

TargetLink

Multirate Modeling Guide

For TargetLink 5.1

Release 2020-B – November 2020

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.

© 2004 - 2020 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.

The ability of dSPACE TargetLink to generate C code from certain MATLAB code in Simulink®/Stateflow® models is provided subject to a license granted to dSPACE by The MathWorks, Inc. MATLAB, Simulink, and Stateflow are trademarks or registered trademarks of The MathWorks, Inc. in the United States of America or in other countries or both.

Contents

| | |
|---|----|
| About This Guide | 7 |
| Introduction to Multirate Modeling | 11 |
| Overview of Multirate Modeling in TargetLink | 11 |
| Development Steps | 12 |
| Multirate Systems and OSEK Real-Time Operating Systems | 15 |
| Mapping Multirate to Multitasking with TargetLink | 15 |
| The OSEK/VDX Standard | 17 |
| Managing RTOS Data for Multirate Models | 19 |
| Creating Tasks and Interrupt Service Routines | 23 |
| Partitioning Tasks and Interrupt Service Routines | 24 |
| Introduction to Partitioning Tasks and Interrupt Service Routines | 24 |
| Methods of Partitioning Tasks and ISRs | 24 |
| Rules for Creating Multirate Models | 27 |
| Basics of Specifying Task Properties | 32 |
| Creating Interrupt Service Routines (ISRs) | 34 |
| Configuring Names in Multirate Models | 35 |
| Working with Tasks and Interrupt Service Routines | 37 |
| How to Group Atomic Subsystems in Tasks Using the Task Block | 38 |
| How to Specify Properties of User-Defined Tasks | 41 |
| How to Specify Properties of Default Tasks | 42 |
| How to Add a Root Function Template and Specify Its Properties | 44 |
| How to Generate Tasks from Stateflow charts | 46 |
| How to Create Interrupt Service Routines | 47 |
| How to Specify Properties of Interrupt Service Routines | 48 |
| How to Specify External Tasks or ISRs | 50 |
| Characteristics of Special Subsystems | 53 |
| Cyclic Enabled Subsystems | 53 |
| Encapsulated Enabled Subsystems | 54 |
| Action-Port-Triggered Subsystems | 58 |
| Edge-Triggered Subsystems | 59 |

| | |
|---|-----------|
| Several Signals Triggering One Edge-Triggered Subsystem..... | 61 |
| Edge-Triggered Subsystem with Task Block..... | 63 |
| Function-Call-Triggered Subsystems..... | 64 |
| Simulink Behavior for Enabled Subsystems..... | 65 |
| Managing Intertask Communication | 67 |
| Message Basics..... | 68 |
| Using Messages to Ensure Data Integrity..... | 68 |
| Mechanisms for Exchanging Messages..... | 73 |
| Optimizing Intertask Communication..... | 74 |
| Message Properties..... | 74 |
| How to Create a New Message..... | 77 |
| How to Group Signals in a Message..... | 78 |
| How to Specify the Properties of an Existing Message..... | 80 |
| How to Assign Signals to Message Components..... | 83 |
| Detection of Unit Delay and Zero Order Hold Blocks for Rate Transition..... | 85 |
| Support of OSEK-Compliant RTOS | 87 |
| Introduction to TargetLink's OSEK Support..... | 88 |
| Different RTOS Types..... | 88 |
| How to Select the RTOS..... | 90 |
| Modeling OSEK Tasks..... | 91 |
| Basics of Modeling OSEK Tasks..... | 91 |
| Grouping Different Noncyclic (Function Call) Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 92 |
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 102 |
| How to Assign an OSEK Event to a Stateflow chart..... | 104 |
| How to Map Stateflow Input Events to OSEK Events..... | 106 |
| Setting Up Alarms for Cyclic OSEK Task Activation..... | 108 |
| Basics of Setting Up Alarms..... | 108 |
| How to Create an Alarm in the TargetLink Data Dictionary..... | 113 |
| How to Set Up Alarms for Cyclic OSEK Tasks Automatically..... | 115 |
| Importing and Exporting OIL Files..... | 117 |
| OIL File Basics..... | 117 |
| Basics on OIL Files..... | 117 |

| | |
|---|----------------|
| OIL File Import..... | 119 |
| OIL File Export..... | 119 |
| OIL File Import and Export Instructions..... | 121 |
| How to Import an OIL File..... | 121 |
| How to Export an OIL File..... | 123 |
| Working with Special RTOS Blocks | 125 |
| Protecting Critical Sections..... | 126 |
| How to Protect an Atomic Subsystem as a Critical Section..... | 126 |
| How to Create a New Protection Mechanism..... | 128 |
| How to Specify the Properties of a Protection Mechanism..... | 129 |
| Setting up OSEK Alarms..... | 131 |
| Basics of OSEK Counters and Alarms..... | 131 |
| Basics of the CounterAlarm Block..... | 133 |
| Connecting Inports and Outports of the CounterAlarm Block..... | 134 |
| Specifying Properties of Alarms and Counters..... | 135 |
| Properties of Alarms..... | 136 |
| Actions to be Performed when an Alarm Expires..... | 137 |
| Properties of Counters..... | 138 |
| How to Set Up an OSEK Alarm..... | 139 |
| Interrupting a Non-Preemptive Task..... | 146 |
| How to Insert a Schedule Command into a Non-Preemptive Task..... | 146 |
| Simulating Multirate Models and Building Multitasking Applications | 149 |
| Simulating Multirate Models..... | 150 |
| Basics of Simulating Multirate Models..... | 150 |
| Building a Multitasking Application..... | 153 |
| Building a Generic RTOS Application..... | 153 |
| Interaction of Tools Generating C Files..... | 153 |
| Interaction of Production Code and Custom Code..... | 154 |
| How to Build a Generic RTOS Application..... | 157 |
| Building an OSEK Application..... | 158 |
| Interaction of the Tools Generating Files for an OSEK Application..... | 158 |
| Interaction of Different Code Parts..... | 160 |
| How to Build an OSEK Application..... | 163 |

| | |
|----------|-----|
| Glossary | 165 |
| Index | 195 |


About This Guide

Introduction to TargetLink

This guide provides information on the concepts and techniques that you can apply to generate code for multirate systems and explains how to develop OSEK-compliant multitasking applications.









Note

A separate license is required for code generation for the OSEK real-time operating system.

Orientation and Overview Guide For an introduction to the use cases and the TargetLink features that are related to them, refer to the  [TargetLink Orientation and Overview Guide](#).

Symbols

dSPACE user documentation uses the following symbols:

| Symbol | Description |
|---|--|
|  | Indicates a hazardous situation that, if not avoided, will result in death or serious injury. |
|  | Indicates a hazardous situation that, if not avoided, could result in death or serious injury. |
|  | Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury. |
|  | Indicates a hazard that, if not avoided, could result in property damage. |
|  | Indicates important information that you should take into account to avoid malfunctions. |
|  | 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. |
|  | 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.

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 Adobe® 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.

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 Multirate Modeling

Where to go from here

Information in this section

| | |
|---|----|
| Overview of Multirate Modeling in TargetLink..... | 11 |
| Development Steps..... | 12 |

Overview of Multirate Modeling in TargetLink

Purpose

dSPACE's TargetLink is a development environment that allows highly efficient C code generation for microcontrollers in electronic control units (ECUs).

Usually, the code running on an ECU is divided into different functional units that are controlled by a multitasking real-time operating system (RTOS). With TargetLink, you can create multirate models containing blocks with different sample times. When generating production code, TargetLink maps the algorithm of your multirate model to RTOS objects and services.

Based on the multirate models, you can create applications that comply with the OSEK/VDX standard. This results in portability and reusability of the application software.

Audience profile

This guide is most useful to design engineers and software specialists with a basic knowledge of multitasking applications and real-time operating systems. It is assumed that you know the OSEK standard, the [TargetLink Preparation and Simulation Guide](#) and how to implement and influence control models. Knowledge in handling the host PC, the Microsoft Windows operating system and MATLAB is also presupposed.

Development steps

The typical procedure from creating a multirate model to generating production code is reflected in the structure of this guide. Refer to [Development Steps](#) on page 12.

Generic multirate code generation without RTOS

While this guide describes multirate code generation with RTOS, you can also use a generic multirate approach without RTOS objects and services. For details, refer to [Generic Multirate Code Generation](#) ([TargetLink Preparation and Simulation Guide](#)).

Related topics**Basics**

[Development Steps](#)..... 12

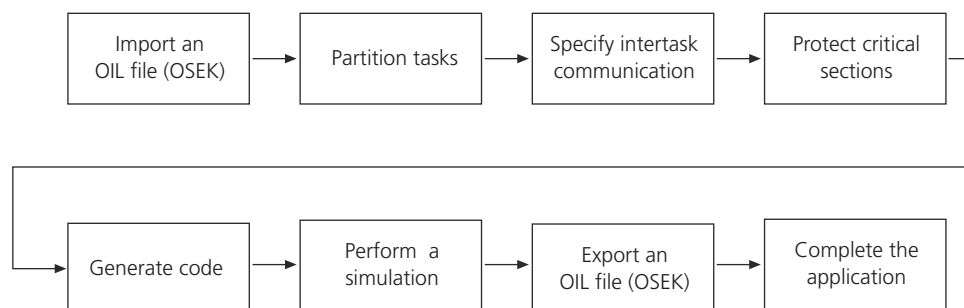
Development Steps

Process of creating multirate models

The process of developing multirate models for multitasking applications is divided into different steps. Basically, there are two typical procedures for creating models for multitasking or OSEK-compliant applications:

- Creating all OS objects such as tasks and ISRs with TargetLink
- Extending an existing multitasking application

The following illustration maps the two procedures of creating multitasking applications. If you want to create all OS objects such as tasks and ISRs with TargetLink, you have to start with the task partitioning of your model. If you want to extend an existing multitasking application, you can assign the functionalities implemented in your model to the existing tasks. In the case of OSEK, you can import tasks and other OS objects with an OIL file.

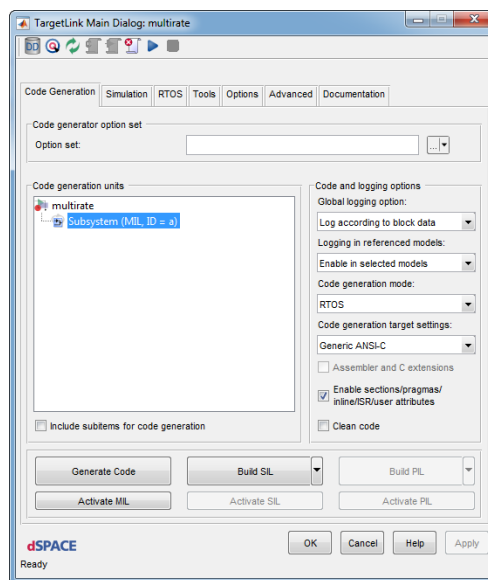


Note

When creating multirate models according to the steps shown above, you must also take into account the general process steps you have to perform when creating standard TargetLink models, for example, selecting data types or scaling. For detailed information, refer to [Getting Started](#) ([TargetLink Orientation and Overview Guide](#)).

Enabling multirate code generation

To enable TargetLink to generate production code from your multirate model, you have to select RTOS from the Code generation mode drop-down list on the Code Generation page of the TargetLink Main Dialog.






For more information, refer to TargetLink Main Dialog Block.

Multirate systems and OSEK

With TargetLink you can create multirate models containing subsystems with different sample times. When generating code, TargetLink maps your multirate model to a multitasking application. If you have purchased the TL_TMOS_OSEK license, you can generate OSEK-compliant code from your model. Additionally, you can adapt the production code to any RTOS that does not comply with the OSEK standard. These RTOSs are called generic RTOSs below.

Tasks and interrupt service routines

Multitasking applications are divided into several tasks performing different jobs simultaneously. TargetLink provides different ways to partition your model into tasks. Interrupt service routines (ISRs) can be regarded as a special kind of tasks, which are connected to a hardware interrupt.

| | |
|------------------------------------|---|
| Intertask communication | The tasks in your model communicate with each other by exchanging signals. Ensuring the signals' data integrity and keeping the data transfer deterministic are important aspects of intertask communication. Refer to Managing Intertask Communication on page 67. |
| OSEK support | If you have purchased a license for using the OSEK support, you can use TargetLink's special OSEK blocks. You are also able to generate the complete OSEK-compliant application. Refer to Support of OSEK-Compliant RTOS on page 87. |
| RTOS blocks | TargetLink's RTOS Blocks library contains special blocks for multirate and OSEK-compliant models. Refer to Working with Special RTOS Blocks on page 125. |
| Simulating multirate models | You should familiarize yourself with some special aspects of simulating multirate models and building multitasking applications with TargetLink. Refer to Simulating Multirate Models and Building Multitasking Applications on page 149. |
| Limitations | Simulink offers great flexibility in creating models. Not all of these models can be used to generate efficient production code. For this reason, there are some limitations applying to TargetLink's multirate feature. Refer to RTOS Multirate Limitations ( TargetLink Limitation Reference). |
| Related topics | <p>Basics</p> <p>Getting Started ( TargetLink Orientation and Overview Guide)</p> <p>Managing Intertask Communication..... 67</p> <p>Simulating Multirate Models and Building Multitasking Applications..... 149</p> <p>Support of OSEK-Compliant RTOS..... 87</p> <p>Working with Special RTOS Blocks..... 125</p> <p>References</p> <p>TargetLink Main Dialog Block ( TargetLink Model Element Reference)</p> |

Multirate Systems and OSEK Real-Time Operating Systems

Generating RTOS-compliant applications

TargetLink allows you to create multirate models containing blocks with different sample times. Based on the multirate models, you can generate applications which comply with real-time operating systems (RTOSs), for example, OSEK/VDX-compliant RTOSs. Compliance with the OSEK/VDX standard supports the portability and reusability of the application software.

Where to go from here

Information in this section

| | |
|--|----|
| Mapping Multirate to Multitasking with TargetLink..... | 15 |
| The OSEK/VDX Standard..... | 17 |
| Managing RTOS Data for Multirate Models..... | 19 |

Mapping Multirate to Multitasking with TargetLink

Multirate models

A Simulink model is constructed of blocks performing certain operations. These blocks are calculated periodically or triggered by events. Multirate models have different periodicities. When a model's functionality is applied on a real electronic control unit (ECU), not only the algorithmic part must be implemented, but also the control of the different functional units. This control is usually done by a multitasking real-time operating system (RTOS).

Operating systems differ in their functionality and their application programming interface (API). However, the OSEK/VDX standard describes a uniform interface. TargetLink directly supports the OSEK standard, for example, it lets you import and export OSEK Implementation Language files (OIL files). For more information

on the OSEK standard and how TargetLink supports it, refer to the following topics:

- [The OSEK/VDX Standard](#) on page 17
- [Support of OSEK-Compliant RTOS](#) on page 87

TargetLink not only supports OSEK, but also lets you adapt the different RTOS objects to operating systems not compliant with this standard. These operating systems are called generic RTOSs below.

Mapping the algorithm to RTOS objects

When generating production code for Simulink models, TargetLink maps the algorithm to RTOS objects and services.

The whole application code must be placed in tasks, which are the basic functional units that are handled by an operating system. A Simulink system initially does not reflect this software structure but is composed of other elements such as subsystems. TargetLink maps this model structure onto RTOS software units.

Partitioning a model into tasks

TargetLink provides different ways to divide a multirate model into different tasks. If you make no specific input, TargetLink uses a default partitioning. Basically, all subsystems that have the same sample time are combined in one task. TargetLink's default task partitioning can be overruled by placing a Task block in a subsystem. The Task block explicitly assigns the subsystem to a certain task. For detailed information, refer to [Creating Tasks and Interrupt Service Routines](#) on page 23.

Intertask communication

Tasks communicate with each other by exchanging data. In the model, the data is represented by signal lines. Usually, the numerous signals that connect one task to another are not independent of each other but can be grouped into units that carry consistent information. In TargetLink, these groups of consistent intertask communication data are called TargetLink messages.

Note

TargetLink messages are not necessarily implemented with the RTOS object message.

The data integrity of these TargetLink messages will be ensured during communication.

Data integrity

Failure to ensure data integrity can lead to problems in preemptive multitasking environments. For example, the receiver of a message could preempt the sender while updating the different signals of the message. In such a case, the message would contain a mix of 'old' and 'new' signals (from different simulation steps). TargetLink supports different mechanisms to protect messages from being inconsistent.

For OSEK, you can select between several protection methods that are provided by the standard, like OSEK messages, resource locks, disabling interrupts etc. For generic RTOSs, you can adapt a general protection [🔗](#) [macro](#) to the services your operating system provides.

Related topics

Basics

| | |
|--|----|
| Creating Tasks and Interrupt Service Routines..... | 23 |
| Managing Intertask Communication..... | 67 |
| Support of OSEK-Compliant RTOS..... | 87 |
| The OSEK/VDX Standard..... | 17 |

The OSEK/VDX Standard

RTOS standard

OSEK/VDX, called OSEK below, specifies a standard for modular and static real-time operating systems which are used mainly in automotive electronic control units. It stands for "Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug/Vehicle distributed executive". The intention of OSEK is to provide standardized interfaces, and the scalability and portability of application software.

The OSEK standard consists of several components, for example:

- The operating system (OSEK-OS)
- The OSEK Implementation Language (OSEK OIL)
- The communication subsystem (OSEK-COM)
- The network management system (OSEK-NM)

The OSEK-OS properties and the part of the OSEK-COM that is relevant to code generation from Simulink models are described below.

The most important objects of the OSEK-OS are tasks, events, interrupts, resources and alarms, which are described below. In addition, you will find information about the scalability of OSEK-compliant applications and the OSEK Implementation Language (OIL).

Tasks

Like in other operating systems, an OSEK-compliant application contains several tasks which process different jobs. The properties of OSEK tasks have an impact on the real-time operation of a system and the production [🔗](#) [code size](#). Typically, an OSEK task has the following properties:

- It is either basic or extended. Basic tasks cannot wait for events. Extended tasks can wait for one or more events. Usually, extended tasks are started only once and are controlled by events.
- It has a fixed priority.
- It is preemptive or not.

Events

Events in an OSEK-compliant application are used for synchronization between tasks. Another aspect of events that is especially important for TargetLink is that they can be used to pass information to a task about the source that has initiated task activation. An OSEK event is used to send binary information from one task to another.

Interrupts

OSEK provides the following categories for interrupts:

- A category 1 interrupt does not interact with the operating system.
- A category 2 interrupt can use operating system services. This results in a longer response time (latency) because the interrupt service routine needs time for the operating system interaction.

TargetLink supports both category 1 and 2.

Using alarms and counters

OSEK provides the objects counter and alarm to handle recurring events. For example, such events may be periodic hardware timer ticks or encoder interrupts that represent a certain camshaft angle. Counters are used to count the events and alarms trigger certain actions like task activations at defined counter values.

Resources

Like common operating systems, OSEK provides means of mutual exclusion of accesses to memory areas, devices etc. that are used by several tasks. The OSEK objects for this purpose are called resources. Resources can be locked and released (similar to semaphores in other operating systems).

Scalability

Extensive applications require many RTOS functionalities and thus need extensive memory and an efficient processor. In contrast to that, small applications have only a cheap (small) processor and not much memory. They do not need such extensive RTOS features, but the RTOS itself must not consume much memory or processor time. The OSEK standard provides four conformance classes which allow you to scale your operating system and thus adapt it to your processor. The conformance classes and their characteristics are explained below:

- Basic Conformance Class 1 (BCC1) allows only basic tasks which are limited to one task per priority and one activation per task.
- Basic Conformance Class 2 (BCC2) provides BCC1 functionality, but more than one task per priority and multiple task activations are allowed.
- Extended Conformance Class 1 (ECC1) provides BCC1 functionality plus extended tasks which can wait for events.
- Extended Conformance Class 2 (ECC2) provides BCC2 functionality plus extended tasks.

Multiple task activations are allowed only for basic tasks.

OSEK messages

The OSEK object used for [intertask communication](#) is the OSEK message. The data stored in an OSEK message can be exchanged between tasks using specific OSEK API functions.

Managing RTOS Data for Multirate Models

TargetLink Data Dictionary basics

The TargetLink Data Dictionary is a container where you can maintain variables, data type definitions, scalings, and further information required for ECU applications. Data Dictionary objects (DD objects) can be referenced by any block in a Simulink model independently of the model partitioning, so the algorithm and its data are clearly separated.

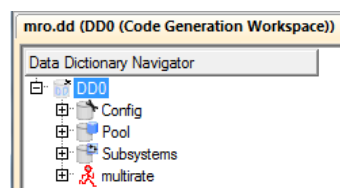
The TargetLink Data Dictionary allows the exchange of data between different development tools and consistent data management throughout all phases of the development process.

The TargetLink Data Dictionary holds a great wealth of additional information, for example, specifics about function calls, tasks, code generation options and RTOS data, to name only a few.

Structure of the TargetLink Data Dictionary

The Data Dictionary object tree (DD object tree) represents the structure of the TargetLink Data Dictionary.

The TargetLink Data Dictionary is basically divided into four main objects (after code has been generated).





Expanding these subtrees displays objects which contain data structured according to its purpose, origin and kind.

Data Dictionary Manager

The TargetLink Data Dictionary Manager is the central component of the TargetLink Data Dictionary (DD). It consists of a graphical user interface that displays DD objects in a tree structure. In addition, it offers you an easy way to handle DD objects and their properties.

Note

- Depending on the selected view mode of the TargetLink Data Dictionary Manager, parts of the DD object tree might not be shown. For further information on view modes, refer to [Basics on Filter Rule Sets for the Data Model](#) ( [TargetLink Data Dictionary Basic Concepts Guide](#)).
- Depending on the selected user mode of the TargetLink Data Dictionary Manager, you might not be allowed to access all objects and objects of the DD object tree. For further information on user / administrator mode, refer to [How to Switch User Modes](#) ( [TargetLink Data Dictionary Basic Concepts Guide](#)).

RTOS data in the TargetLink Data Dictionary

During the process of developing multirate models and generating production code for multitasking applications, RTOS data is read from and written to the following objects in the DD object tree.

- /Poo1/RTOS
- /Subsystems/RTOS

In the subsequent steps of the development process, TargetLink reads this information to build multitasking applications.

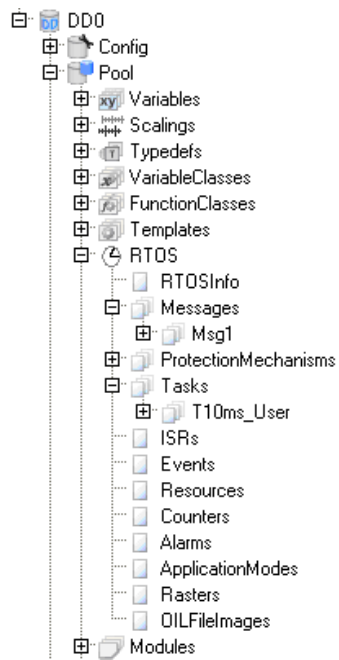
Where RTOS data is written to and read from is described below.

Pool object

The /Poo1/RTOS object contains RTOS-relevant data.

RTOS objects that you create manually in the block dialogs and RTOS data that you import from OIL files are stored in this subtree. When you develop multirate models, special TargetLink RTOS blocks in your model, such as task blocks, reference these objects.

The Code Generator subsequently uses the information from the `/Pool1/RTOS` object for generating code for multitasking applications.

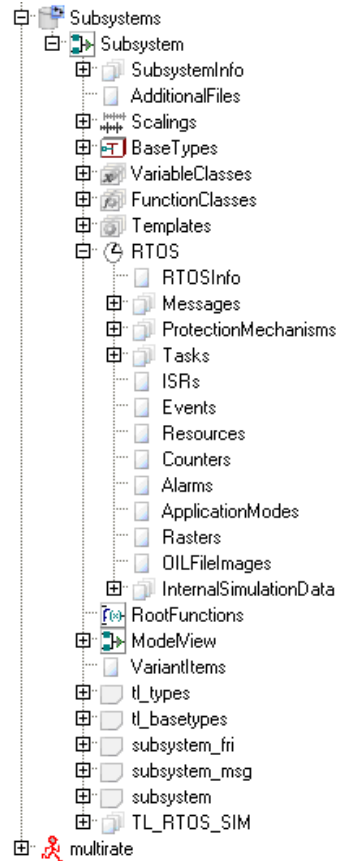


The `/Pool1/RTOS` object contains data and information, for example, on:

- Tasks
- ISRs
- Messages

Subsystems object

The code generation process creates a `<subsystem>` object under the **Subsystems** object of the DD object tree for each TargetLink subsystem in the model.



The `/Subsystems/<subsystem>/RTOS` subtree holds the RTOS data which the Code Generator generates for the respective subsystem. This includes data and information on:

- Tasks
- ISRs
- Messages

The **Subsystem** objects only contains information about RTOS objects that are used in the code of the respective subsystem.

Related topics

HowTos

[How to Switch User Modes \(TargetLink Data Dictionary Basic Concepts Guide\)](#)

Creating Tasks and Interrupt Service Routines

Generating multitasking applications

Tasks and interrupt service routines (ISRs) are parts of a multitasking application, as it is implemented. A multirate Simulink model, on the other hand, consists of certain functional units, usually represented by subsystems. If you want to generate a multitasking application from your multirate model, the model's subsystems must be mapped to tasks and ISRs.

Partitioning Tasks and Interrupt Service Routines

Task partitioning

In a multitasking application, tasks are the basic functional units which are handled by an operating system. A Simulink model does not consist of tasks, but of other units like subsystems. When generating production code, TargetLink maps the model's subsystems onto different tasks.

Where to go from here

Information in this section

| | |
|--|----|
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Working with Tasks and Interrupt Service Routines..... | 37 |
| Characteristics of Special Subsystems..... | 53 |

Introduction to Partitioning Tasks and Interrupt Service Routines

Where to go from here

Information in this section

| | |
|---|----|
| Methods of Partitioning Tasks and ISRs..... | 24 |
| Rules for Creating Multirate Models..... | 27 |
| Basics of Specifying Task Properties..... | 32 |
| Creating Interrupt Service Routines (ISRs)..... | 34 |
| Configuring Names in Multirate Models..... | 35 |

Methods of Partitioning Tasks and ISRs

Task partitioning methods

TargetLink provides two methods to partition a multirate model into different tasks:

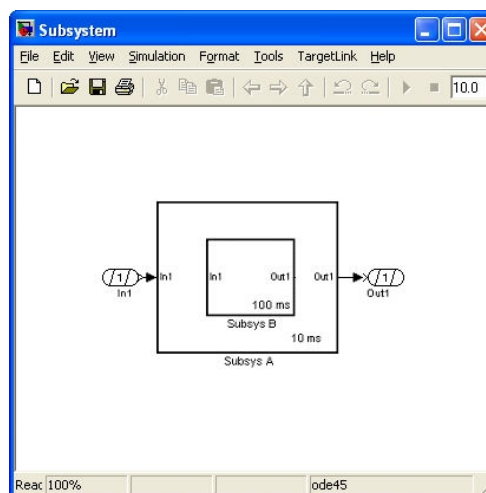
- Default task partitioning
- Explicit task partitioning using the Task block

Default task partitioning

To give you an overview of how TargetLink partitions multirate models, the process is described on the basis of tasks. For a more detailed explanation, refer to [Basics of Specifying Task Properties](#) on page 32. Tasks created by default task partitioning are called default tasks below. If you make no specific input, TargetLink uses the default task partitioning to divide the model into tasks in the following way:

- TargetLink generates a task for each sample time in your model. The code units generated for each subsystem that is executed cyclically with a specific sample time (called cyclic subsystem below) are put into the corresponding task.
- Suppose a cyclic subsystem is embedded in another cyclic subsystem. The code unit of the embedded subsystem is put into the task with the sample time of the surrounding subsystem if the embedded subsystem's sample time is an integer multiple of the surrounding subsystem's sample time. Thus, possible data flow between the two subsystems is not interrupted.

The following example shows a subsystem embedded in another subsystem and the resulting code:

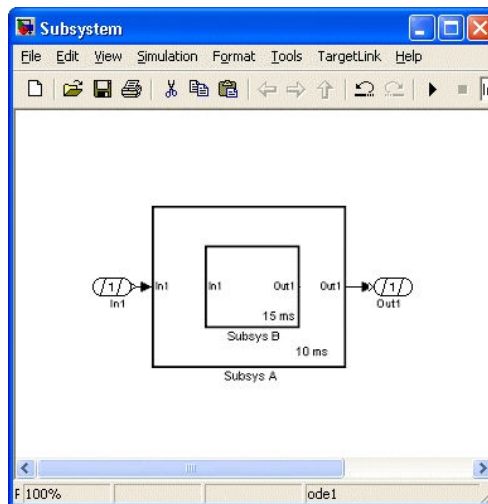


The example model results in the following code:

```
UInt8 Cnt = 1;
Task_10ms()
{
    SubsysA();
    if (--Cnt == 0) {
        SubsysB();
        Cnt = 10;
    }
}
```

- If the embedded subsystem's sample time is not an integer multiple of the surrounding subsystem, TargetLink puts the code unit of the embedded subsystem into the task with the corresponding sample time.

The following example shows two nested subsystems where the inner subsystem's sample time is not an integer multiple of the outer subsystem's sample time.



When generating production code, TargetLink puts the code units for the subsystems into separate tasks:

```
Task_10ms()
{
    SubsystemA();
}
```

```
Task_15ms()
{
    SubsystemB();
}
```

- Each function-call subsystem, that is triggered from outside the TargetLink subsystem (called external function-call-triggered subsystem below) becomes a separate task.

Note

If the code unit of the embedded subsystem is not put into the task of the surrounding subsystem, the Simulink behavior during simulation may differ from the behavior of the generated code because of the different data flow.

Partitioning tasks using the Task block

You can overrule TargetLink's default task partitioning with the Task block. The Task block lets you group practically any kinds of subsystems, for example, subsystems with different sample times. In this case, TargetLink calculates the greatest common divisor of the different sample times for the resulting task to allow the counter mechanism described above. Thus, it executes each subsystem with its specific sample time. Tasks explicitly created by the Task block are called user-defined tasks below.

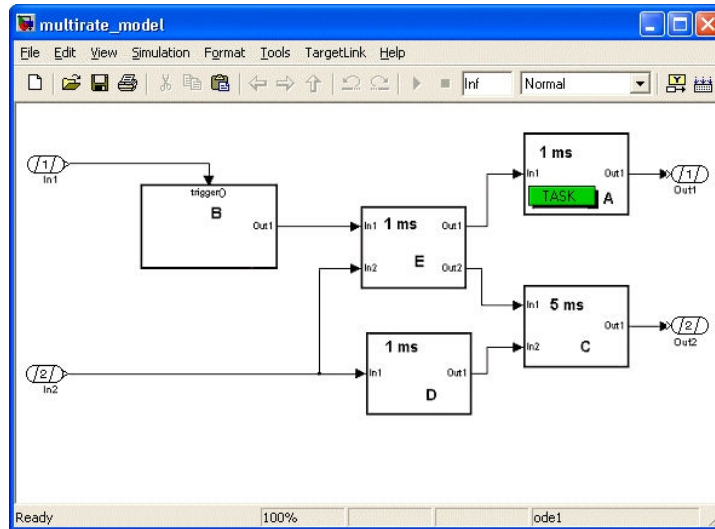
Note

Grouping cyclic and noncyclic subsystems or different noncyclic subsystems in one task is possible only for OSEK-compliant operating systems. Refer to [Modeling OSEK Tasks](#) on page 91.

The rules for task partitioning apply to all levels of an arbitrarily nested system.

Example

Consider the following example model:



When generating production code, TargetLink generates a user-defined task for subsystem A. Subsystems E and D are grouped in one separate default task. Subsystems B and C each become a separate default task.

Related topics**Basics**

| | |
|---|----|
| Basics of Specifying Task Properties..... | 32 |
| Modeling OSEK Tasks..... | 91 |

Rules for Creating Multirate Models

Transferring multirate models to multitasking applications

To transfer a multirate model to a multitasking application, the model must be partitioned into several tasks and interrupt service routines (ISRs).

TargetLink generates a contiguous code unit for each atomic subsystem in your multirate model and puts these code units into tasks. As a precondition, you must take some rules into account when assigning sample times to the atomic subsystems.

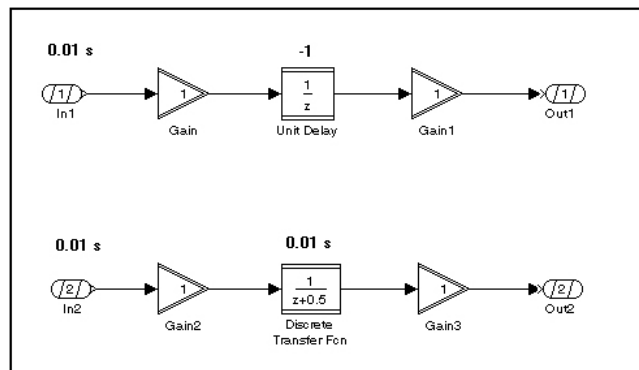
Some kinds of subsystems, such as edge-triggered or action-port-triggered subsystems, have specific behavior in multirate models. For more information on the characteristics of these subsystems, refer to [Characteristics of Special Subsystems](#) on page 53.

Preconditions for task partitioning

As a precondition for having TargetLink partition your model into different tasks, you have to observe the following rules when creating a multirate model:

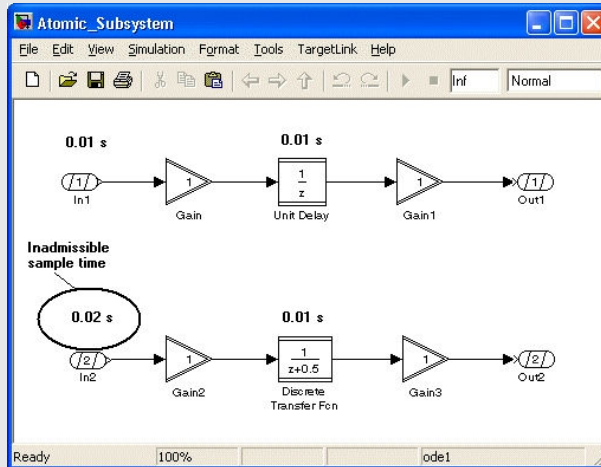
- TargetLink places the code units for atomic subsystems in tasks. TargetLink handles virtual subsystems containing a Task block or a Function block like atomic subsystems.
- All blocks inside one atomic subsystem you can assign a sample time to (called time-specifying blocks below) must have the same sample time.
- A sample time of -1 is allowed. In this case, the compiled sample time of the time-specifying block must be equal to the other specified sample times.

The following example model shows an atomic subsystem with a permissible sample time assignment. All time-specifying blocks within the atomic subsystem have the same sample time:



Note

You are not allowed to use different sample times within an atomic subsystem, as shown in the model below.

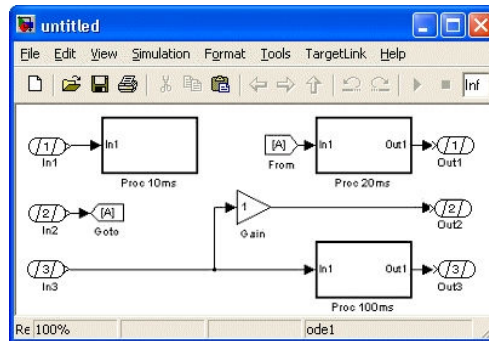


If you do so, the Code Generator outputs an error message.

The following blocks are an exception to the rule:

- Simulink Inport blocks that are not connected to real blocks (real blocks are blocks TargetLink generates code for) and thus do not propagate their sample times to real blocks inside an atomic subsystem.

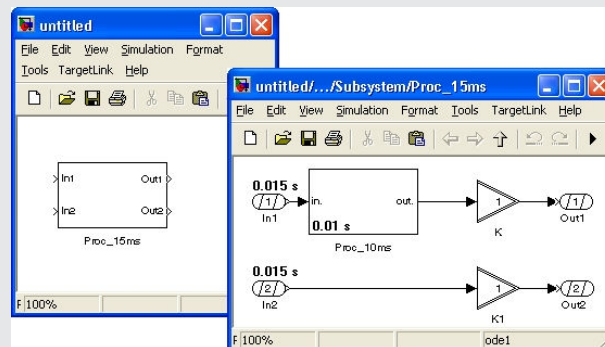
In the example model below, In1 and In2 are such exceptions because they are not connected to any real block in this atomic subsystem. However, In3 is connected to the real Gain block and thus is not an exception.



- The sample time of a subsystem is the same as the sample time you assign to the time-specifying blocks within the subsystem.

Note

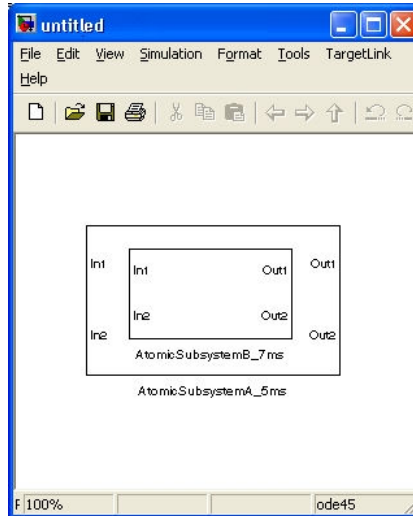
The sample time of a subsystem that is defined in that way can differ from Simulink's compiled sample time. In the following model, the compiled sample time of Proc_15ms is 5 ms because Simulink always uses the greatest common divisor of all containing subsystems and blocks.



In Simulink models, not only real blocks but also the encapsulated subsystems specify the compiled sample time of the surrounding subsystem.

- Different atomic subsystems are allowed to have different sample times. This is also valid for subsystems residing in another atomic subsystem.

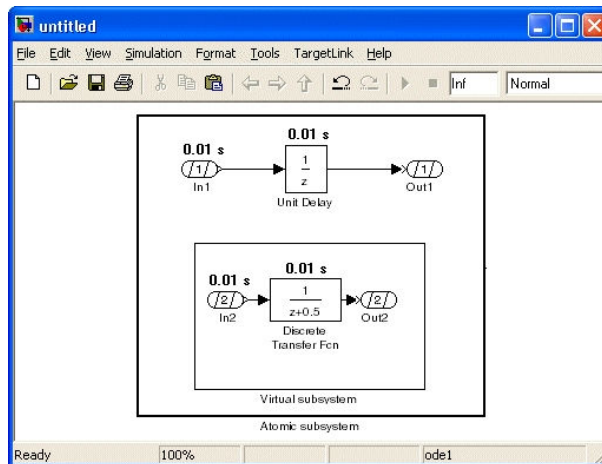
The following example model shows two nested atomic subsystems:



In AtomicSubsystemA_5ms, all time-specifying blocks must have a sample time of 5 ms. In AtomicSubsystemB_7ms, all time-specifying blocks must have a sample time of 7 ms.

