithms (switch-case directives) that use E2E protection data provided by the `CAN_E2E_Demo_advanced.arxml` database. You applied the checksum algorithms to TX and RX messages, and activated the display of raw data for TX and RX messages in ControlDesk for a better understanding of checksum behavior. You also enabled error display for the RX messages in ControlDesk.

What's next

In the next step, you can observe the behavior of the E2E checksum algorithms according to AUTOSAR end-to-end protection profiles in ControlDesk.

Related topics

Basics

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles..... 429
Overview of Lesson 16..... 430

Step 2: How to Check the Behavior of Checksums Based on E2E Protection According to AUTOSAR E2E Profiles in ControlDesk

Objective

In ControlDesk, you can observe the behavior of checksums based on E2E protection according to AUTOSAR E2E profiles.

In this step, you will create an experiment in ControlDesk, generate TX and RX layouts for messages whose E2E protection behavior you want to check, and observe the behavior of the checksums. You will look at E2E profile 05 and profile 11 (variants A and C) as examples.

Part 1

To check the behavior of checksums based on AUTOSAR E2E profile 05

1. Build an application for your model.

   During code generation, the RTI CAN MultiMessage Blockset generates data structures containing the E2E protection data from the database. This data is located in the additional header files `RTICANMM_MAIN_Chassis_M1_E2EP.h` and `RTICANMM_MAIN_Chassis_M2_E2EP.h`, which are included in the S-function of the CAN_Chassis_M1 MainBlock or CAN_Chassis_M2 MainBlock, respectively. The header files contain the E2E protection description data defined in the database and the functions for calling this data in checksum algorithms.

   The E2E protection header files are also saved to the destination folder.

2. Create an experiment in the RTICANMMTutorialProject project in ControlDesk. For instructions, refer to method 2 in How to Create an Experiment in ControlDesk on page 483.

3. From the context menu of the registered real-time hardware in the Platforms/Devices controlbar, select Add to active Experiment to add the platform to the new experiment.

   The Define Platform Name dialog opens, prompting you to specify a name for the platform to be used in the experiment. The name must be unique in the experiment.

4. Click OK to confirm the default platform name.

   The registered platform is added to the experiment.

5. From the platform's context menu in the Project controlbar, select Add Variable Description.

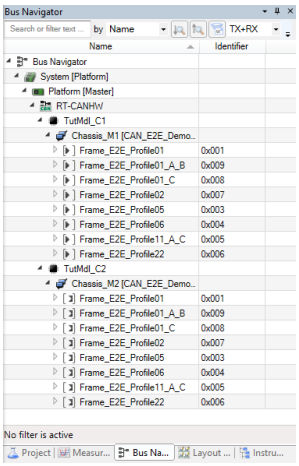
   ControlDesk opens the Select Variable Description dialog.

6. In the Select Variable Description dialog, click Import from file and select the `Lesson16_TutorialModel.sdf` file from the `RTICANMMTutorial_Lesson16` folder.

7. In the Select Variable Description dialog, click OK.

   ControlDesk closes the Select Variable Description dialog and adds the variable description (SDF file) to the platform.

8 Open the Bus Navigator.



9 In the Bus Navigator tree, right-click the Frame_E2E_Profile05 message below the Chassis_M1 [CAN_E2E_Demo_advanced.arxml] node and open the RX and TX layouts.

10 On the Home ribbon, click Status Control – Go Online to start online calibration.

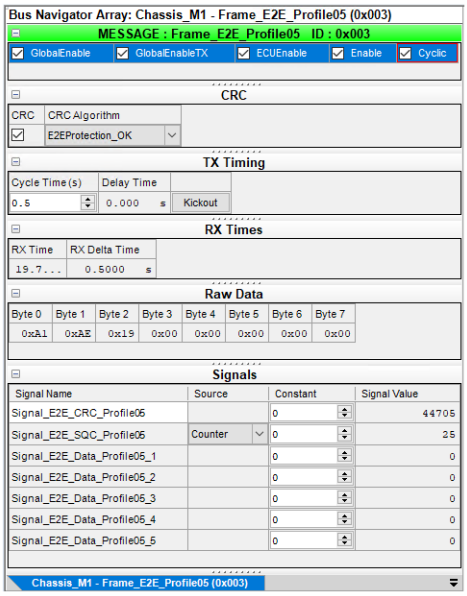
Because you disabled cyclic message transmission by default for all messages, there is no traffic on the bus and no message is transmitted.

11 In the TX layout of Frame_E2E_Profile05, select the Cyclic checkbox.

The message is transmitted via cyclic transmission, the RX time is running.

12 In the Cycle time (s) field, type **`0.500`** and press **`Enter`**.

This improves the visibility of value changes.



Frame_E2E_Profile05 is a message with eight bytes and seven signals. Because you specified Signal_E2E_SQC_Profile05 as a counter signal for the message, its signal value is incremented with each message transmission. According to the counter settings specified in the model, the Signal_E2E_SQC_Profile05 value is in the range 1 to 255 (which corresponds to the value range of counters according to AUTOSAR) and is incremented by 1 each time the message is transmitted. The white color of the Signal_E2E_CRC_Profile05 signal indicates that a checksum algorithm was specified for it.

Because E2EProtection_OK was specified as the default checksum algorithm for Frame_E2E_Profile05, it is selected from the CRC Algorithm list. No error is inserted during checksum calculation on the transmitting side.

`E2E_checksum_advanced.h` contains the `E2E_P05_Protect()` function that is called when checksum calculation is enabled for the Frame_E2E_Profile05 message and when cyclic message transmission is enabled or message kickout is performed. The following illustration shows some code lines from the function:

```
if(CRCBytePos - DataOffsetInBytes > 0)
{
    // Calculate CRC over the bytes before the CRC
    *CheckSum = (UInt32) Crc_CalculateCRC16((UInt32*)&(Msg->RAW_DATA[DataOffsetInBytes]), CRCBytePos - DataOffsetInBytes, 0xFFFF, 1);
    // Calculate CRC over the bytes after the CRC
    *CheckSum = (UInt32) Crc_CalculateCRC16((UInt32*)&(Msg->RAW_DATA[CRCBytePos+2]), DataLengthInBytes - (CRCBytePos - DataOffsetInBytes) - 2, (UInt16) *CheckSum, 0);
}
else
{
    *CheckSum = (UInt32) Crc_CalculateCRC16((UInt32*)&(Msg->RAW_DATA[CRCBytePos+2]), DataLengthInBytes - (CRCBytePos - DataOffsetInBytes) - 2, 0xFFFF, 1);
}

*CheckSum = (UInt32) Crc_CalculateCRC16(&DataIdLowByte,  1, (UInt16) *CheckSum, 0);
*CheckSum = (UInt32) Crc_CalculateCRC16(&DataIdHighByte, 1, (UInt16) *CheckSum, 0);
```

The checksum is calculated using the data bytes, including the counter signal. The unique DataID is also included in the checksum calculation.

On the receiving side, checksum calculation is performed using the received data. Successful transmission is verified by comparing the received CRC with the calculated one.

13 On the Home ribbon, click Status Control – Go Offline to stop online calibration.

14 Save the project.

Interim result

You checked the checksum behavior of the Frame_E2E_Profile05 message, which is protected by AUTOSAR E2E profile 05. You can now observe the checksum behavior of Frame_E2E_Profile11_A_C, where multiple E2E cases are modeled in one message. Continue with part 2.

Part 2

To check the behavior of checksums based on AUTOSAR E2E profile 11

1 In the Bus Navigator tree, right-click the Frame_E2E_Profile11_A_C message below the Chassis_M1 [CAN_E2E_Demo_advanced.arxml] node and open the RX and TX layouts.

2 On the Home ribbon, click Status Control – Go Online to start online calibration.

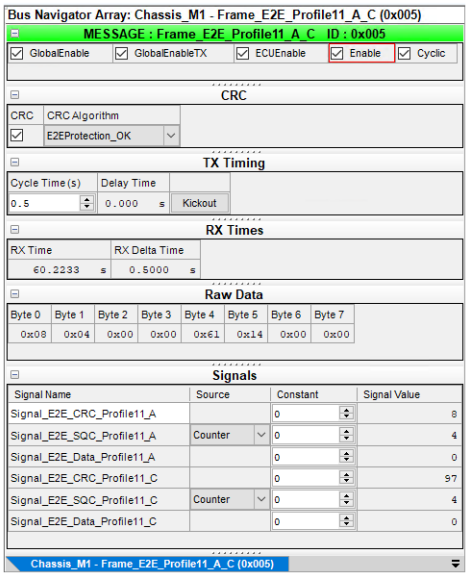
Because you disabled cyclic message transmission by default for all messages, there is no traffic on the bus and no message is transmitted.

3 In the TX layout of Frame_E2E_Profile11_A_C, select the Cyclic checkbox.

The message is transmitted via cyclic transmission, the RX time is running.

4 In the Cycle time (s) field, type **`0.500`** and press **`Enter`**.

This improves the visibility of value changes. The message is transmitted cyclically, the RX time is running.



Because you specified Signal_E2E_SQC_Profile11_A and Signal_E2E_SQC_Profile11_C as counter signals for the Frame_E2E_Profile11_A_C message, their signal values change with each message transmission. According to the counter settings specified in the model, the Signal_E2E_SQC_Profile11_A and Signal_E2E_SQC_Profile11_C values change with each message transmission. They are in the range 0 to 14 (which corresponds to the range defined by AUTOSAR for E2E profile 11) and are incremented by 1 each time the message is transmitted. The white color of the Signal_E2E_CRC_Profile11_A signal indicates that a checksum algorithm was specified for it.

Note

It is not possible to specify two checksum signals for a single message on the Checksum Messages page. One signal is sufficient for the RTI CAN MultiMessage Blockset to call the user code to calculate the checksum. If the database is modeled properly, `E2E_checksum_advanced.h` handles a message with multiple E2E cases correctly, i.e., it behaves as if there are multiple checksum signals in one message. For the example in this lesson, this means: byte 0 and byte 4 are both checksum signals.

E2EProtection_OK was specified as the default checksum algorithm for Frame_E2E_Profile11_A_C. Therefore, it is selected from the CRC Algorithm list. No errors are inserted during checksum calculation on the transmitting side.