The time-specifying blocks of a virtual subsystem residing in an atomic subsystem must have the same sample time as the time-specifying blocks of the atomic subsystem.

In the following example model, the time-specifying blocks of the virtual subsystem must have the same sample time as the time-specifying blocks of the atomic subsystem.



- TargetLink handles Stateflow charts as atomic subsystems.

Related topics

Basics

Characteristics of Special Subsystems..... 53


Basics of Specifying Task Properties

Task properties impact the program flow

The properties of tasks resulting from a multirate model have an impact on the program flow of the production code. TargetLink lets you specify all properties of default tasks and of user-defined tasks. If you make no specific input, TargetLink assigns default values to all default tasks. The tasks and their properties are stored in the TargetLink Data Dictionary.

Note

In TargetLink the names of tasks, ISRs, etc., in the generated production code are set with the corresponding block's dialog or the Data Dictionary Manager. You can define the names manually. The names of default tasks can be generated automatically by TargetLink using name macros. Name macros begin with a \$ character, followed by a letter or several letters in parentheses. For detailed information, refer to the following topics:

- [Configuring Names in Multirate Models](#) on page 35
- [Generating Unique Names via Name Macros](#) ( [TargetLink Customization and Optimization Guide](#))

Root step functions

The partitioning of the production code into different software units is shown on the basis of tasks above. Tasks have a defined shape that depends on the operating system used. This shape is called a task frame below.

Example

The following example shows a task frame for OSEK operating systems:

```
TASK(Taskname)
{
    Code
}
```

In some cases, you may want TargetLink to generate a pure C function instead of a task frame. The software units that TargetLink generates according to the above rules are actually called root step functions in generalization. These become tasks with a task frame, or remain common C functions. Root step functions are the interface between the production code and its environment. They must be called by the operating system or user-defined code.

Step functions

In contrast to root step functions, step functions are called within the production code. They are not interfaces to the environment.

Note

Step functions are clearly assigned to a specific time unit, for example, a sample rate or a trigger event. Utility functions like look-up table functions or filter functions are not step functions.

In an extreme case, the following hierarchy of function calls is possible:

Task → root step function → step function

Using templates in the TargetLink Data Dictionary

Default root step functions can become tasks or C functions, depending on the settings in the TargetLink Data Dictionary. The TargetLink Data Dictionary provides templates to filter out root step functions and specify their properties. The following default templates are delivered with the TargetLink Data Dictionary:

- A template applying to all root step functions generated for cyclic subsystems
- A template applying to all root step functions generated for noncyclic subsystems

You can add your own templates to filter out specific default root step functions, for example, by their sample times or names. If a root step function template contains a subordinate task template, the default root step function filtered out by the root step function template becomes a task. Otherwise, it becomes a C function. Since tasks are significant objects in a multitasking application, this document focuses on tasks in the following descriptions.

You can use the templates to specify the properties of default tasks. The properties are listed in the property value list of the TargetLink Data Dictionary. You can edit the properties stored in the TargetLink Data Dictionary with the Data Dictionary Manager. Refer to [How to Add a Root Function Template and Specify Its Properties](#) on page 44.

Note

Since TargetLink creates the tasks during production code generation, you have to specify the properties before generating production code.

Specifying properties of user-defined tasks

You can specify the properties of tasks marked with a Task block in the Task block dialog. The Task block dialog lets you specify all properties found in the TargetLink Data Dictionary.

Related topics

Basics

| | |
|---|----|
| Configuring Names in Multirate Models..... | 35 |
| Generating Unique Names via Name Macros (📖 TargetLink Customization and Optimization Guide) | |
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |

HowTos

| | |
|---|----|
| How to Add a Root Function Template and Specify Its Properties..... | 44 |
| How to Specify Properties of Default Tasks..... | 42 |
| How to Specify Properties of Interrupt Service Routines..... | 48 |
| How to Specify Properties of User-Defined Tasks..... | 41 |

References

| |
|---|
| Task Block (📖 TargetLink Model Element Reference) |
|---|

Creating Interrupt Service Routines (ISRs)

ISR basics

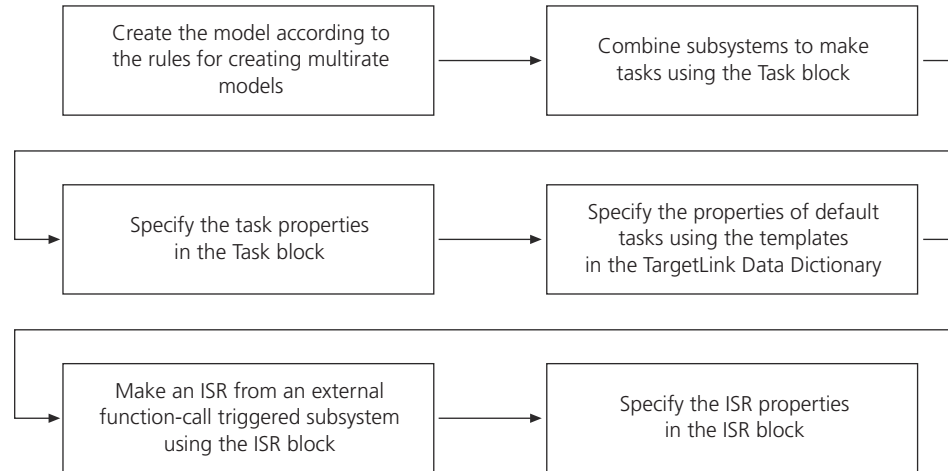
ISRs are functions that are not called by the operating system but by the processor hardware. Usually, there are different interrupt sources, for example, serial interfaces or special interrupt inputs of the processor. If an interrupt occurs, the corresponding ISR is called by the hardware. ISRs can also call operating system services and thus are also defined for OSEK. You can generate an ISR for an external function-call-triggered subsystem by placing an ISR block in it. It is not allowed to group several subsystems in the same ISR. Cyclic subsystems cannot become an ISR.

The following points apply to ISRs:

- ISRs have different properties than tasks.
- There are no default ISRs.

Typical workflow of partitioning tasks and ISRs

The following illustration shows the typical workflow of creating a multirate model:



When you generate production code, TargetLink partitions the multirate model into tasks and ISRs with the specified properties.

Related topics**Basics**

| | |
|--|--------------------|
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Methods of Partitioning Tasks and ISRs..... | 24 |

HowTos

| | |
|--|--------------------|
| How to Create Interrupt Service Routines..... | 47 |
| How to Specify Properties of Interrupt Service Routines..... | 48 |

References

[ISR Block \(📖 TargetLink Model Element Reference\)](#)

Configuring Names in Multirate Models

Name macros

In the generated production code, names of tasks, ISRs etc. in multirate models are either set with the corresponding block's dialog or the Data Dictionary Manager. You can define the names manually or you can let TargetLink generate them automatically by using name macros. Name macros begin with a \$ character, followed by a letter or several letters in parenthesis.


During code generation, the name macros are replaced by the strings they represent:


| Macro | Replaced by | Example |
|-----------------|---|--|
| \$(Ctr) | Counter name | Expands to Counter1 for a counter named Counter1 |
| \$(Event) | Event name | Expands to Event1 for an event named Event1 |
| \$(ISR) | Interrupt service routine (ISR) name | Expands to ISR1 for an ISR named ISR1 |
| \$(Msg) | Message name | Expands to Message1 for a message named Message1 |
| \$(MsgVar) | Message variable ID | Expands to 1 for the first message variable in a TargetLink subsystem |
| \$(Offset) | Offset in a variable unit. The highest time base that allows the offset to be represented as an integer value is used. The possible units are us, ms, s, m, h, d. | Expands to 2ms for an offset of 2 milliseconds |
| \$(Offset100ms) | Offset in 100 milliseconds | Expands to 2 for an offset of 0.2 sec |
| \$(Offset100us) | Offset in 100 microseconds | Expands to 2 for an offset of 0.0002 sec |
| \$(Offset10ms) | Offset in 10 milliseconds | Expands to 2 for an offset of 0.02 sec |
| \$(Offset10us) | Offset in 10 microseconds | Expands to 2 for an offset of 0.00002 sec |
| \$(Offset1d) | Offset in days | Expands to 2 for an offset of 172800 sec |
| \$(Offset1h) | Offset in hours | Expands to 2 for an offset of 7200 sec |
| \$(Offset1m) | Offset in minutes | Expands to 2 for an offset of 120 sec |
| \$(Offset1ms) | Offset in milliseconds | Expands to 2 for an offset of 0.002 sec |
| \$(Offset1s) | Offset in seconds | Expands to 2 for an offset of 2 sec |
| \$(Offset1us) | Offset in microseconds | Expands to 2 for an offset of 0.000002 sec |
| \$(Period) | Sample time in a variable unit. The highest time base that allows the sample time to be represented as an integer value is used. The possible units are us, ms, s, m, h, d. | Expands to 2ms for a sample time of 2 milliseconds |
| \$(Period100ms) | Sample time in 100 milliseconds | Expands to 2 for a sample time of 0.2 sec |
| \$(Period100us) | Sample time in 100 microseconds | Expands to 2 for a sample time of 0.0002 sec |
| \$(Period10ms) | Sample time in 10 milliseconds | Expands to 2 for a sample time of 0.02 sec |
| \$(Period10us) | Sample time in 10 microseconds | Expands to 2 for a sample time of 0.00002 sec |
| \$(Period1d) | Sample time in days | Expands to 2 for a sample time of 172800 sec |
| \$(Period1h) | Sample time in hours | Expands to 2 for a sample time of 7200 sec |
| \$(Period1m) | Sample time in minutes | Expands to 2 for a sample time of 120 sec |
| \$(Period1ms) | Sample time in milliseconds | Expands to 2 for a sample time of 0.002 sec |
| \$(Period1s) | Sample time in seconds | Expands to 2 for a sample time of 2 sec |
| \$(Period1us) | Sample time in microseconds | Expands to 2 for a sample time of 0.000002 sec |
| \$(PObj) | Name of the protected object. If this macro is used in combination with a message, it expands to the message name. Otherwise, it expands to the Simulink subsystem name. | Expands to SubsystemA for a protected object named SubsystemA. |
| \$(Task) | Task name. If this macro is used in an ISR, it expands to the ISR name and the Code Generator issues a message. | Expands to PosControl1 for a task named PosControl |

| Macro | Replaced by | Example |
|---------------|---|---|
| \$(Time) | Sample time and offset in a variable unit. The highest time base that allows the sample time and the offset to be represented as an integer value is used. The possible units are us, ms, s, m, h, d. The offset is placed in only if its value is unequal 0. | Expands to <code>2ms_3ms</code> for a sample time of 2 milliseconds and an offset of 3 milliseconds |
| \$(Time100ms) | Sample time and offset in 100 milliseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.2 sec and an offset of 0.3 sec |
| \$(Time100us) | Sample time and offset in 100 microseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.0002 sec and an offset of 0.0003 sec |
| \$(Time10ms) | Sample time and offset in 10 milliseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.02 sec and an offset of 0.03 sec |
| \$(Time10us) | Sample time and offset in 10 microseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.00002 sec and an offset of 0.00003 sec |
| \$(Time1d) | Sample time and offset in days. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 172800 sec and an offset of 259200 sec |
| \$(Time1h) | Sample time and offset in hours. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 7200 sec and an offset of 10800 sec |
| \$(Time1m) | Sample time and offset in minutes. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 120 sec and an offset of 180 sec |
| \$(Time1ms) | Sample time and offset in milliseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.002 sec and an offset of 0.003 sec |
| \$(Time1s) | Sample time and offset in seconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 2 sec and an offset of 3 sec |
| \$(Time1us) | Sample time and offset in microseconds. The offset is placed in only if its value is unequal 0. | Expands to <code>2_3</code> for a sample time of 0.000002 sec and an offset of 0.000003 sec |

Related topics

Basics

[Basics on Using Name Macros](#) ( [TargetLink Customization and Optimization Guide](#))

[Details on the Expansion of Name Macros](#) ( [TargetLink Customization and Optimization Guide](#))

References

[Block-Related Name Macros](#) ( [TargetLink Model Element Reference](#))

[Stateflow-Related Name Macros](#) ( [TargetLink Model Element Reference](#))

Working with Tasks and Interrupt Service Routines

Objective

You should be familiar with the different methods of creating tasks or interrupt service routines and of specifying their properties.

Where to go from here

Information in this section

| | |
|---|----|
| How to Group Atomic Subsystems in Tasks Using the Task Block..... | 38 |
| How to Specify Properties of User-Defined Tasks..... | 41 |
| How to Specify Properties of Default Tasks..... | 42 |
| How to Add a Root Function Template and Specify Its Properties..... | 44 |
| How to Generate Tasks from Stateflow charts..... | 46 |
| How to Create Interrupt Service Routines..... | 47 |
| How to Specify Properties of Interrupt Service Routines..... | 48 |
| How to Specify External Tasks or ISRs..... | 50 |

How to Group Atomic Subsystems in Tasks Using the Task Block

Objective

If you want to group specific atomic subsystems in tasks, you have to use the Task block.

Combining specific subsystems in tasks

When grouping specific atomic subsystems in tasks using the Task block, you overrule TargetLink's default partitioning. For basic information on combining subsystems with the Task block, refer to [Introduction to Partitioning Tasks and Interrupt Service Routines](#) on page 24.

Precondition

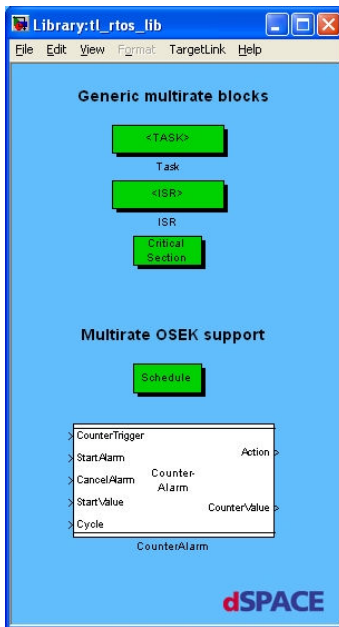
You must have opened a TargetLink model.

Method

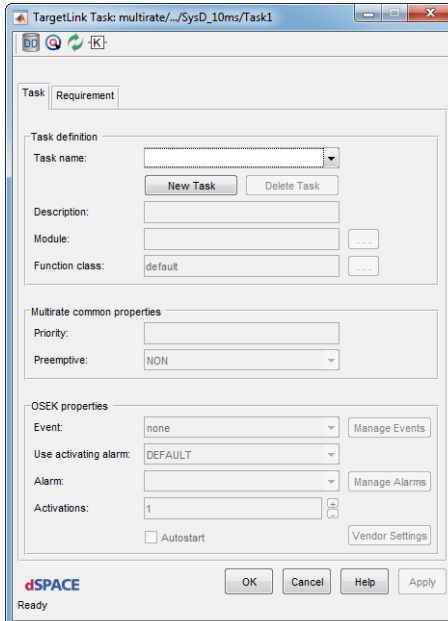
To group subsystems in tasks using the Task block

- 1 In your model, open a subsystem you want to assign to a task.

- 2 In the TargetLink Block Library, double-click the RTOS Blocks sublibrary to open it.



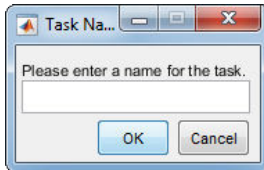
- 3 Copy one Task block to the subsystem and double-click it to open the Task dialog.



Note

You can select a task from the Task Name list instead of creating a new task. In this case, continue with step 6.

- 4 Click New Task to create a new task.
The Task Name dialog opens.



- 5 Enter a name for the task, for example, **PositionControl**, and click OK to close the dialog.
TargetLink creates a new task with the name PositionControl in the TargetLink Data Dictionary. The task appears in the Task Name list in the Task dialog.
- 6 Select the desired task, for example, **PositionControl**, from the Task Name list and click OK to close the Task dialog.
- 7 Copy one Task block to another subsystem you want to assign to the same task, and double-click it to open the Task dialog.
- 8 From the Task Name list, select **PositionControl** and close the Task dialog.

Result When generating production code, TargetLink groups the two subsystems in the PositionControl task.

Next step After you group subsystems in tasks using the Task block, you can specify the task properties. Refer to [How to Specify Properties of User-Defined Tasks](#) on page 41.

Related topics

Basics

| | |
|--|--------------------|
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Partitioning Tasks and Interrupt Service Routines..... | 24 |

HowTos

| | |
|---|--------------------|
| How to Add a Root Function Template and Specify Its Properties..... | 44 |
| How to Specify Properties of User-Defined Tasks..... | 41 |

References

[Task Block \(📖 TargetLink Model Element Reference\)](#)

How to Specify Properties of User-Defined Tasks

Objective

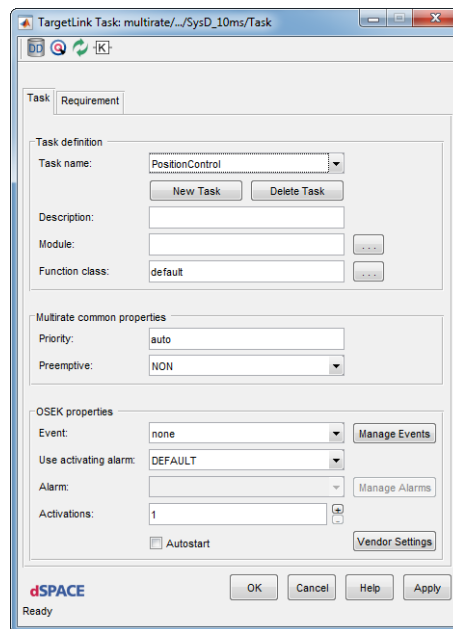
Tasks have several properties, like priority or preemptability. The properties of a task influence its behavior and the flow of the control algorithm represented by the model. The properties of each task are stored in the **DataDictionary/Pool/RTOS/Tasks** subnode in the TargetLink Data Dictionary. The different Task blocks in the model have references to the tasks in the TargetLink Data Dictionary. You can change the properties of a task in any Task block referencing it. Subsequently, you can see the changes in all Task blocks referencing the same task. If you assigned a subsystem to a specific task using the Task block, you can use the Task block dialog to specify the task's properties.

The next section describes how to specify task properties in the Task block dialog. You should adapt the properties to your specific needs.

Method

To specify properties of tasks marked with a Task block

- 1 In your multirate model, open a subsystem containing a Task block.
- 2 Double-click the Task block to open the Task dialog.



- 3 Under Multirate common properties, enter a suitable value to specify the priority of the task, for example, enter 2.
- 4 In the Preemptive list, specify whether the task can be interrupted by other tasks or not, for example, select FULL.

- 5 Under OSEK properties, select the number of possible task activations in the Activations box, for example, select **1**.

Note

OSEK properties apply only to tasks that are created for OSEK-compliant operating systems.

Result

You have now specified several task properties.

Related topics

Basics

[Basics of Specifying Task Properties..... 32](#)

HowTos

[How to Specify Properties of Default Tasks..... 42](#)

References

[Task Block \(📖 TargetLink Model Element Reference\)](#)

How to Specify Properties of Default Tasks

Objective

When generating production code for multirate models, TargetLink creates default tasks collecting atomic subsystems which are not assigned to a user-defined task. If you make no specific input, TargetLink assigns default values to the default tasks' properties. However, you can also specify the properties.

Note

TargetLink creates the default tasks during production code generation. You therefore have to specify the default task properties before production code generation.

To specify the properties, you have to use the templates for [Root step functions](#) provided by the TargetLink Data Dictionary. You can edit the template properties stored in the TargetLink Data Dictionary with the Data Dictionary Manager. For basic information on default tasks and their properties, refer to [Basics of Specifying Task Properties](#) on page 32.

The TargetLink Data Dictionary provides two default templates for root step functions, one for cyclic root step functions and one for noncyclic root step

functions. The following section describes how to specify properties of default tasks on the basis of these default templates.

Precondition

You must have opened a model.

Method

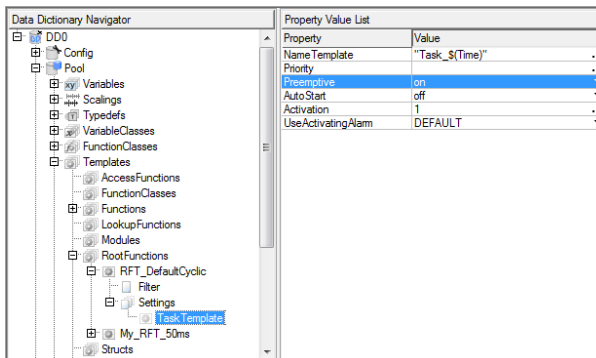
To specify properties of default tasks

- 1 In the MATLAB Command Window, type `dsddman` to open the Data Dictionary Manager.
- 2 In the Data Dictionary Navigator of the Data Dictionary Manager, change to `Pool/Templates/RootFunctions` subnode.
- 3 Open the `RFT_DefaultCyclic/Settings/TaskTemplate` subnode.
- 4 In the NameTemplate edit field in the property value list, enter the name of the task, for example, `MyTask_$(Time)`.

Note

For detailed information on the name macros TargetLink provides, refer to [Generating Unique Names via Name Macros](#) ([TargetLink Customization and Optimization Guide](#)).

- 5 Select on from the Preemptive list.



Result

When generating production code, TargetLink creates tasks for all cyclic root step functions. The tasks are preemptive. The task names begin with `MyTask_`. The rest of each name indicates the sample time. The priority is not defined in this template and will be specified during code generation, if you select the **Assign unspecified priorities** automatically checkbox on the RTOS page of the TargetLink Main Dialog.

Note

If you want TargetLink to generate a C function instead of a task, you have to delete the TaskTemplate subnode in the Settings subnode. You can specify the following C function properties:

- NameTemplate
- FunctionClass
- CodeFile

You can add your own templates in the TargetLink Data Dictionary to filter out specific root step functions. Refer to [How to Add a Root Function Template and Specify Its Properties](#) on page 44.

Related topics

Basics

| | |
|---|----|
| Basics of Specifying Task Properties..... | 32 |
| Generating Unique Names via Name Macros (📖 TargetLink Customization and Optimization Guide) | |

HowTos

| | |
|---|----|
| How to Add a Root Function Template and Specify Its Properties..... | 44 |
| How to Specify Properties of User-Defined Tasks..... | 41 |

How to Add a Root Function Template and Specify Its Properties

Objective

Besides the predefined templates provided with the TargetLink Data Dictionary, you can specify your own templates to filter out specific subsystems in your multirate model. To do so, you have to add templates in the TargetLink Data Dictionary Manager.

TargetLink evaluates the templates in the order of their appearance in the TargetLink Data Dictionary. After TargetLink has found a template that filters out a subsystem, it does not evaluate the remaining templates below this one. You have to move a template to the topmost position in the RootFunctionTemplates sub object if you want TargetLink to evaluate it first.

Note

TargetLink creates the default tasks during production code generation. You therefore have to specify the default task properties before production code generation.

The next section describes how to create a template that filters out the root step function collecting the subsystems with a sample time of 5 ms.

Precondition

You must have opened the Data Dictionary Manager.

Method**To add a root function template and specify its properties**

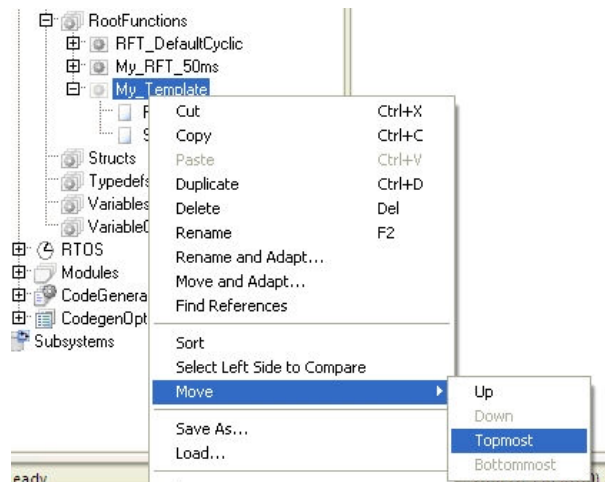
- 1 In the Data Dictionary Navigator of the Data Dictionary Manager, right-click the RootFunctions subnode and select **Create RootFunctionTemplate** from the context menu.

The Data Dictionary Manager adds a new root function template with the **Filter** and **Settings** subnodes.

- 2 Enter a name for the new root function template, for example, **My_Template**.

You have now added a new template. Since TargetLink checks the templates in the order they appear in the RootFunctions subnode, the next step is to move **My_Template** to the topmost position in the subnode to have it checked first.

- 3 Right-click **My_Template** and select **Move** from the context menu.
- 4 Select **Topmost**.



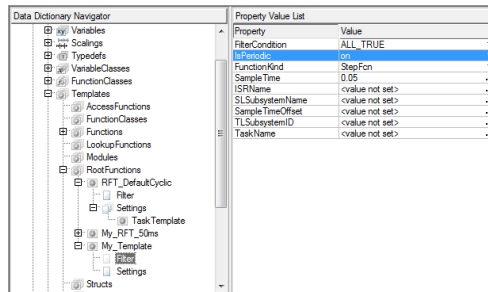
The Data Dictionary Manager moves **My_Template** to the topmost position in the RootFunctions subnode.

Note

Before you proceed with the next step, make sure that the **Show unset properties** button in the Data Dictionary Manager is activated.

- 5 Under **My_Template**, open the **Filter** subnode.
- 6 In the **SampleTime** edit field of the property value list, enter **0.005**.
- 7 From the **FunctionKind** list, select **StepFcn**.

8 From the IsPeriodic list, select on.



Result

You have now added a new template for root step functions collecting the subsystems with a sample time of 5 ms. You can now specify the template properties as described above. When generating production code, TargetLink filters out the default root step function containing the subsystems with a sample time of 5 ms and assigns it the properties you specified in My_Template.

How to Generate Tasks from Stateflow charts

Objective

If you want to assign a Stateflow chart to a task, you have to use the TargetLink [Property Manager](#).

Preconditions

To perform the following steps, several preconditions must be fulfilled:

- You must have opened a model containing a Stateflow chart.
- The task you want to assign the Stateflow chart to must exist in the TargetLink Data Dictionary.
- You should be familiar with the TargetLink Property Manager. Refer to [Modifying Multiple Properties at Once via the Property Manager](#) ([TargetLink Preparation and Simulation Guide](#)).

Method

To generate tasks from Stateflow charts

- 1 In the MATLAB Command Window, type `tlPropman('Start')` to open the TargetLink Property Manager.
- 2 In the Model Navigator of the TargetLink Property Manager, select the Stateflow chart you want to assign to a task.
- 3 In the Property View, add the Task column via the [Column Chooser Dialog](#) ([TargetLink Tool and Utility Reference](#)) if it is not already displayed.
- 4 In the Property View, select the Task cell of the Stateflow chart you want to assign the task to and click the Browse button.

The DD Reference Selection dialog opens and the available tasks in the Pool1/RTOS/Tasks subnode of the TargetLink Data Dictionary are displayed.

- 5 Select the task you want to assign and click Ok.


For more information, refer to [DD Reference Selection Dialog \(Property Manager\)](#) ( [TargetLink Tool and Utility Reference](#)).

Result

You have assigned the Stateflow chart to the selected task. During production code generation, TargetLink places the code resulting from the Stateflow chart in the selected task.

Related topics

Basics

| | |
|--|----|
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Methods of Partitioning Tasks and ISRs..... | 24 |
| Modifying Multiple Properties at Once via the Property Manager ( TargetLink Preparation and Simulation Guide) | |
| Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Rules for Creating Multirate Models..... | 27 |

References

[DD Reference Selection Dialog \(Property Manager\)](#) ( [TargetLink Tool and Utility Reference](#))
[tlPropman](#) ( [TargetLink API Reference](#))

How to Create Interrupt Service Routines

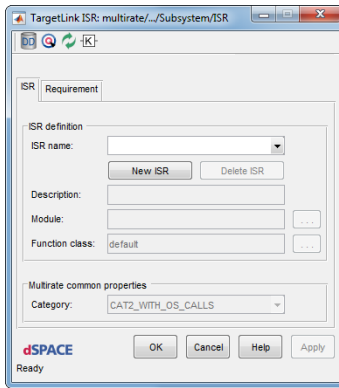
Objective

If you want to create an interrupt service routine from an external function-call-triggered subsystem, you have to use the ISR block.

Method

To create interrupt service routines

- 1 Copy one ISR block from the RTOS Blocks library to the external function-call-triggered subsystem you want to assign to an ISR and double-click it to open the ISR dialog.



- 2 Click **New ISR** to open the **ISR Name** dialog.
- 3 Enter a name for the ISR, for example, **Hardware_Interrupt**, and click **OK** to close the dialog.
TargetLink creates a new ISR with the name **Hardware_Interrupt** in the TargetLink Data Dictionary. The ISR appears in the **ISR Name** list in the **ISR** dialog.
- 4 Click **OK** to close the **ISR** dialog.

Result When generating production code, TargetLink places the code for the external function-call-triggered subsystem in the **Hardware_Interrupt** ISR.

Next step After you created the ISR, you can specify the ISR properties. Refer to [How to Specify Properties of Interrupt Service Routines](#) on page 48.

Related topics

Basics

| | |
|---|----|
| Creating Interrupt Service Routines (ISRs)..... | 34 |
| Methods of Partitioning Tasks and ISRs..... | 24 |
| Rules for Creating Multirate Models..... | 27 |

HowTos

| | |
|--|----|
| How to Specify External Tasks or ISRs..... | 50 |
| How to Specify Properties of Interrupt Service Routines..... | 48 |

How to Specify Properties of Interrupt Service Routines

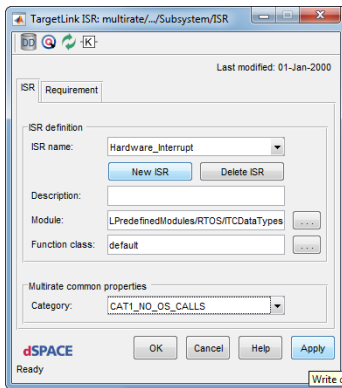
Objective If you make no specific input, TargetLink assigns default values to the ISR properties. However, you can also specify the ISR properties in the **ISR** dialog.

The following section describes how to specify ISR properties. You should adapt the properties to your needs.

Method

To specify properties of Interrupt Service Routines

- 1 In your multirate model, open a subsystem containing an ISR block.
- 2 Double-click the ISR block to open the ISR dialog.



- 3 Under ISR Definition, click the Module browse button to reference a DD Module object in the Select Object dialog.
- 4 Under Multirate Common Properties, select, for example, CAT1_NO_OS_CALLS from the Category list.

Result

You have now specified several ISR properties. When generating production code, TargetLink generates the code of the subsystem assigned to the respective ISR into the referenced DD Module object. If you do not reference a Module object, the code is generated into the file of the parent subsystem.

Related topics

Basics

| | |
|--|----|
| Methods of Partitioning Tasks and ISRs..... | 24 |
| Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Rules for Creating Multirate Models..... | 27 |

References

ISR Block (📖 TargetLink Model Element Reference)

How to Specify External Tasks or ISRs

Process of specifying external tasks

TargetLink provides a method of extending tasks in existing ECU code. If the task frame already exists on your ECU, you can use the Task block to assign several subsystems to the task. If you mark the task as external in the TargetLink Data Dictionary, TargetLink does not generate the task frame, but a C function. You have to call that function in the task. The process of specifying external tasks consists of the following steps:

1. In the TargetLink Data Dictionary, create a module object for the [module](#) the external task resides in.

Note

Both the header and the code file of the module object must have the following FileSpecification properties:

- EmitFile = off
- CodeGenerationBasis = ModelBased
- Responsibility = Default

By setting the EmitFile property to off, you tell TargetLink not to emit the module.

2. In the TargetLink Data Dictionary, create a task subnode for the existing task and assign the subsystems to it using the Task block.
3. In the TargetLink Data Dictionary, specify the function TargetLink has to generate from the subsystem.
4. Call the function in the external task.

External ISRs are specified in the same way.

The following example shows how to extend a task called ExternalTask with the AdditionalFunc function. The external task resides in the C module poscontrol.

Preconditions

The following preconditions must be fulfilled:

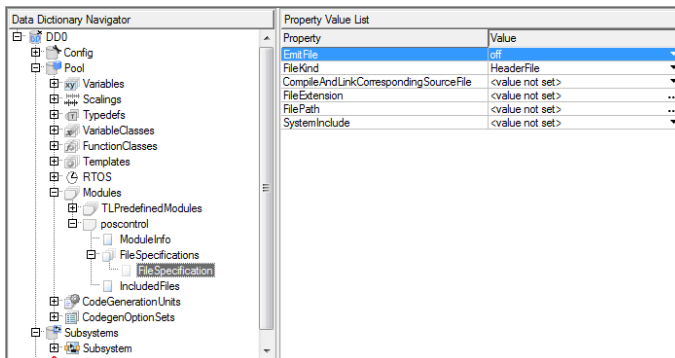
- You must have opened a model.
- You must have opened the TargetLink Data Dictionary Manager.

Method

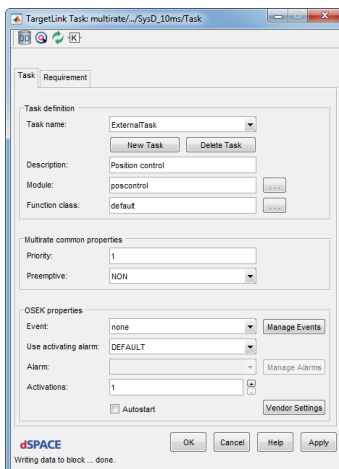
To specify external tasks

- 1 In the TargetLink Data Dictionary, create a new module object in the DataDictionary/Pool/Modules subnode and name it poscontrol.
- 2 Right-click the FileSpecifications subnode of the poscontrol module object, and select Create FileSpecification from the context menu.

- 3 Change to the new **FileSpecification** subnode and set the **EmitFile** property to off for the header file of the poscontrol module.



- 4 Create another **FileSpecification** subnode for the poscontrol module and set the **EmitFile** property to off for the source file of the module.
You have now created a module object in the TargetLink Data Dictionary for the poscontrol module the external task resides in. The next step is to create a task subnode for the existing task in the TargetLink Data Dictionary and assign the subsystem to it using the Task block.
- 5 Copy a Task block from the RTOS Blocks library to the atomic subsystem you want to assign to the external task.
- 6 Double-click the Task block to open the Task dialog.
- 7 Click **New Task**, enter **ExternalTask** in the Task name dialog and click **OK** to close the dialog.
- 8 In the Task name list, select **ExternalTask** and click **Apply** to create the task in the TargetLink Data Dictionary.
- 9 Under **Task Definition**, click the **Module** browse button and select the poscontrol module object in the **Select Object** dialog.



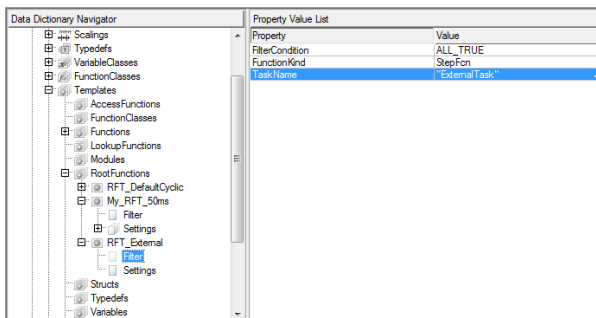
You have now created the external task in the TargetLink Data Dictionary and assigned the subsystem to it. The next step is to specify the function TargetLink generates from the subsystem.

- 10 In the Data Dictionary Navigator, change to the `DataDictionary/Pool/Templates/RootFunctions` subnode and right-click it.
- 11 From the context menu, select `Create RootFunctionTemplate` and enter a name for the new root function template. For example, enter **RFT_External**.

Note

The `TaskTemplate` subnode must not exist in the `Settings` subnode of the new root function template.

- 12 Change to the `Filter` subnode of the new root function template.
- 13 In the Property Value List of the Data Dictionary Manager, select `StepFcn` from the `FunctionKind` list.
- 14 In the `TaskName` edit field, enter **ExternalTask**.



- 15 Change to the `Settings` subnode of the root function template and enter **AdditionalFunc** in the `NameTemplate` edit field.
- 16 Call the `AdditionalFunc` function in your external task.

Result

You have now specified an external task. When generating production code, TargetLink generates a C function called `AdditionalFunc` containing the code of the atomic subsystem.

Related topics

Basics

| | |
|--|----|
| Introduction to Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Methods of Partitioning Tasks and ISRs..... | 24 |
| Partitioning Tasks and Interrupt Service Routines..... | 24 |
| Rules for Creating Multirate Models..... | 27 |

References

ISR Block (TargetLink Model Element Reference)
 Task Block (TargetLink Model Element Reference)

Characteristics of Special Subsystems

Introduction

For some kinds of subsystems in a multirate environment, you have to take some special characteristics into account. You should read this chapter if you want information on how TargetLink handles special types of subsystems listed below. If not, you can skip it.

To explain the behavior of the subsystems, the following example models show how TargetLink handles them and the code resulting from them.

Where to go from here

Information in this section

| | |
|--|----|
| Cyclic Enabled Subsystems..... | 53 |
| Encapsulated Enabled Subsystems..... | 54 |
| Action-Port-Triggered Subsystems..... | 58 |
| Edge-Triggered Subsystems..... | 59 |
| Several Signals Triggering One Edge-Triggered Subsystem..... | 61 |
| Edge-Triggered Subsystem with Task Block..... | 63 |
| Function-Call-Triggered Subsystems..... | 64 |
| Simulink Behavior for Enabled Subsystems..... | 65 |

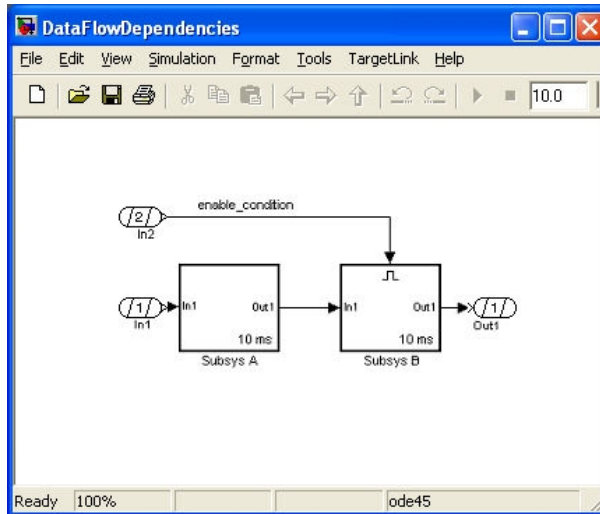
Cyclic Enabled Subsystems

Handling of cyclic enabled subsystems

TargetLink handles cyclic enabled subsystems as cyclic subsystems. You can group them in tasks with cyclic subsystems. The signal driving the enable port is a standard data input. The code checking the enable condition is activated cyclically with the sample time of the enabled subsystem. The functional code for the enabled subsystem is conditionally executed if the condition evaluation results in true.