`E2E_checksum_advanced.h` contains the `E2E_P11_Protect()` function. The function is called when checksum calculation is enabled for the Frame_E2E_Profile11_A_C message and when cyclic message transmission is

enabled or message kickout is performed. The listings below consist of excerpts from the `E2E_P11_Protect()` function:

```
switch(DataIDMode)
{
    case 0:
    {
        // E2E Profile 11A - E2E_P11_DATAID_BOTH
        DataIdLowByte  = DataID & 0xFF;
        DataIdHighByte  = (DataID >> 8) & 0xFF;
        // Calculate CRC over both Data ID bytes (low first)
        *CheckSum = (UInt32) Crc_CalculateCRC8(&DataIdLowByte, 1, 0xFF, 0);
        *CheckSum = (UInt32) Crc_CalculateCRC8(&DataIdHighByte, 1, *CheckSum, 0);
    }
    break;
    case 3:
    {
        // E2E Profile 11C - E2E_P11_DATAID_NIBBLE
        DataIdLowByte  = DataID & 0xFF;
        DataIdHighByte  = 0;
        *CheckSum = (UInt32) Crc_CalculateCRC8(&DataIdLowByte, 1, 0xFF, 0);
        *CheckSum = (UInt32) Crc_CalculateCRC8(&DataIdHighByte, 1, *CheckSum, 0);
        // Put lower nibble of high byte of the Data ID to the byte, where the counter is located
        DataIdHighByte  = (DataID >> 8) & 0xFF;
        Msg->RAW_DATA[(CounterPosition/8)] = (Msg->RAW_DATA[(CounterPosition/8)] & 0x0F) | ((DataIdHighByte & 0x0F)<<4);
    }
    break;
}


if((CRCBytePos - DataOffsetInBytes) > 0)
{
    // Calculate CRC over the bytes before the CRC
    *CheckSum = (UInt32) Crc_CalculateCRC8((UInt32*)&(Msg->RAW_DATA[DataOffsetInBytes]), CRCBytePos - DataOffsetInBytes, *CheckSum, 0);

    if (DataLengthInBytes > (CRCBytePos - DataOffsetInBytes) + 1)
    {
        // Calculate CRC over the bytes after the CRC
        *CheckSum = (UInt32) Crc_CalculateCRC8((UInt32*)&(Msg->RAW_DATA[CRCBytePos+1]), DataLengthInBytes - (CRCBytePos - DataOffsetInBytes) - 1, *CheckSum, 0);
    }
}
else
{
    // Calculate CRC over the bytes before the CRC
    *CheckSum = (UInt32) Crc_CalculateCRC8((UInt32*)&(Msg->RAW_DATA[CRCBytePos+1]), DataLengthInBytes - 1, *CheckSum, 0);
}
```

For CRC calculation, the CRC-8-SAE J1850 algorithm is used. The checksum is calculated using the data bytes including the counter signal. The Data ID is also used in the CRC calculation.

The first ISignal group is protected by E2E profile variant 11A. Both bytes of the 16-bit Data ID are used in the CRC calculation.

The second ISignal group is protected by E2E profile variant 11C. Here, only the low byte of the Data ID is included in the CRC calculation. The low nibble of the high byte of the Data ID is written to the high nibble of the byte containing the counter, and it is included in the checksum calculation. This explains why byte 1 has the value `0x04` and byte 5 the value `0x14`, even though both counter values are identical and have the value 4.

On the receiving side, checksum calculation is performed using the received data. Successful transmission is verified by comparing the received CRC with the calculated one.

5. On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.
6. Save the project.

Result

You checked the behavior of checksums based on E2E protection according to AUTOSAR in ControlDesk. First, you checked the checksum behavior of E2E profile 05. The second example addressed E2E profile 11 (variants 11A and 11C). You observed the checksum behavior of the Frame_E2E_Profile11_A_C message, for which several E2E cases are modeled.

What's next

In the next step (refer to Step 3: How to Use Debug Messages in the Checksum Header File on page 451), you can enable debug messages in the checksum header file and check their output in ControlDesk.

Related topics

Basics

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles 429
Overview of Lesson 16 430

Step 3: How to Use Debug Messages in the Checksum Header File

Objective

You can observe the execution of E2E protection mechanisms by using additional debug information specified in the checksum header file. If debug messages are enabled, ControlDesk writes them to the **Messages** controlbar during execution.

In this step, you will enable the output of E2E debug messages in the checksum header file. The debug messages provide additional information relevant for E2E protection. You will then view the effects in the **Messages** controlbar in ControlDesk.

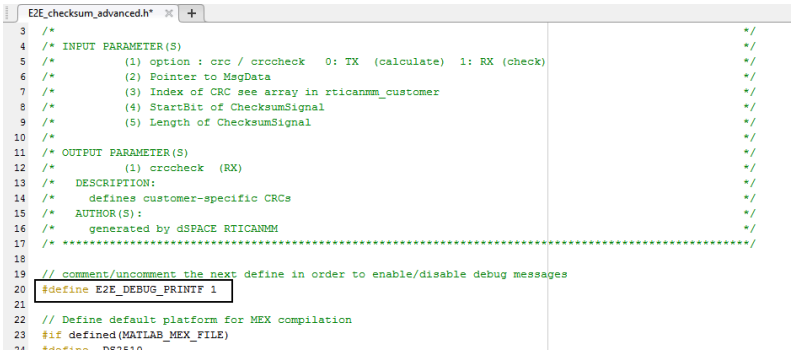
Part 1

To enable debug messages in the checksum header file

1. In MATLAB, double-click the **E2E_checksum_advanced.h** file in the **Current Folder** window.

   The E2E_checksum_advanced.h file opens in a separate editor window.

2. Uncomment line **20**. To do so, you must delete the comment characters that are placed at the beginning of the line. This can be done manually or via the **Uncomment** context menu command.

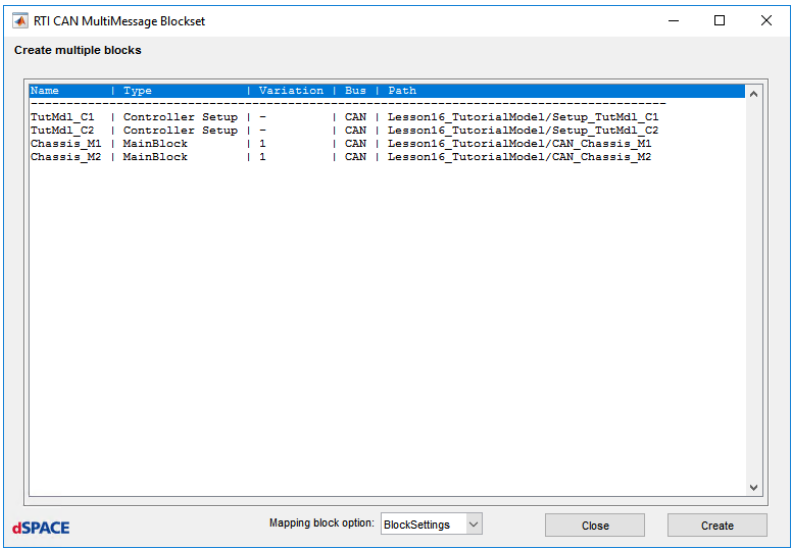


   By uncommenting E2E debug messages in the checksum header file, information relevant for E2E protection will be displayed in the **Messages** controlbar in ControlDesk. This information will give you a better understanding of how E2E protection works.

3. Save the **E2E_checksum_advanced.h** file.

4. Double-click the RTICANMM GeneralSetup block in the model and select **Options - Create S-function for Some CAN Blocks**.

A list containing all the blocks in the model opens.



5 Select the two MainBlocks and click Create.

The S-functions for the selected blocks are updated.

6 Click Close to close the dialog.

7 Click OK to close the RTICANMM GeneralSetup block.

8 Save the model.

Interim result

You enabled debug messages in the checksum header file. You can now check the generated info messages during execution in ControlDesk. Continue with part 2.

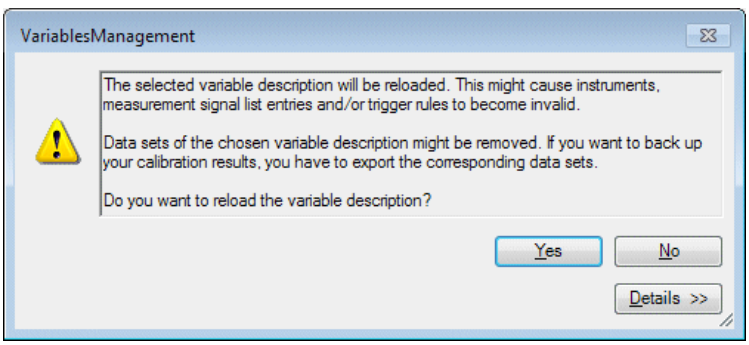
Part 2

To check the output of the debug messages in ControlDesk

1 Build an application for the model.

2 From the context menu of the `Lesson16_TutorialModel.sdf` file in the active ControlDesk experiment, select Reload Variable Description.

The following warning is displayed, informing you that reloading a variable description might cause instruments and other items to become invalid. Click Yes.



3 On the **Home** ribbon, click **Status Control – Go Online** to start online calibration.

Because you disabled cyclic message transmission by default for all messages, there is no traffic on the bus. No message is transmitted.

4 In the Bus Navigator tree, right-click **Frame_E2E_Profile01** below the **Chassis_M1 [CAN_E2E_Demo_advanced.arxml]** node and generate a TX layout.

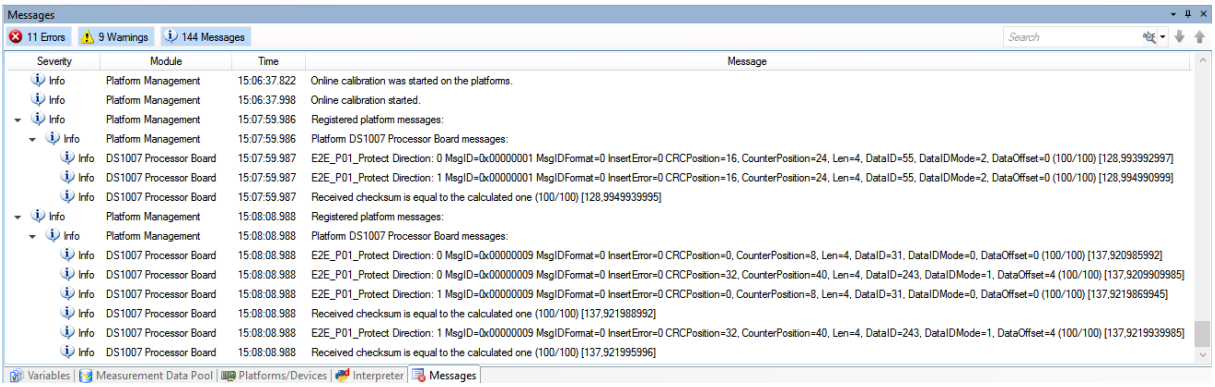
5 Right-click the **Frame_E2E_Profile01_A_B** message below the **Chassis_M1 [CAN_E2E_Demo_advanced.arxml]** node and open the TX layout.

6 In the **Chassis_M1 - Frame_E2E_Profile01** TX layout, click **Kickout**.

The **Frame_E2E_Profile01** message is transmitted once via kickout. The values of the **RX Time** and **RX Delta Time** change.

7 In the TX layout of **Frame_E2E_Profile01_A_B**, click **Kickout**.

8 Check the **Messages** controlbar, which displays the debug messages that are written during the message transmission.



The debug messages provide the following information:

Message Part	Description
Direction	Direction of protection: ▪ 0: RX ▪ 1: TX
MsgID	Message ID of the protected message.
MsgIDFormat	ID format of the protected message.
InsertError	Flag indicating whether an error was inserted intentionally: ▪ 0: No error intentionally inserted. ▪ 1: Error intentionally inserted.
CRCPosition	Position of CRC signal (in bit representation).
CounterPosition	Position of counter signal (in bit representation).
Len	Length of data to be protected (in bytes).
DataID	Data ID, unique in the specified E2E profile protection set.
DataIDMode	Data ID mode, unique in the specified E2E profile protection set.

Message Part	Description
DataOffset	Offset of data to be protected in the IPDU.

9 On the **Home** ribbon, click **Status Control – Go Offline** to stop online calibration.

10 Save the project.

Result

You modified the checksum header file by enabling further debug messages. You then created the S-functions for the CAN_Chassis_M1 MainBlock and the CAN_Chassis_M2 MainBlock again.

In ControlDesk, you reloaded the variable description and performed message transmission via kickout for Frame_E2E_Profile01 and Frame_E2E_Profile01_A_B. In the **Messages** controlbar, you checked the resulting info messages that were output during message transmission.

What's next

With this step, you finished working with checksums based on E2E protection data according to AUTOSAR end-to-end protection profiles. For a summary of your working results and for more information, refer to the result of this lesson.

Related topics

Basics

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles..... 429
Overview of Lesson 16..... 430

Result of Lesson 16

Result

In this lesson, you learned the principles of working with E2E communication protection mechanisms according to AUTOSAR E2E profiles in user-defined checksum calculations:

- You must enable the use of E2E protection parameters. If enabled, an additional header file (named `RTICANMM_MAIN_<CAN name>_E2EP.h`) is generated and included in the S-function of the RTICANMM MainBlock. The E2E protection header file is also saved to the destination folder.
- The generated header file contains the E2E protection description data defined in the database and the functions for calling this data in checksum algorithms. You can use the E2E protection data in checksum algorithm implementations.
- The `E2E_checksum_advanced.h` file you selected as the checksum header file contains E2E protection functions based on the AUTOSAR requirements. It defines two checksum cases (E2E protection without/with error) using end-to-end protection data provided by the database specified for the model.

The `E2E_checksum_advanced.h` file can be used as a template. It covers all E2E profiles supported by the RTI CAN MultiMessage Blockset.

- You can enable E2E debug messages in the checksum header file to display information relevant to E2E protection during execution in ControlDesk. The information is displayed in the **Messages** controlbar in ControlDesk and gives you a better understanding of how E2E protection works.

Prepared demo

You can reproduce the result of this lesson or compare your own results with the **Lesson16_TutorialModel.slx** demo model, refer to Working with the Prepared Demo Results on page 32.

Further information

For further information, refer to:

- Checksum Definition Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Checksum Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- TX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Raw Data Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
- RX Error Display Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)

Where to go from here

You successfully completed the tutorial. For a short overview of the lesson results and for further examples, refer to the summary of this tutorial.

Related topics

Basics

Lesson 16 (Advanced): Working with E2E Communication Protection According to AUTOSAR E2E Profiles 429
Overview of Lesson 16 430

Summary

Where to go from here

Information in this section

Your Working Results.....457
You will find a brief summary of your working results and tips on how to continue working after finishing this tutorial.

Further Examples.....458
You will find a brief introduction to the available demo projects and how to access them.

Your Working Results

What you learned

You have finished your work successfully. The models you created in this tutorial cover typical use cases. You are now able to derive the concepts of the RTI CAN MultiMessage Blockset from the various lessons and transfer them to your daily work.

What's next

You can continue working in different ways:

- If you need further examples, refer to Further Examples on page 458.
- If you want more detailed information on the RTI CAN MultiMessage Blockset, refer to the RTI CAN MultiMessage Blockset Reference 🕮.

- If you want more detailed information on the **Bus Navigator** of ControlDesk, refer to ControlDesk Bus Navigator 🕮.

Related topics

Basics

Further Examples.....458
Overview of Lessons.....35

Further Examples

Demo projects

If you need further examples for using the RTI CAN MultiMessage Blockset, you can refer to the demo projects which come with the RTICANMM Blockset.

Available demo projects The following demo projects are available:

- Simple model
- Multiprocessor system
- Full periphery
- Signal manipulation
- Custom coding
- Checksum coding
- Gateway block
- Gateway signals

Accessing the demo projects

1. Type **`rticanmm`** in your MATLAB Command Window to open the rticanmmlib library.
2. In the rticanmmlib library, double-click Demos.

   The rticanmm_demo_lib library opens and provides access to the available demo projects.

Related topics

Basics

Your Working Results.....457

Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the RTI CAN MultiMessage Blockset documentation.

Where to go from here

Information in this section

A....460
B....460
C....463
D....466
E....467
F....467
G....468
I....470
J....471
K....473
L....473
M....473
P....476
R....477
S....478
T....480
V....482

A

Address claiming (J1939) A process defined in the J1939 standard in which addresses are autonomously assigned to J1939 network nodes during network initialization. If there is an address conflict, the network node with the highest priority gets the claimed address. The other network nodes must claim different addresses. This dynamic node address assignment ensures that each address is unique (J1939 network management).

Alive counter A counter signal that checks the correct transmission of a message. When a counter signal is used in this way, the receiver of a message has to expect a certain counter signal value. If the transmission of a message stops for two transmissions, for example, the expected counter signal value and the real counter signal value can differ, which indicates an error.

You can specify counter signals on the Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

AUTOSAR system description file A file of AUTOSAR XML file type that describes a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. AUTOSAR system description files are instances of the AUTOSAR System Template. The AUTOSAR System Template contains a description of the network communication and hardware topology according to the FIBEX standard defined by ASAM e.V. AUTOSAR system description files can be used to export or exchange information on system descriptions.

You can specify an AUTOSAR system description file as the database for an RTICANMM MainBlock. An AUTOSAR system description file usually describes more than one bus system. You therefore have to select one of the available bus systems on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

B

Broadcast message (J1939) A message that is transmitted to any network node connected to a CAN-J1939 network.

Tip

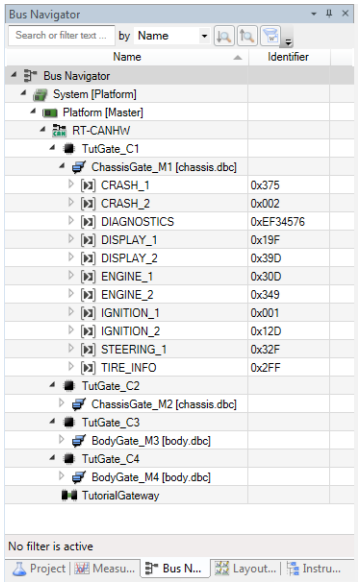
Broadcast messages are also called PDU2 messages.

Bus Navigator A controlbar in ControlDesk for handling bus messages, such as CAN, LIN, and FlexRay messages, frames, and PDUs and Ethernet packets.

All the information on the CAN bus configuration is displayed in the Bus Navigator tree. The Bus Navigator tree lets you perform actions such as the following:

- Create layouts to view RX messages and configure TX messages.
- Trigger the transmission of TX messages.
- Change the active variant of a CAN controller.
- Monitor raw data and physical data, and log the raw data of CAN messages.
- Replay logged CAN bus communications.
- Select several messages in the Bus Navigator tree to open them in one go, etc.

Bus Navigator tree A part of the Bus Navigator that displays the CAN controllers, the assigned database files and messages of a CAN bus configuration hierarchically.



The Bus Navigator tree contains, for example, the following nodes:

- **Bus Navigator**: This is the root node of a bus configuration in the Bus Navigator tree and displays the system(s) used.
- **System [Platform]**: This node displays the system you are working with. You can configure layout names, configure monitoring settings, etc. via context menu.
- **Platform [Master]**: This node displays the name of the bus configuration that is loaded to the processor. This node has no context menu.
- **RT-CAN HW**: This node contains all the CAN controllers configured using RTICANMM blocks. You can open a monitoring list, define filters (for example, for CAN bus monitoring), etc. via context menu.

Tip

When you define filters, the filters are displayed as nodes on this hierarchy level.

- **TutGate_C1**, **TutGate_C2**, ...: These nodes display the controllers configured with the RTICANMM ControllerSetup blocks. You can open a monitoring list, specify filters, etc. via context menu.

Tip

When you define filters, they are displayed as nodes on this hierarchy level.

- **ChassisGate_M1 [chassis.dbc]**, **ChassisGate_M2 [chassis.dbc]**, ...: These nodes display the CAN bus names and database files specified in the corresponding RTICANMM MainBlocks and contain the message configurations. You can open TX status layouts, activate the database (only if you work with variants of a CAN controller), etc. via context menu.
- **TIRE_INFO**, **STEERING_1**, ...: These nodes display the messages specified in the database file. The icons indicate whether the messages are RX messages, TX messages, or RX and TX messages:

Icon	Message Type
[▶]	TX
[◀]	RX
[▶◀]	TX/RX

You can open TX and/or RX message layouts via context menu.

- **TirePressure_FrontLeft**, **TirePressure_FrontRight**, ...: These nodes display the signals of the messages. These nodes have no context menus.
- **TutorialGateway**: This node displays a gateway between the ChassisGate_M2 and BodyGate_M3 CAN bus. In the Simulink model, the gateway is defined by an RTICANMM Gateway block. You can open a gateway layout via context menu.

Note

- The node names depend on the names defined in the model (for example, the CAN bus names) and in the database file(s) (for example, the message names).
- The nodes displayed in the Bus Navigator tree depend on the defined bus configuration.

C

CAN bus configuration Files created during the build process of a Simulink model containing blocks of the RTI CAN MultiMessage Blockset. ControlDesk requires the files for observing and manipulating CAN communication defined in that model. For some communication settings, manipulation via ControlDesk has to be enabled explicitly by selecting TRC, DEFAULT TRC or BOTH on the appropriate property page of the RTICANMM MainBlock.

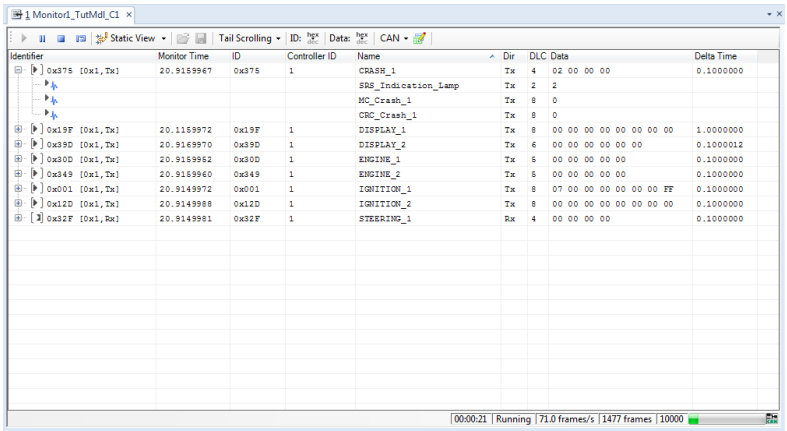
When an SDF file for such an application is loaded in ControlDesk, the CAN bus configuration referenced by the SDF file is also loaded and displayed in the Bus Navigator tree.