Example

The following example shows the code resulting from a cyclic enabled subsystem:



Result

```
Task_10ms()
{
    SubsysA();

    if(enable_condition) {
        SubsysB();
    }
}
```

Encapsulated Enabled Subsystems

Different types of encapsulation

The following cases must be distinguished for encapsulated enabled subsystems:

- A cyclic subsystem is encapsulated in an enabled subsystem.
- An enabled subsystem is encapsulated in an enabled subsystem.

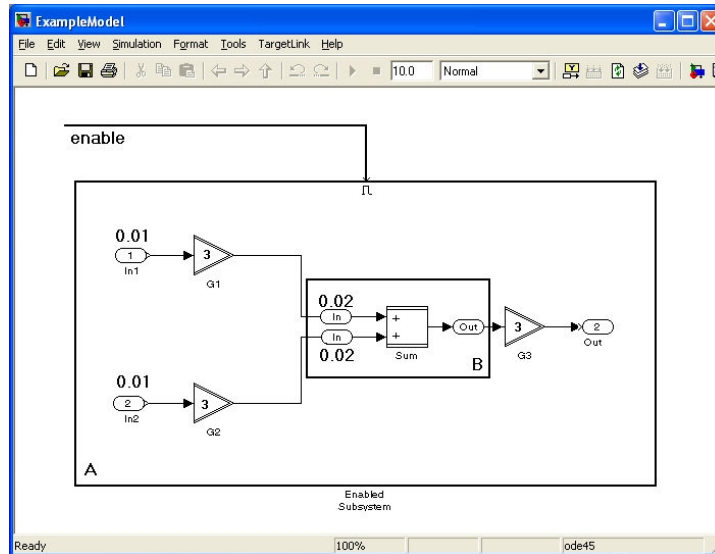
Cyclic subsystem within enabled subsystem

For this kind of model, two cases have to be distinguished:

- The sample time of the inner subsystem is an integer multiple of the outer subsystem's sample time. Refer to [Example of embedded cyclic subsystem with multiple sample time](#) on page 55.
- The sample time of the inner subsystem is not an integer multiple of the outer subsystem's sample time. Refer to [Example of embedded cyclic subsystem with non-multiple sample time](#) on page 55.

Example of embedded cyclic subsystem with multiple sample time

The following example shows a cyclic subsystem encapsulated in an enabled subsystem. The inner subsystem's sample time is an integer multiple of the outer subsystem's sample time.

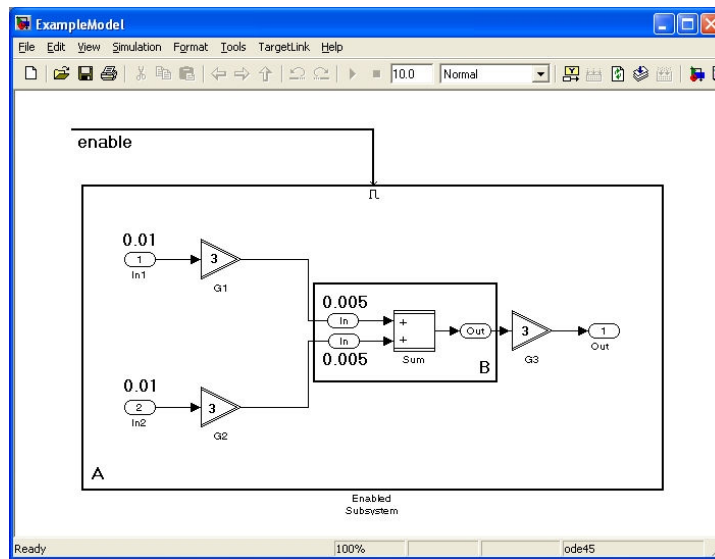


Result The model results in the following pseudo code:

```
Cnt=1; //Init code
Task_10ms()
{
    if (enable)
        G1 = In1 * 3;
        G2 = In2 * 3;
        if (--Cnt==0)
        {
            Sum = G1 + G2;
            Cnt=2;
        }
        Out = Sum * 3;
}
```

Example of embedded cyclic subsystem with non-multiple sample time

The following example shows a cyclic subsystem encapsulated in an enabled subsystem. The inner subsystem's sample time is not an integer multiple of the outer subsystem's sample time.



Result The model results in the following pseudo code:

```
Task_10ms()
{
    if (enable)
    {
        G1 = In1 * 3;
        G2 = In2 * 3;
        Out = Sum * 3;
    }
}

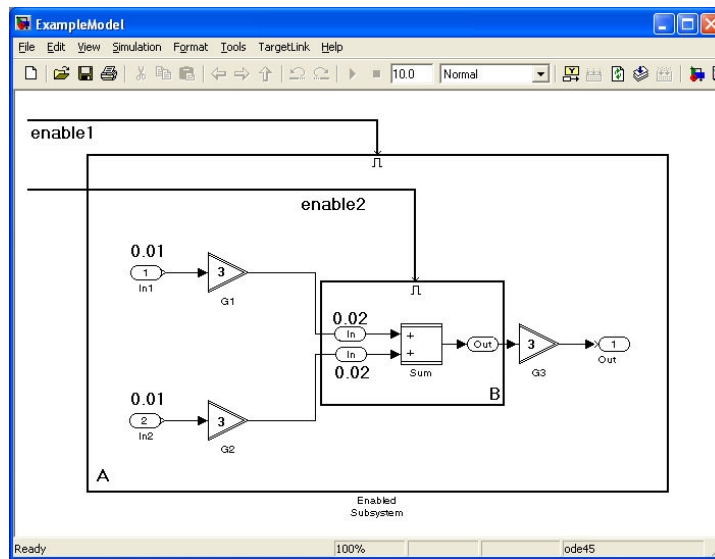
Task_5ms()
{
    if (enable)
    {
        Sum = G1 + G2;
    }
}
```

Nested enabled subsystems For nested enabled subsystems, the same cases as described above must be distinguished:

- The sample time of the inner subsystem is an integer multiple of the outer subsystem's sample time. Refer to Example of embedded enabled subsystem with multiple sample time.
- The sample time of the inner subsystem is not an integer multiple of the outer subsystem's sample time. Refer to Example of embedded enabled subsystem with non-multiple sample time.

Example of embedded enabled subsystem with multiple sample time

The following example shows two nested enabled subsystems. The inner subsystem's sample time is an integer multiple of the outer subsystem's sample time.



Result The model results in the following pseudo code:

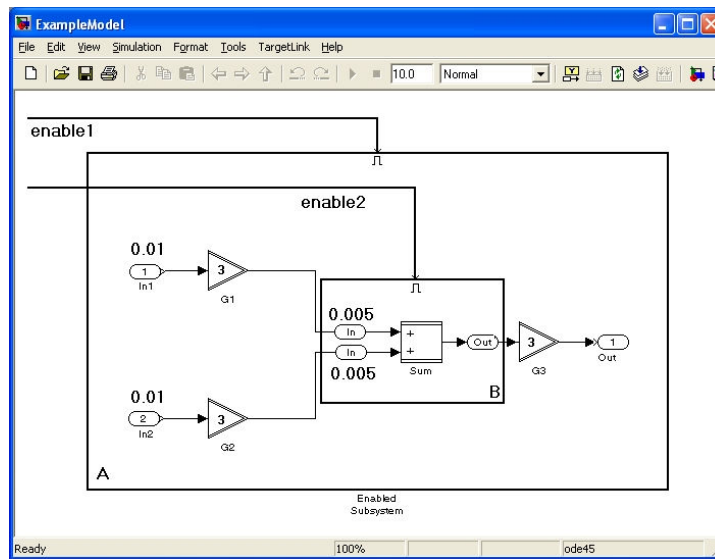
```
Cnt=1;//InitCode

Task_10ms()
{
  if (enable1)
  {
    G1 = In1 * 3;
    G2 = In2 * 3;;
    if (--Cnt==0)
    {
      if (enable2)
      {
        Sum = G1 + G2;
      }

      Cnt=2;
    }
    Out = Sum * 3;
  }
}
```

Example of embedded enabled subsystem with non-multiple sample time

The following example shows two nested enabled subsystems. The inner subsystem's sample time is not an integer multiple of the outer subsystem's sample time.



Result The model results in the following pseudo code:

```
Task_10ms()
{
    if (enable1)
    {
        G1 = In1 * 3;
        G2 = In2 * 3;
        Out = Sum * 3;
    }
}

Task_5ms()
{
    if (enable1 & enable2)
    {
        Sum = G1 + G2;
    }
}
```

Action-Port-Triggered Subsystems

Handling of action-port-triggered subsystems

Action-port-triggered subsystems are handled like function-call-triggered subsystems. Additionally, the states and outputs are reset, if specified, as in enabled subsystems. The If block and the Switch Case block are the only permissible sources for the function call going into the action-port-triggered subsystem.

The rules for enabled subsystems also apply to action-port-triggered subsystems. If an enabled subsystem is encapsulated by an action-port-triggered subsystem, the same algorithm is used as if both subsystems were enabled subsystems. If the

action-port-triggered subsystem becomes a separate task, i.e., the corresponding blocks are computed in a different task than the if condition evaluation for example, its states and outputs cannot be reset. TargetLink displays a warning in this case.

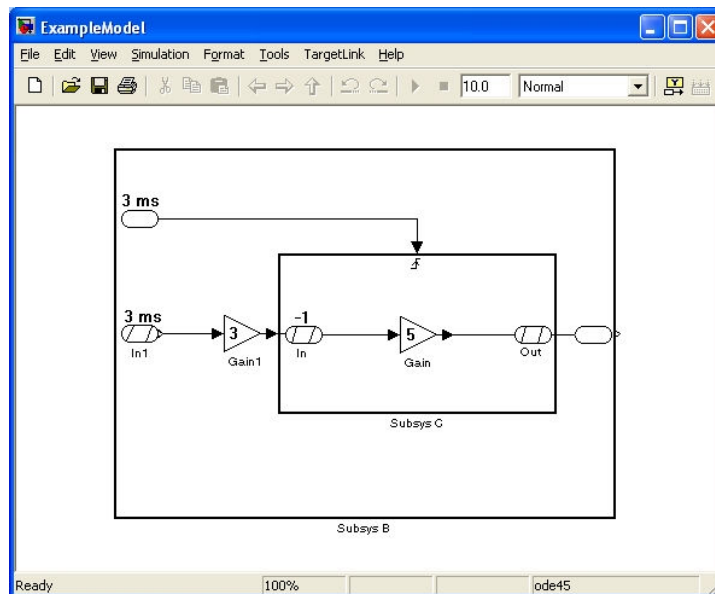
Edge-Triggered Subsystems

Handling of edge-triggered subsystems

In a triggered subsystem, Simulink allows only blocks with a sample time of -1. The blocks inherit the sample time from the signal driving the trigger port. The edge detection of the trigger signal and the execution of the triggered subsystem if the trigger condition is fulfilled have the periodicity of the trigger signal. Usually, detecting edges and executing the subsystem is not done in quick succession, in order to take data flow dependencies into account.

Example

Consider the following example model:



In this example, both the trigger signal and the triggered subsystem, and the source of the triggered subsystem's inports, have the same sample time. Thus, they are executed in the same task. The data flow can be reproduced correctly as shown in the resulting code:

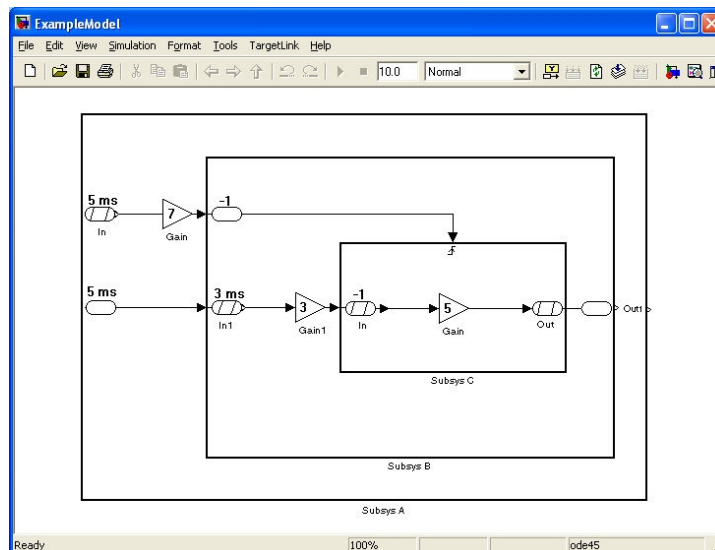
Result

The model results in the following pseudo code:

```
Task_3ms()
{
    if(trigger detected)
        trigger = 1;
    Gain1 = In1 * 3;
    if(trigger)
    {
        trigger = 0;
        SubsysC();
    }
}
```

Example

Suppose the sample time of the trigger signal differs from the sample time of the blocks driving the inputs of the triggered subsystem. In this case the code of the triggered subsystems, that is, the code units for edge detection and the contents of the triggered subsystem, are executed in a different task than the block driving the triggered system's input. Thus, correct data flow is not ensured as shown in the following example model:



Result

The model results in the following pseudo code:

```
Task_5ms() {
    SubsysA();
}

Task_3ms(){
    SubsysB();
}
```

```

SubsysA() {
    Gain = 7 * In;
    if(trigger detected)
        triggerFlag = 1;
    if(triggerFlag) {
        triggerFlag = 0;
        SubsysC();
    }
}

```

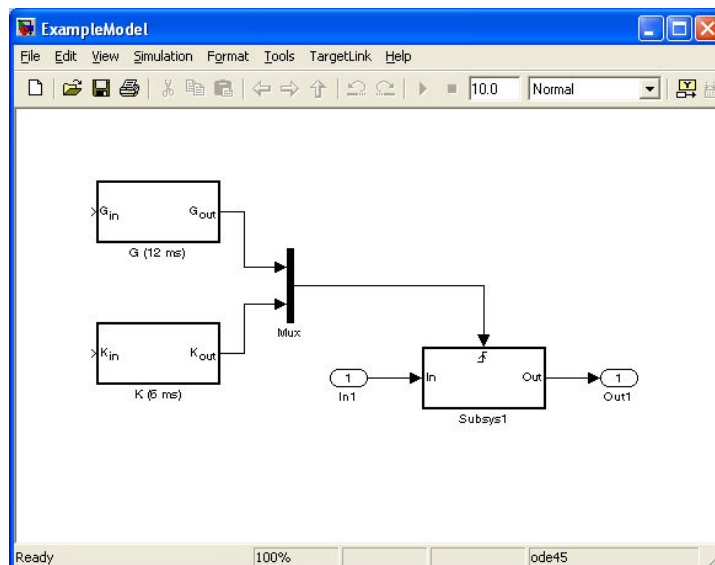
Generally, executing the trigger condition evaluation and, if necessary, the triggered subsystem is performed with the sample time of the source driving the trigger signal. This applies also if the trigger source is located in a function-call-triggered or edge-triggered subsystem.

There is a special case if more than one trigger source exists. Refer to [Several Signals Triggering One Edge-Triggered Subsystem](#) on page 61.

Several Signals Triggering One Edge-Triggered Subsystem

Example

Consider the following model:



The trigger sources G and K can be grouped in the same task (using the Task block), or they can reside in different tasks. Thus, the following cases must be distinguished for the edge-triggered subsystem that can be triggered by several signals:

- Trigger sources G and K are located in the same task.
- Trigger sources G and K are located in different tasks.

Trigger sources residing in the same task

The trigger flag is set after each calculation of Gout and Kout. The evaluation of the trigger flag (and execution of the triggered subsystem) is done according to the signal flow after these two positions.

In the model shown above, there are two trigger sources (Gout and Kout), but there is only one place evaluating the trigger flag and calling the triggered subsystem. This ensures the same behavior as in Simulink simulation.

Result

The model results in the following pseudo code:

```
Task6ms()
{
    if (trigger because of K) {
        TriggerFlag = 1;
    }
    ...
    // every second time:
    if (trigger because of G) {
        TriggerFlag = 1;
    }
    ...
    if (TriggerFlag) {
        Subsys1();
        TriggerFlag = 0;
    }
}
```

Trigger sources residing in different tasks

If the trigger sources are located in different tasks, the trigger flag is set after the calculation of Gout and Kout in each task. The evaluation of the trigger flag and the execution of the triggered subsystem are also done after setting the trigger flag in each task according to the signal flow. This behavior is different from the Simulink simulation. In Simulink, the triggered subsystem is executed once at a certain time step, if both trigger sources detected an edge. In TargetLink, the generated code executes the triggered subsystem twice.

Result

If the trigger sources are located in different tasks, the example model results in the following pseudo code:

```
Task6ms()
{
    if (trigger because of K) {
        TriggerFlag = 1;
    }
    ...
    if (TriggerFlag) {
        Subsys1();
        TriggerFlag = 0;
    }
}
```

```

Task12ms()
{
    if (trigger because of G) {
        TriggerFlag = 1;
    }
    ...
    if (TriggerFlag) {
        Subsys1();
        TriggerFlag = 0;
    }
}

```

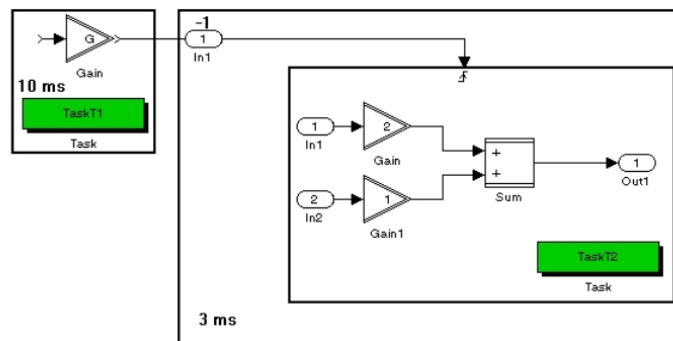
Edge-Triggered Subsystem with Task Block

Using the Task block for partitioning

The previous chapters described how the code units resulting from edge-triggered subsystems are put into the task which in the trigger signal resides. This kind of partitioning can be changed with the help of the Task block.

Example

Consider the following example model:



You can specify that the code of the trigger signal source and the code of the triggered subsystem are put into different tasks by using the Task block. The trigger condition is evaluated in Task T2.

To activate a task, a corresponding operating system function must be called. TargetLink uses the `TlTriggerTask(<Taskname>)` macro to call an appropriate operating system function. You can adapt this [macro](#) to your operating system. For OSEK applications, TargetLink automatically uses `ActivateTask(<Taskname>)`.

For more information on task activation, refer to the following topics:

- [Building a Multitasking Application](#) on page 153
- [Setting Up Alarms for Cyclic OSEK Task Activation](#) on page 108

Limitation

If the code unit of the trigger signal resides in a different task than the code unit of the triggered subsystem, you cannot use the Show Output Port property of the trigger block.

Related topics

Basics

| | |
|---|-----|
| Building a Multitasking Application | 153 |
| Setting Up Alarms for Cyclic OSEK Task Activation | 108 |

Function-Call-Triggered Subsystems

Distinction of cases


In the context of production code generation in a multirate environment, the following two cases must be considered:

- The function-call-triggered subsystem contains a Task block.
- Several function-call-trigger sources reside in different tasks.

Function-call-triggered subsystem with Task block


If the function-call-triggered subsystem contains a Task block, it is not called directly by the source of the function call, but via task activation using the `TlTriggerTask(<Taskname>)` or `ActivateTask(<Taskname>)` macro. Refer to [Edge-Triggered Subsystem with Task Block](#) on page 63.

Several trigger sources in different tasks

If several trigger sources of a function-call-triggered subsystem reside in different tasks, this results in production code where one C function can be called by several tasks. In this case, the C function must be protected by the Critical Section block (refer to [Protecting Critical Sections](#) on page 126) or it must be reentrant, that is, it must be reusable (refer to [Reusing C Functions and Variables for Identical Model Parts \(Function Reuse\)](#) ( [TargetLink Customization and Optimization Guide](#))).

Related topics

Basics

| | |
|---|-----|
| Edge-Triggered Subsystem with Task Block | 63 |
| Protecting Critical Sections | 126 |
| Reusing C Functions and Variables for Identical Model Parts (Function Reuse) | |
| ( TargetLink Customization and Optimization Guide) | |

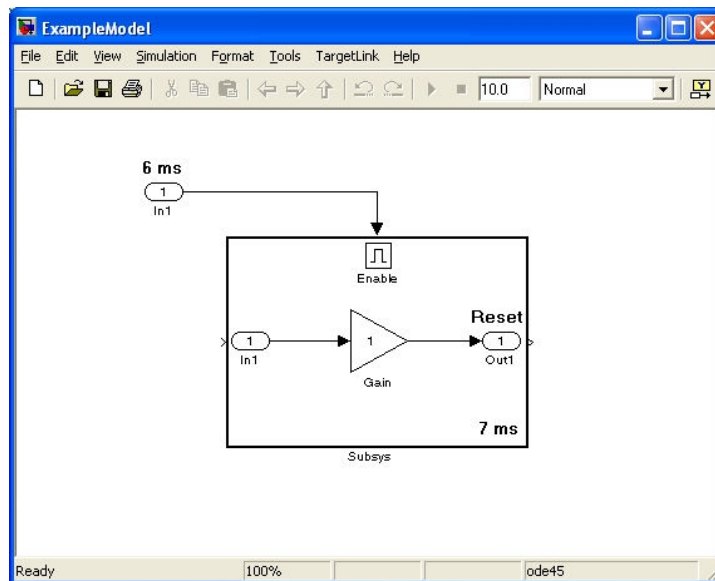
Simulink Behavior for Enabled Subsystems

TargetLink behavior

In special cases, the TargetLink behavior differs from the Simulink behavior for the benefit of more efficient production code.

Example

In the following example model, the blocks within the enabled subsystem have a sample time of 7 ms. The enable signal's sample time is 6 ms.



Since the blocks of the enabled subsystem are calculated with a periodicity of 7 ms, it would be sufficient to evaluate the enable condition every 7 ms as well. However, Simulink provides a method to reset the outputs of the enabled subsystem if the system is disabled. This reset is performed with the periodicity of the enable signal. To reproduce the Simulink behavior exactly, a task with the greatest common divisor of the two sample times, 1 ms in this case, that calculates the subsystem's blocks every seventh time and evaluates the enable condition every sixth time is needed.

Result

Usually, fast tasks like that are undesirable. Therefore, TargetLink generates one task with a sample time of 6 ms calculating the enable condition and one task with a sample time of 7 ms evaluating the enable condition. Additionally, it resets the outputs if necessary. Thus, the reset may occur a little later than in the Simulink simulation. Resetting the outports possibly includes [intertask communication](#) if the receiver of the signal resides in a different task.

Managing Intertask Communication

Using messages to ensure data integrity

Tasks communicate with each other by exchanging signals. Since tasks can interrupt each other during data exchange, it is important to ensure the signals' data integrity. TargetLink provides a mechanism to ensure the data integrity of the exchanged signals. This mechanism is called a message. TargetLink provides the following methods to create messages:

- You can create messages and assign them specific signals.
- You can have TargetLink create messages by default.

All messages are based on a data model provided by the TargetLink Data Dictionary. You can specify the form and the properties of the message you use for data exchange on the basis of this data model. If you make no specific input, TargetLink specifies the message form.

Message Basics

Signal lines represent data

In your model, the data exchanged between tasks is represented by signal lines available at the inports and outports of the subsystems.

The senders and receivers of data do not need to be tasks, they can also be interrupt service routines (ISRs) or standard C functions. The next section focuses on tasks.

Where to go from here

Information in this section

| | |
|--|----|
| Using Messages to Ensure Data Integrity..... | 68 |
| Mechanisms for Exchanging Messages..... | 73 |
| Optimizing Intertask Communication..... | 74 |
| Message Properties..... | 74 |
| How to Create a New Message..... | 77 |
| How to Group Signals in a Message..... | 78 |
| How to Specify the Properties of an Existing Message..... | 80 |
| How to Assign Signals to Message Components..... | 83 |
| Detection of Unit Delay and Zero Order Hold Blocks for Rate Transition..... | 85 |

Using Messages to Ensure Data Integrity

Message basics

In TargetLink, the mechanism for ensuring data integrity for signals exchanged between subsystems is called a message. A message is a group of signals that belong together. A message is represented by one or more plain variables or by a structure with several components called a struct variable below. During data exchange, TargetLink protects the data integrity of a message by ensuring that all signals sent from one task to another come from the same sampling step. TargetLink uses local copies of the message variable at the sender and the receiver for this purpose if necessary.

Note

Do not confuse the message term with OSEK message. For more information on OSEK messages, refer to [Mechanisms for Exchanging Messages](#) on page 73.

Local copy

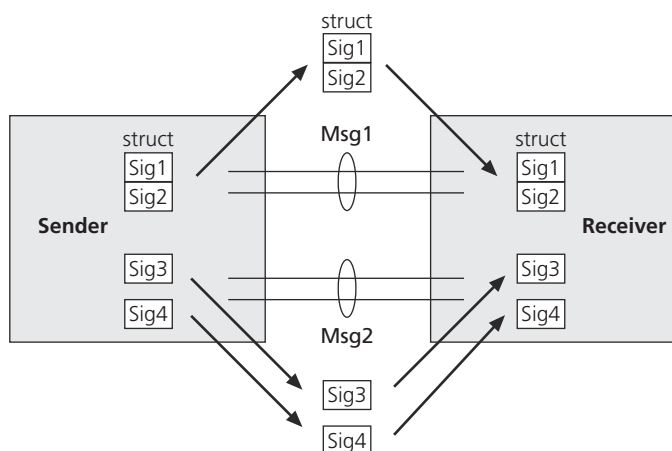
If a task sending a message is preemptive, it must not be interrupted by the receiver of the signals until all signals are stored in the global message variable. TargetLink creates local copies of the message variable at the sender and the receiver to ensure this.

Note

In the description below, the sender and the receiver can interrupt each other, so they both need a local copy. However, there may also be cases where only one of them, or even neither of them, needs a local copy.

The sender uses the local copy to calculate its signals. After the calculation is completed, the sender copies the message data from the local copy to the message variable. The data is copied from the message variable to the receiver's local copy. Subsequently, the receiver uses the data. TargetLink protects the copy process from being interrupted. It lets you select a protection mechanism separately for the sender and each receiver.

The following example shows the message concept on the basis of a sending and a receiving task:



The sender sends four signals to the receiver. Sig1 and Sig2 are combined in Msg1. Msg1 is represented by a struct variable, Sig1 and Sig2 are assigned to separate struct variable components. Msg2 consists of two plain variables that Sig3 and Sig4 are assigned to.

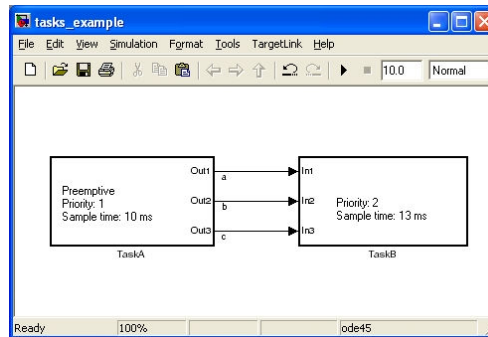
The sender and the receiver each have local copies of the messages. The sender calculates the data for Msg1 and Msg2 and copies it from its local copies to the global message buffers. Afterwards, the receiver copies the data from the global

message buffers to its local copies. To ensure data integrity, TargetLink protects the copy processes of each message using the specified protection mechanism.

The following examples show the two basic use cases in which data integrity must be ensured.

Exchanging several signals between tasks

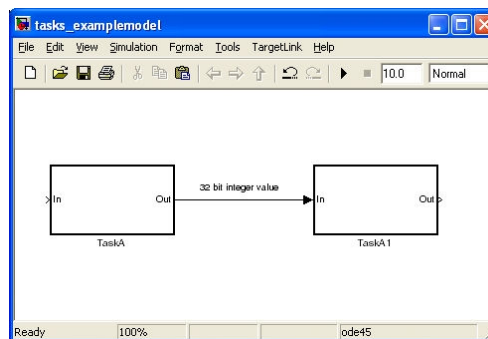
Consider the following example model:




In this model, TaskB has a higher priority than TaskA and can therefore interrupt TaskA. TaskA calculates its output signals every 10 ms. Suppose TaskB interrupts TaskA at a point in time, for example $t = 13$ ms. At this point in time, TaskA may already have calculated output signals a and b, but has not yet finished calculating output signal c. In this case, TaskB uses new values for its input signals a and b, and an old value for input signal c. To prevent the signals coming from different sampling steps, TaskA has to ensure data integrity. This applies accordingly if TaskA sends a vector signal to TaskB instead of several signals. A similar scenario is conceivable if sending task A has a higher priority than task B and interrupts task B while it is reading signals.

Exchanging a scalar signal between tasks

In some special cases it is even necessary to ensure data integrity for a scalar signal. In the following example model, TaskA sends a 32-bit integer value to TaskB:



Using a 16-bit processor, it normally takes two assembler instructions to copy the 32-bit integer value from TaskA to TaskB. If a task switch occurs between the first and the second assembler instructions, Task B uses a wrong value for calculation.

To ensure data integrity in this case, TargetLink provides a property called **AtomicAccess** for all base data types. This property allows you to specify whether the corresponding data type can be accessed with one - not interruptible - assembler instruction or not. You can specify this for each code generation target separately. If **AtomicAccess** is switched off, TargetLink protects access to the scalar variable of the selected base data type. For more information, refer to [Customizing Data Types](#) ( [TargetLink Customization and Optimization Guide](#)).

In the example of exchanging several signals between tasks, TargetLink ensures the data integrity of the exchanged signals when you specify a message and assign all three signals to it.

Specifying messages

TargetLink provides two methods to specify messages:

- If you make no specific input, TargetLink assigns each signal to a separate message.
- You can specify messages yourself.

Typically, messages are specified in the following steps:

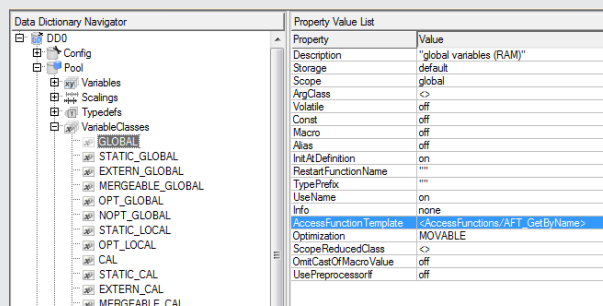
1. Create a message.
2. Assign signals to the message.
3. Specify the form and the properties of the message (optional).

For more information on the message data model, refer to [Message Properties](#) on page 74.

Note

TargetLink actually assigns all signals that are exchanged between root step functions to messages. This is also the case if a root step function receives signals from outside the TargetLink subsystem or sends signals to outside the TargetLink subsystem. However, it does not assign signals to messages if all of the following preconditions are fulfilled:

- The signals are received from outside the TargetLink subsystem or are sent to outside the TargetLink subsystem.
- You have not assigned the signals to messages.
- The signals are received and sent by subsystem inports/outports whose interface variables have access functions specified for them. Access functions can be specified for interface variables by using TargetLink InPort/Bus Inport blocks or TargetLink OutPort/Bus Output blocks and selecting a variable class that references an AccessFunction template in the TargetLink Data Dictionary:



For more information, refer to [Basics on Access Functions](#) ([TargetLink Customization and Optimization Guide](#)).

Thus you can prevent TargetLink from assigning signals to messages at the boundaries of a TargetLink root system by assigning variable classes, that reference an access function, to the interface variables concerned.

Note

The output variable of a TargetLink OutPort block of a sender task must not be merged with the message variable of the TargetLink OutPort block of the sender task if the copy process from the local copy to the message must be protected at the sender task.

The output variable of a TargetLink InPort block of a receiver task must not be merged with the message variable of the TargetLink InPort block of the receiver task if the copy process from the message to the local copy must be protected at the receiver task.

Related topics

Basics

| | |
|--|----|
| Customizing Data Types (📖 TargetLink Customization and Optimization Guide) | |
| Mechanisms for Exchanging Messages..... | 73 |
| Message Properties..... | 74 |

Mechanisms for Exchanging Messages

Data exchange mechanisms

TargetLink provides the following data exchange mechanisms for messages:

- Global buffer
- OSEK message

Global buffer

If you use a global buffer to exchange messages between tasks, TargetLink copies the calculated data from a local copy at the sending task to the global message buffer. To ensure data integrity, it protects the copy process from being interrupted. The receiving task reads the data from the global message buffer. This copy process is also protected.

The global message buffer can take one of the following forms:

- Each signal is assigned to a separate C variable.
- The global message buffer is a C struct variable and each signal is assigned to a separate component of the variable.

TargetLink lets you specify the name and the variable class of the global message buffer variable and the sender's and receiver's local copies.

OSEK Message

If you use an OSEK message to exchange data between tasks, TargetLink uses the OSEK API function `SendMessage (<Msg>, <AccessNameRef>)` to send the data from the sender to the receiver. The receiver reads the data using the OSEK API function `ReceiveMessage (<Msg>, <AccessNameRef>)`. TargetLink creates a local copy of the sender. The local copy is used to calculate the OSEK message's signals. The `SendMessage` command transfers the data to the receiver and ensures data integrity at the sender. The receiver copies the data to its local copy, which is also created by TargetLink, using the `ReceiveMessage` command. Additionally, the `ReceiveMessage` command ensures data integrity at the receiver for all signals belonging to the message. You are not allowed to use OSEK messages without local copies at the sender or receiver. OSEK messages can be used only in applications for OSEK-compliant RTOSs. For more information on OSEK messages, refer to the OSEK communication specification (OSEK COM), version 2.2.2.

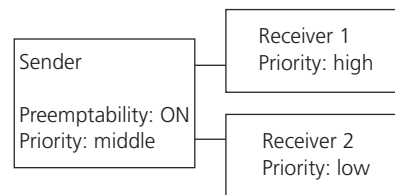
Optimizing Intertask Communication

Rules for optimization

Whether you have to use a local copy and protect access to the global message variable depends on the preemptability and the priority of tasks. The following rules apply to the optimization of intertask communication:

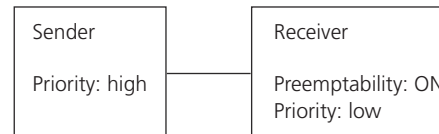
- The sender has to ensure data integrity if it can be interrupted by at least one receiver. The local sender copy and the protection of the message variable access cannot be optimized out.

The example below shows a sender and two receivers. Since the sender is preemptible, it can be interrupted by Receiver 1, which has a higher priority than the sender. Thus, the sender has to ensure data integrity and cannot optimize out its local copy and protection of global message variable access.



- A receiver has to ensure data integrity if it can be interrupted by the sender. The local receiver copy and protection of message variable access cannot be optimized out.

In the example below, the sender can interrupt the receiver due to its higher priority and the preemptability of the receiver. Thus, the receiver has to ensure data integrity and cannot optimize out its local copy and protection of global message variable access.



- Both the sender and the receiver have to ensure data integrity if they can interrupt each other. In this case, neither the sender nor the receiver can optimize out its local copy and protection of message variable access.

Message Properties

Introduction

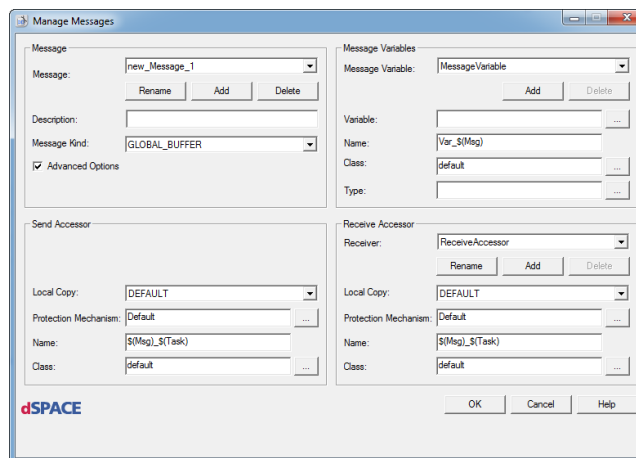
Messages have several properties that let you specify, for example, the name and the [variable class](#) of the global message variable and the local copies. The messages including all properties are stored in the TargetLink Data Dictionary.

Note

TargetLink supports intertask communication for buses. If you map the buses to a struct variable in the production code, you cannot specify intertask communication properties (message, component, and accessor) for the root bus. You must specify the properties for the bus elements separately in the same way as you do for TargetLink InPort blocks. If you do not reference a message for a bus element, TargetLink assigns each bus element to a separate message. For more information on how to use buses in TargetLink, refer to [Customizing the Representation of Buses in Production Code](#) ([TargetLink Customization and Optimization Guide](#)).

Managing messages

To visualize the properties of the data model, the Manage Messages dialog is used. The following illustration shows the Manage Messages dialog and the properties you can specify for messages.



Message group box

In the Message group box, you can create, rename and delete messages. Additionally, you can specify if a message is a global buffer or an OSEK message.

Message Variables group box

In the Message Variables group box, you can specify the form of the global message variable(s). You can use a variable that is defined in the **DataDictionary/Pool/Variables** subnode in the TargetLink Data Dictionary or specify the name, variable class and C data type of the variable(s).

Send Accessor group box

The settings in the Send Accessor group box let you specify properties of the local sender copy. For example, you can specify if TargetLink must protect message access and create a separate local copy variable. You can specify the protection mechanism used and the name and variable class of the local sender copy. However, if the variable class of the local sender copy specifies an

optimizable variable, the local sender copy may be optimized. For example, it can be a block variable.

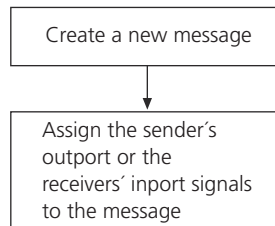
Receive Accessor group box

The properties of the Receive Accessor group box are the same as the Send Accessor group box properties. Since a message can have only one sender but several receivers, TargetLink lets you specify the properties for each local receiver copy separately.