CAN bus monitoring A feature of the Bus Navigator that lets you observe all the messages transmitted on a CAN bus. You can monitor the raw data and the physical data of CAN messages.

You can monitor CAN messages in two ways:

- Unfiltered CAN bus monitoring

  This is a simple way of monitoring all the CAN messages of a CAN communication. You can open an unfiltered monitoring list from the context menus of the system node, a hardware node, and/or a CAN controller node in the Bus Navigator.



- Filtered CAN bus monitoring

  You can specify filters for monitoring selected CAN messages. Each system, hardware, or controller node can have its own filters. Each filter can contain one or more filter items. You can open a filter dialog from the context menu of a filter node in the Bus Navigator to define the filter settings.

> **Tip**
>
> The specified filters are displayed as nodes of the Bus Navigator tree.

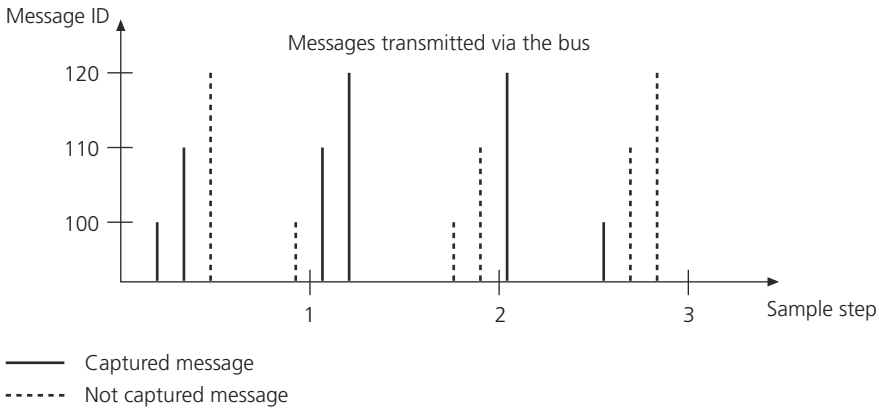
CAN bus monitoring with the Bus Navigator always uses a certain host service. The host service used for CAN bus monitoring is displayed in the RTICANMM

GeneralSetup block for information purposes. Refer to Main Page (RTICANMM GeneralSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

Capture message A message that is independent of the database file and captures messages on a CAN bus. Each capture message can catch one message per sampling step, so the values of the capture messages often change within milliseconds.

- Example of the behavior of capture messages

  Suppose three messages (ID 100, 110, and 120) are transmitted via the bus and you defined two capture messages. The capture behavior could look like this:



  Since each capture message can catch only one message per sampling step, not all of the messages transmitted via the bus are captured.

Capture Message	Message Captured in ...		
	Sample Step 1	**Sample Step 2**	**Sample Step 3**
1	ID 100	ID 110	ID 120
2	ID 110	ID 120	ID 100

- Calculating the maximum number of messages in one sampling step

  To ensure that all the messages of one sampling step are captured, you can use the model's baud rate and sample time to calculate the maximum number of messages available on the bus in one sampling step. This number is the number of capture messages needed to capture all the messages.

  Tip

  For an example on calculating the maximum number of messages available on the bus, refer to Capture Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

- Identifier filter

  You can specify an identifier filter for capture messages. Only messages whose identifiers match the specified filter value are captured.

- Field of application for capture messages

  A typical field of application for capture messages is network management such as that defined by the OSEK-NM standard. For example, capture messages can be used to monitor the state of the CAN bus nodes.

You can specify capture messages on the Capture Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Checksum algorithm (Custom) A user-defined algebraic algorithm that ensures data integrity for the transmission and reception of a message. Checksum algorithms have to be defined in a header file and assigned to a signal of a message to check for correct message transmission/reception. The header file needs to have a C-coded switch-case directive to switch between algorithms. Checksums apply to TX messages and RX messages:

- Checksum calculation for TX messages

  A checksum is calculated according to the applied algorithm and assigned to the TX message. Then the message is transmitted together with the calculated checksum.

- Checksum check for RX messages

  A checksum is calculated for the received message and compared to the checksum that was sent together with the message. If they differ, an error occurs.

You can define checksum algorithms and apply them to messages on the pages located in the Checksum Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Container message (J1939) A J1939 message defined by its PGN (and Data Page bit). A container can receive all the messages with its PGN. If several messages are received in one sampling step, only the last received message is held in the container (J1939-compliant DBC file and RTICANMM).

Counter signal A signal of a TX message that is used to check for correct message transmission or to trigger the transmission of signals in a message.

- Behavior of counter signals

  The value of a counter signal changes with every message transmission. You can specify the counter start, step, and stop values, etc. Each time the counter reaches the stop value, it turns around to the counter start value.

- Use of counter signals

  You can specify counter signals to check for correct transmission of a message or trigger the transmission of signals in a message.

  - *Checking correct message transmission*: The receiver of a message expects a certain counter signal value. For example, if the transmission of a message was stopped for two transmissions, the expected counter signal value and the real counter signal value can differ, which indicates an error. Counter signals used in this way are often called alive counters.
  - *Triggering the transmission of signals*: You can trigger the transmission of signals if you specify a mode signal as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. Because the counter signal value changes with every message transmission,

this triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message. Counter signals used in this way are often called mode counters.

- Using counter signals in ControlDesk

  In ControlDesk, you can transmit the signal's constant, counter, or increment counter value. The increment counter value is the counter value incremented by the signal's constant value.

You can specify counter signals on the Counter Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

Cyclic message transmission A way of transmitting messages.

D

Database file A file containing the definition of CAN communication. You can use one of the following files as a database file:

- DBC
- FIBEX
- MAT file
- AUTOSAR system description file

You have to specify the database file on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference). Alternatively, you can work without a database file by using free raw message and/or capture message only.

DBC A specific database file format containing the entire CAN communication of a CAN bus. A DBC file (Data Base Container file) can contain default values for ECU-, message-, and signal-specific attributes, for example, the cycle time for messages. The RTI CAN MultiMessage Blockset provides a DBC file parser that allows you to specify a DBC file as the database.

- Consistency check for DBC files

  The RTI CAN MultiMessage Blockset's DBC file parser automatically performs a consistency check on the DBC file you specify. This check is similar to the consistency check of Vector Informatik's CANdb++ Editor. In other words: For the RTI CAN MultiMessage Blockset, you can use any DBC file that passes the check of the CANdb++ Editor.

- DBC file with messages of standard and extended identifier format

  You can use DBC files as databases which contain messages of standard and extended identifier format.

For more information on DBC files, refer to General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Dynamic value A signal value that is transmitted for a defined number of times.

- *Behavior of dynamic values*: You can specify to use a dynamic value for each signal to be transmitted. In ControlDesk, you can specify to transmit the signal's constant value or dynamic value. If you switch to the dynamic value, it is transmitted for a defined number of times (countdown value). Then signal manipulation automatically switches back to the signal manipulation option used before the dynamic value.
- *Example of using dynamic values*: Suppose you specify a dynamic value of 8 and a countdown value of 3. If you switch to the dynamic value of the signal, the signal value 8 is sent the next 3 times the TX message is transmitted. Then the signal manipulation option is reset.

You can specify dynamic values on the pages located in the Dynamic Signal Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

E

Error value A static signal value that indicates an error. You can specify an error value for each signal to be transmitted. Alternatively, error values can be defined in a database file. In ControlDesk, you can switch to transmit the signal's error value, constant value, etc. However, you cannot change the error value during run time. In ControlDesk, you can use a Variable Array (MultiState LED value cell type) to indicate if the error value is transmitted.

You can specify error values on the Signal Errors Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

F

FIBEX A specific database file format containing the entire CAN communication for several CAN buses. The Field Bus Exchange (FIBEX) format is an XML exchange file format. It is used for data exchange between different tools that work with message-oriented bus communication. You can specify a FIBEX file as the database for an RTICANMM MainBlock. A FIBEX file usually describes more than one bus system. You therefore have to select one of the available bus systems on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

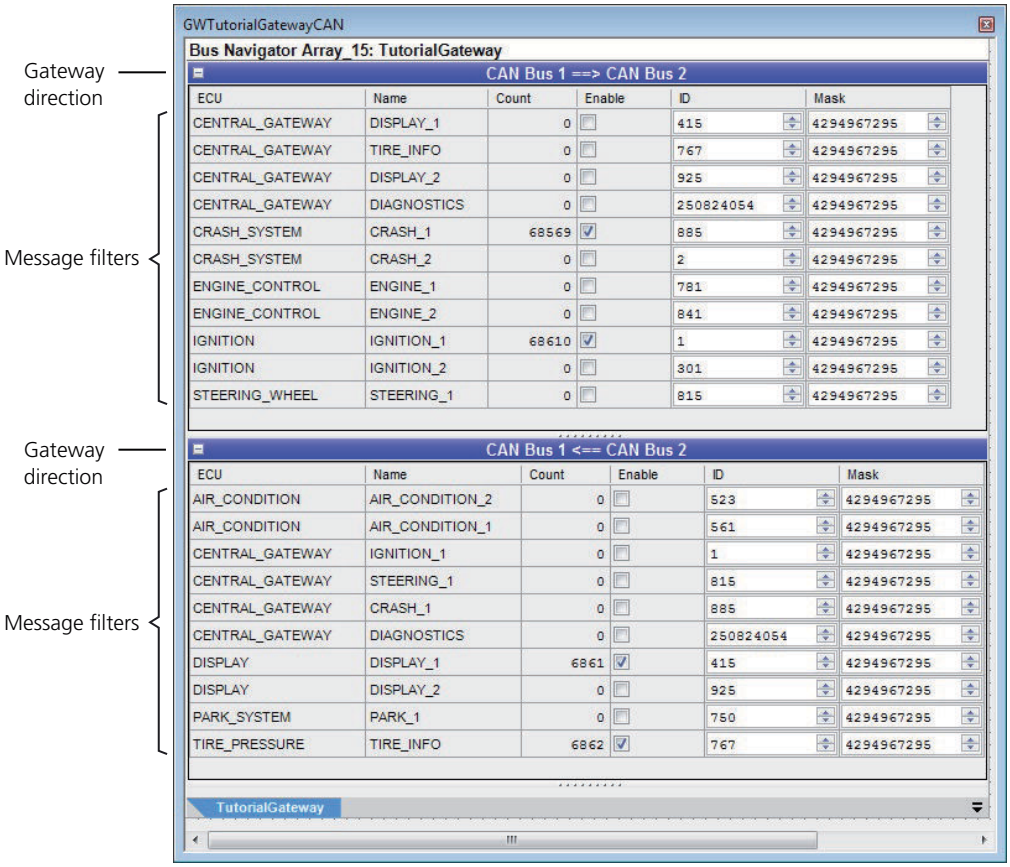
Free raw message A message that is independent of a database file. Free raw messages always consist of eight (CAN) or 64 (CAN FD) 8-bit signals and are TX messages and RX messages. You can modify free raw messages

completely, including their identifiers and message lengths. However, if you change an ID during run time, make sure that no other messages (also no other free raw messages) use this ID at the same time.

You can specify free raw messages on the Free Raw Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

G

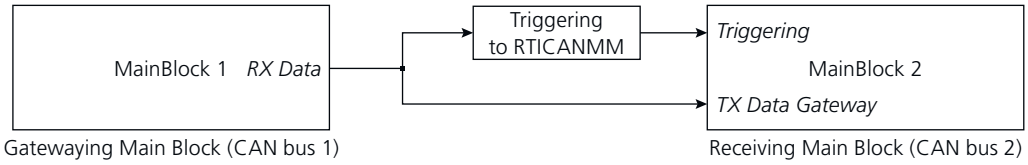
Gateway layout A specific layout of the Bus Navigator for determining which messages are to be gatewayed during run time.



This layout type is available only for gateways using an RTICANMM Gateway block and if changing the message exclude lists during run time is enabled. The layout displays message filters for each gateway direction. You can use the message filters to gateway messages or exclude them from being gatewayed. A hit counter indicates that a message filter is active and working accurately. For

J1939 messages, the message filters also display the message priority, the source ECU, and the destination ECU.

Gateway signal A signal to be manipulated before it is exchanged between two CAN buses. Gateway signals have to be gatewayed via two RTICANMM MainBlocks. You have to specify gateway signals for the receiving MainBlock.



MainBlock1 gateways messages and their signals to MainBlock2. Specifying gateway signals for MainBlock2 adds a **TX Data Gateway** inport to it. The specified gateway signals are transmitted via CAN bus 2 with the signal values received from MainBlock1 when triggered by the messages received from MainBlock1. You therefore have to specify triggered message transmission for the messages of the gateway signals on the Message Cyclic Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference). In addition, you can enable signal value switching for gateway signals during run time, for example, to transmit the signal constant value instead of the gateway value.

Note

Implementing gateway signals at least doubles the number of block inports and therefore reduces performance. If you want to exchange messages between CAN controllers and do not want to perform signal manipulation, you should use the RTICANMM Gateway block instead.

You can specify gateway signals on the pages located in the Gateway Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

Gatewaying messages Exchanging messages via the model between two CAN buses that are not connected physically. With the RTICANMM blockset, there are two ways to gateway messages:

- Gateway via the RTICANMM Gateway block

  Using the RTICANMM Gateway block, you can gateway a large number of messages between two RTICANMM MainBlocks or between two RTICANMM ControllerSetup blocks. Messages gatewayed by the Gateway block cannot be manipulated before gatewaying. On the dialog pages of the Gateway block, you can specify the gateway direction, exclude messages from being gatewayed, enable to change the exclude message lists during run time, etc. If changing the exclude message lists during run time is enabled, you can open a gateway layout in ControlDesk to manipulate the gateway behavior.

  Tip

  For optimum performance, you should gateway as many messages as possible via the Gateway block.

- Gateway via two RTICANMM MainBlocks

  Gatewaying messages using two RTICANMM MainBlocks allows you to manipulate the messages before gatewaying. For the sending MainBlock, you have to specify which messages and signals are to be gatewayed. For the receiving MainBlock, you have to specify gateway signals to transmit the gatewayed signal values via the receiving CAN bus.

  Note

  Gatewaying messages via two RTICANMM MainBlocks reduces performance, so you should gateway as few messages as possible in this way.

Identifier format A CAN message can be of either standard (STD) or extended (EXT) identifier format.

- Messages of STD identifier format

  They contain an 11-bit identifier. It corresponds to the CAN 2.0 A specification.

- Messages of EXT identifier format

  They contain a 29-bit identifier. It corresponds to the CAN 2.0 B specification.

  Tip

  J1939 messages are always of EXT identifier format.

- DBC files with messages of STD and EXT identifier format

  You can use DBC files containing messages of STD and EXT identifier format as databases. On the Setup Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference), you can specify the priorities for the message identifier formats. In the case of communication overflow, the message identifier format with the higher priority will be transmitted preferentially, only messages of the lower-priority identifier format might be lost.

  Note

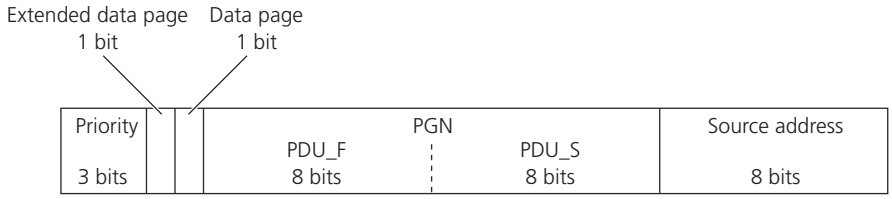
  If you work with a J1939-compliant DBC file, you have to select the EXT identifier format.

Instance message (J1939) A J1939 message defined by its PGN (and Data Page bit), for which the source network node and the destination network node are defined (destination only for peer-to-peer communication).

J

J1939 A vehicle CAN bus standard defined by SAE. The J1939 bus standard is used for communication in heavy-duty vehicles, for example, for communication between a tractor and its trailer.

J1939 messages use an extended 29-bit CAN message identifier (CAN 2.0 B). The 29-bit message identifier is split into different parts (according to SAE J1939/21 Data Link Layer):



29-bit CAN message ID

- The 3-bit *priority* is used during the arbitration process. A value of 0 represents the highest priority, a value of 7 represents the lowest priority.
- The 1-bit *extended data page* can be used as an extension of the PGN. The RTI CAN MultiMessage Blockset lets you specify whether to use the extended data page bit this way. Refer to Code Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).
- The 1-bit *data page* is a selector for the PDU_F in the PGN. It can be taken as an extension of the PGN. The RTI CAN MultiMessage Blockset uses the data page bit in this way.
- The 16-bit *PGN* is the parameter group number. Depending on its PGN, a message defined in a J1939-compliant DBC file can be either a broadcast message or a peer-to-peer message.
- The 8-bit *source address* is the address of the transmitting network node.

You can use J1939-compliant DBC files as databases when you work with RTICANMM blocks.

J1939 network management The management of a network of J1939 network nodes. J1939 network management is based on the address claiming process. The dynamic node address assignment ensures unique identification of network nodes and their primary functions. Each network node in a J1939 network must therefore hold one name and one associated address for identification purposes:

- Name

  The J1939 standard defines a 64-bit identifier to identify each network node. The *name* indicates the main function of the network node with the associated address and provides information on the manufacturer.

Arbitrary Address Capable	Industry Group	Vehicle System Instance	Vehicle System	Reserved	Function	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 bit	3 bit	4 bit	7 bit	1 bit	8 bit	5 bit	3 bit	11 bit	21 bit

- Address

  The 8-bit network node *address* defines the source or destination for J1939 messages in the network. The address of a network node must be unique. If there is an address conflict, the network nodes try to perform dynamic network node addressing (address claiming) to ensure unique addresses, if this is enabled for the network nodes.

You can specify the node addresses and 64-bit names of network nodes for identification purposes on the Network Node Identification Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

The RTI CAN MultiMessage Blockset supports J1939 network management including address claiming for self-configurable address network nodes. This allows network nodes simulated by the RTI CAN MultiMessage Blockset to change their addresses, if necessary, and to update their internal address assignments if addresses of external network nodes are changed.

J1939-compliant DBC file and RTICANMM A J1939-compliant DBC file can be used as the database when working with RTICANMM blocks.

- EXT identifier format

  When you work with a J1939-compliant DBC file in RTICANMM, you have to give messages with an EXT identifier format a higher priority than messages with an STD identifier format. You can select the identifier format on the Setup Page (RTICANMM ControllerSetup) (RTI CAN MultiMessage Blockset Reference 🕮).

- Message attributes in J1939-compliant DBC files

  The ID attribute is one of the attributes describing a message in a J1939-compliant DBC file:

  - In a DBC file created with CANalyzer 5.1 or earlier, the ID attribute describing a message actually is the *message PGN*. Thus, the ID provides no information on the source and destination of the message. The source and destination can be specified by the *J1939PGSrc* and *J1939PGDest* attributes.
  - In a DBC file created with CANalyzer 5.2 or later, the ID attribute describing a message actually is the *CAN message ID, which consists of the priority, PGN, and source address*. Thus, the ID provides information on the source and destination of the message. Further senders of a message can be specified in Vector Informatik's CANdb++ Editor (*_BO_TX_BU* attribute). When a DBC file is read in, RTICANMM automatically creates new instances of the message for the further senders of a message.

- Source/destination mapping

  Each message defined in a J1939-compliant DBC file can be either a broadcast message or a peer-to-peer message. The transmitting network node and the receiving network node (only for peer-to-peer messages) can be specified for each J1939 message.

- Container and instance messages

  The RTI CAN MultiMessage Blockset distinguishes between container messages and instance messages when you work with a J1939-compliant DBC file.

There exists one container for each PGN (except for proprietary PGNs). If you work with DBC files created with CANalyzer 5.1 or earlier, the container can be clearly derived according to its name from the DBC file. With DBC files created with CANalyzer 5.2 or later, several messages with the same PGN might be defined. In this case, either the message with the shortest name or an additionally created message (named `CONT_<shortest_message_name>`) is used as the container for the PGN. You can select the container type on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference). If several messages fulfill the condition of the shortest name, the one that is listed first in the DBC file is used. For messages with proprietary PGNs, each message is its own container (because proprietary PGNs can have different contents according to their source and destination nodes).

K

Kickout A way of transmitting messages once.

L

Layout (RTICANMM) A feature of the Bus Navigator for observing and/or manipulating the transmission of messages and signals. For a model using RTICANMM blocks, various layout types are available:

- TX layout
- RX layout
- Gateway layout

Loopback message A message that is transmitted and received by the same CAN controller. In CAN communication, every message transmitted by a CAN controller is also received as a loopback message. You cannot disable the reception of loopback messages, but you can prevent the RX status of an RTICANMM MainBlock from being changed by loopback messages (refer to Code Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference)).

M

MAT file A specific file format that can contain the entire CAN communication of a CAN bus. You can use a MAT file as the database for an

RTICANMM MainBlock. You can also specify other file formats as the database via the MAT file format. You must convert your specific database files into the MAT file format for this purpose. To use a MAT file as the database, it has to be of a certain RTICANMM-compliant structure. For a detailed description of the required structure, refer to General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Message A CAN bus element to exchange data between network nodes in a CAN network. Messages can be transmitted (TX, TX message) or received (RX, RX message) by a CAN controller. Usually, the payload data of messages is partitioned for the transmission of different signals. According to the CAN bus standard, messages can be of either standard (STD) or extended (EXT) identifier format. J1939 messages are always of EXT identifier format.

Message manipulation Options that are provided by the RTI CAN MultiMessage Blockset for manipulating messages before they are transmitted. Manipulation options can be static or dynamic.

- Static manipulation options

  Static manipulation options let you define checksum algorithms, or enable manipulation of the ID and length of a message during run time, etc. For several options, you can specify if they are adjusted via the model and/or ControlDesk during run time.