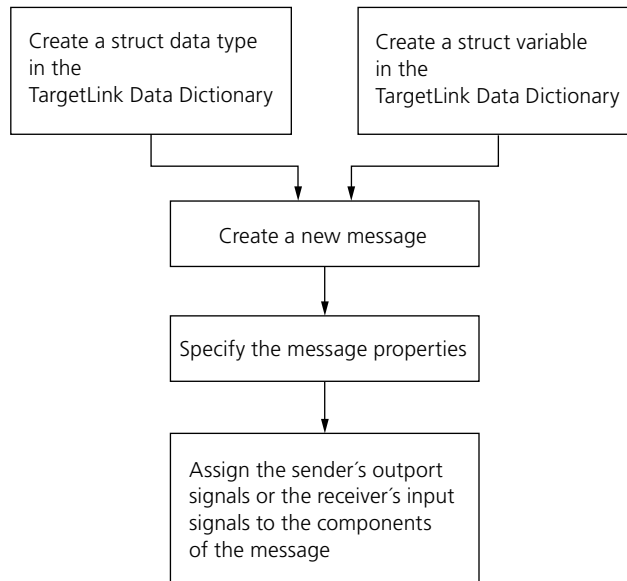
For information on how to specify the properties of a message, refer to [How to Specify the Properties of an Existing Message](#) on page 80.

Typical workflow

Two typical workflows for assigning signals to messages are described below. If you do not want to specify properties of a message, you can have TargetLink do this for you. The typical workflow for this case is shown in the following illustration:



If you make no further input, TargetLink specifies all message properties for you. However, you can also specify the message properties. The typical workflow for specifying messages represented by a struct variable is shown below:



For detailed information, refer to the following topics:

- [How to Create a New Message](#) on page 77
- [How to Group Signals in a Message](#) on page 78
- [How to Specify the Properties of an Existing Message](#) on page 80
- [How to Assign Signals to Message Components](#) on page 83

Related topics

Basics

[Customizing the Representation of Buses in Production Code](#) (📖 TargetLink Customization and Optimization Guide)

HowTos

| | |
|---|--------------------|
| How to Assign Signals to Message Components..... | 83 |
| How to Create a New Message..... | 77 |
| How to Group Signals in a Message..... | 78 |
| How to Specify the Properties of an Existing Message..... | 80 |

How to Create a New Message

Objective

TargetLink lets you assign signals of an inport or outport to a message existing in the TargetLink Data Dictionary. If you want to assign a signal to a message that does not yet exist in the TargetLink Data Dictionary, you have to create the message first.

Preconditions

You must have opened a model.

Method

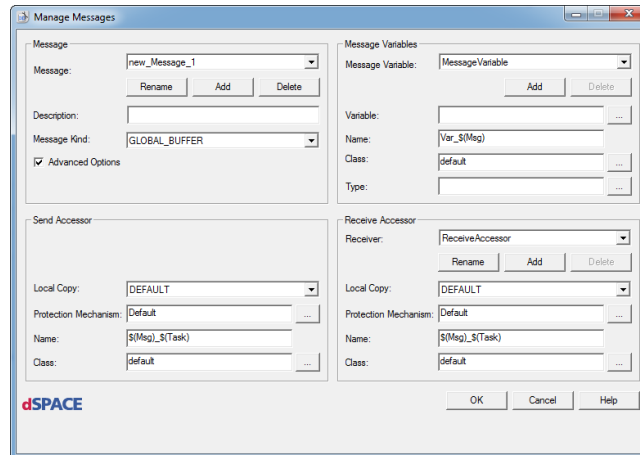
To create a new message

Note

You can perform the following steps with the InPort block.

- 1 In your model, double-click the OutPort block of a subsystem to open the OutPort dialog.

- 2 Click **Manage Messages**. The **Manage Messages** dialog opens.



- 3 Click **Add Message**. The **Add Message** dialog opens.
- 4 Enter the name of the message, for example, **MessageA**, and click **OK** to close the **AddMessage** dialog.
- 5 Click **OK** to close the **Manage Messages** Dialog.

Result TargetLink adds a new message with the name MessageA to the TargetLink Data Dictionary.

Related topics

References

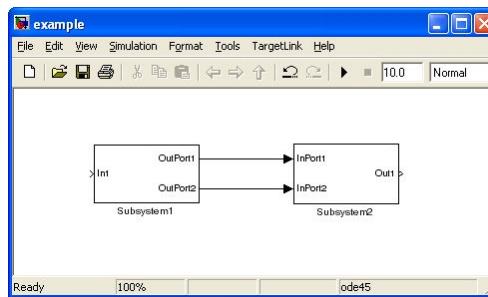
[Bus Inport Block \(TargetLink Model Element Reference\)](#)
[Bus Outport Block \(TargetLink Model Element Reference\)](#)
[InPort Block \(TargetLink Model Element Reference\)](#)
[OutPort Block \(TargetLink Model Element Reference\)](#)

How to Group Signals in a Message

Objective To ensure the data integrity of several signals, you must assign them to the same message. If you make no specific input, TargetLink creates a message for each signal. To overrule this method, you can group several signals in one message.

Preconditions To perform the next steps, the following preconditions must be fulfilled:

- You must have opened a model containing, for example, a subsystem with two OutPort blocks connected to a subsystem with two inport blocks, as shown in the following illustration:

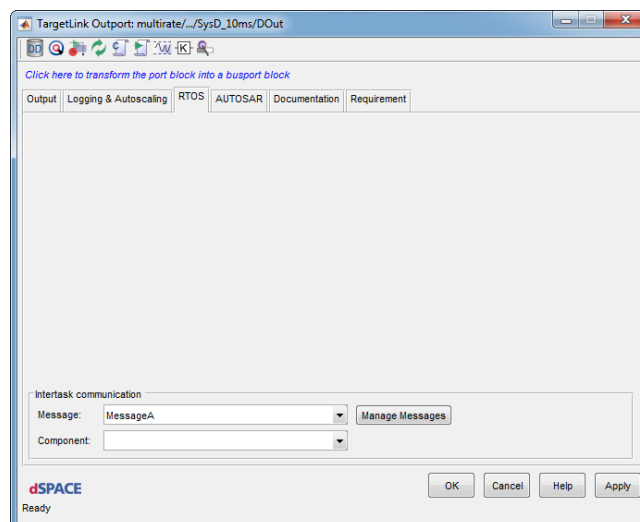


- At least one message must exist in the TargetLink Data Dictionary.

Method

To group signals in a message

- 1 In your model, double-click the first OutPort block of the subsystem sending signals to another subsystem.
The OutPort dialog opens.
- 2 Change to the RTOS page.
- 3 Under Intertask communication, select a message from the Message list, for example MessageA.



- 4 Click OK.
TargetLink assigns the subsystem's output to the selected message.
- 5 Repeat the steps above for the second OutPort block in the subsystem.

Result

TargetLink assigns the two signals between the OutPort blocks of the first subsystem and the InPort blocks of the second subsystem to MessageA and ensures data integrity for the message during data transfer. If you make no further input, TargetLink specifies all properties of MessageA.

Note

Instead of selecting MessageA at the two outports, you can also select it at the two inports of the second subsystem. This results in the same code. You are also allowed to specify the message at the outport and at the inport. If you do, you have to ensure that the same message is selected at the sender and the receiver.

Related topics

Basics

[Message Basics..... 68](#)

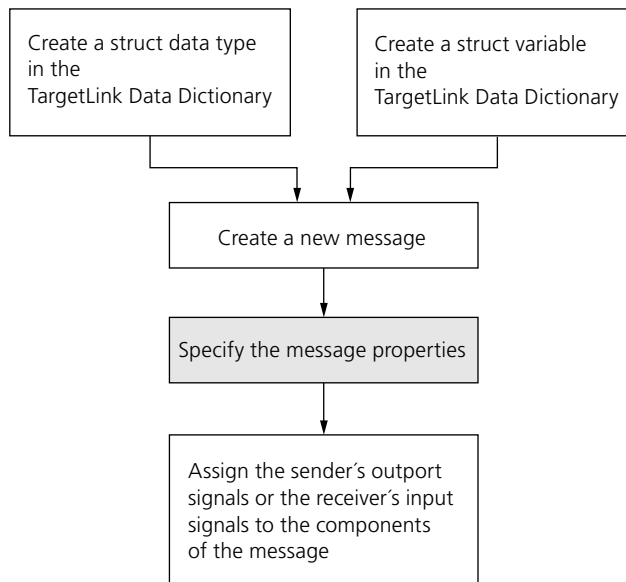
References

[Bus Inport Block \(TargetLink Model Element Reference\)](#)
[Bus Output Block \(TargetLink Model Element Reference\)](#)
[InPort Block \(TargetLink Model Element Reference\)](#)
[OutPort Block \(TargetLink Model Element Reference\)](#)

How to Specify the Properties of an Existing Message

Objective

If you want to assign signals to the components of a struct variable, you have to execute the following workflow:



If you want to use a struct variable, the first step is to create the variable in the TargetLink Data Dictionary. For details on how to create a new struct variable, refer to [How to Create Explicitly Specified Structured Variables](#) ([TargetLink Customization and Optimization Guide](#)). Subsequently, you have to create a message. Refer to [How to Create a New Message](#) on page 77.

Specifying message properties

The next step is to specify the form of the message and all message properties. For more information on the message data model and the message properties, refer to [Message Properties](#) on page 74.

Preconditions

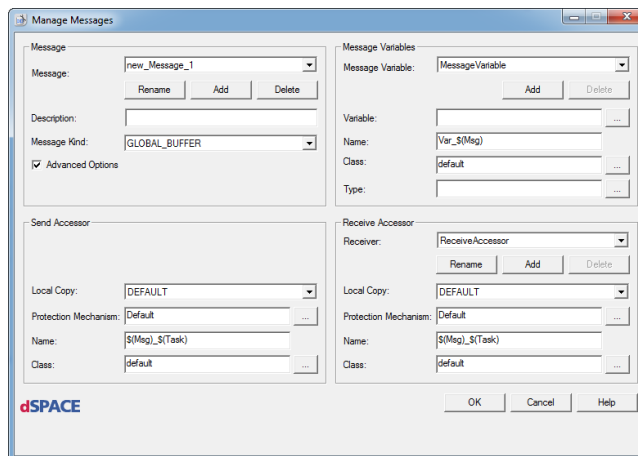
To perform the next steps, the following preconditions must be fulfilled:

- At least one struct variable must exist in the TargetLink Data Dictionary. The variable used is called `My_Variable` below. The struct variable must contain the same number of components as there are signals belonging to the message. The data types of the struct components must be the same as the data types of the corresponding outputs or inputs. In the example below, there are two signals belonging to the message, each of type `Int16`.
- You must have opened a model.
- You must have created a message. Refer to [How to Create a New Message](#) on page 77.

Method

To specify the properties of an existing message

- 1 In your model, double-click an `OutPort` block to open the `OutPort` dialog.
- 2 Click `Manage Messages` to open the `Manage Messages` dialog.



- 3 In the **Message** group box, select the message you want to edit from the **Message** list.

Note

The following rule applies to the next steps: After each step, you can stop and have TargetLink specify the remaining properties.

- 4 Select **GLOBAL_BUFFER** from the **Message Kind** list.
- 5 Select the **Advanced Options** checkbox to enable the further properties of the dialog.
- 6 Click the **Browse** button of the **Variable** edit field and select **My_Variable**.
TargetLink references the **My_Variable** struct variable in the **TargetLink Data Dictionary**. This variable is used for the message **My_Message**.
- 7 Under **Send Accessor**, select **ON** from the **Local Copy** list to specify that the sender must use a local message copy and protect access to the global message variable **My_Variable**.
- 8 From the **Protection Mechanism** list, select **DisableAllInterrupts** to specify that the copy action from the local sender copy to the global message variable is protected by the protection mechanism **DisableAllInterrupts**.
- 9 Enter the name of the local copy variable in the **Name** list in the **Send Accessor** group box. For example, enter **Snd_My_Message**.
- 10 Repeat the three steps above for the receive accessor accordingly.

Result

During production code generation, TargetLink creates the **My_Message** message based on the **My_Variable** struct variable. The global buffer exchange mechanism is used for data exchange. Both the sender and the receiver use a local copy, **Snd_My_Message** or **Rcv_My_Message**, for the message and the protection mechanism **DisableAllInterrupts** to ensure data integrity. The protection mechanism used can be different for sender and receiver.

Next step

Depending on whether you use the **GLOBAL_BUFFER** message kind represented by a struct variable or the **OSEK_MESSAGE** message kind, you have to assign different signals to different message components. Refer to the following topic: [How to Assign Signals to Message Components](#) on page 83.

Related topics

Basics

| | |
|-------------------------|----|
| Message Basics..... | 68 |
| Message Properties..... | 74 |

HowTos

| | |
|---|----|
| How to Assign Signals to Message Components..... | 83 |
| How to Create a New Message..... | 77 |
| How to Create Explicitly Specified Structured Variables (📖 TargetLink Customization and Optimization Guide) | |

References

| |
|--|
| Bus Inport Block (📖 TargetLink Model Element Reference) |
| Bus Outport Block (📖 TargetLink Model Element Reference) |
| InPort Block (📖 TargetLink Model Element Reference) |
| OutPort Block (📖 TargetLink Model Element Reference) |

How to Assign Signals to Message Components

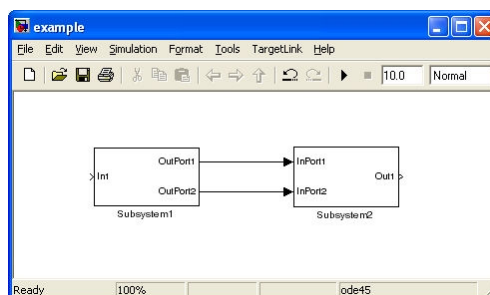
Objective

If you use a struct variable for the message, you have to assign the inport and outport signals of your model to the separate components of the struct variable.

Preconditions

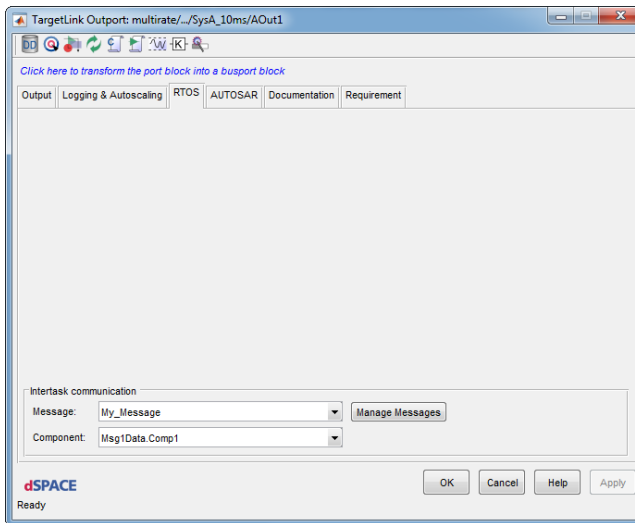
To perform the next steps, the following preconditions must be fulfilled:

- At least one struct variable must exist in the Data Dictionary. For the following steps, it is assumed that the struct variable has the components **Component1** and **Component2**.
- You must have referenced the struct variable to a specific message.
- You must have opened a model containing, for example, a subsystem with two OutPort blocks connected to a subsystem with two InPort blocks, as shown in the following illustration:



Method**To assign signals to separate message components**

- 1 In your model, double-click an OutPort block to open the Outport dialog and change to the RTOS page.



- 2 Under Intertask communication, select **My_Message** from the **Message** list to assign the signal to **My_Message**.
- 3 Select a component from the **Component** list, for example, **Msg1Data.Component1** and click **Apply**.
TargetLink assigns the signal to **Component1**.
- 4 Open the second OutPort dialog and change to the RTOS page.
- 5 Under Intertask communication, select **My_Message** from the **Message** list.
- 6 Select a component from the **Component** list, for example, **Msg1Data.Component2**, and click **Apply**.
TargetLink assigns the second signal to **Component2**.

Note

You do not have to repeat the above steps for the two InPort blocks of the second subsystem. TargetLink evaluates the connections of the OutPort and InPort blocks and uses the correct components at the receiver. If you additionally specify the components at the receiver's inports, you have to use the same components as at the sender's outports that they are connected to.

Result

You have now assigned the signals of the subsystems to specific message components of the **My_Variable** struct variable representing **My_Message**.

Related topics

Basics

[Message Basics..... 68](#)

References

[Bus Inport Block \(TargetLink Model Element Reference\)](#)
[Bus Outport Block \(TargetLink Model Element Reference\)](#)
[InPort Block \(TargetLink Model Element Reference\)](#)
[OutPort Block \(TargetLink Model Element Reference\)](#)

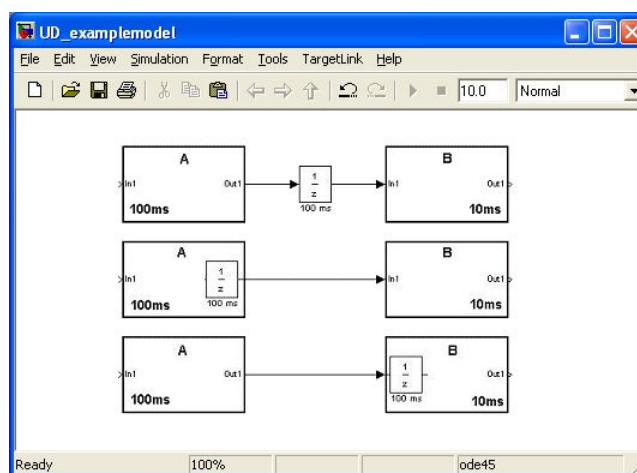
Detection of Unit Delay and Zero Order Hold Blocks for Rate Transition

Introduction

In contrast to the Rate Transition block, which is used solely for rate transition, the Unit Delay block and the Zero Order Hold block are used not only for rate transition but also for the control algorithm. TargetLink detects the purpose of these blocks in your multirate model. TargetLink detects the Unit Delay block as used for rate transition if the following requirements are fulfilled:

- The sample time of the Unit Delay block is identical to the sample time of the slower sending task.
- The sample time of the slower sending task is an integer multiple of the receiving task's sample time.

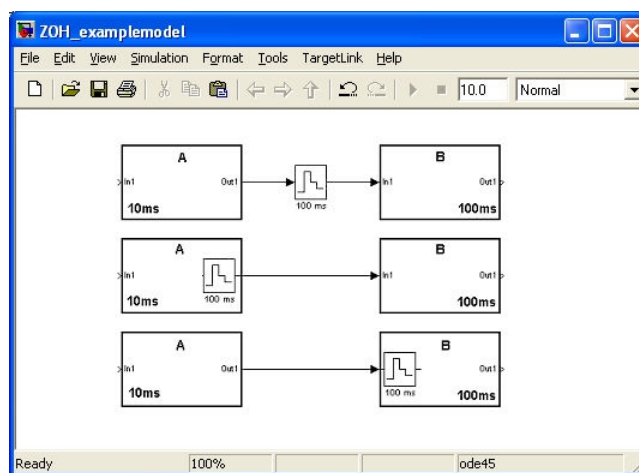
If both preconditions are fulfilled, TargetLink ignores the Unit Delay block. If they are not, the Unit Delay block is used as a real block in the control algorithm. As shown in the illustration below, it is irrelevant where the Unit Delay block is located.



TargetLink detects the Zero Order Hold block as used for rate transition if the following requirements are fulfilled:

- The sample time of the Zero Order Hold block is identical to the sample time of the slow receiver block.
- The sample time of the slow receiver block is an integer multiple of the sending task's sample time.

If both preconditions are fulfilled, TargetLink ignores the Zero Order Hold block. If they are not, the Zero Order Hold block is used as a real block in the control algorithm. As shown in the illustration below, it is irrelevant where the Zero Order Hold block is located.



Related topics

References

[Unit Delay Block](#) (TargetLink Model Element Reference)

Support of OSEK-Compliant RTOS

| | |
|---------------------------------------|---|
| Introduction | If you want to develop OSEK-compliant applications, you should know how TargetLink supports OSEK-specific objects, for example, OSEK events and alarms. |
| OSEK tasks | In modeling tasks for an OSEK-compliant application, the rules are basically the same as for generic multirate models. However, OSEK events provide extended options for task modeling. When generating production code for OSEK-compliant applications, TargetLink uses the OSEK syntax. |
| Setting up alarms | In generic RTOSs, you have to ensure that cyclic tasks are started with the correct sample time. TargetLink can start cyclic OSEK tasks automatically by setting up suitable alarms. It provides several ways to set them up. |
| Import and export of OIL files | You can import an OIL file and use the OSEK objects in TargetLink. You can also create OSEK objects in TargetLink and export them to an OIL file. |

Where to go from here

Information in this section

| | |
|--|---------------------|
| Introduction to TargetLink's OSEK Support..... | 88 |
| Modeling OSEK Tasks..... | 91 |
| Setting Up Alarms for Cyclic OSEK Task Activation..... | 108 |
| Importing and Exporting OIL Files..... | 117 |

Introduction to TargetLink's OSEK Support

Introduction

To use TargetLink's OSEK support, you must have purchased the TL_TMOS_OSEK license.

Where to go from here

Information in this section

| | |
|-----------------------------|----|
| Different RTOS Types..... | 88 |
| How to Select the RTOS..... | 90 |

Different RTOS Types

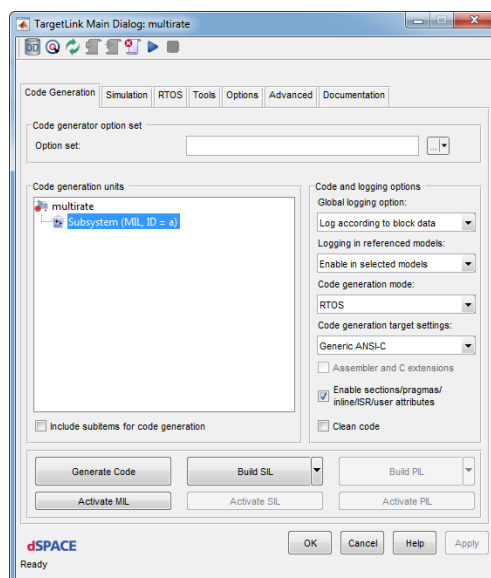
Introduction

TargetLink lets you select one of the following operating system types to generate production code for:

- Generic RTOSs (no TL_TMOS_OSEK license required)
- Generic OSEK RTOSs

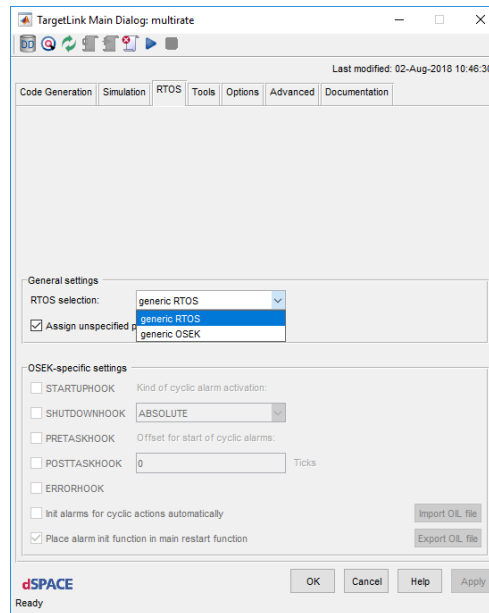
Enabling multirate code generation

To enable TargetLink to generate production code from your multirate model, you have to select RTOS from the Code generation mode drop-down list on the Code Generation page of the TargetLink Main Dialog.



For more information, refer to TargetLink Main Dialog Block.

You have to select the RTOS you use from the RTOS selection list on the RTOS page of the TargetLink Main Dialog:



Generic RTOS

If you want to generate production code for a real-time operating system which is not compliant with OSEK, you have to select **generic RTOS** from the RTOS selection drop-down list.

Generic OSEK RTOS

If you want to generate production code for a real-time operating system that is compliant with OSEK, you have to select **generic OSEK** from the RTOS selection drop-down list. In this case, you can use the following OSEK features:

- You can model OSEK tasks.
- You can create and start alarms for cyclic OSEK task activation automatically.
- You can use OSEK events, OSEK messages, OSEK resources, and the OSEK API for interrupt handling (DisableAllInterrupts, EnableAllInterrupts, etc.).
- You can use the Schedule block. Refer to [How to Insert a Schedule Command into a Non-Preemptive Task](#) on page 146.

For more information, refer to [Modeling OSEK Tasks](#) on page 91 and [Basics of Setting Up Alarms](#) on page 108.

Related topics**Basics**

| | |
|----------------------------------|-----|
| Basics of Setting Up Alarms..... | 108 |
| Modeling OSEK Tasks..... | 91 |

HowTos

| | |
|--|-----|
| How to Insert a Schedule Command into a Non-Preemptive Task..... | 146 |
|--|-----|

References

TargetLink Main Dialog Block ( [TargetLink Model Element Reference](#))

How to Select the RTOS

Objective

To generate production code which is adapted to your RTOS, you have to select the RTOS in the TargetLink Main Dialog.

Method**To select the RTOS**

- 1 On the Code Generation page of the TargetLink Main Dialog, select RTOS from the Code generation mode drop-down list.
- 2 Change to the RTOS page.
- 3 Under General Settings, select the RTOS you want to generate production code for from the RTOS selection drop-down list.

Result

You have now specified the RTOS used. During production code generation, TargetLink adapts the production code to the selected RTOS. For details on the adapted properties, refer to [Introduction to TargetLink's OSEK Support](#) on page 88.

Related topics**Basics**

| | |
|--|----|
| Introduction to TargetLink's OSEK Support..... | 88 |
|--|----|

References

TargetLink Main Dialog Block ( [TargetLink Model Element Reference](#))

Modeling OSEK Tasks

Introduction

Modeling tasks for an OSEK-compliant application offers more ways to combine atomic subsystems within each task than there are in applications for a generic RTOS. Tasks generated for an OSEK-compliant RTOS are called OSEK tasks below.

Where to go from here

Information in this section

| | |
|--|-----|
| Basics of Modeling OSEK Tasks..... | 91 |
| Grouping Different Noncyclic (Function Call) Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 92 |
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 102 |
| How to Assign an OSEK Event to a Stateflow chart..... | 104 |
| How to Map Stateflow Input Events to OSEK Events..... | 106 |

Basics of Modeling OSEK Tasks

Introduction

When generating production code for an OSEK-compliant RTOS, TargetLink creates a code frame for each task using the TASK keyword. The code frame for such an OSEK task is shown below:

```
TASK(<TaskIdentifier>)
{
    // insert generated code here
    TerminateTask();
}
```

Activating OSEK tasks

In an application for generic RTOSs, you have to write your own code to activate the tasks generated by TargetLink. For example, you have to complete the **TlTriggerTask()** macro to activate noncyclic tasks. To activate an OSEK task, TargetLink uses the OSEK API function **ActivateTask** in the generated production code.

Related topics

HowTos

| | |
|--|-----|
| How to Assign an OSEK Event to a Stateflow chart..... | 104 |
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 102 |
| How to Map Stateflow Input Events to OSEK Events..... | 106 |

References

Task Block (📖 TargetLink Model Element Reference)

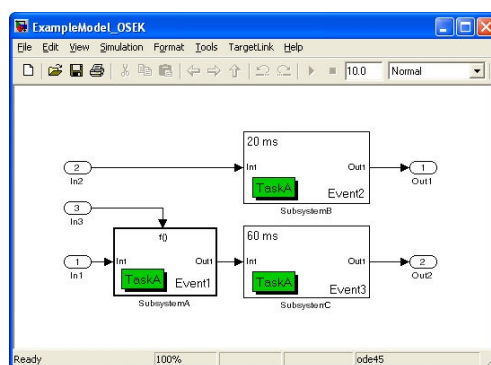
Grouping Different Noncyclic (Function Call) Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task

Introduction

If you group cyclic and noncyclic atomic subsystems in one OSEK task, you have to assign different OSEK events to the atomic subsystems to decouple them. If TargetLink detects such an OSEK task, it generates an endless loop within the task frame. Inside the loop, the task waits for the specified OSEK events. If an OSEK event occurs, the subsystem(s) associated with it is executed. Subsequently, the task again waits for OSEK events to occur.

Example

The following example shows a model containing one noncyclic and two cyclic atomic subsystems that are assigned to the same OSEK task. Each atomic subsystem is assigned to a different OSEK event.



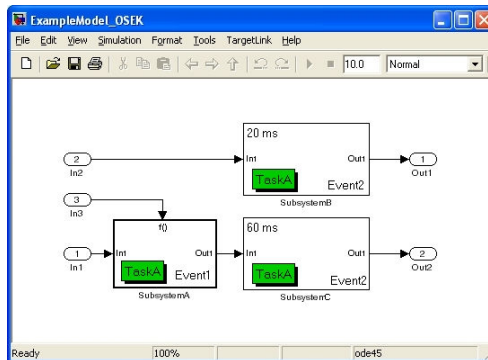
The model results in the following code:

```
TASK(A) {
    EventMaskType event;
    while (true) {
        WaitEvent(Event1 | Event2 | Event3);
        GetEvent(A, &event);
        ClearEvent(event);
        if(event & Event1) {
            SubsystemA();
        }
        if (event & Event2) {
            SubsystemB();
        }
        if (event & Event3) {
            SubsystemC();
        }
    }
    TerminateTask(A);
}
```

The same OSEK event can be assigned to cyclic atomic subsystems with different sample times. TargetLink then calculates the greatest common divisor of the sample times involved to execute each cyclic atomic subsystem with its specific sample time if the corresponding OSEK event occurs.

Example

The following example shows a model containing one noncyclic and two cyclic atomic subsystems that are assigned to the same OSEK task. The two cyclic atomic subsystems are assigned to the same OSEK event.

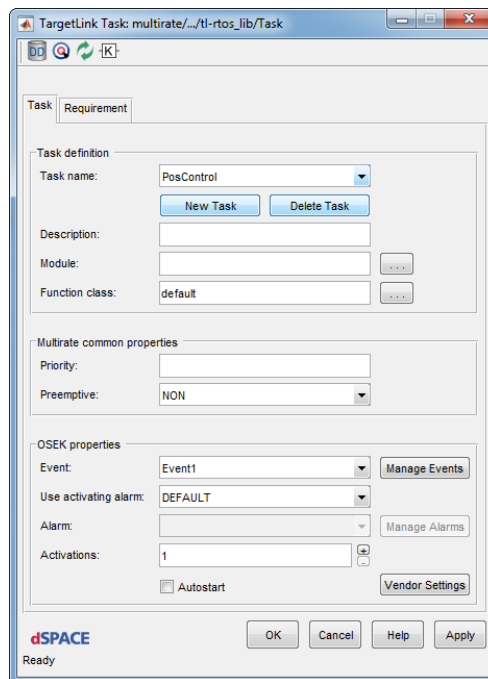


The model results in the following code:

```
TASK(A) {
    EventMaskType event;
    Int8 Cnt = 1;
    while (true) {
        WaitEvent(Event1 | Event2);
        GetEvent(A, &event);
        ClearEvent(event);
        if (event & Event1) {
            SubsystemA();
        }
        if (event & Event2) {
            SubsystemB();
            if (--Cnt == 0) {
                SubsystemC();
                Cnt = 3;
            }
        }
    }
}
TerminateTask(A);
}
```

Assigning OSEK events to subsystems

You can assign an OSEK event to an OSEK task using the Task block. The Task block offers an Event list you can select the desired OSEK event from.



Note

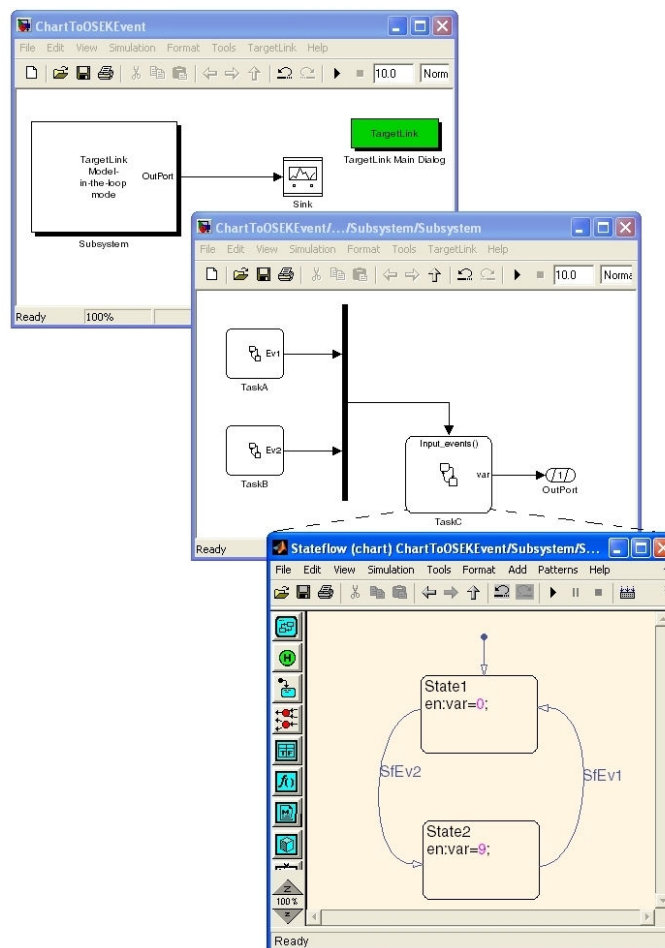
If you want to assign an OSEK event to an OSEK task or a specific atomic subsystem, the OSEK event must exist in the TargetLink Data Dictionary. If it does not, you have to create it. Refer to [How to Create an OSEK Event in the TargetLink Data Dictionary](#) on page 100.

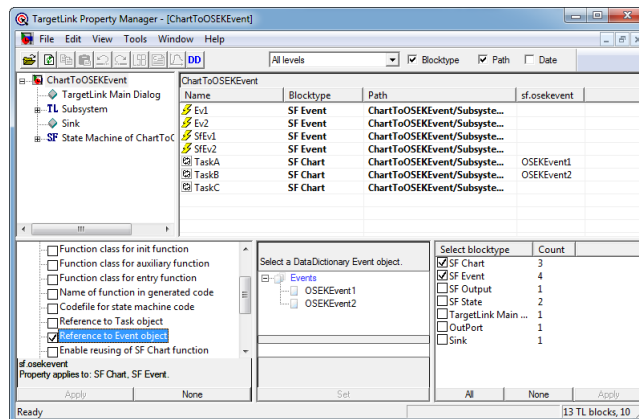
Assigning OSEK events to Stateflow charts

The rules for grouping atomic subsystems in an OSEK task also apply to Stateflow charts. If you want to assign an OSEK event to a Stateflow chart, you have to use the [Property Manager](#). For more information, refer to [How to Assign an OSEK Event to a Stateflow chart](#) on page 104. The Stateflow chart must be assigned to a task. For details, refer to [How to Generate Tasks from Stateflow charts](#) on page 46.

Example

The illustration below shows an example model and the resulting code.





The model results in the following code:

```
Bool TrigByTaskA;
Bool TrigByTaskB;
```

```
TASK(TaskC)
{
    EventMaskType events;
    while(1)
    {
        WaitEvent(OSEKEvent1)
        ClearEvent(OSEKEvent1)
        if(TrigByTaskA) {
            Chart(Ca3_SfEv1);
            TrigByTaskA = false;
        }
        if(TrigByTaskB) {
            Chart(Ca3_SfEv2);
            TrigByTaskB = false;
        }
    }
    TerminateTask();
}
```

```
TASK(TaskA)
{
    ...
    TrigByTaskA = true;
    SetEvent(OSEKEvent1)
    ...
}
```

```
TASK(TaskB)
{
    ...
}
```

```
TrigByTaskB = true;
SetEvent(OSEKEvent1)
...
}
```



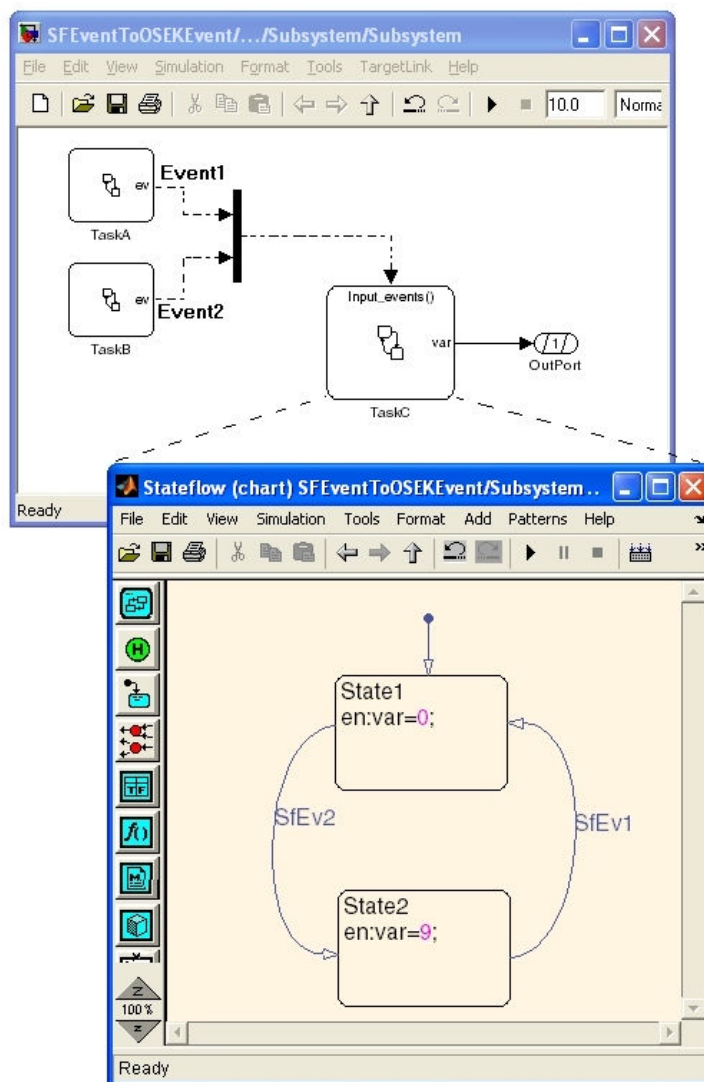
```
void Chart(UInt8 trigger_id)
{
    if (BFa7.Ca4_State1 ) {
        if (trigger_id == Ca3_SfEv2) {
            BFa7.Ca4_State1 = 0;
            BFa7.Ca5_State2 = 1;
            Ca3_var = 9.;
        }
    } else {
        if (BFa7.Ca5_State2) {
            if (trigger_id == Ca3_SfEv1) {
                BFa7.Ca5_State2 = 0;
                BFa7.Ca4_State1 = 1;
                Ca3_var = 0.;
            }
        } else {
            BFa7.Ca4_State1 = 1;
            Ca3_var = 0.;
        }
    }
}
```

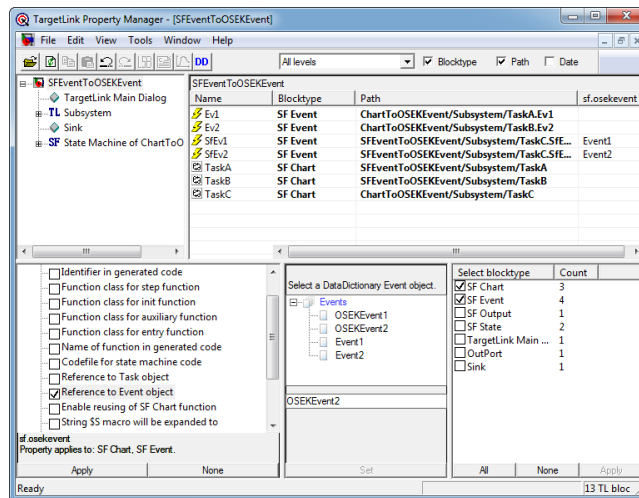
Assigning OSEK events to Stateflow input events

In addition to assigning an OSEK event to a complete Stateflow chart, you can map different OSEK events to different Stateflow input events. If you assign the OSEK events to the Stateflow input events, the OSEK events are evaluated in the chart function instead of the Stateflow input events.