  You can specify static manipulation options on the pages located in the Message Manipulation Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

- Dynamic manipulation options

  Dynamic message manipulation options let you change message settings for a defined number of message transmissions during run time. You can define dynamic checksum, dynamic message length, and timeout manipulation.

  - *Dynamic checksum and dynamic message length*: During run time, you can change the checksum algorithm or the length of a message for a defined number of message transmissions. Afterwards, the previous checksum algorithm/message length applies again.
  - *Timeout*: During run time, you can disable the transmission of a message for a defined number of possible message transmissions. The message is not transmitted even if transmission is triggered by other trigger conditions.

  You can specify dynamic manipulation options on the Triggering Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮), TX Timeout Enable Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮), and Manipulation Options Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

Message transmission Options that are provided by the RTI CAN MultiMessage Blockset to customize the transmission of messages in several ways. For example, you can:

- Enable or disable the transmission of messages for some or all ECUs.
- Enable or disable the transmission of a single message or all the messages configured by one RTICANMM MainBlock.

- Specify cyclic message transmission. This lets you transmit messages cyclically with a certain cycle or delay time. You can specify user-defined cycle or delay times or use the values specified in the database file.
- Specify message transmission via kickout. This lets you transmit messages once immediately, for example, to let the transmission of a message be triggered by another message.

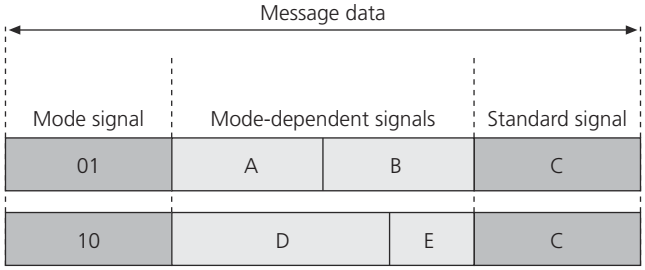
You can enable to change the different transmission options via the model and/or ControlDesk. You can specify the transmission options on the pages located in the Triggering Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

Mode counter A mode signal that is specified as a counter signal. The signal value of the mode signal triggers the transmission of mode-dependent signals. When a mode signal is specified as a counter signal, its signal value changes with every message transmission. This triggers the transmission of the mode-dependent signals. By using counter signals in this way, you can work with signals which use the same bytes of a message.

Mode signals and mode-dependent signals One signal of a message triggering the transmission of other signals of the same message. You can define mode signals and mode-dependent signals in the database file.

- Behavior of mode signals and mode-dependent signals

  The value of the mode signal triggers the transmission of the mode-dependent signals. Several mode-dependent signals can be transmitted via the same bits of a message if they are triggered by different mode signal values.



Tip

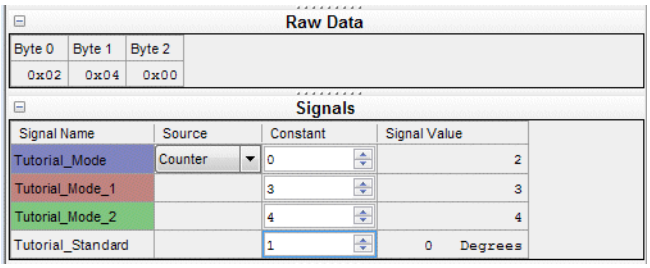
For an overview of the mode signals specified in the database file, click **Show Check Report** on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference) of the RTICANMM MainBlock in your model. For messages with a mode signal, the mode signal, the mode-dependent signals, and the mode values of the mode-dependent signals are listed.

- Mode signals and counter signals

  You can define the mode signal as a counter signal. In this case, the mode signal value changes with every message transmission. This triggers the transmission of different mode-dependent signals.

- Mode signals and mode-dependent signals in ControlDesk

  Mode signals and the transmission states of mode-dependent signals are indicated by layout colors. Mode signals are violet, active mode-dependent signals are green, and inactive mode-dependent signals are red.



P

Parity signal A signal that a parity bit is appended to. You can specify one or more signals of a TX message as parity signals. A parity bit is calculated for the specified signals according to whether even or odd parity is selected. The bit is appended to the signal and the TX message is transmitted with the parity signal.

You can specify parity signals on the Parity Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

Peer-to-peer message (J1939) A CAN message that is transmitted to only one destination network node in a CAN-J1939 network.

> **Tip**
>
> Peer-to-peer messages are also called PDU1 messages.

PGN A 16-bit number in the 29-bit message identifier of a CAN message defined in a J1939-compliant DBC file. Each parameter group number (PGN) references a *parameter group* that groups parameters and assigns them to the 8-byte data field of the message. For example, a parameter group can be the engine temperature including the engine coolant temperature, the fuel temperature, etc. PGNs and parameter groups are defined by the SAE (see SAE J1939/71 Vehicle Application Layer).

The first 8 bits of the PGN represent the *PDU_F* (Protocol Data Unit format). The PDU_F value specifies the communication mode of the message (peer-to-peer or broadcast). The interpretation of the *PDU_S* value (Protocol Data Unit specific) depends on the PDU_F value. For messages with a PDU_F < 240 (peer-to-peer communication), PDU_S is not relevant for the PGN, but contains the destination address of the network node that receives the message. For messages with a PDU_F ≥ 240 (broadcast messages), PDU_S specifies the second 8 bits of the PGN and represents the group extension. The group extension is used to increase the number of messages that can be broadcast in the network.

PDU_F (first 8 bits)	PDU_S (second 8 bits)	Communication Mode
< 240	Destination address	Peer-to-peer (message is transmitted to one destination network node)
≥ 240	Group extension	Broadcast (message is transmitted to any network node connected to the network)

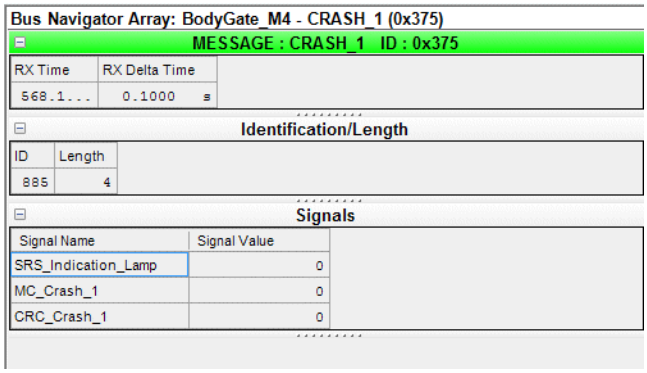
The *Data Page* bit in the 29-bit message identifier of a J1939 message is a selector for the PDU_F in the PGN. It is taken as an extension of the PGN. The RTI CAN MultiMessage Blockset treats the data page bit as an element of the PGN.

R

Raw data Payload data of a CAN message as transmitted on the bus. According to a data type and given scale and offset values, raw data can be interpreted as signals.

Restbus simulation A typical use scenario for the RTI CAN MultiMessage Blockset in which one or more real ECUs are connected via a bus to a simulator that simulates all the other ECUs of the bus. All the messages and signals of the simulated ECUs can be manipulated and the effects on the real ECU(s) observed via the simulator. To check the correct transmission of messages, checksum messages and counter signals can be defined for the simulated ECUs.

RX layout A specific layout of the Bus Navigator for displaying general information of an RX message.



RX layouts are always view-only. As with a TX layout, a green header indicates that the message can be received. However, if the message is actually received is indicated by a running RX time only.

Tip

The header of an RX layout remains red only if the corresponding CAN controller variant is inactive. In this case, no messages can be received.

RX message A message that is received by a CAN controller. You can use RX messages to observe the CAN communication of your model, or let an RX message trigger the transmission of a TX message, etc.

You can specify RX messages on the RX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference) and specify trigger reactions on the Trigger Reaction Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

RX model signal A signal of an RX message whose value can be analyzed from within the model. Since additional code has to be generated, RX model signals reduce performance. For optimum performance, you should specify as few RX model signals as possible.

You can specify RX model signals on the Model Signals (RX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

RX signal A signal of an RX message. All the signals of an RX message are RX signals and can be analyzed via ControlDesk only. If you want to enable the analysis of RX signals via the model, you have to specify them as RX model signals.

S

SDF file A file automatically generated when you build a real-time application for your model. An SDF file (system description file) bundles all the TRC files and additional information for the application. It describes the files to be loaded to the individual components of a simulation platform. An SDF file built for a model containing blocks from the RTI CAN MultiMessage Blockset references a CAN bus configuration which you can display in the Bus Navigator. Whenever you rebuild the model's real-time application, a new SDF file is generated. When you reload the SDF file in ControlDesk, the Bus Navigator checks the SDF file for changes and adapts the CAN bus configuration.

S-function A computer language description of a Simulink block. S-functions (system functions) are written in a high-level language such as M, C, or Fortran. A dSPACE system supports S-functions written in C.

When you work with the RTI CAN MultiMessage Blockset, you do not have to create S-functions manually. For the RTICANMM ControllerSetup and Gateway block, S-functions are created automatically according to the specified database file when you click **OK** in the block dialogs. For the RTICANMM MainBlock, an S-function is created when you click **Create** in the block dialog.

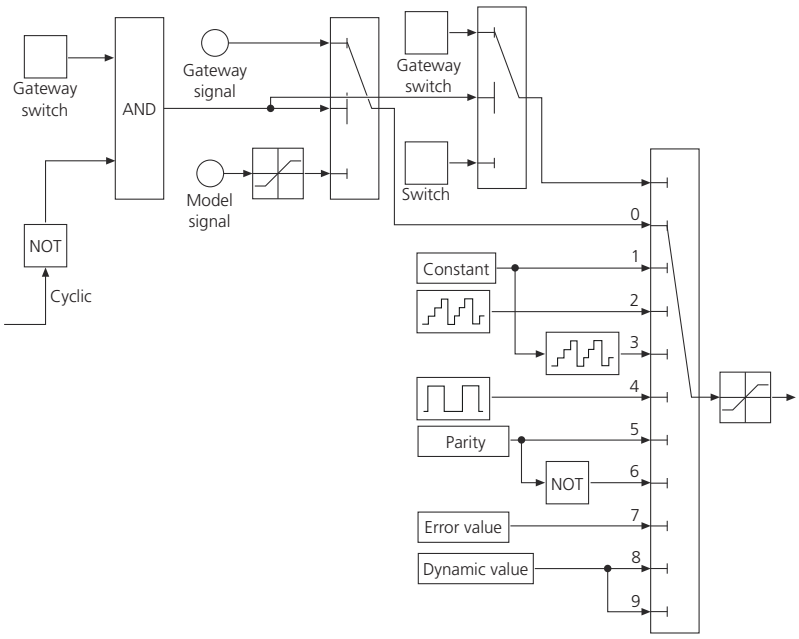
Tip

You can automatically create S-functions for all or just some RTICANMM blocks of your model via the **Options** menu of the RTICANMM GeneralSetup block (refer to Options Menu (RTICANMM GeneralSetup) (RTI CAN MultiMessage Blockset Reference 🕮)).