Example

The illustration below shows an example model and the resulting code.





The model results in the following code:

```
TASK (TaskC){
    EventMaskType events;
    While(1){
        WaitEvent(Event1 | Event2);
        GetEvent(TaskC, &events);
        ClearEvent(events);
        if (events & Event1)
            Chart (Event1);
        if (events & Event2)
            Chart (Event2);
    }
}
```

The Stateflow chart results in the following code:

```
void Chart(EventMaskType mask){
    if (BFb7.Cb4_State1) {
        if (mask & Event2) {
            Bfb7.Cb4_State1 = 0;
            Bfb7.Cb5_State2 = 1;
            var = 9;
        }
    } else{
        if (BFb7.Cb5_State2){
            if (mask & Event1)
                Bfb7.Cb5_State2 = 0;
        }
        Bfb7.Cb4_State1 = 1;
        var = 0;
    }
}
```

You have to use the Property Manager to assign OSEK events to Stateflow input events. For detailed information, refer to [How to Map Stateflow Input Events to OSEK Events](#) on page 106 and to [Modifying Multiple Properties at Once via the Property Manager](#) ([TargetLink Preparation and Simulation Guide](#)).

The Stateflow input events are evaluated in the Stateflow chart function if they are not assigned to an OSEK event.

Related topics

Basics

[Modifying Multiple Properties at Once via the Property Manager](#) (📖 TargetLink Preparation and Simulation Guide)

HowTos

| | |
|--|---------------------|
| How to Assign an OSEK Event to a Stateflow chart..... | 104 |
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Generate Tasks from Stateflow charts..... | 46 |
| How to Map Stateflow Input Events to OSEK Events..... | 106 |

How to Create an OSEK Event in the TargetLink Data Dictionary

Objective

If you want to assign an OSEK event that does not yet exist in the TargetLink Data Dictionary to an atomic subsystem within an OSEK task, you have to create the OSEK event first.

Preconditions

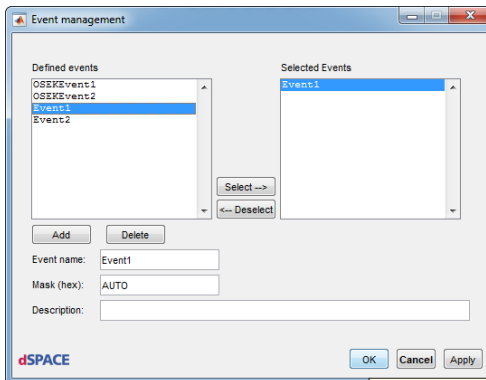
You must have opened a model containing an atomic subsystem with a Task block.

Method

To create an OSEK event in the TargetLink Data Dictionary

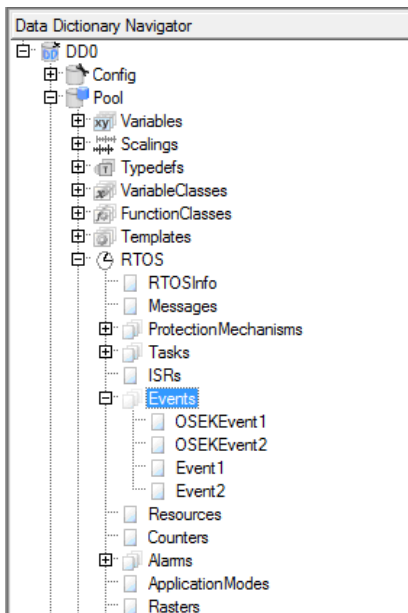
- 1 In your model, double-click the atomic subsystem containing a Task block to open it.
- 2 Double-click the Task block to open the Task dialog.

- 3 Under OSEK properties, click Manage Events to open the Event Management dialog.



- 4 Click Add and enter the name and the description in the corresponding edit fields.
- 5 Click Apply.

TargetLink generates a new OSEK event with the specified properties in the DataDictionary/Pool/RTOS/Events subnode of the TargetLink Data Dictionary.



The OSEK event also appears in the Defined events list of the Event Management dialog.

- 6 In the Defined events list, select the desired OSEK event and click Select to copy it to the Selected events list.
- 7 Click Apply.

Result You have now created a new OSEK event in the TargetLink Data Dictionary. The OSEK events you copied from the Defined events list to the Selected events list are assigned to the task. They also appear in the Events list in the Task dialog.

Next step After you create a new OSEK event in the TargetLink Data Dictionary, you can assign it to a specific atomic subsystem within an OSEK task. Refer to [How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task](#) on page 102.

Related topics

Basics

[Basics of Modeling OSEK Tasks](#)..... 91

HowTos

[How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task](#)..... 102

References

[Task Block](#) ( TargetLink Model Element Reference)

How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task

Objective If you want to group cyclic and noncyclic subsystems in one OSEK task, you have to assign different OSEK events to them to decouple them.

Preconditions

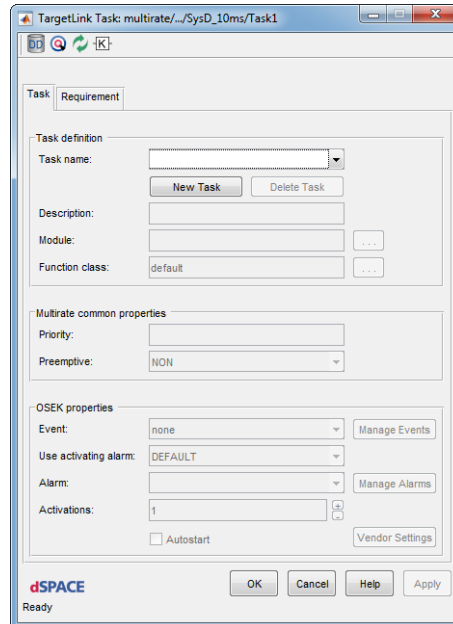
The following preconditions must be fulfilled to perform the steps below:

- You must have opened a model containing a cyclic and a noncyclic atomic subsystem.
- Two different OSEK events must exist in the TargetLink Data Dictionary. Refer to [How to Create an OSEK Event in the TargetLink Data Dictionary](#) on page 100.
- The TargetLink Data Dictionary must contain a task the OSEK events are assigned to. Refer to [How to Group Atomic Subsystems in Tasks Using the Task Block](#) on page 38.

Method

To group noncyclic atomic subsystems or noncyclic and cyclic atomic subsystems in one OSEK task

- 1 Double-click the noncyclic atomic subsystem to open it.
- 2 From the RTOS Blocks library, copy one Task block to the atomic subsystem and double-click it to open the Task dialog.



- 3 Under OSEK Properties, select the desired OSEK event, for example, Event1, from the Event list, and click Apply.
You have now assigned an OSEK event to the noncyclic atomic subsystem.
The next step is to assign another OSEK event to the cyclic atomic subsystem.
- 4 Open the cyclic atomic subsystem and copy one Task block from the RTOS Blocks library to it.
- 5 Double-click the Task block to open the Task dialog.
- 6 Under Task Definition, select the same task from the Task Name list as in the Task block residing in the noncyclic atomic subsystem.
- 7 Under OSEK Properties, select a different OSEK event, for example, Event2, from the Event list, and click Apply.

Result

You have now combined a noncyclic and a cyclic atomic subsystem in one OSEK task and decoupled them using different OSEK events.

Related topics**Basics**

[Modeling OSEK Tasks..... 91](#)

HowTos

[How to Create an OSEK Event in the TargetLink Data Dictionary..... 100](#)
[How to Group Atomic Subsystems in Tasks Using the Task Block..... 38](#)

References

[Task Block](#) ( [TargetLink Model Element Reference](#))


How to Assign an OSEK Event to a Stateflow chart

Objective


If you want to assign an event to a Stateflow chart, you have to use the [Property Manager](#).

Preconditions

To perform the following steps, several preconditions must be fulfilled:


- You must have opened a model containing a Stateflow chart.
- You must have assigned the Stateflow chart to a task. Refer to [How to Generate Tasks from Stateflow charts](#) on page 46.
- The TargetLink Data Dictionary must contain a task that the OSEK events are assigned to. Refer to [How to Create an OSEK Event in the TargetLink Data Dictionary](#) on page 100 and [How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task](#) on page 102.
- You should be familiar with the Property Manager. Refer to [Modifying Multiple Properties at Once via the Property Manager](#) ( [TargetLink Preparation and Simulation Guide](#)).

Method**To assign an OSEK event to a Stateflow chart**

- 1** In the MATLAB Command Window, type `tlPropman('Start')` to open the TargetLink Property Manager.
- 2** In the Model Navigator of the TargetLink Property Manager, select the Stateflow chart.
- 3** In the Property View, add the Event column via the [Column Chooser Dialog](#) ( [TargetLink Tool and Utility Reference](#)) if it is not already displayed.
- 4** In the Property View, select the Event cell of the Stateflow chart and click the Browse button.

The DD Reference Selection dialog opens and the available OSEK events in the Pool1/RTOS/Events subnode of the TargetLink Data Dictionary are displayed.

- 5 Select the OSEK event you want to assign and click **Ok**.

For more information, refer to [DD Reference Selection Dialog \(Property Manager\)](#) ( [TargetLink Tool and Utility Reference](#)).

Note


The Property Manager shows all OSEK events in the TargetLink Data Dictionary. You must ensure you select only OSEK events that are assigned to the task concerned.

Result

You have assigned an OSEK event to the Stateflow chart. If the OSEK event occurs, the code resulting from the Stateflow chart is executed.

Related topics

Basics

| | |
|--|----|
| Modeling OSEK Tasks..... | 91 |
| Modifying Multiple Properties at Once via the Property Manager ( TargetLink Preparation and Simulation Guide) | |

HowTos

| | |
|--|-----|
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Generate Tasks from Stateflow charts..... | 46 |
| How to Group Noncyclic Atomic Subsystems or Noncyclic and Cyclic Atomic Subsystems in One OSEK Task..... | 102 |

References

| |
|--|
| DD Reference Selection Dialog (Property Manager) ( TargetLink Tool and Utility Reference) |
| tlPropman ( TargetLink API Reference) |

How to Map Stateflow Input Events to OSEK Events

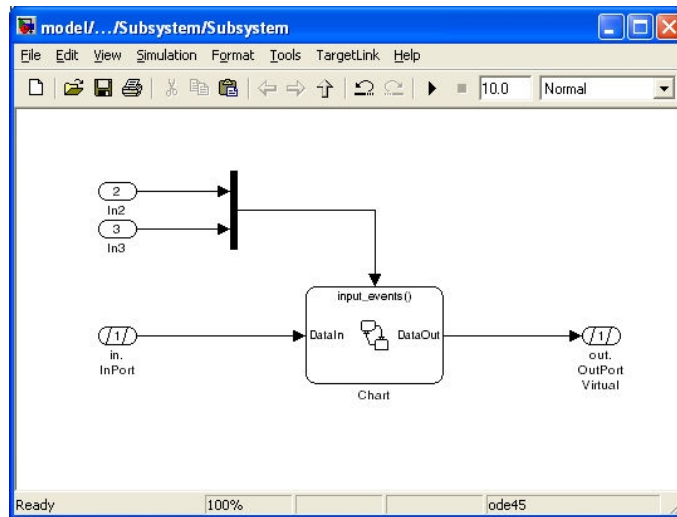
Objective

As described above, TargetLink supports a method of assigning different OSEK events to different Stateflow input events.

Preconditions

To perform the following steps, several preconditions must be fulfilled:

- You must have opened a model containing a Stateflow chart with two or more Stateflow input events. The model used in the example below contains two Stateflow input events.



- You must have assigned the Stateflow chart to a task. Refer to [How to Generate Tasks from Stateflow charts](#) on page 46.
- The OSEK events you want to map to the Stateflow input events must exist in the TargetLink Data Dictionary. Refer to [How to Create an OSEK Event in the TargetLink Data Dictionary](#) on page 100.
- The TargetLink Data Dictionary must contain a task that the OSEK events are assigned to. Refer to [How to Group Atomic Subsystems in Tasks Using the Task Block](#) on page 38.
- You should be familiar with the TargetLink [Property Manager](#). Refer to [Modifying Multiple Properties at Once via the Property Manager](#) ([TargetLink Preparation and Simulation Guide](#)).

Method


To map Stateflow input events to OSEK events

- 1 In the MATLAB Command Window, type `tlPropman('Start')` to open the TargetLink Property Manager.
- 2 In the Model Navigator of the TargetLink Property Manager, select the Stateflow chart.
- 3 In the Property View, add the Event column via the [Column Chooser Dialog](#) ([TargetLink Tool and Utility Reference](#)) if it is not already displayed.

- 4 In the Property View, select the Event cell of the first Stateflow input event and click the Browse button.

The DD Reference Selection dialog opens and the available OSEK events in the **Pool1/RTOS/Events** subnode of the TargetLink Data Dictionary are displayed.

- 5 Select the OSEK event you want to assign and click Ok.

For more information, refer to [DD Reference Selection Dialog \(Property Manager\)](#) ( [TargetLink Tool and Utility Reference](#)).

You have assigned an OSEK event to the first Stateflow input event. If the OSEK event occurs, the corresponding transitions in the Stateflow chart are executed. The next step is to assign another OSEK event to another Stateflow input event.

- 6 Repeat steps 3 to 5 for the next Stateflow input event.

Result

You have now mapped different Stateflow input events to different OSEK events.

Related topics

Basics

[Modeling OSEK Tasks](#)..... 91
[Modifying Multiple Properties at Once via the Property Manager](#) ( [TargetLink Preparation and Simulation Guide](#))

HowTos

[How to Create an OSEK Event in the TargetLink Data Dictionary](#)..... 100
[How to Generate Tasks from Stateflow charts](#)..... 46
[How to Group Atomic Subsystems in Tasks Using the Task Block](#)..... 38

References

[DD Reference Selection Dialog \(Property Manager\)](#) ( [TargetLink Tool and Utility Reference](#))
[tlPropman](#) ( [TargetLink API Reference](#))

Setting Up Alarms for Cyclic OSEK Task Activation

Introduction

In generic RTOSs, you have to ensure that cyclic tasks are started with the correct sample time. TargetLink can start cyclic OSEK tasks automatically by setting up suitable alarms. It provides several ways to set up OSEK alarms.

Where to go from here

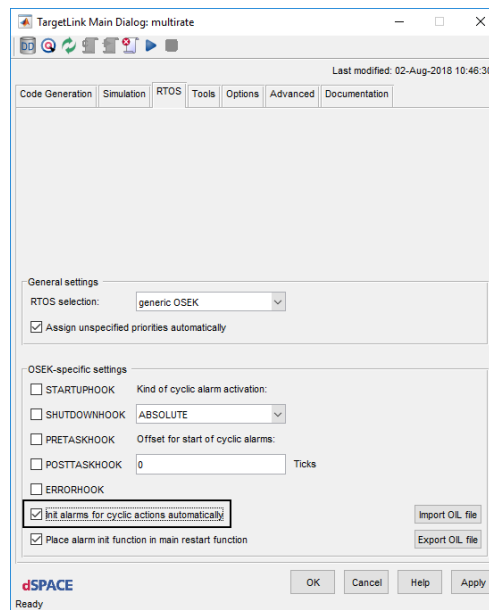
Information in this section

| | |
|---|-----|
| Basics of Setting Up Alarms..... | 108 |
| How to Create an Alarm in the TargetLink Data Dictionary..... | 113 |
| How to Set Up Alarms for Cyclic OSEK Tasks Automatically..... | 115 |

Basics of Setting Up Alarms

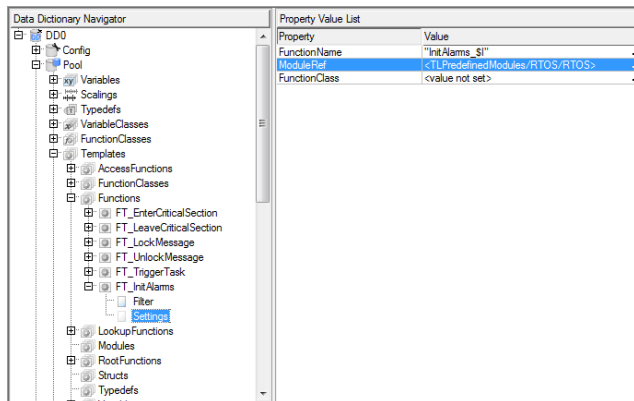
Introduction

If you want TargetLink to set up alarms for cyclic task activation automatically, you have to select the Init alarms for cyclic actions automatically checkbox on the RTOS page of the TargetLink Main Dialog.

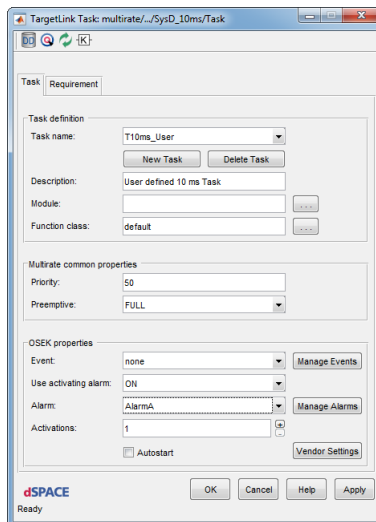


TargetLink then generates the InitAlarms function into the production code. The InitAlarms function starts the alarms for cyclic tasks and events. You can specify the name of the InitAlarms function in the

DataDictionary/Pool/Templates/Functions/FT_InitAlarms/Settings
subnode of the Data Dictionary Manager as shown below:



You can specify the alarm you want to use to activate a task or to set an event with the Use activating alarm and the Alarm properties in each Task block and in each root function template.



The settings of these properties are valid only if you selected the Init alarms for cyclic actions automatically checkbox on the RTOS page.

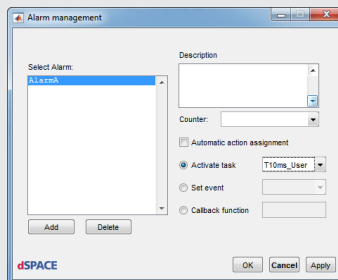
You can choose the following settings from the **Use activating alarms** list in the Task block:

| Setting | Description |
|---------|---|
| DEFAULT | TargetLink searches for an alarm with a suitable action in the DataDictionary/Pool/RTOS/Alarms subnode. If TargetLink does not find an alarm with a suitable action, it creates one in the DataDictionary/Subsystems subnode. TargetLink generates production code that starts the alarm appropriately. |
| ON | TargetLink uses the alarm referenced in the Alarm list of the Task block. The action that is assigned to this alarm must match the task to be activated or to the OSEK event to be set or the Automatic action assignment property must be selected for the alarm. Refer to How to Create an Alarm in the TargetLink Data Dictionary on page 113. |
| OFF | TargetLink does not start an alarm for the selected task or OSEK event, even if the Init alarms for cyclic actions automatically checkbox on the RTOS page is selected. This also applies if an alarm with a suitable action exists in the DataDictionary/Pool/RTOS/Alarms subnode. |

Note

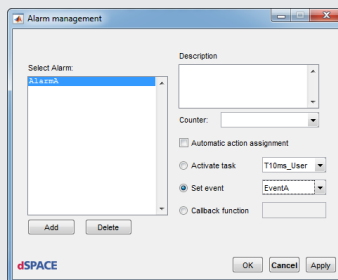
You can assign an action to an OSEK alarm in the Alarm Management dialog (refer to [Alarm Management Dialog](#) ([TargetLink Tool and Utility Reference](#))). An action is suitable for activating a specific task if the following preconditions are fulfilled:

- The selected action is **Activate task**.
- The task to be activated is selected in the list.



An action is suitable for setting an event if the following preconditions are fulfilled:

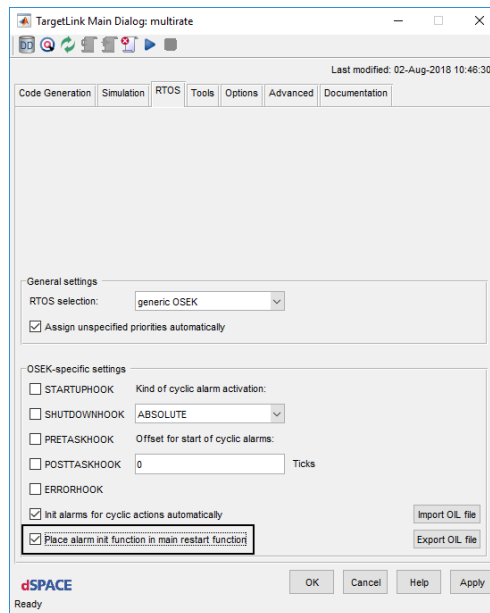
- The selected action is **Set event**.
- The OSEK event to be set is selected in the list.
- The task the OSEK event is assigned to is selected in the list.



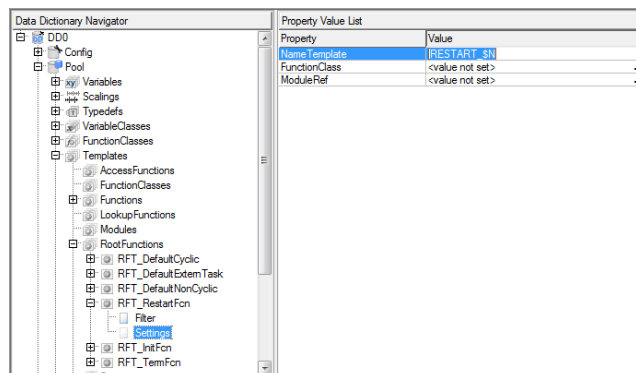
For more information, refer to [How to Create an Alarm in the TargetLink Data Dictionary](#) on page 113.

Starting alarms in the restart function

You can choose to place the InitAlarms function in the restart function during production code generation. To do so, you have to select the Place alarm init function in main restart function checkbox on the RTOS page of the TargetLink Main Dialog.



The restart function calls the InitAlarms function, if the corresponding checkbox is selected. Additionally, it initializes all variables that are not initialized during variables definition. You can specify the name of the restart function in the **DataDictionary/Pool/Templates/RootFunctions/RFT_RestartFcn/Settings** subnode of the Data Dictionary Manager as shown in the illustration below:



Kind of cyclic alarm activation

You can specify whether the OSEK API function `SetAbsAlarm` or `SetRelAlarm` is used. For more information on the OSEK API functions, refer to the OSEK specification.

Offset for start of cyclic alarms

The offset for starting cyclic alarms is added to the Simulink sample time offset for all OSEK alarm activations. As a result, the start of all cyclic tasks is delayed in the RTOS application by the following value:

$\text{TickdurationOfSystemTimer} * \text{AdditionalOffset}$

This is useful for synchronizing cyclic tasks.

Related topics

HowTos

[How to Create an Alarm in the TargetLink Data Dictionary..... 113](#)

References

[Task Block \(TargetLink Model Element Reference\)](#)

How to Create an Alarm in the TargetLink Data Dictionary

Objective

If you want to select an alarm in the Task block dialog to activate an OSEK task cyclically, the alarm must exist in the TargetLink Data Dictionary. If it does not, you have to create it first.

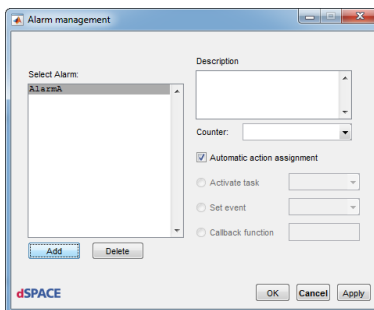
Preconditions

To perform the following steps, you must have opened a model containing an atomic subsystem with a Task block.

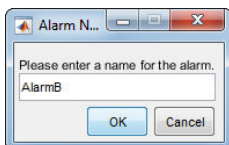
Method

To create an alarm in the TargetLink Data Dictionary

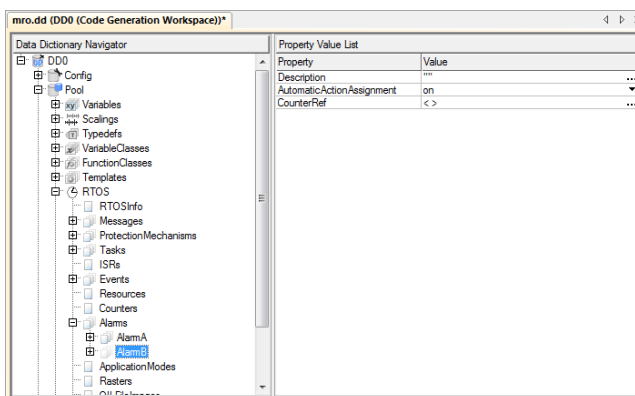
- 1 In your model, open the subsystem containing the Task block.
- 2 Double-click the Task block to open the Task dialog.
- 3 Under OSEK Properties, click Manage Alarms to open the Alarm management dialog.



- 4 Click Add to open the Alarm Name dialog.



- 5 Enter a name for the new alarm and click OK.
The new alarm appears in the Select Alarm list in the Alarm management dialog.
- 6 Specify a counter for the alarm. The alarm's sample time and sample time offset must be an integer multiple of the counter's tick duration.
- 7 Click OK to close the Alarm management dialog.
The new alarm appears in the Alarm list in the Task dialog and in the DataDictionary/Pool/RTOS/Alarms subnode in the TargetLink Data Dictionary.



Result

You have now created a new alarm in the TargetLink Data Dictionary. You can use it for cyclic task activation.

Related topics

Basics

[Setting Up Alarms for Cyclic OSEK Task Activation..... 108](#)

References


[Task Block \(TargetLink Model Element Reference\)](#)

How to Set Up Alarms for Cyclic OSEK Tasks Automatically

| | |
|-------------------------|--|
| Objective | TargetLink allows you to set up alarms for cyclic OSEK task activation automatically. |
| Preconditions | To perform the following steps, you must have opened a model containing an atomic subsystem with a Task block and a TargetLink Main Dialog. |
| Possible methods | <p>TargetLink provides two methods to set up alarms for cyclic OSEK task activation automatically.</p> <ul style="list-style-type: none"> ▪ You can assign a specific alarm to an OSEK task. Refer to Method 1 on page 115. ▪ You can have TargetLink select an alarm for cyclic OSEK task activation. Refer to Method 2 on page 115. |
| Method 1 | <p>To assign a specific alarm to an OSEK task</p> <ol style="list-style-type: none"> 1 Open the TargetLink Main Dialog and change to the RTOS page. 2 Select the Init alarms for cyclic actions automatically checkbox. 3 In your model, double-click a Task block to open the Task dialog. 4 Under Task Definition, select the task you want to be activated cyclically from the Task Name list. 5 Under OSEK Properties, select ON from the Use activating alarms list. 6 Select the desired alarm from the Alarm list. 7 Click Apply. |
| Method 2 | <p>To have TargetLink select an alarm for cyclic OSEK task activation</p> <ol style="list-style-type: none"> 1 Open the TargetLink Main Dialog and change to the RTOS page. 2 Select the Init alarms for cyclic actions automatically checkbox. 3 In your model, double-click a Task block to open the Task dialog. 4 Under OSEK Properties, select DEFAULT from the Use activating alarms list. |
| Result | When generating production code, TargetLink sets up all alarms for cyclic OSEK task activation automatically. |

Related topics

References

[TargetLink Main Dialog Block](#) ( [TargetLink Model Element Reference](#))
[Task Block](#) ( [TargetLink Model Element Reference](#))

Importing and Exporting OIL Files

OSEK Implementation Language

All OSEK system objects such as tasks, events and alarms are defined in a special description language called OSEK Implementation Language (OIL). The file containing the description of the objects (OIL file) is converted into C code by an OSEK vendor tool.

You can import an existing OIL file and use the OSEK objects in TargetLink. You can also create OSEK objects in TargetLink and export them to an OIL file.

Where to go from here

Information in this section

| | |
|--|---------------------|
| OIL File Basics..... | 117 |
| OIL File Import and Export Instructions..... | 121 |

OIL File Basics

Where to go from here

Information in this section

| | |
|--|---------------------|
| Basics on OIL Files..... | 117 |
| OIL File Import..... | 119 |
| OIL File Export..... | 119 |

Basics on OIL Files

Portable software

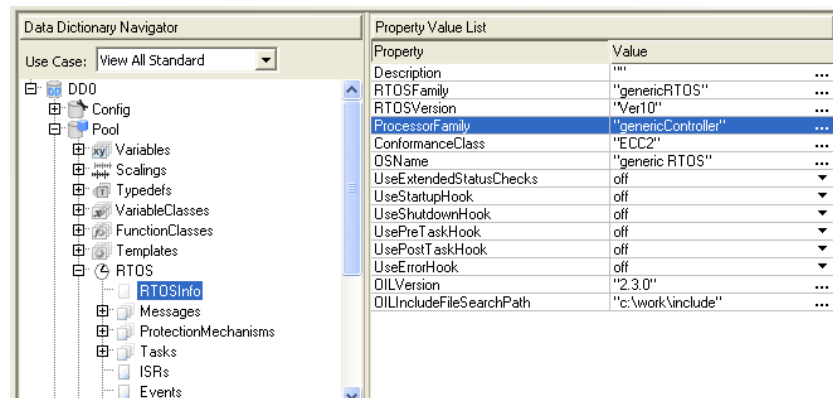
To achieve the OSEK objective of portable software, a way has been defined to describe the configuration of an application using OSEK.

The OSEK Implementation Language (OIL) provides a mechanism to configure an OSEK application for a particular ECU. That means that all OSEK objects, for example, tasks and messages with their properties such as the task priorities, are listed in an OIL configuration file.

An OIL file is divided into an implementation-specific part and an application-specific part. The implementation-specific part is always supplied with the OSEK implementation and must not be changed. The application-specific part of the

OIL file is defined by the user during application development. It consists of objects corresponding to the objects in the OSEK-compliant operating system. Each object is described by a set of attributes. Some attributes are mandatory, for example, a task's priority, and some are optional.

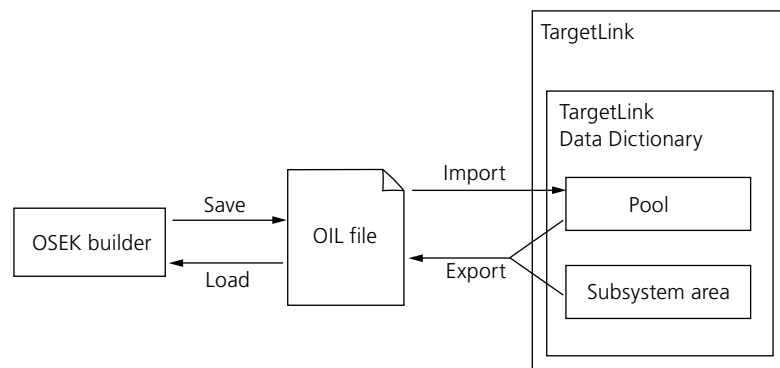
Like a C file, an OIL file can include further OIL files. To find an included OIL file, TargetLink needs to know the folder the included OIL files reside in. You must enter the path in the `OILIncludeFileSearchPath` edit field in the `DataDictionary/Pool/RTOS/RTOSInfo` subnode in the Data Dictionary Manager.



You can enter several search paths separated by semicolons in the same way as with the DOS environment variable PATH.

OIL file import and export

TargetLink allows you to import and export OIL files. The following illustration shows how TargetLink handles the import and export of OIL files:



OIL File Import

Different import modes

An OIL file import stores the OIL objects in the Pool subnode of the TargetLink Data Dictionary. To import an OIL file, you can select one of the following import modes:

- Overwrite mode
- Merge mode

Overwrite mode

In the overwrite mode, all OSEK objects, for example, tasks, events and resources in the TargetLink Data Dictionary, which are not in the imported OIL file are deleted. If an object in the OIL file already exists in the TargetLink Data Dictionary, all properties which are unspecified in the OIL object remain unchanged in the TargetLink Data Dictionary.

Merge mode

In the merge mode, all OSEK objects in the TargetLink Data Dictionary which do not exist in the imported OIL file remain unchanged. If an object which must be imported already exists in the TargetLink Data Dictionary, any of its properties that are not specified in the OIL object remain unchanged in the TargetLink Data Dictionary. For details on how to import an OIL file, refer to [OIL File Import and Export Instructions](#) on page 121.

Related topics

HowTos

[OIL File Import and Export Instructions..... 121](#)

OIL File Export

Different OIL versions

An OIL file export writes the properties specified in the TargetLink Data Dictionary to an OIL file. The current version of TargetLink lets you select one of the following versions of the OSEK Implementation Language Specification:

- V2.2
- V2.3

Export from Pool area

If you export an OIL file from the Pool area, TargetLink reads the OIL objects from the DD Pool area and writes them to the OIL file. The Pool area stores the OSEK objects which have been imported from an OIL file or which you have created during model creation. This is useful if all OSEK objects and their properties are user-defined or specified by importing an OIL file. OSEK objects

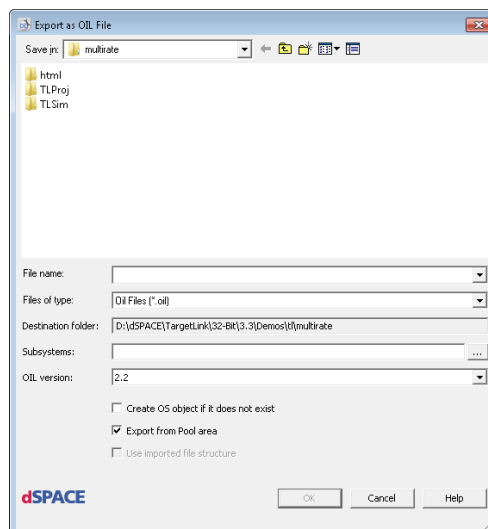
TargetLink generates automatically in the Subsystems subnodes are not written to the OIL file.

Export from Subsystems area

If you export an OIL file from the Subsystems area, TargetLink reads the OIL objects from the Subsystems area and stores them in the OIL file. The Subsystems area stores OSEK objects which are used by the generated code of a specific TargetLink Subsystem. It is possible to export an OIL file from more than one DD Subsystem object. This is useful if all OSEK objects are created by TargetLink during production code generation. In some cases, TargetLink may create only a part of an OIL file which is included in another OIL file. If it does, objects like the system timer may exist in the DD Pool area subnode. They must not be written to the OIL file because they already exist in the master OIL file.

Export from Pool and Subsystems areas

You can export an OIL file from the Pool subnode and one or more Subsystems subnodes. If you do, the objects in the Pool subnode are merged with the objects in the Subsystems subnode. If object properties have different values in the Pool and the Subsystems subnodes, TargetLink uses the values of the Subsystems subnodes. This is useful, if the Pool subnode contains OIL objects that are user-defined or imported with an OIL file, and the Subsystems subnode contains OIL objects that are created by TargetLink. The following illustration shows the properties you can set in the Export as OIL file dialog.



Related topics

References

[Export as OIL File](#) (📖 TargetLink Data Dictionary Manager Reference)

OIL File Import and Export Instructions

| | |
|------------------|--|
| Objective | You should be familiar with methods of importing OSEK objects from or exporting OSEK objects to an OIL file. |
|------------------|--|

| | |
|------------------------------|------------------------------------|
| Where to go from here | Information in this section |
|------------------------------|------------------------------------|

| | |
|--|--|
| | How to Import an OIL File..... 121 |
| | How to Export an OIL File..... 123 |

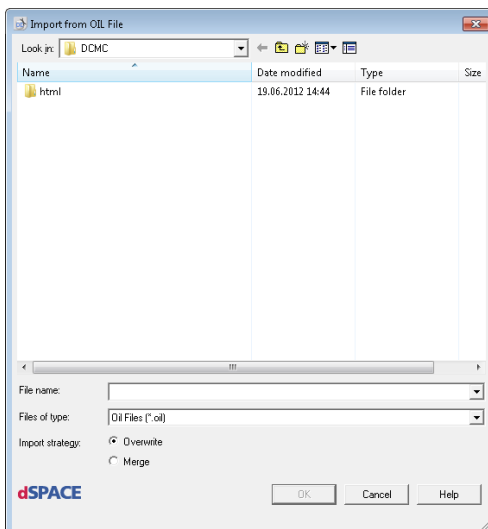
How to Import an OIL File

| | |
|------------------|---|
| Objective | If you want to use OSEK objects which are defined in an existing OIL file in TargetLink, you have to import the OIL file. |
|------------------|---|

| | |
|----------------------|--|
| Preconditions | <p>To import an OIL file, the following preconditions must be fulfilled:</p> <ul style="list-style-type: none">▪ You must have purchased the OSEK license.▪ You must have opened a TargetLink model.▪ An OIL file must exist on your PC. |
|----------------------|--|

Method**To import an OIL file**

- 1 Open the TargetLink Main Dialog and change to the RTOS page.
- 2 Under OSEK specific settings, click Import OIL file to open the following dialog.



- 3 Select the OIL file you want to import.
- 4 Select the Import strategy you want to use.

Result

TargetLink imports the selected OIL file to the Pool subnode of the TargetLink Data Dictionary.

Related topics**Basics**

[OIL File Basics..... 117](#)

HowTos

[How to Export an OIL File..... 123](#)

References

[TargetLink Main Dialog Block \(📖 TargetLink Model Element Reference\)](#)

How to Export an OIL File

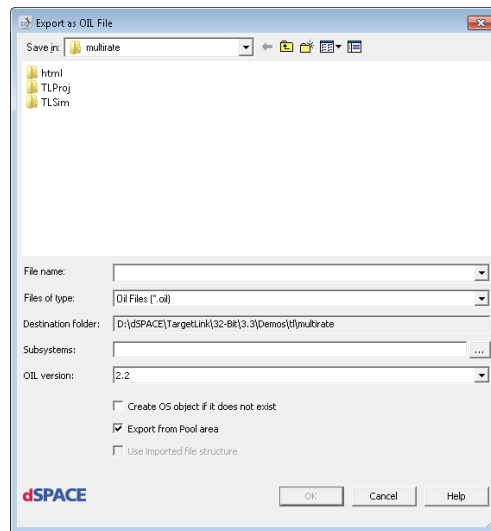
Objective

If you want to use OSEK objects which you have created in TargetLink to build an executable for your RTOS, you have to export an OIL file.