Signal A data element transmitted by a CAN message. A signal is the number of consecutive bits of raw data that is interpreted according to a data type and the specified scale and offset values. Four basic signal types are available with RTICANMM:

- TX signals
- TX model signals
- RX signals
- RX model signals

Signal manipulation Options that are provided by the RTI CAN MultiMessage Blockset to manipulate the values of signals before they are transmitted. The illustration below visualizes the options.



You can specify the following signals:

- Counter signals
- Dynamic value signals
- Error value signals
- Gateway signals
- Parity signals
- Toggle signals
- TX model signals

Source/destination mapping (J1939) The specification of source (transmitting network node) and destination (receiving network node, only for peer-to-peer) for J1939 messages.

Without source/destination mapping, if two or more network nodes sent the same PGN in the same sampling step, these messages would be placed in a common container message. As a result only the J1939 message received last in that sampling step is accessible via model and/or TRC file. With source/destination mapping, the two J1939 messages can be treated individually as separate instances of the message. Additionally, you can define instances to be shown in the container on the Instance to Container Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮). In this case, the container contains the last received J1939 instances.

Messages in a J1939-compliant DBC file that are described only by the PGN have no source/destination mapping. Messages that are described by the CAN message ID (consisting of the priority, PGN, and source address) have source/destination mapping. The source/destination mapping can be specified in the DBC file or you can specify it on the Source Destination Mapping Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

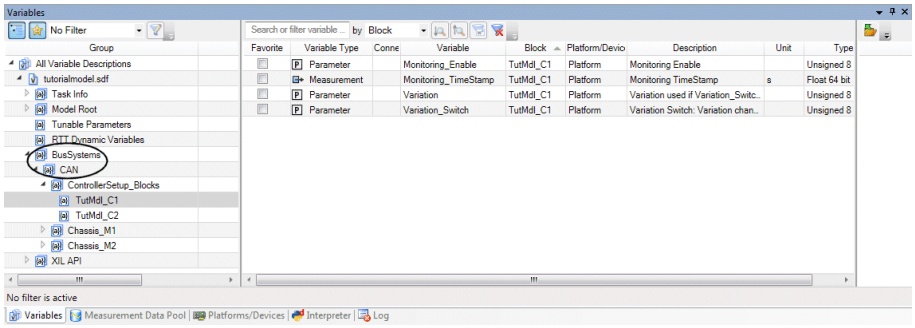
T

Toggle signal A 1-bit signal that can be used, for example, to indicate whether CAN messages are transmitted. If a CAN message is transmitted, the toggle signal value alternates between 0 and 1. Otherwise, the toggle signal value remains constant.

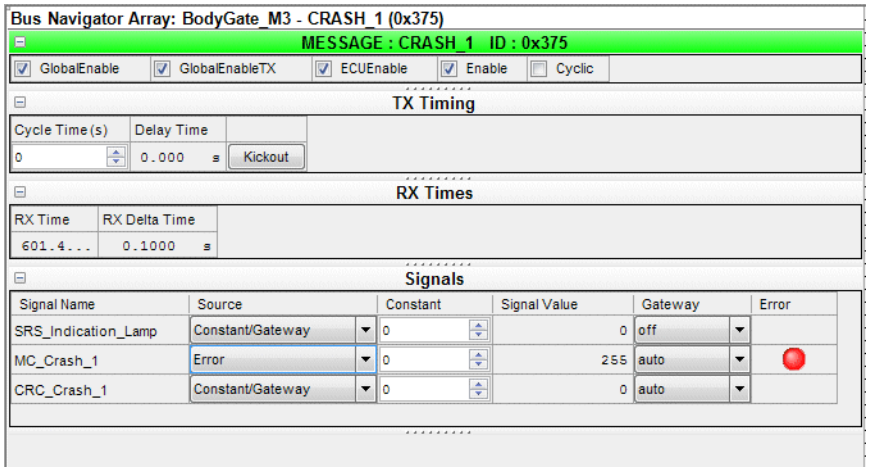
You can specify toggle signals on the Toggle Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮).

TRC file A variable description file for dSPACE real-time hardware. Whenever you create an S-function for an RTICANMM MainBlock, a TRC file is automatically created. The TRC file contains entries for all the CAN signals of the CAN controller concerned. The entries allow you to analyze received signals and change the values of the signals to be transmitted in ControlDesk. The CAN message layouts which can be generated from the Bus Navigator tree are based on the TRC file entries.

The RTICANMM-specific TRC files are included in the model SDF file during the build process. When you open the `<model>.sdf` file in ControlDesk, you will find the entries of the RTI CAN MultiMessage Blockset in the **Variables** controlbar below "BusSystems/CAN".



TX layout A specific layout of the Bus Navigator for displaying and manipulating settings of a TX message.



A TX layout displays all the message-specific information for a TX message specified in the database file and/or in the RTICANMM MainBlock. Via this layout, you can manipulate message settings for which manipulation via ControlDesk is enabled. For most message settings, you can enable manipulation via ControlDesk by selecting TRC, BOTH, or DEFAULT TRC on the corresponding property page in the RTICANMM MainBlock.

A green header indicates that the message can be transmitted. However, if the message is actually transmitted is indicated by a running RX time only.

TX message A message that is transmitted by a CAN controller. You can manipulate TX messages before they are transmitted. You can specify TX messages on the TX Messages Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

TX model signal A signal of a TX message whose value can be changed from within the model. By default, the values of TX model signals cannot be changed in ControlDesk. If you also want to manipulate TX model signals from ControlDesk, you have to select them on the Input Manipulation Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference). Because additional code has to be generated, TX model signals reduce performance. For optimum performance, you should specify as few TX model signals as possible.

You can specify TX model signals on the Model Signals (TX) Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference).

TX signal A signal of a TX message. All the signals of a TX message are TX signals and can be manipulated via ControlDesk only. If you want to enable signal manipulation of TX signals via the model, you have to specify them as TX model signals.

V

Variants of a CAN controller Different CAN communications defined for one CAN controller. You can define up to 20 variants of one CAN controller.

To define variants of a CAN controller, you have to assign two or more RTICANMM MainBlocks to it. For each RTICANMM MainBlock, you have to specify a DBC file and select the variation index on the General Settings Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference). You can realize variants by:

- Specifying different DBC files for the RTICANMM MainBlocks.
- Specifying the same DBC file for the RTICANMM MainBlocks but defining a different CAN communication via the MainBlocks.

Only one variant of a CAN controller can be active at a time. The variant with the variation index 1 is the active variant by default. You can switch between the variants during run time in the Bus Navigator tree.

Appendix

Where to go from here

Information in this section

How to Create an Experiment in ControlDesk.....483
Creating an experiment in ControlDesk is necessary for experimenting with real-time hardware.

Connecting CAN Channels for Working With this Tutorial.....485
To work with this tutorial, the CAN channels of your dSPACE real-time hardware must be interconnected.

How to Create an Experiment in ControlDesk

Objective

In ControlDesk, an experiment is the basis for an experiment task, and a project is a container for experiments. Creating an experiment in ControlDesk is necessary for experimenting with real-time hardware.

In this tutorial, you will create a new experiment for each tutorial lesson.

Precondition

ControlDesk must be started.

Possible methods

You can choose between two methods when creating an experiment:

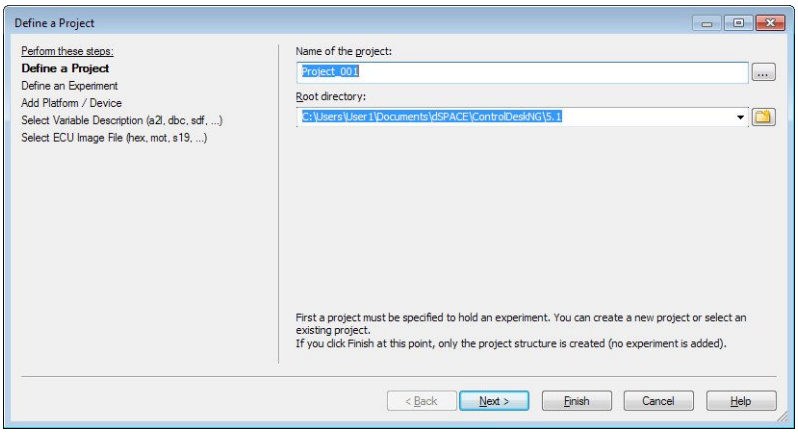
- If you did not yet create a tutorial project in ControlDesk, you must define one first before you can define an experiment in ControlDesk. Refer to Method 1.
- If you already created a tutorial project in ControlDesk, you can open the existing project and create an experiment. Refer to Method 2.

Method 1

To create a project and experiment in ControlDesk

1 On the File ribbon, click New - Create New Project and Experiment.

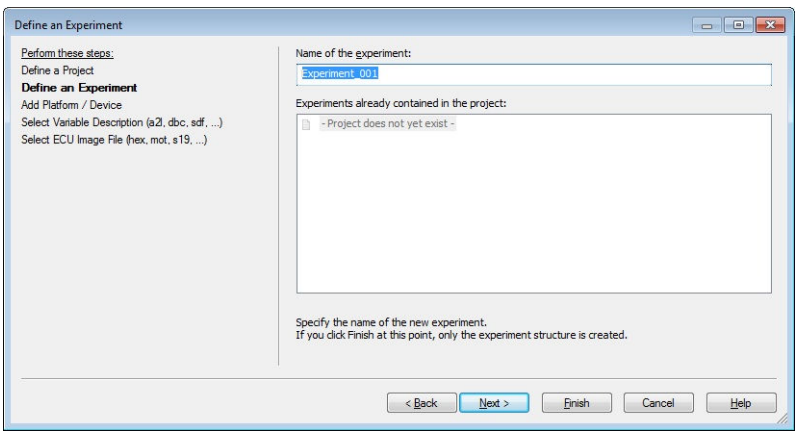
The Define a Project dialog opens.



2 In the Name of the project edit field, specify `RTICANMMTutorialProject` as the name of the project.

3 Click Next.

The Define an Experiment dialog opens.



4 In the Name of the experiment edit field, specify `TutorialExperiment_Lesson<x>` as the name of the experiment. `<x>` denotes the number of the current tutorial lesson.

5 Click Finish to close the dialog.

Result of Method 1

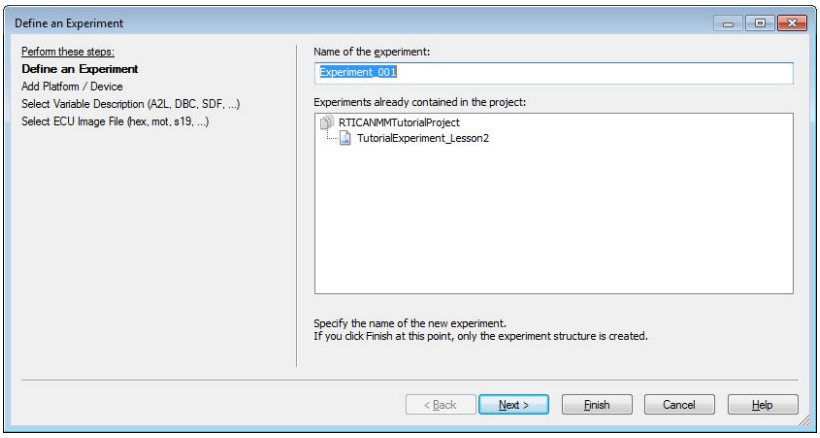
You have created the `RTICANMMTutorialProject` project and an experiment in ControlDesk.