Method

To export an OIL file

- 1 On the RTOS page of the TargetLink Main Dialog, click Export OIL file to open the following dialog:



- 2 If you want to update an OIL file you previously imported, select the Use imported file structure checkbox and proceed with step 5.
- 3 In Save in specify the destination folder to export the OIL file to.
- 4 In the File name edit field, enter a name for the OIL file you want to export.
- 5 In the Subsystems edit field specify the subsystem(s) you want to export from via the Browse button.

Note

If you leave the Subsystems edit field empty, TargetLink exports the OIL file only from the Pool subnode.


- 6 Select the Export from Pool area checkbox.
- 7 From the OIL Version list, select the OIL version you want to use.
- 8 Click OK.

Result



TargetLink exports/updates an OIL file.

Related topics

Basics

Exchanging OIL Files ( TargetLink Interoperation and Exchange Guide)
OIL File Basics..... 117

References

Export as OIL File ( TargetLink Data Dictionary Manager Reference)
TargetLink Main Dialog Block ( TargetLink Model Element Reference)

Working with Special RTOS Blocks

Introduction

When creating multirate models for multitasking applications, you can use special blocks for special multirate purposes. TargetLink provides the RTOS Blocks library, containing blocks to specify multitasking applications.

Special RTOS blocks

The RTOS Blocks library is divided into generic multitasking blocks and blocks you can use only for OSEK models.

The following blocks can be used if code is generated for generic RTOSs and for OSEK-compliant RTOSs:

- Task block
- ISR block
- Critical Section block

For code generation for OSEK-compliant RTOSs, TargetLink additionally provides the following two blocks:

- CounterAlarm block
- Schedule block

You will find a description of how to work with the RTOS blocks below. The use of the Task block and the ISR block is described in detail in [Creating Tasks and Interrupt Service Routines](#) on page 23.

Where to go from here

Information in this section

| | |
|---|---------------------|
| Protecting Critical Sections..... | 126 |
| Setting up OSEK Alarms..... | 131 |
| Interrupting a Non-Preemptive Task..... | 146 |

Protecting Critical Sections

Introduction

Some subsystems can be invoked by several tasks, for example, function-call-triggered subsystems. This can lead to unpredictable results if one instance of the corresponding function is interrupted by a second instance of the same function, and if the function uses RAM variables to store intermediate results. Systems that access memory, devices etc. that are commonly used by several subsystems from different tasks might also need protection from being interrupted. To protect an atomic subsystem as a critical section, you must use the Critical Section block.

Where to go from here

Information in this section

| | |
|---|---------------------|
| How to Protect an Atomic Subsystem as a Critical Section..... | 126 |
| How to Create a New Protection Mechanism..... | 128 |
| How to Specify the Properties of a Protection Mechanism..... | 129 |

How to Protect an Atomic Subsystem as a Critical Section

Objective

With TargetLink you can mark a single atomic subsystem as a critical section to have it protected from being interrupted during execution. This section describes how to mark a subsystem as a critical section.

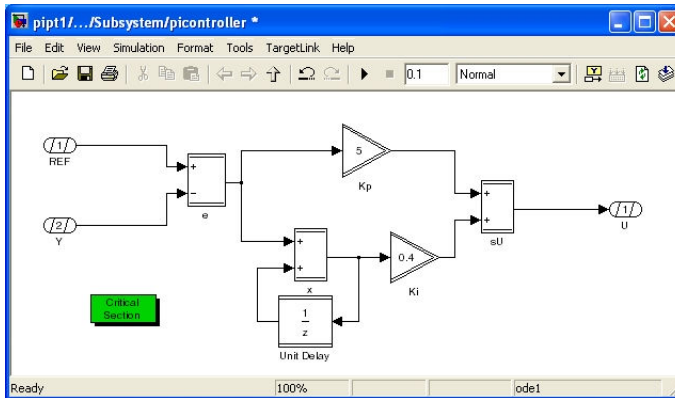
Preconditions

You must have opened a model containing an atomic subsystem.

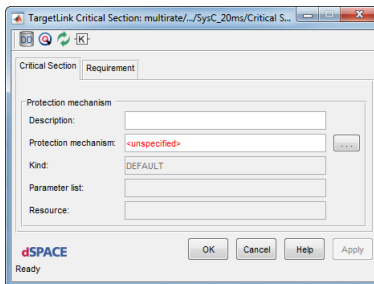
Method

To protect an atomic subsystem as a critical section

- 1 In your model, double-click the desired atomic subsystem to open it.
- 2 Copy one Critical Section block from the RTOS library to the atomic subsystem.



- 3 Double-click the Critical Section block to open the Critical Section dialog.



- 4 Under Critical Section Options, select a protection mechanism from the Protection Mechanism list, for example, select SuspendOSInterrupts.
- 5 Click OK to close the dialog.

Result

During production code generation, TargetLink generates the following API function calls into the production code:

```
SuspendOSInterrupts();
// code for protected subsystem
ResumeOSInterrupts();
```

The macros protect the subsystem from being interrupted during execution.

TargetLink provides several predefined protection mechanisms, for example, SuspendOSInterrupts. Each protection mechanism uses a particular kind of protection, for example, SUSPEND_OS_INTERRUPTS. Different protection mechanisms can use the same kind.

For details, refer to:

- [Critical Section Block](#) ([TargetLink Model Element Reference](#))
- [How to Create a New Protection Mechanism](#) on page 128

Related topics

HowTos

[How to Create a New Protection Mechanism](#)..... 128

How to Create a New Protection Mechanism

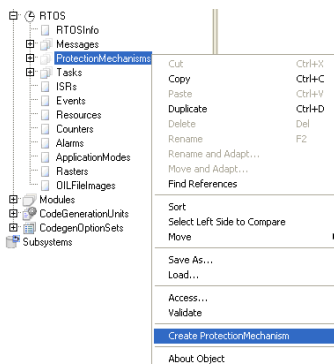
Objective

Instead of using one of the predefined TargetLink protection mechanisms, you might want to specify your own. To do so, you have to add a new protection mechanism in the Data Dictionary Manager and specify its properties.

Method

To create a new protection mechanism

- 1 In the MATLAB Command Window, type `dsddman` to open the Data Dictionary Manager.
- 2 In the Data Dictionary Navigator, change to the `DataDictionary/Pool/RTOS/ProtectionMechanisms` subnode.
- 3 Right-click the `DataDictionary/Pool/RTOS/ProtectionMechanisms` subnode and select `Create Protection Mechanism` from the context menu.



The Data Dictionary Manager creates a new subnode.

- 4 In the new subnode, enter a name for the new protection mechanism, for example, `PROT_ADChannel11`.

Result The Data Dictionary Manager creates a new protection mechanism with the specified name in the `DataDictionary/Pool/RTOS/Protection Mechanism` subnode. You can reference this protection mechanism in the Critical Section dialog.

Next step After you create the new protection mechanism, the next step is to specify its properties. Refer to [How to Specify the Properties of a Protection Mechanism](#) on page 129.

Related topics

HowTos

[How to Specify the Properties of a Protection Mechanism..... 129](#)

References

[Critical Section Block \(📖 TargetLink Model Element Reference\)](#)

How to Specify the Properties of a Protection Mechanism

Kind of protection

TargetLink lets you specify the properties of the protection mechanism. The most important property is the kind of protection. You can select one of the following kinds of protection for both generic RTOS and OSEK-compliant models:

- `GENERIC_USER_DEFINED`
- `DEFAULT`
- `NONE`

If you select `GENERIC_USER_DEFINED`, you can additionally enter a generic argument list in the TargetLink Data Dictionary Manager. The string you enter in the list appears as function arguments in the generated code. The following kinds of protection are available only for OSEK applications:

- `DISABLE_ALL_INTERRUPTS`
- `SUSPEND_ALL_INTERRUPTS`
- `SUSPEND_OS_INTERRUPTS`
- `RESOURCE_LOCK`
- `SCHEDULER_LOCK`

If you select `RESOURCE_LOCK`, you have to reference a resource that is already defined in the TargetLink Data Dictionary. Alternatively, you can specify a resource name template. If you do so, TargetLink creates a resource with the specified name in the `DataDictionary/Subsystems` object.

This section describes how to specify the settings of your user-defined protection mechanism if you choose to disable all interrupts. You should adapt the settings to your needs.

Preconditions

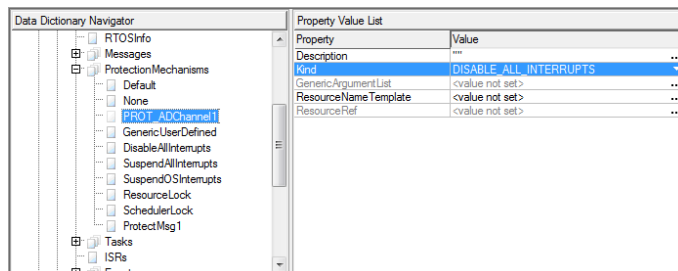
To proceed the steps below, the following preconditions must be fulfilled:

- You must have defined a protection mechanism, for example, PROT_ADChannel1, in the TargetLink Data Dictionary.
- You must have opened the TargetLink Data Dictionary Manager.

Method

To specify the properties of a protection mechanism

- 1 In the Data Dictionary Navigator of the TargetLink Data Dictionary Manager, change to the DataDictionary/Pool/RTOS/Protection Mechanism/PROT_ADChannel1 sub object.



- 2 In the property value list of the Data Dictionary Manager, select DISABLE_ALL_INTERRUPTS from the Kind list.

Result

With the above settings, generating production code results in the following code for the protected subsystem:

```
DisableAllInterrupts();
// protected code of the subsystem containing the Critical Section block;
EnableAllInterrupts();
```

Related topics

Basics

[Protecting Critical Sections..... 126](#)

References

[Critical Section Block \(📖 TargetLink Model Element Reference\)](#)

Setting up OSEK Alarms

| | |
|--------------|---|
| Using alarms | <p>The OSEK standard supports services for processing recurring events. These services are called alarms. Examples for the use of alarms are:</p> <ul style="list-style-type: none">▪ A specified action must be carried out at a specific point in time after a specified event has occurred.▪ If a task does not change to the running state within a specified time period after an event has occurred, a specified action must be carried out (software watchdog). |
|--------------|---|

| | |
|-----------------------|--|
| Where to go from here | Information in this section |
| | <div>Basics of OSEK Counters and Alarms..... 131</div> <div>Basics of the CounterAlarm Block..... 133</div> <div>Connecting Inports and Outports of the CounterAlarm Block..... 134</div> <div>Specifying Properties of Alarms and Counters..... 135</div> <div>Properties of Alarms..... 136</div> <div>Actions to be Performed when an Alarm Expires..... 137</div> <div>Properties of Counters..... 138</div> <div>How to Set Up an OSEK Alarm..... 139</div> |

Basics of OSEK Counters and Alarms

| | |
|----------------------------------|--|
| Alarms are connected to counters | <p>In an OSEK-compliant application, each alarm is associated with a counter. An alarm triggers a task activation or the setting of an OSEK event when the counter has reached a predefined value.</p> |
|----------------------------------|--|

| | |
|----------|---|
| Counters | <p>Counters are system objects that register recurring events. A counter is represented by a counter value that is measured in ticks, and the following counter properties:</p> <ul style="list-style-type: none">▪ The Max allowed value specifies the maximum allowed counter value. After the counter reaches this value, it starts from zero again.▪ The Ticks per base value specifies the number of ticks required to reach a counter-specific unit.▪ The Min cycle value specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter. |
|----------|---|

Alarms

Alarms are system objects that handle an action and the counter values indicating when the action must be performed. Alarms can be single alarms. That means an action is performed only once after the alarm is started. They can also be cyclic. Cyclic alarms perform an action every x counter ticks. Alarms can be started in two different ways:

- Relatively to the current counter value
- Absolutely at a specified counter value

Alarm expiring relative to the current counter value

The following OSEK API function is used to set up alarms relative to the current counter value:

```
StatusType SetRelAlarm (AlarmType <AlarmID>,
                        TickType <increment>,
                        TickType <cycle>)
```

After the ticks specified by the increment value have elapsed, the action assigned to the specified alarm in the AlarmID attribute is performed. The cycle value in ticks is used for cyclic alarms. If a single alarm is set up, the cycle value is 0.

Alarms expiring at an absolute counter value

The following OSEK API function is used to set up an absolute alarm:

```
StatusType SetAbsAlarm (AlarmType <AlarmID>,
                        TickType <start>,
                        TickType <cycle>)
```

If the counter reaches the value specified by the start attribute, the action assigned to the specified alarm in the AlarmID attribute is performed. The cycle value in ticks is also used for absolute cyclic alarms in the same way as for relative cyclic alarms. If a single alarm is set up, the cycle value is 0. For more information on OSEK alarms, refer to the OSEK specification.

Actions

If an alarm expires, you can carry out one of the following actions:

- Activate a task
- Set an OSEK event
- Call an alarm callback function

Which action is to be performed is specified at the alarm object in the OIL file.

Activating a task

If you choose to activate a task, the performed action is similar to the OSEK API function:

```
StatusType ActivateTask (TaskType <TaskID>)
```

After activation, the task referenced in the TaskID attribute is ready to execute.

Setting an event

If you select an OSEK event to be set when the alarm expires, the performed action is similar to the OSEK API function:

```
StatusType SetEvent (TaskType <TaskID>
                    EventMaskType <Mask>)
```

Calling SetEvent transfers the <TaskID> task to the ready state, if it was waiting for at least one of the events specified in <Mask>.

Calling an alarm callback function

You can select to call an alarm callback routine when an alarm expires. An alarm callback routine is a short function provided by the application that gets called immediately when the alarm expires. An alarm callback routine is similar to the task with the highest priority. If you select to execute an alarm callback routine, you must have defined it before using the following syntax:

```
ALARMCALLBACK(AlarmCallbackRoutineName)
```

AlarmCallbackRoutineName specifies the name of the function to be executed. For detailed information, refer to the OSEK specification.

Related topics**Basics**

[Setting up OSEK Alarms..... 131](#)

HowTos

[How to Set Up an OSEK Alarm..... 139](#)

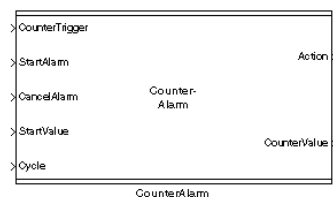
References

[CounterAlarm Block \(📖 TargetLink Model Element Reference\)](#)

Basics of the CounterAlarm Block

Using the CounterAlarm block

In TargetLink, you can use the CounterAlarm block to set up an OSEK alarm. The following illustration shows the inports and outports the CounterAlarm block provides:



For basic information on how to connect the CounterAlarm block's inports and outports, refer to [Connecting Inports and Outports of the CounterAlarm Block](#) on page 134. For details on the properties you can specify in the CounterAlarm block, refer to [Specifying Properties of Alarms and Counters](#) on page 135.

Related topics

Basics

| | |
|--|-----|
| Basics of OSEK Counters and Alarms..... | 131 |
| Connecting Inports and Outports of the CounterAlarm Block..... | 134 |
| Specifying Properties of Alarms and Counters..... | 135 |

HowTos

| | |
|----------------------------------|-----|
| How to Set Up an OSEK Alarm..... | 139 |
|----------------------------------|-----|

References

[CounterAlarm Block \(📖 TargetLink Model Element Reference\)](#)

Connecting Inports and Outports of the CounterAlarm Block

Available inports and outports

The CounterAlarm block provides the following inports and outports:

- CounterTrigger
- StartAlarm
- CancelAlarm
- StartValue
- Cycle
- Action
- CounterValue

Below you find a description of the functionalities of the CounterAlarm block's inports and outports.

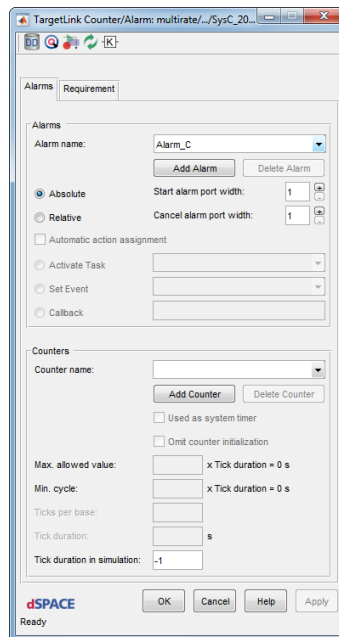
CounterTrigger

The CounterTrigger inport must be connected to a function-call-trigger source which resides outside the TargetLink subsystem. The sample time of this trigger source must be equal to the **Tick duration in simulation** value specified in the CounterAlarm block dialog. Since the trigger must not be activated at time 0, it is recommended to use a Stateflow chart to model the counter trigger. If you have not purchased Stateflow, you must model the trigger using a Simulink Function-Call Generator block, which must not be executed at simulation time 0.

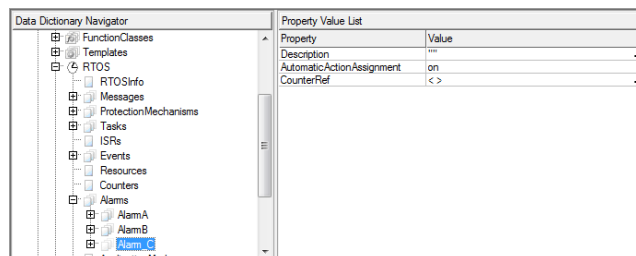
| | |
|---------------------|---|
| StartAlarm | The StartAlarm input is a function-call-trigger input. It must be connected to the part of the model that detects an occurring event. If the event occurs, the alarm is started using the OSEK API function SetRelAlarm or SetAbsAlarm. Whether the alarm is started absolutely or relatively depends on the setting in the CounterAlarm dialog. |
| CancelAlarm | The CancelAlarm input is a function-call trigger input. It must be connected to the part of the model that detects an event that cancels the alarm. If the event occurs, the alarm is canceled using the OSEK API function CancelAlarm. |
| StartValue | The StartValue input is a data input. It must be connected to the absolute or relative start value of the alarm. It must be an integer multiple of the tick duration on the target (Tick duration property) and the tick duration in simulation. The StartValue can be specified in seconds. If the tick duration on the target ECU is -1, it must be specified in counter ticks. |
| Cycle | The Cycle input is a data input. It must be connected to the cycle value in the case of cyclic alarms. If you want to set up a single alarm, the cycle value must be 0. The cycle can also be specified in seconds. If the tick duration on the target ECU is -1, it must be specified in counter ticks. |
| Action | The action output is a function-call-trigger output. It activates a task, calls an alarm call-back routine or sets an OSEK event depending on the settings in the CounterAlarm block. The action port must be connected to the function-call-triggered subsystem you want to trigger. |
| CounterValue | The CounterValue output must be connected to a Terminator block. |

Specifying Properties of Alarms and Counters

| | |
|----------------------------|--|
| CounterAlarm dialog | In TargetLink, you can specify the properties of alarms and counters as well as the action to be performed when an alarm expires in the CounterAlarm dialog. The following illustration shows the CounterAlarm dialog: |
|----------------------------|--|



The CounterAlarm dialog lets you add new alarms and counters. Additionally, it lets you reference or delete existing alarms and counters. The alarm and counter objects, including all properties, are stored in the DataDictionary /Pool/RTOS/Alarms and /Pool/RTOS/Counters subnodes of the TargetLink Data Dictionary.



Properties of Alarms

Introduction

You can select an alarm from the alarms in the DataDictionary/Pool/RTOS/Alarms subnode in the TargetLink Data Dictionary. You can reference the selected alarm in the CounterAlarm block. The CounterAlarm block lets you specify the following properties of a specified alarm:

- Kind of alarm activation (relative or absolute)
- Number of StartAlarm trigger sources (Start alarm port width)
- Number of CancelAlarm trigger sources (Cancel alarm port width)

| | |
|--|---|
| Relative alarm activation | <p>When the alarm is set up, the counter connected to it has a specific value. If the CounterAlarm block sets up a relative alarm, the specified action is performed at the following point in time:</p> $t = \text{Counter value} + \text{StartValue}$ <p>Setting up a relative alarm results in a call of the OSEK API function SetRelAlarm in the production code.</p> |
| Absolute alarm activation | <p>If the CounterAlarm block sets up an absolute alarm, the specified action is performed at the following point in time:</p> $t = \text{StartValue}$ <p>Setting up an absolute alarm results in a call of the OSEK API function SetAbsAlarm in the production code.</p> |
| Start alarm and cancel alarm port width | <p>TargetLink lets you specify the port width of the StartAlarm and the CancelAlarm inputs. This value must be set to the number of sources used to trigger the StartAlarm or the CancelAlarm input.</p> |

Actions to be Performed when an Alarm Expires

| | |
|----------------------|---|
| Introduction | <p>If an alarm expires, you can specify to perform one of the following actions:</p> <ul style="list-style-type: none"> ▪ Activate a task ▪ Set an event ▪ Call an alarm callback routine <p>Additionally, you can specify that TargetLink assigns an action to the alarm automatically.</p> |
| Activate task | <p>If you select to have TargetLink activate a task when an alarm expires, you have to reference the task by selecting it from the list in the CounterAlarm block. The task must exist in the TargetLink Data Dictionary. For details on how to create a new task in the TargetLink Data Dictionary, refer to How to Group Atomic Subsystems in Tasks Using the Task Block on page 38. The function-call-triggered subsystem connected to the Action output of the CounterAlarm block must contain a Task block. The action must reference the same task as the Task block in that function-call-triggered subsystem.</p> |
| Set event | <p>Instead of activating a task, you can specify to set an OSEK event when the alarm expires. You have to reference the OSEK event in the list and the task owning the event. The OSEK event must exist in the TargetLink Data Dictionary.</p> |

Refer to [How to Create an OSEK Event in the TargetLink Data Dictionary](#) on page 100. The same applies here as for task activation: The appropriate task and the appropriate event must be selected in the Task block of the connected subsystem.

Callback

If you want to perform a callback when the selected alarm expires, you have to enter the name of the function to be called in the Callback edit field. The function-call-triggered subsystem connected to the Action output of the CounterAlarm block must contain a Function block. The name of the function specified in the CounterAlarm block dialog must correspond to the name specified in the Function block.

Automatic action assignment

You can have TargetLink assign an action automatically. If you do, TargetLink activates the subsystem connected to the CounterAlarm block's Action output, using the activation method specified at the corresponding subsystem, for example, ActivateTask, SetEvent etc.

Related topics

HowTos

| | |
|--|-----|
| How to Create an OSEK Event in the TargetLink Data Dictionary..... | 100 |
| How to Group Atomic Subsystems in Tasks Using the Task Block..... | 38 |

Properties of Counters

Available counter properties

The CounterAlarm block lets you specify the following counter properties:

- Max allowed value
- Ticks per base
- Min cycle
- Tick duration
- Tick duration in simulation
- Omit counter initialization

The max allowed value, ticks per base and min cycle properties are already explained above. Refer to [Basics of OSEK Counters and Alarms](#) on page 131.

Tick duration

The tick duration is the duration of a counter tick on the target ECU. The value is specified in seconds. If the counter is not cyclic but event driven, you must enter -1.

| | |
|------------------------------------|---|
| Tick duration in simulation | TargetLink generates a counter S-function which is triggered by the CounterTrigger input. Suppose the trigger source triggers the counter with a very short period, for example, 1 μ s. The alarm, however, expires after 10 s. In this case, not only is the counter also triggered 10^5 times, the S-function is executed just as often which has a negative impact on the time needed to perform a simulation. To avoid loss of time during simulation in such a case, you can specify a tick duration in simulation with a longer period, for example, 1s. The S-function is then called only 10 times until the alarm expires. In this case, you have to ensure that the Function-Call Generator block connected to the CounterTrigger inport has the same sample time as the tick duration in simulation. The value is specified in seconds. The tick duration in simulation must be an integer multiple of the tick duration. If you enter -1, the tick duration in simulation gets the same value as the tick duration. |
| Omit counter initialization | The Omit counter initialization checkbox allows you to specify whether TargetLink initializes the counters or not. If an OSEK-compliant RTOS needs manual initialization of the counters, TargetLink performs this for the counter used in the CounterAlarm block. If you do not want TargetLink to initialize the counter, you have to select this checkbox. |
| Related topics | <div>Basics</div> <div>Basics of OSEK Counters and Alarms..... 131</div> |

How to Set Up an OSEK Alarm

| | |
|------------------------------|--|
| Objective | If you want to manage recurring events in an OSEK-compliant application, you can use OSEK alarms for this purpose. |
| Basics of OSEK alarms | The OSEK standard provides alarms which allow you to perform recurring actions like activating tasks, setting events or calling alarm callback routines. For details, refer to Basics of OSEK Counters and Alarms on page 131. |
| Process steps | <p>Suppose you want to set up an OSEK alarm but there are no counter or alarm objects in the TargetLink Data Dictionary. In this case, the process of setting up an OSEK alarm is divided into the following steps:</p> <ol style="list-style-type: none"> 1. Copying a CounterAlarm block to your model. 2. Creating an alarm. 3. Creating an new counter. |

4. Specifying the properties of the alarm and the counter in the CounterAlarm dialog.
5. Connecting the inports and outputs of the CounterAlarm block to the objects in your model.

Each process step (except step 1) is described in detail below. The process describes an example of setting up an OSEK alarm which activates the Reaction task 30 ms after an event has occurred.

Preconditions

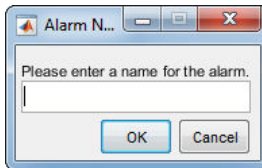
To perform the steps below, the following preconditions must be fulfilled:

- You must have purchased the TL_TMOS_OSEK license to be able to use the CounterAlarm block.
- You must have copied a CounterAlarm block to the model concerned.
- A "Reaction" task must exist in the TargetLink Data Dictionary. For details on how to create a new task in the TargetLink Data Dictionary, refer to [How to Group Atomic Subsystems in Tasks Using the Task Block](#) on page 38.
- The model must contain a Stateflow chart with a function-call trigger output or a Function-Call Generator block. In the following steps a Stateflow chart is used.

Method 1

To create an alarm

- 1 In your model, double-click the CounterAlarm block to open the CounterAlarm dialog.
- 2 Under Alarms, click Add Alarm to open the Alarm Name dialog.



- 3 Enter a name for the alarm in the edit field, for example, **Alarm1**.
- 4 Click OK to close the dialog.
- 5 In the CounterAlarm dialog, click Apply.

Result

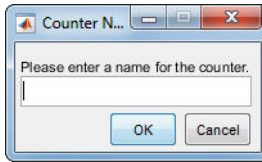
You have now created a new alarm. It is stored in the DataDictionary/Pool/RTOS/Alarms subnode in the TargetLink Data Dictionary. Additionally, it appears in the Alarm name list in the CounterAlarm dialog.

Next step

The next step is to create a new counter. Refer to the following topic.

Method 2**To create a counter**

- 1 Under Counters in the CounterAlarm block, click Add Counter to open the Counter Name dialog.



- 2 Enter a name for the counter in the edit field, for example, **Counter1**.
- 3 Click OK to close the dialog.
- 4 In the CounterAlarm dialog, click Apply.

Result

You have now created a new counter. It is stored in the DataDictionary/Pool/RTOS/Counters subnode in the TargetLink Data Dictionary. Additionally, it appears in the Counter name list in the CounterAlarm dialog.

Next step

The next step is to specify the properties of the alarm and the counter.

Method 3**To specify the properties of the alarm and the counter in the CounterAlarm dialog****Note**

The values specified in the steps below show an example of an OSEK alarm that activates the Reaction task 30 ms after an event has occurred. You should adapt the values to your needs.

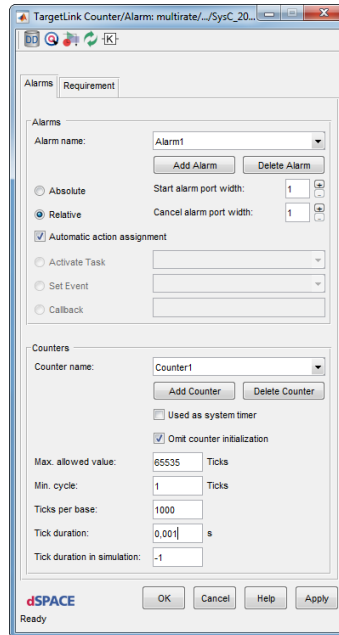
- 1 Under Alarms in the CounterAlarm dialog, select Alarm1 from the Alarm name list.
- 2 Select Relative.
- 3 In the Start alarm port width box, select or enter 1.
- 4 In the Cancel alarm port width box, select or enter 1.
- 5 Select the Automatic action assignment checkbox.
You have now specified the alarm properties. The next step is to specify the counter properties.
- 6 Under Counters, select Counter1 from the Counter name list.
- 7 In the Max allowed value edit field, enter 65535.
- 8 In the Min cycle edit field, enter 1.
- 9 In the Ticks per base edit field, enter 1000.

10 In the Tick duration edit field, enter **0.001** to specify a tick duration of 1 ms.

11 In the Tick duration in simulation edit field, enter **-1**.

Result

You have specified all properties in the CounterAlarm block as shown in the illustration below:



Next step

The next step is to connect the inports and outports of the CounterAlarm block to the objects in your model.

Method 4

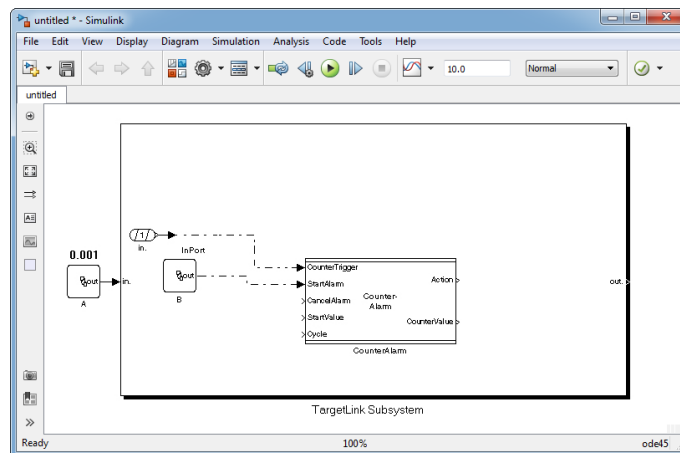
To connect the inports and outports of the CounterAlarm block

- 1 Connect the CounterAlarm block's CounterTrigger inport to the trigger source of the counter.

Note

Since the trigger must not be activated at time 0, it is recommended to use a Stateflow chart to model the counter trigger. The sample time of the Stateflow chart must correspond to the tick duration, so you should enter 0.001 for the sample time. The Stateflow chart must ensure that the function call is not triggered at simulation time 0.

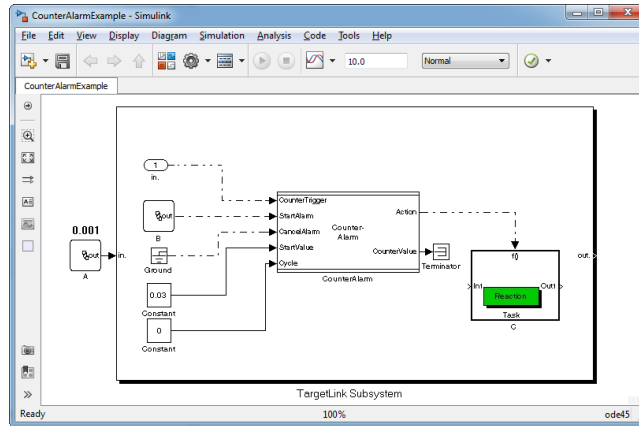
- 2 Connect the StartAlarm input to the part of the model detecting the event that starts the alarm, for example, a Stateflow chart.



- 3 From the Simulink Library, copy a Ground block to the model and connect it to the CancelAlarm input.
- 4 From the TargetLink Library, copy two Constant blocks to the model and connect them to the StartValue and the Cycle input.
- 5 Assign the value 0.03 to the Constant block connected to the StartValue input.
- 6 Assign the value 0 to the Constant block connected to the Cycle input.
You have now connected the inputs of the CounterAlarm block to the objects in your model. The next step is to connect the CounterAlarm blocks Action output to the "Reaction" task that must be activated by the alarm.
- 7 From the Simulink Library, copy a function-call-triggered subsystem to your model and connect it to the CounterAlarm block's Action output.
- 8 Open the function-call-triggered subsystem and copy a Task block from the RTOS Blocks library to the subsystem.
- 9 Double-click the Task block to open the Task dialog and select Reaction from the Task name list.
- 10 Click OK to close the dialog.
- 11 Connect the function-call-triggered subsystem to the CounterAlarm block's Action output.
- 12 From the Simulink Library, copy a Terminator block to the model and connect it to the CounterValue block.

Result

You have now connected the inputs and outputs of the CounterAlarm block to the corresponding objects in your model as shown in the illustration below:



The following objects result from the model above:

- Production code
- OIL file

An excerpt from the production code setting up the relative alarm and an excerpt from the OIL file are shown below:

Production code excerpt

```
B()
{
    ...
    if (event)
        SetRelAlarm(Alarm1,30,0)
    ...
}
```

OIL file excerpt

When an OIL file has been exported, it contains the following excerpt:

```
ALARM Alarm1{
    COUNTER=Counter1;
    ACTION=ACTIVATETASK{
        TASK=Reaction;
    }
}
```

```
COUNTER Counter1{
    MAXALLOWEDVALUE = 65535;
    TICKSPERBASE = 1000;
    MINCYCLE = 1;
}
```


Related topics


Basics

| | |
|---|-----|
| Basics of OSEK Counters and Alarms..... | 131 |
|---|-----|

HowTos

| | |
|---|----|
| How to Group Atomic Subsystems in Tasks Using the Task Block..... | 38 |
|---|----|

References

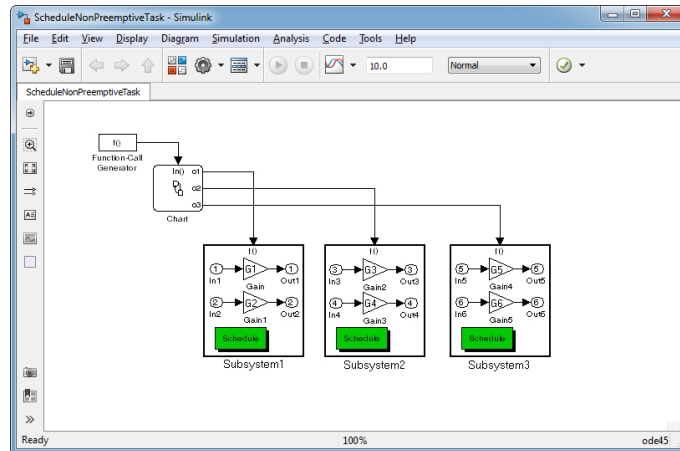
| | |
|--|--|
| CounterAlarm Block ( TargetLink Model Element Reference) | |
|--|--|

Interrupting a Non-Preemptive Task

How to Insert a Schedule Command into a Non-Preemptive Task

| | |
|--|--|
| Objective | To call the scheduler between the calculation of two atomic subsystems, you can use the Schedule block. The Schedule block has no input and no output. If you copy a Schedule block to an atomic subsystem, TargetLink places a call of the OSEK API function <code>Schedule()</code> in the code unit representing the atomic subsystem as the last statement. |
| Preemptive and non-preemptive systems | The disadvantage of a completely preemptive system is that interrupts can occur, and tasks can influence each other, at any point. Thus, real-time behavior is extremely complex, and therefore difficult to define. The disadvantage of non-preemptive systems is that high-priority tasks might have to wait a long time for lower-priority tasks to finish computing. A compromise approach is to subdivide a non-preemptive task into parts that must not be interrupted (sometimes called processes), but a high-priority task can interrupt between these parts by explicitly calling the scheduler. |
| Precondition | <p>To perform the steps described below, the following preconditions must be fulfilled:</p> <ul style="list-style-type: none"> ▪ You must have purchased the TL_TMOS_OSEK license. ▪ You must have opened a model. |
| Method | <p>To insert a schedule command into a non-preemptive task</p> <ol style="list-style-type: none"> 1 In your model, open an atomic subsystem you want to put the scheduler call in. |

- 2 Copy a Schedule block from the RTOS Blocks library to the subsystem.



- 3 Close the subsystem.

Result

During production code generation, TargetLink places a call of the OSEK API function `Schedule()` as the last statement into the code unit representing the atomic subsystem. The code resulting from the subsystems is shown below:

```
void Chart(void)
{
    Out1 = G1 * In1;
    Out2 = G2 * In2;
    Schedule();
    Out3 = G3 * In3;
    Out4 = G4 * In4;
    Schedule();
    Out5 = G5 * In5;
    Out6 = G6 * In6;
    Schedule();
}
```

Related topics

Basics

[Interrupting a Non-Preemptive Task..... 146](#)

References

[Schedule Block \(📖 TargetLink Model Element Reference\)](#)

Simulating Multirate Models and Building Multitasking Applications

Building a multitasking application

If you simulate multirate models, you have to take special characteristics into account. Apart from these, the simulation of multirate models does not differ from the simulation of standard single-rate models. After you simulate the generated production code, you have to use it and other modules such as the RTOS to assemble the final real-time application. This section gives you information on TargetLink's tasks in this process and how you can influence it.

Where to go from here

Information in this section

| | |
|--|---------------------|
| Simulating Multirate Models..... | 150 |
| Building a Multitasking Application..... | 153 |


Simulating Multirate Models

Introduction

To verify the behavior of your multirate model and calculate relationships between its inputs, states and outputs, you have to simulate it.




Basics of Simulating Multirate Models

Simulation modes

For information on how to work with the different simulation modes and how to analyze the simulation results, refer to [Basics on Simulation Modes and Preconditions](#) ( [TargetLink Preparation and Simulation Guide](#)).

Characteristics of simulating OSEK models

During simulation, TargetLink must handle multirate models in a different way than single-rate models. For software-in-the-loop (SIL) and processor-in-the-loop (PIL) simulations, the production code remains unchanged in the case of generic RTOS applications. However, for OSEK applications, the production code contains OSEK OS calls that must be emulated for simulations. TargetLink provides an OSEK emulator for this purpose. It contains a scheduler and supports all OSEK API functions. It behaves like an OSEK-compliant operating system without real-time capabilities.

To perform a simulation, TargetLink replaces the model's TargetLink subsystem with an S-function frame. The S-function frame serves as an interface between the simulation application and Simulink. It sends input data to the simulation application on the simulation target, which can be the host computer or a microcontroller evaluation board ( [physical](#) or  [virtual](#)). It calls functions and receives the calculated output data. For detailed information, refer to [Simulating Models and Analyzing Simulation Results](#) ( [TargetLink Preparation and Simulation Guide](#)).

The process steps of simulating single-rate and multirate models are shown below to explain the differences between the two simulation types.

Simulation of single-rate models

The following process steps are executed to start a simulation process for a standard single-rate model:

1. The S-function sends input data for a step function to the target.
2. The step function is executed on the target.
3. The S-function reads the calculated output data from the step function.

Simulation of multirate models

The goal of simulating multirate models is to visualize as many effects as possible that may appear on the target in real time. Since the simulation does not run in real time, there are some circumstances that it cannot take into account – for

example, the fact that tasks need a certain period of time to finish calculation and thus can be interrupted by a task with a higher priority in the meantime. One aspect that a simulation can show, however, is the effect of cyclic tasks' calculation sequences (according to their properties) on the simulation result, where cyclic tasks are activated simultaneously in a certain time step. For OSEK applications, TargetLink uses alarms to activate cyclic tasks not only in the production code, but also in the simulation. As a result, the simulation checks whether the alarm initialization is done correctly. The steps of the simulation process are explained in detail below:

- The simulation application is initialized, that is, the current system time is set up on the target, all autostart tasks are executed, and in the case of OSEK, the OSEK API function StartupHook is also executed. Refer to [Interaction of Different Code Parts](#) on page 160. This initialization is done every time a [SIL simulation](#) or [PIL simulation](#) begins.
- Cyclic root step functions are executed.
- Noncyclic root step functions are executed.

Simulation of cyclic root step functions

The simulation of cyclic root step functions differs from standard simulations. The execution of the cyclic root step functions is scheduled on the target by the OSEK emulator. To simulate cyclic root step functions, the following steps are performed:

1. The S-function identifies all cyclic root step functions to be executed at the current simulation step.
2. The S-function sends the input data of all cyclic root step functions to be executed at the current simulation step to the simulation application on the target.
3. The S-function sends the time of the current simulation step to the target.
4. The system timer on the target is incremented until its value is identical to the time of the current simulation step. As a result, the cyclic root step functions are executed on the target.
5. The S-function reads the output data of all tasks or C functions belonging to the root step functions.

Simulation of noncyclic root step functions

The simulation of noncyclic root step functions proceeds in a similar way to standard simulations:

1. The S-function sends the input data for all tasks or C functions belonging to the noncyclic root step function to the simulation application.
2. The S-function executes a function which calls a C function, activates a task or sets an event. If the root step function is not a task, a C function is called. Otherwise, the ActivateTask or SetEvent function is called, depending on the activation kind you chose for the task.
3. The S-function reads the output data of all tasks or C functions belonging to the root step function.

Related topics

Basics

| | |
|--|-----|
| Interaction of Different Code Parts..... | 160 |
| Simulating Models and Analyzing Simulation Results (📖 TargetLink Preparation and Simulation Guide) | |

Building a Multitasking Application

Introduction

If you want to run your implemented algorithm on a target ECU, you have to build the complete multitasking application.

Where to go from here

Information in this section

| | |
|--|-----|
| Building a Generic RTOS Application..... | 153 |
| Building an OSEK Application..... | 158 |

Building a Generic RTOS Application

Introduction

To build a generic RTOS application, you should be familiar with the following two aspects:

- Interaction of the various tools generating files needed for the application
- Interaction of C code parts generated by TargetLink and user-defined C code parts during system startup

Where to go from here

Information in this section

| | |
|---|-----|
| Interaction of Tools Generating C Files..... | 153 |
| Interaction of Production Code and Custom Code..... | 154 |
| How to Build a Generic RTOS Application..... | 157 |

Interaction of Tools Generating C Files

Introduction

The following tools are involved in the process of building an application:

- TargetLink
- Make tool
- Compiler
- Linker
- Loader

TargetLink generates C files containing the application. The production code interacts with other code which is supplied by the vendor of your operating system or which you have to write yourself such as the main function. All code parts can be compiled and linked to an application. Afterwards, you can download it to your target.

Interaction of Production Code and Custom Code

Interaction of code parts

The following code parts generated by TargetLink and user-defined code parts interact with each other during the system startup:

- Startup code that comes with the compiler or the operating system
- Main function (user-defined)
- Operating system
- Restart task
- Tasks and ISRs containing the application code generated by TargetLink
- TlTriggerTask macro
- TlLockMessage and TlUnlockMessage macros
- TlEnterCriticalSection and TlLeaveCriticalSection macros

System startup

The startup code is usually provided by the vendor of your compiler. However, you can also implement it yourself or adapt it to your specific needs. It initializes the processor and starts the main function.

Main function

To build an application, you need a main function that starts the operating system by calling the StartOS function. TargetLink provides a template for the main function containing a StartOS function call.

Operating system

The operating system is provided by the vendor as C code or libraries.

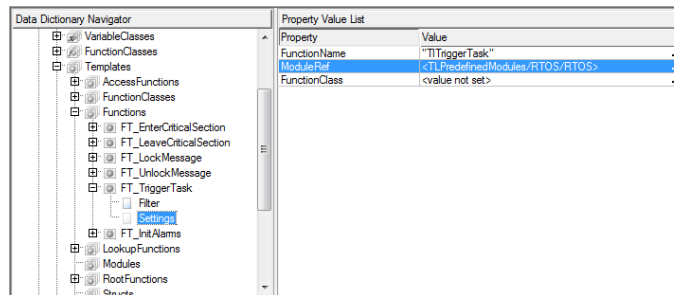
Tasks and ISRs

TargetLink partitions multirate models into tasks and interrupt service routines (ISRs). The operating system schedules the execution of the tasks and ISRs.

TlTriggerTask macro

To activate noncyclic tasks in generic RTOS applications, TargetLink uses the **TlTriggerTask(x)** macro. x stands for the name of the noncyclic task to be activated. TargetLink places this macro at every function-call trigger source triggering a subsystem with a Task block. You can specify the name of this macro and the header file containing its definition in the TargetLink Data Dictionary. You have to complete the macro with your own code. It depends on your

operating system how the macro is completed. Refer to the documentation of your operating system.



Macros for protection of critical sections

For generic RTOS applications, TargetLink provides the `TlEnterCriticalSection()` and `TlLeaveCriticalSection()` macros to protect critical sections from being interrupted during calculation. If you copy a Critical Section block to an atomic subsystem, TargetLink generates the macro into the production code in the following way:

```
TlEnterCriticalSection(<ArgList>;
// code for protected subsystem
TlLeaveCriticalSection (<ArgList>;
```

The `TlEnterCriticalSection` and `TlLeaveCriticalSection` macros both contain an argument list. You can enter the arguments in the protection mechanism. It is also possible to leave the argument list empty. You can specify the names of the macros and the header files containing their definitions in the TargetLink Data Dictionary. For more information, refer to [Protecting Critical Sections](#) on page 126.

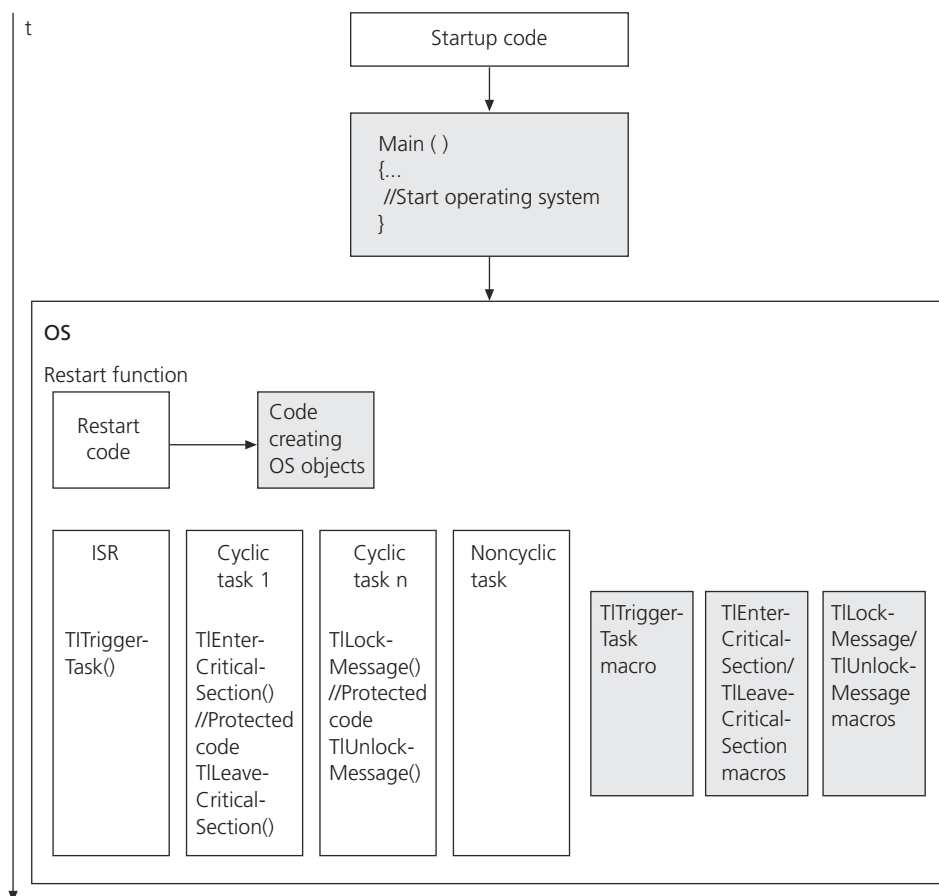
Macros for ensuring message data integrity

For generic RTOS applications, TargetLink generates the `TlLockMessage()` and `TlUnlockMessage()` into the production code to ensure the data integrity of messages. If you select the protection mechanism `GENERIC_USER_DEFINED`, TargetLink generates the following code:

```
TlLockMessage();
// protected code
TlUnlockMessage();
```

As for the macros mentioned above, you can specify their names and the header files containing their definitions in the TargetLink Data Dictionary. Like `TlEnterCriticalSection` and `TlLeaveCriticalSection`, these macros can also contain argument lists.

The following illustration shows the interaction of the described code parts. The code parts which are highlighted in the illustration are user-defined.



The steps you have to perform when building a generic multirate application are described in the following topic [How to Build a Generic RTOS Application](#) on page 157.

Related topics

Basics

[Protecting Critical Sections.....](#) 126

HowTos

[How to Build a Generic RTOS Application.....](#) 157

How to Build a Generic RTOS Application

Objective To run the production code on a generic RTOS, you have to build an application.

Precondition To perform the steps described below, the following preconditions must be fulfilled:

- You must have opened the Data Dictionary Manager.
- You must have opened a model.

Method

To build a generic RTOS application

- 1 If necessary, adapt the following macro calls to your RTOS using the templates in the **DataDictionary/Pool/Templates/Functions** subnode of the Data Dictionary Manager:
 - TITriggerTask
 - TIEnterCriticalSection and TILeaveCriticalSection
 - TILockMessage and TIUnlockMessage
- 2 If necessary, complete the above macros to adapt them to the needs of your operating system.
- 3 On the Code Generation page of your model's TargetLink Main Dialog, click **Generate Code**.
TargetLink generates the production code for your multirate model.
- 4 Write the main function that starts the operating system.
- 5 Write the code that creates all necessary OS objects.
- 6 In your operating system, make sure that the following preconditions have been met:
 - All cyclic tasks are activated with their corresponding sample times.
 - All noncyclic tasks are connected to the corresponding trigger sources outside the model.
 For details, refer to the documentation of your operating system.
- 7 Compile the application and download it to your target.

Result You have now built a generic RTOS application.

Related topics

Basics

| | |
|--|-----|
| Building a Generic RTOS Application..... | 153 |
| Building an OSEK Application..... | 158 |

Building an OSEK Application

Introduction

To build an OSEK-compliant application, there are two aspects you should be familiar with:

- Interaction of the various tools generating files needed for the OSEK application.
- Interaction of C code parts generated by TargetLink with user-defined C code parts during system startup.

Where to go from here

Information in this section

| | |
|--|-----|
| Interaction of the Tools Generating Files for an OSEK Application..... | 158 |
| Interaction of Different Code Parts..... | 160 |
| How to Build an OSEK Application..... | 163 |

Interaction of the Tools Generating Files for an OSEK Application

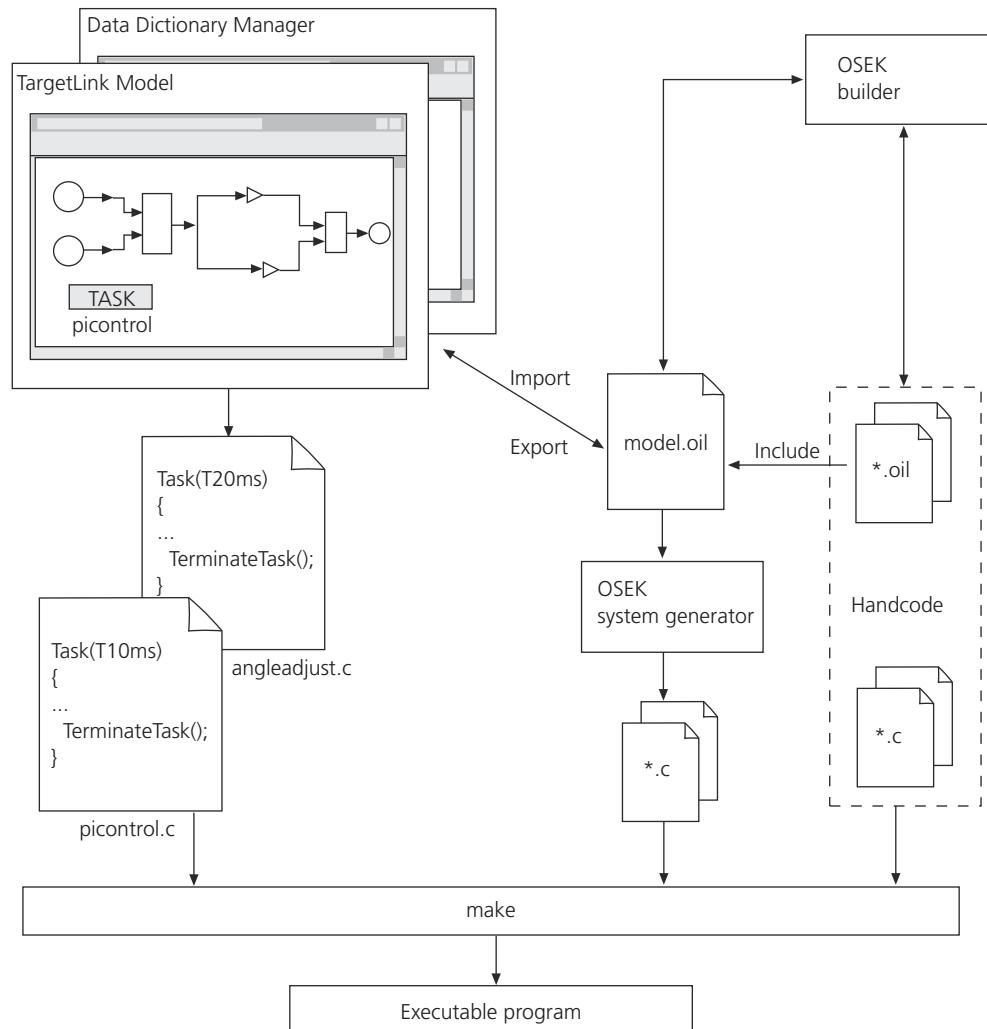
Tools for building an OSEK application

The following tools are (potentially) involved in the process of building an OSEK-compliant application:

- TargetLink
- The OSEK builder for OIL file generation
- The system generator to build the operating system according to the OIL specifications
- The make tool
- The compiler and the linker
- The loader

Application building process

The following illustration shows how to compose an OSEK-compliant application:



After you create a multirate model with TargetLink, you can generate production code for it. TargetLink then generates not only production code but also an OIL file. The OIL file can be used directly as input for the system generator or it can be edited with the OSEK builder to add or change specifications. Moreover, you can create an OIL file with the OSEK builder, import it into the TargetLink Data Dictionary and edit and use the OIL objects in TargetLink. You can use the system generator to convert an OIL file to C code that fits a specific, OSEK-compliant RTOS. The system generator is delivered together with the OSEK operating system. The make process creates an executable program from all C files.

For detailed information on how to compose an OSEK-compliant application using the tools and files involved, refer to [How to Build an OSEK Application](#) on page 163.

Interaction of Different Code Parts

| | |
|--|--|
| Code parts of an OSEK application | <p>The following code parts are involved in starting an OSEK-compliant application on a target processor:</p> <ul style="list-style-type: none"> ▪ The startup code ▪ The main function ▪ The operating system ▪ The InitAlarms function ▪ The RestartTask ▪ Alarms ▪ Tasks ▪ Hook scripts |
| System startup | <p>The startup code is usually provided by the vendor of your compiler. However, you can also implement it yourself or adapt it to your specific needs. It initializes the processor and starts the main function.</p> |
| Main function | <p>To build an application, you need a main function that starts the operating system by calling the StartOS function. TargetLink provides a template for the main function containing a StartOS function call.</p> |
| Operating system | <p>The OSEK standard describes a static operating system, that means its objects and attributes are known at the time of compilation. As a result, the operating system can be adapted to a specific application, i.e. it can be scaled. Thus, it provides a method to achieve a highly efficient implementation. Usually, the C files containing the operating system consist of a static part (as C code or libraries) and a part generated by the OSEK system generator.</p> |
| Setting up alarms | <p>Besides the application code, TargetLink generates OSEK objects such as alarms which are used for the application. Alarms are used for cyclic task activation and must be set up during system initialization. TargetLink generates a corresponding InitAlarms function which contains OSEK API function calls setting up alarms for cyclic OSEK task activation. For more information, refer to Setting Up Alarms for Cyclic OSEK Task Activation on page 108.</p> |
| Initializing variables | <p>TargetLink provides a method to generate a task called RestartTask. The RestartTask initializes variables which are not automatically initialized at variables definition of the production code and calls the InitAlarms function. The Autostart property must be enabled for the RestartTask. For more information on task properties, refer to Task Block (TargetLink Model Element Reference) in the TargetLink Block and Object Reference.</p> |

Tasks and ISRs

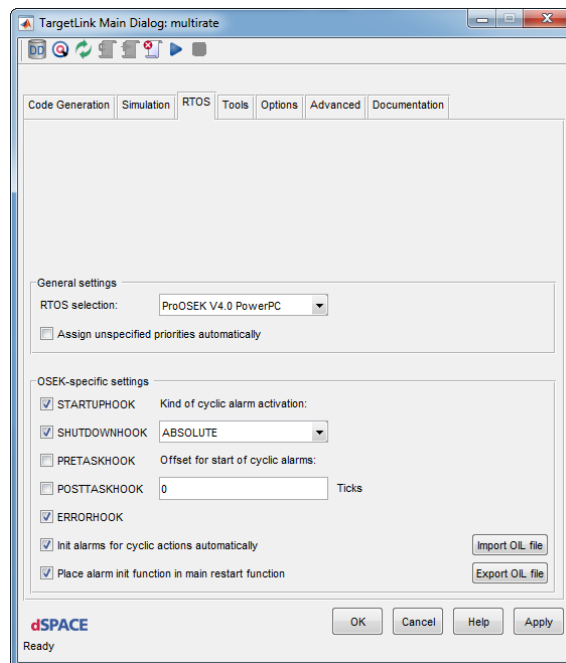
TargetLink partitions multirate models into tasks and interrupt service routines (ISRs). The operating system schedules the execution of the tasks and ISRs.

Hook scripts

OSEK operating systems allow you to make use of hook scripts to handle specific RTOS internal events, for example:

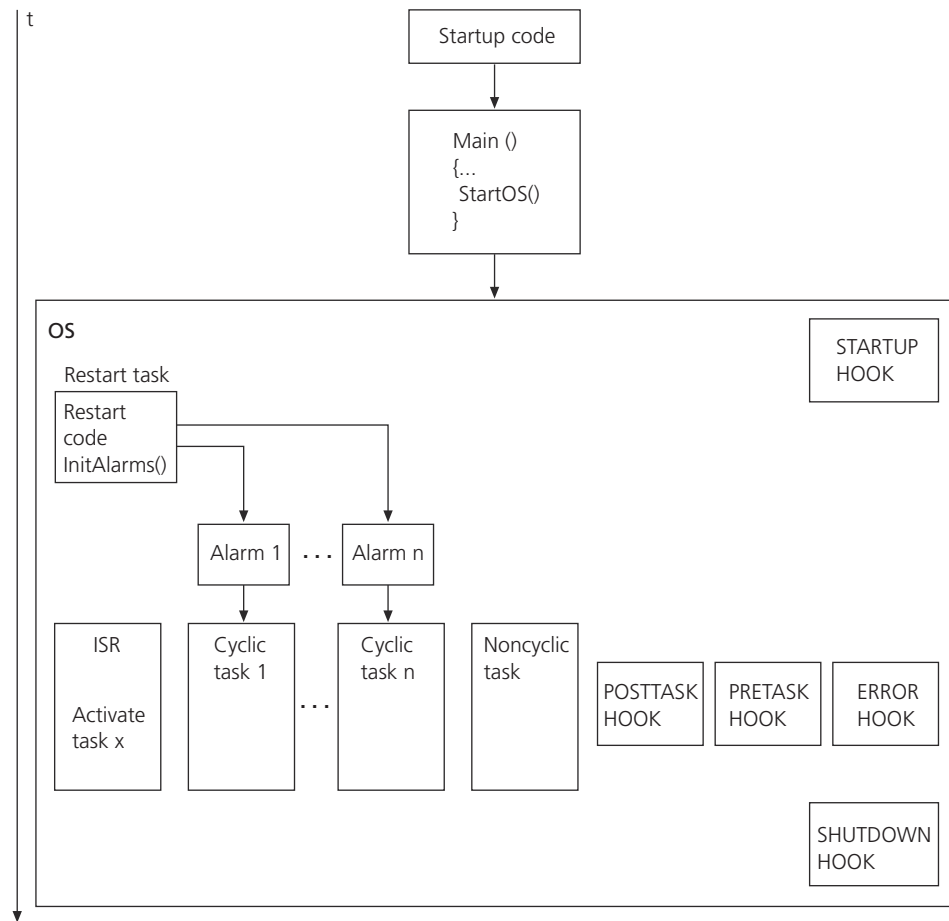
- System startup
- System shutdown
- Error handling

If one of the above events occurs, the operating system calls the corresponding hook script. You must inform your operating system which hook scripts exist. You can specify the existing on the RTOS page of the TargetLink Main Dialog as shown in the following illustration:



The specification of the hook scripts is exported to an OIL file. Usually, hook scripts are used for diagnostic purposes and have no influence on the algorithms specified in TargetLink.

The following illustration shows how the above code parts interact with each other during system startup:



After you switch on a microcontroller's power supply, the startup code initializes it. At the end of the startup code, the main function is started.

The main function calls the StartOS function to start the operating system. The operating system starts the RestartTask, which initializes the variables and starts the InitAlarms function. InitAlarms sets up the alarms for cyclic task activation. From now on, the application tasks are activated according to their periods, or in the case of noncyclic tasks, their triggers.

Related topics

Basics

[Setting Up Alarms for Cyclic OSEK Task Activation.....](#) 108

How to Build an OSEK Application

Objective

To run an application on an OSEK-compliant operating system, the application must be OSEK-compliant. For more information on the build process, refer to [Building an OSEK Application](#) on page 158.

Method

To build an OSEK application

Note

When creating a multirate model for an OSEK application, you can import an existing OIL file to make use of the OIL objects in your model. Refer to [OIL File Import and Export Instructions](#) on page 121.

- 1 Create a TargetLink multirate model and select the RTOS used. For details, refer to [How to Select the RTOS](#) on page 90.
- 2 On the RTOS page of the model's TargetLink Main Dialog, make sure that the Init alarms for cyclic actions automatically checkbox is selected. For details, refer to [How to Set Up Alarms for Cyclic OSEK Tasks Automatically](#) on page 115.
- 3 In the TargetLink Main Dialog, change to the Code Generation page.
- 4 Click Generate Code.
TargetLink generates C files containing the production code and enters new OIL objects in the TargetLink Data Dictionary if necessary. The production code contains the InitTask and the InitAlarms functions according to your settings on the RTOS page of the TargetLink Main Dialog.
- 5 Connect all noncyclic tasks in your application with the corresponding trigger source. For details, refer to the documentation of your OSEK operating system.
- 6 Export the OIL file.
For detailed information on how to export an OIL file, refer to [How to Export an OIL File](#) on page 123.

Note

Before converting the OIL file into C code, you can edit and change it with your OSEK builder.

- 7 Convert the OIL file into C code with your system generator.

- 8 From the several C files including the startup code and the main function, generate an executable program with your make tool.

Note

If your OSEK-compliant operating system requires several h files to be included in the generated code, you have to write one h file that includes the others. Specify its name in the NameTemplate edit field in the `DataDictionary/Pool/Templates/Modules/MT_osek/Settings` subnode of the TargetLink Data Dictionary.

Result

You have built an OSEK-compliant application. You can now download the application to the target ECU with your loader.

Related topics

Basics

[Building an OSEK Application.....](#) 158

HowTos

[How to Build a Generic RTOS Application.....](#) 157
[How to Export an OIL File.....](#) 123
[How to Select the RTOS.....](#) 90
[How to Set Up Alarms for Cyclic OSEK Tasks Automatically.....](#) 115
[OIL File Import and Export Instructions.....](#) 121

Glossary

Introduction

The glossary briefly explains the most important expressions and naming conventions used in the TargetLink documentation.

Where to go from here

Information in this section

| | |
|---------------|-----|
| Numerics..... | 166 |
| A..... | 167 |
| B..... | 170 |
| C..... | 171 |
| D..... | 174 |
| E..... | 176 |
| F..... | 177 |
| G..... | 178 |
| I..... | 178 |
| L..... | 180 |
| M..... | 181 |
| N..... | 183 |
| O..... | 184 |
| P..... | 185 |
| R..... | 186 |
| S..... | 188 |
| T..... | 190 |
| U..... | 192 |
| V..... | 192 |
| W..... | 193 |

Numerics

1-D look-up table

output value (y).

A look-up table that maps one input value (x) to one

2-D look-up table

output value (z).

A look-up table that maps two input values (x,y) to one

Abstract interface An interface that allows you to map a project-specific, physical specification of an interface (made in the TargetLink Data Dictionary) to a logical interface of a [modular unit](#). If the physical interface changes, you do not have to change the Simulink subsystem or the [partial DD file](#) and therefore neither the generated code of the modular unit.

Access function (AF) A C function or function-like preprocessor macro that encapsulates the access to an interface variable.

See also [read/write access function](#) and [variable access function](#).

Acknowledgment Notification from the [RTE](#) that a [data element](#) or an [event message](#) have been transmitted.

Activating RTE event An RTE event that can trigger one or more runnables. See also [activation reason](#).

Activation reason The [activating RTE event](#) that actually triggered the runnable.

Activation reasons can group several RTE events.

Active page pointer A pointer to a [data page](#). The page referenced by the pointer is the active page whose values can be changed with a calibration tool.

Adaptive AUTOSAR Short name for the AUTOSAR *Adaptive Platform* standard. It is based on a service-oriented architecture that aims at on-demand software updates and high-end functionalities. It complements [Classic AUTOSAR](#).

Adaptive AUTOSAR behavior code Code that is generated for model elements in [Adaptive AUTOSAR Function subsystems](#) or [Method Behavior subsystems](#). This code represents the behavior of the model and is part of an adaptive application. Must be integrated in conjunction with [ARA adapter code](#).

Adaptive AUTOSAR Function A TargetLink term that describes a C++ function representing a partial functionality of an adaptive application. This function can be called in the C++ code of an adaptive application. From a higher-level perspective, [Adaptive AUTOSAR](#) functions are analogous to runnables in [Classic AUTOSAR](#).

Adaptive AUTOSAR Function subsystem An atomic subsystem used to generate code for an [Adaptive AUTOSAR Function](#). It contains a Function block whose AUTOSAR mode is set to **Adaptive** and whose Role is set to **Adaptive AUTOSAR Function**.

ANSI C Refers to C89, the C language standard ANSI X3.159-1989.

Application area An optional DD object that is a child object of the DD root object. Each Application object defines how an [ECU](#) program is built from the generated subsystems. It also contains some experiment data, for example, a list of variables to be logged during simulations and results of code coverage tests.

Build objects are children of **Application** objects. They contain all the information about the binary programs built for a certain target platform, for example, the symbol table for address determination.

Application data type Abstract type for defining types from the application point of view. It allows you to specify physical data such as measurement data. Application data types do not consider implementation details such as bit-size or endianness.

Application data type (ADT) According to AUTOSAR, application data types are used to define types at the application level of abstraction. From the application point of view, this affects physical data and its numerical representation. Accordingly, application data types have physical semantics but do not consider implementation details such as bit width or endianness.

Application data types can be constrained to change the resolution of the physical data's representation or define a range that is to be considered.

See also [implementation data type \(IDT\)](#).

Application layer The topmost layer of the [ECU software](#). The application layer holds the functionality of the [ECU software](#) and consists of [atomic software components \(atomic SWCs\)](#).

ARA adapter code Adapter code that connects [Adaptive AUTOSAR behavior code](#) with the Adaptive AUTOSAR API or other parts of an adaptive application.

Array-of-struct variable An array-of-struct variable is a structure that either is non-scalar itself or that contains at least one non-scalar substructure at any nesting depth. The use of array-of-struct variables is linked to arrays of buses in the model.

Artifact A file generated by TargetLink:

- Code coverage report files
- Code generation report files
- [Metadata files](#)
- Model-linked code view files
- [Production code](#) files
- Simulation application object files
- Simulation frame code files
- [Stub code](#) files

Artifact location A folder in the file system that contains an [artifact](#). This location is specified relatively to a [project folder](#).

ASAP2 File Generator A TargetLink tool that generates ASAP2 files for the parameters and signals of a Simulink model as specified by the corresponding TargetLink settings and generated in the [production code](#).

ASCII In production code, strings are usually encoded according to the ASCII standard. The ASCII standard is limited to a set of 127 characters implemented by a single byte. This is not sufficient to display special characters of different languages. Therefore, use another character encoding, such as UTF-8, if required.

Asynchronous operation call subsystem A subsystem used when modeling *asynchronous* client-server communication. It is used to generate the call of the `Rte_Call` API function and for simulation purposes.

See also [operation result provider subsystem](#).

Asynchronous server call returns event An [RTE event](#) that specifies whether to start or continue the execution of a [runnable](#) after the execution of a [server runnable](#) is finished.

Atomic software component (atomic SWC) The smallest element that can be defined in the [application layer](#). An atomic SWC describes a single functionality and contains the corresponding algorithm. An atomic SWC communicates with the outside only via the [interfaces](#) at the SWC's [ports](#). An atomic SWC is defined by an [internal behavior](#) and an [implementation](#).

Atomic software component instance An [atomic software component \(atomic SWC\)](#) that is actually used in a controller model.

AUTOSAR Abbreviation of AUTomotive Open System ARchitecture. The AUTOSAR partnership is an alliance in which the majority of OEMs, suppliers, tool providers, and semiconductor companies work together to develop and establish a de-facto open industry-standard for automotive electric/electronics (E/E) architecture and to manage the growing E/E complexity.

AUTOSAR import/export Exchanging standardized [software component descriptions](#) between [AUTOSAR tools](#).

AUTOSAR subsystem An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to `Classic`. See also [operation subsystem](#), [operation call with runnable implementation subsystem](#), and [runnable subsystem](#).

AUTOSAR tool Generic term for the following tools that are involved in the ECU network software development process according to AUTOSAR:

- Behavior modeling tool
- System-level tool
- ECU-centric tool

TargetLink acts as a behavior modeling tool in the ECU network software development process according to AUTOSAR.

Autoscaling Scaling is performed by the Autoscaling tool, which calculates worst-case ranges and scaling parameters for the output, state and parameter variables of TargetLink blocks. The Autoscaling tool uses either worst-case ranges or simulated ranges as the basis for scaling. The upper and lower worst-case range limits can be calculated by the tool itself. The Autoscaling tool always focuses on a subsystem, and optionally on its underlying subsystems.

B

Basic software The generic term for the following software modules:

- System services (including the operating system (OS) and the [ECU State Manager](#))
- Memory services (including the [NVRAM manager](#))
- Communication services
- I/O hardware abstraction
- Complex device drivers

Together with the [RTE](#), the basic software is the platform for the [application layer](#).

Batch mode The mode for batch processing. If this mode is activated, TargetLink does not open any dialogs. Refer to [How to Set TargetLink to Batch Mode](#) ([TargetLink Orientation and Overview Guide](#)).

Behavior model A model that contains the control algorithm for a controller (function prototyping system) or the algorithm of the controlled system (hardware-in-the-loop system). Can be connected in [ConfigurationDesk](#) via [model ports](#) to build a real-time application (RTA). The RTA can be executed on real-time hardware that is supported by [ConfigurationDesk](#).

Block properties Properties belonging to a TargetLink block. Depending on the kind of the property, you can specify them at the block and/or in the Data Dictionary. Examples of block properties are:

- Simulink properties (at a masked Simulink block)
- Logging options or saturation flags (at a TargetLink block)
- Data types or variable classes (referenced from the DD)
- Variable values (specified at the block or referenced from the DD)

Bus A bus consists of subordinate [bus elements](#). A bus element can be a bus itself.

Bus element A bus element is a part of a [bus](#) and can be a bus itself.

Bus port block Bus Inport, Bus Outport are bus port blocks. They are similar to the TargetLink Input and Output blocks. They are virtual, and they let you configure the input and output signals at the boundaries of a TargetLink subsystem and at the boundaries of subsystems that you want to generate a function for.

Bus signal Buses combine multiple signals, possibly of different types. Buses can also contain other buses. They are then called [nested buses](#).

Bus-capable block A block that can process [bus signals](#). Like [bus port blocks](#), they allow you to assign a type definition and, therefore, a [variable class](#) to all the [bus elements](#) at once. The following blocks are bus-capable:

- Constant
- Custom Code (type II) block
- Data Store Memory, Data Store Read, and Data Store Write

- Delay
- Function Caller
- ArgIn, ArgOut
- Merge
- Multiport Switch (Data Input port)
- Probe
- Sink
- Signal Conversion
- Switch (Data Input port)
- Unit Delay
- Stateflow Data
- MATLAB Function Data

C

Calibratable variable Variable whose value can be changed with a calibration tool during run time.

Calibration Changing the [calibration parameter](#) values of [ECUs](#).

Calibration parameter Any [ECU](#) variable type that can be calibrated. The term *calibration parameter* is independent of the variable type's dimension.

Calprm Defined in a [calprm interface](#). Calprms represent [calibration parameters](#) that are accessible via a [measurement and calibration system](#).

Calprm interface An [interface](#) that is provided or required by a [software component \(SWC\)](#) via a [port \(AUTOSAR\)](#).

Calprm software component A special [software component \(SWC\)](#) that provides [calprms](#). Calprm software components have no [internal behavior](#).

Canonical In the DD, [array-of-struct variables](#) are specified canonically. Canonical means that you specify one array element as a representative for all array elements.

Catalog file (CTLG) A description of the content of an SWC container. It contains file references and file category information, such as source code files (C and H), object code files (such as O or OBJ), variable description files (A2L), or AUTOSAR files (ARXML).

Characteristic table (Classic AUTOSAR) A look-up table as described by [Classic AUTOSAR](#) whose values are measurable or calibratable. See also [compound primitive data type](#)

Classic AUTOSAR Short name for the AUTOSAR *Classic Platform* standard that complements [Adaptive AUTOSAR](#).

Classic initialization mode The initialization mode used when the Simulink diagnostics parameter Underspecified initialization detection is set to **Classic**.

See also [simplified initialization mode](#).

Client port A require port in client-server communication as described by [Classic AUTOSAR](#). In the Data Dictionary, client ports are represented as DD ClientPort objects.

Client-server interface An [interface](#) that describes the [operations](#) that are provided or required by a [software component \(SWC\)](#) via a [port \(AUTOSAR\)](#).

Code generation mode One of three mutually exclusive options for generating TargetLink standard [production code](#), AUTOSAR-compliant production code or RTOS-compliant (multirate RTOS/OSEK) production code.

Code generation unit (CGU) The smallest unit for which you can generate code. These are:

- TargetLink subsystems
- Subsystems configured for incremental code generation
- Referenced models
- DD CodeGenerationUnit objects

Code output style definition file To customize code formatting, you can modify a code output style definition file (XML file). By modifying this file, you can change the representation of comments and statements in the code output.

Code output style sheets To customize code formatting, you can modify code output style sheets (XSL files).

Code section A section of generated code that defines and executes a specific task.

Code size Amount of memory that an application requires specified in RAM and ROM after compilation with the target cross-compiler. This value helps to determine whether the application generated from the code files fits in the ECU memory.

Code variant Code variants lead to source code that is generated differently depending on which variant is selected (i.e., variant at code generation time). For example, if the Type property of a variable has the two variants Int16 and Float32, you can generate either source code for a fixed-point ECU with one variant, or floating-point code with the other.

Compatibility mode The default operation mode of RTE generators. The object code of an SWC that was compiled against an application header generated in compatibility mode can be linked against an RTE generated in compatibility mode (possibly by a different RTE generator). This is due to using standardized data structures in the generated RTE code.

See also [vendor mode](#).

Compiler inlining The process of replacing a function call with the code of the function body during compilation by the C compiler via [inline expansion](#).

This reduces the function call overhead and enables further optimizations at the potential cost of larger [code size](#).

Composition A structuring element in the [application layer](#). A composition consists of [software components](#) and their interconnections via [ports](#).

Compound primitive data type A primitive [application data type \(ADT\)](#) as defined by [Classic AUTOSAR](#) whose category is one of the following:

- COM_AXIS
- CUBOID
- CUBE_4
- CUBE_5
- CURVE
- MAP
- RES_AXIS
- VAL_BLK
- STRING

Compute-through-overflow (CTO) Calculation method for additions and subtraction where overflows are allowed in intermediate results without falsifying the final result.

Concern A concept in component-based development. It describes the idea that components separate their concerns. Accordingly, they must be developed in such a way that they provide the required functionality, are flexible and easy to maintain, and can be assembled, reused, or replaced by newer, functionally equivalent components in a software project without problems.

Config area A DD object that is a child object of the DD root object. The Config object contains configuration data for the tools working with the TargetLink Data Dictionary and configuration data for the TargetLink Data Dictionary itself. There is only one Config object in each DD workspace. The configuration data for the TargetLink Data Dictionary is a list of included DD files, user-defined views, data for variant configurations, etc. The data in the Config area is typically maintained by a Data Dictionary administrator.

ConfigurationDesk A dSPACE software tool for implementing and building real-time applications (RTA).