Method 2

To create a further experiment in an open project in ControlDesk

1 On the File ribbon, click New - New Experiment.

The Define an Experiment dialog opens.

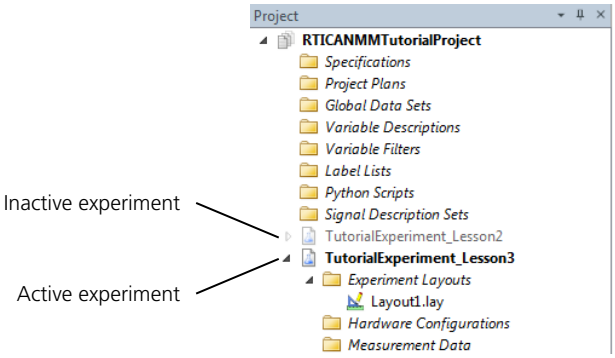


2 In the Define an Experiment dialog, specify `TutorialExperiment_Lesson<x>` as the name of the experiment. `<x>` denotes the number of the current tutorial lesson.

3 Click Finish.

ControlDesk closes the dialog and creates an experiment. The new experiment is active by default.

As an example, the illustration below displays the Project controlbar after creating `TutorialExperiment_Lesson3`.



Result of Method 2

You have created a further experiment within an open project in ControlDesk.

Connecting CAN Channels for Working With this Tutorial

Overview

To work with this tutorial, the CAN channels of your dSPACE real-time hardware must be interconnected as follows:

- Channel 1 is connected to channel 2
- Channel 3 is connected to channel 4 (needed only for lessons 12 and 13)

Mapping information

For the mapping of CAN controller numbers to CAN signals and pins of dSPACE hardware, refer to the topics in the table below:

dSPACE Hardware		Topic with Mapping Information
DS2202		Signal Mapping to I/O Pins (PHS Bus System Hardware Reference 🕮)
DS2210		Signal Mapping to I/O Pins (PHS Bus System Hardware Reference 🕮)
DS2211		Signal Mapping to I/O Pins (PHS Bus System Hardware Reference 🕮)
DS4302		Signal Mapping to I/O Pins (PHS Bus System Hardware Reference 🕮)
DS4505		Signal Mapping to I/O Pins (DS4505/DS4342) (PHS Bus System Hardware Reference 🕮)
MicroAutoBox II	1401/1507	Signal Mapping to I/O Pins (1401/1507) (refer to Interfaces (MicroAutoBox II Hardware Reference 🕮))
	1401/1511	Signal Mapping to I/O Pins (1401/1511) (refer to Interfaces (MicroAutoBox II Hardware Reference 🕮))
	1401/1511/1514	Signal Mapping to I/O Pins (1401/1511/1514) (refer to Interfaces (MicroAutoBox II Hardware Reference 🕮))
	1401/1513	Signal Mapping to I/O Pins (1401/1513) (refer to Interfaces (MicroAutoBox II Hardware Reference 🕮))
	1401/1513/1514	Signal Mapping to I/O Pins (1401/1513/1514) (refer to Interfaces (MicroAutoBox II Hardware Reference 🕮))
MicroLabBox		Signal Mapping to I/O Pins (refer to CAN Connector (Sub-D) (MicroLabBox Hardware Installation and Configuration 🕮))
SCALEXIO	DS2671 Bus Board	Signal Mapping of the DS2671 Bus Board (SCALEXIO Hardware Installation and Configuration 🕮)
	DS2672 Bus Module	Signal Mapping of the DS2672 Bus Module (SCALEXIO Hardware Installation and Configuration 🕮)
	DS6301 CAN/LIN Board	Signal Mapping of the DS6301 CAN/LIN Board (SCALEXIO Hardware Installation and Configuration 🕮)
	DS6341 CAN Board	Signal Mapping of the DS6341 CAN Board (SCALEXIO Hardware Installation and Configuration 🕮)
	DS6342 CAN Board	Signal Mapping of the DS6342 CAN Board (SCALEXIO Hardware Installation and Configuration 🕮)

Related topics

Basics

Overview of Lessons 35

A

accessing the RTI CAN MultiMessage Blockset 21
adapting a model to a new database manually 220
adding a bus hierarchy to the model 120
adding RTICANMM MainBlocks to a model 48
adding RTICANMM setup blocks to a model 41
application for a model
  building 56
  downloading to the real-time hardware 56
AUTOSAR E2E protection profiles 430

B

basics
  E2E protection 400
  end-to-end protection parameters 400
  gatewaying 260
  J1939 316
  M file as the database 373
  restbus simulation 136
  RTI CAN MultiMessage Blockset 14
  RTICANMM MainBlock 22
  variants of a CAN controller 225
bidirectional gateway
  building a bidirectional gateway using two RTICANMM MainBlocks 287
building a bidirectional gateway using two RTICANMM MainBlocks 287
building a model for gatewaying 263
building a model for using a J1939-compliant DBC file 320
building a model for using an M file as the database 377
building a unidirectional gateway using two RTICANMM MainBlocks 273
building an application for a model 56
bus hierarchy
  adding to the model 120

C

CAN communication
  configuring 48
CAN controller
  specifying variants 227
  working with variants in ControlDesk 235
CAN layouts in ControlDesk
  creating 63
  triggering message transmission 68
CAN messages
  monitoring 72
CAN-J1939 communication
  configuring the communication of your model 329
  gatewaying messages using the RTICANMM Gateway block 363
  observing communication in ControlDesk 357
checking message transmission settings in ControlDesk 94
checking restbus simulation in ControlDesk 161
checking the gateway behavior in ControlDesk 304
checksum algorithms based on E2E protection parameters 401
checksum header files
  adding checksum cases 421
checksums
  E2E profiles 434
  specifying 150
Common Program Data folder 10
configuring CAN communication 48
configuring the CAN-J1939 communication of your model 329
connecting a Simulink bus to the Triggering to RTICANMM and Mapping to RTICANMM blocks 130
counter signals
  specifying 145
creating a model 41
creating an experiment in ControlDesk 483
creating an inport for the Mapping to RTICANMM block 127
creating an inport for the Triggering to RTICANMM block 122
creating CAN layouts in ControlDesk 63
customized messages
  working in ControlDesk 181
customized signals
  working in ControlDesk 202

D

database
  adapting the model to a new database manually 220
  updating and checking automatic effects on the model 217
  using capture messages instead of a database 250
  using free raw messages instead of a database 244
database file
  specifying 48
default signal manipulation
  specifying 197
defining CAN communication via an M file 381
demo results
  working with 32
detection of checksums and cycle time misbehavior
  preparing 176
detection of counter signals
  preparing 199
Documents folder 11
downloading an application to the real-time hardware 56
dynamic message manipulation options
  enabling 174
dynamic signal values
  enabling 193

E

E2E profiles 434
E2E protection
  according to AUTOSAR E2E profiles 434
  AUTOSAR E2E profiles 430
  basics 400
E2E protection checksum algorithms 401
E2E protection header files
  adding checksum cases 421
ECU
  simulating real ECU 137
enabling dynamic message manipulation options 174
enabling dynamic signal values 193
enabling ID manipulation of TX messages 171
enabling length manipulation of TX messages 171
enabling manipulation of TX model signals via ControlDesk 109
enabling or disabling message transmission 83
enabling static signal error values 195
end-to-end protection parameters
  basics 400
example used in this tutorial 28
experiment in ControlDesk
  creating 483
extending an E2E protection header file 421

G

gateway
  basics 260
  building a bidirectional gateway using two RTICANMM MainBlocks 287
  building a model for gatewaying 263
  building a unidirectional gateway using two RTICANMM MainBlocks 273
  checking the behavior in ControlDesk 304
  gatewaying J1939 messages using the RTICANMM Gateway block 363
  gatewaying messages using the RTICANMM Gateway block 296
  simulating "real" ECUs 270

I

introducing a typical use scenario 24
introducing the example used in this tutorial 28

J

J1939
  basics 316
J1939 network management
  observing network management in ControlDesk 349
J1939-compliant DBC file
  building a model 320

L

lessons
  overview 35
Local Program Data folder 11

M

M file as the database
  basics 373
  defining CAN communication 381
manipulation of TX messages
  enabling ID manipulation 171
  enabling length manipulation 171
Mapping to RTICANMM block
  connecting a Simulink bus 130
  creating an inport 127
message transmission
  checking settings in ControlDesk 94
  enabling or disabling 83
  specifying cyclic transmission 89
  specifying transmission via kickout 89
  triggering message transmission with CAN layouts 68
model
  creating 41
monitoring CAN messages 72

O

observing CAN-J1939 communication in ControlDesk 357
observing network management in ControlDesk 349
overview of lessons 35

P

preparing the detection of checksum misbehavior 176
preparing the detection of counter signal misbehavior 199
preparing the detection of cycle times misbehavior 176

R

real ECU
  simulating 137
restbus
  simulating 141
restbus simulation
  basics 136
  checking in ControlDesk 161
RTI CAN MultiMessage Blockset
  accessing 21
  basics 14
  supported platforms 16
RTICANMM MainBlock
  basics 22
RTICANMM MainBlocks
  adding to model 48
RTICANMM setup blocks
  adding to a model 41

S

signal manipulation via ControlDesk
  enabling manipulation of TX model signals 109
signals to be manipulated via the model
  specifying 105
simulating "real" ECUs for gatewaying 270
simulating a real ECU 137
simulating a restbus 141
Simulink bus
  adding to the model 120
specifying a database file 48
specifying checksums 150
specifying counter signals 145
specifying cyclic message transmission 89
specifying default signal manipulation 197
specifying message transmission via kickout 89
specifying signals to be manipulated via the model 105
specifying TX model signals 105
specifying variants of a CAN controller 227
static signal error values
  enabling 195

T

triggering message transmission with CAN layouts 68
Triggering to RTICANMM block
  connecting a Simulink bus 130
  creating an inport 122
tutorial
  working with 30
typical use scenario 24

U

unidirectional gateway
  building a unidirectional gateway using two RTICANMM MainBlocks 273
updating the database and checking automatic effects on the model 217
using an M file as the database
  building a model 377
using capture messages instead of a database 250
using free raw messages instead of a database 244

V

variants of a CAN controller
  basics 225

W

working with customized messages in ControlDesk 181
working with customized signals in ControlDesk 202
working with the demo results 32
working with this tutorial 30
working with variants of a CAN controller in ControlDesk 235
working without a database
  using capture messages instead of a database 250
  using free raw messages instead of a database 244