Constant value expression An expression for which the Code Generator can determine the variable values during code generation.

Constrained range limits User-defined minimum (Min) or maximum (Max) values that the user ensures will never be exceeded. The Code Generator relies on these ranges to make the generated [production code](#) more efficient. If no

Min/Max values are entered, the [implemented range](#) limits are used during production code generation.

Constrained type A DD Typedef object whose Constraints subtree is specified.

Container A bundle of files. The files are described in a catalog file that is part of the container. The files of a container can be spread over your file system.

Container Manager A tool for handling [containers](#).

Container set file (CTS) A file that lists a set of containers. If you export containers, one container set file is created for every TargetLink Data Dictionary.

Conversion method A method that describes the conversion of a variable's integer values in the ECU memory into their physical representations displayed in the Measurement and Calibration (MC) system.

Custom code Custom code consists of C code snippets that can be included in production code by using custom code files that are associated with custom code blocks. TargetLink treats this code as a black box. Accordingly, if this code contains custom code variables you must specify them via [custom code symbols](#). See also [external code](#).

Custom code symbol A variable that is used in a custom code file. It must be specified on the Interface page of custom code blocks.

Customer-specific C function An external function that is called from a Stateflow diagram and whose interface is made known to TargetLink via a scripting mechanism.

D

Data element Defined in a [sender-receiver interface](#). Data elements are information units that are exchanged between [sender ports](#), [receiver ports](#) and [sender-receiver ports](#). They represent the data flow.

Data page A structure containing all of the [calibratable variables](#) that are generated during code generation.

Data prototype The generic term for one of the following:

- [Data element](#)
- [Operation argument](#)
- [Calprm](#)
- [Interrunnable variable \(IRV\)](#)
- Shared or PerInstance [Calprm](#)
- [Per instance memory](#)

Data receive error event An [RTE event](#) that specifies to start or continue the execution of a [runnable](#) related to receiver errors.

Data received event An [RTE event](#) that specifies whether to start or continue the execution of a [runnable](#) after a [data element](#) is received by a [receiver port](#) or [sender-receiver port](#).

Data semantics The communication of [data elements](#) with last-is-best semantics. Newly received data elements overwrite older ones regardless of whether they have been processed or not.

Data send completed event An [RTE event](#) that specifies whether to start or continue the execution of a [runnable](#) related to a sender [acknowledgment](#).

Data transformation A transformation of the data of inter-ECU communication, such as end-to-end protection or serialization, that is managed by the [RTE](#) via [transformers](#).

Data type map Defines a mapping between [implementation data types](#) (represented in TargetLink by DD Typedef objects) and [application data types](#).

Data type mapping set Summarizes all the [data type maps](#) and [mode request type maps](#) of a [software component \(SWC\)](#).

Data variant One of two or more differing data values that are generated into the same C code and can be switched during ECU run time using a calibratable variant ID variable. For example, the Value property of a gain parameter can have the variants 2, 3, and 4.

DataltemMapping (DIM) A DataltemMapping object is a DD object that references a [ReplaceableDataltem \(RDI\)](#) and a DD variable. It is used to define the DD variable object to map an RDI object to, and therefore also the [implementation variable](#) in the generated code.

DD child object The [DD object](#) below another DD object in the [DD object tree](#).

DD data model The DD data model describes the object kinds, their properties and constraints as well as the dependencies between them.

DD file A DD file (*.dd) can be a [DD project file](#) or a [partial DD file](#).

DD object Data item in the Data Dictionary that can contain [DD child objects](#) and DD properties.

DD object tree The tree that arranges all [DD objects](#) according to the [DD data model](#).

DD project file A file containing the [DD objects](#) of a [DD workspace](#).

DD root object The topmost [DD object](#) of the [DD workspace](#).

DD subtree A part of the [DD object tree](#) containing a [DD object](#) and all its descendants.

DD workspace An independent organizational unit (central data container) and the largest entity that can be saved to file or loaded from a [DD project file](#). Any number of DD workspaces is supported, but only the first (DD0) can be used for code generation.

Default enumeration constant Represents the default constant, i.e., the name of an [enumerated value](#) that is used for initialization if an initial value is required, but not explicitly specified.

Direct reuse The Code Generator adds the [instance-specific variables](#) to the reuse structure as leaf struct components.

E

ECU Abbreviation of *electronic control unit*.

ECU software The ECU software consists of all the software that runs on an [ECU](#). It can be divided into the [basic software](#), [run-time environment \(RTE\)](#), and the [application layer](#).

ECU State Manager A piece of software that manages [modes](#). An ECU state manager is part of the [basic software](#).

Enhanceable Simulink block A Simulink® block that corresponds to a TargetLink simulation block, for example, the Gain block.

Enumerated value An enumerated value consists of an [enumeration constant](#) and a corresponding underlying integer value ([enumeration value](#)).

Enumeration constant An enumeration constant defines the name for an [enumerated value](#).

Enumeration data type A data type with a specific name, a set of named [enumerated values](#) and a [default enumeration constant](#).

Enumeration value An enumeration value defines the integer value for an [enumerated value](#).

Event message Event messages are information units that are defined in a [sender-receiver interface](#) and exchanged between [sender ports](#) or [receiver ports](#). They represent the control flow. On the receiver side, each event message is related to a buffer that queues the received messages.

Event semantics Communication of [data elements](#) with first-in-first-out semantics. Data elements are received in the same order they were sent. In simulations, TargetLink behaves as if [data semantics](#) was specified, even if you specified event semantics. However, TargetLink generates calls to the correct RTE API functions for data and event semantics.

ExchangeableWidth A DD object that defines [code variants](#) or improves code readability by using macros for signal widths.

Exclusive area Allows for specifying critical sections in the code that cannot preempt/interrupt each other. An exclusive area can be used to specify the mutual exclusion of [runnables](#).

Executable application The generic term for [offline simulation applications](#) and [real-time applications](#).

Explicit communication A communication mode in [Classic AUTOSAR](#). The data is exchanged whenever data is required or provided.

Explicit object An explicit object is an object in [production code](#) that the Code Generator created from a direct specification made at a [DD object](#) or at a [model element](#). For comparison, see [implicit object](#).

Extern C Stateflow symbol A C symbol (function or variable) that is used in a Stateflow chart but that is defined in an external code module.

External code Existing C code files/modules from external sources (e.g., legacy code) that can be included by preprocessor directives and called by the C code generated by TargetLink. Unlike [Custom code](#), external code is used as it is.

External container A container that is owned by the tool with that you are exchanging a software component but that is not the tool that triggers the container exchange. This container is used when you import files of a software component which were created or changed by the other tool.

F

Filter An algorithm that is applied to received [data elements](#).

Fixed-Point Library A library that contains functions and macros for use in the generated [production code](#).

Function AF The short form for an [access function \(AF\)](#) that is implemented as a C function.

Function algorithm object Generic term for either a MATLAB local function, the interface of a MATLAB local function or a [local MATLAB variable](#).

Function class A class that represents group properties of functions that determine the function definition, function prototypes and function calls of a function in the generated [production code](#). There are two types of function classes: predefined function class objects defined in the `/Pool/FunctionClasses` group in the DD and implicit function classes (default function classes) that can be influenced by templates in the DD.

Function code Code that is generated for a [modular unit](#) that represents functionality and can have [abstract interfaces](#) to be reused without changes in different contexts, e.g. in different [integration models](#).

Function inlining The process of replacing a function call with the code of the function body during code generation by TargetLink via [inline expansion](#). This reduces the function call overhead and enables further optimizations at the potential cost of larger [code size](#).

Function interface An interface that describes how to pass the inputs and outputs of a function to the generated [production code](#). It is described by the function signature.

Function subsystem A subsystem that is atomic and contains a Function block. When generating code, TargetLink generates it as a C function.

Functional Mock-up Unit (FMU) An archive file that describes and implements the functionality of a model based on the Functional Mock-up Interface (FMI) standard.

G

Global data store The specification of a DD DataStoreMemoryBlock object that references a variable and is associated with either a Simulink.Signal object or Data Store Memory block. The referenced variable must have a module specification and a fixed name and must be global and non-static. Because of its central specification in the Data Dictionary, you can use it across the boundaries of [CGUs](#).

I

Implementation Describes how a specific [internal behavior](#) is implemented for a given platform (microprocessor type and compiler). An implementation mainly consists of a list of source files, object files, compiler attributes, and dependencies between the make and build processes.

Implementation data type (IDT) According to AUTOSAR, implementation data types are used to define types on the implementation level of abstraction. From the implementation point of view, this regards the storage and manipulation of digitally represented data. Accordingly, implementation data types have data semantics and do consider implementation details, such as the data type.

Implementation data types can be constrained to change the resolution of the digital representation or define a range that is to be considered. Typically, they correspond to typedef statements in C code and still abstract from platform specific details such as endianness.

See also [application data type \(ADT\)](#).

Implementation variable A variable in the generated [production code](#) to which a [ReplaceableDataItem \(RDI\)](#) object is mapped.

ImplementationPolicy A property of [data element](#) and [Calprm](#) elements that specifies the implementation strategy for the resulting variables with respect to consistency.

Implemented range The range of a variable defined by its [scaling](#) parameters. To avoid overflows, the implemented range must include the maximum and minimum values the variable can take in the [simulation application](#) and in the ECU.

Implicit communication A communication mode in [Classic AUTOSAR](#). The data is exchanged at the start and end of the runnable that requires or provides the data.

Implicit object Any object created for the generated code by the TargetLink Code Generator (such as a variable, type, function, or file) that may not have been specified explicitly via a TargetLink block, a Stateflow object, or the TargetLink Data Dictionary. Implicit objects can be influenced via DD templates. For comparison, see [explicit object](#).

Implicit property If the property of a [DD object](#) or of a model based object is not directly specified at the object, this property is created by the Code Generator and is based on internal templates or DD Template objects. These properties are called implicit properties. Also see [implicit object](#) and [explicit object](#).

Included DD file A [partial DD file](#) that is inserted in the proper point of inclusion in the [DD object tree](#).

Incremental code generation unit (CGU) Generic term for [code generation units \(CGUs\)](#) for which you can incrementally generate code. These are:

- Referenced models
- Subsystems configured for incremental code generation

Incremental CGUs can be nested in other model-based CGUs.

Indirect reuse The Code Generator adds pointers to the reuse structure which reference the indirectly reused [instance-specific variables](#).

Indirect reuse has the following advantages to [direct reuse](#):

- The combination of [shared](#) and [instance-specific variable](#).
- The reuse of input/output variables of neighboring blocks.

Inline expansion The process of replacing a function call with the code of the function body. See also [function inlining](#) and [compiler inlining](#).

Instance-specific variable A variable that is accessed by one [reusable system instance](#). Typically, instance-specific variables are used for states and parameters whose value are different across instances.

Instruction set simulator (ISS) A simulation model of a microprocessor that can execute binary code compiled for the corresponding microprocessor. This allows the ISS to behave in the same way as the simulated microprocessor.

Integration model A model or TargetLink subsystem that contains [modular units](#) which it integrates to make a larger entity that provides its functionality.

Interface Describes the [data elements](#), [NvData](#), [event messages](#), [operations](#), or [calibration parameters](#) that are provided or required by a [software component \(SWC\)](#) via a [port \(AUTOSAR\)](#).

Internal behavior An element that represents the internal structure of an [atomic software component \(atomic SWC\)](#). It is characterized by the following entities and their interdependencies:

- [Exclusive area](#)
- [Interrunnable variable \(IRV\)](#)
- [Per instance memory](#)
- [Per instance parameter](#)
- [Runnable](#)
- [RTE event](#)
- [Shared parameter](#)

Interrunnable variable (IRV) Variable object for specifying communication between the [runnables](#) in one [atomic software component \(atomic SWC\)](#).

Interrupt service routine (ISR) function A function that implements an ISR and calls the step functions of the subsystems that are assigned by the user or by the TargetLink Code Generator during multirate code generation.

Intertask communication The flow of data between tasks and ISRs, tasks and tasks, and between ISRs and ISRs for multirate code generation.

Is service A property of an [interface](#) that indicates whether the interface is provided by a [basic software service](#).

ISV Abbreviation for instance-specific variable.

L

Leaf bus element A leaf bus element is a subordinate [bus element](#) that is not a [bus](#) itself.

Leaf bus signal See also [leaf bus element](#).

Leaf struct component A leaf struct component is a subordinate [struct component](#) that is not a [struct](#) itself.

Legacy function A function that contains a user-provided C function.

Library subsystem A subsystem that resides in a Simulink® library.

Local container A container that is owned by the tool that triggers the container exchange.

The tool that triggers the exchange transfers the files of a [software component](#) to this container when you export a software component. The [external container](#) is not involved.

Local MATLAB variable A variable that is generated when used on the left side of an assignment or in the interface of a MATLAB local function. TargetLink does not support different data types and sizes on local MATLAB variables.

M

Look-up function A function for a look-up table that returns a value from the look-up table (1-D or 2-D).

Macro A literal representing a C preprocessor definition. Macros are used to provide a fixed sequence of computing instructions as a single program statement. Before code compilation, the preprocessor replaces every occurrence of the macro by its definition, i.e., by the code that it stands for.

Macro AF The short form for an [access function \(AF\)](#) that is implemented as a function-like preprocessor macro.

MATLAB code elements MATLAB code elements include [MATLAB local functions](#) and [local MATLAB variables](#). MATLAB code elements are not available in the Simulink Model Explorer or the Property Manager.

MATLAB local function A function that is scoped to a [MATLAB main function](#) and located at the same hierarchy level. MATLAB local functions are treated like MATLAB main functions and have the same properties as the MATLAB main function by default.

MATLAB main function The first function in a MATLAB function file.

Matrix AF An access function resulting from a DD AccessFunction object whose VariableKindSpec property is set to APPLY_TO_MATRIX.

Matrix signal Collective term for 2-D signals implemented as [matrix variable](#) in [production code](#).

Matrix variable Collective term for 2-D arrays in [production code](#) that implement 2-D signals.

Measurement Viewing and analyzing the time traces of [calibration parameters](#) and [measurement variables](#), for example, to observe the effects of ECU parameter changes.

Measurement and calibration system A tool that provides access to an [ECU](#) for [measurement](#) and [calibration](#). It requires information on the [calibration parameters](#) and [measurement variables](#) with the ECU code.

Measurement variable Any variable type that can be [measured](#) but not [calibrated](#). The term *measurement variable* is independent of a variable type's dimension.

Memory mapping The process of mapping variables and functions to different [memory sections](#).

Memory section A memory location to which the linker can allocate variables and functions.

Message Browser A TargetLink component for handling fatal (F), error (E), warning (W), note (N), and advice (A) messages.

MetaData files Files that store metadata about code generation. The metadata of each [code generation unit \(CGU\)](#) is collected in a DD Subsystem object that is written to the file system as a partial DD file called `<CGU>_SubsystemObject.dd`.

Method Behavior subsystem An atomic subsystem used to generate code for a method implementation. From the TargetLink perspective, this is an [Adaptive AUTOSAR Function](#) that can take arguments. It contains a Function block whose AUTOSAR mode is set to **Adaptive** and whose Role is set to **Method Behavior**.

Method Call subsystem An atomic subsystem that is used to generate a method call in the code of an [Adaptive AUTOSAR Function](#). The subsystem contains a Function block whose AUTOSAR mode is set to **Adaptive** and whose Role is set to **Method Call**. The subsystem interface is used to generate the function interface while additional model elements that are contained in the subsystem are only for simulation purposes.

Microcontroller family (MCF) A group of [microcontroller units](#) with the same processor, but different peripherals.

Microcontroller unit (MCU) A combination of a specific processor with additional peripherals, e.g. RAM or AD converters. MCUs with the same processor, but different peripherals form a [microcontroller family](#).

MIL simulation A simulation method in which the function model is computed (usually with double floating-point precision) on the host computer as an executable specification. The simulation results serve as a reference for [SIL simulations](#) and [PIL simulations](#).

MISRA Organization that assists the automotive industry to produce safe and reliable software, e.g., by defining guidelines for the use of C code in automotive electronic control units or modeling guidelines.

Mode An operating state of an [ECU](#), a single functional unit, etc..

Mode declaration group Contains the possible [operating states](#), for example, of an [ECU](#) or a single functional unit.

Mode manager A piece of software that manages [modes](#). A mode manager can be implemented as a [software component \(SWC\)](#) of the [application layer](#).

Mode request type map An entity that defines a mapping between a [mode declaration group](#) and a type. This specifies that mode values are instantiated in the [software component \(SWC\)](#)'s code with the specified type.

Mode switch event An [RTE event](#) that specifies to start or continue the execution of a [runnable](#) as a result of a [mode change](#).

Model Compare A dSPACE software tool that identifies and visualizes the differences in the contents of Simulink/TargetLink models (including Stateflow). It can also merge the models.

Model component A model-based [code generation unit \(CGU\)](#).

Model element A model in MATLAB/Simulink consists of model elements that are TargetLink blocks, Simulink blocks, and Stateflow objects, and signal lines connecting them.

Model port A port used to connect a [behavior model](#) in [ConfigurationDesk](#). In TargetLink, multiple model ports of the same kind (data in or data out) can be grouped in a [model port block](#).

Model port block A block in [ConfigurationDesk](#) that has one or more [model ports](#). It is used to connect the [behavior model](#) in [ConfigurationDesk](#).

Model port variable A DD Variable object that represents a [model port](#) of a [behavior model](#) in [ConfigurationDesk](#).

Model-dependent code elements Code elements that (partially) result from specifications made in the model.

Model-independent code elements Code elements that can be generated from specifications made in the Data Dictionary alone.

Modular unit A submodel containing functionality that is reusable and can be integrated in different [integration models](#). The [production code](#) for the modular unit can be generated separately.

Module A DD object that specifies code modules, header files, and other arbitrary files.

Module specification The reference of a DD Module object at a **Function Block** ([TargetLink Model Element Reference](#)) block or DD object. The resulting code elements are generated into the [module](#). See also [production code](#) and [stub code](#).

ModuleOwnership A DD object that specifies an owner for a module (module owner) or module group, i.e. the owning [code generation unit \(CGU\)](#) that generates the [production code](#) for it or declares the [module](#) as external code that is not generated by TargetLink.

N

Nested bus A nested bus is a [bus](#) that is a subordinate [bus element](#) of another bus.

Nested struct A nested struct is a [struct](#) that is a subordinate [struct component](#) of another struct.

Non-scalar signal Collective term for vector and [matrix signals](#).

Non-standard scaling A [scaling](#) whose LSB is different from 2^0 or whose Offset is not 0.

Nv receiver port A require port in NvData communication as described by [Classic AUTOSAR](#). In the Data Dictionary, nv receiver ports are represented as DD NvReceiverPort objects.

Nv sender port A provide port in NvData communication as described by [Classic AUTOSAR](#). In the Data Dictionary, nv sender ports are represented as DD NvSenderPort objects.

Nv sender-receiver port A provide-require port in NvData communication as described by [Classic AUTOSAR](#). In the Data Dictionary, nv sender-receiver ports are represented as DD NvSenderReceiverPort objects.

NvData Data that is exchanged between an [atomic software component \(atomic SWC\)](#) and the [ECU's NVRAM](#).

NvData interface An [interface](#) used in [NvData](#) communication.

NVRAM Abbreviation of *non volatile random access memory*.

NVRAM manager A piece of software that manages an [ECU's NVRAM](#). An NVRAM manager is part of the [basic software](#).

O

Offline simulation application (OSA) An application that can be used for offline simulation in VEOS.

Online parameter modification The modification of parameters in the [production code](#) before or during a [SIL simulation](#) or [PIL simulation](#).

Operation Defined in a [client-server interface](#). A [software component \(SWC\)](#) can request an operation via a [client port](#). A software component can provide an operation via a [server port](#). Operations are implemented by [server runnables](#).

Operation argument Specifies a C-function parameter that is passed and/or returned when an [operation](#) is called.

Operation call subsystem A collective term for [synchronous operation call subsystem](#) and [asynchronous operation call subsystem](#).

Operation call with runnable implementation subsystem An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to **Classic** and whose Role is set to **Operation call with runnable implementation**.

Operation invoked event An [RTE event](#) that specifies to start or continue the execution of a [runnable](#) as a result of a client call. A runnable that is related to an [operation invoked event](#) represents a server.

Operation result provider subsystem A subsystem used when modeling *asynchronous* client-server communication. It is used to generate the call of the `Rte_Result` API function and for simulation purposes.

See also [asynchronous operation call subsystem](#).

Operation subsystem A collective term for [operation call subsystem](#) and [operation result provider subsystem](#).

OSEK Implementation Language (OIL) A modeling language for describing the configuration of an OSEK application and operating system.

P

Package A structuring element for grouping elements of [software components](#) in any hierarchy. Using package information, software components can be spread across or combined from several [software component description \(SWC-D\)](#) files during [AUTOSAR import/export](#) scenarios.

Parent model A model containing references to one or more other models by means of the Simulink Model block.

Partial DD file A [DD file](#) that contains only a DD subtree. If it is included in a [DD project file](#), it is called [Included DD file](#). The partial DD file can be located on a central network server where all team members can share the same configuration data.

Per instance memory The definition of a data prototype that is instantiated for each [atomic software component instance](#) by the [RTE](#). A data type instance can be accessed only by the corresponding instance of the [atomic SWC](#).

Per instance parameter A parameter for measurement and calibration unique to the instance of a [software component \(SWC\)](#) that is instantiated multiple times.

Physical evaluation board (physical EVB) A board that is equipped with the same target processor as the [ECU](#) and that can be used for validation of the generated [production code](#) in [PIL simulation](#) mode.

PIL simulation A simulation method in which the TargetLink control algorithm ([production code](#)) is computed on a [microcontroller target](#) ([physical](#) or [virtual](#)).

Plain data type A data type that is not struct, union, or pointer.

Platform A specific target/compiler combination. For the configuration of platforms, refer to the Code generation target settings in the TargetLink Main Dialog Block block.

Pool area A DD object which is parented by the DD root object. It contains all data objects which can be referenced in TargetLink models and which are used for code generation. Pool data objects allow common data specifications to be reused across different blocks or models to easily keep consistency of common properties.

Port (AUTOSAR) A part of a [software component \(SWC\)](#) that is the interaction point between the component and other software components.

Port-defined argument values Argument values the RTE can implicitly pass to a server.

Preferences Editor A TargetLink tool that lets users view and modify all user-specific preference settings after installation has finished.

Production code The code generated from a [code generation unit \(CGU\)](#) that owns the module containing the code. See also [stub code](#).

Project folder A folder in the file system that belongs to a TargetLink code generation project. It forms the root of different [artifact locations](#) that belong to this project.

Property Manager The TargetLink user interface for conveniently managing the properties of multiple model elements at the same time. It can consist of menus, context menus, and one or more panes for displaying property-related information.

Provide calprm port A provide port in parameter communication as described by [Classic AUTOSAR](#). In the Data Dictionary, provide calprm ports are represented as DD ProvideCalPrmPort objects.

R

Read/write access function An [access function \(AF\)](#) that *encapsulates the instructions* for reading or writing a variable.

Real-time application An application that can be executed in real time on dSPACE real-time hardware such as SCALEXIO.

Receiver port A require port in sender-receiver communication as described by [Classic AUTOSAR](#). In the Data Dictionary, receiver ports are represented as DD ReceiverPort objects.

ReplaceableDataItem (RDI) A ReplaceableDataItem (RDI) object is a DD object that describes an abstract interface's basic properties such as the data type, scaling and width. It can be referenced in TargetLink block dialogs and is generated as a global [macro](#) during code generation. The definition of the RDI macro can then be generated later, allowing flexible mapping to an [implementation variable](#).

Require calprm port A require port in parameter communication as described by [Classic AUTOSAR](#). In the Data Dictionary, require calprm ports are represented as DD RequireCalPrmPort objects.

RequirementInfo An object of a DD RequirementInfo object. It describes an item of requirement information and has the following properties: Description, Document, Location, UserTag, ReferencedInCode, SimulinkStateflowPath.

Restart function A production code function that initializes the global variables that have an entry in the RestartfunctionName field of their [variable class](#).

Reusable function definition The function definition that is to be reused in the generated code. It is the code counterpart to the [reusable system definition](#) in the model.

Reusable function instance An instance of a [reusable function definition](#). It is the code counterpart to the [reusable system instance](#) in the model.

Reusable model part Part of the model that can become a [reusable system definition](#). Refer to [Basics on Function Reuse](#) ([TargetLink Customization and Optimization Guide](#)).

Reusable system definition A model part to which the function reuse is applied.

Reusable system instance An instance of a [reusable system definition](#).

Root bus A root bus is a [bus](#) that is not a subordinate part of another bus.

Root function A function that represents the starting point of the TargetLink-generated code. It is called from the environment in which the TargetLink-generated code is embedded.

Root model The topmost [parent model](#) in the system hierarchy.

Root module The [module](#) that contains all the code elements that belong to the [production code](#) of a [code generation unit \(CGU\)](#) and do not have their own [module specification](#).

Root step function A step function that is called only from outside the [production code](#). It can also represent a non-TargetLink subsystem within a TargetLink subsystem.

Root struct A root struct is a [struct](#) that is not a subordinate part of another struct.

Root style sheet A root style sheet is used to organize several style sheets defining code formatting.

RTE event The abbreviation of [run-time environment event](#).

Runnable A part of an [atomic SWC](#). With regard to code execution, a runnable is the smallest unit that can be scheduled and executed. Each runnable is implemented by one C function.

Runnable execution constraint Constraints that specify [runnables](#) that are allowed or not allowed to be started or stopped before a runnable.

Runnable subsystem An atomic subsystem that contains a Function block whose AUTOSAR mode property is set to **Classic** and whose Role is set to **Runnable**.

Run-time environment (RTE) A generated software layer that connects the [application layer](#) to the [basic software](#). It also interconnects the different [SWCs](#) of the application layer. There is one RTE per [ECU](#).

Run-time environment event A part of an [internal behavior](#). It defines the situations and conditions for starting or continuing the execution of a specific [runnable](#).

S

Scaling A parameter that specifies the fixed-point range and resolution of a variable. It consists of the data type, least significant bit (LSB) and offset.

Sender port A provide port in sender-receiver communication as described by [Classic AUTOSAR](#). In the Data Dictionary, sender ports are represented as DD SenderPort objects.

Sender-receiver interface An [interface](#) that describes the [data elements](#) and [event messages](#) that are provided or required by a [software component \(SWC\)](#) via a [port \(AUTOSAR\)](#).

Sender-receiver port A provide-require port in sender-receiver communication as described by [Classic AUTOSAR](#). In the Data Dictionary, sender-receiver ports are represented as DD SenderReceiverPort objects.

Server port A provide port in client-server communication as described by [Classic AUTOSAR](#). In the Data Dictionary, server ports are represented as DD ServerPort objects.

Server runnable A [runnable](#) that provides an [operation](#) via a [server port](#). Server runnables are triggered by [operation invoked events](#).

Shared parameter A parameter for measurement and calibration that is used by several instances of the same [software component \(SWC\)](#).

Shared variable A variable that is accessed by several [reusable system instances](#). Typically, shared variables are used for parameters whose values are the same across instances. They increase code efficiency.

SIC runnable function A void (void) function that is called in a [task](#). Generated into the [Simulink implementation container \(SIC\)](#) to call the [root function](#) that is generated by TargetLink from a TargetLink subsystem. In [ConfigurationDesk](#), this function is called *runnable function*.

SIL simulation A simulation method in which the control algorithm's generated [production code](#) is computed on the host computer in place of the corresponding model.

Simple TargetLink model A simple TargetLink model contains at least one TargetLink Subsystem block and exactly one MIL Handler block.

Simplified initialization mode The initialization mode used when the Simulink diagnostics parameter Underspecified initialization detection is set to Simplified.

See also [classic initialization mode](#).

Simulation application An application that represents a graphical model specification (implemented control algorithm) and simulates its behavior in an offline Simulink environment.

Simulation code Code that is required only for simulation purposes. Does not belong to the [production code](#).

Simulation S-function An S-function that calls either the [root step functions](#) created for a TargetLink subsystem, or a user-specified step function (only possible in test mode via API).

Simulink data store Generic term for a memory region in MATLAB/Simulink that is defined by one of the following:

- A Simulink.Signal object
- A Simulink Data Store Memory block

Simulink function call The location in the model where a Simulink function is called. This can be:

- A Function Caller block
- The action language of a Stateflow Chart
- The MATLAB code of a MATLAB function

Simulink function definition The location in the model where a Simulink function is defined. This can be one of the following:

- [Simulink Function subsystem](#)
- Exported Stateflow graphical function
- Exported Stateflow truthtable function
- Exported Stateflow MATLAB function

Simulink function ports The ports that can be used in a [Simulink Function subsystem](#). These can be the following:

- TargetLink ArgIn and ArgOut blocks
These ports are specific for each [Simulink function call](#).
- TargetLink InPort/OutPort and Bus InPort/Bus OutPort blocks
These ports are the same for all [Simulink function calls](#).

Simulink Function subsystem A subsystem that contains a Trigger block whose Trigger Type is `function-call` and whose Treat as Simulink Function checkbox is selected.

Simulink implementation container (SIC) A file that contains all the files required to import [production code](#) generated by TargetLink into [ConfigurationDesk](#) as a [behavior model](#) with [model ports](#).

Slice A section of a vector or [matrix signal](#), whose elements have the same properties. If all the elements of the vector/matrix have the same properties, the whole vector/matrix forms a slice.

Software component (SWC) The generic term for [atomic software component \(atomic SWC\)](#), [compositions](#), and special software components, such as [calprm software components](#). A software component logically groups and encapsulates single functionalities. Software components communicate with each other via [ports](#).

Software component description (SWC-D) An XML file that describes [software components](#) according to AUTOSAR.

Stateflow action language The formal language used to describe transition actions in Stateflow.

Struct A struct (short form for [structure](#)) consists of subordinate [struct components](#). A struct component can be a struct itself.

Struct component A struct component is a part of a [struct](#) and can be a struct itself.

Structure A structure (long form for [struct](#)) consists of subordinate [struct components](#). A struct component can be a struct itself.

Stub code Code that is required to build the simulation application but that belongs to another [code generation unit \(CGU\)](#) than the one used to generate [production code](#).

Subsystem area A DD object which is parented by the DD root object. This object consists of an arbitrary number of Subsystem objects, each of which is the result of code generation for a specific [code generation unit \(CGU\)](#). The Subsystem objects contain detailed information on the generated code, including C modules, functions, etc. The data in this area is either automatically generated or imported from ASAM MCD-2 MC, and must not be modified manually.

Supported Simulink block A TargetLink-compliant block from the Simulink library that can be directly used in the model/subsystem for which the Code Generator generates [production code](#).

SWC container A [container](#) for files of one [SWC](#).

Synchronous operation call subsystem A subsystem used when modeling *synchronous* client-server communication. It is used to generate the call of the `Rte_Call` API function and for simulation purposes.

T

Table function A function that returns table output values calculated from the table inputs.

Target config file An XML file named `TargetConfig.xml`. It contains information on the basic data types of the target/compiler combination such as the byte order, alignment, etc.

Target Optimization Module (TOM) A TargetLink software module for optimizing [production code](#) generation for a specific [microcontroller](#)/compiler combination.

Target Simulation Module (TSM) A TargetLink software module that provides support for a number of evaluation board/compiler combinations. It is used to test the generated code on a target processor. The TSM is licensed separately.

TargetLink AUTOSAR Migration Tool A software tool that converts classic, non-AUTOSAR TargetLink models to AUTOSAR models at a click.

TargetLink AUTOSAR Module A TargetLink software module that provides extensive support for modeling, simulating, and generating code for AUTOSAR software components.

TargetLink Base Suite The base component of the TargetLink software including the [ANSI C](#) Code Generator and the Data Dictionary Manager.

TargetLink base type One of the types used by TargetLink instead of pure C types in the generated code and the delivered libraries. This makes the code platform independent.

TargetLink Blockset A set of blocks in TargetLink that allow [production code](#) to be generated from a model in MATLAB/Simulink.

TargetLink Data Dictionary The central data container that holds all relevant information about an ECU application, for example, for code generation.

TargetLink simulation block A block that processes signals during simulation. In most cases, it is a block from standard Simulink libraries but carries additional information required for production code generation.

TargetLink subsystem A subsystem from the TargetLink block library that defines a section of the Simulink model for which code must be generated by TargetLink.

Task A code section whose execution is managed by the real-time operating system. Tasks can be triggered periodically or based on events. Each task can call one or more [SIC runnable functions](#).

Task function A function that implements a task and calls the functions of the subsystems which are assigned to the task by the user or via the TargetLink Code Generator during multirate code generation.

Term function A function that contains the code to be executed when the simulation finishes or the ECU application terminates.

Terminate function A [runnable](#) that finalizes a [SWC](#), for example, by calling code that has to run before the application shuts down.

Timing event An [RTE event](#) that specifies to start or continue the execution of a [runnable](#) at constant time intervals.

tlilib A TargetLink block library that is the source for creating TargetLink models graphically. Refer to [How to Open the TargetLink Block Library](#) ([TargetLink Orientation and Overview Guide](#)).

Transformer The [Classic AUTOSAR](#) entity used to perform a [data transformation](#).

TransformerError The parameter passed by the [run-time environment \(RTE\)](#) if an error occurred in a [data transformation](#). The `Std_TransformerError` is a struct whose components are the transformer class and the error code. If the error is a hard error, a special runnable is triggered via the [TransformerHardErrorEvent](#) to react to the error. In AUTOSAR releases prior to R19-11 this struct was named `Rte_TransformerError`.

TransformerHardErrorEvent The [RTE event](#) that triggers the [runnable](#) to be used for responding to a hard [TransformerError](#) in a [data transformation](#) for client-server communication.

Type prefix A string written in front of the variable type of a variable definition/declaration, such as `MyTypePrefix Int16 MyVar`.

U

Unicode The most common standard for extended character sets is the Unicode standard. There are different schemes to encode Unicode in byte format, e.g., UTF-8 or UTF-16. All of these encodings support all Unicode characters. Scheme conversion is possible without losses. The only difference between these encoding schemes is the memory that is required to represent Unicode characters.

User data type (UDT) A data type defined by the user. It is placed in the Data Dictionary and can have associated constraints.

Utility blocks One of the categories of TargetLink blocks. The blocks in the category keep TargetLink-specific data, provide user interfaces, and control the simulation mode and code generation.

V

Validation Summary Shows unresolved model element data validation errors from all model element variables of the Property View. It lets you search, filter, and group validation errors.

Value copy AF An [access function \(AF\)](#) resulting from DD AccessFunction objects whose AccessFunctionKind property is set to READ_VALUE_COPY or WRITE_VALUE_COPY.

Variable access function An [access function \(AF\)](#) that *encapsulates the* access to a variable for reading or writing.

Variable class A set of properties that define the role and appearance of a variable in the generated [production code](#), e.g. CAL for global calibratable variables.

VariantConfig A DD object in the [Config area](#) that defines the [code variants](#) and [data variants](#) to be used for simulation and code generation.

VariantItem A DD object in the DD [Config area](#) used to variant individual properties of DD Variable and [ExchangeableWidth](#) objects. Each variant of a property is associated with one variant item.

V-ECU implementation container (VECU) A file that consists of all the files required to build an [offline simulation application \(OSA\)](#) to use for simulation with VEOS.

V-ECU Manager A component of TargetLink that allows you to configure and generate a V-ECU implementation.

Vendor mode The operation mode of RTE generators that allows the generation of RTE code which contains vendor-specific adaptations, e.g., to reduce resource consumption. To be linkable to an RTE, the object code of an SWC must have been compiled against an application header that matches the RTE code generated by the specific RTE generator. This is the case because the data structures and types can be implementation-specific.

See also [compatibility mode](#).

VEOS A dSPACE software platform for the C-code-based simulation of [virtual ECUs](#) and environment models on a PC.

Virtual ECU (V-ECU) Software that emulates a real [ECU](#) in a simulation scenario. The virtual ECU comprises components from the application and the [basic software](#), and provides functionalities comparable to those of a real ECU.

Virtual ECU testing Offline and real-time simulation using [virtual ECUs](#).

Virtual evaluation board (virtual EVB) A combination of an [instruction set simulator \(ISS\)](#) and a simulated periphery. This combination can be used for validation of generated [production code](#) in [PIL simulation](#) mode.

W

Worst-case range limits A range specified by calculating the minimum and maximum values which a block's output or state variable can take on with respect to the range of the inputs or the user-specified [constrained range limits](#).

A

- absolute alarm 132
- action 132
 - outport 135
- ActivateTask 132
- activating OSEK tasks 91
- alarm 108
 - callback 133
- automatic action assignment 138

B

- basic task 17
- building
 - OSEK applications 158

C

- CancelAlarm 135
- Common Program Data folder 8
- CommonProgramDataFolder 8
- counter 18
- counters 131
- CounterTrigger 134
- CounterValue 135
- creating
 - alarms 113
 - messages 77
 - OSEK events 100
- critical section 126
- Cycle 135
- cyclic subsystem 25

D

- data
 - exchange 68
 - integrity 68
- default
 - task 25
 - task partitioning 25
 - templates 33
- Documents folder 8
- DocumentsFolder 8

E

- events 18
- exporting
 - OIL files 119
- extended task 17
- external
 - ISR 50
 - task 50

G

- generic
 - OSEK RTOS 89
 - RTOS 89
- global buffer 73
- grouping signals 78

H

- Hook scripts 161

I

- importing OIL files 119
- interrupt 18
- ISR 34
 - block 47

L

- local copy 69
- Local Program Data folder 8
- LocalProgramDataFolder 8

M

- mapping OSEK events 106
- Max allowed value 131
- message 68
 - component 68
 - properties 74
- Min cycle 131
- modeling rules 27
- multirate model 15
- multitasking RTOS 15

N

- noncyclic subsystem 25
- non-preemptive task 146

O

- OIL 117
 - file 117
- OSEK
 - event 92
 - implementation language 117
 - message 73
 - OSEK/VDX 17
 - task 91

P

- protection mechanism 127

R

- relative alarm 132
- resource 18
- root step functions 32
- RTOS Blocks library 125

S

- scalability 18
- Schedule block 146
- SetEvent 133
- simulating
 - multirate models 150
 - OSEK models 150
- StartAlarm 135

- StartValue 135
- step functions 32

T

- task 16
 - partitioning 24
 - properties 32
 - reference 41
 - template 33
- Task block 26
- templates 33
- tick duration 138
 - in simulation 139
- Ticks per base 131
- time-specifying block 27
- TL_TMOS_OSEK license 88
- TlEnterCriticalSection 155
- TlLeaveCriticalSection 155
- TlLockMessage 155
- TlTriggerTask 154
- TlUnlockMessage 155

U

- user-defined task 26

